算法基础 2022 春 Homework 5

任课教师：陈雪 due: May 10, 23:59

（1）只准讨论思路，严禁抄袭

（2）只能阅读 bb 上的材料和教材算法导论。严禁网上搜寻任何材料，答案或者帮助

以下所有算法的设计请给出伪代码，证明算法正确性以及时间复杂度。

问题 1 (20 分). 用若干个先进后出的栈模拟一个先进先出的队列。

(a) 给出入队和出队操作的伪代码。

(b) 在上述队列中，初始时队列为空，对队列进行 n 次合法操作，每个操作为（a）中操作之一。用课上所学的三种分析方式中的任意一种，分析并证明该操作序列中每个操作的均摊代价。

Answer:

(a) The algorithm as follows.

Algorithm 1 queue algorithm

```
function ENQUEUE(queue, x)
    queue.in-stack.PUSH(x)
function DEQUEUE(queue)
    if queue.out-stack.ISEMPTY is true then
        while queue.out-stack.ISEMPTY is false do
            x = queue.in-stack.POP()
            queue.out-stack.PUSH(x)
    return queue.out-stack.POP()
```

(b) Choosing the potential method. Let $\phi(queue) = |in\text{-}stack|$, $\widetilde{cost}_{enqueue} = 2$ and $\widetilde{cost}_{dequeue} = 1$. So $\phi(queue) \geq 0$ at any time, then $\sum cost \leq \sum \widetilde{cost} \leq 2 \cdot n$. Therefore, each operating amortizes $\Theta(1)$.

问题 2 (30 分). 给定一棵树 $T = (V, E)$，在线性时间 $O(|V|)$ 内找到 T 中最长的简单路径。

(a) 使用动态规划设计算法

(b) 使用贪心算法设计算法

请给出伪代码，证明算法正确性以及时间复杂度。

Answer:

(a) Dynamic programming:

Let one of vertexes in tree T be the root arbitrarily, then the direction of the tree is also determined. If we have known the deepest path in a subtree, which ends with its root, we can calculate the longest path include root of the whole tree.

Algorithm 2 longest path algorithm

function LONGEST-PATH-RE($root$)
    if root has not a child **then**
        $root.depth = 1$
        $root.length = 1$
    else
        for $child$ of $root$ **do**
            LONGEST-PATH-RE($child$)
        $root.depth = \max\{child.depth\} + 1$
        if root only has one child **then**
            $root.length = root.depth$
        else
            find the The maximum two $depth$ d_1, d_2 in children
            $root.length = d_1 + d_2 + 1$
    return $\max\{root.length, max\{child.length\}\}$
function LONGEST-PATH($root$)
    $length =$ LONGEST-PATH-RE $(root)$
    Find a vertex has the $length$
    Output the longest path

The algorithm 2 evolves from BFS, so its time complexity is $O(|V|)$. There are two case for a vertex v in the Graph. In the fisth case, the longest path is ends with v. It infers v's degree is 1. If not, the path can keep on expending. For the last case, v is a intermediate point of the longest path.

(b) Greedy algorithm:

Algorithm 3 longest path algorithm

```
1: function LONGEST-PATH(G)
2:     start path P in a leaf u
3:     BFS to find the farthest vertex v from u
4:     BFS to find the farthest vertex u' from v
5:     return u' ~ v
```

The algorithm 3 also bases on BFS, so its time complexity is also $O(|V|)$. If v is a endpoint of a longest path, then line 4 can get the other endpoint of the path. So we just need prove that v is a endpoint of a longest path.

Assuming v isn't a endpoint of a longest path. Let the longest path be $p \sim q$, and a intermediate point $t \in p \sim q, (t \sim v) \cap (p \sim q) = t$. One of $p \sim t \sim v$ and $q \sim t \sim v$ would be as long as $p \sim q$ at least, which contradicts to assumption. Therefore, v is a endpoint of a longest path.

