CS60007 Algorithm Design and Analysis 2018
Assignment 2

Palash Dey and Swagato Sanyal
Indian Institute of Technology, Kharagpur

Please submit the solutions of problem 1, 5, 6, and 20. The deadline is September 4, 2018 in the class.

General Instructions

- ▷ Please prove correctness of every algorithm you design.
- ▷ Please compute running time of every algorithm you design.
- ▷ If not explicitly specified otherwise, please assume that the graphs under consideration are finite, weighted, and directed graphs which does not contain any self loop.

Notation

- ▷ $\mathbb{N} = \{0, 1, 2, 3, ...\}$ is the set of natural numbers. $\mathbb{R}$ is the set of real numbers.
- ▷ For $n \in \mathbb{N}$, $[n] = \{i \in \mathbb{N} : 1 \leqslant i \leqslant n\}$.
- ▷ For $a, b \in \mathbb{R}$ with $a < b$, we define $[a, b] = \{x \in \mathbb{R} : a \leqslant x \leqslant b\}$, $[a, b) = \{x \in \mathbb{R} : a \leqslant x < b\}$, $(a, b] = \{x \in \mathbb{R} : a < x \leqslant b\}$, $(a, b) = \{x \in \mathbb{R} : a < x < b\}$.

This is a preliminary version and may contain errors. Please send an email to the instructors if you find any error. Thank you for your cooperation.

---

1. [10 Marks] Let $\mathcal{G}$ be a different kind of graph where we have weights on vertices instead of weights on edges. Let us define the weight of a path to be the sum of the weights of the vertices on the path. Can there exist any algorithm for the single source shortest path problem in this graph? If there exists any such algorithm, describe it. Otherwise prove why there cannot be any such algorithm.

   Proof. The following algorithm solves this problem. Let $\mathcal{G}$ be any input graph with weight function $w : \mathcal{V}[\mathcal{G}] \longrightarrow \mathbb{R}$. We construct a weighted graph $\mathcal{H}$ from $\mathcal{G}$ as follows:

$$\begin{aligned}
\mathcal{V}[\mathcal{H}] &= \{\ell_u, r_u : u \in \mathcal{V}[\mathcal{G}]\} \\
\mathcal{E}[\mathcal{H}] &= \{(r_u, \ell_v) : (u, v) \in \mathcal{E}[\mathcal{G}]\} \\
&= \{(\ell_u, r_u) : u \in \mathcal{V}[\mathcal{G}]\} \\
w'(e) &= \begin{cases} w(v) & \text{if } e = (\ell_u, r_u) \in \mathcal{E}[\mathcal{H}] \text{ for some } u \in \mathcal{V}[\mathcal{G}] \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

   We now claim that for any two vertices $u, v \in \mathcal{V}[\mathcal{G}]$, there is a path from u to v in $\mathcal{G}$ of weight λ if and only if there is a path from ℓ_u to r_v in $\mathcal{H}$ of weight λ. To prove the claim, in one direction, let us assume that there is a path $u_0(= u), u_1, \ldots, u_t(= v)$ from u to v in $\mathcal{G}$ of weight $\sum_{i=0}^{t} w(u_i)$. Then the path $\ell_{u_0}, r_{u_0}, \ell_{u_1}, r_{u_1}, \ldots, \ell_{u_t}, r_{u_t}$ is a path from ℓ_u to r_v in $\mathcal{H}$ of weight $\sum_{i=0}^{t} w(u_i)$. In the other direction, let there be a path $u_0(= \ell_u), u_1, \ldots, u_t(= r_v)$ from ℓ_u to r_v in $\mathcal{H}$. Since the only outgoing edge on any vertex $\ell_x, x \in \mathcal{V}[\mathcal{G}]$, is to r_x and the only incoming edge on any vertex $r_y, y \in \mathcal{V}[\mathcal{G}]$, is from ℓ_y, we conclude that t is an even integer and for every $j \in [t/2]$ we have $u_{2j-1} = \ell_{z_j}$ and $u_{2j} = r_{z_j}$ for some $z_j \in \mathcal{V}[\mathcal{G}]$. Hence, there is a path $z_1(= u), z_2, \ldots, z_{t/2}(= v)$ from u to v in $\mathcal{G}$ of weight $\sum_{i=1}^{t/2} w(z_i) = \sum_{i=1}^{t/2} w'(\ell_{z_i}, r_{z_i})$ which proves the claim. Hence our problem reduces to the shortest path problem and thus can be solved in polynomial time. □

