Math 2603 - Lecture 23
Section 12.3 & 12.5 Minimal Spanning Trees & Depth-First Search

Bo Lin

November 14th, 2019

Minimal Spanning Tree Algorithms

The problem

As we introduced last time, given a weighted connected graph, we would like to find a spanning tree with minimal weight.

The problem

As we introduced last time, given a weighted connected graph, we would like to find a spanning tree with minimal weight.

Remark

There might be multiple minimal spanning trees. But usually we only need to find one.

Greedy algorithms

In 1950s, J.B. Kruskal and R.C. Prim both discovered algorithms for finding minimal spanning tree.

Greedy algorithms

In 1950s, J.B. Kruskal and R.C. Prim both discovered algorithms for finding minimal spanning tree.

Remark

In both algorithms, they produce one edge at a time, and throughout the algorithm they make sure that the choices of edges are optimal.

Greedy algorithms

In 1950s, J.B. Kruskal and R.C. Prim both discovered algorithms for finding minimal spanning tree.

Remark

In both algorithms, they produce one edge at a time, and throughout the algorithm they make sure that the choices of edges are optimal.

Remark

This kind of algorithms are called **greedy algorithms***. They are an important type of algorithms, widely used in optimization problems.*

Kruskal's algorithm

Algorithm 1 `Kruskal's Algorithm`

function $\textsc{Kruskal}(\mathcal{G})$

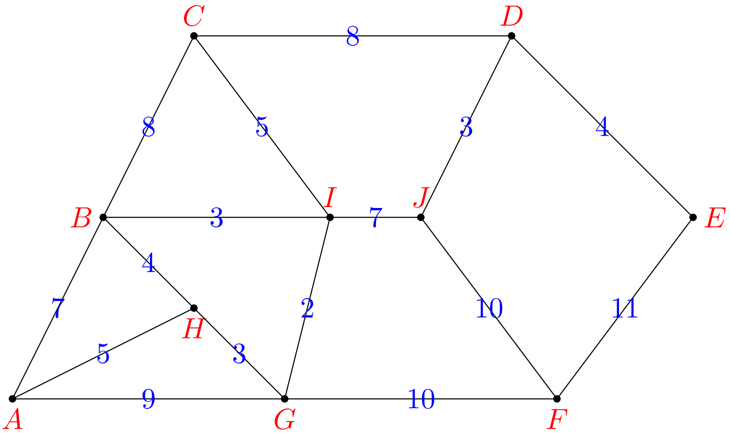
Input: a weighted connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n > 1$

Output: a set S of the $n-1$ edges of a minimal spanning tree of $\mathcal{G}$.

$e_1 \leftarrow$ an edge in $\mathcal{E}$ with minimal weight.
$k \leftarrow 1$
$S \leftarrow \{e_1\}$
while $k < n - 1$ **do**
  if $\exists e \in \mathcal{E}$ such that $\{e\} \cup S$ doesn't contain a circuit **then**
    $e_{k+1} \leftarrow$ such an edge with minimal weight
    $S \leftarrow S \cup \{e_{k+1}\}$
    $k \leftarrow k + 1$
Return S

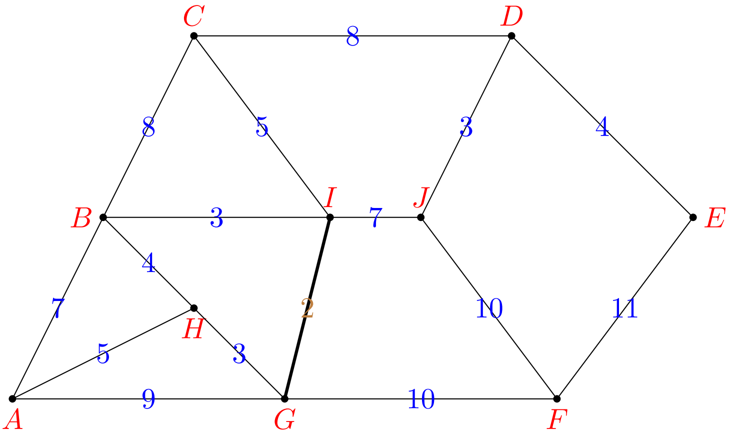
An Example

Find a minimal spanning tree:

