Homework 2

Haoyu Zhen

April 5, 2022

Problem 1

Let $T(n)$ be the numebr of all the bit operation we have done after n ADDs, e.g., $T(1) = 1$, $T(2) = 3$. $C_n \triangleq T(n) - T(n-1)$ and $S_k \triangleq T(2^k - 1)$. NB: $(2^k - 1)_2 = \underbrace{11\cdots 1}_{k}$.

Then we have

$$S_k = C_{2^{k-1}} + 2S_{k-1} = k + 2S_{k-1}$$

which entails

$$T(2^k - 1) = S_k = 2^{k+1} - k - 2 \quad \text{and} \quad T(2^k) = 2^k - 1.$$

Thus $\forall n \in \left\{2^{k-1}, \cdots, 2^k - 1\right\}$

$$C(1) + C(2) + \cdots + C(n) = T(n) < T(2^k) < 4n \Longrightarrow \text{LHS} = n\mathcal{O}(1).$$

Finally, the amortized cost of ADD is $\mathcal{O}(1)$.

Problem 2

(a). The counter-example:

$$\begin{tikzcd} & 2 \arrow[dr, "-2"] & \\ 1 \arrow[ur, "-1"] \arrow[rr, "-2"'] & & 3 \end{tikzcd}$$

The shortest path from 1 to 3 should be:$1 \to 2 \to 3$. While this algorithm will return $1 \to 3$ directly by $(C - 2) < (C - 1) + (C - 2)$ where $C > 2$ is the number added to each edge.

(b). Worked.
Trivially, $\forall$ path p form s to v_l, $p = (s, v_1, v_2, \cdots, v_l)$ where v_i belongs to layer i.
Let $p_0 = \arg\min_p$(p is the shortest path from s to v_l). Then

$$p_0 = \arg\min_p \left(l \times C + \sum_{\text{weight } w \text{ in } p} w \right) = \arg\min_p \left(\sum_{\text{weight } w \text{ in } p} w \right)$$

where RHS is what we want.

Problem 3

(a). I was inspired by the Hint from Chihao Zhang. Let s be the beginning vertex. The algorithm is as follows.

1. Construct a graph H having the same topology of G. In H, weight $w = rc_{uv} - p_v$ w.r.t. edge (u, v).
2. Determine if there is a negative loop that s could reach (implement SPFA or Bellman-Ford).
3. If 2. returns true, then $\exists C$: $\sum_{(u,v)\in C} rc_{uv} - p_v < 0$ which means that $r < r^*$. Otherwise $r \geq r^*$.

The complexity is $\mathcal{O}(|V||E|)$.

(b). For simplicy, ALG.a denotes the algorithm demonstrated in (a).

Algorithm 1 Find the cycle which has a good profit-to-cost ratio in the given graph

---

Input: A graph G
Output: A good enough cycle C.
  $left \leftarrow 0$, $right \leftarrow R$.
  while $right - left > \varepsilon$ **do**
    $r \leftarrow (right + left)/2$
    Implement ALG.a with the input: ratio r and graph G
    if $r > r^*$ **then**
      $left \leftarrow r$
    else
      $right \leftarrow r$
    end if
  end while
  Construct the graph H following the rules shown in (a). step 1 with $r = left$
  if H dose not has a negative loop (Under this case, $left = r^*$.) **then**
    We could reconstruct the graph H with $r = left - \varepsilon/2$
  end if
  Find a negative loop C in H contained s by Bellman-Ford (More details are in Appendix ALG.2)
  return C

Soundness of ALG.1: Obviously, $\forall left, right : left \leq r^* \leq right$. When the "while" operation ends, we have $r^* - \varepsilon \leq left \leq r^*$. Thus

$$\sum_{(u,v)\in C} (left \times c_{uv} - p_v) \leq 0 \Longrightarrow r^* - \varepsilon \leq left \leq r(C).$$

The complexity of ALG.1:

$$T(|V|, \varepsilon, R) = \mathcal{O}\left[|V||E| \log_2 \left(\frac{R}{\varepsilon}\right)\right] = \mathcal{O}\left[|V|^3 \log \left(\frac{R}{\varepsilon}\right)\right].$$

