CSC373 – Problem Set 7

Remember to write your **full name(s)** and **student number(s)** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due TUESDAY, Nov 24, 2020, 22:00; required files: ps7.pdf, ps7.tex

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

[15 points]

The Michelin Star Restaurant Umami offers a large selection of dinner main courses. Unfortunately, as the ingredients the chef uses are unique and come in limited quantities, the restaurant can prepare only a limited number of main courses. This often leads to the restaurant running out of sought-after main courses, which in turn makes their customers unhappy.

To minimize customer dissatisfaction, Umami is launching a pre-order online system. Each customer selects their favourite main dishes before dinner time. Then the system uses an algorithm to pre-assign main courses to customers. Those customers who are not assigned a main course from one of their selections get a 15 CAD coupon. Naturally, Umami wants to minimize the number of coupons it distributes to the customers.

You are given m types of main courses, $a_1, a_2, \ldots, a_m$, and the quantity q_j of each main course type a_j, where $j = 1, \ldots, n$. Additionally, you are given n customers $c_1, \ldots, c_n$, where each customer c_i is associated with a set of favourite main courses A_i with $|A_i| \geqslant 1$, where $i = 1, \ldots, n$.

Your goal is to assign to each customer either (i) exactly one main course of his/her choice or (ii) a coupon of 15 CAD while satisfying the quantity constraints and minimizing the number of coupons assigned to the customers. Follow the steps below to design an algorithm for this problem that has worst-case time complexity of $O(mn^2)$.

1. **(1 point)** Describe how you will model the problem, and the data-structure(s) that you will use to store the input information.

   We can model the problem by building a flow graph, and find the minimum number of coupons by utilizing the property of the maximum flow in the flow graph.

   For data structure,

we will use a python set to store all the customers who are assigned with a main course, and a graph ADT built using python dictionaries and list to represent our flow graph. (link for reference: https://www.python.org/doc/essays/graphs/)

2. **(12 points)** Propose an algorithm that solves the problem in $O(mn^2)$ time and prove the correctness of your algorithm.

Our algorithm consists of the following steps:

(1).Create a flow-graph G based on the customer's orders and the main courses.(explained in part(1))

(2).Use "modified" BFS in Ford-Fulkerson algorithm to find max-flow while storing the customers who are assigned a main course.

(3). Find the number of coupons given using the customers stored.

pseudo-code:

```
    def Bottleneck(p,f) = min (cf(u,v).inverse)
    # we will take the edge closer to the target

    def AUGMENT(f,p):
        let b <- Bottleneck(p,f)
        for all (u,v) in p:
            if (u,v) is a forward edge:
                f(e) <- f(e) + b
            if (u,v) is a backward edge:
                f(e) <- f(e) - b

    def Max_Flow(G,s,t,served):
        set f(e)<- 0 for all e in E(G)
        while there exist s-t path in Gf:
            p = a simple (s,t) path in Gf
            f' <- AUGMENT(f,p)
            set f <- f'
            compute Gf'
            set Gf <- Gf'
            served.add(the customer c_i associated with this s-t path)

        return f


    def Umami(a,q,c,A):
    """
    a is a list which elements are main course type
    q is a list which include the quantity qj of each main course type (ord
    c is a list which elements are customers.
    A is a list which element are also list represent a set of favourite ma
    """
    # initialize graph
    G <- (s,none) ---a directed graph wich only a source point s
    for i in range(len(a)):
        connect a forward edge between s and a[i] wich capacity q[i]
    for i in range(len(c)):
```

```
        for j in range(len(A[i])):
            connect a forward edge between A[i][j] and c[i] with capacity 1
    for i in range(len(c)):
        connect a forward edge between c[i] with target t with capacity 1
    served = set()

    # run Ford-Fulkerson
    Max-Flow =  Max_Flow(G,s,t,served)

    coupons = set(c).difference(served)


    return len(c)-Max-Flow, served, coupons
```

To prove the correctness of our algorithm, we will:

(1). Show how to build flow graph G

(2). Prove "if the restaurant has to give out a minimum number of k coupons, then n - max-flow(G) = k where n is the total number of customers".

(3). Prove "if n - max-flow(G) = k where n is the total number of customers, then the restaurant has to give out a minimum order of k coupons".

