Contents

1 Graph theory basics **3**

1.1 Graphs, adjacency and incidence . . . 3
1.1.1 Example G_{ex_1} . . . 3
1.1.2 Example G_{ex_2} . . . 5
1.1.3 Adjacency and incidence . . . 6
1.1.4 Example G_{ex_3} . . . 7
1.1.5 Example WG . . . 10
1.1.6 Example $S(n,k)$ . . . 10
1.1.7 Example O_n . . . 11
1.1.8 Example $RPG(8,15)$ . . . 12
1.1.9 Notation . . . 13
1.2 Simple finite graphs . . . 14
1.3 Degree . . . 16
1.3.1 Petersen graph . . . 18
1.3.2 A 3-regular example . . . 19
1.3.3 A 4-regular example . . . 19
1.4 Handshake lemma . . . 20
1.5 Graph representation . . . 21
1.5.1 Adjacency list . . . 21
1.5.2 Adjacency matrix . . . 25
1.5.3 Incidence matrix . . . 27
1.5.4 Relations between representations . . . 28
1.6 Graph isomorphism . . . 29
1.7 Walks, paths and cycles . . . 33
1.8 Special graphs . . . 37
1.8.1 Empty graph E_n . . . 37
1.8.2 Complete graph K_n . . . 38
1.8.3 Bipartite graphs . . . 38
1.8.4 Path P_n . . . 40

1.8.5 Cycle C_n 41
1.9 Common graph measures 41
1.10 Subgraphs 45
1.11 Connected graphs 52
1.11.1 Bridges and cut vertices 56
1.12 Trees 58
1.13 Spanning trees 63
1.14 BFS 66
1.14.1 Example: connected graph 67
1.14.2 Example: disconnected graph 76
1.14.3 Properties of BFS trees 81

Chapter 1

Graph theory basics

1.1 Graphs, adjacency and incidence

The term *graph* will refer to a mathematical object that has wide application in many areas. There are various equivalent definitions of graphs. We present two of them.

Definition 1. *[graph] The following are equivalent definitions of graphs*

1. *A* graph *G is a non-empty set, $V(G)$, of objects, called* vertices*, together with a set, $E(G)$, of unordered pairs of (distinct) vertices. The elements of $E(G)$ are called* edges*;*

2. *A* graph *G is a triple consisting of non-empty set $V(G)$, of objects, called* vertices*, a set, $E(G)$, of objects called* edge *and a relation I (incidence relation) that associates each edge with a pair of vertices;*

Since this is an introductory material as stated in the definition the set of vertices $V(G)$ will be a non-empty set. Vertices are sometimes called *nodes* or *points*.

The set of edges $E(G)$, sometimes called *lines*, may be empty but will at all times satisfy $E(G) \cap V(G) = \emptyset$. That is the set of vertices and set of edges are disjoint. In some text you may see a definition for edges given as $E(G) \subseteq V(G) \times V(G)$, which says that the set of edges is a relation on the set of vertices. Should you prefer that formalization, for the graphs studied here the edges imply a symmetric and non-reflexive relation.

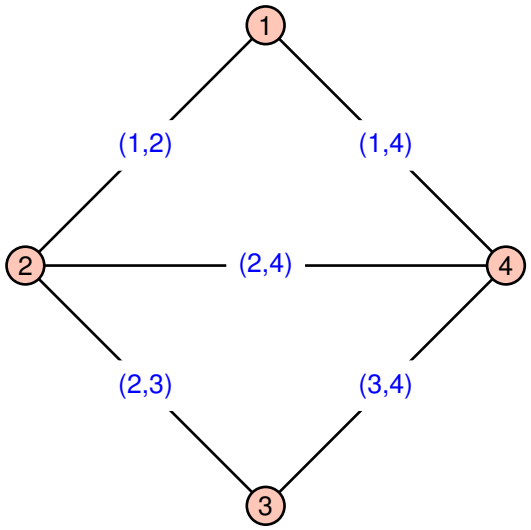
1.1.1 Example G_{ex_1}

Define the graph G_{ex_1} via

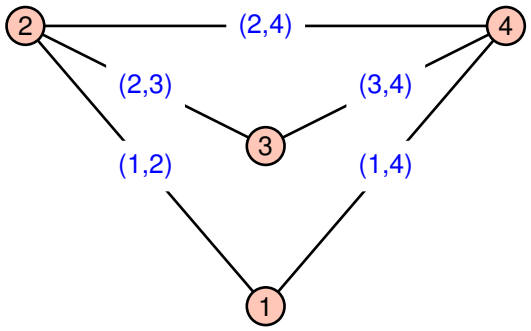
vertex set $V(G_{ex_1}) = \{1, 2, 3, 4\}$

edge set $E(G_{ex_1}) = \{(1, 2), (1, 4), (2, 3), (2, 4), (3, 4)\}$.

For most graphs there is a corresponding figure that associates a point with every vertex and there is a line between two vertices if there is an edge that is associated (Definition 1 part 2) with those two vertices. For the graph defined here a corresponding drawing is



These drawings are known as *embeddings*. Embeddings can be formalized similar to the graph of a function such as $y = x^2$ – intuitively embedding is a pair of function from the vertex set to $\mathbb{R}^2$ and from the edge set to $\mathbb{R}^2$ whose result is the above "drawing". Unlike the drawings (graphs or embeddings) of functions, embeddings of graphs (graph as in Definition 1) are not unique. In fact there may be infinitely many embeddings of a graph. For example G_{ex_1} has alternative embedding



It is therefore crucial to distinguish a graph from its embedding: graphs are incidence structures. Embeddings are simply a tool to visualize the ideas.

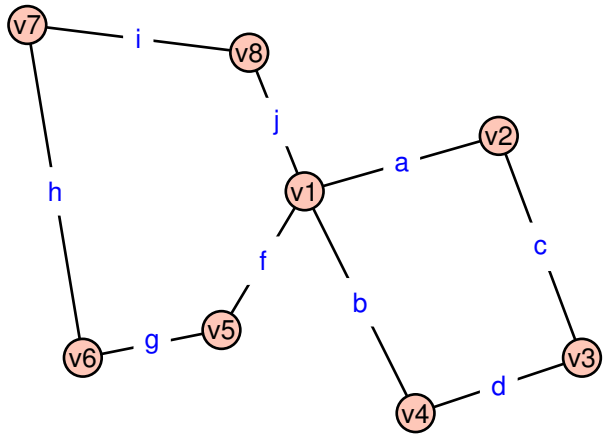
1.1.2 Example G_{ex_2}

Define the graph G_{ex_2} via

vertex set $V(G_{ex_2}) = \{v1, v2, v3, v4, v5, v6, v7, v8\}$

edge set $E(G_{ex_2}) = \{a, b, c, d, f, g, h, i, j\}$.

One embedding of G_{ex_2} is



1.1.3 Adjacency and incidence

The following terminology relates vertices and edges:

Adjacency:

- if edge $e = (u, v)$ then we say vertices u and v are *adjacent*;
- vertices adjacent to a vertex u are *neighbours* of u; the set of neighbours of u is denoted by $N(u)$;

Example: consider the graph G_{ex_1} from §1.1.1:

- vertices 1 and 2 are adjacent, which also means that vertices 2 and 1 are adjacent;
- vertices 1 and 4 are adjacent;
- vertices 2 and 3 are adjacent;
- vertices 2 and 4 are adjacent;
- vertices 3 and 4 are adjacent;

Further

- The neighbours of vertex 1 consists of vertices 2 and 4, that is

$$N(1) = \{2, 4\}$$

- for vertex 2 we have $N(2) = \{1, 3, 4\}$
- for vertex 3 we have $N(3) = \{2, 4\}$
- for vertex 4 we have $N(4) = \{1, 2, 3\}$

If there is no edge joining two vertices u and v we say that those vertices are *not* adjacent. For example, in G_{ex_1} vertex 1 is not adjacent to vertex 3.

Incidence:

- if edge $e = (u, v)$ then we say edge e is *incident* with u and v;
- if edge $e = (u, v)$ then we say edge e is *joins* (also *connects*) u and v;

Example: consider the graph G_{ex_2} from §1.1.2:

- Edge i is incident with vertex v7, similarly edge i is incident with vertex v8;
- Edge d is incident with vertices v3 and v4;
- Edge c joins vertices v2 and v3;
- Edge h joins vertices v6 and v7;

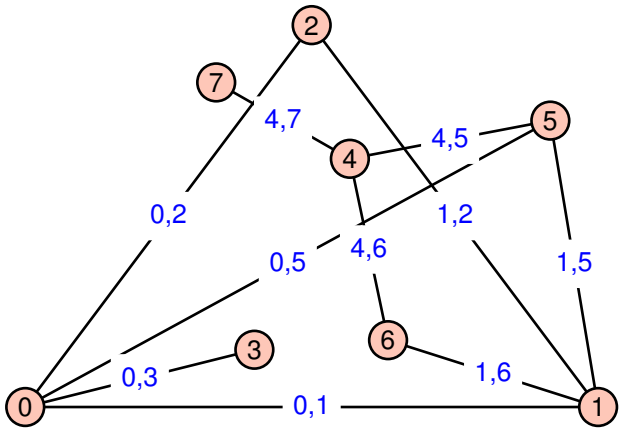
1.1.4 Example G_{ex_3}

To simplify notation we will often (but not always) define the vertex set of a graph as the integers $1, \ldots, n$ as done for G_{ex_1} in §1.1.1 or $0, \ldots, n-1$. The edge set will be denoted as size two subsets of the vertex set. For example G_{ex_3} defined as

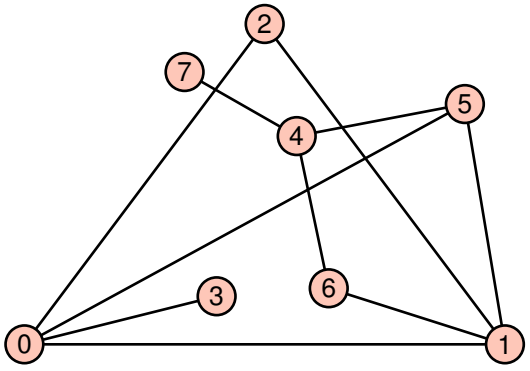
vertex set $V(G_{ex_3}) = \{0, 1, 2, 3, 4, 5, 6, 7\}$

edge set $E(G_{ex_3}) = \{\{0, 1\}, \{0, 2\}, \{0, 3\}, \{0, 5\}, \{1, 2\}, \{1, 5\}, \{1, 6\}, \{4, 5\}, \{4, 6\}, \{4, 7\}\}$.

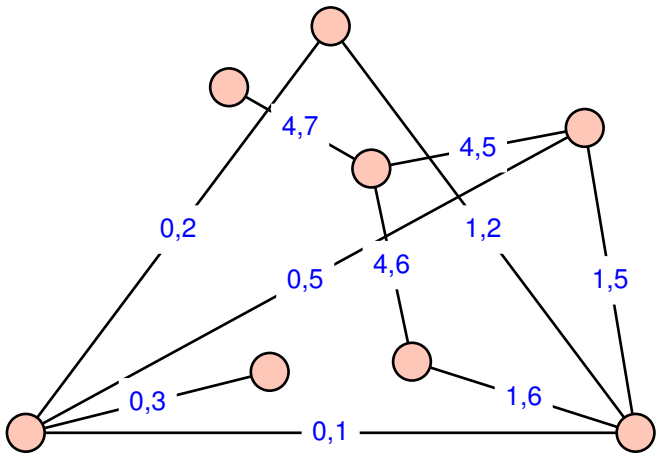
One embedding of G_{ex_3} is



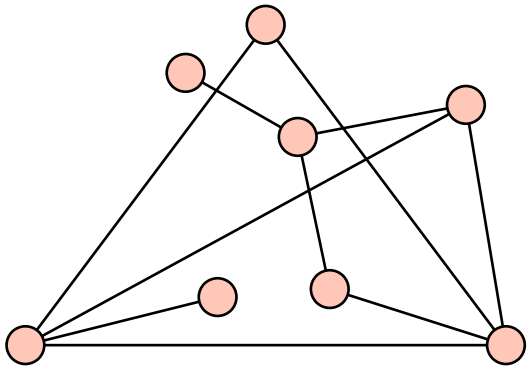
From the embedding it is trivial to associate edge $\{2, 6\}$ with its "line", therefore in embeddings we will omit the edge label. The corresponding embedding of G_{ex_3} is



While it may require a marginally larger effort a graph embedding may omit vertex labels and keep edge labels only. As before if necessary vertex labels can be deduced. The corresponding G_{ex_3} embedding is



Lastly, all label may be omitted such as



we will get back to this issue once isomorphic graphs are discussed.

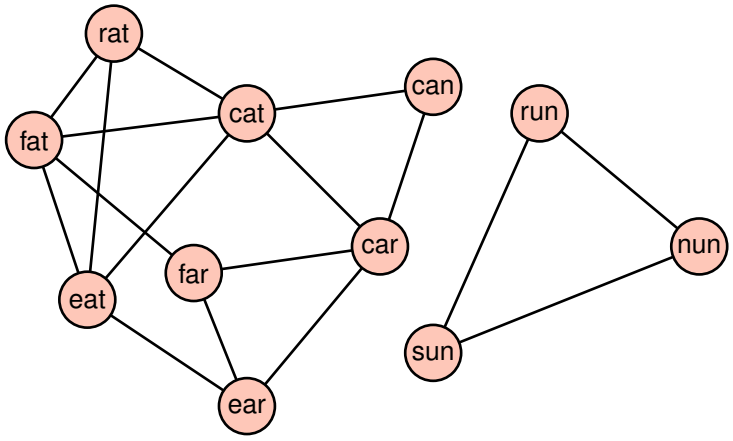
1.1.5 Example WG

Graphs are used to model (real world problems: for example page ranking of WWW pages) and often the description of the vertex set, the edge set or both is better conveyed by an approach different from exhaustive list. For example define a graph WG with vertex set

vertex set $V(WG) = \{$ can, car, cat, ear, eat, far, fat, nun, rat, run, sun $\}$

edge set two vertices are adjacent if and only if their strings differ in exactly one letter position-wise.

In the graph WG the vertices "can" and "car" are adjacent, but there is no edge incident with both vertex "fat" and vertex "nun". An embedding of WG is



The edge labels are omitted in the above embedding since they do not add any extra information in this case. In other scenarios only edge labels may be included and vertices left unlabeled.

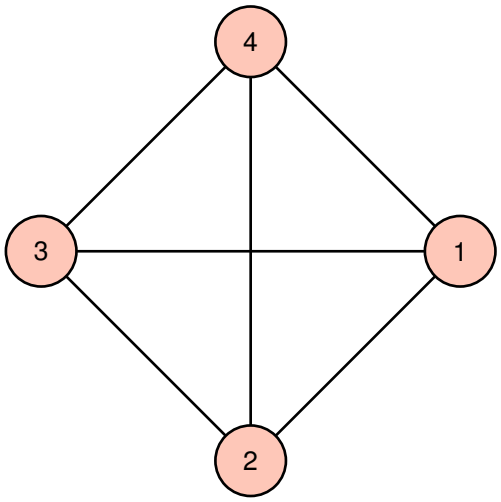
1.1.6 Example $S(n, k)$

Here is an example where both the vertex set and the edge set are not given as lists

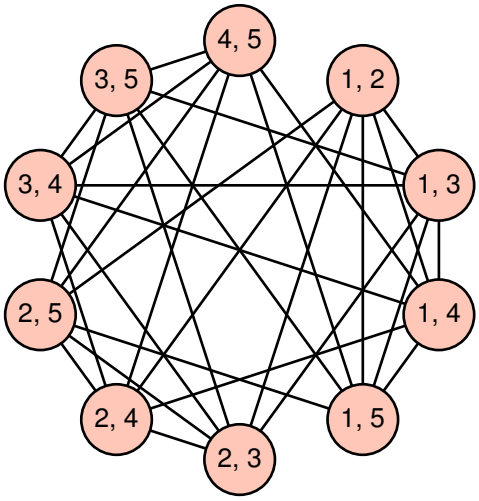
vertex set All k subsets of $\{1, \ldots, n\}$

edge set two vertices are adjacent if and only if their intersection contains exactly $k-1$ elements.