Problem 4

(a). Trivially, G only has 2 vertices. G: $1 - -2$ and G': $1 \longrightarrow 2$

(b). In G, we want to generate 2 paths p_1 and p_2 such that they share no edge, i.e., $\forall(u_1, v_1)$ in p_1 and (u_2, v_2) in p_2, $(u_1, v_1) \neq (u_2, v_2)$. We do it by following steps:

1. In G', there exist 2 paths p_1: $u \to v$ and p_2: $v \to u$.
2. If p_1, p_2 have the same edge (u', v'), go to step.3.
3. Assume that $p_1 = (u \Rightarrow u' \to \underline{v' \Rightarrow v})$ and $p_2 = (\underline{v \Rightarrow u'} \to v' \Rightarrow u)$ where $a \to b$ implies a, b are adjacent and $a \Rightarrow b$ represents a path from a to b.
4. Then update: $p_1 = (u \Rightarrow \underline{u' \Rightarrow v})$ and $p_2 = (\underline{v \Rightarrow v'} \Rightarrow u)$. Here $u' \Rightarrow v$ is the inverse of $v \Rightarrow u'$ and $v \Rightarrow v'$ is the inverse of $v' \Rightarrow v$.
5. Repeat step 2 until p_1 and p_2 share no edge.

Trivially, p_1 and p_2 is what we want. Thus, forall u, v in V, there are 2 paths from u to v and they share no edge in $G \implies$ removing any single edge from G will still give a connected graph.

(c). From a root r we get a tree T generated by DFS.

Lemma 1. *Suppose that a vertex u has several subtrees $T_1, T_2, \cdots, T_k$. And we define: T_i is strongly connected in G if $\forall u, v \in$, u could reach v and v could reach u in G. Then:*

$$\forall i,\ T_i \text{ is strongly connected in } G \implies T_u \text{ is strongly connected in } G$$

where T_u is the G's subtree with root u.

Proof. We only need to prove $\forall i$, $\exists$ vertex v in T_i, (v, u') is a back edge in E where u' is the ancestor of u or $u = u'$. If not, assume T_j does not satisfy this property. Then we cut the edge "e" from u to T_j. Noe we use the property of tree generated by DFS: "if $(v_1, u_1) \in E$ where v_1 is in T_j and u_1 is not in, then u_1 is the ancestor of T_j." Now we have there is no edge from T_j to u. Contradiction. □

Here we use induction:
every leaf is strongly connected in G. Then by lemma.1, the whole "tree" is connected in G, which means G' is strongly connected.

(d). Assume that G is connected (or it is trivial). we design the algorithm intuitively,

1. $n \leftarrow 0$
2. Generate G' through the given rule. (Easy to implement)
3. Analyse G' using algorithm in "Slide-06 Page §127 Super Plan" to find all SCCs
4. For every $(u, v) \in E_{G'}$, if u, v belong to different SCCs, then $n \leftarrow n + 1$

The correctness of this algorithm holds naturaly because SCCs in Step.2 is a tree which only has tree edges and back edges.
The complexity is $\mathcal{O}(|V| + |E|)$.

Problem 5

Get the smallest number n by the following steps:

1. Find all the SCCs in G, regard every of them as a node and keep the edges between different SCCs. Then we get a DAG: G'.

2. Separate G' into k isolated component $C_1, C_2, \cdots, C_k$ such that there is no edges between different components and C_i is weakly connected.

3. $T_i \triangleq$ number of tail vertex in C_i and $H_i \triangleq$ number of head vertex in C_i.

4. $n = k - 1 + \sum_{i=1}^{k} \max(T_i, H_i)$.

Reference: Question.12681785, StackOverflow.

Problem 6

About 15 hours. Difficulty 4. Talked with Yilin Sun and Wei Jiang.

Appendix

How to find a negative loop in graph H? Here's the algorithm:

Algorithm 2 Find the cycle which has a negative loop in the given graph

Implement Bellman-Ford algorithm, during which we record the parents for every updated vertex.
In Bellman-Ford algorithm, vertex v has been traversed for $|V|$ times.
Initiate an array: $\{a_i\} \leftarrow 0$.
while $\forall i$, $a_i \le 1$ **do**
  $a_v \leftarrow a_v + 1$
  $v \leftarrow v$'s parent
end while
Suppose that $a_t = 2$
return $(t$, t's parent, $\cdots$, $t)$