ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2024

Problem Set 3

Zhou Shouchen

StudentID: 2021533042

Due: 23:59, April 30, 2024

1. Submit your solutions to the course Gradescope.
2. If you want to submit a handwritten version, scan it clearly.
3. Late homeworks submitted within 24 hours of the due date will be marked down 25%. Homeworks submitted more than 24 hours after the due date will not be accepted unless there is a valid reason, such as a medical or family emergency.
4. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious penalties.

Problem 1:

Given a network $G(V, E, s, t)$, give a polynomial time algorithm to determine whether G has a unique minimum $s - t$ cut.

Solution:

Since we have known that the minimum $s-t$ cut of the graph G is the same as the maximum flow of the graph G, we can use the Ford-Fulkerson algorithm to get the minimal $s - t$ cut of the graph G.

We can get the minimal $s-t$ cut of the graph G in polynomial time such as using the Ford-Fulkerson algorithm with capacity scaling in $O(|E|^2 \log C)$ time, where C is the maximum capacity of the edges in the graph.

Suppose that after running the Ford-Fulkerson algorithm, we get the minimal $s - t$ cut of the graph G is $e_1, e_2, \cdots e_k$.

Then each time we can remove one edge e_i from the graph G, and then run the Ford-Fulkerson algorithm again to get the new minimal $s-t$ cut. If the new minimal $s - t$ cut is the same as the original minimal $s - t$ cut, then the graph G doesn't have a unique minimal $s - t$ cut.

After sequentially remove e_i each time, if all the new minimal $s - t$ cuts are different from the original minimal $s - t$ cut, then the graph G has a unique minimal $s - t$ cut.

So the algorithm is shown in 1.

Algorithm 1 Determine whether G has a unique minimum $s - t$ cut

```
1: Run the Ford-Fulkerson algorithm to get the minimal s − t cut of G.
2: Let e_1, e_2, ··· e_k be the edges in the minimal s − t cut.
3: for i = 1 to k do
4:     Remove the edge e_i from G and get G'.
5:     Run Ford-Fulkerson algorithm on G' to get the new minimal s − t cut.
6:     if The new minimal s − t cut == the original minimal s − t cut then
7:         return False
8:     end if
9: end for
10: return True
```

The total time complexity is $O(|E|^3 \log C)$, as there are at most $|E|$ edges in the minimal $s - t$ cut, and each time we need to run the Ford-Fulkerson algorithm in $O(|E|^2 \log C)$ time.

Problem 2:

We are given an $n \times m$ array of cells. Denote the cell in the r'th row and the c'th column as (r, c), where $1 \leq r \leq n$ and $1 \leq c \leq m$. A robot is at cell $(1, 1)$ and wants to reach cell (n, m). From cell (x, y), the robot can only move to $(x, y+1)$ or $(x+1, y)$. In addition, some cells contain impassable obstacles and the robot cannot move to these cells. Your goal is to put additional obstacles in the minimum number of cells (other than cell (n, m)) such that the robot cannot reach (n, m). Find an algorithm based on max flow.

Solution:

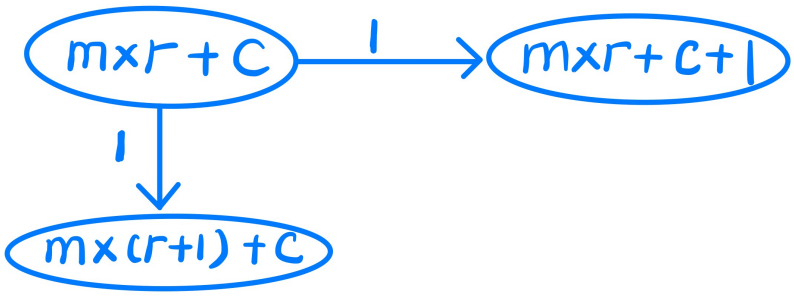

Figure 1: Way to contrust the graph

We can construct the graph as the Figure 1 shows.

The details are as follows:

For a cell whose location is (r, c), we can give it a number $m * r + c$ as the node's number.

Since from cell (r, c), we can reach $(r+1, c)$ or $(r, c+1)$ is there is no obstacles.

So we can construct the graph:

If there is no obstacle on (r, c) and $(r+1, c)$, then we set up an edge from node $m * r + c$ to node $m * (r+1) + c$, and the capacity is 1.

