
Problem Set 6 Solutions

Exercise 6-1. Do exercise 23.1-5 on page 566 of CLRS. Assume that e is part of a minimum spanning tree G . Consider the cut $(S, V - S)$ formed by the two subtrees of G when e is removed. Consider a light edge e' crossing the cut. Since e is a maximum-weight edge, $w(e') \leq w(e)$. Replacing e with e' will give us another spanning tree with weight no more than the original minimum spanning tree.

Exercise 6-2. Do exercise 24.1-3 on page 591 of CLRS.

Solution:

At the start of each pass, initialize a boolean variable *AnyValueChanged* to false. If the value of $d[v]$ changed during any relaxation steps in the pass, set *AnyValueChanged* to true. At the end of the pass, check the variable and terminate the algorithm if the variable has value false. This algorithm has the same outcome as the Bellman-Ford algorithm as no further change to the value of $d[v]$ will occur in any future passes.

Exercise 6-3. Do exercise 24.2-4 on page 595 of CLRS.

Solution:

Let the number of vertices be n . The vertices are first topologically sorted into an array $A[1..n]$ using depth-first search in time $O(V + E)$. For each vertex v , we associate a field $sum[v]$ to store the number of paths from earlier nodes to v .

The equation for $sum[v]$ in terms of the edges in E and the values of $sum[u]$ for $u \in V$.

$$sum[v] = \sum_{(u,v) \in E} (sum[u] + 1)$$

Initialize each entry $sum[v]$ to zero.

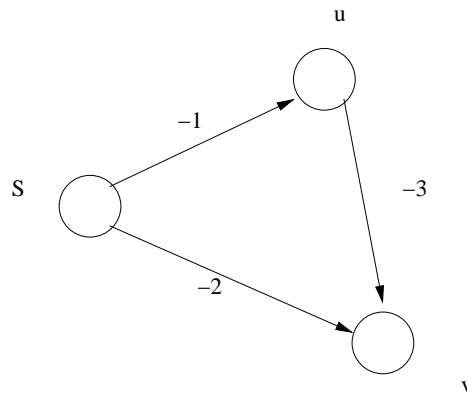
```
i ← 0
while i < j
  do u ← A[i]
    for each vertex v ∈ Adj[u]
      do sum[v] = sum[v] + sum[u] + 1
```

After running the code, the number of paths in the DAG can be found by summing up $sum[v]$ for all vertices. The running time for the code fragment is $O(V + E)$, hence the total running time is $O(V + E)$.

Exercise 6-4. Do exercise 24.3-2 on page 600 of CLRS.

Solution:

Consider Dijkstra's algorithm run from source s in the following graph. The vertex v will be removed from the queue with $d[v] = -2$ even though the shortest path to it is -4 .



Dijkstra's algorithm assumes that if v' is a node on the shortest path between u and v , then the length of the path from u to v' is shorter than the length of the path from u to v , which is not true when negative weights are allowed.

Exercise 6-5. Do exercise 25.2-6 on page 635 of CLRS.

Solution:

Check whether the shortest path from any node to itself is negative after running the Floyd-Warshall algorithm.

Exercise 6-6. Do exercise 25.3-4 on page 640 of CLRS.

Solution:

The lengths of paths are modified by this transformation. In particular, it is increased by $|w^*| \times \text{pathlength}$, hence may change the shortest path from one vertex to another.

Problem 6-1. Nesting boxes

- (a) We want to show that when A can be nested within B and B can be nested within C , then A can be nested within C . Let π_A be the permutation that allows A to be nested within B and π_B be the permutation that allows B to be nested within C . To nest A within C , simply apply π_A to A and then π_B on the result to get a permutation that allows nesting of A within C . After the first application, we know that every dimension of A is less than the corresponding dimension in B . Hence, these dimensions will also be less than the corresponding dimension in C after the second permutation.

- (b) Sort the dimensions for both boxes. Box A can nest within box B if and only if every dimension in the sorted list for A is smaller than the corresponding dimension in the sorted list for B . The running time for doing this using merge sort is $O(d \lg d)$. To see that this works, assume that the dimensions of B is sorted and another permutation of A where the smallest dimension is not the first dimension satisfies the nesting requirement. We can swap the first dimension of the permutation with the smallest dimension and still satisfy the nesting requirement. This process can be repeated for each dimension until the dimensions of A is sorted. Conversely, if the sorted permutation of A fails to satisfy the requirement for each dimension, no other permutation can satisfy the requirement. To see this, let u be the first sorted dimension in A that fails to be smaller than the corresponding dimension in B . There is no permutation that maps any element larger than u to $\{1, \dots, u\}$ that can satisfy the nesting requirement. At the same time u cannot be placed in any position in $\{1, \dots, u\}$ and still satisfy the nesting requirement. Hence if A nests in B if and only if the sorted dimensions of A are smaller than the corresponding sorted dimensions of B .
- (c) Assign a vertex to every box. For every pair of boxes A and B , join vertex A to vertex B with a directed edge with weight 1 if A nests in B and vice versa. Create a new source vertex S and join S to every vertex with a directed edge with weight 0. Create a destination vertex D and join every vertex to it with a directed edge of weight 0. This gives a directed acyclic graph. To see that the graph is acyclic, note that a box cannot be nested within itself. Assume that there is a cycle. The transitive property of the relation means that if there is a cycle in the graph a box can be nested within itself, giving a contradiction. The longest path from S to D in the DAG gives a longest sequence of boxes. This longest path can be found by topologically sorting the graph and running Bellman Ford algorithm in time $O(V + E)$. Since there are n vertices and $O(n^2)$ edges, the running time for this part is $O(n^2)$. The running time for constructing the graph is $O(dn^2)$ and for sorting the dimensions is $O(nd \lg d)$. Hence the total running time is $O(dn^2)$.