SE284: Introduction to Graph Algorithms

Katerina Taškova

Email: katerina.taskova@auckland.ac.nz
Room: 303S.483

Outline

Minimum spanning trees (continued)

Lecture Notes 33, Textbook 6.5

Acknowledgment for slide content: Michael Dinneen, Simone Linz

Minimum spanning tree (MST) problem

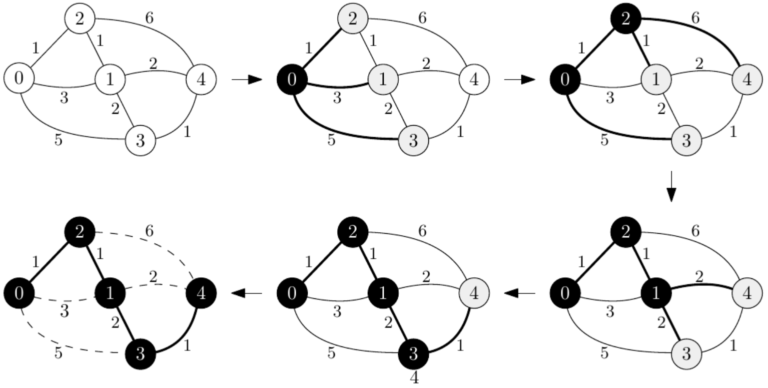
- Given a connected weighted graph, find a spanning tree (subgraph containing all vertices that is a tree) of minimum total weight.
- We introduced Prim's algorithm, an efficient greedy algorithm that optimal solves the MST problem.
- The greedy choice: select edges in order of increasing weight subject to two constraints
  - the subgraph build so far is connected,
  - the subgraph build so far is acyclic.
- Prim's algorithm maintains a tree at each stage that grows to span; It has very similar implementation to Dijkstra, runs in best case in time $O(m + n \log n)$.

Prim's algorithm – reminder

algorithm Prim(weighted graph (G, c), vertex s)
$S \leftarrow \{s\}$ $\boxed{\text{first vertex added to MST}}$
$E \leftarrow \emptyset$
while $S \neq V(G)$ **do**
  find a minimum weight edge $e = \{u, v\}$ such that $u \in S$, $v \notin S$
  $S \leftarrow S \cup \{v\}$ $\boxed{\text{adding } \{u, v\} \text{ to MST}}$
  $E \leftarrow E \cup \{e\}$
end while
return E

Prim's algorithm – example

Example 33.3. Application of Prim's algorithm on a graph, choosing node with lowest index where there is a choice.



Kruskal's algorithm

- Kruskal's is another efficient greedy algorithm to solve the MST problem.
- The greedy choice: select edges in order of increasing weight subject to single constraint
  - the subgraph build so far is acyclic.
- Kruskal's algorithm maintains a forest whose trees coalesce into one spanning tree.
- Kruskal can be implemented to run in time $O(m \log n)$.

Kruskal's algorithm

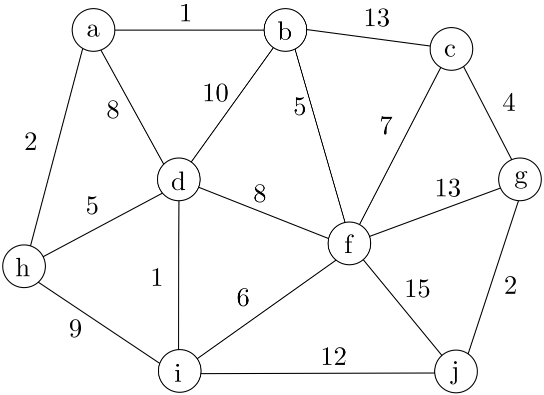
algorithm Kruskal(weighted graph (G, c))
$T \leftarrow \emptyset$
insert $E(G)$ into a priority queue
for $e = \{u, v\} \in E(G)$ in increasing order of weight **do**
    if u and v are not in the same tree **then**
        $T \leftarrow T \cup \{e\}$
        merge the trees of u and v
    end if
end for
return T

Keep track of the trees using disjoint sets ADT, with standard operations FIND and UNION. They can be implemented efficiently so that the main time taken is the sorting step.

