Algorithm Design Assignment 7

Jianan Lin, 662024667, linj21@rpi.edu

Problem 7.29. We want to move some software from $1 \sim n$ from the old system to the new system. If we move software i, then we will benefit b_i. If software i and j are in different systems, then we have to pay x_{ij} for them. Notice, software 1 cannot move. Design a polynomial algorithm to find the best way to move software with largest "benefit minus payment".

Solution.

We divide our algorithm into two parts.

Part 1. We construct a directed graph with source node s and each software v_i. We use v_1 as the sink node. For $\forall i \in [1, n]$, we have directed edge (s, v_i) with capacity b_i. For $\forall i \neq j, i, j \in [1, n]$, we have directed edges $(v_i, v_j), (v_j, v_i)$ with capacity x_{ij} respectively. The time complexity of this part is $O(n^2)$ because there are $n + 1$ nodes and the number of edges between two nodes is at most 2. Here, we require that all b_i and x_{ij} are integers. If not, we have to multiply some constants to achieve this.

The reason why we construct such a graph is that, consider a cut (S, T) in graph. Here, $s \in S, v_1 \in T$. Use B to denote $\sum_i b_i$ (including $i = 1$) and X to denote $\sum_{i \neq j} x_{ij}$. Then we know the capacity of this cut is

$$C(S,T) = \sum_{i \in T} b_i + \sum_{i \in S, j \in T} x_{ij} = B - \left(\sum_{i \in S} b_i - \sum_{i \in S, j \in T} x_{ij} \right) = B - A$$

Here we can find that, if we regard S as the new system and T as the old system, then A is the result with "benefit minus payment" (we call it pure benefit). Since we want to make A as large as possible, we have to make $C(S, T)$ as small as possible, i.e. to find a cut with minimum capacity.

Part 2. We use Ford-Fulkerson algorithm to find the maximum $s - v_1$ flow. Then according to theorem 7.11 in textbook, we only need to spend $O(E) = O(n^2)$ time to find the cut with minimum capacity. Here, E is the number of edges. The Ford-Fulkerson algorithm takes $O(Ef)$ to finish, and f is the max flow. We know $f \leq B + X$ although this bound may not be tight. So we know the time complexity of this part is $O(Ef + E) = O(Ef) = O(n^2(B + X))$.

After we find this cut (S^*, T^*), we naturally know that, all the software nodes in S^* should be moved to the new system. Therefore, the total time complexity is $O(n^2(B+X)) + O(n^2) = O(n^2(B+X))$.

Problem 8.9. Prove that Path-Selection problem is NP-Complete. Here, Path-Selection problem means: given a directed graph $G = (V, E)$ and c paths $P_1, ..., P_c$. Can we select at least k paths from them so that no two of the selected paths share any nodes?

Solution.

Obviously, this problem is in NP because given some paths, we can check whether they share some node in polynomial time. We use independent set problem: Given a graph, can we find a set of k nodes so that no two nodes are connected? This is a famous NPC problem. If we can reduce independent set problem in polynomial time to path selection problem, then we prove that path selection problem is also NPC because it is at least as hard as independent set problem.

Suppose there is an arbitrary graph for independent set problem called G_0. We construct a graph G in path selection problem according to G_0. For each edge in G_0, we add a node in G. For each vertex in G_0, we add a source node, a sink node and a path from the source to the sink in G. The specific operation is as follows:

Considering vertex v in G_0, if there are k edges connected to it, then we decide an arbitrary order of these edges. Then we link the edges in G following this order. This forms a path from the source node s_v to the sink node t_v which goes through all the corresponded vertices in G.

We consider the two graphs and can find that a vertex set $\{v_1, v_2, ..., v_k\}$ is an independent set, if and only if the corresponding paths $p_1, ..., p_k$ do not share any vertex. This reduction is obviously in polynomial time, so we know we can reduce any independent set problem to path selection problem in polynomial time. Therefore we prove that path selection problem is also NPC.

To understand the process better, there is a graph below. The left is the G_0 in the independent set problem (with 4 nodes and 5 edges) and the right is G in the path selection problem (with 4 + 5 + 4 nodes and 4 paths). It is easy to see that, any set of nodes in G_0 is an independent set if and only if the corresponding paths in G do not share any node.

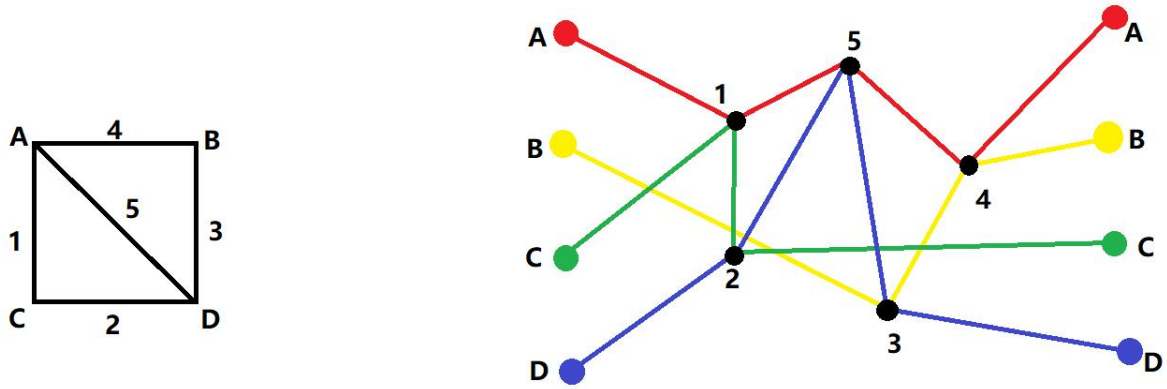

Figure 1: An example. 4 colors are 4 paths. All the paths go from left to right.