Chapter 23: Minimum Spanning Trees
Chapter 24: Single-Source Shortest Paths

23.1-1) Let (u, v) be a minimum-weight edge in a graph G. Show that (u, v) belongs to some minimum spanning tree of G.

Solution:
Proof: (by contradiction)
Let (u, v) be a minimum weight edge in a connected graph G. Assume (u, v) does not belong to a MST. Let T be a MST without the minimum weight edge (u, v). Add edge (u, v) to T. Then we have a cycle in the graph containing (u, v).

case1: From the cycle, erase the edge, e_α, with greatest weight to obtain the spanning tree, $\hat{T}$. (u, v) is obviously not removed because it is a minimum weight edge.
Then $(u, v) < e_\alpha \implies (u, v) - e_\alpha < 0$. Then we have that
$w(\hat{T}) = w(T) + w((u, v)) - w(e_\alpha) < w(T)$
This is a contradiction because T is a minimum spanning tree.
$\implies \hat{T}$ cannot weigh less than a MST
$\implies w(\hat{T}) = w(T)$
$\implies (u, v)$ must belong to some MST, $\hat{T}$.

case2: From the cycle, if there is no edge of greater size, delete an edge, e_β, of equal size as (u, v) (i.e. $(u, v) = e_\beta$) to obtain a spannning tree $\tilde{T}$. Then,
$w(\tilde{T}) = w(T) + w((u, v)) - w(e_\beta) = w(T)$
$\implies \tilde{T}$ is a minimum spanning tree and thus, (u, v) belongs to some MST

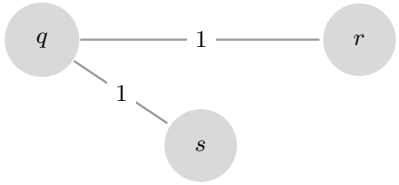
23.1-6) Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

Solution:

(a) Proof: (by contradiction)
Let $G = (V, E)$ be an undirected connected graph. Let T be a MST for G. Assume that G does not have a unique minimum spanning tree and assume that for every cut of the graph, there is a unique light edge crossing the cut. Then there exists $\tilde{T}$ different from T such that $w(T) = w(\tilde{T})$. Since $T \neq \tilde{T}$, $\exists$ an edge $(u, v) \in T : (u, v) \notin \tilde{T}$. Let T_1 and T_2 be the two trees connected by (u, v) such that $T_1 \cup T_2 = T$. Partition $\tilde{T}$ into two forests $\tilde{F_1}$ and $\tilde{F_2}$ such that $\tilde{F_1}$ has the same vertices as T_1 and $\tilde{F_2}$ has the same vertices as T_2. Let $(S, V - S)$ be a cut dividing $\tilde{F_1}$ and $\tilde{F_2}$. Then $(S, V - S)$ must respect $A = \{(x, y) : (x, y) \in \tilde{F_1}$ or $(x, y) \in \tilde{F_2}\}$. Necessarily, $\exists$ an edge $(\tilde{u}, \tilde{v})$ such that $\tilde{u}$ is part of $\tilde{F_1}$ and $\tilde{v}$ is part of $\tilde{F_2}$.
$\implies (\tilde{u}, \tilde{v})$ must be a light edge crossing the cut $(S, S - V)$
(u, v) is also a light edge crossing the cut $(S, S - V)$

$\implies$ Since light edges of cuts are unique, $(\tilde{u}, \tilde{v}) = (u, v)$
$\implies$ $(u, v) \in \tilde{T}$.
$\implies$ (u, v) was an arbitrary edge in T. Thus, $\forall (u, v) \in T, (u, v) \in \tilde{T}$
$\implies$ $T = \tilde{T}$
$\therefore$ G has a unique minimum spanning tree $\square$

(b) The converse of the statement is not true. If G has a unique minimum spanning tree, every cut does not necessarily have a unique light edge. Such is the case when we have a cut having two or more light edges (edges have the same weight).
The graph below obviously has a unique MST. However, the cut dividing the set of vertices $\{q\}$ and $\{r, s\}$ does not have a unique light edge. Both (q, r) and (q, s) are both light edges of the same cut.



