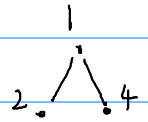
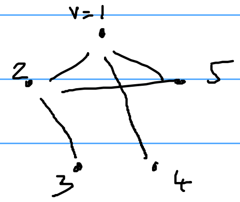
4.3 Paths and Cycles

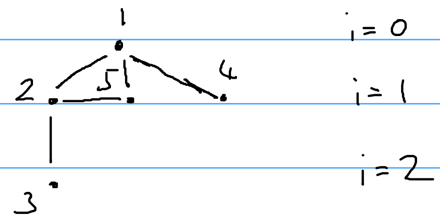
Theorem Let G be a graph and $v \in V(G)$. Let T be the tree obtained by running breadth-first search from v. Then, for any $u \in V(T)$, the unique v-u-path in T has minimum length among all v-u-paths in G.

Proof. Focus on the connected component of G that contains v. Arrange the vertices into layers such that layer i contains all vertices u such that a shortest v-u-path in G has length i.

bf-search
$U = 1, 2, 4$

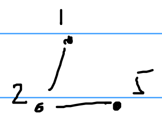






Note: all edges are between consecutive layers or within the same layer

df-search
$U = 1, 2, 5$

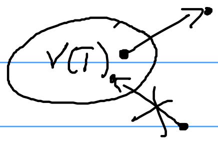


We now argue by induction over layers. The claim trivially holds for layer 0, which contains only vertex v. Now assume that the claim holds for vertices in layer i, and consider a vertex in layer $i+1$. By definition of the layers, G contains a edge between u and a vertex in layer i, and no edges between u and any vertex in layers $\ell < i$.
Note that vertices in layer i appear in the ordered sequence U used by breadth-first search before any vertices in layers $\ell > i$. The algorithm thus adds an edge between u and a vertex w in layer i. Together with a v-w-path of minimum length in G, which is contained in T by the induction hypothesis, the edge wu forms a v-u-path of minimum length in G. $\square$

For shortest cycles, see exercises.

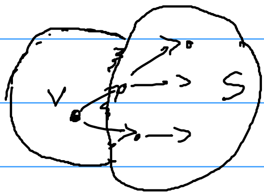
Idea: a shortest cycle consists of an edge uv plus a shortest u-v-path not containing that edge.

For shortest directed paths, run breadth-first search but only add arcs with tail in $V(T)$ and head in $V(G) \setminus V(T)$



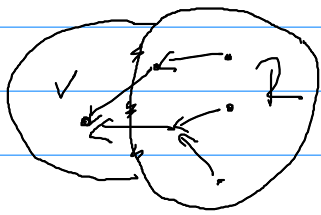
4.4 Strongly connected components

Idea: run tree search twice, once following arcs in the forward direction (from tail to head), and once following arcs in the backward direction (from head to tail.



$s \in S$

set S of vertices in G such that there exist a directed v-s-path in G



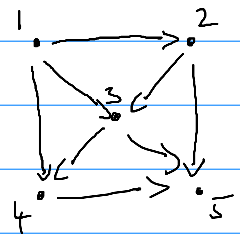
set R of vertices $r \in R$ such that there exists a directed r-v-path in G

$S \cap R$ is the set of vertices in the same strongly connected component as v

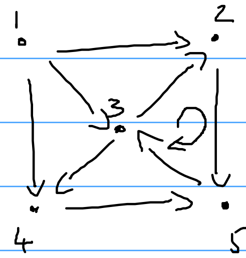
The strongly connected component is $G[S \cap R]$

4.5 Directed Cycles

Definition A digraph is a directed acyclic graph (or dag) if it does not contain any directed cycles.

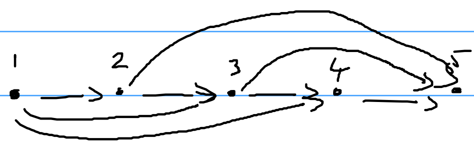


directed acyclic graph



not a directed acyclic graph

direct cycle 2, 5, 3, 2



all arcs go from left to right

any directed cycle must contain an arc that goes from right to left

therefore, no directed cycles, so a dag

Definition Let D be a digraph. A topological ordering of D is a total order $<$ of $V(D)$ such that $u < v$ whenever $uv \in A(D)$.

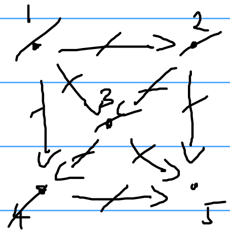
What we have drawn above is the topological ordering

$1 < 2 < 3 < 4 < 5$

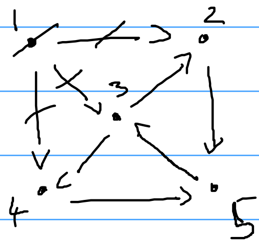
$d_D^-(v)=3$

Algorithm Let D be a digraph. Start with the empty sequence, then repeat the following steps until no vertices remain in D:

1. Let $v \in V(D)$ such that $d_D^-(v)=0$. If no such vertex exists, stop: there is no topological ordering.
2. Write down v as the next vertex in the sequence.
3. Remove v from D, along with all its incident arcs.