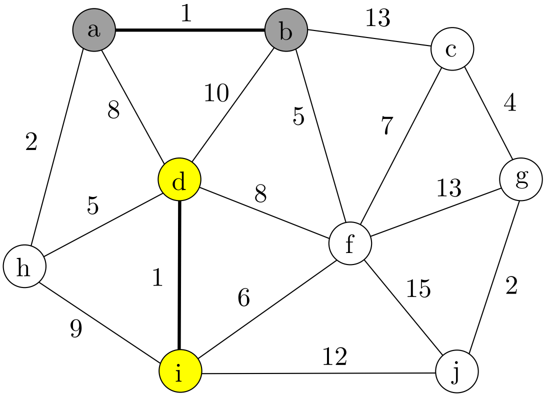
Kruskal's algorithm

```
function KRUSKAL(weighted digraph (G, c))
    disjoint sets ADT A
    initialize A with each vertex in its own set
    sort the edges in increasing order of cost
    for each edge {u, v} in increasing cost order do
        if not A.set(u) = A.set(v) then
            add this edge
            A.union(A.set(u), A.set(v))
    return A
```

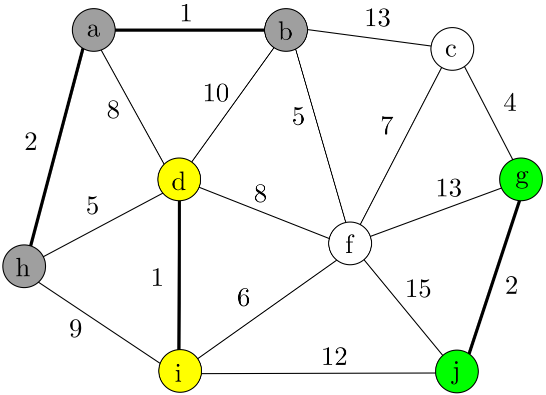
Illustrating Kruskal's algorithm



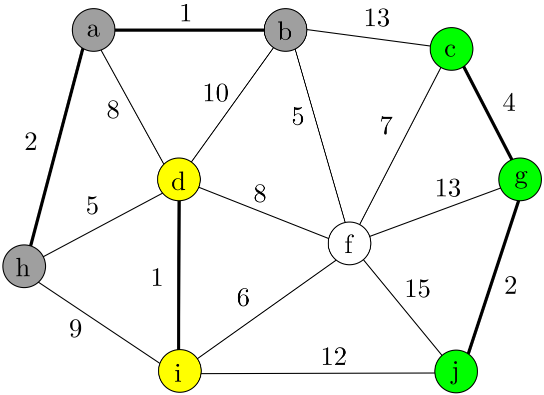
Illustrating Kruskal's algorithm



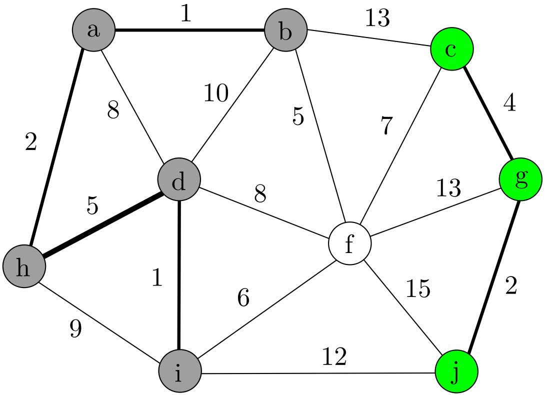
Illustrating Kruskal's algorithm



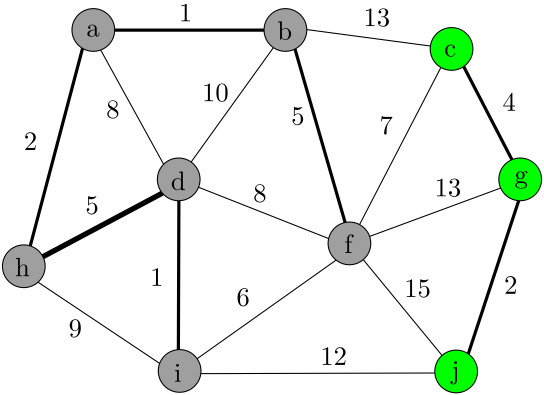
Illustrating Kruskal's algorithm



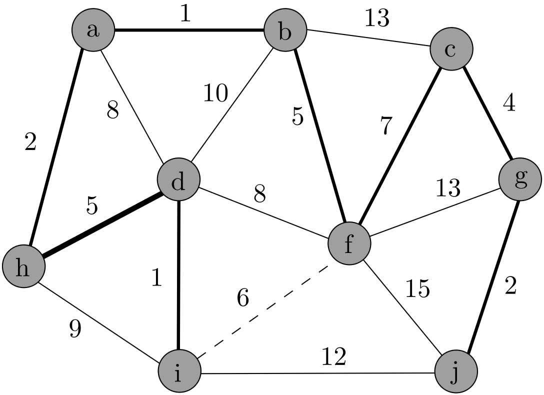
