Question 1

Suppose that instead of selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

The algorithm is sorted the activities by the finish time in descent. Then choose activities from the beginning successively guaranteeing that the latter chosen activities is compatible with all previously selected activities.

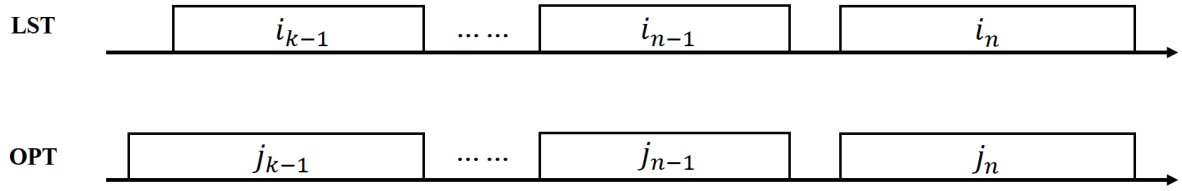


Figure 1.1 A way of Optimal choice Vs the way of last activity to start

Proof: Let $i_1, i_2, ..., i_n$ be the way of selecting the last activity to start(LST). Let $j_1, j_2, ..., j_n$ be the optimal way(OPT). We bring each activity into alignment by finished time. Suppose that the most and last $n-k+1$ activities are the same as between LST and OPT. As shown in Fig 1.1, we can set that the $(k-1)$-th activity is the first different activities which has different finish time. According to the strategy of greedy algorithm, we can replace $(j-1)$-th activity by another activity $(i-1)$ which is no worse than $(i-1)$-th activity. That contradicts to the suppose (that the most number of the last same activities between LST and OPT is $n-k+1$). So Selecting the last activity to start that is compatible with all previously selected activities is an optimal solution.

Question 2

Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

Because only those edges that leave the source vertex s may have negative weights, so the triangle inequality in the proof of traditional Dijkstra's algorithm is still correct. As

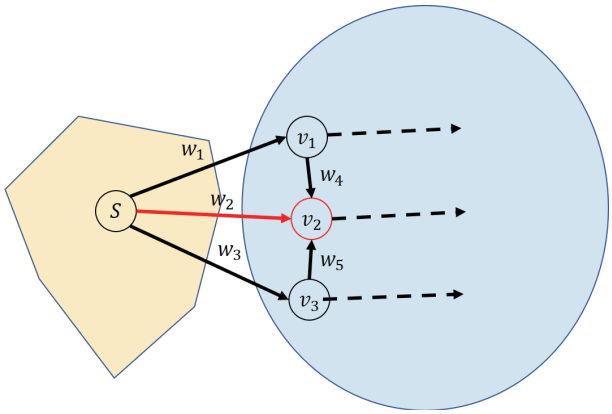


Figure 2.1 Only edges that leave the source vertex s may have negative weights

shown in Fig 2.1, suppose v_2 is the first chosen vertex, then we have $w_2 \leq w_1$ and $w_2 \leq w_3$. No matter is w_2 positive or negative, we always have $w_1 + w_4 \geq w_2 + w_4 \geq w_2$(because w_4 is positive). Therefore, Dijkstra's algorithm correctly finds shortest paths from s in this graph.

Question 3

Most graph algorithms that take an adjacency-matrix representation as input require time $\Omega(V^2)$, but there are some exceptions. Show how to determine whether a directed graph G contains *universal sink* —— a vertex with in-degree$|V| - 1$ and out-degree 0 —— in time $O(V)$, given an adjacency matrix for G.

Firstly, we define that if there is an direct edge from v to s, we say that s is adjacent to v. Let A denote the adjacency-matrix. We can divide the vertexes into groups each of which has 2 vertex. We visit each group(containing v_i and v_j) and judge the value $A[v_i][v_j]$. If it is 1, then we delete v_i, else we delete v_j. We recursively execute searching and deleting in the remaining vertexes until there is only one vertex left. Finally, we check whether each of the other vertexes is adjacent to v_k. If so, we say we find the *universal sink*, else do not.

We now analyze the time complex of this algorithm. Let n be the quantity of vertexes, we have $T(n) = T(n/2) + n/2$. So the costs of searching and deleting process is $O(n)$. The last step costs $O(n)$. So it costs $O(n)$ time to find the result.

Question 4

Let us say that a graph $G = (V, E)$ is *near-tree* if it is connected and has at most $n + 8$ edges, where $n = |V|$. Give an algorithm with running time $O(n)$ that takes a near-tree G with costs on its edges, and returns minimum spanning tree of G. You may assume that all the edge costs are distinct.