Here is an embedding of $S(4,1)$



and an embedding of $S(5,2)$



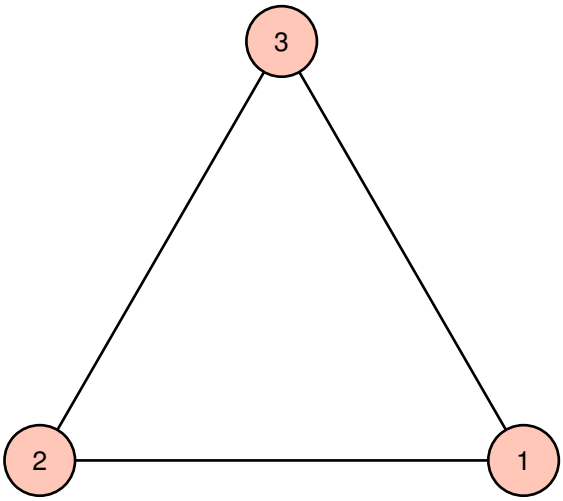
1.1.7 Example O_n

The odd graph O_n is the graph for which

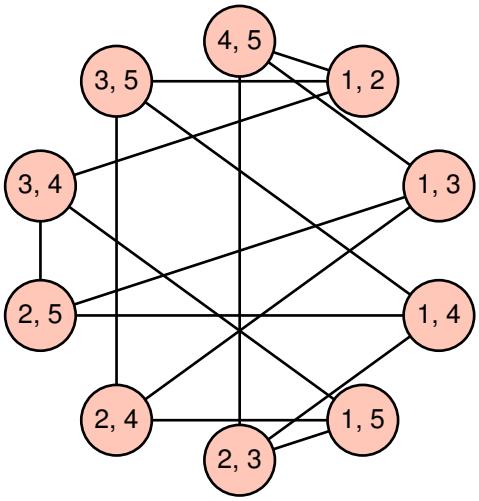
vertex set The vertex set is the set of n subsets of $\{1, \ldots, 2n+1\}$

edge set two vertices are adjacent if and only if the are disjoint

An embedding of O_1

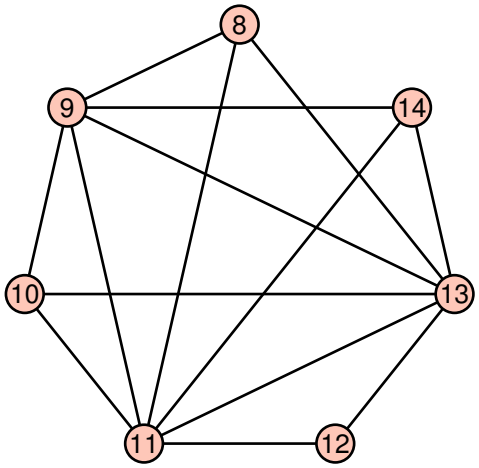


and O_2



1.1.8 Example $RPG(8, 15)$

Very often we will describe a graph via its one of its embeddings. Consider the graph $RPG(8, 15)$ with embedding



This is the graph with vertex set that contains all integers v such that

$$8 \leq v < 15$$

and two vertices are adjacent if and only if corresponding integers are relatively prime (i.e, their greatest common divisor is one).

1.1.9 Notation

The set theoretic symbol $\in$ and $\notin$ used to identify if an element belongs or not to a set is also used with graphs. For vertex u the notation $u \in G$ means that u is in the set of vertices of the graph G, likewise $u \notin G$ means vertex u is not in the vertex set of graph G. Similarly for an edge e the notation $e \in G$ says the edge e is in the edge set of G and the notation $e \notin G$ implies that the edge e is not in the edge set of G. To expand $uv \in G$ implies that for vertices u and v we have $u \in G$, $v \in G$ and the vertices u and v are adjacent i.e., there is an edge that joins u and v.

Example: for $RPG(8, 15)$ from §1.1.8 the following are true for vertices

- $3 \notin RPG(8, 15)$
- $9 \in RPG(8, 15)$
- $15 \notin RPG(8, 15)$
- $29 \notin RPG(8, 15)$

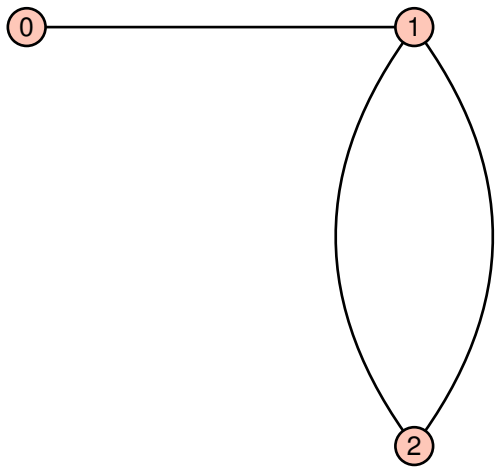
the following are true for edges

- $\{10, 13\} \in RPG(8, 15)$
- $\{10, 8\} \notin RPG(8, 15)$
- $\{8, 15\} \notin RPG(8, 15)$
- $\{3, 29\} \notin RPG(8, 15)$

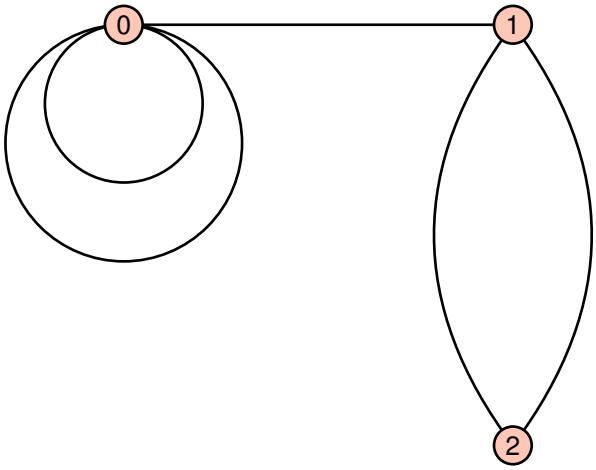
1.2 Simple finite graphs

If there are more than one edge incident with the same pair of edges then the graph has *multiple edges*; if an edge joins vertex with itself the edge is called a *loop*. The following terminology is widely adopted:

multigraph a graph with multiple edges and *no* loops, for example



pseudograph a graph with multiple edges and loops, for example



We exclude loops and multiple edges from our discussion.

Definition 2 (simple graphs)**.** *A graph is called* simple *if it has no loops and no multiple edges.*

The graph described in §1.1.8 can be generalized to $RPG(8, \infty)$ to mean the graph has vertex set all integers greater than or equal to eight. Such graph will have infinitely many vertices and infinitely many edges. We also exclude such graphs from our discussion.

Definition 3 (finite graphs)**.** *A graph is called* finite *if both the vertex set and the edge set are finite; otherwise the graph is called* infinite*.*

A simple graph can be both finite and infinite. Likewise an finite graph can be simple or alternatively contain loops or multiple edges. Those two ideas are independent from each other. From now on unless explicitly stated a graph will mean a simple finite graph. With every simple finite graph we associate order and size.

Definition 4 (order)**.** *The* order *of a graph G is the number of vertices in G.*

Definition 5 (size)**.** *The* size *of a graph G is the number of edges in G.*

Examples: for the graphs defined in §1.1

section	graph	order	size
§ 1.1.1	G_{ex_1}	4	3
§ 1.1.2	G_{ex_2}	8	9
§ 1.1.4	G_{ex_3}	8	10
§ 1.1.5	WG	11	17
§ 1.1.6	$S(4,1)$	4	4
§ 1.1.6	$S(5,2)$	10	30
§ 1.1.7	O_1	3	3
§ 1.1.7	O_2	10	15
§ 1.1.8	$RPG(8,15)$	7	14

The order of a graph is an integer that is greater than or equal to one (vertex set cannot be empty). The size is an non-negative integer.

1.3 Degree

For a vertex u the size of its the neigbourhood $N(u)$ plays central role.

Definition 6 (degree)**.** *The number of edges incident with a vertex u is called the* degree *of u and denoted by* $\deg(u)$.

In other words

$$\deg(u) = |N(u)|\,.$$

Examples:

- for WG we have

$$\begin{aligned}
\deg(can) &= 2\\
\deg(car) &= 4\\
\deg(cat) &= 5\\
\deg(ear) &= 3\\
\deg(eat) &= 4\\
\deg(far) &= 3\\
\deg(fat) &= 4\\
\deg(nun) &= 2\\
\deg(rat) &= 3\\
\deg(run) &= 2\\
\deg(sun) &= 2
\end{aligned}$$

- for $RPG(8, 15)$ we have

$$\begin{aligned} \deg(8) &= 3 \\ \deg(9) &= 5 \\ \deg(10) &= 3 \\ \deg(11) &= 6 \\ \deg(12) &= 2 \\ \deg(13) &= 6 \\ \deg(14) &= 3 \end{aligned}$$

- for $S(5, 2)$ the degree of every vertex is six.

The list of all degrees has its uses as far as graphs are concerned

Definition 7 (degree sequence)**.** *The degree sequence of a graph is the list of vertex degrees listed in a decreasing order as* $d_1 \geq d_2 \geq \cdots \geq d_n$

Examples

- The degree sequence of WG is

$$5, 4, 4, 4, 3, 3, 3, 2, 2, 2, 2$$

- The degree sequence of $RPG(8, 15)$ is

$$6, 6, 5, 3, 3, 3, 2$$

- The degree sequence of $S(5, 2)$ is

$$6, 6, 6, 6, 6, 6, 6, 6, 6, 6$$

Of particular interest are the first and the last entry of the degree sequence which correspond to the maximum and the minimum of all degrees

Definition 8 (minimum degree)**.** *The minimum degree of a graph* G *denoted by* $\delta(G)$ *is the minimum of the vertex degrees,*

$$\delta(G) = \min_{u \in V(G)} \deg(u)$$

Likewise

Definition 9 (maximum degree)**.** *The maximum degree of a graph* G *denoted by* $\Delta(G)$ *is the maximum of the vertex degrees,*

$$\Delta(G) = \max_{u \in V(G)} \deg(u)$$

Examples

- for WG we have $\delta(WG) = 2$ and $\Delta(WG) = 5$
- for $RPG(8, 15)$ we have $\delta(RPG(8, 15)) = 2$ and $\Delta(RPG(8, 15)) = 6$
- for $S(5, 2)$ we have $\delta(S(5, 2)) = \Delta(S(5, 2)) = 6$

For any graph G

$$\delta(G) \leq \Delta(G).$$

Graphs such as $S(5, 2)$ for which the minimum and the maximum degrees are equal to each other are a special class of graphs

Definition 10. *A graph G for which*

$$\delta(G) = \Delta(G) = r$$

*is called r-*regular *graph.*

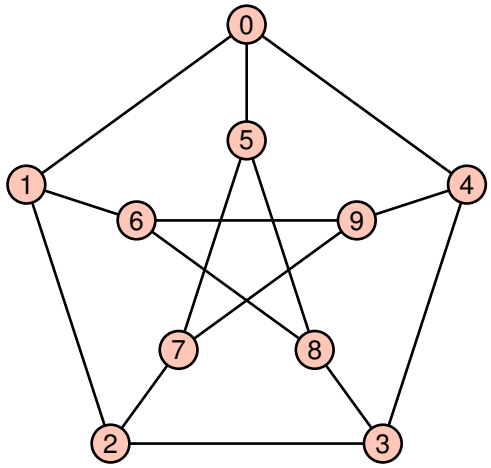
Example:

- the graph $S(4, 1)$ is 3-regular,
- the graph $S(5, 2)$ is 6-regular,
- the graph O_1 is 2-regular
- the graph O_2 is 3-regular

No other graphs in §1.1 regular, however those graphs are highly symmetric (which can formalized in mathematical term). We present a few more regular graph examples. The first one is the so called Petersen graph

1.3.1 Petersen graph

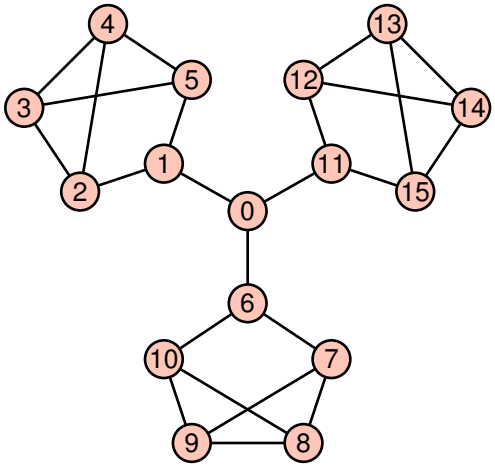
The Petersen graph is



It is also a highly symmetric graph.

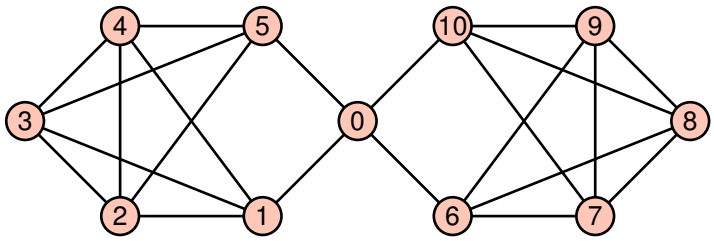
1.3.2 A 3-regular example

The following graph is a three regular graph that is not as symmetric as those in §1.1



1.3.3 A 4-regular example

The following graph is a four regular graph that is not as symmetric as those in §1.1



1.4 Handshake lemma

We next proceed with a very well known and powerful result concerning graphs.

Theorem 1 (Handshake lemma)**.** *For a graph G having e edges*

$$\sum_{u \in V(G)} d(u) = 2e$$

Proof. Consider the collection of unordered pairs $\{u, v\}$ where each pair corresponds to an edge in the graph. There are in total $2e$ vertices counted with multiplicities in the list. Another way to count the same value is look at how many times a vertex appears in a pair $\{u, v\}$, than answer is the degree of the vertex. Hence the result follows. □

Examples

- For WG which has 18 edges Handshake lemma says

$$5 + 4 + 4 + 4 + 3 + 3 + 3 + 2 + 2 + 2 + 2 = 2 \times 17$$

- For $RPG(8, 15)$ which has 14 edges Handshake lemma says

$$6 + 6 + 5 + 3 + 3 + 3 + 2 = 2 \times 14$$

- For $S(5,2)$ which has 30 edges Handshake lemma says

$$6+6+6+6+6+6+6+6+6+6 = 2 \times 30$$

Handshake lemma answers may question related to graphs such as

Question: Can we have a graph with 3 vertices of degree one and all other vertices of degree two?

Theorem 2. *The number of vertices of odd degree in a graph is even*

Proof. If a graph has odd number of odd degree vertices then $\sum_{u \in V(G)} \deg(u)$ is odd. Handshake lemma says the sum must be even, thus the number of odd degree vertices in a graph is necessarily even. □

The graph WG has two vertices of odd degree. The graph $RPG(8,15)$ has four vertices of odd degree. The graph $S(5,2)$ has zero vertices of odd degree.

1.5 Graph representation

Given the practical importance of graphs it is important to represent a graph in manner that is useful for computations. We will discuss three such representations:

1.5.1 Adjacency list

One way to store a graph is to keep for each vertex u a list of vertices that are adjacent to u (the *neighbourhood* of u).