If there is no obstacle on (r, c) and $(r, c+1)$, then we set up an edge from node $m * r + c$ to node $m * r + c + 1$, and the capacity is 1.

After constructed the map, since we start from $(1, 1)$ to (n, m), so we can regard the node representing $(1, 1)$ to be the source node s, and the node representing (n, m) to be the sink node t.

Since we want to get the minimum number of obstacles to put, we can actually getting the minimum cut of the constructed graph.
For an edge (u, v), if it is in the minimum cut and $v \neq t$, then we can put an obstacle to the grid representing to the node v.
If (u, v) is in the minimum cut and $v = t$, then we can put an obstacle to the grid representing to the node u.
Then we can get the minimum cut to get the places to put the obstacles.

Since what we have learned that the minimum $s - t$ cut is the same as the maximum flow.
So we can use the Ford-Fulkerson algorithm to find the maximum flow of the constructed graph.

So the minimum number of obstacles we need to set is the maximum flow on the constructed graph.
And the methods to set up obstacles are mentioned above.
The time complexity is $O(n^2m^2)$.
Since there are total nm nodes, and there will have atmost 2 edges come out from each node. And all the edge's capacity is 1.

Problem 3:

Consider a restaurant with m different menu items. Each customer to the restaurant want to order one item, and each of them has a set of items they are willing to order from. The restaurant makes d_i amount of item i, for $1 \leq i \leq m$, so that at most d_i customers can order item i. Given n customers, where the j'th customer has a list C_j of items they are willing to order, design an efficient algorithm to maximize the number of customers you can serve.

Solution:

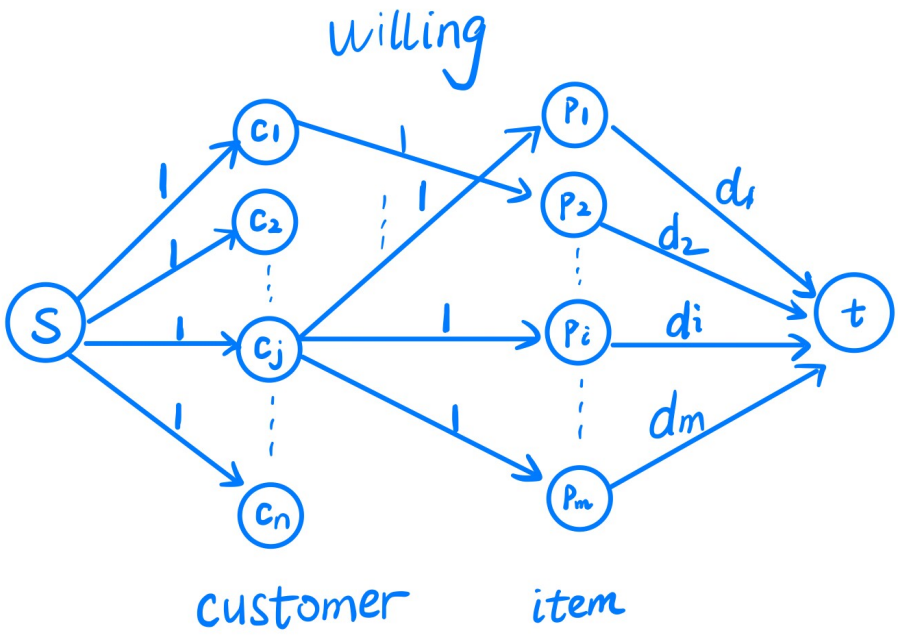

Figure 2: Way to contrust the graph

We can construct the graph as the Figure 2 shows.

The details are as follows:

Let node c_j represents the j-th customer, node p_i represents the i-th item. And s to be the source node, t to be the sink node.

Sice each customer will be satisfied if them get one item they want, so we can set up an edge from s to each of the customer c_j with capacity 1.

Since the j-th customer has a list C_j of items they are willing to order, so we can set up an edge from c_j to each of the item $p_j \in C_j$ with capacity 1.

Since the i-th item has been made in the number of d_i, so we can set up an edge from d_i to t with capacity d_i.

We can use the Ford-Fulkerson algorithm to find the maximum flow of the constructed graph.

