HW4

宋子維 B03902042

Problem 1

(1) Reference: http://www.geeksforgeeks.org/full-and-complete-binary-tree-from-given-preorder-and-postorder-traversals/

Since the number of elements in left and right subtree can not be obtained from preorder and postorder list, there may exist different binary trees which have the same preorder and postorder. Hence, consider the preorder list $a, b...$, I assume that the element next to a, i.e. b, is the left child of the root a. Then the algorighm can be shown as

```
// pre: pre-order traversal list
// post: post-order traversal list
// l/r keeps track of the left/right index of post-order traversal
// preindex: current index of pre-order list
// n: size of list
preindex = 0
Node* construct(&preindex, l, r)
{
    if preindex >= n or l > r
        return NULL

    Node *root = new Node(pre[preindex])
    preindex = preindex + 1

    if l == r
        return root

    /* Take O(N) time to find the root in post */
    index = 0
    for i in range [l..r]
        if pre[preindex] == post[i]
            index = i
            break

    if index <= r
        root->left = construct(preindex, l, index)
        root->right = construct(preindex, index+1, r-1)

    return root
}
```

Then call `construct(preindex, 0, n-1)` to construct the tree. `construct()` can be called N times(from preindex = 0 to n-1), and thus the complexity is

$$N \times \mathcal{O}(N) = \mathcal{O}(N^2)$$

which is under $\mathcal{O}(N^3)$.

(2) In the preorder list, the leftmost element is the root of the tree. In the inorder list, the elements on the left side of the root are those in the left subtree, and the elements on the right side of the root are those in the right subtree. Thus the algorithm to construct the tree can be written as

```
// pre: pre-order traversal list
// in: in-order traversal list
// preindex: current index of preorder list
preindex = 0
Node* construct(start, end)
{
    if start == end
        return NULL

    Node *root = new Node(pre[preindex])
    preindex = preindex + 1

    if start == end
        return root

    /* Take O(N) time to find the root */
    index = 0
    for i in range [start..end]
        if in[i] == pre[preindex]
            index = i
            break

    /* left goes first since preorder traverse the left child first
    root->left = construct(start, index-1)
    root->right = construct(index+1, end)
}
```

Then call `construct(0, n-1)` where n is the size of in/pre-order list to construct the binary tree. The complexity can be analyzed as

$$T(N) = T(N-a) + T(a) + \mathcal{O}(N)$$

which can be solved as

$$T(N) = \mathcal{O}(N^2)$$

which is under $\mathcal{O}(N^3)$.

(3) In the postorder list, the rightmost element is the root of the tree. In the inorder list, the elements on the left side of the root are those in the left subtree, and the elements on the right side of the root are those in the right subtree. Thus the algorithm to construct the tree can be written as

```
// post: post-order traversal list
// in: in-order traversal list
// postindex: current index of postorder list
postindex = n - 1
Node* construct(start, end)
{
    if start == end
        return NULL

    Node *root = new Node(post[postindex])
    postindex = postindex - 1

    if start == end
        return root

    /* Take O(N) time to find the root */
    index = 0
    for i in range [start..end]
        if in[i] == post[postindex]
            index = i
            break

    /* right goes first since preorder traverse the right child fir
    root->right = construct(index+1, end)
    root->left = construct(start, index-1)
}
```

Then call `construct(0, n-1)` where n is the size of in/pre-order list to construct the binary tree. The complexity can be analyzed as

$$T(N) = T(N-a) + T(a) + \mathcal{O}(N)$$

which can be solved as

$$T(N) = \mathcal{O}(N^2)$$

which is under $\mathcal{O}(N^3)$.

Problem 2

(1) Discuss with B03902044 謝致有.
To determine which group A belongs to, find the number of 1 in A 's binary representation. If it is odd,

then put A into G_1 ; otherwise, put A into G_2 .

Prove: Let A and B are in G_1 . Suppose they have exactly one different bit. Then the number of 1 in A is equal the number of 1 in B minus or plus one(turn a bit off or on), which is a contraction that A and B both have the odd number of 1. Thus Alice cannot eliminate any two numbers at once in G_1 . Similarly, any two numbers in G_2 cannot be eliminated at once, either.