Illustrating Kruskal's algorithm



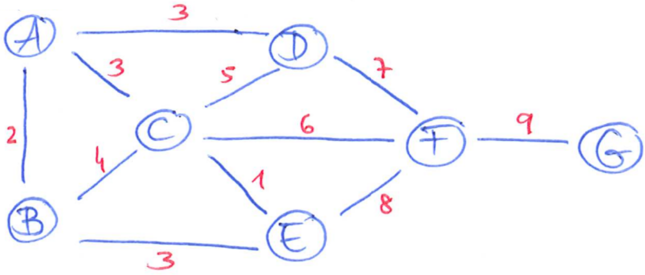
Illustrating Kruskal's algorithm



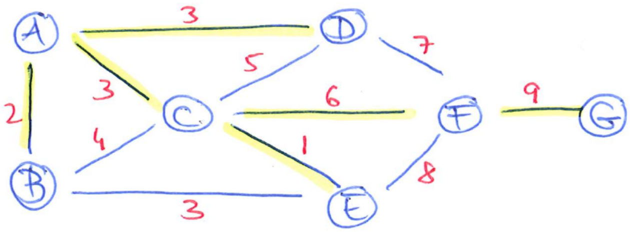
Illustrating Kruskal's algorithm



Kruskal's algorithm – example



Kruskal's algorithm – example

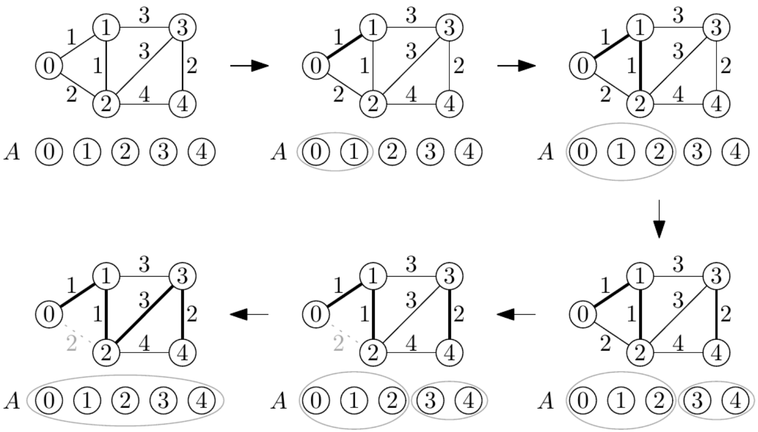


$$(\underset{1}{\{C,E\}},\ \underset{2}{\{A,B\}},\ \underset{3}{\{A,C\}},\ \underset{3}{\{A,D\}},\ \underset{6}{\{C,F\}},\ \underset{9}{\{F,G\}})$$

$\boxed{\text{MST}}$ of weight $\underline{24}$

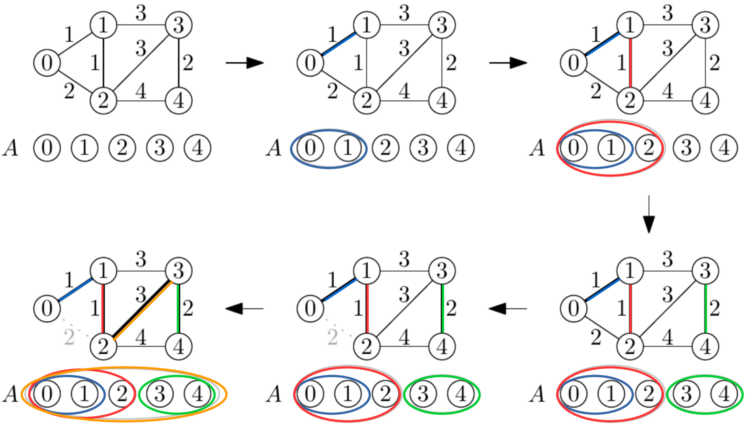
Kruskal's algorithm – example

Example 33.6. Application of Kruskal's algorithm on a graph shown until an MST is found. Note that the edge $\{0, 2\}$ with weight 2 is not added, because 0 and 2 are already in the same set in A.