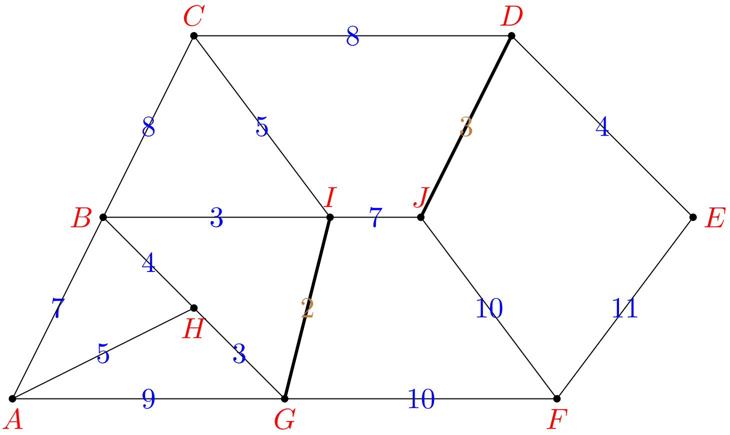


An Example

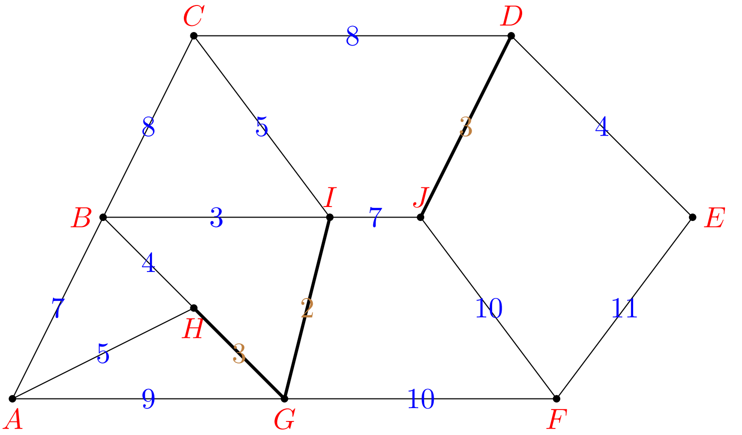
We denote labels of edges in S by brown color.