The complexity is

$$\mathcal{O}(NK)$$

where K is the total bit of input numbers.

(2) Reference: https://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm
https://users-cs.au.dk/bromille/dOpt/ek.pdf

Edmonds-Karp algorithm is usded to calculate maximum flow by augmenting path method. Unlike Ford-Fulkerson algorithm, Edmonds-Karp algorithm finds the shortest augmenting path each time, so it is also called “Shortest Augmenting Path algorithm”. To justify the complexity, there are two properties.

1. The distance between source and sink never decreases.
2. The number of augmentations in the Edmonds-Karp algorithm is $\mathcal{O}(VE)$.

```
while Find-shortest-augmenting-path(G, V, E, s, t, pi) /* O(VE) */
    f = f + augment(G, V, E, s, t, pi) /* O(E) */
```

Then the total complexity is

$$\mathcal{O}(VE) \times \mathcal{O}(E) = \mathcal{O}(VE^2)$$

(3) First, separate the numbers into two groups in (1). Then construct a graph G with source s and sink t . Build the edges from s to all vertices $v_1 \in G_1$ and the edges from all vertices $v_2 \in G_2$ to t . Then using brutal method to determine whether the vertex $v_1 \in G_1$ and the vertex $v_2 \in G_2$ have exactly one different bit, which takes $\mathcal{O}(N^2)$. If they are, build an edge $\langle v_1, v_2 \rangle$; otherwise, there isn’t an edge between v_1 and v_2 . Then apply Edmonds-Karp algorithm to find the maximum flow in G . The maximum flow is the number that Alice can eliminate two numbers at once(one in G_1 and another in G_2). Thus, the minimum number of magic power is

$$f + (N - 2f) = N - f$$

where f is the maximum flow.

Problem 3

(1) Since G is a con-word graph, that is, $\mu^*(G) = \min_c \mu(c)$, then for all cycles c , $\mu(c) \geq 0$. Thus, G has no negative cycles.

(2)

(a) Suppose not, there exists a shortest path from s to v with $X \geq N$ edges and total cost T . Since

one edge connects two vertices, X edges connects $X+1 \geq N+1$ vertices. Then there exists a vertex u that is traversed twice in the shortest path, which forms a cycle C

$$u \to \cdots \to u$$

By (1), $\mu(C) \geq \mu^*(G) \geq 0$. Thus by removing the cycle C , there is another path with cost $T^* = T - i\mu(C) \leq T$ where C contains i edges. Keep removing all cycles in the path until there exists no cycle in the path. Then we can get a path p with cost $T' \leq T$. If $T' < T$, it is a contradiction to the assumption. If $T' = T$, then p is a shortest path with at most $N-1$ edges since there doesn't exist any cycle in p . Both cases yield to $d(v) = \min\limits_{0 \leq k \leq N-1} d_k(v)$, that is, there exists a shortest path from s to v with at most $N-1$ edges.

(b) By the proof of (a), the path with $N > N-1$ edges will have $d_N(v) \geq d(v) = \min\limits_{0 \leq k \leq N-1} d_k(v)$. Let $d(v) = d_i(v)$ where $0 \leq i \leq N-1$. Then $d_N(v) - d_i(v) = d_N(v) - d(v) \geq 0$. And since $N - i \geq 1$, we have

$$\max_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} \geq \frac{d_N(v) - d_i(v)}{N-i} \geq 0$$

(3)

(a) Consider the path p from v back to u using edges of c . Since c is a con-weight cycle, $\sum\limits_{e_i \in p} w(e_i) + w(e) = 0$. Let h be the shortest path from s to v with cost $d(v)$. Thus there exists a path from s to u passing through p with cost $d(v) + \sum\limits_{e_i \in p} w(e_i) = d(v) - w(e) \geq d(u)$ and a path from s to v passing through e with cost $d(u) + w(e) \geq d(v)$. Then we have

$$d(v) \geq d(u) + w(e) \quad \text{and} \quad d(v) \leq d(u) + w(e)$$

Therefore, $d(v) = d(u) + w(e)$.

