Assignment-6

Design and Analysis of Algorithms

Trishan Mondal

⚠ Disclaimer. Consider the following set of students

$$\mathcal{P} = \{\text{Trishan, Aaratrick basu}\}$$

Discussion of solutions to the assignment problems are limited to the people of set $\mathcal{P}$ only. Most of the problems in this assignment has general solution. If any other person have same solution as mine is not my fault.

§ Problem 25

(a) We can assume $r > 0$ as the graph has a perfect matching vacuously if $r = 0$. Now, as $|U|r = |V|r$ by counting edges going out of U and V, we get $|U| = |V|$. Then, U is a vertex cover as every edge of the graph must have an endpoint at some vertex of U. This is a minimum vertex cover for G because if every edge avoids some vertex of U, then the degree of that vertex is 0, a contradiction. Hence, the minimum vertex cover is of size $|U|$ and by the König-Egervaŕy theorem, this must also be the size of a maximum matching. Therefore, there is a perfect matching in G. We remove all the edges occurring in this matching to get a $(r-1)-$regular graph. If $r > 1$, we can repeat the above arguments to get a new perfect matching. Hence, we get $E = E_1 \cup \cdots \cup E_r$, where each E_i is the edge set of a perfect matching and no two E_i are the same.

(b) Consider the bipartite graph on $U \sqcup V$, where both U and V have n vertices, representing buses. An edge $(u, v)_s$ exists for each student s on bus u in the Principal's plan and on bus v in the Vice Principal's plan, resulting in a $20-$regular bipartite graph. According to (a), this graph possesses a perfect matching. Therefore, as each edge represents a student, we have a set of n students who can serve as bus monitors under both plans.

(c) Let $n = \frac{|H|}{|K|}$. Consider the bipartite graph on $L \sqcup K$, where an edge $(uK, Kv)_g$ is present in the graph for each group element g that is present in both the left coset uK and the right coset Kv. Hence, we get a $|K|-$regular bipartite graph, because every coset has $|K|$ many elements. By (a), this graph has a perfect matching, and thus, as each edge arises because of some group element, we get a set of n elements who can serve as common coset representatives. ■

§ Problem 26

(a) For this and the next problem, let $n = |V|$ and $m = |E|$, where V is the vertex set of G and E is the edge set of G. For this problem, we can assume without loss of generality that f is not the max flow, i.e $\Delta > 0$, as there are always flows of value 0 and all $s - t$ paths in the residual graph of a max-flow must have bottleneck capacity 0. We first note that the capacities of the residual graph G^f are:

$$c^f(u, v) = \begin{cases} c(u, v) - f(u, v), \text{ if } (u, v) \in E, f(u, v) < c(u, v) \\ f(v, u), \text{ if } (v, u) \in E, f(v, u) > 0 \end{cases}$$

Hence, the following map

$$g^*(u,v) = \begin{cases} f^*(u,v) - f(u,v), \text{ if } (u,v) \in E, f(u,v) < c(u,v) \\ 0 \quad \text{if } (v,u) \in E, f(v,u) > 0 \end{cases}$$

satisfies the feasibility condition for flows, because $f^*(u,v) \leq c(u,v)$ and $0 \leq f(u,v)$. Further, for all $u \in V \setminus \{s,t\}$,

$$\sum_{(x,u)\in E(G^f)} g^*(x,u) = \sum_{(x,u)\in E} (f^*(x,u) - f(x,u)) = \sum_{(u,y)\in E} (f^*(u,y) - f(u,y)) = \sum_{(u,y)\in E(G^f)} g^*(u,y)$$

where the first and third equalities follow from the definition of g^* and the second equality follows from the conservation law of flows f and f^*. Hence, g^* is a flow on G^f and it has value

$$\sum_{(s,x)\in E(G^f)} g(s,x) = \sum_{(s,x)\in E} (f^*(s,x) - f(s,x)) = \Delta$$

as was required.

(b) Let $G_0 = G^f$ and $G_1 = (G^f)^{g^*}$ represent the residual graph concerning g^*. We observe that in G^f, at least one edge of non-zero capacity must be saturated by g^*. This is because there exists at least one edge in G where $f(u,v) < f^*(u,v) = c(u,v)$, and g^* must saturate the corresponding edge in G_1. Consequently, G_1 has one fewer edge of non-zero capacity compared to G_0. Therefore, we can iterate this process up to m times until the new flow g^*_m on G_m becomes zero. However, this scenario contradicts the case where all $s-t$ paths in G_1 have bottleneck capacities strictly less than $\frac{\Delta}{m}$. Hence, at least one $s-t$ path in G^f has a bottleneck capacity of at least $\frac{\Delta}{m}$. ■