Example: the graph WG has adjacency list

$$\begin{aligned}
can &\rightarrow [car, cat],\\
car &\rightarrow [can, cat, ear, far],\\
cat &\rightarrow [can, car, eat, fat, rat],\\
ear &\rightarrow [car, eat, far],\\
eat &\rightarrow [cat, ear, fat, rat],\\
far &\rightarrow [car, ear, fat],\\
fat &\rightarrow [cat, eat, far, rat],\\
nun &\rightarrow [run, sun],\\
rat &\rightarrow [cat, eat, far],\\
run &\rightarrow [nun, sun],\\
sun &\rightarrow [nun, run]
\end{aligned}$$

Example: the graph $RPG(8, 15)$ has adjacency list

$$\begin{aligned}
8 &\rightarrow [9, 11, 13],\\
9 &\rightarrow [8, 10, 11, 13, 14],\\
10 &\rightarrow [9, 11, 13],\\
11 &\rightarrow [8, 9, 10, 12, 13, 14],\\
12 &\rightarrow [11, 13],\\
13 &\rightarrow [8, 9, 10, 11, 12, 14],\\
14 &\rightarrow [9, 11, 13]
\end{aligned}$$

Example: the graph $S(5,2)$ has adjacency list

$$\begin{aligned}
\{4,5\} &\rightarrow [\{3,5\},\{3,4\},\{2,5\},\{2,4\},\{1,5\},\{1,4\}],\\
\{3,5\} &\rightarrow [\{4,5\},\{3,4\},\{2,5\},\{2,3\},\{1,5\},\{1,3\}],\\
\{3,4\} &\rightarrow [\{4,5\},\{3,5\},\{2,4\},\{2,3\},\{1,4\},\{1,3\}],\\
\{2,5\} &\rightarrow [\{4,5\},\{3,5\},\{2,4\},\{2,3\},\{1,5\},\{1,2\}],\\
\{2,4\} &\rightarrow [\{4,5\},\{3,4\},\{2,5\},\{2,3\},\{1,4\},\{1,2\}],\\
\{2,3\} &\rightarrow [\{3,5\},\{3,4\},\{2,5\},\{2,4\},\{1,3\},\{1,2\}],\\
\{1,5\} &\rightarrow [\{4,5\},\{3,5\},\{2,5\},\{1,4\},\{1,3\},\{1,2\}],\\
\{1,4\} &\rightarrow [\{4,5\},\{3,4\},\{2,4\},\{1,5\},\{1,3\},\{1,2\}],\\
\{1,3\} &\rightarrow [\{3,5\},\{3,4\},\{2,3\},\{1,5\},\{1,4\},\{1,2\}],\\
\{1,2\} &\rightarrow [\{2,5\},\{2,4\},\{2,3\},\{1,5\},\{1,4\},\{1,3\}]
\end{aligned}$$

The more edges a graph has the more memory requirement there will be to store that graph. For so called *dense* graphs, that is graphs with a lot of edges, rather than storing the list of neighbors, it is more efficient if one stores the list of vertices that are *not* adjacent to a given vertex. Such idea motivates

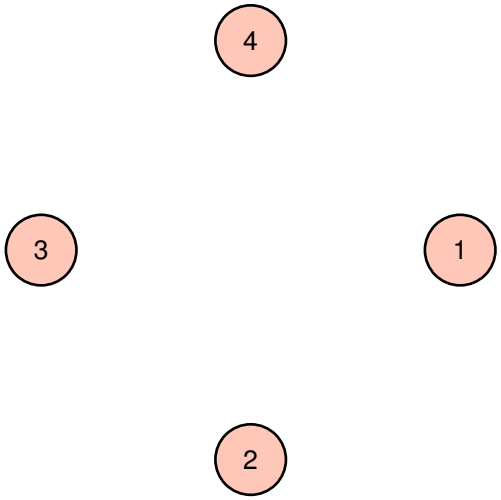
Definition 11 (Complement of a graph)**.** *Let $G = (V,E)$ be a graph with vertex set V and edge set E. The* complement *of G is the graph $\overline{G} = (V,E')$ where*

$$\{u,v\} \in E' \quad \Leftrightarrow \quad \{u,v\} \notin E$$

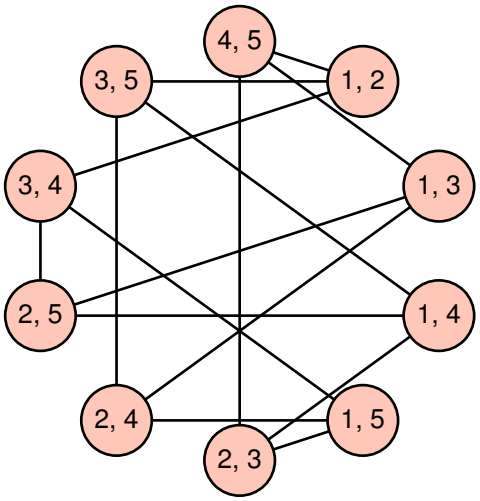
that is u and v are adjacent in $\overline{G}$ if and only if u and v are not adjacent in G.

In other words rather than storing the adjacency list of store the adjacency list of the complement of the graph.

Example: the complement of $S(4,1)$ denoted by $\overline{S(4,1)}$ is



and the complement of $S(5,2)$ denoted by $\overline{S(5,2)}$ is



The adjacency list of $\overline{S(5,2)}$ is

$$\begin{aligned}
\{4,5\} &\rightarrow [\{2,3\},\{1,3\},\{1,2\}],\\
\{3,5\} &\rightarrow [\{2,4\},\{1,4\},\{1,2\}],\\
\{3,4\} &\rightarrow [\{2,5\},\{1,5\},\{1,2\}],\\
\{2,5\} &\rightarrow [\{3,4\},\{1,4\},\{1,3\}],\\
\{2,4\} &\rightarrow [\{3,5\},\{1,5\},\{1,3\}],\\
\{2,3\} &\rightarrow [\{4,5\},\{1,5\},\{1,4\}],\\
\{1,5\} &\rightarrow [\{3,4\},\{2,4\},\{2,3\}],\\
\{1,4\} &\rightarrow [\{3,5\},\{2,5\},\{2,3\}],\\
\{1,3\} &\rightarrow [\{4,5\},\{2,5\},\{2,4\}],\\
\{1,2\} &\rightarrow [\{4,5\},\{3,5\},\{3,4\}]
\end{aligned}$$

and since $\overline{S(5,2)}$ has fewer edges than $S(5,2)$ it is has smaller memory requirements.

1.5.2 Adjacency matrix

The second representation is based on the adjacency relation

Definition 12 (adjacency matrix)**.** *Let G be a graph with vertex set $V(G) = \{v_1,\ldots,v_n\}$. The* adjacency matrix *of G is a $n \times n$ matrix $A = \{a_{ij}\}$ where $a_{ij} = 1$ if v_i and v_j are adjacent and zero otherwise.*

Example: the adjacency matrix G_{ex_1} from §1.1.1 is

$$A_{G_{ex_1}} = \begin{pmatrix} 0 & 1 & 0 & 1\\ 1 & 0 & 1 & 1\\ 0 & 1 & 0 & 1\\ 1 & 1 & 1 & 0 \end{pmatrix}$$

In the matrix row i corresponds to vertex i and column j corresponds to vertex j. The entry $A_{G_{ex_1}}[1,3] = 0$ implies there is no edge between vertex one and vertex three. If there is no edge incident with vertex one and vertex three then also entry $A_{G_{ex_1}}[3,1] = 0$. Entry $A_{G_{ex_1}}[2,3] = A_{G_{ex_1}}[3,2] = 1$ implies there is an edge between vertex two and vertex three. In general for an adjacency matrix A we have $A[i,j] = A[j,i]$ which means $A = A^t$, that is the adjacency matrix of a graph is symmetric. For graphs we consider, there are no loops so diagonal entries are all zeroes.

Example: the adjacency matrix $RPG(8, 15)$ is

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

where row i corresponds to vertex $i+7$ and column j corresponds to vertex $j + 7$.

Example: the adjacency matrix $S(4, 1)$ is

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

and the adjacency matrix of its complement is

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Naturally, there is a relation between the adjacency matrix of a graph and the adjacency matrix of its complement: for off diagonal entries zeroes and ones are flipped.

Example: the adjacency matrix $S(5, 2)$ is

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

and the adjacency matrix of $\overline{S(5,2)}$ is

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Remark: Matrices do not carry information about the labels on the graphs. Depending on the association of rows with vertices a graph may have more than one adjacency matrix, for example if row one is associated with vertex five and vertex five with row one the result will be a different adjacency matrix.

1.5.3 Incidence matrix

The last graph representation is based on the incidence relation of a graph:

Definition 13 (incidence matrix)**.** *Let G be a graph with vertex set $V(G) = \{v_1, \ldots, v_n\}$ and edge set $E(G) = \{e_1, \ldots, e_k\}$. The* incidence matrix *of G is a $n \times k$ matrix $B = \{b_{ij}\}$ where $b_{ij} = 1$ if v_i is incident with e_j and zero otherwise.*

Example: G_{ex_1} has incidence matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

where as its complement has incidence matrix

$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

which once again shows that dense graphs are better stored via their complement. Observe however that unlike adjacency matrix the to obtain incidence matrix of a graph from the incidence matrix of its complement requires some work.

Example: G_{ex_2} has incidence matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Example: $S(4,1)$ has incidence matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Example: $RPG(8,15)$ has incidence matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Remark: another way to see the Handshake lemma is to count the number of entries that equals one in the incidence matrix of a graph.

1.5.4 Relations between representations

The order of a graph is the order of its adjacency matrix. The size of a graph is the number of columns of its incidence matrix. For example the incidence matrix of $S(5,2)$ has a total of 30 columns. One can easily switch from one representation to the other. The next result shows a further relation between the notions that were presented so far.

Theorem 3. *For a graph G with adjacency matrix A and incidence matrix B we have*

$$BB^T = A + \begin{pmatrix} d(v_1) & 0 & \dots & 0 \\ 0 & d(v_2) & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & d(v_n) \end{pmatrix}$$

Proof. Self study problem. □

1.6 Graph isomorphism

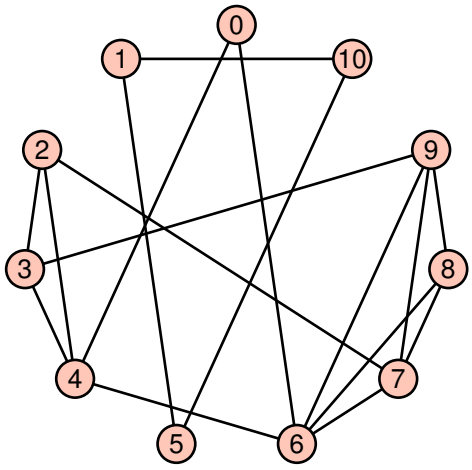
Labels. So far we considered special examples and for easiness we mostly labeled the vertices. However, whether we call a vertex v_0 or 1 is not important. The incidence structure of the graph is important. On the WWW "http://www.iyte.edu.tr" is human friendly reference but the underlying algorithms call it "193.140.249.2". In that respect two graphs are the same if and only if they have the same incidence structure, the labels on the vertices and edges is not important. This is captured in the following definition.

Definition 14 (isomorphic graphs)**.** *Two graphs G_1 and G_2 are* isomorphic *if and only if there exists a bijection $f : V(G_1) \to V(G_2)$ such that*

$$(u, v) \in E(G_1) \iff (f(u), f(v)) \in E(G_2).$$

In that case we call f an isomorphism from G_1 to G_2.

Example: the graph WG from §1.1.5 and the graph WG_2



are isomorphic via the isomorphism $f : V(WG_1) \to V(WG)$

u	0	1	2	3	4	5	6	7	8	9	10
$f(u)$	can	sun	ear	far	car	nun	cat	eat	rat	fat	run

Since isomorphisms are invertible with inverses that are also isomorphisms, graph isomorphisms is an equivalence relation (transitive, symmetric and reflexive relation).

Moreover, the isomorphism is not necessarily unique. Here is an other isomorphism from WG_1 to WG:

u	0	1	2	3	4	5	6	7	8	9	10
$g(u)$	can	nun	ear	far	car	sun	cat	eat	rat	fat	nun

Example: The graph $\overline{S(5,2)}$ and the graph O_2 have the same set of vertices and are isomorphic via the identity map

$$id\,(\{p,q\}) = \{p,q\}.$$

Furthermore O_2 and the Petersen graph are isomorphic via the map

u	0	1	2	3	4
$f(u)$	$\{1,2\}$	$\{4,5\}$	$\{2,3\}$	$\{1,4\}$	$\{3,5\}$
u	5	6	7	8	9
$f(u)$	$\{3,4\}$	$\{1,3\}$	$\{1,5\}$	$\{2,5\}$	$\{2,4\}$

Example: A bijection has to one-to-one which means that for two graphs to be isomorphic they must have the same number of vertices. Therefore G_{ex_1} and G_{ex_2} cannot be isomorphic.

Example: G_{ex_2} and G_{ex_3} have the same order, but they cannot be isomorphic since they have a different number of edges. Alternatively, G_{ex_2} has no vertices of degree three, whereas $5 \in G_{ex_3}$ has degree three. Since adjacency is preserved under graph isomorphism, this is another reason why the two graphs are *not* isomorphic.

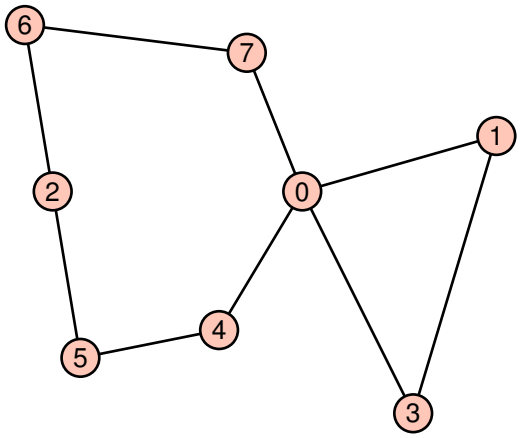
Remark: (sub)graph isomorphism problem is a highly non-trivial problem! Some of the most exciting developments related algorithms in recent times are precisely related to the hardness of this problem. On the one hand if you need to show that two graphs are *not* isomorphic it suffices to find

one property in one of the graphs that is not present in the second graph. For example one of the graphs has a vertex of degree three the other has no vertices of degree three. On the other hand if you need to show that two graphs are isomorphic you *must* give an isomorphism between these graphs as done above. Do not fall into the trap of listing a few (however many) graph results and claim that those are enough to show that graphs are isomorphic.

Once again a claim like:

> any graph with a degree sequence
>
> $$4, 2, 2, 2, 2, 2, 2, 2$$
>
> must be isomorphic to G_{ex_2}

is false. While the above statement implicitly give the graph order (eight) and size (nine) it does not guarantee isomorphism. For this particular example the graph

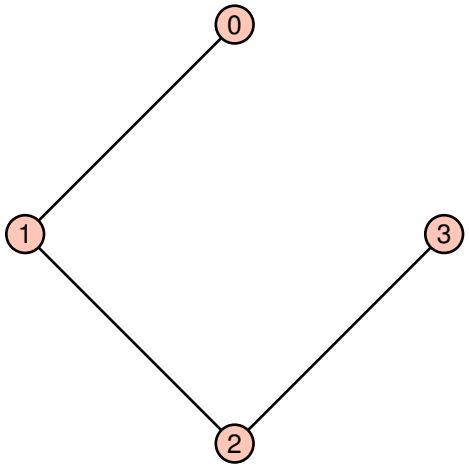


have the same size, order and degree sequence, nevertheless these two graphs are *not* isomorphic – in the graph above the vertex of degree four has two neighbours that are adjacent where as all neighbours of vertex $v1 \in G_{ex_2}$ which has degree four are pairwise not adjacent.

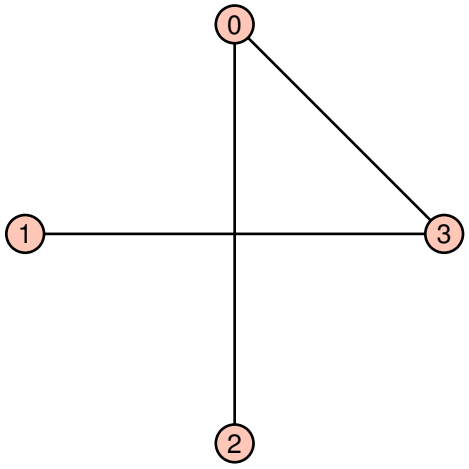
The graph $S(5, 2)$ has size 30 and its complement has size 15. Therefore $S(5, 2)$ cannot be isomorphic to $\overline{S(5, 2)}$. Graphs for which $G \equiv \overline{G}$ have special name.

Definition 15. *A graphs is* self-complementary *if it is isomorphic to its complement.*

Example: the graph P_4



is isomorphic to its complement



via the isomorphism

0	1	2	3
2	0	3	1

Thus P_4 is a self complementary graph.

Remark: whenever a graph is described via an embedding with no labels as done in §1.1.4, it will refer to any graph from the isomorphism class that has the illustrated adjacency structure.

1.7 Walks, paths and cycles

We start with a general definition that is later specialized

Definition 16 (walk). *A* walk *from vertex a to vertex b in a graph G is an alternating sequence $v_0e_1v_1e_2v_2\dots e_nv_n$ of vertices $v_0, v_1, \dots, v_n$ and edges $e_1, e_2, \dots, e_n$ in G such that $a = v_0$, $b = v_n$ and for all $1 \leq i \leq n$ edge $e_i = (v_{i-1}v_i)$;*

Recall G_{ex_2} from §1.1.2 the following sequence W is a walk:

$$W = v7, i, v8, j, v1, a, v2, c, v3, d, v4, b, v1, a, v2, a, v1, f, v5$$

Similar to embeddings a walk can be given just as list of vertices, for the walk W above

$$W = v7, v8, v1, v2, v3, v4, v1, v2, v1, v5$$

or as a list of edges, for the walk W above

$$W = i, j, a, c, d, b, a, a, f$$

the relation is unambiguous so depending on the focus all of the above are equivalent and either can be recovered by the others

$$\begin{aligned} W &= v7, i, v8, j, v1, a, v2, c, v3, d, v4, b, v1, a, v2, a, v1, f, v5 \\ &= v7, v8, v1, v2, v3, v4, v1, v2, v1, v5 \\ &= i, j, a, c, d, b, a, a, f \end{aligned}$$

If a walk starts with vertex a and ends with vertex b it is called an ab walk. The walk W above is a $(v7, v5)$ walk. We will typically denote an ab walk as W_{ab}. For graphs that we discussed so far, edges have no "direction" so if we reverse the sequence of a walk we still have a walk. In that case if the original walk is ab walk the reverse sequence is a ba walk. The reverse walk is denoted by W^{-1} for the example above we have

$$\begin{aligned} W^{-1} &= v5, f, v1, a, v2, a, v1, b, v4, d, v3, c, v2, a, v1, j, v8, i, v7 \\ &= v5, v1, v2, v1, v4, v3, v2, v1, v8, v7 \\ &= f, a, a, b, d, c, a, j, i \end{aligned}$$

Further W can be denoted as $W_{v7,v5}$ and $W^{-1}_{v7,v5} = W_{v5,v7}$.

