Algorithm HW#3

B02901178 江誠敏

January 3, 2017

Collaborators: None

1 Problem 1.

We shall define our vertices set as $V = \{(x_1, x_2, \cdots, x_m) : 0 \leq x_i \leq c_i, \forall i\}$. Which $(x_1, x_2, \cdots, x_m)$ represent the state such that the i-th cup has x_i mls of water. Now the three types of operations correspond to 3 types of edge.

1. Completely fill one cup: For each $1 \leq i \leq m$ and a vertex $(x_1, x_2, \cdots, x_m)$. If $x_i < a_i$, there is an edge from $(x_1, \cdots, x_i, \cdots, x_m)$ to $(x_1, \cdots, a_i, \cdots, x_m)$ with cost $a_i - x_i$ (since we are minimizing the total water used). Let these edges be E_1.
2. Empty one cup: For each $1 \leq i \leq m$ and a vertex $(x_1, x_2, \cdots, x_m)$. If $x_i > 0$, there is an edge from $(x_1, \cdots, x_i, \cdots, x_m)$ to $(x_1, \cdots, 0, \cdots, x_m)$ with cost 0. Let these edges be E_2.
3. Pour water from cup c_i to cup c_j until either c_i is empty or c_j is full: For each $1 \leq i, j \leq m, i \neq j$ and a vertex $(x_1, x_2, \cdots, x_m)$. If $x_i > 0$ and $x_j < a_j$, let $y = \min(x_i, a_j - x_j)$, then there is an edge from $(x_1, \cdots, x_i, \cdots, x_j, \cdots, x_m)$ to $(x_1, \cdots, x_i - y, \cdots, x_j + y, \cdots, x_m)$ with cost 0. Let these edges be E_3.

Then let $E = E_1 \cup E_2 \cup E_3$. Now the problem become "Finding the shortest path from $s = (0, 0, \cdots, 0)$ to $t = (c_1, 0, \cdots, 0)$". The number of vertex is $(a_1+1)(a_2+1)\ldots(a_m+1) = n$. And notice that if $a_i \geq 1, \forall i$ (If some $a_i = 0$, we could just ignore those cups.), Then $\prod(a_i + 1) \geq 2^m \geq m$.

Now, for a vertex $v = (x_1, x_2, \cdots, x_m)$, there are at most m edges in E_1, at most m edges in E_2, at most m^2 edges in E_3 adjacent to v, so $\mathcal{O}(E) \leq \mathcal{O}(m^2 n) \leq \mathcal{O}(n^3)$.

Hence by using shortest path algorithm, such as Dijkstra's algorithm, the problem could be solve in $\mathcal{O}(E + V \log V) = \mathcal{O}(n^3 + n \log n)$ time, which is polynomial in n.

2 Problem 2.

First we observe that if $(e_1, e_2, \cdots, e_k)$ are a path, and let p_i be the probability that e_i is cancelled. Then the probability that no flight is cancelled in the path is $\prod(1 - p_i)$. And

since $-\log(\cdot)$ is decreasing in $[0,1]$, so

$$\text{Maximize } \prod(1-p_i) \iff \text{Minimize } -\log\prod(1-p_i) = \sum -\log(1-p_i)$$

So if there is a flight from A_i to A_j with cancelling probability $p_{i,j}$, we could think it as there is an edge from A_i to A_j with cost $c_{i,j} \triangleq -\log(1-p_{i,j})$. The problem becomes "Finding the shortest path from A_1 to A_n". And notice that $1-p_{i,j} \leq 1 \implies c_{i,j} = -\log(1-p_{i,j}) \geq 0$. So there is no negative edges, hence using Dijkstra's algorithm, the problem could be solve in $\mathcal{O}(E + n\log n)$ where E is the number of flights.

3 Problem 3

Add an additional node C_0. We construct the (undirected) edges as follow.

1. for all $1 \leq i \leq n$. There is an edge (C_0, C_i) with cost equals p_i.
2. for all $1 \leq i, j \leq n, i \neq j$. There is an edge (C_i, C_j) with cost equals $w_{i,j}$.