We can use Kruskal algorithm to handle this problem. Because the the graph has most $n+8$ edges, we can use count sorting to sort the edges which costs $O(n)$. We visit each edge and test whether the current edge's adding will construct a circle. If not, add it to the result. We can use two node n_l and n_r, which are pointed by two parts divided by the first vertex, respectively. In this way, we can test whether an edge will construct a circle in $O(1)$ time. Because the graph G has most $n + 8$ edges, the cost of constructing minimum spanning tree of G is $O(n)$.

Question 5

Consider the Minimum Spanning Tree Problem on an undirected graph $G = (V, E)$, with a cost $c_e \leq 0$ on each edge, where the costs may not all be different. If the costs are not all distinct, there can in general be many distinct minimum-costs solutions. Suppose we are given a spanning tree $T \subseteq E$ with the guarantee that for every $e \in T$, e belongs to *some* minimum-cost spanning tree in G. Can we conclude that T itself must be a minimum-cost spanning tree in G? Give a proof or a counterexample with explanation.

T may not be a minimum-cost spanning tree in G.

Let us consider the following instance shown in Fig 5.1. Two of its minimum spanning

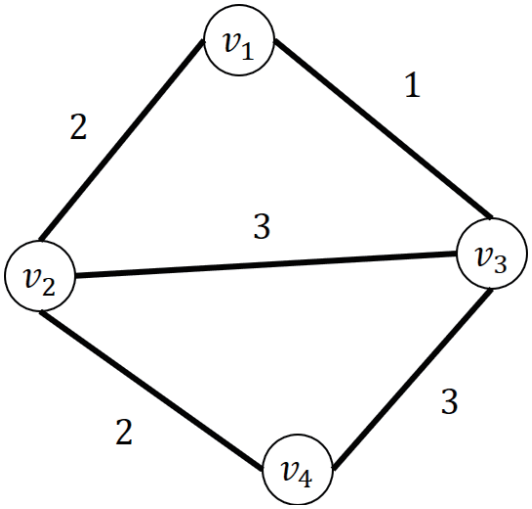
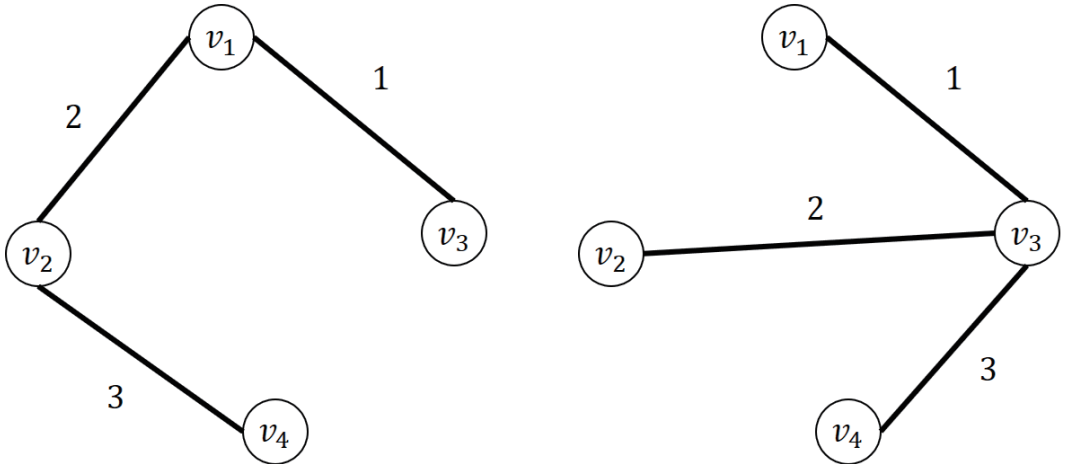


Figure 5.1 An counterexample

trees are shown in Fig 5.2. Then we illustrate an example shown in Fig 5.3. It is not a minimum-costs spanning tree.



(a) minimum spanning tree (b) minimum spanning tree

Figure 5.2 minimum spanning tree. (a)minimum spanning tree 1. (b)minimum spanning tree 2.

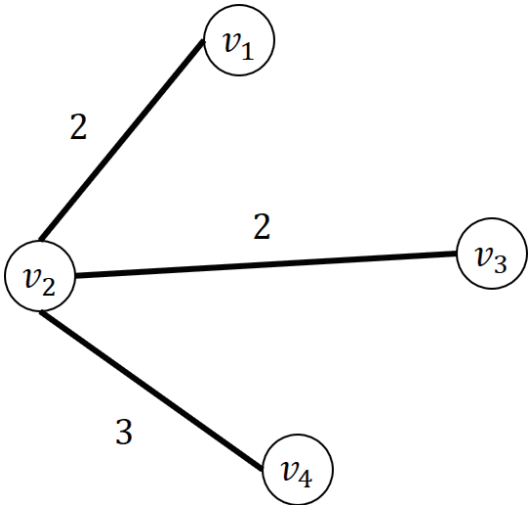


Figure 5.3 An counterexample