1. (Exercise 17.3-4) What is the total cost of executing *n* of the stack operations PUSH, POP, and MULTIPOP, assuming that the stack begins with s_0 objects and finishes with s_n objects?

Solution:

Let Φ be the potential function that simply returns the number of elements in the stack. We know that for this potential function, we have amortized cost 2 for the Push operation and amortized cost 0 for Pop and MULTIPOP operations.

Amortized cost of *i*-th operation is defined as

$$\hat{c}_i = c_i + \Phi(\Delta_i) - \Phi(\Delta_{i-1})$$

Thus, the total amortized cost is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \big(c_i + \Phi(\Delta_i) - \Phi(\Delta_{i-1})\big) = \sum_{i=1}^{n} c_i + \Phi(\Delta_n) - \Phi(\Delta_0)$$

Using the aforementioned potential function and the known amortized costs, we rewrite the equation as

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i + \Phi(\Delta_0) - \Phi(\Delta_n)$$

$$= \sum_{i=1}^{n} \hat{c}_i + s_0 - s_n$$

$$\leq 2n + s_0 - s_n$$

In particular, observe that if $s_n \geq s_0$, then $\sum_{i=1}^{n} c_i \leq 2n = O(n)$; that is, if the stack grows, then the work done is limited by the number of operations. If, on the other hand, $s_0 \geq s_n$, then $\sum_{i=1}^{n} c_i = O\big(n + (s_0 - s_n)\big)$, as it is possible there have been multiple MULTIPOP operations with large parameters.

2. (Exercise 22.4-5) Another way to perform topological sorting on a directed acyclic graph $G = (V, E)$ is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph. Explain how to implement this idea so that it runs in time $O(V + E)$. What happens to this algorithm if G has cycles?

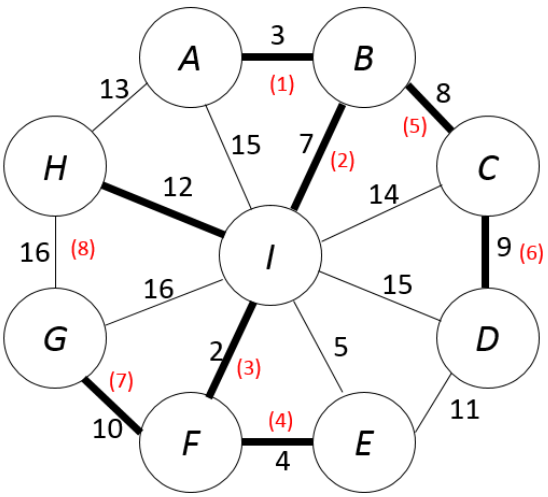
Solution:

First, for each vertex, we need to count how many incoming edges it has. We store these counts in a dictionary keyed by the vertices. This takes $\Theta(V + E)$ time. For each vertex with 0 incoming edges, store it in a special dictionary. Call that dictionary zero. We repeatedly do the following: Take a vertex out of zero. For every edge coming out of that vertex, decrement the count of that edge's target vertex. If you happen to decrement a count to 0, add that vertex to zero. This touches every vertex once and every edge once, so it also takes $\Theta(V + E)$ time. If there is a cycle, at some time when we try to take a vertex out of zero, we won't find any.

3. (Chapter 23: 10%) Consider the problem of finding the minimum spanning tree in the graph below. If we use Prim's algorithm, please show the tree growing sequence when the tree starts from vertex *A*. (Remember to explain Prim's algorithm briefly and time complexity, don't just show the answer.)

Solution:

Start from a single vertex A and grows until the tree spans all the vertices in {*A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I*}. Each step adds to the tree *T* a light edge that connects *T* to an isolated vertex—one on which no edge of *T* is incident. When the algorithm terminates, the edges in *T* form a minimum spanning tree. The computed MST is shown as follows:



The tree growing sequence is {*A*, *B*, *I*, *F*, *E*, *C*, *D*, *G*, *H*}. The time complexity using adjacency matrix, binary heap, Fibonacci heap are $O(V^2 + E)$, $O(E \log V)$, $O(E + V \lg V)$, respectively.