Any configuration would correspond to an edge set E' by following:

1. If we construct a power station at city C_i, then the edge (C_0, C_i) is in E'.
2. If we construct an undirected power line between C_i, C_j, the edge (C_i, C_j) is in E'.

Now, if C_i has a power station, then $(C_0, C_i) \in E'$ so C_i is connected to C_0 in E'. And if there is a power supply path (a series of power lines) that connect C_i to another city C_j with a power station, then C_i is connected to C_j and C_j is connected to C_0, hence C_i is connected to C_0 in E'. Thus the contrain is satisfied if and only if every node C_i is connected to V_0, which is equivalent that every node is connected to each other.

But the odd thing is that the statement says that some costs are negative, so finding a minimum spanning tree might not be the best solution. (Imagine that all p_i and $w_{i,j}$ are negative. Then obviously build anything would be the best solution.) What we shall find is not a minimum spanning tree, but a "Minimum spanning graph", which is a subgraph $G' = (V, E')$ in G such that each $v \in V$ is in G' and are connected in G'.

So we modify the Kruskal's algorithm as following. It is easy to see that the algorithm is correct. Since G' need not to be a tree anymore, there is no lost that we choose all the negative edge to be in E'. After choosing these edge, vertices may already form some connected components. Seeing these components as new "vertices", since if all edge is positive the minimum spanning tree is always one of the minimum spanning graph, we could applying the normal Kruskal's algorithm and find the answer.

Algorithm 1: Modified Kruskal's algorithm

input : A graph $G = (V, E)$

output: Its minimum spanning graph $G' = (V, E')$

$(E^-, E^+) \leftarrow \big(\{e \in E : e \text{ has cost } < 0\}, \{e \in E : e \text{ has cost } \geq 0\}\big)$
$E' \leftarrow \varnothing$
foreach $e = (u, v) \in E^-$ *sorted by cost increasingly* **do**
    $E' = E' \cup \{(u, v)\}$
    `Union(`u`,` v`)`
end
foreach $e = (u, v) \in E^+$ *sorted by cost increasingly* **do**
    if `Find(`u`)` $\neq$ `Find(`v`)` **then**
        $E' = E' \cup \{(u, v)\}$
        `Union(`u`,` v`)`
    end
end
return $G' = (V, E')$

The running time is same as Kruskal's algorithm, $\mathcal{O}(E \log E) = \mathcal{O}(E \log V)$.

4 Problem 4

Let w_i be the node represent each TA, so $V = \{C_i\} \cup \{w_i\} \cup \{s, t\} \triangleq C \cup W \cup \{s, t\}$. Where s, t is the source and sink node which will be used in constructing flow graph.

Now construct (directed) edges as following:

- For all $w_i \in W$, $(s, w_i) \in E$ which has capacity 2.
- For all $c_i \in C$, if $w_j \in S_i$, then $(w_i, C_i) \in E$ and has capacity 1.
- For all $c_i \in C$, $(C_i, t) \in E$ and has capacity n_i.

Let $N = \sum n_i$. We shall proof that a flow with capacity N correspond to a valid solution and vise versa. But we need the following lemma.

Lemma 1. *If f is an s-t flow such that each edge has integer flow. Then f could be decomposed to unit flow $f = f_1 + f_2 + \cdots + f_k$. That is, f_i send 1 unit from s to t.*

Proof. Start from s, find an edge (s, u_1) which $f(s, u_1) > 0$. Since each vertex v except s, t has $f^+(v) = f^-(v)$, we could find u_2 so that $f(u_1, u_2) > 0$. Continue the procedure and eventually we would reach t. That is, we would find a path $s, u_1, u_2, \cdots, t$ such that each edge in the path has positive flow. Since each edge has integer flow, these edge has flow greater than 1. So let f_1 be the flow which has 1 unit of flow on these edge and let

$f' = f - f_1$, then f' is still an integer flow. By induction we decompose f to unit flows $f_1 + f_2 + \cdots + f_k$. □

By lemma. If f is a flow with size N, decomposed f to unit flow $f = f_1 + f_2 + \cdots + f_N$. We know that each flow has the form $s \to w_i \to C_j \to t$. We shall let this flow represent that "Letting w_i to be one of the TA of course C_j". Then we check that:

- Each w_i would only be TA of at most 2 courses, since $s \to w_i$ has capacity 2, so there are at most 2 f_i which start with $s \to w_i \cdots$.
- Each pair (w_i, C_j) would only be in one f_i since $w_i \to C_j$ has capacity 1. So there would not be a strange scenario that a TA is counted at two of the TAs of a course C_j.
- Since f has size N, and the sum of capacity of all edges into t is exactly N. So the flow in these edge has to be equal to their capacity, n_i. Hence each course would have exactly n_i TAs.

