CSCI570 Fall2023 HW4

Vinit Pitamber Motwani (USC ID: 8187180925)

October 2023

1. **(Dynamic Programming)** Given n balloons, indexed from 0 to $n-1$. Each balloon is painted with a number on it represented by array $nums$. You are asked to burst all the balloons. If you burst the balloon i you will get $nums[left] \cdot nums[i] \cdot nums[right]$ coins, where left and right are adjacent indices of i. After the bursting the balloon, the left and right then becomes adjacent. Assume, $nums[1] = nums[n] = 1$ and they are not real therefore you can not burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

   Solution :

   (a) Let $OPT(i, j)$ be the maximum number of coins you can obtain from balloons $i, i+1, ..., j-1, j$.

   (b) The key observation is that to obtain the optimal number of coins for balloon from i to j, we choose the balloon which is the last one to burst. Let balloon k is the last one you burst, then balloons from i to $k-1$ and balloons from $k+1$ to j must burst first. So,the following recurrence relation:
   $OPT(i, j) = max_{i \le k \le j}\{nums[i-1] * nums[k] * nums[j+1]$
   $+ OPT(i, k-1) + OPT(k+1, j)\}$

   (c) **Code:**

```
Let OPT[0..n][0..n] be a new 2D array
OPT[i][j] = 0 for all i < j (0 <= i, j < n)

for len = 1 to n do
    for i = 0 to n–len do
        j = i + len – 1
        for k = i to j do
            OPT[i][j]= max(OPT[i][j], nums[i–1]*nums[k]*nums[j+1]
            + OPT[i][k–1] + OPT[k+1][j])
        end for
    end for
end for
```

```
return OPT[0][n-1]
```

(d) i. Base case is: $OPT(i, j) = 0$ for all $i < j (0 \leq i, j < n)$
ii. Solution can be found at : $OPT(0, n-1)$

(e) For running time analysis, we in total have $\mathcal{O}(n^2)$ and computation of each state takes $\mathcal{O}(n)$ time. Therefore, the total time is $\mathcal{O}(n^3)$.

2. **(Dynamic Programming)** Suppose you are in Casino with your friend, and you are interested in playing a game against your friend by alternating turns. The game contains a row of n coins of values v_i, where n is even. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money you can definitely win if you move first. Analyze the running time of your algorithm.

Solution :

(a) Let $OPT(i, j)$ be the maximum number of amount obtained by player (who selects first) for set of coins i to j.

(b) Recurrence Relation:
$OPT(i, j) = max\{arr[i] + min\{OPT(i+2, j), OPT(i+1, j-1)\}$,
$arr[j] + min\{OPT(i, j-2), OPT(i+1, j-1)\}\}$;

(c) **Code:**

```
int maximumAmount(int arr[], int n){
    vector<vector<int>>OPT(n+1,vector<int>(n+1,0));
    for(int i=n-2;i>=0;i--){
        for(int j=i+1;j<n;j++){
            if(i>=j) continue;
            int x=arr[i]+min(OPT[i+2][j],OPT[i+1][j-1]);
            int y=arr[j]+min(OPT[i][j-2],OPT[i+1][j-1]);
            OPT[i][j]=max(x,y);
        }
    }
    return OPT[0][n-1];
}
```

(d) i. Base case is: $OPT(i, j) = 0$ for all $i \geq j$
ii. Solution can be found at : $OPT(0, n-1)$

(e) For running time analysis, we in total have $\mathcal{O}(n^2)$ because of two loops. Therefore, the total time is $\mathcal{O}(n^2)$.

3. **(Dynamic Programming)** Jack has gotten himself involved in a very dangerous game called the octopus game where he needs to pass a bridge which has some unreliable sections. The bridge consists of $3n$ tiles as shown below. Some tiles are strong and can withstand Jack's weight, but some tiles are weak and will break if Jack lands on them. Jack has no clue which tiles are strong or weak but we have been given that information in an array called **BadTile(3,n)** where **BadTile(j, i) = 1 if the tile is weak and 0 if the tile is strong**. At any step Jack can move either to the tile right in front of him (i.e. from tile (j, i) to $(j, i+1)$), or diagonally to the left or right (if they exist). (No sideways or backward moves are allowed and one cannot go from tile $(1, i)$ to $(3, i + 1)$ or from $(3, i)$ to $(1, i + 1)$). Using dynamic programming find out how many ways (if any) there are for Jack to pass this deadly bridge. Analyze the running time of your algorithm.

   Figure below shows bad tiles in gray and one of the possible ways for Jack to safely cross the bridge alive (See Fig. 1).

