Algorithm Design & Analysis: Homework #4

Due on 2015/12/21

王冠鈞 (b03902027)

I applied and slightly modified the LaTeX template from 劉彥廷(b03902028).

Problem 1

(0) References: None

(1) When both lists cannot identify left or right nodes, the following algorithm will make them be at the left.

```
Data: two arrays: pre[], post[]; start, end: for post[]; index initialized to start in global
    (for pre[])
Result: a binary tree
Make-Tree-Init(pre[], post[], start, end)
Make node of pre[index], let N be the node
if start ≠ end then
    index++
    Make-Tree(pre[], post[], start, end - 1)
end
if index ≠ end then
    Make-Tree(pre[], post[], index - 1, end - 1)
end
return N

Make-Tree(pre[], post[], start, end)
Make node of pre[index], let N be the node
if start ≥ end then
    return node N
end
Search pre[index] at post[], let post[i] be the result (taking $O(n)$ time)
index++
if i ≠ start then
    N.left = Make-Tree(start, i - 1)
else
    N.left = NIL
end
if i ≠ end then
    N.right = Make-Tree(pre, in, i + 1, end)
else
    N.right = NIL
end
return N
```

The time complexity is at most $O(n^2)$.

(2) Construct the tree recursively.

```
Data: two arrays: pre[], in[]; index initialized to start in global (for pre[])
Result: a binary tree
MAKE-TREE(pre[], in[], start, end)
Make node of pre[index], let N be the node
if $start \geq end$ then
    return node N
end
Search pre[index] at in[], let in[i] be the result (taking $O(n)$ time)
index++
if $i \neq start$ then
    N.left = MAKE-TREE(pre, in, start, i - 1)
else
    N.left = NIL
end
if $i \neq end$ then
    N.right = MAKE-TREE(pre, in, i + 1, end)
else
    N.right = NIL
end
return N
```

The time complexity is at most $O(n^2)$.

(3) Construct the tree recursively; basically same as (2).

```
Data: two arrays: post[], in[]; index initialized to end in global (for post[])
Result: a binary tree
MAKE-TREE(post[], in[], start, end)
Make node of post[index], let N be the node
if $start \geq end$ then
    return node N
end
Search post[index] at in[], let in[i] be the result (taking $O(n)$ time)
index–
if $i \neq end$ then
    N.right = MAKE-TREE(pre, in, i + 1, end)
else
    N.right = NIL
end
if $i \neq start$ then
    N.left = MAKE-TREE(pre, in, start, i - 1)
else
    N.left = NIL
end
return N
```

The time complexity is at most $O(n^2)$.

Problem 2

(0) References:

[1] CLRS 3rd ed., Chapter 26

[2] `http://www.csie.ntnu.edu.tw/~u91029/Flow.html`

[3] Discussed with 劉彥廷(b03902028)

(1) For all the numbers, change them into binary representation. If the number of 1-s is odd, put them into one group; if it's even, put them into the other.

(2) The *Edmonds-Karp Algorithm* is an enhanced version of the *Ford-Fulkerson Algorithm.* While finding an augmented path, The Edmonds-Karp algorithm uses BFS to find augmented paths of shortest length, which will guarantee that some of the flow in the edges in the residue network will increase without any of them decreasing. As for the time complexity, the algorithm uses $O(VE)$ time to perform flow augmentations, and there are E iterations in the original Ford-Fulkerson algorithm. Thus, in total it takes $O(VE^2)$ time.

(3) Create a bipartite-like graph. Add a source (s) and a sink (t) and connect all the nodes in one group with s with edges of maximum flow = 1, and connect the other group with t with edges of the same maximum flows. Since for any two numbers in the same group, they

can never be eliminated once. Thus, we connect all the pairs from left to right with flow $= 1$ that can be eliminated together, after which process it will form a bipartite-like graph. Then, perform Edmonds-Karp algorithm to find the maximum flow f_{max}, which represents the maximum number of pairs that can be eliminated together. Then the answer will be $f_{max} + (N - 2f_{max}) = N - f_{max}$.

Problem 3

(0) References:

[1] `http://www.csie.ntnu.edu.tw/~u91029/Cycle.html`

[2] CLRS 3rd ed., Chapter 24, Problem 24-5, p680-681

[3] `http://courses.csail.mit.edu/6.046/fall01/handouts/ps9sol.pdf`

(1) If G has negative cycles, then $\mu^*(G)$ should be negative, but G is said to be a *con-word graph*, where $\mu^*(G) = 0$. Thus G has no negative cycles.