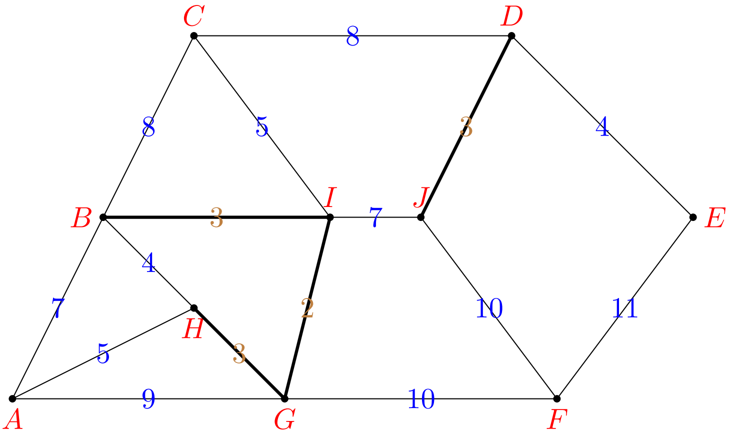
An Example



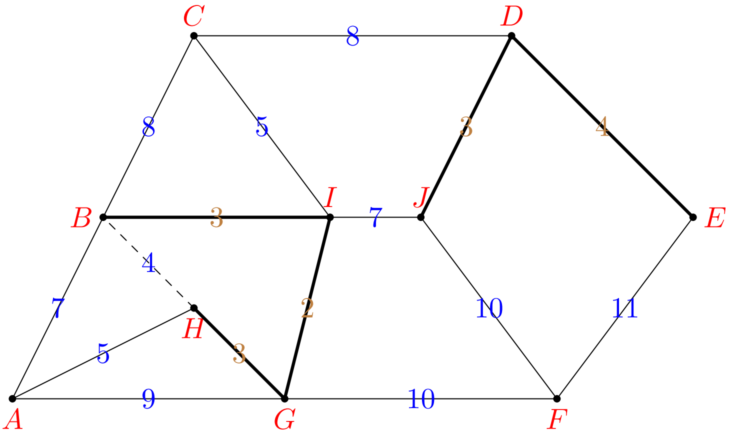
An Example



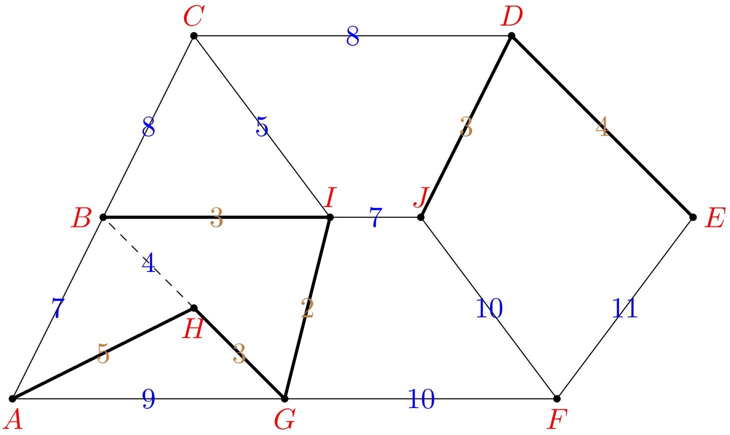
An Example



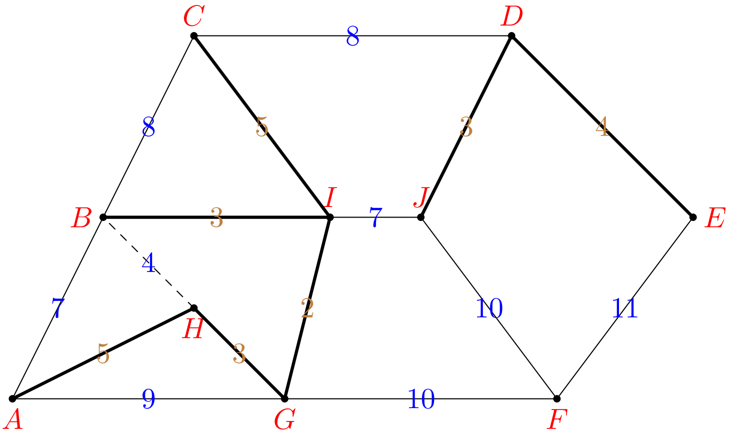
An Example



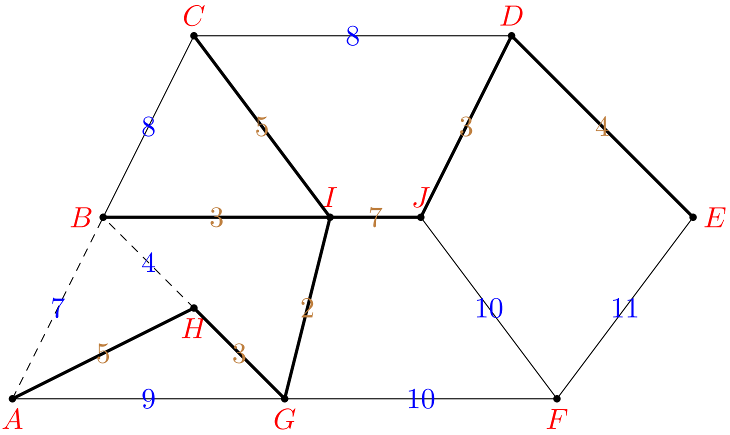
An Example



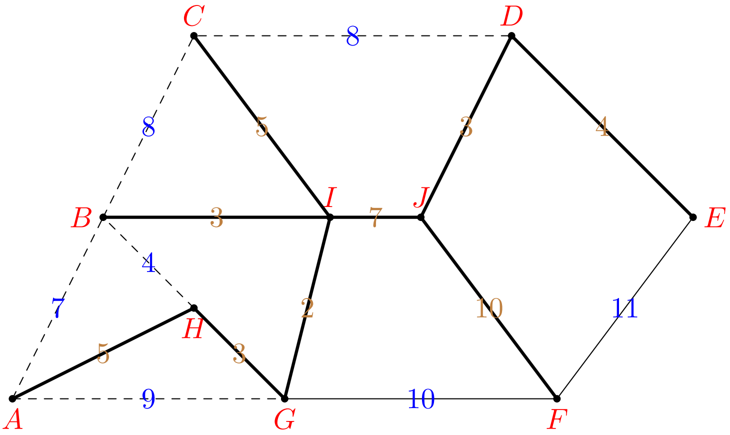
An Example



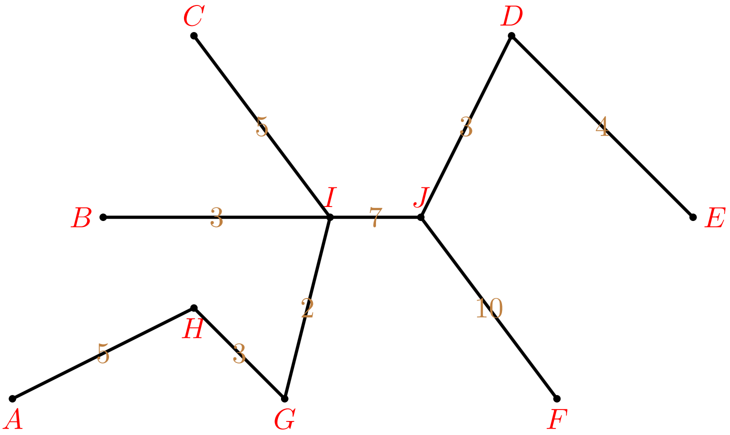
An Example



An Example



An Example - the spanning tree



Prim's algorithm

Algorithm 2 `Prim's Algorithm`

function $\textsc{Prim}(\mathcal{G})$

Input: a weighted connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n > 1$

Output: a set S of the $n-1$ edges of a minimal spanning tree of $\mathcal{G}$.