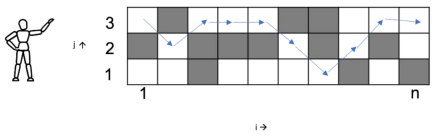


Figure 1

Solution :

(a) Let OPT(j, i) = The number of ways Jack can reach the block (j, i) safely from the bridge start

(b) Recurrence Relation:

$$OPT(j,i) = \begin{cases} 0, & \text{if } BadTile(j,i) = 1 \\ OPT(j,i-1) + OPT(j+1,i-1), & \text{if } j = 1 \\ OPT(j,i-1) + OPT(j+1,i-1) + OPT(j-1,i-1), & \text{if } j = 2 \\ OPT(j,i-1) + OPT(j-1,i-1), & \text{if } j = 3 \end{cases}$$

(c) **Code:**

```
Given BadTile (3,n)
Initialize OPT(3,n) with 0 for each element
OPT(1,1) = 1 if BadTile (1,1) equals to 0, else 0
OPT(2,1) = 1 if BadTile (2,1) equals to 0, else 0
OPT(3,1) = 1 if BadTile (3,1) equals to 0, else 0
for i in 2, 3,4, ..., n:
    for j in 1, 2, 3:
        if BadTile(j, i) equals to 1:
            OPT(j, i) = 0
```

```
            if j == 1:
                OPT(j, i) = OPT(j, i-1) + OPT(j+1, i-1)
            if j == 2:
                OPT(j, i) = OPT(j,i-1)+OPT(j+1,i-1)+OPT (j-1,i-1)
            if j == 3:
                OPT(j, i) = OPT(j,i-1) + OPT (j-1,i-1)
    return OPT(1, n)+OPT(2, n)+OPT(3, n)
```

(d) Base Cases:

i. OPT(1,1) = 1 if BadTile (1,1) equals to 0, else 0

ii. OPT(2,1) = 1 if BadTile (2,1) equals to 0, else 0

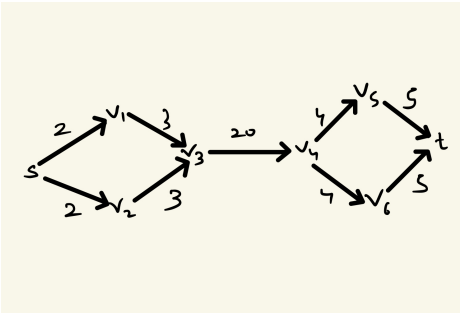
iii. OPT(3,1) = 1 if BadTile (3,1) equals to 0, else 0

(e) For running time analysis, we in total have $\mathcal{O}(n)$ because outer loop runs till n and inner loop runs just 3 times. Therefore, the total time is $\mathcal{O}(n)$.

4. Given a flow network with the source s and the sink t, and positive integer edge capacities c. Prove or disprove the following statement: if deleting edge e reduces the original maximum flow more than deleting any other edge does, then edge e must be part of a minimum s—t cut in the original graph.

Solution : False

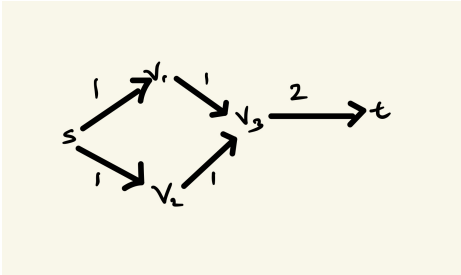
Counter Example: Deleting the edge with capacity 20 will reduce the flow by the most, while it is not part of minimum s—t cut.



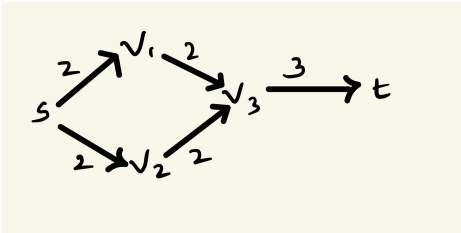
5. Given a flow network with the source s and the sink t, and positive integer edge capacities c. Let s—t be a minimum cut. Prove or disprove the following statement: If we increase the capacity of every edge by 1, then s—t still be a minimum cut.

Solution : False

Counter Example: Initially, a s—t min cut is S:$\{s, v_1, v_2\}$ and T:$\{v_3, t\}$



After increasing capacity of every edge by 1, s—t is no longer a min-cut. We now have S':$\{s, v_1, v_2, v_3\}$ and T':$\{t\}$



6. **(Network Flow)**

   (a) For the given graph G1 (see Fig. 2), find the value of the max flow. Edge capacities are mentioned on the edges. (You don't have to show each and every step of your algorithm).