2. Design an algorithm to find, for all pairs of vertices u and v, the number of paths from u to v.

3. Reduce the following generalization of the maximum flow problem to the basic maximum flow problem.

   (a) **(Vertex capacities)** In this generalization, every vertex v also have a capacity $c(v)$. A feasible flow in this setting is a flow which satisfies the following constraint (which says that flow into any vertex can be at most its capacity) in addition to non-negativity, capacity, and conservation constraints:

$$\sum_{(u,v)\in E} f(u, v) \leqslant c(v) \text{ for every vertex } v$$

   The problem is to find a maximum flow in this setting.

(b) **(Multiple source-sink pairs)** In this generalization, we are given multiple source-sink pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ for some positive integer k. Everything remains same from the basic flow problem except we now do not want flow conservation property to hold at any vertex in $\{s_i, t_i : i \in [k]\}$. The problem is to find a maximum flow in this setting.

(c) **(Demands on edges)** In this generalization, we are also given a demand $d(e)$ for every edge e. A feasible flow in this setting is a flow which satisfies the following constraint (which says that flow in any edge is at least its demand) in addition to non-negativity, capacity, and conservation constraints:

$$f(e) \geqslant d(e) \text{ for every edge } e$$

The problem is to find if there exists a feasible flow in this setting.

(d) **(Supplies and demands on nodes)** In this generalization, we are also given a demand $d(v)$ for every vertex v. A feasible flow in this setting is a flow which satisfies the following constraint in addition to non-negativity and capacity:

$$\sum_{(u,v)\in E} f(u,v) - \sum_{(v,w)\in E} f(v,w) = c(v) \text{ for every vertex } v$$

The problem is to find if there exists a feasible flow in this setting.

4. Consider the following problem. You are given a flow network with unit-capacity edges: It consists of a directed graph $G = (V, E)$, a source $s \in V$, and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter k. The goal is to delete k edges so as to reduce the maximum $s - t$ flow in G by as much as possible. In other words, you should find a set of edges $F \subset E$ so that $|F| = k$ and the maximum $s - t$ flow in $G = (V, E \setminus F)$ is as small as possible subject to this. Give a polynomial-time algorithm to solve this problem. This problem is taken from [KT06].

5. [15 Marks] Prove using flows that the maximum number of edge disjoint $s - t$ paths in an unweighted graph $\mathcal{G}$ is the same as the size of minimum $s - t$ cut. This is known as Menger's Theorem.

Proof. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $s, t \in \mathcal{V}$ be any unweighted graph and $(\mathcal{U}, \mathcal{V} \setminus \mathcal{U})$ for some $\mathcal{U} \subset \mathcal{V}$ with $\mathcal{U} \neq \emptyset$ and $\mathcal{U} \neq \mathcal{V}$ be a minimum $s - t$ cut in $\mathcal{G}$. Any edge $(u, v) \in \mathcal{E}$ with $u \in \mathcal{U}$ and $v \in \mathcal{V} \setminus \mathcal{U}$ is called a $(\mathcal{U}, \mathcal{V} \setminus \mathcal{U})$-cut edge. Let $\mathcal{P}$ be any set of edge disjoint $s - t$ paths in $\mathcal{G}$. We observe that any path in $\mathcal{P}$ can use at most one $(\mathcal{U}, \mathcal{V} \setminus \mathcal{U})$-cut edge. Since $\mathcal{P}$ is any set of edge disjoint $s - t$ paths in $\mathcal{G}$, the maximum number of edge disjoint $s - t$ paths in $\mathcal{G}$ is at most the size of minimum $s - t$ cut.