$v \leftarrow$ an arbitrary vertex in $\mathcal{V}$

$e \leftarrow$ an edge incident to v with minimal weight

$R \leftarrow \{v\}$ ▷ the set of covered vertices

$S \leftarrow \{e\}$ ▷ the set of edges in the tree

while $|R| < n$ **do**

$E \leftarrow \{e \in \mathcal{E} \mid e = \{u, x\}, u \in R, x \notin R\}$

$e = \{u, x\} \leftarrow$ an edge in E with minimal weight

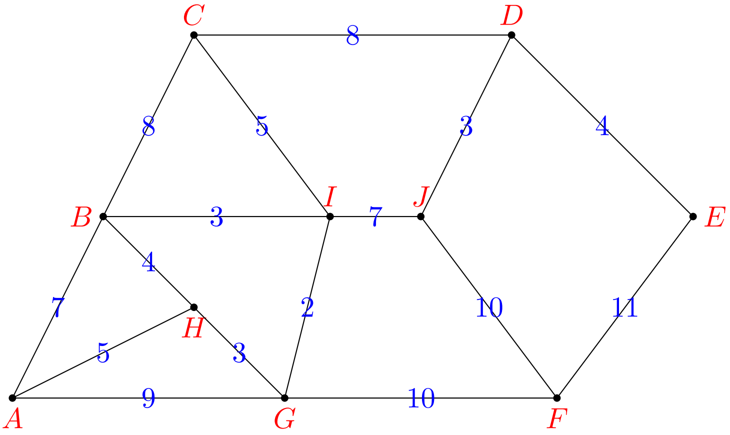
$R \leftarrow R \cup \{x\}$

$S \leftarrow S \cup \{e\}$

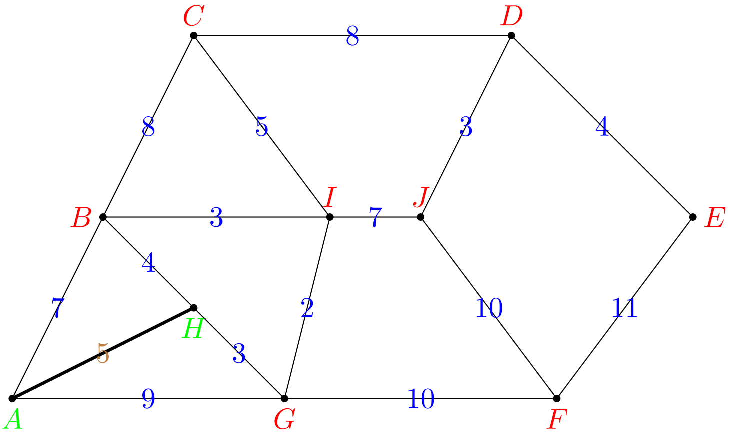
Return S

The same example

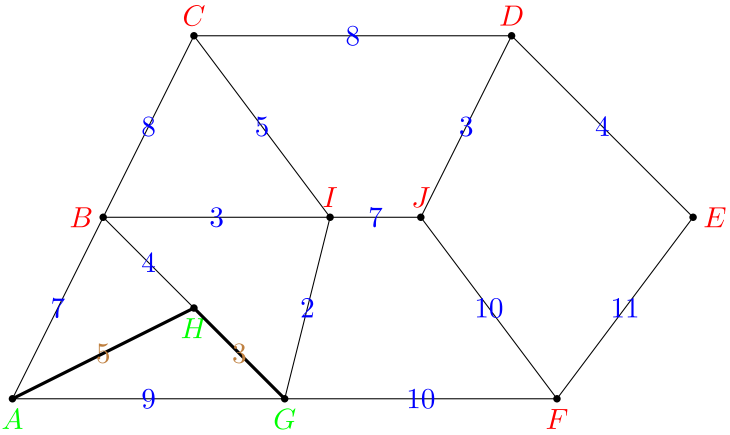
We denote vertices in T by green color and edges in S by brown color. We begin with vertex A.