Definition 17 (length of a walk)**.** *The* length of a walk $W = v_0 e_1 \dots e_n v_n$ *is the number of edges in the walk.*

The length of W given above is nine, since it contains nine edges.

Definition 18 (trail)**.** *A* trail *is a walk with no repeated edges.*

The walk W above contains edge a three times and is therefore *not* a trail. We can obtain a trail from a walk as follows:

$$
\begin{array}{rcl}
1 & : & v7, i, v8, j, v1, a, v2, c, v3, d, v4, b, v1, \overbrace{a, v2, a, v1,}^{\text{remove}} f, v5 \\
2 & : & v7, i, v8, j, v1, a, v2, c, v3, d, v4, b, \underbrace{v1, \qquad\qquad f}_{\text{glue}}, v5 \\
3 & : & v7, i, v8, j, v1, a, v2, c, v3, d, v4, b, v1, f, v5
\end{array}
$$

The result in the last step

$$
\begin{aligned}
W_2 &= v7, i, v8, j, v1, a, v2, c, v3, d, v4, b, v1, f, v5 \\
&= v7, v8, v1, v2, v3, v4, v1, v5 \\
&= i, j, a, c, d, b, f
\end{aligned}
$$

has no repeated edges and thus W_2 is a trail; its length is seven.

Remark: if a walk has a repeated edge then it necessarily has a repeated vertex. But a walk may have repeated vertices and have no repeated edges.

Definition 19 (path)**.** *A* path *is a walk with no repeated vertices.*

Remark: a path cannot have repeated edges since if an edge appears at least twice on a walk at least one vertex appears twice on the walk.

One can obtain a path from any walk/trail using the above procedure:

$$
\begin{array}{rcl}
1 & : & v7, i, v8, j, v1, \overbrace{a, v2, c, v3, d, v4, b, v1,}^{\text{remove}} f, v5 \\
2 & : & v7, i, v8, j, v1, \underbrace{\qquad\qquad\qquad f}_{\text{glue}}, v5 \\
3 & : & v7, i, v8, j, v1, f, v5
\end{array}
$$

The result is $v7, v5$-path

$$\begin{aligned} P_{v7,v5} &= v7, i, v8, j, v1, f, v5 \\ &= i, j, f \\ &= v7, v8, v1, v5 \end{aligned}$$

The length of the path is three.

In the above example there is only one repeated vertex but the procedure should be executed for all repeated vertices to obtain a path, which shows the following.

Theorem 4. *In a graph G if there is an ab-walk, then there is an ab-path.*

Proof. Formally induction on the length of the walk. If the walk has length one then it is a path. Suppose every ab walk of length at most n contains an ab path. Let W be any ab walk of length $m = n + 1$. If W is a path nothing to show. If W is not a path it has a repeated vertex say u. Let $W = v_0e_1v_1 \dots v_{m-1}e_mv_m$ that is $v_0 = a$, $v_m = b$ and $m \geq n + 1$ suppose $u = v_i$ and $u = v_j$ with $i < j$. Then from

$$W = v_0e_1v_1 \dots v_{i-1}e_iv_i \dots v_{j-1}e_jv_je_{j+1}v_{j+1} \dots v_{m-1}e_mv_m$$

obtain

$$\tilde{W} = v_0e_1v_1 \dots v_{i-1}e_iv_ie_{j+1}v_{j+1} \dots v_{m-1}e_mv_m$$

which is a well defined walk since e_{j+1} is incident with $v_j = v_i$. The length of $\tilde{W}$ is less than or equal to the length of W so its length is at most n. By induction $\tilde{W}$ contains a ab-path. And since $\tilde{W}$ is contained in W then W contains a ab-path, which concludes the argument. □

Remark: in the above argument we used the term *contain*. It means, as done in the above examples, one walk is obtain from another by a sequence of remove and glue steps.

Theorem 5. *If there is a path from a to b and from b to c in G then there is a path from a to c in G.*

Proof. Suppose there is a ab-path

$$P_{ab} = av_1, \dots, v_{n-1}b$$

and a bc-path

$$P_{bc} = bu_1, \dots, u_{m-1}c.$$

Then

$$W_{ac} = av_1, \ldots, v_{n-1}bu_1, \ldots, u_{m-1}c$$

is a ac-walk and by Theorem 4 the result follows. □

Definition 20 (closed walk)**.** *A walk* $v_0e_1 \ldots e_nv_n$ *is* closed *if* $v_0 = v_n$*.*

Recall G_{ex_3} from §1.1.4 the following sequence W is a closed walk:

$$W = 1, 2, 0, 3, 0, 1, 5, 4, 7, 4, 6, 1, 5, 1$$

The length of this walk is thirteen. This walk has both repeated edges, for example edge $\{4, 7\}$ appears twice, and repeated vertices, for example vertex 1 appear three times.

Definition 21 (circuit)**.** *A* circuit *is a closed trail.*

The above example is not a circuit but we can obtain a circuit from it.

$$\begin{aligned}
0 \quad &: \quad 1, 2, 0, 3, 0, 1, 5, \overbrace{4, 7,}^{\text{remove}} \ 4, 6, 1, 5, 2 \\
1 \quad &: \quad 1, 2, 0, 3, 0, 1, 5, \underbrace{\quad}_{\text{glue}} 4, 6, 1, 5, 1 \\
2 \quad &: \quad 1, 2, \overbrace{0, 3,}^{\text{remove}} \ 0, 1, 5, 4, 6, 1, 5, 1 \\
3 \quad &: \quad 1, 2, \underbrace{\quad}_{\text{glue}} 0, 1, 5, 4, 6, 1, 5, 1 \\
4 \quad &: \quad 1, 2, 0, 1, 5, 4, 6, 1 \overbrace{, 5, 1}^{\text{remove}} \\
\rightarrow \quad &\quad 1, 2, 0, 1, 5, 4, 6, 1
\end{aligned}$$

The result is a circuit $1, 2, 0, 1, 5, 4, 6, 1$ of length seven. By the nature of closed walks there is always a repeated vertex: initial vertex is the same as the last vertex of the walk. The class of closed walks for which the only repeated vertex is the first and the last one is of special importance.

Definition 22 (cycle)**.** *A* cycle *is a closed walk* $v_0e_1v_1 \ldots, v_{n-1}e_nv_n$ *such that vertices* $v_0, \ldots, v_{n-1}$ *are all distinct.*

Note that the index of vertices reaches $n - 1$ not n. For closed walks $v_0 = v_n$.

The circuit $1, 2, 0, 1, 5, 4, 7, 1$ is not a cycle since vertex 1 appear as the forth vertex in the sequence. So there are two vertices in the set $v_0, \ldots, v_{n-1}$ that are the same. However, that circuit contains two cycles:

$$1, 2, 0, 1 \quad \text{and} \quad 1, 5, 4, 6, 1$$

For any closed walk the first and the last vertex are always the same. For this reason when describing a closed walk the last vertex will be omitted. The cycle $1, 2, 0, 1$ will be written as $1, 2, 0$ and the cycle $1, 5, 4, 6, 1$ would be identified with $1, 5, 4, 6$. Note also that the cycle $1, 2, 0$ is the same as the cycle $2, 0, 1$ and the cycle $0, 1, 2$. Likewise, $1, 5, 4, 6$, $5, 4, 6, 1$, $4, 6, 1, 5$ and $6, 1, 5, 4$ are the same cycle.

Remark: given a sequence $1, 2, 0$ it is important to distinguish this sequence as a path, which has length two, from the cycle, which has length three. The path with sequence $1, 2, 0$ corresponds to the walk

$$1\{1, 2\}2\{0, 2\}0$$

whereas the cycle $1, 2, 0$ corresponds to the walk

$$1\{1, 2\}2\{0, 2\}0\{0, 1\}1.$$

Remark: some text identify circuits with cycles. Here we distinguish those two terms.

1.8 Special graphs

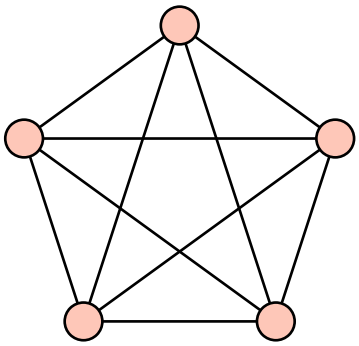
The following graphs are frequently encountered.

1.8.1 Empty graph E_n

The graph on n vertices with no edges is called the *empty* graph E_n. Then E_1 is a single vertex, E_2 is a pair of vertices, E_3 has three vertices and so forth. An embedding of E_3 is for example

1.8.2 Complete graph K_n

The graph on n vertices, where each pair of vertices is adjacent is called the *complete* graph K_n. For $n = 1$ the graph K_1 is a single vertex and isomorphic to E_1, K_2 is a single edge, K_3 is a cycle of length three, K_4 is isomorphic to $S(4, 1)$; K_5 is

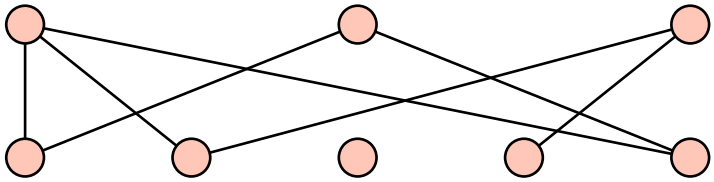


1.8.3 Bipartite graphs

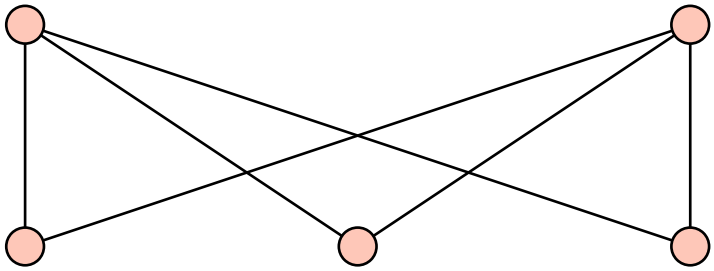
Partition: recall a partition of a set S is a collection of its subsets $U_1, \ldots, U_n$ such that: *(i)* if $i \neq j$ then $U_i \cap U_j = \emptyset$; and *(ii)* $S = U_1 \cup U_2 \cup \cdots \cup U_n$.

Definition 23 (bipartite graph)**.** *A graph G whose vertex set can be partitioned into two sets A and B such that every edge in $E(G)$ is incident with one vertex in A and one vertex in B is called* bipartite *graph. The sets A and B are called a* bipartition*.*

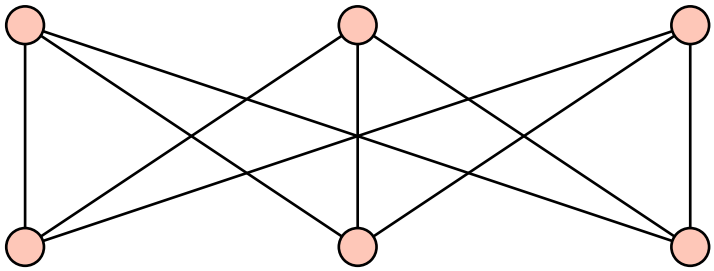
Bipartite graphs encountered so far: P_4, for any n the graph E_n, the graph K_2. Here is another bipartite graph:



If for a bipartite graph with bipartition A and B, every vertex in A is adjacent to every vertex in B the graph is known as the *complete bipartite graph*. Further if the number of elements in A is n and the number of elements in B is m the graph is denoted by $K_{n,m}$ for example $K_{2,3}$ is



and $K_{3,3}$ is

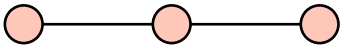


Observe that K_2 is the same as $K_{1,1}$.

Bipartite graphs have some interesting properties and at the same time are used to model quite a few real world problems: for example major search engines use bipartite graphs to sell ad space. Bidding by advertisers for keywords is settled as a bipartite graph problem: the vertices in one partition are the advertisers, the vertices in the other partition are the keywords and the edges are the bids advertisers place on queries.

1.8.4 Path P_n

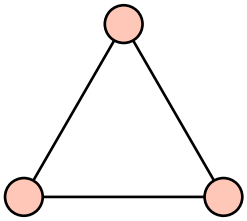
A path of length $n-1$ will be denoted as P_n, such paths can be viewed as graphs. For example P_1 is one vertex, that is P_1 is isomorphic to E_1 and K_1. For P_3 we have



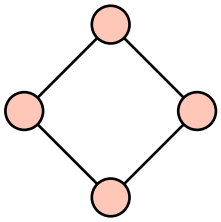
For any n the graph P_n is bipartite graph.

1.8.5 Cycle C_n

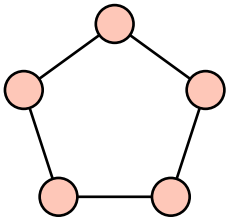
A cycle of length $n \geq 3$ will be denoted as C_n, such cycle can be viewed as graphs. For example C_3 is



For obvious reasons C_3 is often called a triangle; further C_3 is isomorphic to K_3. For C_4 we have



and for C_5 we have



The graph C_n is bipartite if and only if n is even number.

1.9 Common graph measures

We describe some common terms derived about graphs

Definition 24 (girth)**.** *The girth of a graph is the length of a cycle of the shortest length. If the graph has no cycles we will say its girth is $+\infty$ i.e., positive infinity.*

Remark: for a graph without cycles some text may assign alternative value for the girth.

Example: G_{ex_1} contains three cycles:

1. $1, 2, 4$ of length three;
2. $2, 3, 4$ of length three; and
3. $1, 2, 3, 4$ of length four.

The shortest cycle is of length three thus its girth is three.

Example: G_{ex_2} contains two cycles:

1. $v1, v2, v3, v4$ of length four.
2. $v1, v5, v6, v7, v8$ of length five.

The shortest cycle is of length four thus its girth is four.

Examples: the girth of WG is three, the girth of the Petersen graph is five. C_n has girth n, P_n has girth ∞.

Definition 25 (distance)**.** *Let G be a graph and u and v be two vertices of G the* distance *between u and v denoted by $d(u, v)$ is the length of a shortest u, v-path.*

Example: consider G_{ex_3}. The distance between vertex 7 and vertex 3 is four, since the path $7, 4, 5, 0, 3$ is a $7, 3$-path of length four and no shorter path from 7 to 3 exists. Likewise, the distance between $3, 7$ is also four. In general it is straightforward to justify that $d(u, v) = d(v, u)$.

For the same graph $d(7, 1) = 3$ and there are two $7, 1$-paths of length three: $1, 6, 4, 7$ and $1, 5, 4, 7$. Here is the list of all distances between pair of vertices:

	0	1	2	3	4	5	6	7
0	0	1	1	1	2	1	2	3
1	1	0	1	2	2	1	1	3
2	1	1	0	2	3	2	2	4
3	1	2	2	0	3	2	3	4
4	2	2	3	3	0	1	1	1
5	1	1	2	2	1	0	2	2
6	2	1	2	3	1	2	0	2
7	3	3	4	4	1	2	2	0

Example: the table for graph WG is

	can	car	cat	ear	eat	far	fat	nun	rat	run	sun
can	0	1	1	2	2	2	2	∞	2	∞	∞
car	1	0	1	1	2	1	2	∞	2	∞	∞
cat	1	1	0	2	1	2	1	∞	1	∞	∞
ear	2	1	2	0	1	1	2	∞	2	∞	∞
eat	2	2	1	1	0	2	1	∞	1	∞	∞
far	2	1	2	1	2	0	1	∞	2	∞	∞
fat	2	2	1	2	1	1	0	∞	1	∞	∞
nun	∞	∞	∞	∞	∞	∞	∞	0	∞	1	1
rat	2	2	1	2	1	2	1	∞	0	∞	∞
run	∞	∞	∞	∞	∞	∞	∞	1	∞	0	1
sun	∞	∞	∞	∞	∞	∞	∞	1	∞	1	0

Definition 26 (eccentricity)**.** *Let G be a graph and u be a vertex of G. The* eccentricity *of u is the maximum distance from u to any vertex in G that is*

$$e(u) = \max_{v \in V(G)} d(u, v)$$

Example: in a table of distance as above the maximum of each row (or column since the table is symmetric along its diagonal) is the eccentricity of the corresponding vertex. We have

- for G_{ex_3}

u	0	1	2	3	4	5	6	7
$e(u)$	3	3	4	4	3	2	3	4

- for WG

u	can	car	cat	ear	eat	far	fat	nun	rat	run	sun
$e(u)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

Definition 27 (diameter)**.** *Let G be a graph. The maximum of all eccentricity of the vertices of G is the* diameter *of G*

$$d(G) = \max_{u \in V(G)} e(u)$$

Example: the maximum of the second row of each table of eccentricities as above is the diameter of the graph. We have

- G_{ex_3} has diameter four;
- WG has diameter ∞.

