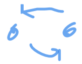
Minimum Spanning Trees (MST)

Trees

Skriv tekst her

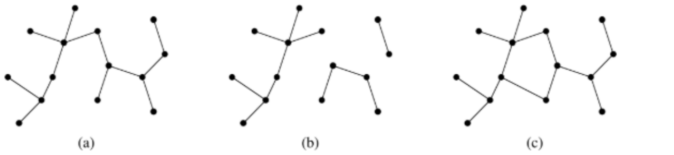
A (free/unrooted) tree is an undirected graph G = (V, E) that is:

- Connected: there is a path between every pair of vertices.
- Acyclic: there are no cycles. cycle starts and end same place



not allowed

no longer a root, since no direction



Tree

Forest

not connected) (not tree)

Graph with cycle (not a tree)

connected, but has a cycle (not tree)

(Undirected, acyclic graph = forest of trees)

collection of trees (forrest)

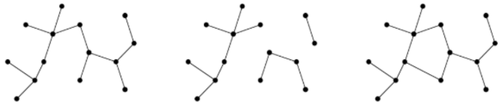
Trees

n = |V| = number of vertices/nodes
m = |E| = number of edges/lines

Theorem (B.2): For an undirected graph G = (V, E), the following are equivalent (if one holds, all hold):

if one is true all holds

1. G is a tree (i.e., connected and acyclic).
2. G is connected but becomes disconnected if any edge is removed.
3. G is connected and m = n − 1. one less edge than vertrices
4. G is acyclic but becomes cyclic if any edge is added.
5. G is acyclic and m = n − 1.
6. There is exactly one path between every pair of vertices.



(a) (b) (c)

Proof (not in syllabus): see appendix B.5.
Read (in syllabus) appendix B.4 and B.5 for basic graph definitions.

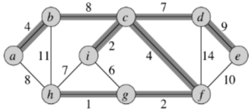
Minimum Spanning Tree (MST)

Spanning tree for an undirected, connected graph G = (V, E):

a subgraph $T = (V, E')$, $E' \subseteq E$, that is a tree. MUST STILL BE A TREE

only subset of edges

Note: same vertex set V. We think of T simply as E'.
By Theorem B.2, all spanning trees have n − 1 edges.



trying to minmize edges, while still kepping vertex connceted

Minimum Spanning Tree (MST) for a weighted undirected, connected graph G: a spanning tree for G with the smallest possible sum of edge weights (i.e., no spanning tree has a smaller sum).

Motivation: connect points in a supply network (electricity, oil, ...) as cheaply as possible. Edge in G: possible connection; weight: cost to establish the connection. This was the motivation for the first algorithm for the problem (Borůvka, 1926, Austria-Hungary, now Czech Republic).

Algorithm for MST

Greedy algorithm, always select the best solution to any subproblem
to hopefullt build the optimal global solution

Basic idea is a greedy algorithm: build the MST by choosing edges one by one according to an appropriate rule.

Correctness: via the usual invariant for greedy algorithms: “What we have built so far is part of an optimal solution.”

The invariant:

$\boxed{\text{There exists an MST containing the set A of chosen edges.}}$

Terminology: A safe edge for A is an edge that can be added without breaking the invariant (at least one exists when the invariant holds and $|A| < n - 1$).

Algorithm for MST

Invariant:There exists an MST containing the chosen edge set A.

```
GENERIC-MST($G, w$)
  $A = \emptyset$                          empty set of choosen edges
  while $A$ is not a spanning tree         a spanning tree for n vertices has n−1 edges,
                                           and connects all vertices
      find an edge $(u, v)$ that is safe for $A$
      $A = A \cup \{(u, v)\}$              the found safe edges is added to to the set A
  return $A$
```

- Initialization: Every connected graph has at least one spanning tree (via the theorem in B.5, point 2 – remove edges until connected), and therefore has an MST, which contains edge set $\varnothing$.
- Maintenance: By the definition of safe edges.
- Termination: Any (M)ST has exactly n − 1 edges. Since A grows by one edge per iteration, the invariant implies the algorithm terminates and that A is an MST (it is contained in an MST and has the same number of edges).