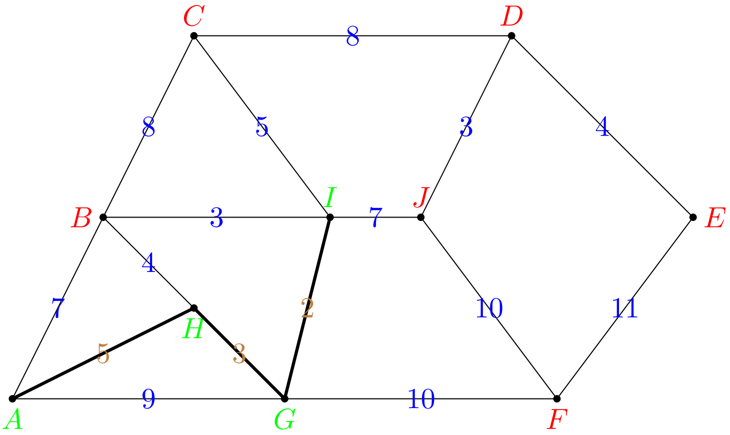
The same example



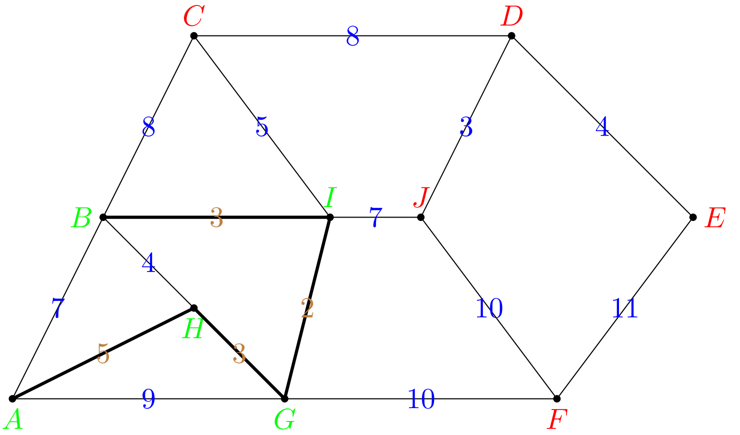
The same example



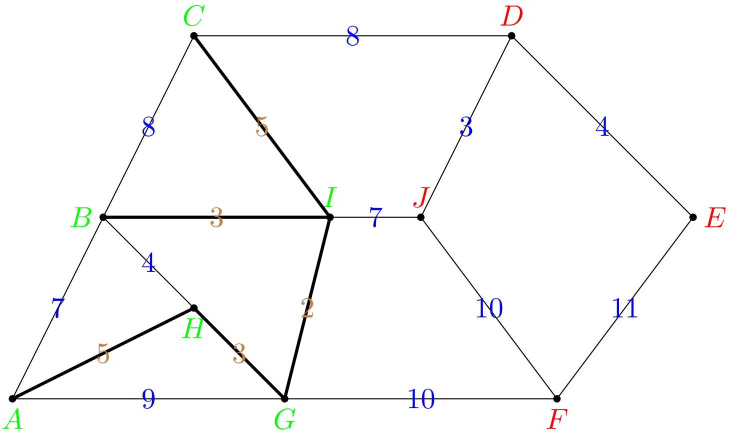
The same example



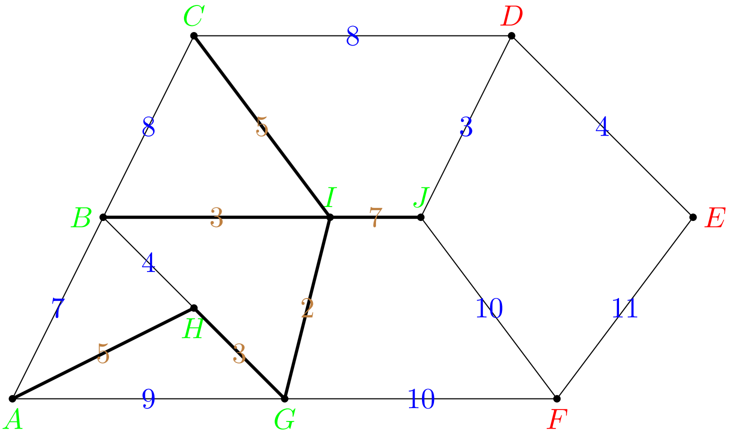
The same example



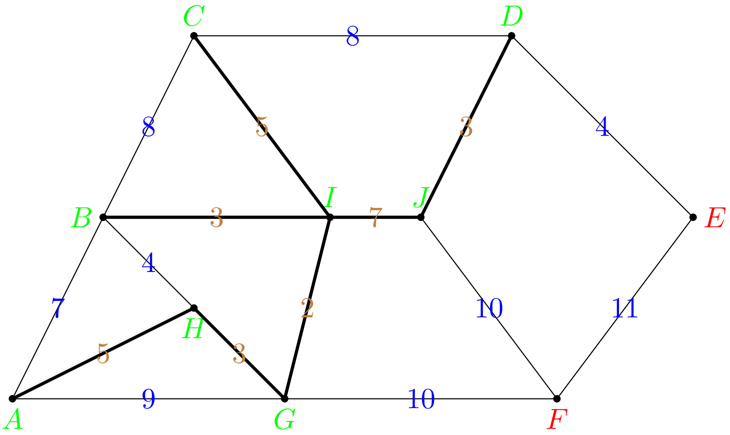
The same example



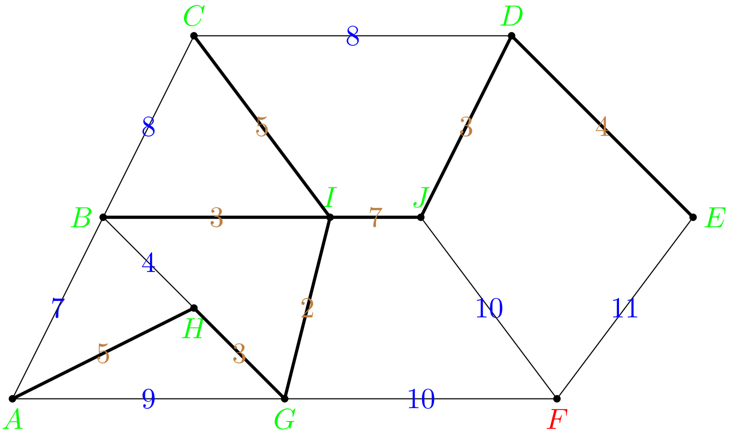
The same example



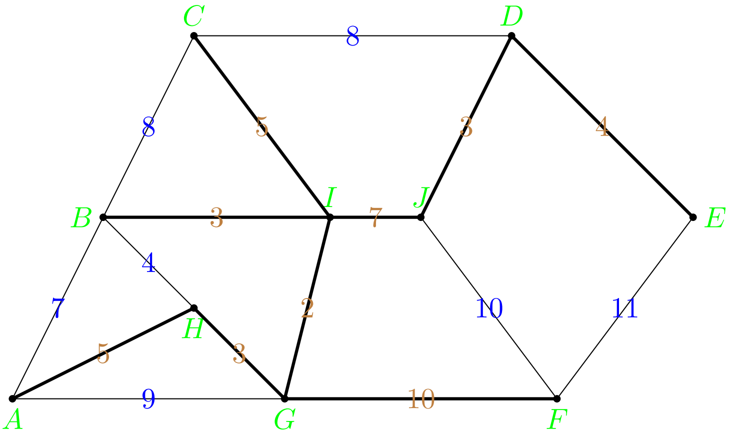
The same example



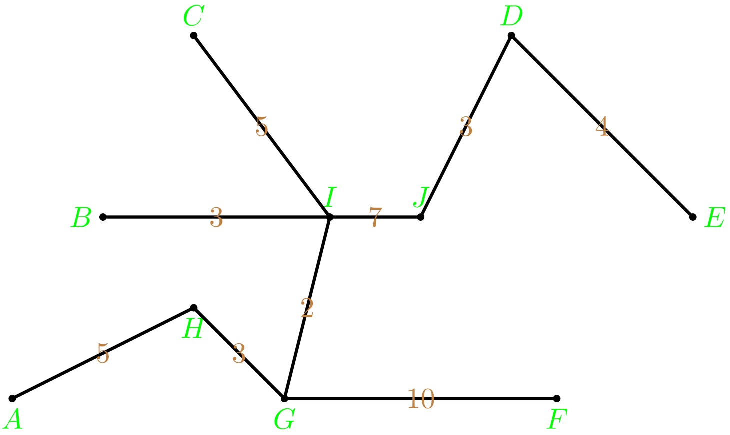
The same example



The same example



The spanning tree



Analysis of complexity

Proposition

Kruskal's Theorem can be done with $\mathcal{O}(N \log N)$ comparisons, where N is the number of edges in the graph $\mathcal{G}$.

Analysis of complexity

Proposition

Kruskal's Theorem can be done with $\mathcal{O}(N \log N)$ comparisons, where N is the number of edges in the graph $\mathcal{G}$.

Proposition

Prim's Theorem can be done with $\mathcal{O}(n^2)$ comparisons, where n is the number of vertices in the graph $\mathcal{G}$.

Remark

For proofs, see Exercise 12 and 13.

Analysis of complexity

Proposition

Kruskal's Theorem can be done with $\mathcal{O}(N \log N)$ comparisons, where N is the number of edges in the graph $\mathcal{G}$.

Proposition

Prim's Theorem can be done with $\mathcal{O}(n^2)$ comparisons, where n is the number of vertices in the graph $\mathcal{G}$.

Remark

For proofs, see Exercise 12 and 13.

Remark

Which algorithm is better? It depends on the number of vertices and edges. If the graph is sparse, Kruskal's algorithm is better; otherwise, Prim's algorithm is better.

Correctness of Prim's algorithm

Proof.

First, since each time we add a new vertex, the output is without circuit. By the connectedness of the graph, all vertices will be reached eventually (in other words, the algorithm will not terminate until we get $n-1$ edges). Hence the output is a spanning tree.

Correctness of Prim's algorithm

Proof.

First, since each time we add a new vertex, the output is without circuit. By the connectedness of the graph, all vertices will be reached eventually (in other words, the algorithm will not terminate until we get $n-1$ edges). Hence the output is a spanning tree. It suffices to prove that it has minimal weight. We prove that for $1 \le m \le n-1$, the subgraph consisting of the first m edges obtained by the algorithm is a subgraph of a minimal spanning tree.