4. (Exercise 23.1-9) Let *T* be a minimum spanning tree of a graph *G* = (*V*, *E*), and let *V'* be a subset of *V*. Let *T'* be the subgraph of *T* induced by *V'*, and let *G'* be the subgraph of *G* induced by *V*. Show that if *T'* is connected, then *T'* is a minimum spanning tree of *G'*.

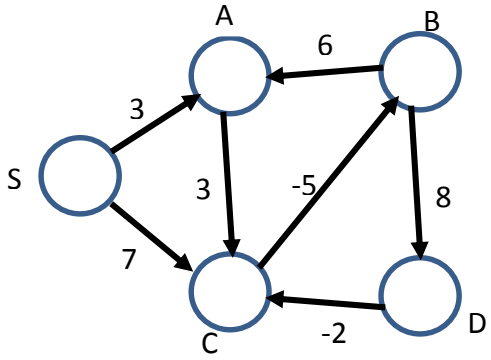
Solution 1:

Suppose that *T'* is not a minimum weight spanning tree in graph *G'* and *T''* is a minimum weight spanning tree in *G'*. Then, if we joined the subset of edges *T* –*T'* to *T''*, then we would obtain a spanning tree *S* in the graph *G*. The weight of *S* would be smaller than the weight of *T* and this contradicts the condition that *T* is a minimum weight spanning tree. Thus, our assumption is false and *T'* is a minimum weight spanning tree in the graph *G'*.

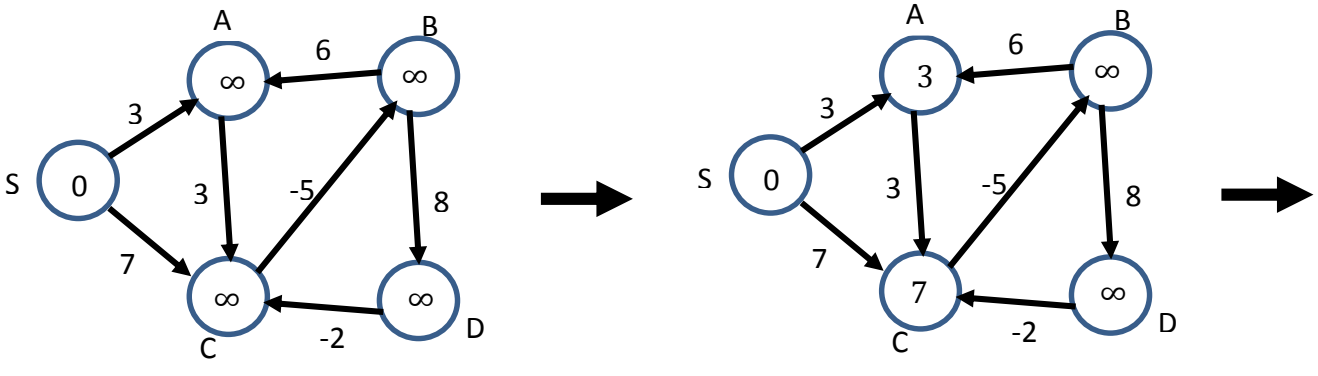
Solution 2:

Let $E' = \{(u, v) \in E \mid u, v \in V'\}$, those edges in $E - E'$ are not related to minimum spanning tree of G'. If there is a spanning tree T'' of G' with $w(T'') < w(T')$, then $T'' \cup (T - T')$ is a tree of G with $w(T'' \cup (T - T')) < w(T)$. Note that, number of edges in $T'' \cup (T - T')$ is the same as number of edges in T. So, $T'' \cup (T - T')$ is also a spanning tree with weight less than T. This leads to a contradiction with the fact that T is a minimum spanning tree of G. Thus, T' is a minimum spanning tree of G'.

5. Run the Bellman-Ford algorithm on the directed graph of Figure 1 to find one-to-all shortest path from source *S*. You need to detail each computational step.