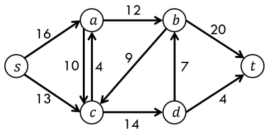


Figure 2: G1

   (b) For the given graph, find the value of the min-cut. (You don't have to show each and every step of your algorithm).

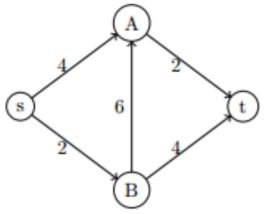


Figure 3

Solution :

(a) Run Ford-Fulkerson

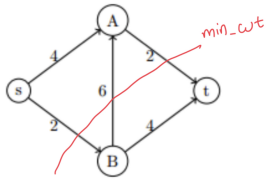
i. Select path s-a-b-t so the flow=12

ii. Select path s-c-d-t so the flow=4

iii. Select path s-c-d-b-t so the flow=7

So Max-flow of the given graph is 12+3+7=23

(b) Refer the figure below for min-cut



Min cut is S:$\{s, A\}$ and T:$\{B, t\}$ and the value of min-cut is 4

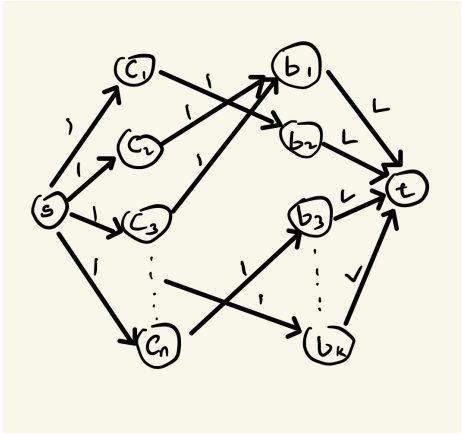
7. **(Network Flow)** Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible base stations. We'll suppose there are n clients, $c_1, c_2, ..., c_n$, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations, $b_1, b_2, ..., b_k$; the position of each of these is specified by (x, y) coordinates as well. For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a range parameter R which means that a client can only be connected to a base station that is within distance R. There is also a load parameter L which means that no more than L clients can be connected to any single base station. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station. Prove the correctness of the algorithm.

Solution :

(a) **Step 1:**

Let c_i be vertex for every client and b_i be vertex every base station. Connect each client to a base station if that station is within the range R. Assign capacity 1 to those edges. Let s be a source vertex and t be a sink vertex. For every client vertex, add an edge (s, c_i) of capacity 1. For every base station vertex, add an edge (b_i, t) of capacity L.

Refer the figure below



(b) **Step 2:Claim**

The problem has a solution (i.e every client is connected simultaneously to a base station) if and only if the network has a max-flow of value n.

(c) **Step 3:Proof of Correctness**

i. **Forward Claim:** If the problem has solution then the max-flow of network is n

Proof: If there exists a feasible solution for the problem, this means that every client is connected to a base station. By the law of conservation, the outward flow at the client vertex must be equal to the incoming flow to client vertex.Therefore, all the edges from source to client vertices will be saturated as every client was able to connect to base station. Similarly on the edges between clients and stations, we assign a flow of 1 or 0, depending whether a client is connected by a particular station or not. On the edges between stations and the sink, we assign a flow value equal to the number of clients covered by that station. This is possible, since we have a valid solution. It follows the max-flow is n.

ii. **Backward Claim:** If the max-flow of network is n then the problem has solution

Proof: If the problem has a max flow of n. This means that along any path from source to sink vertex there is a client who is connected to base station.

Because the flow is max flow. This means that each client will get a flow of one unit. We also observe that no base station is overloaded due to the capacity condition L.Hence, the problem has solution if the max-flow of network is n

8. **(Network Flow)** You are given a flow network with unit-capacity edges: it consists of a directed graph $G = (V, E)$ with source s and sink t, and $u_e = 1$ for every edge e. You are also given a positive integer parameter k. The goal is delete k edges so as to reduce the maximum s —t flow in G by as much as possible. Give a polynomial-time algorithm to solve this problem. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum s —t flow in the graph $G' = (V, E - F)$ is as small as possible. Give a polynomial-time algorithm to solve this problem.

   Solution :

   (a) Assume the value of max-flow of given flow network $G(V, E)$ is g. By removing k edges, the resulting max-flow can never be less than $g-k$ when $|E| \geq k$, since each edge has a capacity 1.

   (b) According to max-flow min-cut theorem, there is an s—t cut with g edges.

      i. If $g \leq k$, then remove all edges in that s—t cut, decreasing max-flow to 0 and disconnecting s and t.

      ii. Else if, $g > k$, then remove k edges from g, and create a new cut with $g - k$ edges.