To prove the other inequality, we construct the following flow network $\mathcal{H} = (\mathcal{V}, \mathcal{E}, s, t, c) : \mathcal{V}[\mathcal{H}] = \mathcal{V}[\mathcal{G}], \mathcal{E}[\mathcal{H}] = \mathcal{E}[\mathcal{G}], c(e) = 1$ for every $e \in \mathcal{E}[\mathcal{H}]$. Since all the capacities in $\mathcal{H}$ are integers (they are actually 1), there exists a maximum $s - t$

flow f which assigns integer flow values to every edge. However, since $c(e) = 1$ for every $e \in \mathcal{E}[\mathcal{H}]$, we have $f(e) = 0$ or $f(e) = 1$ for every $e \in \mathcal{E}[\mathcal{H}]$. Using flow decomposition theorem, there exists a collection $\mathcal{Q}$ of edge disjoint $s - t$ paths in $\mathcal{G}$ with $|\mathcal{Q}| = \text{val}(f) =$ size of minimum $s - t$ cut. Hence, the maximum number of edge disjoint $s - t$ paths in $\mathcal{G}$ is at least the size of minimum $s - t$ cut which proves the statement. □

6. [10+2 Marks] Prove that the number of minimum cuts in an n vertex graph is at most $\binom{n}{2}$. Given an example of a graph where the number of minimum cuts is $\binom{n}{2}$.

   Proof. For any graph $\mathcal{G}$, let the minimum cuts be $\mathcal{U}_i \subset \mathcal{V}[\mathcal{G}], i \in [\ell]$. We need to prove that $\ell \leqslant \binom{n}{2}$. Let us denote the event that the Karger's algorithm outputs $\mathcal{U}_i$ by E_i for $i \in [\ell]$. From the analysis of Karger's algorithm, we know the following.

$$\Pr\left[E_i\right] \geqslant \frac{2}{n(n-1)}$$

   We now observe that the events $E_i, i \in [\ell]$ are mutually exclusive. Hence, we have the following.

$$\Pr\left[\bigcup_{i \in [\ell]} E_i\right] = \frac{2\ell}{n(n-1)}$$

   Since probability of any event is at most 1, we have $\ell \leqslant \binom{n}{2}$. □

7. Let $\mathcal{G}$ be an undirected and weighted graph. For any two vertices $u, v \in \mathcal{V}$ with $u \neq v$, a pair $(\mathcal{A}, \mathcal{V} \setminus \mathcal{A})$ is called a $u - v$ cut if $u \in \mathcal{A}$ and $v \notin \mathcal{A}$. The capacity of $(\mathcal{A}, \mathcal{V} \setminus \mathcal{A})$ is the number of edges in $\mathcal{G}$ with one end point in $\mathcal{A}$ and other end point not in $\mathcal{A}$. Let us denote the capacity of a minimum $u - v$ cut by $f(u, v)$. Then prove the following:

   (a) For every $u, v, w \in \mathcal{V}$, we have

$$f(u, v) \geqslant \min\{f(u, w), f(w, v)\}$$

   (b) For every $u, v, w_1, w_2, \ldots, w_r \in \mathcal{V}$, we have

$$f(u, v) \geqslant \min\{f(u, w_1), f(w_1, w_2), f(w_2, w_3), \ldots, f(w_r, v)\}$$

8. Give an algorithm to find the number of vertex disjoint paths in a directed graph between two given nodes.

9. Prove or disprove the following statements:

(a) If all capacities are even integers then there exists a maximum $s-t$ flow f such that $f(e)$ is even for every $e \in \mathcal{E}$.