Correctness of Prim's algorithm

Proof.

First, since each time we add a new vertex, the output is without circuit. By the connectedness of the graph, all vertices will be reached eventually (in other words, the algorithm will not terminate until we get $n-1$ edges). Hence the output is a spanning tree. It suffices to prove that it has minimal weight. We prove that for $1 \le m \le n-1$, the subgraph consisting of the first m edges obtained by the algorithm is a subgraph of a minimal spanning tree. Basis step: when $m = 1$, suppose there is a minimal spanning tree T not containing e_1. Then $T \cup e_1$ contains a circuit, and the first vertex $v \in R$ is incident to another edge in the circuit.

Correctness of Prim's algorithm

Proof.

First, since each time we add a new vertex, the output is without circuit. By the connectedness of the graph, all vertices will be reached eventually (in other words, the algorithm will not terminate until we get $n-1$ edges). Hence the output is a spanning tree. It suffices to prove that it has minimal weight. We prove that for $1 \le m \le n-1$, the subgraph consisting of the first m edges obtained by the algorithm is a subgraph of a minimal spanning tree. Basis step: when $m = 1$, suppose there is a minimal spanning tree T not containing e_1. Then $T \cup e_1$ contains a circuit, and the first vertex $v \in R$ is incident to another edge in the circuit. Now in T, replace this edge by e_1, we get another spanning tree, and by the choice of e_1, the total weight is no greater. So the new tree is also a minimal spanning tree, basis step is done. (to be continued) □

Correctness of Prim's algorithm

Proof.

Inductive step: suppose the statement is true when $m = k$ for some positive integer $k \leq n - 2$, we consider the case $m = k + 1$. The idea is almost the same. Suppose there is a minimal spanning tree not containing e_{k+1}, the $(k + 1)$-th edge added in Prim's algorithm, then adding e_{k+1} would produce a circuit.

Correctness of Prim's algorithm

Proof.