(4). Prove our python set contains the correct set of customers who are assigned a main course.

(1). let's first create the source vertex s and the sink vertex t.

Then, for each main course a_i, we create a vertex a_i and make an edge (s, a_i) with capacity$(s, a_i) = q_i$(the number of a_i the restaurant can cook).

After that, for each customer c_j, we create a vertex c_j and make an edge (a_i, c_j) **if and only if** $a_i \epsilon A_j$(meaning the main course a_i is in the customer c_j's favourite main courses), with capacity$(a_i, c_j) = 1$.

Finally, we create edge (c_j, t) for each customer c_j to connect them to the sink vertex t, with capacity$(c_j, t) = 1$.

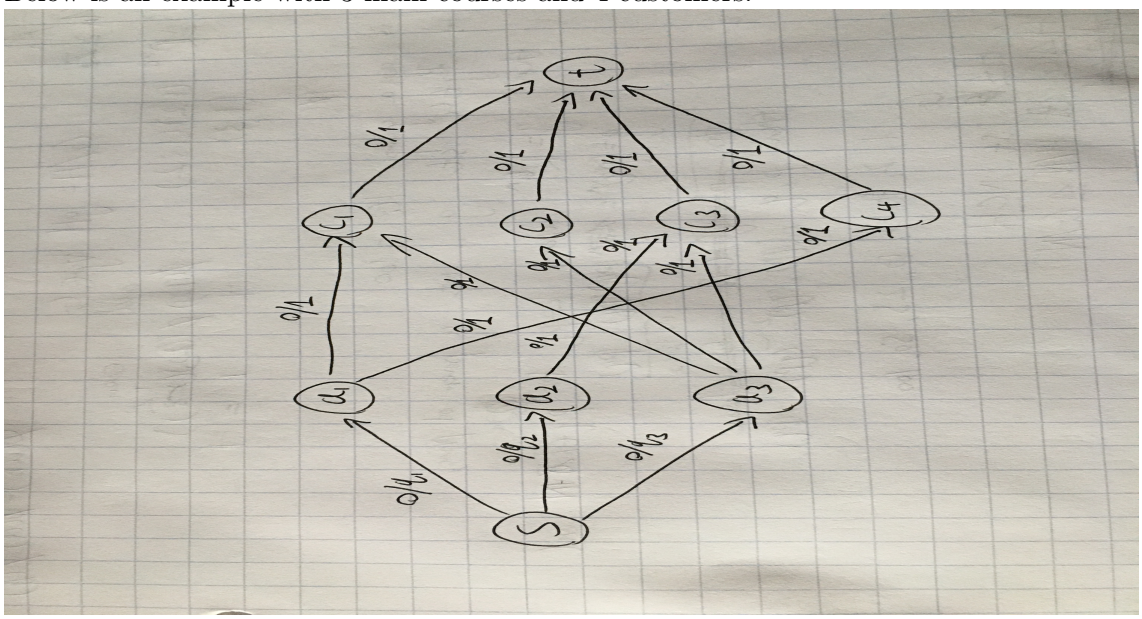
let G be a graph such that:

$V = \{s, t\} \bigcup \{a_1, a_2 ... a_m\} \bigcup \{c_1, c_2, ..., c_n\}$

$E = \{(s, a_i) : i = 1, 2..., m\} \bigcup \{(a_i, c_j) : i = 1, 2, ...mj = 1, 2...n\} \bigcup \{(c_j, t) : j = 1, 2, ...n\}$

capacity$(s, a_i) = q_i$, capacity(a_i, c_j) =capacity$(c_j, t) = 1$

Below is an example with 3 main courses and 4 customers:



(2). To prove this claim, let's assume that the restaurant has to give out a minimum number of k coupons to customers, then this implies that k customers didn't get assigned a main course.

First, since q_i is the number of a_i that can be made in the restaurant, and we cannot make part of the main course nor server part of the main course, and customers can only be satisfied or unsatisfied. So, all flows are **integers**.

Consider a max-flow in G,

For each vertex a_i

(a). Capacity$(s, a_i) = q_i$, because the restaurant only has ingredients to make q_i orders of a_i.