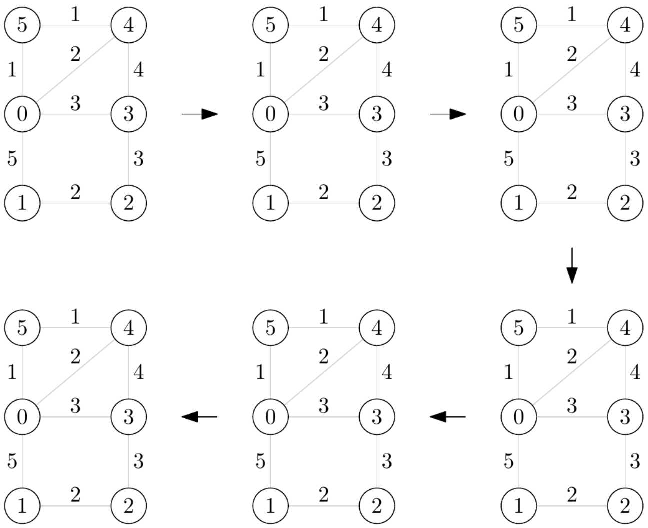
Kruskal's algorithm – example

Example 33.6. Application of Kruskal's algorithm on a graph shown until an MST is found. Note that the edge $\{0, 2\}$ with weight 2 is not added, because 0 and 2 are already in the same set in A.



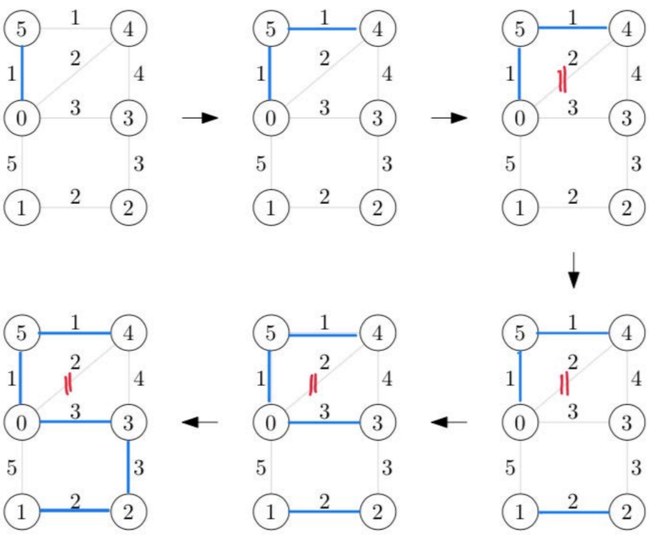
Kruskal's algorithm – example

Example 33.7. Execute Kruskal's algorithm by adding or crossing out the next edge. Stop when you reached an MST.



Kruskal's algorithm – example

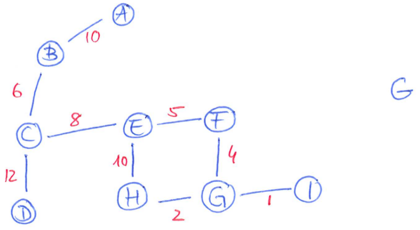
Example 33.7. Execute Kruskal's algorithm by adding or crossing out the next edge. Stop when you reached an MST.



Properties of minimum spanning trees

Fact

1. *The most expensive edge, if unique, of a cycle in an weighted graph G is not in any MST.*
   (Otherwise, at least one of those equally expensive edges of the cycle must not be in each MST.)



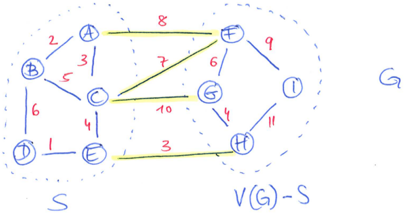
MST does not contain $\{E, H\}$.

2. *The minimum cost edge, if unique, between any non-empty strict subset S of $V(G)$ and the $V(G) \setminus S$ is in the MST.*
   (Otherwise, at least one of these minimum cost edges is in each MST.)

Properties of minimum spanning trees

Fact

1. *The most expensive edge, if unique, of a cycle in an weighted graph G is not in any MST.*
   (Otherwise, at least one of those equally expensive edges of the cycle must not be in each MST.)
2. *The minimum cost edge, if unique, between any non-empty strict subset S of $V(G)$ and the $V(G) \setminus S$ is in the MST.*
   (Otherwise, at least one of these minimum cost edges is in each MST.)



S={A,B,C,D,E}
V(G)-S = {F,G,H,I}
MST of G contains {E,H}.

Summary

- The MST of a connected weighted graph is a spanning tree of minimum total weight.
- The MST of a connected weighted graph is not always unique.
- Two efficient greedy algorithms for solving the MST problem: Prim's and Kruskal's.
- Each selects edges in order of increasing weight but avoids creating a cycle.
- Prim's algorithm maintains a tree at each stage that grows to span; Kruskal's algorithm maintains a forest whose trees are merged until one spanning tree is obtained.
- Prim runs in best case in time $O(m + n \log n)$; Kruskal can be implemented to run in time. $O(m \log n)$.

Thank you!