§ Problem 27

Let the bottleneck capacity of a vertex be the maximum capacity. We use the following modification to Dijkstra's algorithm (from the midsem exam) to find a path with maximum bottleneck capacity. Let, for $v \in V$, $v.\texttt{bcap}$ denote the capacity of a maximum $s-v$ bottleneck path.

```
Input: G, with source s and sink t, stored using adjacency lists, with attributes for
 storing capacities of edges and bottleneck capacities of vertices.
Output: The output is an s−t path of maximum bottleneck capacity.

for v ∈ V:
    v.prev = None, v.bcap = 0
s.bcap = ∞
H = makeheap(V)
While H ≠ ∅:
          u = deletemax(H)
 for (v, cap) ∈ u.out_nbrs:      #cap is the capacity of(u, v)
     if v.bcap < min(u.bcap, cap):
     v.bcap = max(u.bcap, cap)
      v.prev = u
      bubble_up(H, v)
P = [] #Stores the vertices in thes−t maximum bottleneck path in order
v = t
while v ≠ None
    P = P.append(v)
    v = v.prev
return P
```

The correctness of the algorithm follows from the definition of bottleneck capacity and the correctness of Dijkstra's algorithm. The time complexity of this algorithm is $O((m+n)\log n)$, the same as Dijkstra's algorithm, as the last loop runs at most n times and the rest of the algorithm has the same complexity as Dijkstra's

algorithm. We rely on the outcome established in problem 26, where any flow f that isn't the maximum flow f^* is guaranteed to possess an augmenting path with a bottleneck capacity of at least $\frac{\Delta}{m}$, where $\Delta = \mathsf{val}(f^*) - \mathsf{val}(f)$. Let f_j represent the flow obtained after the j^{th} iteration, f_0 denote the zero flow, $\Delta_j = \mathsf{val}(f^*) - \mathsf{val}(f_j)$, and α_j stand for the maximum bottleneck capacity in the residual graph G^{f_j}. Consequently,

$$\mathsf{val}(f_{j+1}) = \mathsf{val}(f_j) + \alpha_j \geq \mathsf{val}(f_j) + \frac{\Delta_j}{m} \implies \Delta_{j+1} \leq \Delta_j \left(1 - \frac{1}{m}\right)$$

By induction, $\Delta_j \leq \Delta_0 \big(1 - \frac{1}{m}\big)^j = \mathsf{val}(f^*)\big(1 - \frac{1}{m}\big)^j$. Given that capacities are integers, the number of augmentations needed is N_0, where N_0 is the least integer satisfying

$$\mathsf{val}(f^*)\left(1 - \frac{1}{m}\right)^{N_0} < 1$$

For $N = \lceil m \ln \mathsf{val}(f^*) \rceil$, we have

$$N \geq m \ln \mathsf{val}(f^*) \implies 0 \geq \ln \mathsf{val}(f^*) - \frac{N}{m} > \ln \mathsf{val}(f^*) + N \ln \left(1 - \frac{1}{m}\right) = \ln \left(\mathsf{val}(f^*)\left(1 - \frac{1}{m}\right)^N\right)$$

Here, the strict inequality arises from the fact that for $m \in \mathbb{Z}_{\geq 1}$,

$$e^{-\frac{1}{m}} > 1 - \frac{1}{m} \implies -\frac{1}{m} > \ln \left(1 - \frac{1}{m}\right)$$

Therefore, $N_0 \leq \lceil m \ln \mathsf{val}(f^*) \rceil$, as required. ■

§ Problem 28

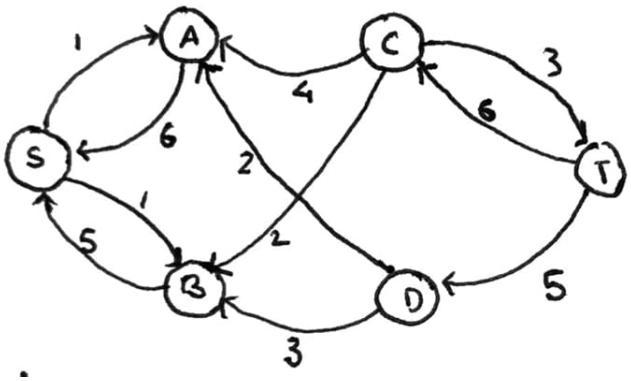
(a) A max flow is given by:

$$f(S,A) = 6,\ f(S,B) = 5,\ f(A,C) = 4,\ f(A,D) = 2$$
$$f(B,C) = 2,\ f(B,D) = 3,\ f(C,T) = 6,\ f(D,T) = 5$$

The value of this flow is $\mathsf{val}(f) = 6 + 5 = 11$.

A min cut is given by $(\{S, A, B\}, \{C, D, T\})$, with capacity $4 + 2 + 2 + 3 = 11$.

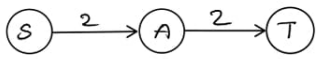
(b) The residual graph G^f is:



The labels of the edges are the edge capacities. The vertics A, B are reachable from S, and T is reachable from the vertex C.