Inductive step: suppose the statement is true when $m = k$ for some positive integer $k \leq n - 2$, we consider the case $m = k + 1$. The idea is almost the same. Suppose there is a minimal spanning tree not containing e_{k+1}, the $(k+1)$-th edge added in Prim's algorithm, then adding e_{k+1} would produce a circuit. Let R_k be the set R after k steps, then $|R_k| = k$. Note that, there are at least 2 edges connecting a vertex in R_k and a vertex not in R_k! So other than e_{k+1}, there is still another such edge in the circuit. If we replace that edge by e_{k+1}, the new subgraph is still a tree and its weight is no greater, hence it is also a minimal spanning tree. The inductive step is done. $\square$

Depth-First Search

Motivating example

Example

Suppose you are in a labyrinth of rooms and one of the rooms contains treasure. You have many markers to leave in rooms, how do you search for the treasure?

Motivating example

Example

Suppose you are in a labyrinth of rooms and one of the rooms contains treasure. You have many markers to leave in rooms, how do you search for the treasure?

Remark

It's all about when you are at a location with multiple rooms to go next, what to do.

Motivating example

Example

Suppose you are in a labyrinth of rooms and one of the rooms contains treasure. You have many markers to leave in rooms, how do you search for the treasure?

Remark

It's all about when you are at a location with multiple rooms to go next, what to do. A straightforward idea is that always go to the next room that one haven't searched. And for rooms without treasure, you can leave a marker to indicate that it's already searched.

Motivating example

Example

Suppose you are in a labyrinth of rooms and one of the rooms contains treasure. You have many markers to leave in rooms, how do you search for the treasure?

Remark

It's all about when you are at a location with multiple rooms to go next, what to do. A straightforward idea is that always go to the next room that one haven't searched. And for rooms without treasure, you can leave a marker to indicate that it's already searched.

Remark

This is the idea of **depth-first search**.

The algorithm

Algorithm 3 `Depth-first Search`

function $\textsc{Depth-First Search}(\mathcal{G})$
Input: a graph $\mathcal{G}$ with n vertices
Output: labels $l(v)$ of distinct integers in $\{1, 2, \ldots, n\}$ on a subset of vertices v of $\mathcal{G}$.
  $v \leftarrow$ is an arbitrary vertex; $l(v) \leftarrow 1$; $L \leftarrow \{2, \ldots, n\}$; $k \leftarrow 1$
  while there is an unlabeled vertex **do**
    if $\exists w \in \mathcal{V}$ such that $l(w)$ is undefined and $\{w, v\}$ is an edge **then**
      $l(w) \leftarrow$ smallest element in L
      $L \leftarrow L - \{l(w)\}$; $k \leftarrow l(w)$; $pred(w) \leftarrow v$
      $v \leftarrow w$
    else if $k = 1$ **then**
      Return the labels $l(v)$
    else
      $v \leftarrow pred(v)$; $k \leftarrow l(v)$

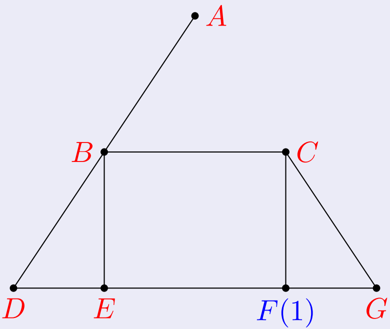
Example

Example

We apply the depth-first search to the right graph. We begin with vertex F.

k	v	$\min(L)$
1	F	2

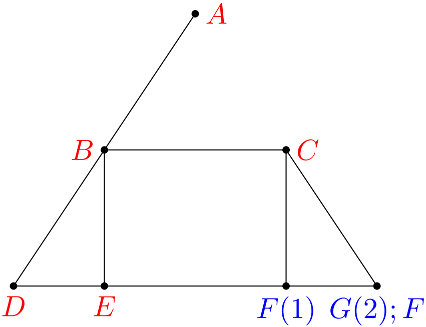
Blue vertices are labeled; red are unlabeled.



Example

We found C, E, G, arbitrarily take G and label it by 2.

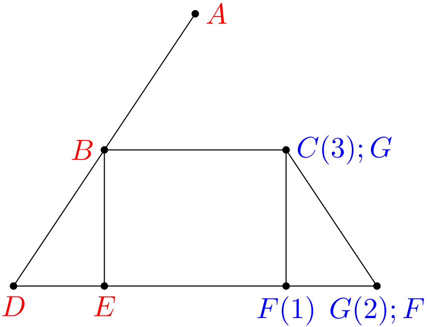
k	2
v	G
w	$\in \{C, E, G\}$
$\min(L)$	3



Example

We found C and label it by 2.

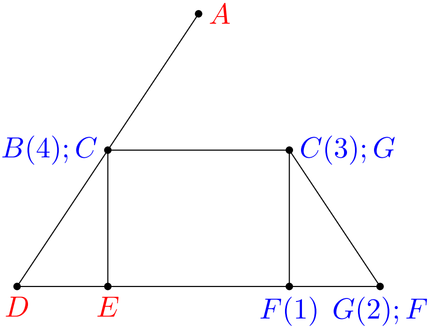
k	3
v	C
w	$\in \{C\}$
$\min(L)$	4



Example

We found B and label it by 4.