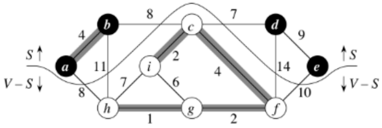
Cuts

How do you find a safe edge?

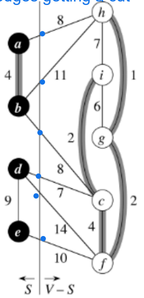
Cut: A subset S ⊆ of the vertices.

It can be seen as a partition of the vertices into S and V − S.

the edges getting a cut



(a)



(b)

same drawing, but moving it around
a bit so the white and blacks are on different sides

Edge crossing the cut: an edge in S × (V − S).

Cut Theorem

Theorem:

IF

- There exists an MST containing A,
- S is a cut with no edges in A crossing it,
- e is the lightest edge crossing that cut,

Then

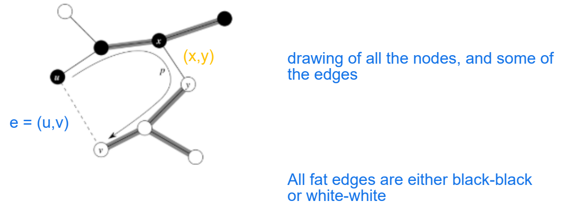
- e is safe for A (i.e., there exists an MST containing $A \cup \{e\}$).

Cut Theorem

Proof:

- There exists an MST T that contains A.
- We will construct an MST T′ that contains A ∪ {e}.

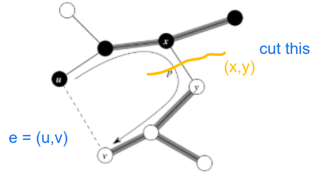
Let e = (u, v) be the lightest edge crossing the cut S.
Since T is connected, there must be a path in T between u and v which includes at least one edge (x, y) that also crosses the cut S.
Let T′ be the tree obtained by replacing (x, y) in T with e = (u, v).


x
(x,y)
p
y
u
e = (u,v)
v
drawing of all the nodes, and some of the edges
All fat edges are either black-black or white-white

(Shown edges = T, bold edges = A, the cut is indicated by vertex colors.)

Cut Theorem

Form T′ by replacing (x, y) in T with e = (u, v).



Since T′ remains connected (in every path, the edge (x,y) can be replaced by the remainder of the path from u to v together with the edge (u,v)), and has n vertices and n − 1 edges, T′ is therefore a tree (by the earlier theorem). It can only be lighter than T, so T′ is also an MST.

Moreover, T′ contains A ∪ {e}, because the removed edge (x,y) was not in A—A has no edges crossing the cut.

Use of the Cut Theorem in MST Algorithms

```
GENERIC-MST(G, w)
  A = ∅
  while A is not a spanning tree
      find an edge (u, v) that is safe for A
      A = A ∪ {(u, v)}
  return A
```

Skriv tekst her

Invariant: There exists an MST containing the chosen edges A.

- An edge (u, v) with both endpoints in the same component of G′ = (V, A) would create a cycle and break the invariant; such edges are never safe

would create a cycle

- An edge (u, v) with endpoints in different components C1 and C2 of G′ = (V, A) is safe if it is the lightest edge out of C1: apply the cut theorem to the cut defined by C1.

lightes edge = edge with less weight

It is easy to see that if A is extended by an edge whose endpoints lie in different connected components C_1 and C_2 of G′, then the connected components of G′ change by merging C_1 and C_2 into a single connected component.

Prim–Jarnik MST Algorithm (Prim 1957, Jarnik 1930)

Start from an (arbitrary) root node r. Continually expand the connected component of r in G′.