问题 3 (25 分)**.** 给定一个有向图 $G = (V, E)$，$V = v_1, v_2, \cdots, v_n$。对于 G 中的任意一个顶点 v_i，设 $R(v_i)$ 为从 v_i 可以到达的顶点集合（包括 v_i 本身）。定义 $min(v_i) = \min_{v_j \in R(v_i)} j, 1 \leq i \leq n$，即从 v_i 出发能到达的最小顶点。请给出一个时间复杂度为 $O(|V| + |E|)$ 的算法来计算 G 中所有顶点 v 的 $min(v)$。请给出伪代码，证明算法正确性以及时间复杂度。

Answer:

Algorithm 4 smallest reachable vertex algorithm

```
function LONGEST-PATH(G)
    G^SCG = STRONGLY-CONNECT-COMPONENTS(G)
    label each component in G^SCG with the smallest vertex in the component
```

4: $order$ = TOPOLOGICAL-SORT(G^{SCG})
5: **for** c in $order$ reversely **do**
6: label c with its smallest neighborhood's label
7: **for** v in G **do**
8: label v with its component's label

The complexity of STRONGLY-CONNECT-COMPONENTS and TOPOLOGICAL-SORT is $\Theta(|V|+|E|)$. And both for-loop is $O(|V|+|E|)$. So The complexity of the algorithm 4 is $\Theta(|V|+|E|)$.

$R(v_i)$ can be divided into several strong connnect components $S_1, S_2, \cdots, S_p$, so $\min(v_i) = \min\limits_{v_j \in R(v_i)} j = \min\limits_{1 \le k \le p} \min\limits_{v_j \in S_k} j$. And $R(v_j)$ is same for vertexes in a strong connnect components. line 3 get $\min\limits_{v_j \in S_k} j$ for all strong connnect components. Then line 8 calculate $\min\limits_{1 \le k \le p} \min\limits_{v_j \in S_k} j$ for all vertexes.

问题 4 (25 分)**.** 设 $G = (V, E)$ 为一个有向图，设计一个算法，在 $O(|V|+|E|)$ 时间内判断 G 中是否存在奇数长度的环路。请给出伪代码，证明算法正确性以及时间复杂度。

Answer:

Algorithm 5 odd cycle algorithm

function ODD-CYCLE(G)
  for each vertex $u \in G.V$ **do**
    u.color = WHITE
  for each vertex $u \in G.V$ **do**
    if u.color == WHITE and **odd-cycle-visit**$(G, u, 0)$ == True **then**
      return True
  return False
function ODD-CYCLE-VISIT($G, u, parity$)
  $u.parity = parity$
  u.color = GRAY
  for each vertex $v \in G : Adj[u]$ **do**
    if v.color == WHITE and **odd-cycle-visit**$(G, v, parity\ xor\ 1)$ == True **then**
      return True
    else if v.color == GRAY and v.color xor u.color == 0 **then**

            return True
    u.color = BLACK
    return False

The algorithm 5 evolves from DFS, so its time complexity is $O(|V| + |E|)$.

As we know, a undirected graph without odd-cycle is a bipartite graph. And we can dye a bipartite graph with two color. So algorithm 5 works in undirected graphs.

For directed graphs, we can also prove it. Fisrt, it's sufficient to tell G has a odd-cycle if **odd-cycle**(G) is ture. Because the result is ture when DFS find a odd-cycle. And it's necessary as well. If u is the fisrt visited vertex in a odd-cycle, DFS wolud return u after explore all cycle. And there must exists the fisrt visited vertex in a odd-cycle.

问题 5 (25 分)**.**

Answer:

Algorithm 6 limited spanning tree algorithm

function SPANNING-TREE(G, S)
    if $|V| \leq 2$ **then**
        return G
    else
        delete all edges that both of its endpoints is in S
        for v in S **do**
            delete all adjacent edges of v except the smallest one
        return PRIM(G) or KRUSKAL(G)

The algorithm 6's time complexity is same to PRIM or KRUSKAL.

If $|V| \leq 2$, it's trivial to find the minimum spanning tree of G. So the key is the case that $|V| > 2$. Assuming that u and v both are in S. If we add uv into the minimum spanning tree T, then we need connnect $\{u, v\}$ with the rest of G, which infers u or v wouldn't be a leaf. So $uv \notin T$. To guarantee v is a leaf in T, we have to keep only one of its adjacent edges in T. So we delete all adjacent edges of v except the smallest one. At last, we get a graph G' by delete those edges, and G' has the same minimum spanning tree with G.