CS 217 – Algorithm Design and Analysis

Shanghai Jiaotong University, Fall 2019

Handed out on Friday, 2020-04-17
First submission and questions due on Friday, 2019-04-24
You will receive feedback from the TA.
Final submission due on Tuesday, 2020-05-01

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

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

> **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.

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

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 $O(m \log m)$ running time. So we only do $\log(s(m))$ iterations. Here s is a place holder to decide later

Consider the following algorithm

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

$i = 0, U = \infty, L = 0$
while $i < \log s(m)$ **do**
  Determine the median of $\{e \mid e \in E', c(e) \leq U\}$.
  if (V, E_2) is s- t connected **then**
    $E' \leftarrow E_2, L = M,$
  else
    $U = M$
  end if
  $i = i + 1$
end while
Number t edges in set $\{e \in E' \mid c(e) \leq U\}$ according to increasing order $e_1, e_2, \ldots$.
Solve instance by **Algorithm 1** with the following ordering, i.e. the place of e in the sorted array

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

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

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

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

Thus the running time is

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

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

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

Hence

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

□

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 $O(m \log \ldots \log m)$ (arbitrary number of log),

Consider a set of edge E with $\frac{m}{k}$ elements. We apply **median of median** algorithm k rounds. Each round we break one block into 2 parts. For example, in the first round, we find the median M of E, and break E into

$$E_1 = \{e \mid c(e) < M, e \in E\}, E_2 = \{e \mid c(e) \geq M, e \in E\}.$$

And in the second round, we can obtain 4 blocks by apply MOM to E_1, E_2. Finally we have 2^k blocks $A_1, \ldots, A_{2^k}$, while each block has $\frac{m}{k2^k}$ elements. Plus by the property of median

$$\forall i < j, \forall e_1 \in A_i, e_2 \in A_j, c(e_1) < c(e_2).$$

Hence we can apply the idea we used in **Exercise 2**. That is,

Algorithm 3 The key idea to solve MCP in time $n \log \ldots \log n$

```
E' = ⋃_{i≤2^k} A_i, L = 0, U = max(E)
for i in range(2^k) do
    if (V, A_i) is s- t connected then
        E' ← A_i, L = min(A_i)
    else
        U = min(A_i)
    end if
end for
```

Hence our problem is reduced to a edge set E' with whose size is $O\left(\frac{m}{k2^k}\right)$, we call this one iteration. And this costs

$$T = \sum_{i=1}^{k} \sum_{j=1}^{i} O\left(\frac{m}{k2^i}\right) = kO\left(\frac{m}{k}\right) = O(m).$$

And let $f(k) = k2^k$, assume that after r iterations, we can decide c^*, we have

$$f^{r-1}(1) < m.$$

Which leads that $m \geq g^{r-1}(1)$, in which $g(x) = 2^x$. Hence r is the smallest integer such that $1 \leq \log\log\ldots\log m$ with r log, the overall running time is $O(rm)$. Actually we can prove $\forall s$, $\exists m$

$$r < \log\log\ldots\log m \,(\text{with } s \log).$$

Otherwise $1 < \log\ldots\log r$ with $r - s$ log. Hence $r > f^{r-s}(1)$, which leads to a contradiction since m and r can be arbitrarily large but s is fixed. Hence the running time could be written as $O(m \log\log\ldots\log m)$ (arbitrary number of log). □