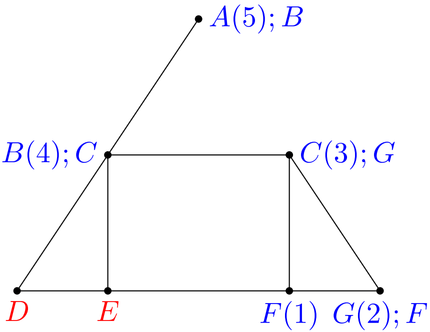
k	4
v	B
w	$\in \{B\}$
$\min(L)$	5



Example

We found A, D, E, arbitrarily take A and label it by 5.

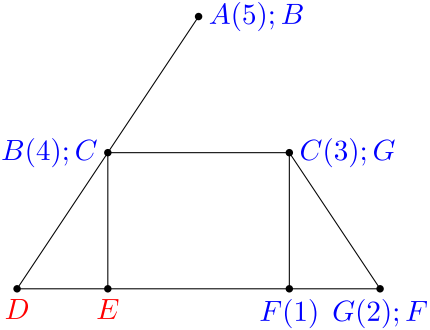
k	5
v	A
w	$\in \{A, D, E\}$
$\min(L)$	6



Example

We found nothing, thus backtrack to the predecessor B of current $v = A$.

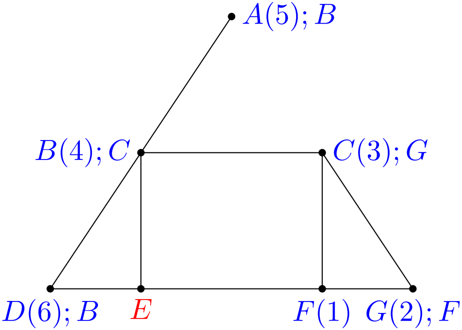
k	4
v	B
w	N/A
$\min(L)$	6



Example

We found D, E, arbitrarily take D and label it by 6, the minimal label in L.

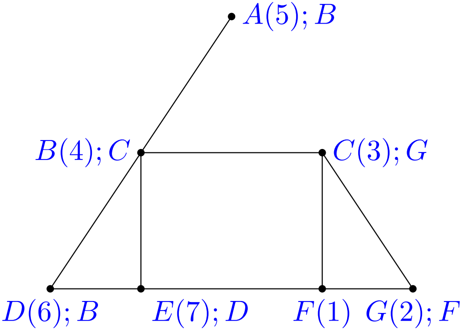
k	6
v	D
w	$\in \{D, E\}$
$\min(L)$	7



Example

We found E and label it by 7, the minimal label in L.

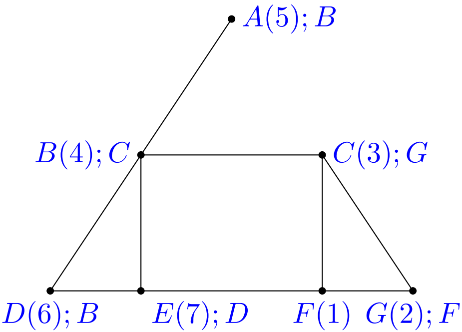
k	7
v	E
w	$\in \{E\}$
$\min(L)$	N/A



Example

We keep backtrack until $k = 1$.

k	v	$pred(v)$
7	E	D
6	D	B
4	B	C
3	C	G
2	G	F
1	F	N/A



The purpose

Proposition

A vertex is labeled by the depth-first search algorithm if and only if there is a path from v to it.

The purpose

Proposition

A vertex is labeled by the depth-first search algorithm if and only if there is a path from v to it.

Corollary

All vertices are labeled if and only if the graph is connected.

The purpose

Proposition

A vertex is labeled by the depth-first search algorithm if and only if there is a path from v to it.

Corollary

All vertices are labeled if and only if the graph is connected.

Remark

The depth-first search algorithm is used to check whether a graph is connected or not.

The purpose

Proposition

A vertex is labeled by the depth-first search algorithm if and only if there is a path from v to it.

Corollary

All vertices are labeled if and only if the graph is connected.

Remark

The depth-first search algorithm is used to check whether a graph is connected or not.

Remark

The edges covered by the algorithms forms a spanning tree of the connected component with v.

Complexity

Remark

Thinking of edges, one can check that the complexity of the depth-first search algorithm is $\mathcal{O}(n^2)$ where n is the number of vertices.

Complexity

Remark

Thinking of edges, one can check that the complexity of the depth-first search algorithm is $\mathcal{O}(n^2)$ where n is the number of vertices.

Proof.

For an arbitrary edge, suppose its endpoints have labels $a < b$. Then a was assigned first, and b was assigned later. The edge is considered twice - first, when $k = a$ and one found the unlabeled neighbor and labeled it by b; second, when $k = b$ and all neighbor are already labeled, then back track to the predecessor with label a. So the number of operations is up to $2|\mathcal{E}| \leq 2\binom{n}{2} = \mathcal{O}(n^2)$. □

Homework Assignment #13 - today

Section 12.3 Exercise 1(c), 2(d), 4(a).
Section 12.5 Exercise 1(c)(d), 4.