(b) By (2), there exists some $0 \leq k \leq N-1$ such that $d_k(u) = d(u)$ where u on c . I claim that there exists $d_{k+i}v = d(v)$ for some v on c where $i \geq 1$. I will show it by induction on i .

Prove:
When $i = 1$: Since c is a cycle, edge e_1 from u and end at v_1 exists where v_1 is on c . By (3-a), $d_{k+1}(v_1) = d(v_1) = d(u) + w(e) = d_k(u) + w(e)$.
Suppose it holds when $i = j \geq 2$, that is, $d_{k+j}(v_j) = d(v_j)$.
Since c is a $cycle$, edge e_j from v_j and end at v_{j+1} exists where v_{j+1} is on c . By (3-a), $d_{k+(j+1)}(v_{j+1}) = d(v_{j+1}) = d(v_j) + w(e_j) = d_{k+j}(v_j) + w(e_j)$.
Thus, by induction, there exists $d_{k+i}(v) = d(v)$ for some v on c where $i \geq 1$. Take $i = N-k$, there exists some v on c such that

$$d_N(v) = d_{k+(N-k)}(v) = d(v)$$

and hence

$$\max_{0\le k\le N-1}\frac{d_N(v)-d_k(v)}{N-k}=\max_{0\le k\le N-1}\frac{d(v)-d_k(v)}{N-k}=\max_{0\le k\le N-1}\frac{d_i(v)-d_k(v)}{N-k}=0$$

(By (2-a), $d(v)=d_i(v)$ for some $0\le i\le N-1$.)

(c) Since G is a con-word graph, $\mu^*(G)=\min_c \mu(c)=0$, that is, there exists a con-weight cycle c . By (3-b), there exists w on c such that

$$\max_{0\le k\le N-1}\frac{d_N(w)-d_k(w)}{N-k}=0$$

For others vertices $u\in V$, by (2-b),

$$\max_{0\le k\le N-1}\frac{d_N(u)-d_k(u)}{N-k}\ge 0$$

Thus,

$$\min_{v\in V}\max_{0\le k\le N-1}\frac{d_N(v)-d_k(v)}{N-k}=\max_{0\le k\le N-1}\frac{d_N(w)-d_k(w)}{N-k}=0$$

(4)

(a) Since one edge connects two vertices, p has N edges which implies that p contains $N+1$ vertices. However, there are only N vertices. Thus, there exists at least one vertex u which is traversed twice(or more) in p . Then at least one cycle c

$$u\to\cdots\to u$$

forms.
Let path q be path p after removing cycle c . Suppose the cost of q is T_q . Since the cost of c is W , we have

$$T_q=d_N(v)-W$$

And since c has length l , q has length $N-l$. Then

$$T_q\ge d_{N-l}(v)$$

Thus

$$T_q=d_N(v)-W\ge d_{N-l}(v)\Rightarrow d_N(v)-d_{N-l}(v)\ge W$$

(b) By (4-a), let c be arbitrary cycle such that $c\in p$ with length l and cost of c is $W\ge 0$ (G has no negative cycle) with

$$d_N(v)-d_{N-l}(v)\ge W$$

Since

$$\max_{0\le k\le N-1}\frac{d_N(v)-d_k(v)}{N-k}=0$$

and

$$0\le N-l\le N-1$$

we have

$$0=\max_{0\le k\le N-1}\frac{d_N(v)-d_k(v)}{N-k}\ge\frac{d_N(v)-d_{N-l}(v)}{N-(N-l)}\ge\frac{W}{l}\ge 0$$

Hence,

$$W=0$$

Since p contains at least one cycle and c is arbritary cycle in p , then p contains at least one con-weight cycle, but does't contain any positive-weight cycle.