(c) The bottleneck edges are AC and BC. Clearly increasing the capacity of any of these edges will result in an increased max flow because the edges (S, A) and (S, B) are not saturated. No other edge is a bottleneck edge because every other edge is adjacent to a saturated one.

(d) Consider the network:



where the edge labels are the capacities. Clearly, a max flow is the flow saturating both edges. It is clear that there are no bottleneck edges, because both edges are saturated and an increase in the capacity of one will not affect the max flow.

(e) Consider the algorithm outlined as follows:

1. Employ the standard algorithm on the network to obtain a maximum flow, and retain the final residual graph, storing solely those edges with non-zero capacity.
2. Perform Depth-First Search (DFS) initiated at S and compile all reachable vertices from S into an array L_1.
3. Execute DFS on the reversed graph, commencing from T, and capture all vertices from which T is reachable, storing them in an array L_2.
4. Output all edges in the original network that link vertices in L_1 to vertices in L_2. This can be executed efficiently by hashing the adjacency lists of the vertices in L_1 and L_2, followed by an iteration through all edges while seeking the required edges utilizing the hash table.

Correctness: According to the definition of the residual graph and considering that the max flow algorithm concludes at this stage, all edges produced by the aforementioned algorithm must have reached maximum capacity. Consequently, if any of these edges had their capacities increased, it would have enabled another iteration of the max flow algorithm due to the emergence of a new $S - T$ path. Thus, the algorithm exclusively outputs bottleneck edges.

Furthermore, considering that any bottleneck edge must originate from a vertex accessible from S and terminate at a vertex reachable from T, it is established that no other bottleneck edges exist beyond those generated by the algorithm.

Time Complexity: Each subsequent step following the initial one can be executed in linear time. Hence, the overall time complexity stands at $O(m + n)$, in addition to the time complexity of the max flow algorithm, contingent upon the chosen heuristic and the value of the max flow in the network. ■

§ Problem 29

Let A represent a polynomial-time algorithm that determines whether a graph contains a Hamilton-Rudrata path. Given a graph $G = (V, E)$ as input, we execute the following steps:

1. Compute $A(G)$. If $A(G) = 0$, we conclude that the graph lacks a Rudrata-Hamilton path and terminate. If $A(G)$ equals 1, proceed to the next step.
2. Initialize an empty array P to record the edges forming the path.
3. Iterate through each edge $e \in E$. For every edge e, calculate $A(V, E \setminus \{e\})$. If the result is 1, remove this edge from the set, i.e., $E \leftarrow E \setminus \{e\}$.
4. If the output is 0, add this edge to P.

Finally, sort P to arrange the edges' vertices in order. This algorithm correctly identifies a Rudrata-Hamilton path if one exists. The process ensures $A(V, E' \setminus \{e\}) = 0$ and $A(V, E') = 1$ only if e is a mandatory part of all Rudrata-Hamilton paths in (V, E). Given that A operates in polynomial time, this algorithm too functions within a polynomial time.

§ Problem 30

(a) To demonstrate, for any graph G with a maximum degree at most 3, and a proposed solution for CLIQUE-3 with parameter k, we aim to verify this solution in polynomial time relative to the size of G. The proposed solution entails a subset of vertices that should form a complete subgraph. Notably, given the degree constraint, the maximum clique size is 4. Rejecting the solution if $k > 4$, we can verify the proposed solution's completeness in $O(1)$ time for 4 vertices and in 9 edge checks for 3 vertices. Therefore, using this verification process, CLIQUE-3 is in **NP**. ■

(b) The showcased "proof" demonstrates CLIQUE-3 $\leq_m^p$ CLIQUE, signifying an instance reduction from CLIQUE-3 to CLIQUE. However, this doesn't offer any insights into the complexity of CLIQUE-3. To establish that a problem in **NP** is **NP**-complete, we necessitate reducing a known **NP**-complete problem to the original one.

(c) The flaw in this proof lies in the assertion that a subset $C \subseteq V$ is a vertex cover if and only if $V \setminus C$ is a clique. According to the vertex cover definition, C is a vertex cover if $V \setminus C$ constitutes an independent set, i.e., none of the vertices in $V \setminus C$ are adjacent. Consequently, they cannot form a clique.

1. Leveraging the insight from (a) that any clique in a graph with a maximum degree at most 3 has a size of either 3 or 4, we handle parameters exceeding 4 by outputting the absence of a clique larger than the parameter. Otherwise, we iterate over all size 4 and size 3 subsets of V, checking if they form a clique as in (a). The algorithm's correctness is evident, and its time complexity is $O(\binom{n}{4} + \binom{n}{3}) = O(n^4)$, where $n = |V|$, meeting the requirements.

The extra credit problem is written in hands and attached to this pdf please note below !!