23.2-1) Kruskal's algorithm can return different spanning trees for the same input graph G, depending on how ties are broken when the edges are sorted into order. Show that for each minimum spanning tree T of G, there is a way to sort the edges of G in Kruskal's algorithm so that the algorithm returns T.
Solution:

Let T be a MST of G. Let $E_T = \{e_1, e_2, ..., e_{|V|-1}\}$ be the edges connecting the vertices in T. Assume $e_1, e_2, ..., e_{|V|-1}$ are listed in nondecreasing order by weight. That is, $w(e_1) \leq w(e_2) \leq ... \leq w(e_{|V|-1})$. The $MST\text{-}KRUSKAL(G, W)$ algorithm adds safe edges starting with the lightest edge from a list of edges sorted in a nondecreasing order. Minimimum spanning trees are not always unique. However, any MST tree, T, can be generated using $MST\text{-}KRUSKAL$ if we let the sorted list $L = \{e_1, e_2, ..., e_{|V|-1}\}$ and then we insert each of the remaining edges $e_i = (u, v) \in G.E$ before the next largest $e_j \in L$. This guarantess that $e_1, e_2, ...,$ and $, e_{|V|-1}$ are looked at before any other safe edge with equal weight and is added to the MST, T. Thus, each MST in G can be generated if the edges are sorted as described above.

23.2-2) Suppose that the graph G = (V, E) is represented as an adjacency matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(V^2)$ time.
Solution:

MST-PRIM(G,r)

```
1  for i = 0 to |V| - 1
2    key[i] = ∞
3    parent[i]=NIL
4  key[r]=0
5  Create MIN–HEAP, Q, for the indices of the vertices using their key values
6  while Q ≠ NIL
7    i = EXTRACT–MIN(Q)
8    for j = 0 to |V| - 1
9      if G[i][j] > 0 AND j ∈ Q AND G[i][j] < key[j]
10       parent[j] = i
11       key[j] = G[i][j]
```

lines 1-4: initialize values, setting the root node with a key value of zero; runs in $O(V)$
line 5: creates a MIN-HEAP in $O(lgV)$

line 6-11: there are two main loops, the while and for loop. Each of them iterates through the number of vertices. Since the for loop is nected in the while loop, the total running time here is $O(V^2)$

line 7: will execute $O(VlgV)$ total times

line 9: will execute $O(ElgV)$ total times

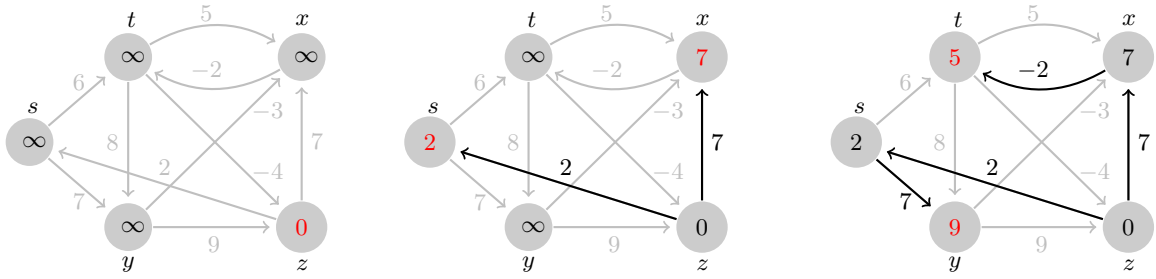
Total running time: $O(V) + O(lgV) + O(V^2) = O(V^2)$

24.1-1) Run the Bellman-Ford algorithm on the directed graph of Figure 24.4, using vertex z as the source. In each pass, relax edges in the same order as in the figure, and show the d and values after each pass. Now, change the weight of edge (z, x) to 4 and run the algorithm again, using s as the source

Solution:

