CS 217 – Algorithm Design and Analysis

Shanghai Jiaotong University, Fall 2019

Handed out on Thursday, 2019-10-10
First submission and questions due on Monday, 2019-10-14
You will receive feedback from the TA.
Final submission due on Monday, 2019-10-21

4 Bottleneck Paths

Let $G = (V, E)$ be a directed graph with an edge capacity function $c : E \to \mathbb{R}^+$. For a path $p = u_0 u_1 \dots u_t$ define its *capacity* to be

$$c(p) := \min_{1 \leq i \leq t} c(\{u_{i-1}, u_i\}) \ . \tag{1}$$

> **Maximum Capacity Path Problem (MCP).** Given a directed graph $G = (V, E)$, an edge capacity function $c : E \to \mathbb{R}^+$, and two vertices $s, t \in V$, compute the path p^* maximizing $c(p)$. We denote by p^* the optimal path and by $c^* := c(p^*)$ its cost.

Exercise 1. Suppose the edges $e_1, \dots, e_m$ are sorted by their cost. Show how to solve MCP in time $O(n + m)$.

Proof. Design a algorithm following Pseudocode shows(Suppose the edges are sorted decreased, See in Algorithm 1):

And then we think about the correctness and complexity.

The algorithm means we can enum the answer. When we find a edge between the point visited and not visited, we can go through all the edges which's costs is higher than this edge. If now s is connected to t, means this edge is the largest edge while going through all edges higher than it from s to t. That fits the answer we want.

Now, let's think about the complexity. For every node, it may be in queue at least once. And for every edge, it may be in G' and used in bfs at least once. So the time complexity is $O(n + m)$.

Algorithm 1 Solve MCP in time $\Theta(n+m)$ with all the edges sorted by their cost

```
procedure MCP(G, s, t)
    visited[s] = True
    G' = NULL
    for e ∈ G do
        if visited[e.from]&&!visited[e.to] then
            bfs_graph(e.to, G')
        else
            G'.add_edge(e)
        end if
        if visited[t] then
            return e.weight
        end if
    end for
end procedure
procedure BFS_GRAPH(s, G)
    visited[s] = True
    queue.push(s)
    while !queue.empty() do
        top = queue.top()
        queue.pop()
        for e ∈ G[top] do
            if !visited[e.to] then
                visited[e.to] = True
                q.push(e.to)
            end if
        end for
    end while
end procedure
```

□

Exercise 2. Give an algorithm for MCP of running time $O(m \log \log m)$. **Hint:** Using the median-of-medians algorithm, you can determine an edge e such that at most $m/2$ edges are cheaper than e and at most $m/2$ edges are more expensive than e. Can you determine, in time $O(n+m)$, whether $c^* < c(e)$, $c^* = c(e)$, or $c^* > c(e)$? Iterate to shrink the set of possible values for c^* to $m/4$, $m/8$, and so on.

Proof. As is shown by the hint, during each iteration, we can shrink the set of possible values of c^*, i.e.

$$E_1 = \{e \mid c\left(e\right) \leq M, e \in E'\}, E_2 = \{e \mid c\left(e\right) > M, e \in E'\}$$

We we can find a path in E_2, then the lower bound L of c^* can be updated to M. Since if $c^* > L$, we must have a path e with all the edges larger then L.

Otherwise, the upper bound U will be M as there is no such path with all edges larger than M. However, if we take iterations until $L = U$, we will have $O\left(m \log m\right)$ running time. So we only do $\log(s\left(m\right))$ iterations. Here s is a place holder to decide later

Consider the following algorithm

Algorithm 2 Solve MCP in time $n \log \log n$

---

while $t < \log s\left(m\right)$ **do**

  Determine the median of $\{e \mid e \in E', c\left(e\right) \leq U\}$.

  if **then** (V, E_2) is s- t connected

    $E' \leftarrow E_2, L = M,$

  else

    $U = M$

  end if

end while

Number t edges in set $\{e \in E' \mid c\left(e\right) \leq U\}$ according to increasing order $e_1, e_2, \ldots$.

Solve instance by **Algorithm 1** with the following ordering

$$l\left(e\right) = \begin{cases} 1, & c\left(e\right) \leq L \\ i, & \exists i, e = e_i \\ t, & c\left(e\right) > U \end{cases} .$$

Now we prove this algotihm is $n \log \log n$, just notice that

$$t = O\left(\frac{m}{s\left(m\right)}\right).$$

Since every iteration we make the set $\{e \mid e \in E', c\left(e\right) < U\}$ shrinks to $\frac{1}{2}$. That is the size $t = |E'| \leq \frac{m}{2^{\log s(m)}}$.

Thus the running time is

$$\log\left(s\left(m\right)\right) \cdot m + t \log t + t$$

considering the sorting of the t edges plus the running time of **Algorithm 1** , now we choose $s\left(m\right) = \log m$ to minimize the time which is

$$m\left(\frac{\log m}{s} - \frac{\log s}{s}\right).$$

Hence

$$T = O\left(\frac{m}{\log m} \log\left(\frac{m}{\log m}\right)\right) + O\left(\log m\right) = O\left(m \log \log m\right).$$

□

Exercise 3. Give an algorithm for MCP that runs in time $O(m \log \log \log m)$? How about $O(m \log \log \log \log m)$? How far can you get?

Proof. We can have a better algorithm runs in $O\left(m \log \log \log m\right)$, since there is no need to sort the t edges. In **Algorithm 1** we just choose edges from large to small one by one, hence we can just use a complex priority queue[1] with running time of insertion $O\left(\log \log n\right)$ to maintain $\{e \mid e \in E', c\left(e\right) \leq U\}$.

But can we do better? I think it depends on how fast the sorting or priority queue could run, and currently there seems no better result. □

[1]only supoort `insert` and `find_max`, `delete_max` actions, check this paper for detail: `On RAM priority queues`.