4<8

8< 11, so dont matter what 8 we pick

make a cut between the components (black) and single nodes (white)and take the minmum weight

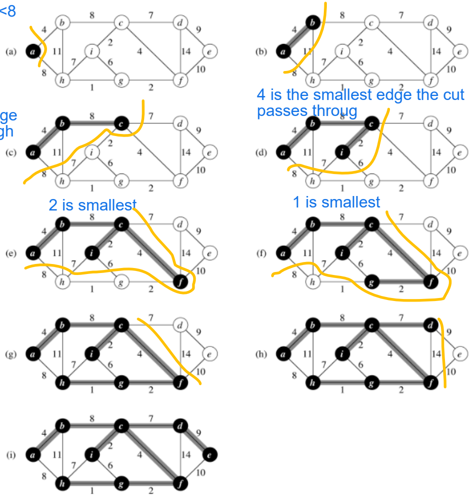
smalles value in the edge that our cut pass through = 2

4 is the smallest edge the cut passes throug

as we select paths, we increate the size of the component, by choosing the smallest edge, to connect the nodes

2 is smallest

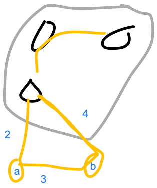
1 is smallest



Prim–Jarnik MST Algorithm

keep the edge from the component (black) to each of the nodes (white) in a min_priority que. update the que every time we include a white node in the component. (since this new included node, could have edges with lower weight to the now bigger component

Start from an (arbitrary) root node r. Continuously expand r’s connected component C in G′ = (V, A).
Each vertex v ∈ V − C stores in v.key and v.π the information about its shortest edge crossing the cut C. The set A is {(v, v.π) | v ∈ C − {s}}.
The vertices in V − C are kept in a (min-)priority queue Q.



once a is includede in the component the edge from a to b (3), is less than component to b (4). So update the min_PQ from component to b

```
PRIM(G, w, r)
  Q = ∅
  for each u ∈ G.V
      u.key = ∞
      u.π = NIL
      INSERT(Q, u)
  DECREASE-KEY(Q, r, 0)      // r.key = 0
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      for each v ∈ G.Adj[u]
          if v ∈ Q and w(u, v) < v.key
              v.π = u
              DECREASE-KEY(Q, v, w(u, v))
```

$\}$ initzialize

Correctness: Via the Cut Theorem and the invariant.
Running time: n Insert, n ExtractMin, m DecreaseKey operations on a priority queue of size O(n), for a total of O(m log n).

Kruskal MST Algorithm (1956)

Attempts to add edges to A in global lightest-first order.
Recall:

1. An edge (u, v) can never be added to A if u and v lie in the same connected component of G′ = (V, A).

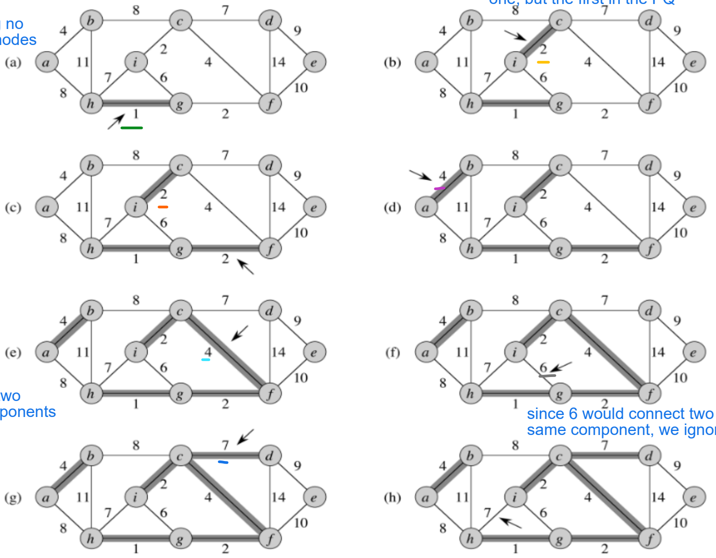
2. If an edge (u, v) is added to A, those two connected components merge into one.