(a) Bellman-Ford with source z; Figure 24.4 unmodified

- relaxation



v	d	π
s	2	z
t	4	x
x	6	y
y	9	s
z	0	NIL

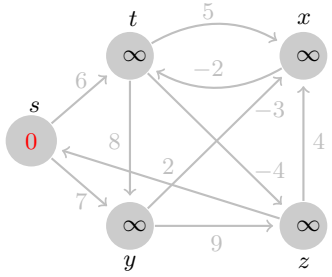
- Check for negative weight cycles:

u, v	$v.d \leq u.d + w(u, v)$	T/F
t, x	$6 \leq 4 + 5$	T
t, y	$9 \leq 2 + 8$	T
t, z	$0 \leq 4 + -4$	T
x, t	$4 \leq 6 + -2$	T
y, x	$6 \leq 9 + -3$	T
y, z	$0 \leq 9 + 9$	T
z, x	$6 \leq 0 + 7$	T
z, s	$2 \leq 0 + 2$	T
s, t	$4 \leq 2 + 6$	T
s, y	$9 \leq 2 + 7$	T

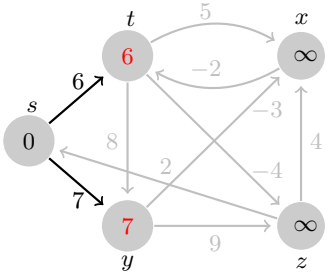
Every edge satisfyies the triangle inequality. Therefore, the BELLMAN-FORD algorithm returns TRUE, meaning that the graph does not have any negative-weight cycles.

(b) Bellman-Ford with source s; Figure 24.4 modified: $w(z, x) = 4$

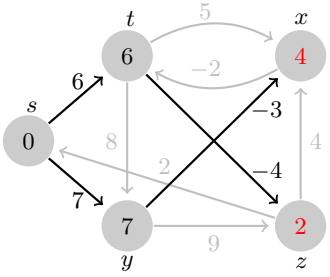
- relaxation



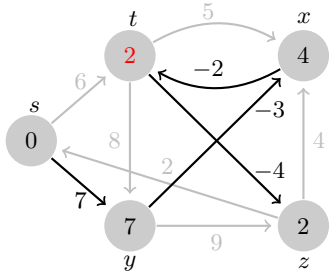
(initialization)



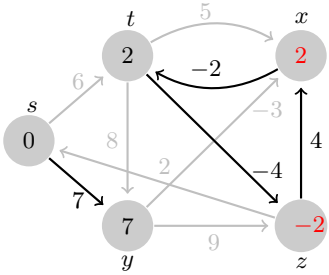
(1)



(2)



(3)



(4)

v	d	π
s	0	NIL
t	2	x
x	2	z
y	7	s
z	-2	t

- Check for negative weight cycles:

u, v	$v.d \leq u.d + w(u, v)$	T/F
t, x	$2 \leq 2 + 5$	T
t, y	$7 \leq 2 + 8$	T
t, z	$-2 \leq 2 + -4$	T
x, t	$2 \leq 2 + -2$	F
y, x	$2 \leq 7 + -3$	T
y, z	$-2 \leq 7 + 9$	T
z, x	$2 \leq -2 + 4$	T
z, s	$0 \leq -2 + 2$	T
s, t	$2 \leq 0 + 6$	T
s, y	$7 \leq 0 + 7$	T

The triangle inequality is not satisfied for edge (x, t). Therefore, the BELLMAN-FORD algorithm will return FALSE, meaning that the graph contains a negative-weight cycle.

24.1-3) Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all pairs of vertices $u, v \in V$ of the minimum number of edges in a shortest path from u to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

Solution:

The BELLMAN-FORD algorithm converges after the $|V|-1$ iterations of edge relaxation, assuming that G contains no negative-weight cycles. However, it's possible to reach converge before the $|V|-1^{th}$ iteration. We can optimize the BELLMAN-FORD algorithm by keeping track of any change done during the relaxations at each iteration. If the algorithm converges at the m^{th} iteration, we do not know that we have reach convergence until the $(m+1)^{th}$ iteration because we must check that no additional changes are made. Below is the modified BELLMAN-FORD algorithm.