(5)

(a) Discusee with 林義聖 B03902048

Reference: http://webcourse.cs.technion.ac.il/236359/Spring2013/ho/WCFiles/MMC.ps

If G is a con-word graph, $\mu^*(G)=\min_c \mu(c)=0$, that is, G has at least one con-weight cycle. By (3-c),

$$\min_{v\in V}\max_{0\le k\le N-1}\frac{d_N(v)-d_k(v)}{N-k}=0=\mu^*(G)$$

If G is not a con-word graph, subtract the weight of every edge by $\mu^*(G)$. Let $\mu'(G)$ be the weight of the minimum mean cycle after subtracting. We have

$$\mu'(G)=\frac{i(\mu^*(G))-i(\mu^*(G))}{i}=0$$

where i is the length of origin minimum mean cycle.

Hence G is a con-word graph after subtracting every edges by $\mu^*(G)$, By the property above, we have

$$\mu'(G)=\min_{v\in V}\max_{0\le k\le N-1}\frac{d'_N(v)-d'_k(v)}{N-k}=0$$

where $d'_i(v)$ is the shortest path passing through exactly i edges after substracting.

Since $d'_i(v)$ goes through exactly i edges, we have

$$d'_i(v) = d_i(v) - i(\mu^*(G))$$

Hence

$$\begin{aligned}
0 &= \mu'(G) \\
&= \min_{v \in V} \max_{0 \le k \le N-1} \frac{d'_N(v) - d'_k(v)}{N-k} \\
&= \min_{v \in V} \max_{0 \le k \le N-1} \frac{(d_N(v) - N(\mu^*(G))) - (d_k(v) - k(\mu^*(G)))}{N-k} \\
&= \min_{v \in V} \max_{0 \le k \le N-1} \frac{(d_N(v) - d_k(v)) - (N-k)(\mu^*(G))}{N-k} \\
&= \min_{v \in V} \max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k} - \mu^*(G) \\
\Rightarrow \mu^*(G) &= \min_{v \in V} \max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k}
\end{aligned}$$

(b) Reference: http://webcourse.cs.technion.ac.il/236359/Spring2013/ho/WCFiles/MMC.ps
By the definition, we know that

$$d_i(v) = \min_{u:(u,v) \in E} d_{i-1}(u) + w((u,v))$$

Then it takes $\mathcal{O}(NM)$ to compute $d_k(v)$ for each vertices $v \in V$ and $k \le N$.

```
// source: s
d[0][s] = 0
for v != s in V
    d[0][v] = INFTY

// calculate d[i][v]
for i in range [1..N]
    for v in V
        d[i][v] = min {d[i-1][v] + w(u, v)}, u: (u, v) in E
        f[i][v] = arg min {d[i-1][v] + w(u, v)}, u: (u, v) in E
```

where $f[i][v]$ is used to record the shortest path from s to v with exactly i edges, which is needed to backtrack.
By (5-a), calculate

$$\mu^*(G) = \min_{v \in V} \max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k}$$

which takes $\mathcal{O}(N^2)$.
For v such that,

$$\mu^*(G) = \max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k}$$

we can backtrack $f_N(v)$ to find the minimum mean cycle in $\mathcal{O}(N)$.
Hence, the total complexity is

$$\mathcal{O}(NM) + \mathcal{O}(N^2) + \mathcal{O}(N) = \mathcal{O}(N^2 + NM)$$

Prove the correctness:
If G is a con-word graph, by (3) and (4), there exist at least one con-weight cycle c in $f_N(v)$. Thus,

$$\mu(c) = \mu^*(G) = 0$$

implies c is a minimum mean cycle.
If G is not a con-word graph, subtract all edges by $\mu^*(G)$, and let G' be the graph after subtracting. By the prove of (5-a),

$$\mu^*(G') = 0$$

Thus, G' is a con-word graph, and

$$0 = \mu^*(G') = \max_{0 \le k \le N-1} \frac{d'_N(v) - d'_k(v)}{N-k}$$

Since G' is a con-word graph, by (3) and (4), there exist at least one con-weight cycle c in $f_N(v)$ (subtracting weights doesn't influence vertices in the path). Thus, let l be the length of cycle c , and we have

$$\mu(c) = \frac{l(\mu'(c) + \mu^*(G))}{l} = \mu^*(G)$$

implies c is a minimum mean cycle.