   (c) In both the cases, whether the max-flow is 0 or $g - k$, the max-flow cannot be decreased any further thus giving us the maximum reduction in flow.

   (d) The algorithm has polynomial run-time. It takes polynomial time to compute the minimal-cut and linear time in k to remove k edges.

9. **(Network Flow)** Counter Espionage Academy instructors have designed the following problem to see how well trainees can detect SPY's in an nn grid of letters S,P, and Y. Trainees are instructed to detect as many disjoint copies of the word SPY as possible in the given grid. To form the word SPY in the grid they can start at any S, move to a neighboring P , then move to a neighboring Y . (They can move north, east, south or west to get to a neighbor.) The following figure shows one such problem on the left, along with two possible optimal solutions with three SPY's each on the right (See Fig. 4). Give an efficient network flow-based algorithm to find the largest number of SPY's.
   Note: We are only looking for the largest **number** of SPY's, not the actual location of the words. No proof is necessary.

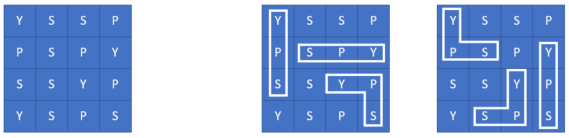


Figure 4

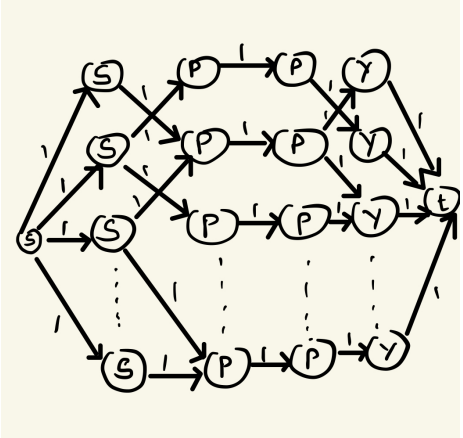
Solution :

(a) Create one layer of nodes for all the S's in the grid. Connect this layer to a source vertex s, directing edges from s to S with edges of capacity 1.

(b) Create two layers of nodes for all the P's in the grid. Connect the first layer to the S's based on whether or not they are adjacent in the grid, directing edges from S to P. Also, connect the first layer to the second layer by connecting the nodes that correspond to the same location. In other words, we are representing P's as an edge with capacity 1.
Note: By creating two layers of P helps us solving the edge case where P is adjacent to $2S'$s and $2Y'$s

(c) Create a layer of nodes for all the Y's in the grid. Connect these to a sink vertex t, directing edges from Y to t. Also, connect them to the second layer of P's based if the P and Y are adjacent in the grid, directing edges from P to Y with edge capacity 1.

Refer the figure below



Answer: Value of Max Flow in this Flow Network will give us the maximum number of disjoint SPYs.

10. **(Network Flow)** USC students return to in person classes after a year long interval. There are k in-person classes happening this semester, $c_1, c_2, ..., c_k$. Also there are n students, $s_1, s_2, ..., s_n$ attending these k classes. A student can be enrolled in more than one in-person class and each in-person class consists of several students.

(a) Each student s_j wants to sign up for a subset p_j of the k classes. Also, a student needs to sign up for at least m classes to be considered as

a full time student. (Given: $p_j \geq m$) Each class c_i has capacity for at most q_i students. We as school administration want to find out if this is possible. Design an algorithm to determine whether or not all students can be enrolled as full time students. Prove the correctness of the algorithm.

(b) If there exists a feasible solution to part (a) and all students register in exactly m classes, the student body needs a student representative from each class. But a given student cannot be a class representative for more than r (where $r < m$) classes which s/he is enrolled in. Design an algorithm to determine whether or not such a selection exists. Prove the correctness of the algorithm. (Hint: Use part (a) solution as starting point).

Solution :

(a) i. We can solve the problem by constructing a network flow graph and then running Ford–Fulkerson algorithm to get the max flow. If the max flow is equal to nm then all students can be enrolled as full time students.

ii. We can make the graph as follows:

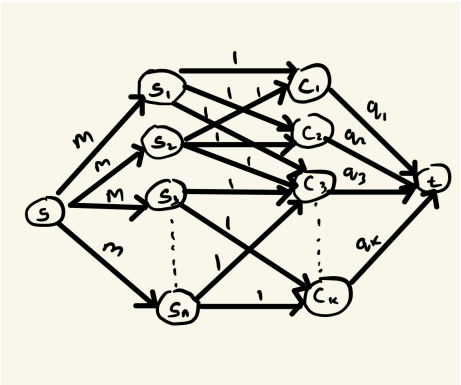
A. Make a layer of n students as vertices and a layer for k classes as vertices. Connect each student s_j with all the classes in p_j using edges with capacity of 1.