Then, flow$(s, a_i) \leqslant q_i$ or the restaurant will be "lying" to customers about their order, which cannot happen.

Therefore, the conservation constraint holds for all a_i.

(b). Capacity$(a_i, c_j) = 1$, because each customer should be assigned **exactly** 1 order of their favourite main courses, and flow$(a_i, c_j) = 1$ **if and only if** the customer c_j ordered the main course a_i and the restaurant still has enough ingredients to make a_i.

So, flow$(s, a_i) = \sum_{j=1}^{n}$flow$(a_i, c_j)$, as the restaurant will only prepare a_i if a customer orders it and the restaurant still has enough ingredients to make it.

Then, there is as many a_i being cooked in the restaurant as the number of customer ordering it, which implies flow_in(a_i)=flow_out(a_i).

Therefore, the conservation constraint holds.

For each vertex c_i

(a). Capacity$(a_j, c_i) = 1$, and flow$(a_j, c_i) = 1$ **if and only if** c_i ordered a_j and flow$(s, a_j) \leqslant$capacity(s, a_j) (in other words, the restaurant still has enough ingredients to make one more a_j).

So, flow$(a_j, c_i) \epsilon \{0, 1\}$(0 when c_i is not assigned a_j, or 1 otherwise), and since capacity$(a_j, c_i) =$ 1, flow$(a_j, c_i) \leqslant$capacity(a_j, c_i).

Therefore, the capacity constraint holds.

(b). Capacity$(c_i, t) = 1$ and flow$(c_i, t) = 1$ **if and only if** customer c_i is assigned one of his/her favourite main courses and is satisfied.

And flow$(c_i, t) = 1$ **if and only if** flow$(a_j, c_i) = 1$ for some a_j depending on the customer, so flow_in$(c_i) =$flow_out(c_i).

A special case would be "What if c_i is assigned to more than 1 main courses?"

Let's assume c_i has been assigned more than 1 main course, then there exist a_k such that $a_k \epsilon A_i$ and flow$(a_k, c_i) = 1$.

But, this case cannot happen, since once flow$(a_j, c_i) = 1$, flow$(c_i, t) = 1$, and blocking off all other paths using edge (c_i, t) to reach vertex t. A path $s => a_k => c_i => t$ will not be found by our BFS algorithm, since edge (c_i, t) is taken and reached maximum flow.

Therefore, we conclude that the conservation constraint holds.

Now, by proving the 2 constraints hold on all internal vertices, we have shown that the Ford-Fulkerson algorithm will terminate and returns the maximum flow to us.

Then, by construction of G, we know that max-flow(G) = "the number of customers that are assigned a main course and are satisfied" = m for some integer m. Then out of the n customers in total, we have m customers who are satisfied, so we have n-m customers that are unsatisfied and will need to give them coupons.

Hence, we have n-m=k customers that will need to give coupons, finishing the proof.

Note: the key to why we use Ford-Fulkerson is because of the power of residual graph. Consider, a case where some main course a_j is assigned to c_i at one point, but a_j is not assigned to c_i in any of the optimal solution when the algorithm terminates.

Then, that implies there exists another main course a_k such that, a_k is a better choice for c_i. Then our algorithm will use the path "$s => a_k => c_i => a_j => c_k => t$" to reach the sink where the edge $c_i => a_j$ is the backward edge created by our algorithm and it un-does our previous mistake.

(3). To prove this claim, we assume that a max-flow "F" in G with flow value m, we can show that the restaurant will have to give out at least n-m coupons.

Similarly, consider

for all edges "e" in f, s.t f(e) =1 (or e wouldn't be in f, since f is a max-flow containing only edges in max-flow0.

Define $X = \{(a_i, c_j) | f(a_i, c_j) = 1\}$. Let's consider the cut $(S \bigcup \{a_1, a_2, ...a_m\}, T \bigcup \{c_1, c_2, ...c_n\})$. All edges in X crossing this cut travels from a_i to c_j for $i, j \in \mathbb{N}$, s.t $i \leqslant m, j \leqslant n$.