Further we have

- Petersen graph has diameter two;
- K_n has diameter one;
- P_n has diameter $n-1$
- C_n has diameter $\left\lfloor \frac{n}{2} \right\rfloor$;

Theorem 6. *If a simple graph G has diameter at least three then its complement has diameter at most three.*

Proof. If diameter of G is at least three it means there are two vertices u and v such that they do not have a common neighbour. Then for any vertex in $x \in V(G) - \{u, v\}$ the vertex x is adjacent to at most one of u or v in G. Thus in the complement of G the vertex x is adjacent to at least on of u or v. Then for any two vertices x and y in the complement of G they either have the same neighbour in $\{u, v\}$ or since the edge uv is in the complement of G there is path of length three along the edge uv from x to y. □

Definition 28 (radius)**.** *Let G be a graph. The minimum of all eccentricity of the vertices of G is the* radius *of G*

$$r(G) = \min_{u \in V(G)} e(u)$$

Example: the minimum of the second row of each table of eccentricities as above is the radius of the graph. We have

- G_{ex_3} has radius two;
- WG has radius ∞.

Further we have

- Petersen graph has radius two;
- K_n has radius one;

- P_n has radius $\left\lfloor \frac{n-1}{2} \right\rfloor$;
- C_n has radius $\left\lfloor \frac{n}{2} \right\rfloor$;

Definition 29 (center)**.** *The set of all vertices in a graph G with minimum eccentricity is the* center *of G*

$$C(G) = \{u \in V(G) \mid e(u) = r(G)\}$$

Examples: the center of

- G_{ex_3} is the set $\{5\}$;
- WG is the set $V(WG)$;
- similar to WG the graphs K_n, C_n and Petersen graph have center the set of their vertices
- the path P_{10} with vertex set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, where i is adjacent to $i + 1$ and $i - 1$ has center

$$\{4, 5\}$$

1.10 Subgraphs

Definition 30 (subgraph)**.** *A subgraph G' of a graph G is a graph such that $V(G') \subseteq V(G)$ and the edges of G' are subset of the edges of G such that each edge in G' is incident only with vertices in $V(G')$.*

Remark: at a superficial glance the definition seems to have two conditions. However, it has three conditions:

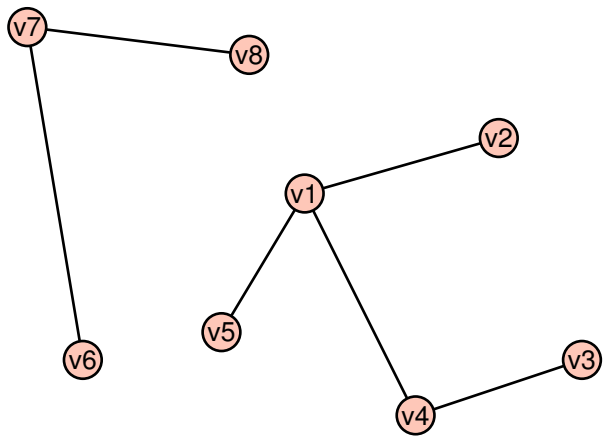
- the vertex set of the subgraph is a subset of the vertex set of the graph;
- the edge set of the subgraph is a subset of the edge set of the graph; *and*
- the subgraph itself must be a graph.

For example,

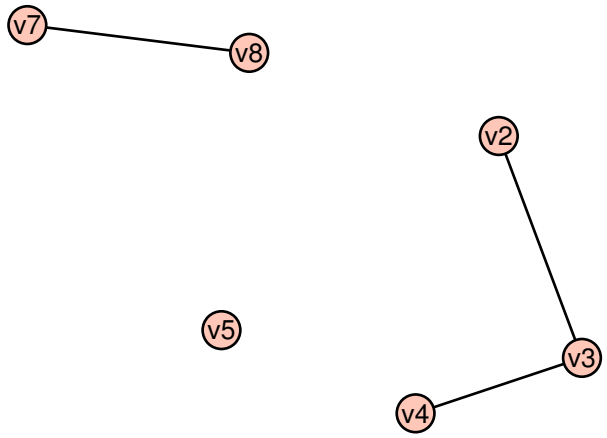
- $\tilde{V} = \{v1, v3, v7, v8\} \subseteq V\left(G_{ex_2}\right)$

- $\tilde{E} = \{a, i, j\} \subseteq E\,(G_{ex_2})$

but the pair $(\tilde{V}, \tilde{E})$ is *not* a graph (why?) therefore it cannot be a subgraph G_{ex_2}. The graphs SG_1



and SG_2



are both subgraphs of G_{ex_2}. Naturally G_{ex_2} is a subgraph of itself.

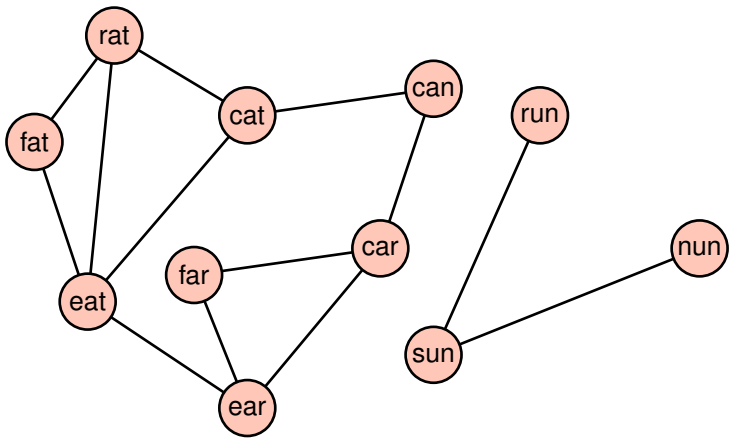
Proper subgraphs. By definition every graph is a subgraph of itself. This is a trivial subgraph. Any other subgraph is called *proper* subgraph.

Remark: according to the above definition a graph on a single vertex E_1 that is labelled 0 is *not* a subgraph of G_{ex_2} since

$$\{0\} \not\subset \{v1, v2, v3, v4, v5, v6, v7, v8\}.$$

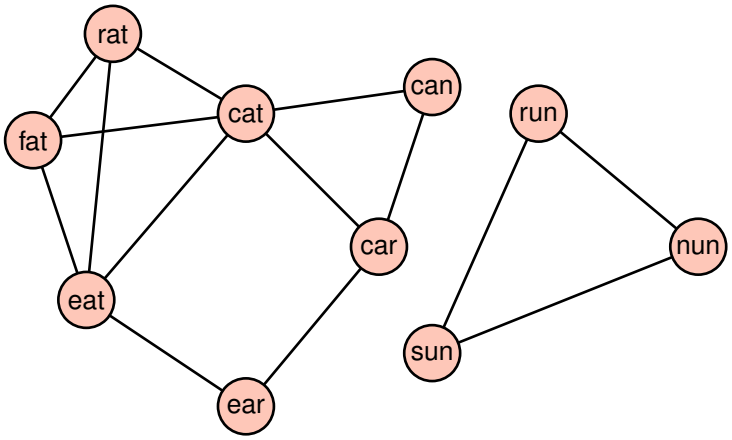
However, E_1 is isomorphic to the subgraph of G_{ex_2} defined with vertex set $\{v4\}$ and empty edge set. In that sense E_1 is a subgraph of G_{ex_2}. When labels are omitted a subgraph H of a graph G will refer to any graph to isomorphic to H.

Example: consider WG. The graphs SWG_1

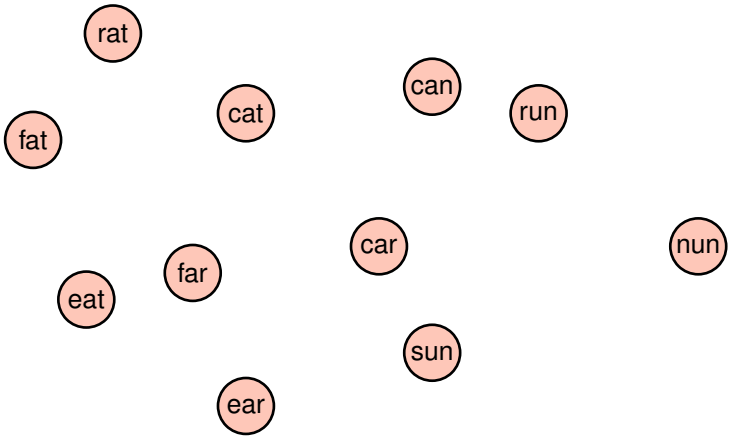


,

SWG_2



and SWG_3



are subgraphs of WG.

Definition 31 (spanning subgraph)**.** *A spanning subgraph of G is a subgraph G' of G such that $V(G') = V(G)$.*

Example: from the examples described above SG_1 is a spanning subgraph of G_{ex_2}, whereas SG_2 is *not* a spanning subgraph of G_{ex_2}. The graphs SWG_1 and SWG_3 are both spanning subgraphs of WG. The graph SWG_2 is *not* a spanning subgraph of WG.

Vertex/edge deletion. If $v \in V(G)$ then $G - v$ denotes the graph what has vertex $V(G) - v$ and edge set

$$E(G) - \{e \in E(G) \mid e \text{ is incident with } v\}.$$

For example SWG_2 is the graph $WG-$'far'.

Similarly for an e in G the graph $G - e$ is the graph with vertex set $V(G)$ and edge set $E(G) - e$. The idea can be generalized for a set of edges and set of vertices. For example the graph

$$SG_1 = G_{ex_2} - \{c, g, j\}\,.$$

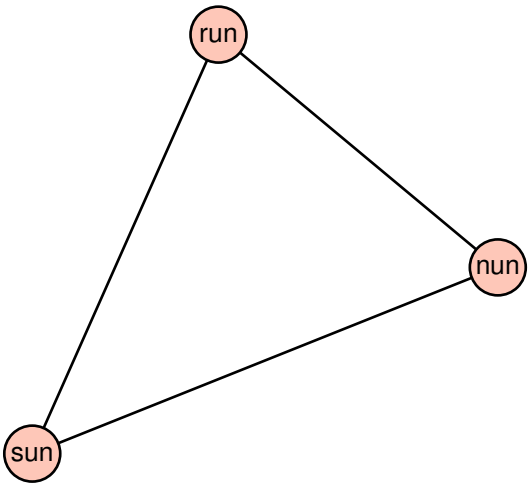
Natural extensions to deleting both edges and vertices are also possible.

Induced graphs. Denote the complement of a set S via $\overline{S}$. For a vertex set $T \subseteq V(G)$ define the graph *induced* by T as

$$G[T] = G - \overline{T}.$$

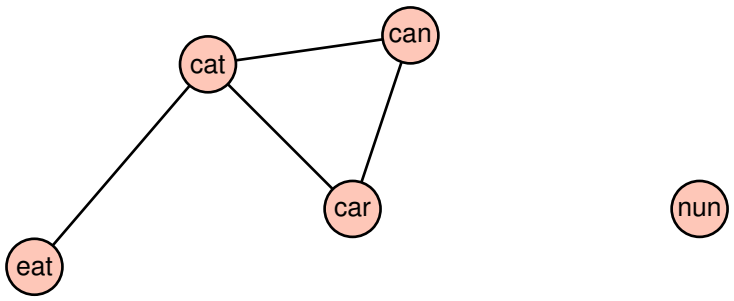
By construction induced graphs are subgraphs.

Example we have $WG[\{\text{run, sun, nun}\}]$ is



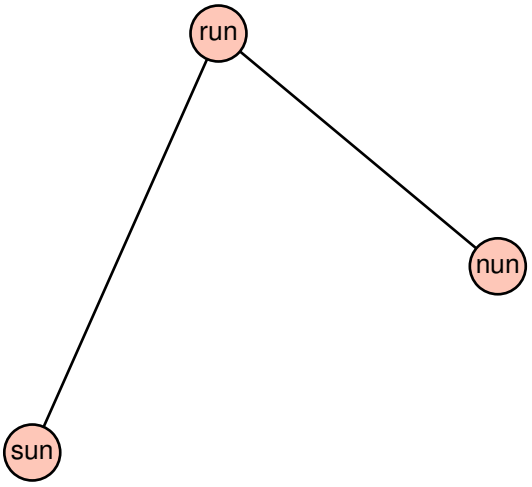
,

the graph induced by {eat, cat, can, car, nun} is



,

Remark a set T induces unique graph. For example the graph SWG_{4a}



,

is not equal (nor isomorphic) to the graph $WG[\{\text{run, sun, nun}\}]$.

Definition 32 (clique)**.** *A* clique *of a graph G is a subset T of the vertex set $V(G)$ such that $G[T]$ is isomorphic to the complete graph on $|T|$ vertices.*

Any pair of adjacent vertices is a clique of size two. For WG the set {run, sun, nun} induces a clique of size three. Another clique of size three is $WG[\{\text{fat, rat, eat}\}]$. The graph WG also has a clique of size four, namely $WG[\{\text{fat, rat, eat, cat}\}]$, but has no cliques of larger size.

Definition 33 (stable/independent set)**.** *An* independent set *of a graph G is a subset T of the vertex set $V(G)$ such that $G[T]$ is isomorphic to the empty graph on $|T|$ vertices.*

Any pair of vertices that are not adjacent is an independent set of size two. The graph WG has an independent sets

- {car, nun} of size two;
- {far, nun, rat} of size three;
- {car, fat, nun} of size three;
- {can, eat, far} of size three;
- {can, eat, far, sun} of size four;
- {can, eat, far, nun} of size four.

Maximum vs maximal. The term *maximal* means "cannot be enlarged". The term *maximum* means "of largest size". For example, the independent set {car, fat, nun} is maximal since no other independent set contains is properly, but it is not maximum since there is another independent set of larger size. The independent set {can, eat, far, nun} is maximum size independent set. For the independent set {can, eat, far} we have

$$\{\text{can, eat, far}\} \subsetneq \{\text{can, eat, far, nun}\}$$

therefore it is *not* maximal. It is not maximum as well. In terms of cliques

- {nun, sun, run} is maximal but not maximum;
- {rat, fat, eat} is neither maximal nor maximum;
- {rat, fat, cat} is neither maximal nor maximum;
- {rat, fat, cat, eat} is both maximal and maximum;
- {cat, can, car} is maximal but not maximum.

1.11 Connected graphs

Definition 34 (connected graph)**.** *A connected graph G is a graph such that there is a path between any two vertices in G.*

Of the graph examples in discussed in §1.1 only WG is *not* connected. The rest are connected graphs. K_n, C_n, P_n and E_1 are connected graphs. For $n > 1$ the graph E_n is not connected. Graphs that are not connected are often called *disconnected*.

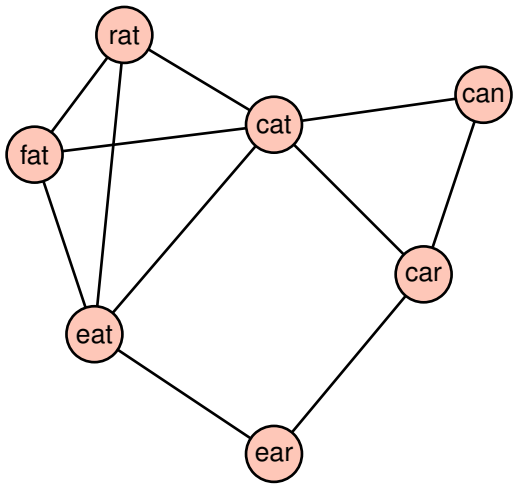
Theorem 7. *Every connected graph of size greater or equal to two has at least two vertices of the same degree.*

Proof. Self study problem (hint: pigeonhole principle). □

Definition 35 (maximal connected subgraph)**.** *A* maximal connected subgraph *of G is a subgraph H such that if G' is connected subgraph of G and H is subgraph of G' then $H = G'$.*

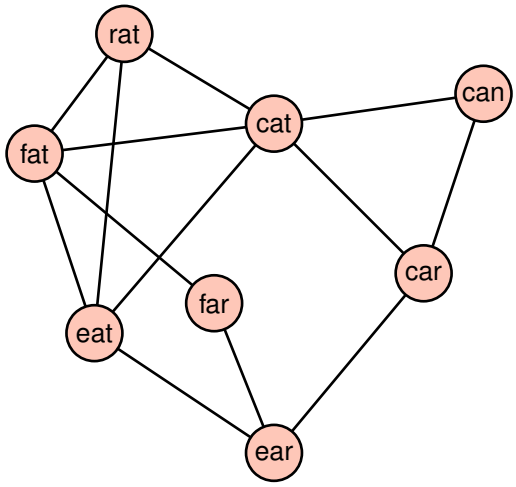
A connected graph will have a single maximal connected subgraph since the graph it self is not proper subgraph of any of its subgraphs.