B. Place a sink vertex which is connected to all the classes. Also, place a source vertex which is connected to all the students.

C. Connect all s_j student vertices to the source vertex with capacity of m.

D. Connect all c_i class vertices to the sink vertex with edge capacity of q_i.

iii. Refer the graph below:



iv. **Claim:** The enrollment is feasible if and only if there exists a max flow of nm.

v. **Proof of Correctness**

A. **Forward Claim:**If the problem has a feasible solution then the max flow is nm

Proof: If there exists a feasible solution for the problem, this means that all the in-person classes has at max of q_i students and each student was able to sign up for at least m classes. By the law of conservation, the outward flow at the student vertex must be equal to the incoming flow to student vertex. Therefore, all the edges from source to the student vertices will be saturated with value of m, as each of the student was able to sign up for there classes. Similarly all the incoming flow at student vertex will be distributed to some of the out going edge with value of 1. This total flow from student vertices to the class vertex will be nm and that will eventually flow to the sink vertices with edges of max capacity q_i. Therefore, if the problem has a feasible solution the the max flow will be nm.

B. **Backward Claim:** If the max flow is nm then the problem has a feasible solution

Proof: If the problem has a max flow of nm. This means that along any path from source to sink vertex there is a student who was able to sign up for a particular class.

Because the flow is max flow, this means that each of the student vertices receive a flow of m and the outward flow at each of the student vertices will be exactly m. Thus, the student will be able to sign up for at least m classes, which satisfies the constraint of the problem. Also, by the construction of network flow each class can accommodate at most q_i, which satisfies another constraint as the flow along the edge will be at most q_i (that is less than or equal to the capacity). Hence, the problem has a feasible selection if the max flow is nm.

(b) i. We can solve the problem by starting from the solution from part (a). We will modify the constructed network flow graph slightly and then will run Ford–Fulkerson algorithm to get the max flow. If the max flow is equal to the number of classes k then a selection exists otherwise no such selection exists.

ii. We can make the graph as follows:

A. The nodes will be same as constructed in Part (a).

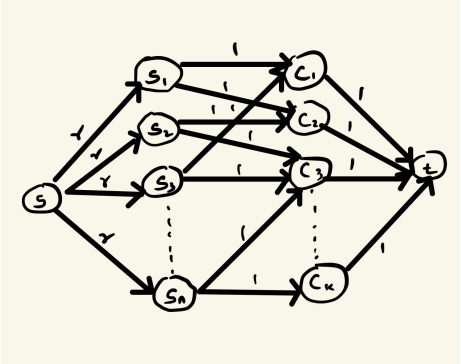
B. The edges between students and courses will be reduced to what is available in the solution from part (a) i.e. we will remove some edges between students and courses that they did not get to sign up for.

C. The capacity on student to class edges will same remain same (i.e. 1).

D. Assign r capacity for edges between source and student vertices.

E. Assign 1 as the capacity from classes to sink vertex, as there can only be 1 student representative from each class.

iii. Refer the graph below:



iv. **Claim:** The selection is feasible if and only if there exists a max flow of k.

v. **Proof of Correctness**

A. **Forward Claim:** If the problem has a feasible selection then the max flow is k

Proof: If there exists a feasible selection for the problem, this means that all the in-person classes had a valid selection and each student was at max selected for r selections. By the law of conservation, the outward flow at the class vertex must be equal to the incoming flow to class vertex. Therefore, all the edges from class vertices to sink will be saturated as each of the class had exactly 1 selection. This means that, k was the inward flow at class vertices. (law of conservation). Similarly, the number of selections per student will have the maximum value of r classes. And the total flow from source vertex to the student vertices will be k (again, by the law of conservation). Therefore, if the problem has a feasible solution the the max flow will be k.

B. **Backward Claim:** If the max flow is k then the problem has a feasible selection

Proof: If the problem has a max flow of k. This means that along any path from source to sink vertex there is a selection of student from a class.

Because the flow is max flow, this means that all the class vertices receive a flow of 1 and the outward flow at the class vertices will be exactly 1. Thus, only 1 selection of student

per class is made, which satisfies the constraint of the problem. Therefore, each class has selection of 1 student only. Also, by the construction of network flow each student can only be selected for at max r classes, which satisfies another constraint as the flow along the edge will at most r. Hence, the problem has a feasible selection if the max flow is k.