Kruskal MST Algorithm

Order edges in ascending order:
1,2,2,4,4,6,7...

pick one of the 2's, it does not matter which one, but the first in the PQ

in the beginning no edge between nodes so all nodes are their own component



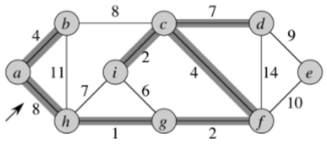
you can connect two DIFFERENT components

since 6 would connect two vertices in the same component, we ignore it

again this 7 is ignore because i and h, are alread connected by a different path, and there by in the same component continue from last slide

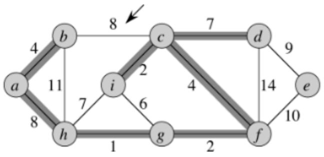
Kruskal MST Algorithm

(i)

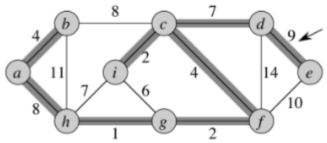


ignore 8, because b and c are already in the same components

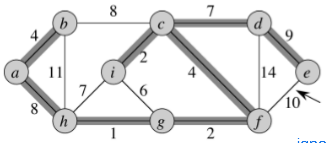
(j)



(k)

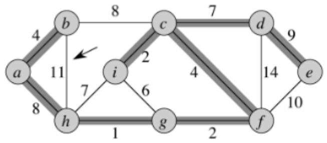


(l)



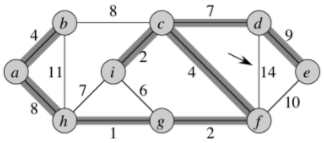
ignore 10

(m)



ignore 11

(n)



ignore 14

Now we have iterated over the whole PQ, and visted every edge, but only made connections to the one that where in different component or singleton

Kruskal MST Algorithm

Maintains the connected components in G′ = (V, A) using a disjoint-set data structure on V:

Make-Set(*x*), Union(*x, y*) Find-Set(*x*)

More precisely:

```
KRUSKAL($G, w$)
  $A = \emptyset$
  for each vertex $v \in G.V$
      MAKE-SET($v$)
  sort the edges of $G.E$ into nondecreasing order by weight $w$
  for each $(u, v)$ taken from the sorted list
      if FIND-SET($u$) $\neq$ FIND-SET($v$)        not in the same component
          $A = A \cup \{(u, v)\}$
          UNION($u, v$)
  return $A$
```

Kruskal MST Algorithm

```
KRUSKAL($G, w$)
  $A = \emptyset$
  for each vertex $v \in G.V$
      MAKE-SET($v$)
  sort the edges of $G.E$ into nondecreasing order by weight $w$
  for each $(u, v)$ taken from the sorted list
      if FIND-SET($u$) $\neq$ FIND-SET($v$)
          $A = A \cup \{(u, v)\}$
          UNION($u, v$)
  return $A$
```

It is clear from the discussion on page 11 about connected components that:

1. The data structure maintains the connected components of G′ = (V, A).
2. Any edge examined in the IF statement has both endpoints in the same connected component after the test, regardless of its outcome. Since connected components in G′ only merge as the algorithm proceeds, this remains true for that edge throughout the rest of the algorithm.

Kruskal, correctness

At the moment the algorithm adds an edge (u, v) to A, u and v lie in two different connected components C_1 and C_2 of G' = (V, A). (This follows from point 1 and the IF-test.)

We consider the cut defined by C_1, the component containing u. All lighter edges have already been examined, so by point 2 each such edge has both endpoints in the same component of G'. Hence (u, v) is the lightest edge crossing this cut, and we may apply the cut theorem.

When the algorithm terminates, every edge in the input graph G = (V, E) has been considered. Consequently, each remaining edge has both endpoints in the same component of G' = (V, A) (point 2), and thus cannot be added without creating a cycle.

Therefore the set A itself is the MST (by the invariant), since no further edges can be added. This establishes the correctness of the algorithm.

Note also that exactly n-1 Union operations are performed—one for each edge added to A—and an MST has exactly n-1 edges.

Kruskal, Running Time

Work:

Sort the m edges
Perform n Make-Set, $n-1$ Union, and m Find-Set operations
From earlier: there is a disjoint-set data structure for which

- n Make-Set(x)
- n − 1 Union(x, y)
- m Find-Set(x)

together take $O(m + n \log n)$ time.

Overall running time of Kruskal:

$$O(m \log m)$$

since $m \geq n - 1$, for a connected input graph.