This implies that each main course vertex a_i is connected to a customer c_j in f **if and only if** a_i is acceptable to $c_j (a_i \in A_j)$ and there is enough ingredient to make a_i (flow$(s, a_i) \leqslant$ capacitya_1). Also, since f is a max-flow, a_i is the optimal main course to c_j. Then flow on cut edges = flow value = k and $|X| = flowvalue = k$.

Now, we show x that represents the customers assigned a main course.

Let's consider an arbitrary edge in X, (a_i, c_j) s.t $f(a_i, c_j) = 1$

since $(a_i, c_j) \in X$ and

$f(a_i, c_j) = 1$ **if and only if** c_j has been served a_i.

Then c_j is satisfied since $a_i \in A_j$ or the edge $(a_i, c_j$ wouldn't have been initialized by construction of G. So c_j is satisfied.

Then we know $\forall(a_i, c_j) \in X$, c_j is satisfied, then $|X|$ = the number of customers who are satisfied. Then by the restaurant policy, we will only need to give n - $|X|$ number of coupons, where n is the number of customers in total.

(4). By adding the corresponding c_i to the set, it guarantees us to have the correct customers who were served when this algorithm terminates. To show this, we assume the set contains the wrong set of customers. Then there are 2 cases:

Case 1: We added customer that didn't get served with anything. Then $\exists c_i \in Set$ s.t c_i didn't get served. Since c_i is in set **if and only if** c_i is added (and later augmented) by the algorithm, the only reason customer didn't get served anything is because this customer's course a_j was "assigned" to another customer (undone by Ford-Fulkerson).

However, in order to be "undone", the FF algorithm will need to use another edge (a_k, c_i) to get to vertex c_i, then use the backward edge (c_i, a_j) to undo the order(in other words, the customer c_i will need to be assigned another main course a_k which he/she likes). Then c_i is served (namely, with a_k), so we have a contradiction.

Case 2: We didn't add customer that got served to the set.

Since our BFS algorithm adds the customers c_i to the set based on the reachability of S to t through c_i.

If (a_i, c_i) has been used in some paths from s to t, then c_i will be added by the algorithm. So, this case is impossible to happen.

Therefore, by proving all 4 claims, we show that the algorithm is correct.

Time complexity:

We first consider the runtime of building the flow-graph G:

(1). initializing all vertices in G and connect each a_i to s and each c_j to t. (O(1+m+n+1)=>O(m+n) where m="number of main courses" and n="number of customers")

(2). build edges based on each customer's set of favourite main courses. (O(mn), worst case being all n customers can accept all m main courses).

So, the total time complexity of building a graph is "O(m+n)+O(mn)=>O(mn)".

Our modification to the BFS algorithm(checking if (c_i, t) is used) takes constant time (look up in list and add to set), which is dominated by BFS's generic time complexity, which is then dominated by Ford-Fulkerson's complexity, so we will focus solely on complexity of Ford-Fulkerson.

Time complexity of the FF algorithm depends on the max-flow on G and the number of edges in G.

For the number of edges, let's consider the 4 layers of vertices in G.

(s, a_i): At this layer, we have m edges with m being the total number of main courses.

(a_i, c_j): At this layer, we can have maximum number of mn edges, which is the scenario where all main courses are acceptable to all customers(then wee connect each a_i with every customers).

(c_j, t): At this layer, we have n edges, where each edge represent if that customer has

gotten a main course.

In total, we have O(m + mm + n) => O(mn) edges. And the maximum max-flow is n, which is the case where all n customers got served. In conclusion, we have $O(nmn) = O(mn^2)$ for our Ford-Fulkerson algorithm.

Combining with the time complexity of building the flow-graph, we have $O(mn)+O(mn^2)$ => $O(mn^2)$ time complexity, which satisfies the requirement.

3. **(2 points)** Use the algorithm from Part 2 to list all the customers who have been assigned a favourite main course and all those who have been assigned a coupon in $O(mn^2)$.

Because of our modification to the BFS algorithm, the customers that have been assigned a favourite main course have been stored in our set. To find the customers that received a coupon(didn't get assigned), we can

(1) insert all customers to set ($n \cdot O(1)$ => $O(n)$)
(2) find set difference between the 2 set (O(n))
(3) return set difference (O(1))

So, in total, we have O(n) + O(n) + O(1) => O(n) complexity to find the customers who received a coupon, which satisfies the requirement.