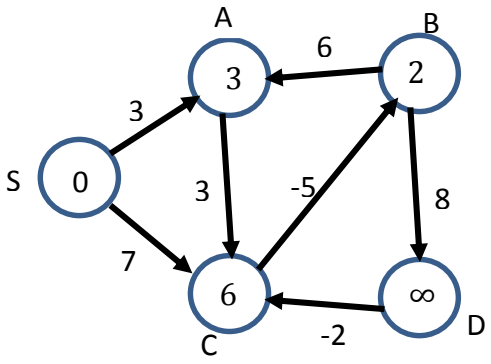


Solution:

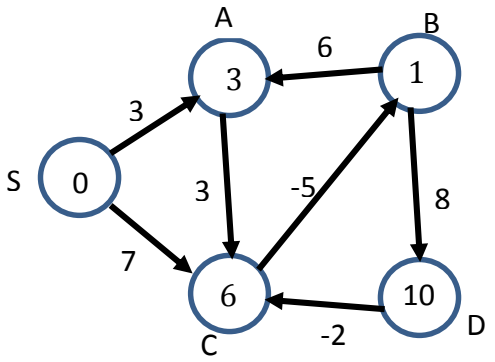


Step1: initialize

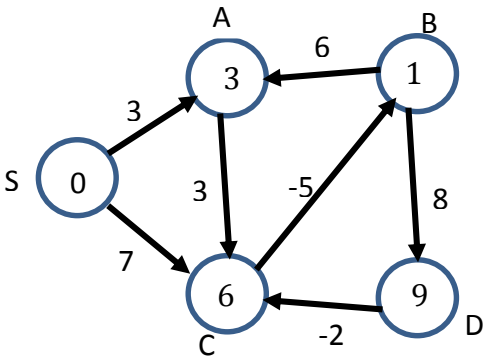
Step2: 1st loop



Step3: 2nd loop



Step4: 3rd loop



Step5: 4th loop

Step6: check there do not have any d(v) > d(u)+w(u,v) in step5.

6. Run the Floyd-Warshall algorithm on the directed graph of Figure 1 to find all-pairs shortest-paths. You need to detail each computational step.

Solution:

$$\begin{array}{c} \\ D^{(0)} = \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{ccccc} S & A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 3 & \infty & 7 & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & 6 & 0 & \infty & 8 \\ \infty & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & -2 & 0 \end{pmatrix} \end{array} \quad \begin{array}{c} \\ D^{(1)} = \\ \\ \\ \\ \\ \end{array}\begin{pmatrix} 0 & 3 & \infty & 7 & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & 6 & 0 & \infty & 8 \\ \infty & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & -2 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & \infty & 6 & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & 6 & 0 & 9 & 8 \\ \infty & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & -2 & 0 \end{pmatrix} \quad D^{(3)} = \begin{pmatrix} 0 & 3 & \infty & 6 & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & 6 & 0 & 9 & 8 \\ \infty & 1 & -5 & 0 & 3 \\ \infty & \infty & \infty & -2 & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & 1 & 6 & 9 \\ \infty & 0 & -2 & 3 & 6 \\ \infty & 6 & 0 & 9 & 8 \\ \infty & 1 & -5 & 0 & 3 \\ \infty & -1 & -7 & -2 & 0 \end{pmatrix} \quad D^{(5)} = \begin{pmatrix} 0 & 3 & 1 & 6 & 9 \\ \infty & 0 & -2 & 3 & 6 \\ \infty & 6 & 0 & 6 & 8 \\ \infty & 1 & -5 & 0 & 3 \\ \infty & -1 & -7 & -2 & 0 \end{pmatrix}$$

7. (Chapter 34) Please prove that the Traveling-salesman problem (TSP) is NP-complete. (Hint: Transform the Hamiltonian-cycle problem to the TSP.)

