Introduction to Algorithms **Guide to HW 6**
CS 4820, Spring 2019

(1) *(10 points)* You are trying to sabotage a flow network $G = (V, E)$ on n nodes and m edges by removing at most k edges from G to form a new flow network G'. Assume that every edge in G has unit capacity. Design an algorithm that given as input a flow network G and an integer $k > 0$ outputs a list of at most k edges to be removed from G such that value of the maximum flow of the resulting flow network G' is minimized. Your algorithm should run in time $O(mn)$.

Solution. We use the following algorithm to compute the k-edges.

Algorithm 1 DELETE-EDGE(G,k)

```
Compute a maximum flow $f$ of $G$ using the Ford-Fulkerson algorithm.
Construct the residual graph $G_f$.
Let $A = \{u \in V : \text{ there is a path from } s \text{ to } u \text{ in } G_f\}$, and $B = V \setminus A$.
Define the set $E_{A,B} = \{e = (u, v) \in E : u \in A \text{ and } v \in B\}$, and
let $k_0$ denote the cardinality of $E_{A,B}$.
if $k_0 \geq k$ then
    Output any $k$ edges from the set $E_{A,B}$.
else
    Output all $k_0$ edges in the set $E_{A,B}$.
end if
```

The running time of Step 1 is $O(mn)$ since $\sum_{e=(s,u)} c(s, u) \leq m$, where we use the fact that each edge has unit capacity. The other steps of the algorithm can be executed in $O(m)$ time, which gives a total running time of $O(mn)$.

We now turn to proving correctness of the algorithm. Suppose that $k_0 \geq k$. We observe that (A, B) is an s-t cut in G, and is in fact a min-cut (see Proposition 7.2, Pg 348 in the Kleinberg-Tardos book). Using the max-flow min-cut theorem, we know that $v(f) = c(A, B)$. Further, notice that the capacity of the min-cut of any G' (obtained by deleting some k edges of G) is at least $v(f) - k$, since the capacity of any edge in G is 1. Thus, the max-flow of any such G' is at least $v(f) - k$.

Finally, it is easy to observe that the s-t cut (A, B) in the graph G', which was constructed from G by deleting any k edges from the set $\{e = (u, v) \in E : u \in A \text{ and } v \in B\}$, has capacity exactly $v(f) - k$. It follows that this must be a min-cut of G'. Hence the max-flow in G' has value $v(f) - k$, proving optimality of our algorithm by the argument above.

In the case when $k_0 < k$, it follows by arguing as above that the min-cut in the modified graph G' has capacity 0, and thus the max-flow in this graph is 0. Thus, optimality of our algorithm is direct in this case.

(2) *(15 points)* m tourists are wandering in the streets of Manhattan, and they are hungry. Fortunately, there are m hotels as well, and you have been assigned the task to match tourists with hotels. We make the following assumptions:

- Model Manhattan as the $n \times n$ grid $[0, n-1] \times [0, n-1]$.
- Tourists can walk either horizontally or vertically on this grid. The time taken by a tourist to walk from (x_1, y_1) to (x_2, y_2) is given by $|x_1 - x_2| + |y_1 - y_2|$.

- Tourists can walk in parallel on a segment. (This models the fact that roads are wide enough for multiple people to walk simultaneously.)
- Each hotel can accommodate at most one tourist.
- Each tourist has a list of at most r hotels (which is a subset of the m available hotels) which they would be happy to go to.

You are given as input the initial locations of the m tourists, their hotel preferences, and the locations of the m hotels. Assume that time $T = 0$ initially. Define the cost of an allocation to be the minimum time T_0 such that all tourists reach their respective hotels (starting from their initial locations) at time $T = T_0$. Recall that we are assuming that the tourists walk simultaneously to their hotels.

Design an algorithm to find the minimum cost allocation that makes all the tourists happy (the algorithm should output 'No' if no such allocations exist). The running time of your algorithm should be $O(m^2 r \log n)$.

Example: Suppose there are 2 tourists a_1, a_2 located at $(0, 0)$ and $(1, 0)$ respectively. Further suppose there are 2 hotels h_1, h_2 located at $(0, 1)$ and $(1, 1)$ respectively. Let the lists of both a_1 and a_2 be $\{h_1, h_2\}$ and $r = 2$. Then, an assignment of a_1 to t_1 and a_2 to t_2 makes both of them happy. Further, under this allocation t_1 and t_2 reach their respective hotels at time $T = 1$. In the other assignment, which is a_1 to h_2 and a_2 to h_1 also makes both of them happy. However, they both reach their hotels at time $T = 2$, which is worse than the previous allocation. Thus, in this case your output should be (t_1, h_1) and (t_2, h_2).

Solution. We create a weighted bipartite graph $G = (A \times B, E)$ as follows: Let A be the list of tourists and B be the list of hotels, and $e = (a, b)$ is an edge in G if tourist a has the hotel b in their 'happy' list. Further, the weight w_e of such an edge $e = (a, b)$ is set to the time taken by the tourist a to walk to the hotel b from their initial position. Now, for any integer $k > 0$, we define $G_k = (A \times B, E_k)$, where $E_k \subseteq E$ defined as $E_k = \{e \in E : w_e \leq k\}$.