The graph SWG_{4a} from §1.10 is a connected subgraph of WG since there is a path between any pair of vertices in SWG_{4a}. However, that graph is not *maximal* since there is a subgraph of WG, namely SWG_4 such that SWG_{4a} is a subgraph of SWG_4. Take any subgraph H of WG, such that SWG_4 is a subgraph of H as well. Then either H is disconnected or $SWG_4 = H$. Which means SWG_4 is a maximal connected subgraph of WG. Further, neither



,

nor

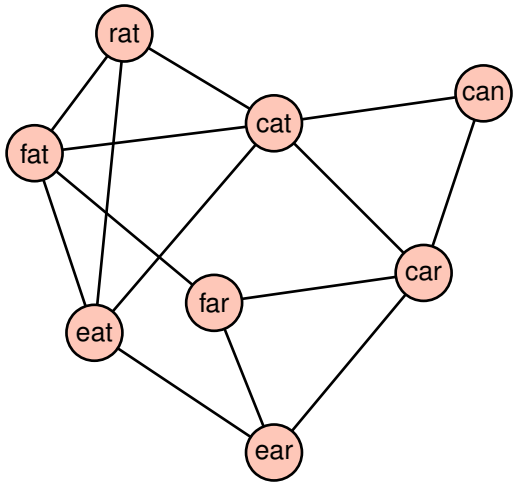


,

are maximal connected subgraphs of WG. The former is contained in the latter where as the latter is contained in the graph induced by

$$\{\text{fat, rat, eat, cat, far, ear, car, can}\}$$

namely,



,

In other words $WG\,[\{\text{fat, rat, eat, cat, far, ear, car, can}\}]$ is a maximal connected subgraph of WG.

Definition 36 (component)**.** *A maximal connected subgraph of a graph G is called a* component *of G.*

Examples: connected graphs have one component. The graph WG has two components namely,

- WG[{fat, rat, eat, cat, far, ear, car, can}] and
- WG[{nun, run, sun}]

An equivalent description of connected graphs is given via

Theorem 8. *A graph G is connected if and only if, there is vertex u in $V(G)$ such that there is a path from u to x for all vertices x in $V(G)$.*

Proof. Suppose G is connected and let u be any vertex in $V(G)$. Since G is connected there is a path from u to x for any vertex $x \in V(G)$.

Suppose now G has a vertex u such that there is a path from u to any other vertex in $V(G)$. Let x, y be any two vertices in $V(G)$. By assumption

1. there is a path from x to u, i.e., there is a path

$$p_x = xe_1v_1e_2v_2 \dots e_{n-1}v_{n-1}e_nu$$

2. there is a path from y to u, i.e., there is a path

$$p_y = ue'_1v'_1e'_2v'_2 \dots e'_{k-1}v'_{k-1}e'_ky$$

Combining p_x and p_y we obtain the walk

$$p_{xy} = xe_1v_1e_2v_2 \dots e_{n-1}v_{n-1}e_nue'_1v'_1e'_2v'_2 \dots e'_{k-1}v'_{k-1}e'_ky$$

There is a walk from x to y, by Theorem 4 there is a path from x to y. Since x and y were arbitrary vertices it follows that the graph is connected. □

Theorem 9. *For every connected graph G the vertices can be enumerated*

$$v_1, v_2, \dots, v_n$$

such that the graph G_i induced by $v_1, \dots, v_i$ i.e., $G_i = G[v_1, v_2, \dots, v_i]$ is connected for every i.

Proof. Pick a vertex at random and denote it as v_1. G_1 is connected and assume by induction that for i, $v_1, \dots, v_i$ are such that the graph G_i is connected. Let $v \in G - G_i$. Since G is connected there is a path from v_1 to v in G. Define v_{i+1} as the first vertex on the path from v_1 to v that is not in G_i. Then v_{i+1} has a neighbour in $v_1, \dots, v_i$ and the connectedness of G_i follows by induction. □

The above results are concerned with connected graphs. Let us discuss disconnected graphs a bit.

Definition 37 (induced cut)**.** *Let G be a graph and $X \subset V(G)$. The set of all edges incident with exactly one vertex in X is called the* cut induced by X.

Example: in WG we have

- the cut induced by $\{\text{far}\} \subseteq V(WG)$ is

$$\{\{\text{far, car}\}, \{\text{far, ear}\}, \{\text{far, fat}\}\} \subseteq E(WG)$$

- the cut induced by $\{\text{far, car}\} \subseteq V(WG)$ is

$$\{\{\text{far, ear}\}, \{\text{far, fat}\}, \{\text{car, ear}\}, \{\text{car, can}\}, \{\text{car, cat}\}\} \subseteq E(WG)$$

- the cut induced by $\{\text{far, can, nun }\} \subseteq V(WG)$ is

$$\{\{\text{far, ear}\}, \{\text{far, fat}\}, \{\text{can, car}\}, \{\text{can, cat}\}, \{\text{nun, sun}\}\{\text{nun, run}\}\} \subseteq E(WG)$$

- the cut induced by $\{\text{nun, run, sun }\} \subseteq V(WG)$ is empty.

Induced cuts let us prove that certain graphs are disconnected

Theorem 10. *A graph G is disconnected if and only if there is a proper non-empty subset X of the vertex set of G such that the cut induced by X is empty.*

Proof. Let G be is connected graph on at least two vertices and suppose by contradiction there is proper subset X of the vertex set of G such that the cut induced by X is empty. Let $u \in X$ and $w \in V(G) - X$. The vertices u and w exists since X is proper subset of the vertex set. Since G is connected there is uw path in G, say the path is $v_0, v_1, \ldots, v_{k-1}, v_k$ where $v_0 = u$ and $v_k = w$. Let i be the first index such that $v_i \in X$ and $v_{i+1} \in V(G) - X$. Such index exists since $v_0 = u \in X$ and $v_k = w \in V(G) - X$. Then the edge $v_i v_{i+1}$ has exactly one end in X thus the cut induced by X is non-empty, a contradiction.

Suppose now G is disconnected, which means it has at least two components H_1 and H_2. Let X denote the vertex set of H_1. Then X is a proper subset of the vertex set of G. If $e = (uv)$ is an element of the cut induced by X then either u or v but not both are element of X. Without loss of generality assume $u \in X$. Consider the subgraph $\tilde{H}$ of G defined as

- $V(\tilde{H}) = X \cup v$
- $E(\tilde{H}) = E(H_1) \cup (uv)$.

Then H_1 is a proper subgraph of $\tilde{H}$ and $\tilde{H}$ is connected contradicting maximality of H_1. Therefore the cut induced by X is empty completing the argument. □

1.11.1 Bridges and cut vertices

Definition 38. *A* cut edge *of a graph G is an edge $e \in E(G)$ such that the number of components of $G - e$ is strictly greater than the number of components of G.*

Remark: a cut edge is often called a *bridge*. Some software tools (and likely texts) define bridges as cut edges of *connected graphs*. Thus disconnected graphs may contain cut edges but no bridges. In these notes any cut edge is a bridge.

Example: The edges $(0, 3)$ and $(4, 7)$ are both bridges in G_{ex_3}. The graph in §1.3.2 has bridges $(0, 11)$, $(0, 6)$ and $(0, 1)$. Note that if you remove a bridge the number of components increases by one and the vertices incident with the bridge are in different components. This is not a coincidence.

Theorem 11. *If $e = (x, y)$ is a bridge of a connected graph G, then $G - e$ has precisely two components; furthermore x and y are in different components.*

Proof. Let $e = (u, v)$. Suppose e is a bridge then $G - e$ has at least two components. Let V_u be the set of vertices in $G - e$ such that there is a path from any vertex $x \in V_u$ to u. Let y be any vertex of $G - e$ such that $y \notin V_u$. There is at least one such vertex since $G - e$ has at least two components. Since there is a path from y to u in G and no path from y to u in $G - e$, then any path from y to u in G is of the form

$$ueve_1v_1 \dots v_{n-1}e_ny$$

Then there is a path $ve_1v_1 \dots v_{n-1}e_ny$ from any vertex in $G - e$ to v and therefore every vertex not in V_u is in the same component as v. □

Neither WG nor G_{ex_2} contain any bridges. The fact can be establish by the following theorem.

Theorem 12. *Edge e is a cut-edge of a graph G if and only if e is not in any cycle of G.*

Proof. First we show that if $e = (ab)$ is an edge in a cycle then e cannot be a bridge. Let $ae_1v_1e_2v_2 \dots v_{n-1}e_nbea$ be a cycle that contains e. Then $ae_1v_1e_2v_2 \dots v_{n-1}e_nb$ is a path from a to b in $G - e$. If e were a bridge then in $G - e$ by Theorem 11 a and b would be in different components a contradiction. Therefore if e is a bridge then e is not in any cycle of G.

If e is not in any cycle, then there is only one path from a to b which is the path aeb. Indeed if there were to be another path $ae_1v_1 \dots v_{n-1}e_nb$ that do not contain e then $ae_1v_1 \dots v_{n-1}e_nbea$ would be a cycle that contains e. Since e is not in the graph $G - e$ then a and b are in different components of $G - e$. And furthermore $G - e$ has more components than G. Thus e is a bridge. □

The above result implies the following

Theorem 13. *If there are two distinct paths from vertex v to vertex u in G then G has a cycle.*

Proof. Self study problem. □

Definition 39. *A vertex $u \in V(G)$ is a* cut vertex *if $G - u$ has more components than G.*

Example in G_{ex_2} the vertex 1 is a cut vertex. In G_{ex_3} cut vertices are 0 and 4. Note that vertices 3 and 7 are not cut vertices.

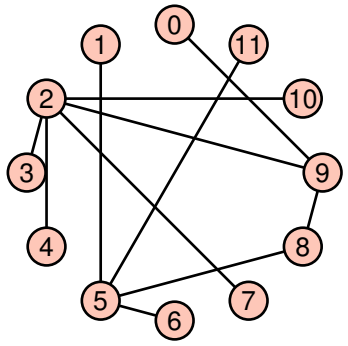
Example in §1.3.3 the vertex 0 is a cut vertex and it is not incident with any bridge.

Remark it is fairly trivial to come up with examples where removing u from G adds more that one component: e.g., consider vertex 0 in the graph in §1.3.2. Further a cut vertex may be part of a cycle as illustrated by vertex 1 for the graph G_{ex_3}.

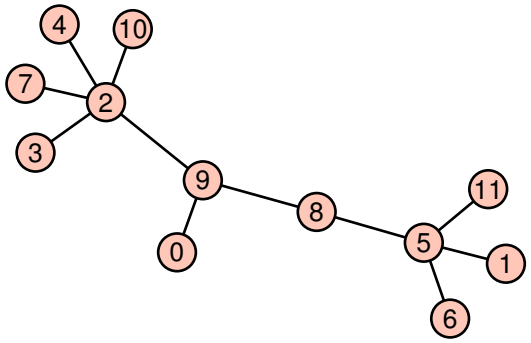
1.12 Trees

Definition 40 (tree)**.** *A graph without cycles or an* acyclic *graph is called a* forest*. An acyclic connected graph is called a* tree*.*

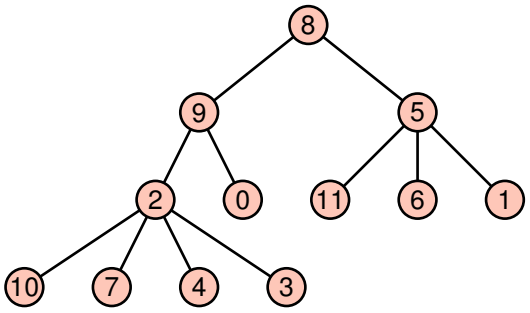
Examples: E_n is a forest, P_n is a tree for any n. The following graph is also a tree



However, trees and forests can be drawn with no edge crossings, one such embedding for the tree above is



Last but not least trees are drawn on levels such as



Trees have wide ranging use in graph theory and posses a number of important properties. The following theorem established multiple equivalent ways to define a tree.

Theorem 14. *The following are equivalent for a graph T:*

1. *T is a tree;*
2. *any two vertices in T are connected by a unique path;*
3. *T is minimally connected, that is every edge is a cut edge;*
4. *T is maximally acyclic, that is $T + xy$ contains a cycle for any two non-adjacent vertices x and y in T;*

$(2 \Rightarrow 1)$. Since there is a path between any two vertices the graph is connected. If there is a cycle then there are two vertices that are connected with at least two distinct paths contradicting the assumption and hence the graph has no cycles. Thus it must be a tree. □

$(3 \Rightarrow 1)$. By assumption the graph is connected. Since every edge is a cut edge, by Theorem 12 no edge is in a cycle and thus there are no cycles hence the graph is a tree. □

$(4 \Rightarrow 1)$. Let u, v be two vertices. Suppose they are not adjacent. Then by assumption $T + (uv)$ contains a cycle and since T does not contain a cycle then the edge (uv) is in the cycle. Thus there are two paths from $T + (uv)$ from u to v. One of those paths does not contain (uv) and it must therefore lie all within T. Hence T is connected and acyclic thus a tree. □

$(1 \Rightarrow 2)$. Suppose vertices u, v are connected by two distinct paths

$$P_1 = ue_1w_1e_2w_2 \dots e_nv$$

and

$$P_2 = ue'_1w'_1e'_2w'_2 \dots e'_nv.$$

Let w_k be such that for all $j \leq k$ we have that $e_jw_j = e'_jw'_j$. Let w_l be the first vertex on both paths such that $l > k$. Then there is a cycle in the graph from w_k to w_l along the two paths a contradiction. □

$(1 \Rightarrow 3)$. By Theorem 12 an edge is a bridge if it is in no cycles. Since a tree contains no cycles, none of the edges in a tree can be part of a cycle and therefore each edge is a bridge. □

$(1 \Rightarrow 4)$. Let u, v be any two non-adjacent vertices in T. Since T is connected there is a path from u to v in T. Along with the edge (uv) this forms a cycle. □

Motivation: at least how many edges you need to connect dots on the plane. If you have more do you necessarily have a cycle? The answer to this question establishes a relation between the size and the order of a tree.

Theorem 15. *For any tree with q edges and p vertices we have that $q = p - 1$.*

Proof. By induction on the number of vertices.
Base case: $p = 1$: since there are no edge in a tree with one vertex $p = 1$ and $q = 0 = p - 1 = 1 - 1$ and the result holds.
Assumption: assume the statement holds for any tree with at most p vertices.
Inductive step: Let T be a tree with $p + 1$ vertices. Take any edge e of T and consider $T - e$. Since e is a bridge and T is connected then $T - e$ has two components, say T_1 and T_2. Suppose that T_1 has p_1 vertices and T_2 has p_2 vertices. Both T_1 and T_2 contain no cycles, are connected and therefore they are trees. We apply the inductive hypothesis to these trees separately and obtain $q_1 = p_1 - 1$ and $q_2 = p_2 - 1$, where q_1 and q_2 are the number of vertices in T_1 and T_2, respectively. By construction the number of vertices in T is $p_1 + p_2$, and number of edges is $q = q_1 + q_2 + 1$ where the additional 1 is for the edge e. Substituting with the values for q_1 and q_2 we obtain $q = q_1 + q_2 + 1 = p_1 - 1 + p_2 - 1 + 1 = p - 1$, which concludes the argument. □

The next result concerns the number of degree one vertices in a tree. Such vertices are called *leaves*. For the tree defined in the beginning of this section the leaves are $0, 1, 3, 4, 6, 7, 10, 11$. There are three alternative proofs for the theorem since each one illustrates a different technique that can be used with graphs.