The maximum number that the customers we can serve is the maximum flow on the constructed graph.
The time complexity is $O((nC+m)^2 \log D)$,
where $C = \max\limits_{j=1,2,\cdots,n} \{|C_j|\}, D = \max\limits_{i=1,2,\cdots,m} \{d_i\}$

Problem 4:

Consider an $n \times n$ grid containing n rows and n columns of vertices. Each vertex (i, j) has edges to four neighbors $(i-1, j)$, $(i+1, j)$, $(i, j-1)$, $(i, j+1)$, except for the boundary vertices, which have edges to two or three neighbors. Each vertex and edge also has a positive integer capacity. We are given m start vertices, and for each vertex we want to find a path which connects the vertex to an arbitrary vertex on the grid's boundary. Furthermore, we need to ensure that the number of paths which pass through each vertex or edge does not exceed its capacity. Give an algorithm to determine whether this is possible.

Hint: First transform the graph to convert the vertex capacities to edge capacities.

Solution:

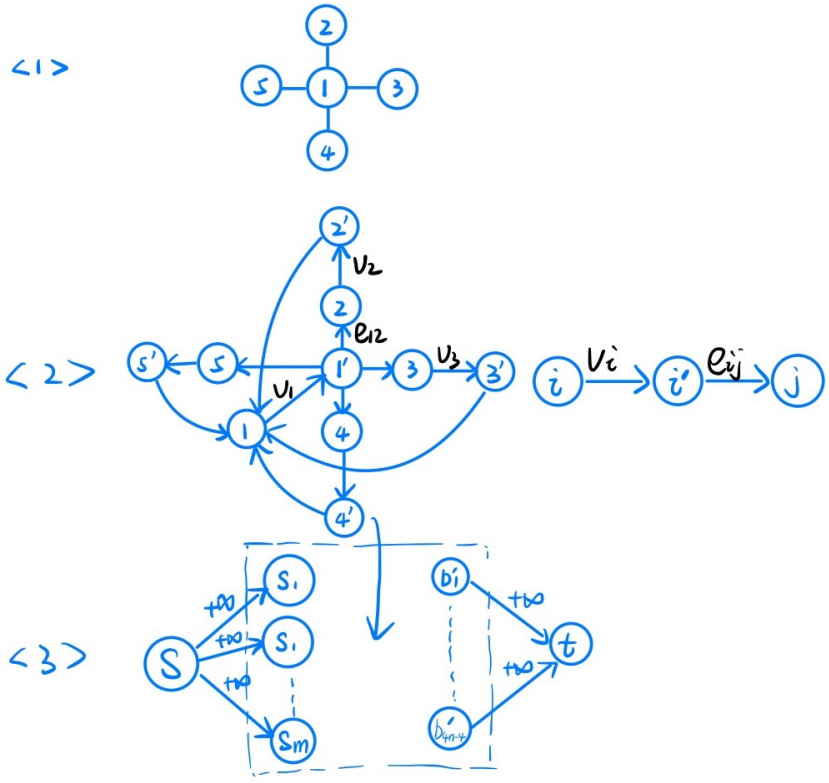

Figure 3: Way to contrust the graph

We can construct the graph as the Figure 3 shows.

The details are as follows:

Supoose each node is given a number to represent a grid. The neibor nodes (i, j)'s edge's capacity is $c_{i,j}$, the node i's capacity is v_i.

The start vertices are $s_1, s_2, \cdots, s_m$, the boundary nodes are $b_1, \cdots, b_{4n-4}$.

1. To make sure the limitations of vertex and edge's capacity, we need to split each node i into i and i', and we let i to recieve connections from other nodes, i' to give out connections to other nodes.

As shown in Figure 3's <2> part, we set edge from i to i' with capacity v_i.

If node i and node j have connections such as in <1>, then we set up edge from i' to j with capacity $e_{i,j}$, and similarly, an edge from j' to i with capacity $e_{i,j}$.

2. We set up edges from s to $s_i, i = 1, 2, \cdots, m$ with capacity $+\infty$.

3. We set up edges from the boundary nodes $b'_1, \cdots, b'_{4n-4}$ to t with capacities $+\infty$.

Then the graph is contrusted.

Let $C = \max\{\max\{c_{i,j}\}, D = \max\{v_i\}\}$

From our construction methods, we can find the sink node s is only sending flow to $s_1, \cdots, s_m$, and the boundary nodes b'_j recieves flow only from b_j.

So we can just set the $+\infty$ to be C.

With above construction, we can use the Ford-Fulkerson algorithm to find the maximum flow of the constructed graph.

If the maximum flow is the number of paths the graph can have which suits the requirements.

The time complexity is $O(n^2 \log C)$.

Since each grid is seperate into two nodes, but the nodes' edges are constant, so the number of the edges in the constructed graph is $O(n)$.