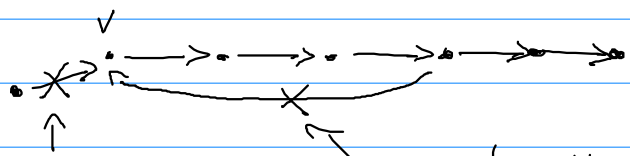
1, 2, 3, 4, 5



1, stop: no topological ordering

Lemma Let D be a directed acyclic graph. Then there exists $v \in V(D)$ such that $d_D^-(v)=0$.

Proof. Consider a directed path P of maximum length in D, and let v be the first vertex of P.



does not exist because P has maximum length

does not exist because there are no direct cycles

Therefore $d_D^-(v)=0$. $\square$

Theorem Let D be a digraph. Then D is a dag if and only if it has a topological ordering.

Proof, direction from right to left: obvious

direction from left to right: by the lemma, the algorithm runs until there are no vertices left and therefore constructs a topological ording.

The running time of the algorithm is $O(|V(D)|^3)$.

It deletes a vertex in every round where it doesn't stop, so it runs for at most $|V(D)|$ rounds. A vertex with indegree zero can be found and deleted along with its incident arcs using a constant number of basic operations for each of the $|V(D)|^2$ entries of the adjacency matrix of D. We find a column i in the matrix with all entries equal to 0 and remove column i and row i from the matrix.

5. Minimum Spanning Trees and Shortest Paths in Networks

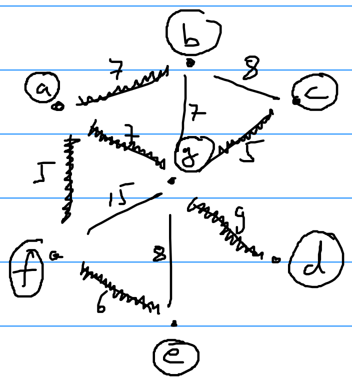
Definition Consider a network (G, w) and let weight of a subgraph H of G be $\sum_{e \in E(H)} w(e)$. A spanning tree T of G is a minimum spanning tree of (G, w) if it has minimum weight among all spanning trees of G. A u-v-path of G is a shortest u-v-path of (G, w) if it has minimum weight among all u-v-paths of G.

5.1 Minimum Spanning Trees

We will consider three different algorithms. They all find minimum spanning trees. They are all "greedy" algorithms: they construct a solution in small steps; in each step they optimise an objective without looking into the future.

<u>Algorithm (Prim)</u> Consider a network (G,w) and $s \in V(G)$. <u>Prim's algorithm</u> starts from the tree T with $V(T)=\{s\}$ and $E(G)=\emptyset$ and then repeats the following steps:

1. Let $F=\{ uv \in E(G) : u \in V(T), v \in V(G) \setminus V(T)\}$. If $F=\emptyset$, then stop.
2. Let $uv \in F$ such that $w(uv) = \min_{xy \in F} w(xy)$.
3. Add v to $V(T)$ and uv to $E(T)$.



Prim starting at a adds

af

fe

ab ← we could have added ag first

ag ← we could have added bg instead

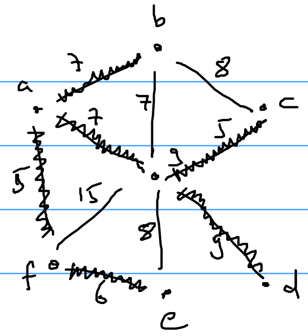
gc

gd

We obtain a spanning tree with weight $5+6+7+7+5+9=39$

<u>Algorithm (Kruskal)</u> Consider a network (G,w). Let F be a sequence of the edges of G in non-decreasing order of weight. Then <u>Kruskal's algorithm</u> starts from T with $V(T)=V(G)$ and $E(T)=\emptyset$. and then repeats the following steps until F is empty.

1. Let uv be the first element of F. Remove it from F.
2. Unless this would create a cycle with the edges in $E(T)$, add uv to $E(T)$.



F

af
cg

in any order

ef
ab
ag
~~bg~~

in any order

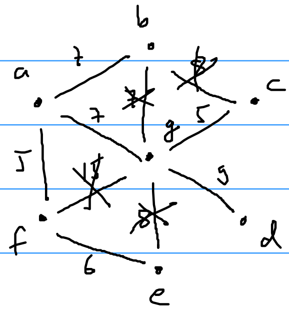
~~be~~
~~eg~~

in any order

dg
~~fg~~

<u>Algorithm (reverse-delete)</u> Consider a network (G, w). Let F be a sequence of the edges of G in non-decreasing order of weight. The <u>reverse-delete algorithm</u> starts from T with $V(T) = V(G)$ and $E(T) = E(G)$ and then repeats the following steps until F is empty:

1. Let uv be the last element of F. Remove it from F.
2. Unless this would result in a graph that is not connected, remove uv from $E(T)$.



F

af
cg
ef
ab
ag
~~bg~~
~~be~~
~~eg~~
dg
~~fg~~