Theorem 16. *A tree with at least two vertices has at least two leaves.*

Direct proof. Let $p = v_0e_1v_1 \dots v_{n-1}e_nv_n$ be a longest path in T. (A longest path is a path for which no other path in the tree has a longer length, for the result maximal path also suffices). This path has length at least one since it contains at least two vertices and hence at least one edge. Consider vertex v_0. If the degree of v_0 is greater than 1 then there must be another edge incident with v_0 other than e_1. Say this is the edge e'. Let e' be (v_0, w). There are two possible scenarios: either $w = v_i$ for some i or w is not a vertex that appears in the path p. In the first case $v_0e_1 \dots v_{i-1}e_iwe'v$ is a cycle which contradicts the fact that the graph is a tree. In the second scenario $we'v_0e_1v_1 \dots v_{n-1}e_nv_n$ is a path in the graph that has length strictly greater than the length of p contradicting the fact that p is a longest path. In either scenario we have a contradiction and therefore the degree of v_0 is one. Similar argument applies to v_n and hence in the three at least v_0 and v_n have degree one completing the argument. □

Proof by counting. Let n_i be the number of vertices of degree $i \geq 0$. Then for p then number of vertices in the tree we have

$$p = n_1 + n_2 + \cdots$$

By the Handshake lemma 1 we have

$$2q = \sum d(v) = n_1 + 2n_2 + 3n_3 + \cdots ,$$

where q is the number of edges in the tree. By Theorem 15 we have that

$$q = -1 + p = -1 + n_1 + n_2 + n_3 + \dots .$$

Substituting p in the first equation and rearranging its terms we get

$$n_1 = 2 + n_3 + 2n_4 + 3n_5 + \cdots \geq 2$$

□

Proof by induction. By induction on the number of vertices.
Base case: $p = 2$: a tree with two vertices is isomorphic to K_2 and it contains two vertices of degree one.
Assumption: assume the statement holds for any tree with at most $p \geq 2$ vertices.
Inductive step: Let T be a tree with $p+1 \geq 3$ vertices. Take any edge e of T and consider $T-e$. Since e is a bridge and T is connected then $T-e$ has two components, say T_1 and T_2. Suppose that T_1 has p_1 vertices and T_2 has p_2 vertices. Both T_1 and T_2 contain no cycles, are connected and therefore they are trees. Further since $p_1 + p_2 = p + 1 \geq 3$ either $p_1 \geq 2$ or $p_2 \geq 2$, or both. Without loss of generality assume $p_1 \geq 2$. Then by induction T_1 contains at least two vertices of degree one say u_1 and u_2. Suppose e is incident with vertices x and y, without loss of generality $x \in T_1$ and $y \in T_2$. Thus x is incident with at most one of u_1 or u_2. Then in T at least one of u_1 or u_2 is a vertex of degree one. For the other vertex of degree one consider the cases:

1. if $p_2 \geq 2$ then by induction T_2 contains at least two vertices of degree one say v_1 and v_2. Since e is incident with at most one of them then at least one of them is of degree one in T.

2. if $p_2 = 1$ edge e is incident with the (single) vertex of T_2 and therefore in T that vertex has degree one.

In either case T contains at least two vertices of degree one completing the argument. □

The result generalizes to forests. To conclude this section we prove.

Theorem 17. *Trees are bipartite graphs.*

Proof. Let T be a tree and r be a vertex of T. Define the set $A \subseteq V(T)$ be the set of all vertices in T such that the path from r to $u \in A$ is of even length. This is a well defined function as there is a unique path from r to u by Theorem 14. Let B be the set of all vertices in T such that the length of the path from r to $u \in B$ is of odd length. Then A and B partition $V(T)$. Let $e = (xy)$ be an edge incident with vertices x and y. Denote the unique path from r to x with $P_{rx} = re_1v_1e_2v_2 \dots e_{k-1}v_{k-1}e_kx$. Then either $P_{rx}ey$ is the path from r to y or $v_{k-1} = y$ and $e_k = e$; else there is more than one path to either x or y in T a contradiction. Since no consecutive integers have the same parity the vertices x and y cannot be both in A or both in B. Thus e is incident with one vertex in A and one vertex in B, showing that T is a bipartite graph. □

1.13 Spanning trees

When considering a graph, it is in general useful to consider a component thus reducing the problem to connected graphs. From the set of subgraphs of a connected graph a subgraph that preserves connectivity with the fewest possible edges is an important starting point.

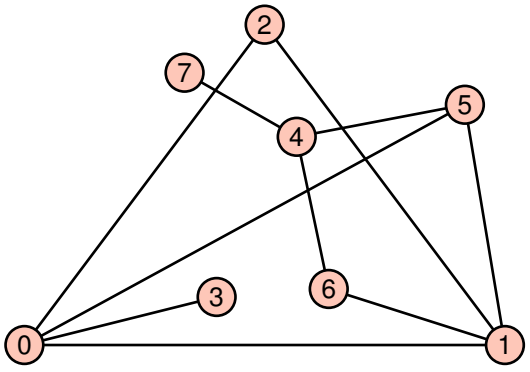
Definition 41. *A spanning subgraph that is also a tree is called a* spanning *tree.*

To establish formally that every connected graph has a spanning tree the following results suffice.

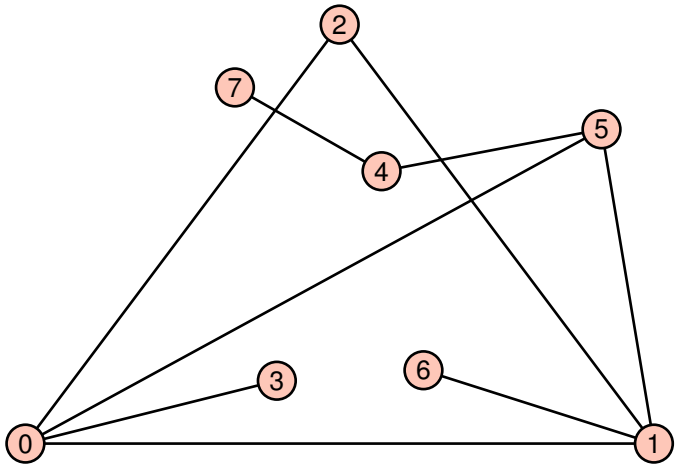
Theorem 18. *Every connected graph has a spanning tree.*

Proof. Let G be a connected graph. Delete edges from the cycles of G one by one until the graph is acyclic. Since the edges that are deleted are in a cycle the subgraph is also connected. Thus it is a tree with the same vertex set meaning it is a spanning tree. □

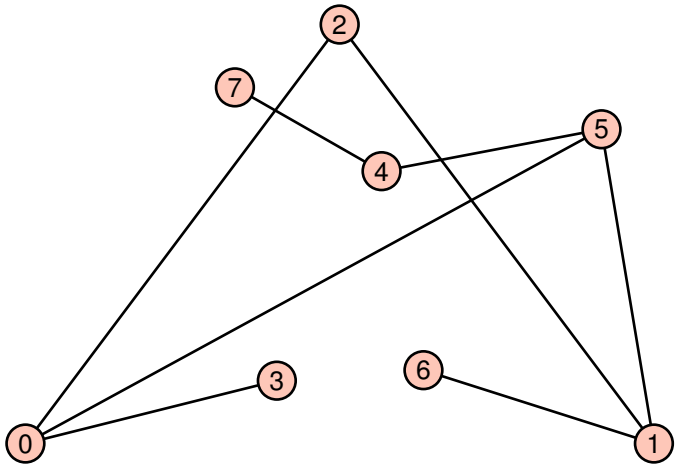
Before moving onto the second result here is an example that illustrates the above argument. Recall G_{ex_3}



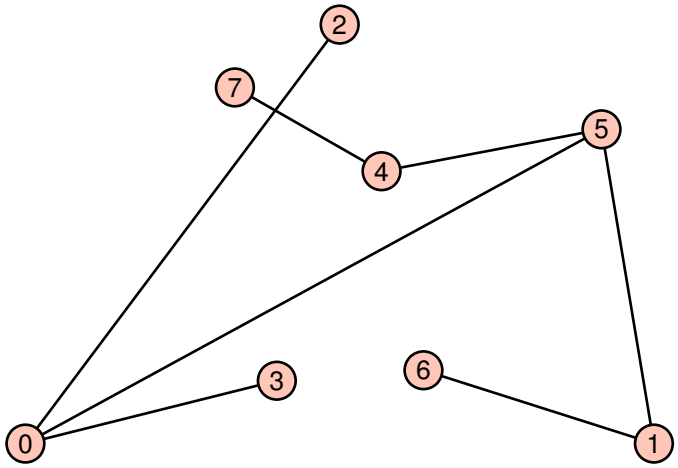
Edge $\{4, 7\}$ is a cut edge so it cannot be removed. Edge $\{4, 6\}$ is in a cycle for example the cycle $4, 6, 1, 0, 5$ so look at graph $S_1 = G_{ex_3} - \{4, 6\}$



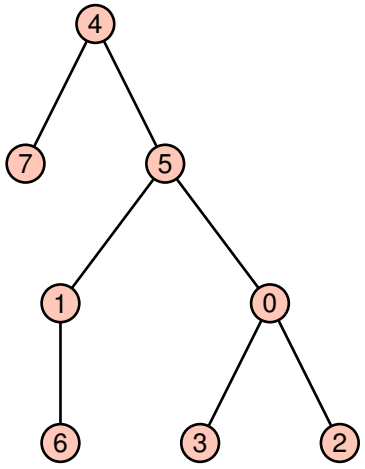
In the new graph edge $\{6, 1\}$ is a cut edge so it cannot be removed. But $\{0, 1\}$ is not a cut edge so consider $S_1 - \{0, 1\} = G_{ex_3} - \{\{0, 1\}, \{4, 6\}\}$



In the last graph $\{1, 2\}$ is not a cut edge since it is in the cycle $1, 2, 0, 5$ so it can be removed to obtain $G_{ex_3} - \{\{0, 1\}, \{1, 2\}, \{4, 6\}\}$



The resulting graph contains no cycles, it is connected and therefore a tree. Further it is a spanning graph and therefore a spanning tree. An alternative embedding which illustrate the fact it is a tree is



Theorem 19. *If a graph has a spanning tree then the graph is connected.*

Proof. If T is a spanning tree of G then every path in T is also a path in G. Since T is a tree it is connected and thus between any pair of vertices there a T-path (a path that has only edge in T). Since every edge of T is also an edge in G any T path is also a G path and thus any pair of vertices in G is connected. Thus G is connected. □

The above two theorems imply that a graph is connected if and only if it as a spanning tree. Furthermore.

Theorem 20. *If G is connected with p vertices and $p-1$ edges then G is a tree.*

Proof. By Theorem 19 the graph G has a spanning tree T. Since T is spanning tree it has p vertices and by Theorem 15 it has $p-1$ edges. Since G has $p-1$ edges every edge of G is in T and thus the edge set and vertex set of T and G coincide, thus they are the same graph. Hence G is a tree. □

A corollary to the results established so far is the following classification of bipartite graphs.

Theorem 21. *A graph is bipartite if and only if it has only even cycles.*

Proof. Self study problem. □

1.14 BFS

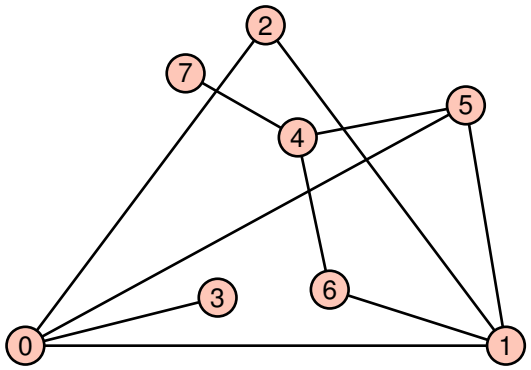
The steps below describe the so called Breadth First Search tree algorithm for finding a spanning tree of a graph if one exists. It is a first in first out (FIFO) type algorithm and has a natural last in first out (LIFO) variant known as Depth First Search (DFS) tree algorithm.

1. start with an empty (FIFO) list l and a (random) vertex u_1;
2. set $pr(u_1) = \emptyset$;
3. start with a graph T where $V(T) = u_1$ and $E(T) = \emptyset$
4. label vertex u_1 active
5. for every edge $e = (v, u)$ incident with the active vertex v do
   (a) if $u \in V(T)$ skip e and u
   (b) else if $u \notin V(T)$ add u to the list l;
   (c) $V(T) = V(T) \cup u$
   (d) $E(T) = E(T) \cup (v, u)$
   (e) set $pr(u) = v$
6. remove the vertex v from the list and
   (a) if l is not empty label the first vertex active and go to Step 5
   (b) else terminate the algorithm by outputting T

Remark. In practice rather than selecting a random initial vertex u_1 one chooses the starting vertex depending on the problem at hand. That initial vertex is known as the *root* of the BFS tree. The BFS tree is output of the algorithm.

1.14.1 Example: connected graph

Recall G_{ex_3} from §1.1.4

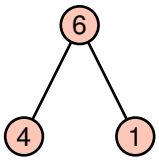


We will apply the algorithm with root vertex 6. As describe the algorithm itself at is ambiguous about the order of which edges are processed at Step 5. Since graphs in many ways are stored as matrices adopt the convention that edges are processed by ordering the labels on incident with edges. For example when vertex i is active it is natural to sweep the ith row of the adjacency matrix from left to right that before checking if vertex j is incident with vertex i all vertices with index smaller than j should already have been processed. Let us denote the BFS tree of graph G with root r using such convention with BFS(G_{Ex_3},6). That is

BFS(G_{Ex_3},6) random vertex = 6 set initial T

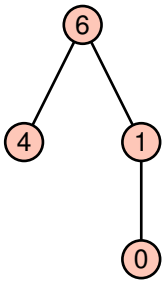
FIFO= [6]
Active vertex = 6
process edge (1,6):
add vertex 1 and edge (1,6) to T

add vertex 1 to FIFO
FIFO=[6, 1]
set $pr(1) = 6$
process edge (4,6):
add vertex 4 and edge (4,6) to T



add vertex 4 to FIFO
FIFO=[6, 1, 4]
set $pr(4) = 6$
remove 6 from FIFO

FIFO= [1, 4]
Active vertex = 1
process edge (0,1):
add vertex 0 and edge (0,1) to T



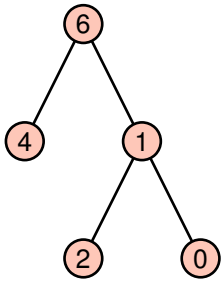
add vertex 0 to FIFO

FIFO=[1, 4, 0]

set $pr(0) = 1$

process edge (1,2):

add vertex 2 and edge (1,2) to T



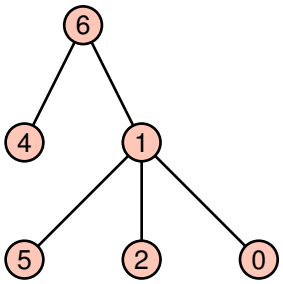
add vertex 2 to FIFO

FIFO=[1, 4, 0, 2]

set $pr(2) = 1$

process edge (1,5):

add vertex 5 and edge (1,5) to T



add vertex 5 to FIFO

FIFO=[1, 4, 0, 2, 5]

set $pr(5) = 1$

process edge (1,6): 6 in T - skip

remove 1 from FIFO

FIFO= [4, 0, 2, 5]

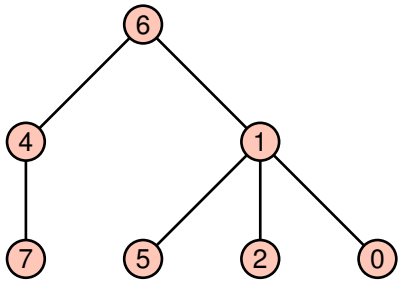
Active vertex = 4

process edge (4,5): 5 in T - skip

process edge (4,6): 6 in T - skip

process edge (4,7):

add vertex 7 and edge (4,7) to T



add vertex 7 to FIFO

FIFO=[4, 0, 2, 5, 7]

set $pr(7) = 4$

remove 4 from FIFO

FIFO= [0, 2, 5, 7]

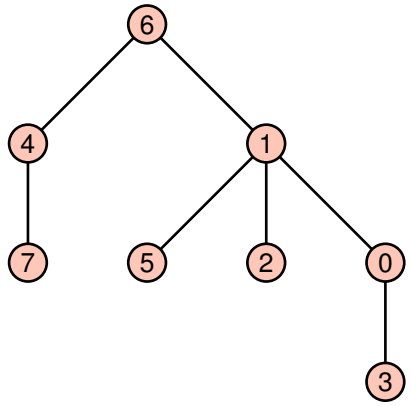
Active vertex = 0

process edge (0,1): 1 in T - skip

process edge (0,2): 2 in T - skip

process edge (0,3):

add vertex 3 and edge (0,3) to T



add vertex 3 to FIFO
FIFO=[0, 2, 5, 7, 3]
set $pr(3) = 0$
process edge (0,5): 5 in T - skip
remove 0 from FIFO

FIFO= [2, 5, 7, 3]
Active vertex = 2
process edge (0,2): 0 in T - skip
process edge (1,2): 1 in T - skip
remove 2 from FIFO

FIFO= [5, 7, 3]
Active vertex = 5
process edge (0,5): 0 in T - skip
process edge (1,5): 1 in T - skip
process edge (4,5): 4 in T - skip
remove 5 from FIFO

FIFO= [7, 3]
Active vertex = 7
process edge (4,7): 4 in T - skip
remove 7 from FIFO

FIFO= [3]
Active vertex = 3

process edge (0,3): 0 in T - skip
remove 3 from FIFO

FIFO= []
terminate and output T

Remark. Consider vertex 1 it has neighbourhood $\{0, 2, 5, 6\}$. When 1 was active according to the convention adopted for Step 5 first edge $(1, 0)$ was processed then $(1, 2)$ then $(1, 5)$ and finally $(1, 6)$. The order in which the vertices entered the tree is $6, 1, 4, 0, 2, 5, 7, 3$.