(b) If all capacities are odd integers then there exists a maximum $s-t$ flow f such that $f(e)$ is odd for every $e \in \mathcal{E}$.

10. Let $\mathcal{G}$ be a graph with integral capacities. Let f be a maximum flow in $\mathcal{G}$ and e be any edge in $\mathcal{G}$.

    (a) We increase the capacity of e by 1. Let the resulting graph be $\mathcal{H}_1$.

    (b) We decrease the capacity of e by 1. Let the resulting graph be $\mathcal{H}_2$.

    Design a $\mathcal{O}(m+n)$ algorithm to compute a maximum flow in $\mathcal{H}_1$ and $\mathcal{H}_2$.

11. Prove or disprove the following statement: if the capacities of all the edges are rational numbers in a graph, then Ford-Fulkerson algorithm terminates.

12. Construct graphs with following properties:

    (a) Exponentially (as function of n) many minimum cuts and a unique maximum flow.

    (b) Many maximum flows and a unique minimum cut.

    (c) Many maximum flows and many minimum cuts.

13. Prove or disprove the following statement: Let $(A, \mathcal{V} \setminus A)$ be a minimum $s-t$ cut in a graph $\mathcal{G}$. We now increase the capacity of every edge in $\mathcal{G}$ by 1. The resulting graph be $\mathcal{H}$. Then $(A, \mathcal{V} \setminus A)$ is a minimum $s-t$ cut in $\mathcal{H}$ too.

14. König's Theorem states that the size of maximum matching in a bipartite graph is the same as the size of a minimum vertex cover. Prove König's Theorem using flow arguments.

15. Give an example of a network where Ford-Fulkerson algorithm does not even converge to some maximum flow.

16. **(Flow Decomposition)** Let f be a flow in a graph $\mathcal{G}$. Then prove that there exists feasible flows $f_1, f_2, \ldots, f_k$ and $s-t$ paths $p_1, p_2, \ldots, p_k$ with the following properties:

    (a) $k \leqslant m$

    (b) $val(f) = \sum_{i=1}^{k} val(f_i)$

    (c) For every $i \in [k]$, $e \in E[\mathcal{G}]$, $f_i(e) > 0$ if and only if e belongs to p_i

    Design an efficient algorithm to find a flow decomposition of a given flow.

17. Recall the linear time algorithm for finding any order statistic done in the lecture. Remember that to compute the median we partitioned the inputs into blocks of size 5? Is the choice of 5 important? What is the complexity when the block size is chosen to be (a)3, (b)7? If you claim that the complexity is super-linear for a particular choice of block size, argue that there exists an input that forces the algorithm to run in super-linear time.

18. Suppose that an algorithm uses only comparisons to find the i-th smallest element in a set of n elements. Show that it can also find the $i-1$ smaller elements and the $n-i$ larger elements without performing any additional comparisons.

19. Describe an $O(n)$-time algorithm that, given a set S of n distinct numbers and a positive integer $k \leqslant n$, determines the k numbers in S that are closest to the median of S.

20. You are given as input n distinct numbers $x_1, \ldots, x_n$ and their respective weights $w_1, \ldots, w_n$. The weights are positive real numbers and they add up to 1, i.e., $\sum_{i=1}^{n} w_i = 1$. The weighted lower median (wlm) is defined to be x_k where $k \in \{1, \ldots, n\}$ is the unique index that satisfies

    (a) $\sum_{x_i < x_k} w_i < \frac{1}{2}$, and

    (b) $\sum_{x_i > x_k} w_i \leqslant \frac{1}{2}$.

    (i) [3 marks] prove that wlm is well-defined, i.e., for each input $x_1, \ldots, x_n$ and $w_1, \ldots, w_n$ there is a unique k for which the above two conditions (a) and (b) are satisfied.

    (ii) [5 marks] Give an algorithm that computes wlm using sorting and runs in worst-case $O(n \log n)$ time.