Assume that RELAX, and BELLMAN-FORD both have access to the same boolean variable *change*.

```
RELAX(u,v,w)
1 if v.d > u.d + w(u,v)
2   v.d = u.d + w(u,v)
3   v.π = u
4   change = true;
```

```
BELLMAN-FORD(G,w, s)
1 INITIALIZE–SINGLE–SOURCE(G,s)  \\initialization
2 change = TRUE
3 iterations = 1;
4 while change == TRUE AND iterations ≤ |G.V| − 1  \\relaxations
5   iterations ++
6   change = FALSE
7   for each edge (u,v) ∈ G.E
8     change = RELAX(u, v, w)
9 for each edge (u,v) ∈ G.E   \\checking for negative−weight cycles
10  if v.d > u.d + w.(u,v)
11    return FALSE
12 return TRUE
```

[2]

24-2) A d-dimensional box with dimensions $(x_1, x_2, ..., x_d)$ **nests** within another box with dimensions $(y_1, y_2, ..., y_d)$ if there exists a permutation π on $\{1, 2, ..., d\}$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, ..., x_{\pi(d)} < y_d$.

a. Argue that the nesting relation is transitive.

b. Describe an efficient method to determine whether or not one d-dimensional box nests inside another.

c. Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, ..., B_n\}$. Describe an efficient algorithm to determine the longest sequence $< B_{i_1}, B_{i_2}, ..., B_{i_k} >$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, ..., k-1$. Express the running time of your algorithm in terms of n and d.

Solution:

a. Let X, Y, and Z be d-dimensional boxes with dimensions $(x_1, x_2, ..., x_d)$, $(y_1, y_2, ..., y_d)$, and $(z_1, z_2, ..., z_d)$ respectively. Let X nest in Y and Y nest in Z. (must show X nests in Z)

X nests in $Y \implies \exists$ a permutation π_x on $\{1, 2, ..., d\}$ such that $x_{\pi_x(i)} < y_i \ \forall i \in \{1, 2, ..., d\}$

Y nests in $Z \implies \exists$ a permutation π_y on $\{1, 2, ..., d\}$ such that $y_{\pi_y(j)} < z_j \quad \forall j \in \{1, 2, ..., d\}$

Let $i = \pi_y(j)$. Then we have that $x_{\pi_x(i)} < y_i = y_{\pi_y(j)} < z_j \implies x_{\pi_x(i)} < z_j$
Let $\pi_{xz}(j) = \pi_x(\pi_y(j))$ for $j \in \{1, 2, ..., d\}$
Then $\exists$ a permutation π_{xz} on $\{1, 2, ..., d\}$ such that $x_{\pi_{xz}(j)} < z_j \; \forall j \in \{1, 2, ..., d\}$. Therefore, X nests in Z □

b. Sort the dimensions for each box in ascending order. Assuming that the first entry of X is smaller than the first entry of Z, we can check whether X nests in Z by checking if each $x_i < z_i$. If this fails at some point, then X does not nest in Z.

c. NESTED-BOXES uses RELAX-BOX to check whether a pair of boxes is nested. If a condition is violated, then the box that violates the condition is removed from the list of nested boxes within the RELAX-BOX algorithm.

removes the mth box if box m is not nested in box n

RELAX-BOX(B[m],B[n],d)

```
1 if B[m][d] > B[n][d]
2   remove B[m] from the array list
```

NESTED-BOXES(B[][])

```
1 sort the dimensions of each of the boxes (each B[i])
2 sort the set of d–dimensional boxes based on their first, smallest
    dimension
3 for i = 1 to d
4   for each pair of box B[j], B[j+1] ∈ {B[k]}
5     RELAX–BOX(B[j], B[j+1], i)
```

Line 1: runs in $nO(d\ lgd)$
Line 2: runs in $O(lgn)$
Line 3-5: runs in $O(dn)$
Total running time: $O(d\ lgd + lgn + dn)$

References

[1] Cormen, Thomas. H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, Second Edition.* MIT Press, Cambridge, MA, 2009.

[2] http://student.csuci.edu/ douglas.holmes253/Assignment7.html