Problem 5:

Given a set C, a collection D of subsets of C, and a value k, the k-SET-PACKING problem asks if there exist k subsets from D which are mutually disjoint, i.e. no two of the subsets share an element. Show that the SET-PACKING problem is in NP.

Solution:

1. Verifier.

Certificate $D_1, D_2, \cdots, D_k$ are k subsets from D.
Check whether the k are mutually disjoint.
If so, output 1, otherwise, output 0.

2. If D is yes instance.

Then there exist a combination of k subsets from D which are mutually disjoint.
Let $D_1, \cdots, D_k$ be this assignment, and give it to V.
Clearly, V outputs 1.

3. If D is no instance.

Then there has no combination of k subsets from D which are mutually disjoint.
So no matter how we combine the k subsets from D, there will also exists unions.
So V outputs 0, for any input $D_1, \cdots, D_k$.

4. V runs in polynomial time.

If each subset D_i has atmost $|C|$ element, then to check each pair of sets D_i, D_j will have $|C|^2$ time.
And there are total $O(k^2)$ pairs.
So V runs in $O(k^2|C|)$ time, which is a polynomial time.

So above all, we have proved that the k-SET-PACKING problem is in NP.

Problem 6:

Consider the following problem about scheduling courses. The problem is defined by 3 sets C, S and R. Here, C represents a set of courses, S repreesents the available time slots for the courses, and R contains a collection of course requests from students, where each request is a subset of C representing the the courses that a particular student wishes to take. The goal is to schedule all the courses without any conflicts.

For example, if $C = \{a, b, c, d\}, S = \{1, 2, 3\}, R = \{\{a, b, c\}, \{a, b, d\}, \{b, d\}\}$, then it is possible to schedule the courses without conflicts by assigning course a to time slot 1, b to slot 2 and c, d to slot 3. However, if $C = \{a, b, c, d\}, S = \{1, 2, 3\}, R = \{\{a, b, c\}, \{a, c\}, \{b, c, d\}\}$, then it is not possible to schedule the courses without any conflicts. Prove that the problem of determining whether a conflict-free schedule exists is NP-complete.

Hint: Use a reduction from the 3-COLOR problem.

Solution:

Since we are given that $|S| = 3$, so we can use a reduction from the 3-COLOR problem.

1. Conflict-free schedule $\in$ NP problem.

For each given schedule, we can directly check whether each of the $|R|$ student's at most $|C|$ courses has the same time in the time slot.
This could be done in $O(|R||C|)$ time, which is a polynomial time.
So Conflict-free schedule $\in$ NP problem.

2. 3-COLOR $\leq_P$ Conflict-free schedule.

We need to prove that 3-coloring problem on graph G is a yes-instance of 3-coloring problem iff (C, S, R) is a yes-instance of the Conflict-free schedule problem:

<1>. "$\Rightarrow$":

Construction: for any instance of 3-coloring problem on graph G:
Let each node u to be a course, which constructs C.
Let each edge (u, v) to be a student whose requests is $\{u, v\}$, which constructs R.
Let each color we use becomes a slot, which constructs S.
Then if G is a yes-instance of 3-coloring problem, then according to our construction:

Since for each edge (u, v), u and v will be painted with different color, then for each student's requests will not be scheduled to the same slot, which means the Conflict-free schedule problem is also a yes-instance.

<2>. "$\Leftarrow$": Construction: for any instance of the Conflict-free schedule (C, S, R):
Let each course in C to be a node in graph G, which constructs $|V|$.
Let each pair of cources in R, suppose the two cources are u, v, then set (u, v) to be an edge in graph G, which constructs $|E|$.
Let each time in the time slot to a color, which constructs the color set.
Then if (C, S, R) is a yes-instance of the Conflict-free schedule problem, then their are no conflicts for a students' requests pair (u, v).
Then each edge's two nodes are in different colors.
Since $|S| = 3$, so we can painting the nodes in constructed graph G according to color set within 3 colors.
So it is also a yes-instance of 3-coloring problem.

So we have proved that 3-COLOR $\leq_P$ Conflict-free schedule.

Since 3-COLOR problem $\in$ NP-complete problem, Conflict-free schedule $\in$ NP problem.
3-COLOR $\leq_P$ Conflict-free schedule.
Therefore, Conflict-free schedule $\in$ NP-complete problem.

So above all, we have proved that Conflict-free schedule is a NP-complete problem.