CS217 – Algorithm Design and Analysis
Homework 4

Not Strong Enough

April 19, 2020

1

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

Solution. The algorithm is executed in iterations. Suppose that $c(e_1) \geq c(e_2) \geq \cdots \geq c(e_m)$. In iteration i, consider e_i.

There is a set of vertices which is reachable from s, and the initial status of the set is $\{s\}$. Each vertex has an predecessor recording the predecessor in the path by which s reaches the vertex. Its initial status is *null*. Each vertex also has a set of unresolved edges whose start point is the vertex. Its initial status is $\emptyset$.

In each iteration, denote the edge we are handling by $e = (u, v)$.

If u is not reachable, simply add e into the unresolved edge set of u.

If u, v are already reachable, nothing needs to be done.

If u is already reachable but v is not, set v reachable and set its predecessor to u. Then handle all unresolved edges of v. (Let the end point of each edge be reachable if it is not.) Since this may introduce new reachable vertices, handle them too. This procedure is like a BFS.

Once t is reached, the algorithm terminates, and the cost of the newly introduced edge is the cost of MCP. Using the predecessor of each vertex, the maximum capacity path from s to t can be generated. □

First prove that each vertex in the reachable set is really reachable(so it is well-defined):

Proof. The reachable set has the property that, except for s, all vertices in the set have a predecessor also in the set.

This is trivial since when we add a vertex to the reachable set, we set its predecessor by a reachable vertex.

Using this property, since its predecessor is reachable, the vertex itself is reachable as well. □

Then prove that all reachable vertices using $\{e_1, \ldots e_n\}$ are in reachable set after iteration n:

Proof. Assume that there is a path from s to v using edges $e_{i_1}, e_{i_2} \ldots e_{i_k}, (i_1, i_2 \ldots i_k \in \{1, 2, \ldots n\})$. And $\forall j \in \{1, 2 \ldots k\}, e_{i_j} = (u_j, u_{j+1}); u_1 = s, u_{k+1} = v$. Let $M_j = \max\{i_1, i_2 \ldots i_j\}$, $m_j = \min\{i_1, i_2 \ldots i_j\}$

Assume that after iteration M_{q-1} $u_1, u_2 \ldots u_q$ are in reachable set and u_q is added in iteration M_{q-1}. If $i_q < M_{q-1}$, e_q is in unresolved set of u_q. So in iteration $M_{q-1} = M_q$, when adding u_q in reachable set, all unresolved edges of u_q are handled so u_{q+1} is added as well. If $i_q > M_{q-1}$, in iteration $M_q = i_q$, u_q is in reachable set so u_{q+1} is added in reachable set.

Consider the initial status that after iteration $M_1 = i_1$, $u_1 = s, u_2$ are in reachable set and u_2 is added in iteration M_1.

By induction, it is proved that after iteration M_k, $u_1, u_2 \dots u_{k+1}$ are all in reachable set. $\square$

Now it is trivial to prove the correctness of the algorithm:

If the cost of MCP is $c^* = e_n$, then the path only uses edges in $\{e_1, \dots e_n\}$, so exactly in iteration n, t is reachable.

Then prove the complexity of the algorithm is $O(n+m)$:

Proof. Consider an arbitrary edge $e_k = (u, v)$. If in iteration k, u is already reachable, it is not added into the unresolved set so it will not be visited. Otherwise, it is added into the unresolved set of u, and when u is added into the reachable set, it will then be visited.

Since u can only be added to reachable set once, e_k is visited at most twice. So the complexity is $O(m)$.

In the initial step, each vertex is initialized, so the complexity is $O(n)$.

Combine the two steps together, the total complexity is $O(n+m)$. $\square$

2

Give an algorithm for MCP of running time $O(m \log \log m)$.

Solution.

The pseudocode of the algorithm is in the next page.

The correctness of the algorithm comes from below. First, we initialize our c^* candidates to be all capacities of the edges in G. Then, in each iteration, we divide it into many divisions of approximately the same size in order, and test which division c^* is in. Finally, there will be only one candidate left, and the only candidate is c^*.

Now we'll analyze the running time. Note that using median-of-medians algorithm to find the median of a set S has running time $O(|S|)$. In "Loop 2" of function MCP, we find the median of $S_1, \ldots, S_k$ in 2^d iterations. Note that there is an invariant equation $\sum_{i=1}^{k} |S_k| = |C|$. So finding the median requires $O(|C| \cdot 2^d)$ in total. The GETDIVISION function may need to add all the edges into the graph in the worst case. Testing the connectivity will use $O(m)$ time in total if we record a set of reachable vertices as decribed in Exercise 1. So the running time of GETDIVISION is $O(m)$.

We now prove that $|C| \cdot 2^d = O(m)$ by induction. Initially, $|C| = m$ and $d = 0$. It obviously holds. Suppose $|C_{old}| \cdot 2^{d_{old}} = O(m)$. We have $|C_{new}| = O(|C_{old}| \cdot (\frac{1}{2})^{2^{d_{old}}})$, $d_{new} = d_{old} + 2^{d_{old}}$. So $|C_{new}| \cdot 2^{d_{new}} = O(m)$.

Now we know that a single iteration of "Loop 1" takes time $O(m)$. Note that $2^d \leqslant m$. Since d increases exponentially, "Loop 1" will iterate $O(\log^* m)$ times. Finally, we get the running time is $O(m \log^* m)$. Obviously this algorithm has a better running time than $O(m \log \log m), O(m \log \log \log m), O(m \log \log \log \log m)$ when m is sufficiently large. □

3

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

Solution. See Problem 2. □

Algorithm 1 An algorithm for MCP problem.

function MCP($V, E = \{e_1, e_2, \dots, e_m\}, c$)
    $C \leftarrow \{e_1, \dots, e_m\}$
    $oldEdges \leftarrow \emptyset$
    $d \leftarrow 0$
    while there are multiple capacities of the edges in C **do** ▷ Loop 1
        $divisions \leftarrow [C]$ ▷ Let *divisions* be a list only containing the set C.
        for $i = 1$ to 2^d **do** ▷ Loop 2
            Let $S_1, S_2, \dots, S_k$ denote the edge sets in *divisions* in order, i.e., representing *divisions* as $[S_1, \dots, S_k]$.
            Use median-of-medians algorithm k times to find the median capacity of the capacities of the edges in each set(if there are two medians, choose the larger one). Denote them as $m_1, m_2, \dots, m_k$ respectively.
            $newDivisions \leftarrow [\,]$ ▷ Let *newDivisions* be an empty list.
            for $j = 1$ to k **do** ▷ Loop 3
                Append $\{s \in S_j : c(s) \geqslant m_j\}$ to *newDivisions*.
                Append $\{s \in S_j : c(s) < m_j\}$ to *newDivisions*.
            end for
            $divisions \leftarrow newDivisions$
        end for
        $C, oldEdges \leftarrow$ GetDivision(*divisions*, *oldEdges*)
        $d \leftarrow 2^d + d$
    end while
    return the capacity of the edges in C
end function

function GetDivision(*divisions*, *oldEdges*) ▷ Using the algorithm of exercise 1.
    $G \leftarrow (V, oldEdges)$
    for *div* in *divisions* in the order of the list **do**
        Add all edges in *div* to G.
        if t is now reachable from s **then**
            return *div*, *oldEdges*
        else
            $oldEdges \leftarrow oldEdges \cup div$
        end if
    end for
end function