(2) (a) From (0), we know that there isn't any negative cycles, and thus the minimum won't form a cycle, then the minimum must be in a path of length less than $|V| = N$, so it will be at most $N - 1$ long.

(b) First, since $0 \le k \le N - 1$, then $N - k > 0$. Second, since $d_N(v)$ forms a cycle, and since there's no negative cycles, then $d_N(v) \ge d(v)$, which means that

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

Thus, the inequality is proven.

(3) (a) First, to reach vertex v, one of the paths might pass through u via edge e, thus we can get

$$d(u) + w(e) \ge d(v)$$

, because all the other possible paths must be no greater than the path described above. Second, to reach vertex u via v, we can get

$$d(u) \le d(v) - w(e)$$

where $-x$ is the weight along the cycle other than e. Since $w(c) = 0$, then $w(c \backslash \{e\}) = 0 - w(e) = -w(e)$. Then we get the inequality.
From the two inequalities above, we can get $d(v) = d(u) + w(e)$.

(b) According to (2)(a), we can definitely find a vertex v on c with $d_k(v) = d(v), 0 \le k \le N - 1$. Let the next vertex of v along c is u, then according to the result of (3)(a), $d(u) = d(v) + w(e), e$ is the edge on cycle c from v to u, where $d(u) = d_{k+1}(u)$. Thus, by doing such process (add the weight of next edge of c on the minimum weight of the former vertex will become the minimum weight of the latter vertex) inductively, we can eventually find a vertex v', such that $d(v') = d_N(v')$, which is what we want.

(c) According to (2)(a) and (3)(b), there must exist a vertex v such that $d_N(v) = d(v) = d_k(v)$, in some $0 \le k \le N-1$. Thus

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

must be valid since $d_N(v) - d_k(v) = 0$, then the minimum of the total equation will be 0.

(4) (a) First, since there are only N vertices, we can at most take $N-1$ edges in order not to form a cycle. Thus, when a path contains N edges, there must contain a cycle.
If a path p contain a cycle of length l, then we can delete the cycle and make the remaining weight be $d_N(v) - W$. Furthermore, we know that such modified path may or may not be the minimum path of length $N-l$, so the minimum weight with path length $N-l$ must satisfy $d_{N-l}(v) \le d_N(v) - W$. Thus we get $d_N(v) - d_{N-l}(v) \ge W$.

(b) When a vertex v satisfies such equation, according to (3)(c), it implies that the vertex v satisfies $d_N(v) = d(v) = d_k(v)$, in some $0 \le k \le N-1$, otherwise $d_N(v) - d_k(v) \ne 0$, which will contradict with the equation of the current sub-problem. We have known from (4)(a) that there must exist a cycle, and because $d_N(v) = d_k(v)$, it implies that there must contain con-weight cycles with a total of $N-k$ edges, such that when we delete these cycles, the remaining edges of the path will be k and will not affect the weight. If there exists any positive-weight cycles, $d_k(v)$ must be able to reach a smaller value, which contradicts with the fact that $d_N(v) = d_k(v)$.

(5) (a) Consider the equation found in the previous part:

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

Then we consider a constant x. When we add x on every edges of G, the minimum mean cycle will become $\mu'^*(G) = x$. In the meantime, $d_N(v)$ will increase by Nx and, $d_k(v)$ will increase by kx, which will still keep the equation balanced(since $d'_N(v) - d'_k(v) = d_N(v) + Nx - (d_k(v) + kx) = (N-k)x$). Thus the equation is verified correct.

(b) Compute $d_k(v)$ for $k = 0, 1, \ldots, N$ and evaluate the equation from (3)(a) every time. This will take (scan M edges in every turn) $\times$ (N turns) $= NM = O(NM)$.
Then compute the fraction from (5)(a) and compare the maximum and minimum, which takes (compute fraction and decide maximum from $k = 0$ to $N-1$) $\times$
(decide minimum of N vertices) $= O(N^2)$ time. If $\mu^*(G) < \infty$, verify the vertex, subtract all the edges with $\mu^*(G)$ and go back to the 1st step while this time store the edges passed. When a cycle is detected, then it's done. The total running time is $O(NM) + O(N^2) = o(N^2 + NM)$.