We use the notation that if H is a weighted graph, then $\overline{H}$ denotes the corresponding unweighted graph. We will also slightly abuse notation, and by an allocation we will mean a valid allocation (i.e all tourists matched up with a hotel that makes them happy).

We now prove a couple of simple yet crucial claim.

Claim 1 If there exists a perfect matching in $\overline{G_k}$, then the cost of the optimal allocation is at most k.

Proof. Let M be a perfect matching in $\overline{G_k}$. Consider the allocation of tourists to hotels according to this allocation. Since all the edges in $\overline{G_k}$ appear in G_k, it follows that under this allocation the maximum time a tourist takes to reach their respective hotel is bounded by k. Thus, the cost of an optimal allocation is bounded above by k. □

Claim 2 Assume that there exists an optimal allocation. If there exists does not exist a perfect matching in $\overline{G_k}$, then the cost of the optimal allocation is more than k.

Proof. Suppose if possible there exists an allocation M with cost $\leq k$. This implies that all the edges corresponding to this allocation in the graph G has weight at most k, which implies that each such edge also belongs to G_k and hence forming a perfect matching in G_k. This contradicts the assumption that G_k does not have a perfect matching. Thus, the cost of any allocaiton must be more than k. □

A direct corollary of the above claims is the following.

Corollary 1 Assume that there exists an optimal allocation. Then, cost of the optimal allocation is the smallest allocation k such that G_k has a perfect matching.

Given the above claims, the strategy of our algorithm is simple. We find using a binary search, the smallest k (if any) such that the graph G_k has a perfect matching. Once we find this, we output the allocation that is naturally defined by this matching.

The following pseudocode executes this strategy.

Algorithm 2 OPT-ALC

```
Construct the graph $G$ as defined above.
if $\overline{G_{2n}}$ does not have a perfect matching then
    Output No.
else
    Let $L$ be a sorted list of the edges in $G$, in increasing order according to weight.
    $k \leftarrow$ BSEARCH-MATCH$(1, n, G, L)$.
    if $k \neq 0$ then
        Compute a perfect matching $M$ in $\overline{G_k}$.
        Ouput allocation defined by $M$.
    end if
end if
```

Algorithm 3 BSEARCH-MATCH(i,j,G,L)

```
if $i = j$ then
    Construct the graph $G_i$ defined above.
    if $\overline{G_i}$ has a perfect matching then
        return $i$
    else
        return $0$
    end if
else
    $k \leftarrow \lfloor \frac{i+j}{2} \rfloor$.
    Construct the graph $G_k$ defined above.
    if $\overline{G_k}$ has a perfect matching then
        return BSEARCH-MATCH(i,k,G,L)
    else
        return BSEARCH-MATCH(k+1,j,G,L)
    end if
end if
```

We now analyze the run-time of the algorithm OPT-ALC. The first step to construct G takes time $O(mr)$ since there are m tourists and for each tourist we compute the time taken to reach at most r hotels. Constructing the list L requires $O(mr\log(mr))$ time using any standard sorting algorithm. Further, note that given access to L, constructing G_k for any k can be done in time $O(mr)$. The BSEARCH-MATCH does a standard binary search in the range $1, \ldots, 2n$ and thus makes at most $O(\log n)$ recursive calls. The work done in each call is computing G_k which can be done in time $O(mr)$ and checking for a perfect matching which can be done in time $O(m^2r)$ using the Ford-Fulkerson algorithm (recall that the running time of the bipartite-matching algorithm on a bi-partite graph with α nodes on each side and β edges is $O(\alpha\beta)$). Thus, if $T(n)$ denotes the run-time for BSEARCH-MATCH(i,j,G,L) for any $j = i + n$, then $T(n) = T(n/2) + O(m^2r)$, which yeilds that $T(n) = O(m^2r\log(n))$. Thus overall, the running time of OPT-ALC is $O(mr + mr\log(mr) + m^2r\log n + m^2r) = O(m^2r\log n)$.

The correctness of the procedure follows directly by combining the claim below with Corollary 1.

Claim 2 For all $i \leq j$, the algorithm BSEARCH-MATCH(i,j,G,L) outputs the smallest integer $\ell \in \{i, \ldots, j\}$ such that $\overline{G_\ell}$ has a perfect matching, and outputs 0 if no such k exists.

Proof We prove this by induction on $n = j - i$. The base case when $n = 0$ is trivially true. Now suppose that the claim is true for all $i \leq j$ such that $j - i < n$, and we prove the claim for all $i < j$

such that $j - i = n$, where $n > 0$. Let $k = \lfloor \frac{i+j}{2} \rfloor$. Clearly, if $\overline{G_k}$ has a perfect matching, then $i \leq \ell \leq k$, and thus our output is correct using the induction hypothesis on BSEARCH-MATCH(i,k,G,L). The case when $\overline{G_k}$ does not have a perfect matching, either $k < \ell \leq j$ or there is no G_r that has a perfect matching for $i \leq r \leq j$. In either case, the correctness of our algorithm follows by using the induction hypothesis on BSEARCH-MATCH(k+1,j,G,L). □

The correctness of the algorithm OPT-ALC now follows by using the above claim with $i = 1$ and $j = n$, and involing Corollary 1.