Again, since the sum of capacity of all edges into t is exactly N, if f is a flow of size N, it is surely a maximum flow. Hence we turn the problem into "Checking if the maximum flow of the graph is N". Using any flow algorithm (except Ford-Fulkerson), such as Edmonds-Karp or Dinic's, the problem could be solved in polynomial time of V, E. For example, if Dinic's algorithm is used, the algorithm runs in $\mathcal{O}(VE \log V) \leq \mathcal{O}(V^3 \log V)$.

5 Problem 5

We shall proof that "Set covering" could be reduced to this problem, and hence this problem ("Steiner tree in graph") is NP-complete (Assume that we already know this problem is in NP.)

Recall that the set covering is state as following: Given $U = \{1, 2, \cdots, n\}$, $S = \{S_1, \cdots, S_m\}$ with $S_i \subseteq U$, and k, is there a subgroup T of S with size k such that $\bigcup_{S_i \in T} S_i = U$?

Now construct a graph $G = (V, E)$. $V = X \cup Y \cup \{z\}$. Where

- $X \triangleq \{x_1, x_2, \cdots, x_n\}$, correspond to each element in U.
- $Y \triangleq \{y_1, y_2, \cdots, y_m\}$, correspond to each set in S.

And the (undirected) edges is construct as following:

- For all $x_i \in X, y_j \in Y$, $(x_i, y_j) \in E \iff i \in S_j$. These edges has cost 0. Denote these edges by a set E_0.
- For all $y_j \in Y$, $(y_j, z) \in E$ and has cost 1. Denote this edge by e_j.

Let the steiner vertices V' as in the statement to be $V' \triangleq X \cup \{z\}$. Then, if $T = \{S_{i_1}, S_{i_2}, \cdots, S_{i_h}\}$ is a set cover of size h, then choose $E' = \{e_{i_t} : 1 \leq t \leq h\} \cup E_0$, which

has cost h. Every vertex in V' is connected to z, since for all i, $i \in S_\alpha$ for some $S_\alpha \in T$. Then $x_i \xrightarrow{0} y_\alpha \xrightarrow{e_\alpha} z$ is a path by the fact that $e_\alpha \in E'$. So every vertex in V' is connected to z and thus they are connected.

Conversely, if E' is a solution to the problem with cost h, let $T = \{S_i : e_i \in E'\}$. Then $|T| = h$ since $|E' \cap \{e_i\}| = h$. Then since E' connects all V', so for each i, there is a path from x_i to z and we know that the path has the form $x_i \to y_j \xrightarrow{e_j} z$, so a $e_j \in E'$, and then $i \in S_j$ and $S_j \in T$. Thus S_j is a set cover with size h.

Hence there is a set cover with size no larger than h if and only if there is a solution with cost no larger that h of this problem about this certain graph. Hence this problem is harder than set cover, which is an NP-complete problem. Thus the problem is in NP-complete.

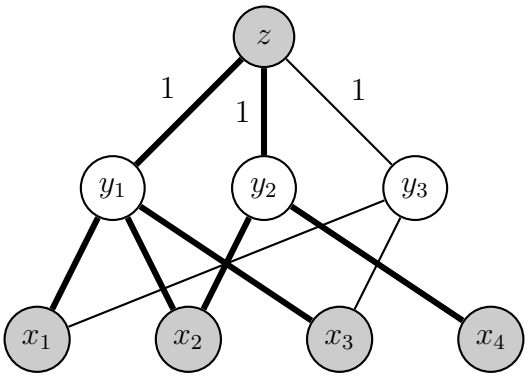

Figure 1: The steiner graph corresponding to the set cover problem
$S = \{\{1, 2, 3\}, \{2, 4\}, \{1, 3\}\}$.