Non-tree edges. The set of edges $E(G_{Ex_3}) - E(T)$ that is the set of all edges of G_{Ex_3} that are *not* in the tree T are called non-tree edges. For this particular example the non-tree edges are

$$(0, 2), (0, 5), (4, 5)$$

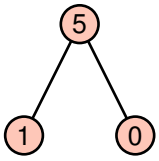
Example for BFS(G_{Ex_3},5) we have

BFS(G_{Ex_3},5) random vertex = 5 set initial T

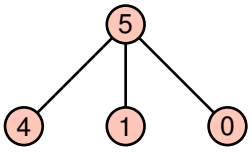
FIFO= [5]
Active vertex = 5
process edge (0,5):
add vertex 0 and edge (0,5) to T

add vertex 0 to FIFO
FIFO=[5, 0]
set $pr(0) = 5$

process edge (1,5):
add vertex 1 and edge (1,5) to T

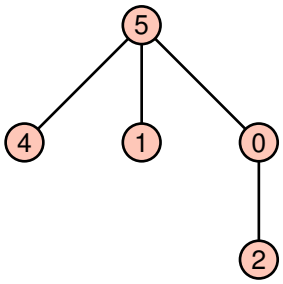


add vertex 1 to FIFO
FIFO=[5, 0, 1]
set $pr(1) = 5$
process edge (4,5):
add vertex 4 and edge (4,5) to T



add vertex 4 to FIFO
FIFO=[5, 0, 1, 4]
set $pr(4) = 5$
remove 5 from FIFO

FIFO= [0, 1, 4]
Active vertex = 0
process edge (0,1): 1 in T - skip
process edge (0,2):
add vertex 2 and edge (0,2) to T



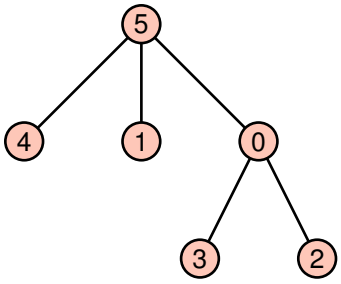
add vertex 2 to FIFO

FIFO=[0, 1, 4, 2]

set $pr(2) = 0$

process edge (0,3):

add vertex 3 and edge (0,3) to T



add vertex 3 to FIFO

FIFO=[0, 1, 4, 2, 3]

set $pr(3) = 0$

process edge (0,5): 5 in T - skip

remove 0 from FIFO

FIFO= [1, 4, 2, 3]

Active vertex = 1

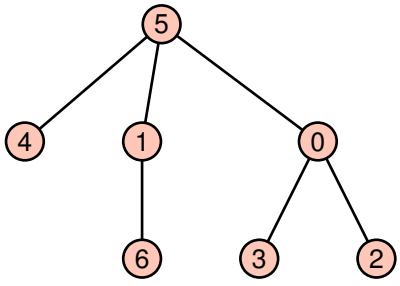
process edge (0,1): 0 in T - skip

process edge (1,2): 2 in T - skip

process edge (1,5): 5 in T - skip

process edge (1,6):

add vertex 6 and edge (1,6) to T



add vertex 6 to FIFO

FIFO=[1, 4, 2, 3, 6]

set $pr(6) = 1$

remove 1 from FIFO

FIFO= [4, 2, 3, 6]

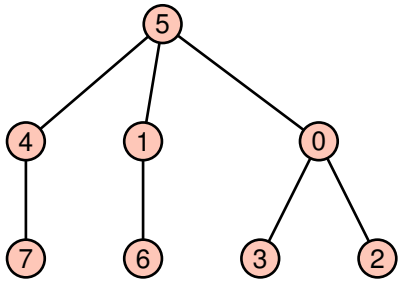
Active vertex = 4

process edge (4,5): 5 in T - skip

process edge (4,6): 6 in T - skip

process edge (4,7):

add vertex 7 and edge (4,7) to T



add vertex 7 to FIFO

FIFO=[4, 2, 3, 6, 7]

set $pr(7) = 4$

remove 4 from FIFO

FIFO= [2, 3, 6, 7]

Active vertex = 2

process edge (0,2): 0 in T - skip

process edge (1,2): 1 in T - skip

remove 2 from FIFO

FIFO= [3, 6, 7]

Active vertex = 3

process edge (0,3): 0 in T - skip

remove 3 from FIFO

FIFO= [6, 7]

Active vertex = 6

process edge (1,6): 1 in T - skip

process edge (4,6): 4 in T - skip
remove 6 from FIFO

FIFO= [7]
Active vertex = 7
process edge (4,7): 4 in T - skip
remove 7 from FIFO

FIFO= []
terminate and output T
In BFS(G_{Ex_3},5) vertices enter the tree in order

$$5, 0, 1, 4, 2, 3, 6, 7$$

the non-tree edges are

$$(0, 1), (1, 2), (4, 6)$$

Exercise check that BFS(G_{Ex_3},7) vertices enter the tree in order

$$7, 4, 5, 6, 0, 1, 2, 3$$

and the non-tree edges are

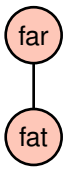
$$(0, 1), (1, 2), (1, 6)$$

1.14.2 Example: disconnected graph

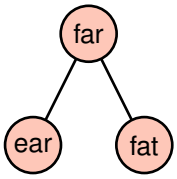
Recall the disconnected graph WG from §1.1.5.

Example random vertex = far set initial T

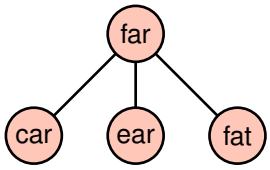
FIFO= ['far']
Active vertex = far
process edge (far,fat):
add vertex fat and edge (far,fat) to T



add vertex fat to FIFO
FIFO=['far', 'fat']
set $pr(fat) = far$
process edge (ear,far):
add vertex ear and edge (ear,far) to T

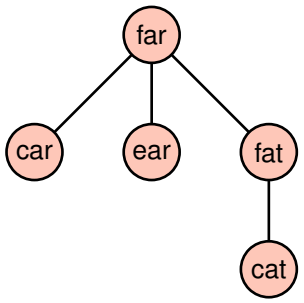


add vertex ear to FIFO
FIFO=['far', 'fat', 'ear']
set $pr(ear) = far$
process edge (car,far):
add vertex car and edge (car,far) to T



add vertex car to FIFO
FIFO=['far', 'fat', 'ear', 'car']
set $pr(car) = far$
remove far from FIFO

FIFO= ['fat', 'ear', 'car']
Active vertex = fat
process edge (cat,fat):
add vertex cat and edge (cat,fat) to T



add vertex cat to FIFO

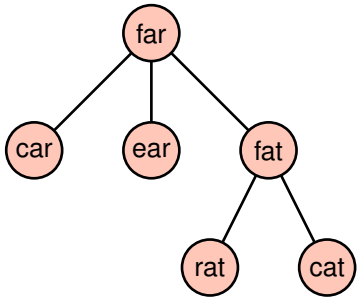
FIFO=['fat', 'ear', 'car', 'cat']

set $pr(cat) = fat$

process edge (far,fat): far in T - skip

process edge (fat,rat):

add vertex rat and edge (fat,rat) to T



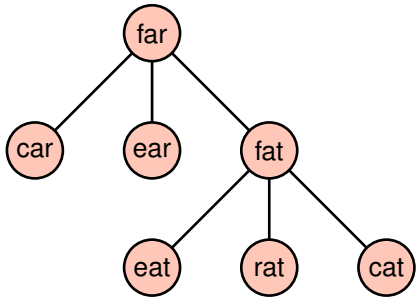
add vertex rat to FIFO

FIFO=['fat', 'ear', 'car', 'cat', 'rat']

set $pr(rat) = fat$

process edge (eat,fat):

add vertex eat and edge (eat,fat) to T



add vertex eat to FIFO

FIFO=['fat', 'ear', 'car', 'cat', 'rat', 'eat']

set $pr(eat) = fat$

remove fat from FIFO

FIFO= ['ear', 'car', 'cat', 'rat', 'eat']

Active vertex = ear

process edge (ear,far): far in T - skip

process edge (ear,eat): eat in T - skip

process edge (car,ear): car in T - skip

remove ear from FIFO

FIFO= ['car', 'cat', 'rat', 'eat']

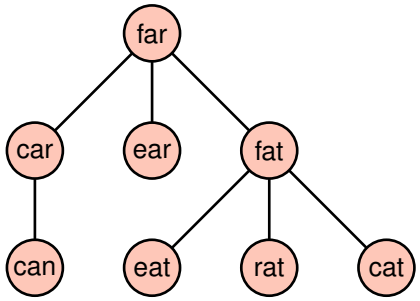
Active vertex = car

process edge (car,cat): cat in T - skip

process edge (car,far): far in T - skip

process edge (can,car):

add vertex can and edge (can,car) to T



add vertex can to FIFO

FIFO=['car', 'cat', 'rat', 'eat', 'can']

set $pr(can) = car$

process edge (car,ear): ear in T - skip

remove car from FIFO

FIFO= ['cat', 'rat', 'eat', 'can']

Active vertex = cat

process edge (cat,rat): rat in T - skip

process edge (cat,eat): eat in T - skip

process edge (can,cat): can in T - skip

process edge (cat,fat): fat in T - skip
process edge (car,cat): car in T - skip
remove cat from FIFO

FIFO= ['rat', 'eat', 'can']
Active vertex = rat
process edge (cat,rat): cat in T - skip
process edge (eat,rat): eat in T - skip
process edge (fat,rat): fat in T - skip
remove rat from FIFO

FIFO= ['eat', 'can']
Active vertex = eat
process edge (cat,eat): cat in T - skip
process edge (eat,rat): rat in T - skip
process edge (eat,fat): fat in T - skip
process edge (ear,eat): ear in T - skip
remove eat from FIFO

FIFO= ['can']
Active vertex = can
process edge (can,cat): cat in T - skip
process edge (can,car): car in T - skip
remove can from FIFO

FIFO= []
terminate and output T

Remark. Observe that the output tree does not contain all vertices that are in WG. In particular vertices

$$\text{nun}, \text{sun}, \text{run}$$

are not in the output tree T.

Remark. Check that any BFS tree rooted at "sun" does not contain vertices

$$\text{far}, \text{fat}, \text{ear}, \text{eat}, \text{cat}, \text{car}, \text{can}, \text{rat}$$

1.14.3 Properties of BFS trees

An important algorithmic issue is running time and termination of an algorithm. For the BFS algorithm described here observe that once a vertex enters the tree T that vertex is never added to the FIFO list. Therefore every vertex enters the FIFO list at most once. We have a finite number of vertices so to the FIFO only a finite number of vertices is added throughout the algorithm. At Step 6 the FIFO list looses a vertex. Thus at some point the FIFO list becomes empty and the algorithm terminates. The focus from now on is on the properties of the output.

Theorem 22. *The BFS algorithm terminates by outputting a tree T.*

Proof by induction. Base case. The graph T contains a single vertex thus it is a tree.
Assumption. For $k \geq 0$ the graph T is a tree with $k+1$ vertices and k edges.
Inductive step. If the algorithm terminates as step $k+1$ we are done. Otherwise, the algorithm adds the same number of edges and vertices, preserves connectivity (there is a path from any vertex to the active vertex). A connected graph of size one less than the order is a tree and the result follows. □

Corollary 1. *If T has less vertices than G then G is disconnected, otherwise T is a spanning tree.*

So far nothing was said about Step 2 and Step 5(e). Let

- $pr^0(u) = u$ and
- for $k \geq 1$ define $pr^k(u) = pr\left(pr^{k-1}(u)\right)$

for any vertex. We can now talk about the *parent - child* relation between vertices established via the relation pr. The root r vertex has no parent. Every vertex for which $pr(u) = r$ has parent r an every such vertex is a child of r. Likewise every vertex for which $pr(u) = v$ has parent v an every such vertex is a child of v. For vertices u and v if there is an integer k such that $pr^k(v) = u$ then we say that u is an *ancestor* of v. To that end the root is an ancestor of all other vertices in a BFS tree.

Example: for BFS(G_{Ex_3},6) obtained in §1.14.1 we have parent of vertex 0 is vertex 1, the vertices 1 and 6 are ancestors of vertex 0. Vertex 0 has one child vertex 3. Vertex 1 has three children, vertex 0, vertex 2 and vertex 5. Its parent is vertex 6. Vertex 6 is the only ancestor of vertex 1. Vertex 6 has

children vertex 1 and vertex 4. The parent of vertex 2 is vertex 1 and its ancestors are vertex 1 and vertex 6.

Example: The parent child relation *depends* on the BFS tree! For BFS(G_{Ex_3},5) obtained in §1.14.1 we have parent of vertex 2 is vertex 0, the vertices 0 and 5 are ancestors of vertex 2. The parent of 3 is also vertex 0 and its ancestors are also 0 and 5. The parent of vertex 1 is 5. Vertex 1 had child vertex 6. The (only) ancestor of vertex 1 is vertex 5. Vertex 7 has no children, it has parent vertex 4 and ancestors 4 and 5.

Definition 42. *A level of a vertex u in a BFS tree with root r is the integer k such that $pr^k(u) = r$.*

Example: for BFS(G_{Ex_3},6) obtained in §1.14.1. Vertex 6 has level zero; vertex 1 and vertex 4 have level one; vertex 0, vertex 2, vertex 5 and vertex 7 have level two; and lastly vertex 3 has level three. The vertices enter the tree in order putting the levels of each vertex below it we have

u	6	1	4	0	2	5	7	3
level(u)	0	1	1	2	2	2	2	3

For BFS(G_{Ex_3},5) similar table is

u	5	0	1	4	2	3	6	7
level(u)	0	1	1	1	2	2	2	2

The fact that the levels given the order in which vertices enter a tree is weakly increasing is not a coincidence.

Theorem 23. *Vertices enter a BFS tree in non-decreasing order of level*

Proof. By induction. The first vertex is at level zero. Assume for the first m vertices the result holds. Consider the next vertex v that joins the tree at stage $m+1$. Then $pr(v) = u$, where u is the active vertex. level(v) = level$(u)+1$. Consider any other non-root vertex x in the tree at stage $m+1$. Let $pr(x) = y$, hence level(x) = level$(y)+1$. There are two cases $y = u$ or y was active before u by induction hypothesis

$$\text{level}(y) \leq \text{level}(u).$$

Then

$$\text{level}(x) = \text{level}(y) + 1 \leq \text{level}(u) + 1 = \text{level}(v)$$

hence if x was added before v then level(x) is no larger than level(v). □

Any edge of a BFS tree is incident with vertices that are *exactly* one level apart. Consider the non-tree edges in a BFS tree. In BFS(G_{Ex_3},6) obtained in §1.14.1 for each non-tree edge

- $(0, 2)$ joins vertices at the same level;
- $(0, 5)$ joins vertices at the same level;
- $(4, 5)$ joins vertices one level apart.

In BFS(G_{Ex_3},5) for each non-tree edge we have

- $(0, 1)$ joins vertices at the same level;
- $(1, 2)$ joins vertices one level apart.
- $(4, 6)$ joins vertices one level apart.

This is an important property of BFS trees.

Theorem 24. *In a connected graph G with a breadth-first search tree T each edge $e \in E(G)$ connects vertices that are at most one level apart.*

Proof. let $e = (u, v) \in E(G)$ and without loss of generality let u join the tree before v, then by Theorem 23 we have

$$\text{level}(u) \leq \text{level}(v).$$

Case 1: v is not in the tree when u is active. Then v and e are added to the growing tree and

$$\text{level}(v) = \text{level}(u) + 1.$$

In this case the edge e is in the tree and shows that tree edges join vertices exactly one level apart.

Case 2: v is in the tree when u is active. Let $pr(v) = w$ which means

$$\text{level}(v) = \text{level}(w) + 1.$$

Since v is in the tree when u is active w was added to the tree before u and by Theorem 23

$$\text{level}(w) \leq \text{level}(u).$$

Thus

$$\text{level}(u) \leq \text{level}(v) = \text{level}(w) + 1 \leq \text{level}(u) + 1$$

equivalently

$$\text{level}(u) \leq \text{level}(v) \leq \text{level}(u) + 1$$

Thus two adjacent in G vertices u and v are at most one level apart in T. □

Note that in the above result the adjacency comes from the whole graph where as the level information is based on the tree. In particular, non-tree edges join vertices *at most* one level apart, that is non-tree edges join vertices either at the same level or exactly one level apart. The above property can be applied to obtain various information about the graph itself.

Theorem 25. *A connected graph G with BFS tree T has an odd cycle if and only if there is a non-tree edge that joins vertices at the same level*

Proof. Suppose the graph has a non-tree edge e that joins vertices at the same level. Follow the path to the first common ancestor from both vertices incident with the edge e. Along with e the result is an odd cycle.

Suppose all non-tree edges join vertices at different levels. By Theorem 24 the levels are exactly one level apart in this case. Then the set of vertices at even level and the set of vertices of odd level form a bipartition of the graph. Thus the graph is bipartite and by Theorem 21 it contains no odd cycle. □

Theorem 26. *The length of a shortest path from u to v in a connected graph G equals the level of v in any BFS tree of G with u as root*

Proof. Self study problem. □