    (iii) [7 marks] Suppose you are given access to an algorithm MEDIAN that takes in n distinct numbers $x_1, \ldots, x_n$, computes their (lower) median, and runs in $O(n)$ time (we have seen such an algorithm in the lecture). Using MEDIAN as a black-box [1] design an algorithm that computes wlm and runs in worst-case $O(n)$ time.

Solution:

(i) First we prove that there cannot be two different values of k that satisfy conditions (a) and (b). Towards a contradiction, assume that k_1 and k_2 are two different values of k that satisfy conditions (a) and (b). Without loss of generality, assume that $x_{k_1} < x_{k_2}$. We thus have,

[1] your algorithm is allowed to call MEDIAN on various inputs, but is not allowed to modify MEDIAN any any way.

i. $\sum_{x_i \leqslant x_{k_1}} w_i \leqslant \sum_{x_i < x_{k_2}} w_i < \frac{1}{2}$, and

ii. $\sum_{x_i > x_{k_1}} \leqslant \frac{1}{2}$.

Adding i and ii we have that $\sum_{i=1}^{n} w_i < 1$ which is a contradiction.

Now we prove that there exists a value of k that satisfies properties (a) and (b). To this end, arrange the x_i's in increasing order. Let the order be $x_{j_1} < \ldots < x_{j_n}$. Let p be the largest number for which the condition $\sum_{x_i < x_{j_p}} w_i < \frac{1}{2}$ is satisfied. Thus choosing $k = j_p$ satisfies property (a). We will prove that $k = j_p$ also satisfies property (b). Towards a contradiction, assume that $\sum_{x_i > x_{j_p}} w_i > \frac{1}{2}$. By the choice of p we also know that $\sum_{x_i \leqslant x_{j_p}} w_i = \sum_{x_i < x_{j_{p+1}}} w_i \geqslant \frac{1}{2}$. Adding the two inequalities we have that $\sum_{i=1}^{n} w_i > \frac{1}{2} + \frac{1}{2} = 1$, which is a contradiction.

(ii) Sort the x_i's in ascending order. Let the sorted sequence be $x_{j_1} < \ldots < x_{j_n}$. Recall from part (i) that wlm is x_{j_p} for the largest p that satisfies $\sum_{x_i < x_{j_p}} w_i < \frac{1}{2}$. This can be found in linear time by scanning the sorted list and maintaining the cumulative sums of w_i's. Time complexity of the algorithm is dominated by the time complexity of sorting which is $O(n \log n)$.

(iii) We will write an algorithm WLM whose input is a set of x_i's, their corresponding y_i's, and a threshold ℓ. On such an input, WLM returns the largest x_k for which the condition $\sum_{x_i < x_k} w_i < \ell$ holds. Note that by part (i), the required wlm is the output of the algorithm WLM on input $x_1, \ldots, x_n, w_1, \ldots, w_n, \frac{1}{2}$. Let $T(t)$ be the

Algorithm 1 WLM $(x_1, \ldots, x_t, w_1, \ldots, w_t, \ell)$

```
if t = 1 then
    return x_1.
end if
m ← MEDIAN(x_1, ..., x_t).
L ← {i : x_i ⩽ m}.
R ← {1, ..., t}\L.
S ← Σ_{i∈L} w_i.
if S < ℓ then
    return WLM((x_i, w_i : i ∈ R), ℓ−S).
else
    return WLM((x_i, w_i : i ∈ L), ℓ).
end if
```

time taken by WLM on this input. Since m is the (lower) median of $x_1, \ldots, x_t$, we have that $|L|, |R| \leqslant \frac{t+1}{2}$. MEDIAN runs in linear time. The partitioning of the x_i's into L and R can be done in linear time (recall quicksort). Computing S takes time linear in $|L|$. We thus have $T(t) \leqslant T(\frac{t-1}{2}) + O(t)$ which implies that $T(t) = O(t)$.

References

[KT06] Jon Kleinberg and Eva Tardos. Algorithm design. Pearson Education India, 2006.