Solution:

1. We can verify the answer of TSP in O(|V|) => TSP ∈ NP
2. Show that HAM-CYCLE ≤p TSP

Let $G = (V, E)$ be an instance of HAM-CYCLE. We construct an instance of TSP as follows. We form the complete graph $G' = (V, E')$, where $E' = \{\{i , j\},: i , j, \in V,$ and $i \neq j\}$ and we define the cost function.

$$c(i,j) = \begin{cases} 0 \; if \; (i,j) \in E \\ 1 \; if \; (i,j) \notin E \end{cases}$$

We can show that graph G has a Hamiltonian cycle iff graph G' has a tour cost at most 0.

8. (35.1 the vertex-cover problem)

Given a graph $G = (V, E)$, we want to select the minimum number of vertices such that the two vertices of each edge has at least one vertex selected. Describe a polynomial-time approximation algorithm with approximation ratio **two** for the problem. What is the time complexity? Prove that your algorithm has a ratio bound two.

Solution:

APPROX_VERTEX_COVER (G)

1 $C = \emptyset$

2 $E' = G.E$

3 **while** $E' \neq \emptyset$

4 let (u, v) be an arbitrary edge of E'

5 $C = C \cup \{u, v\}$

6 remove from E' every edge incident on either u or v

7 **return** C

The running time of this algorithm is $O(V+E)$.

APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.

Proof

The set C of vertices that is returned by APPROX-VERTEX-COVER is a vertex cover, since the algorithm loops until every edge in $G.E$ has been covered by some vertex in C.

To see that APPROX-VERTEX-COVER returns a vertex cover that is at most twice the size of an optimal cover, let A denote the set of edges that line 4 of APPROX-VERTEX-COVER picked. In order to cover the edges in A, any vertex cover—in particular, an optimal cover C^*—must include at least one endpoint of each edge in A. No two edges in A share an endpoint, since once an edge is picked in line 4, all other edges that are incident on its endpoints are deleted from E' in line 6. Thus, no two edges in A are covered by the same vertex from C^*, and we have the lower bound

$$|C^*| \geq |A|$$

on the size of an optimal vertex cover. Each execution of line 4 picks an edge for which neither of its endpoints is already in C, yielding an upper bound (an exact upper bound, in fact) on the size of the vertex cover returned:

$$|C| = 2|A|$$

Combining the two equations above, we obtain

$$|C| = 2|A| \leq 2|C^*|$$

9. Please answer the following question.

a. The solution to the recurrence $T(n) = 4\mathrm{T}\left(\frac{n}{2}\right) + n$ is $T(n)$ = ____

b. The time complexity of solving the strongly connected component problem in a directed graph $G = (V, E)$ is ____

c. The time complexity of using Dijkstra's algorithm to solve one-to-all shortest path problem in a weighted graph $G = (V,E)$ with the Fibonacci heap implementation is ____

d. The time complexity of using Johnson algorithm to solve all-to-all shortest path problem in a directed graph $G = (V, E)$ is ____

Solution:

a. n^2

b. $O(|V| + |E|)$

c. $O(E + V\log V)$

d. $O(V^2 \log V + VE)$

10. For each of the following statements, determine whether it is true or false. If the statement is correct, briefly state why. If the statement is wrong, explain why. Tour answer will be evaluated based on your explanation and not the True/False marking alone.

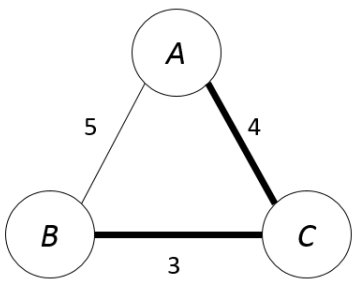
Solution:

(1) False. You also need to show that problem *A* belongs to NP.

(2) False. The halting problem cannot be solved by any computer no matter how much time allowed.

(3) False. It is unlikely we can find a polynomial-time algorithm to solve the NP-complete problem in worst-case.

(4) False. For example, in the following graph, MST is edge (*A*, *C*) and (*B*, *C*). The path from *A* to *B* in *T* has weight 7. However, the shortest path from *A* to *B* has weight 5.



(5) False. W may be exponential function of n, then we need exponential time to solve it. Actually, the 0-1 knapsack problem is NP-complete.
(6) True. With amortized cost analysis, the total amortized cost need to be larger than total actual cost.