B03902062 資工二 董文捷

Problem 1:

Given two kind of traversal lists, we can use a function to reconstruct the tree and return the root. First find out the root of the tree and two traversal lists of left sub-tree and right sub-tree, recursively call the function with two lists to reconstruct left sub tree and right sub-tree, connect their root as root's left child and right child, then return root.

(1) The first member in pre-order list is the root, the second member in pre-order list is the left child, the second-last member in post-order list is the right child. In pre-order list, list from left-child before right-child is the pre-order list of left sub-tree, list from right-child to the end is the pre-order list of right sub-tree. In post-order list, list from begin to left-child is the post-order list of left sub-tree, list after left-child to right-child is the post-order list of right sub-tree.
Time complexity :
Find root : O(1)
Find left-child and right-child : O(1)
Find left and right lists : O(N)
Reconstruct left sub-tree and right sub-tree : T(#left) + T(#right)
Worst case : T(N) = T(N – 1) + O(N)
$O(N^2)$

(2) The first member in pre-order list is the root. In in-order list, list from begin before root is the in-order list of left sub-tree, list after root to the end is the in-order list of right sub-tree, record size of left list and right list. In pre-order list, those #left members after root form the pre-order list of left sub-tree, and those #right members after left sub-tree form the pre-order list of right sub-tree.
Time complexity :
Find root : O(1)
Find left and right lists : O(N)
Reconstruct left sub-tree and right sub-tree : T(#left) + T(#right)
Worst case : T(N) = T(N – 1) + O(N)
$O(N^2)$

(3) The last member in post-order list is the root. In in-order list, list from

begin before root is the in-order list of left sub-tree, list after root to the end is the in-order list of right sub-tree, record size of left list and right list. In post-order list, those #left members from begin form the post-order list of left sub-tree, and those #right members after left sub-tree and before root form the post-order list of right sub-tree.
Time complexity :
Find root : O(1)
Find left and right lists : O(N)
Reconstruct left sub-tree and right sub-tree : T(#left) + T(#right)
Worst case : T(N) = T(N – 1) + O(N)
$O(N^2)$

Problem 2:
(1) Consider each integer as a vertex in an undirected graph, and two vertices have an edge if they differ by exactly one bit. After connecting all edges, use a DFS to traverse vertices. An extra parameter group_num should be passed in DFS function to indicate parent's group_num, and a child should be placed in different group with parent. Therefore, change group_num from 0 to 1 or 1 to 0, push child (now node) into that group, continue DFS. Two groups are separated after each vertex is visited.
Time complexity : $O(N^2)$
(2)
Reference : Chapter 26 of Introduction to Algorithm
Edmonds-Karp algorithm is one specific implementation of the Ford-Fulkerson method. Ford-Fulkerson method is a method for solving maximum-flow problem by increasing the flow when finding an augmenting path in the residual network G_f. However, naïve implementation may takes $\Theta(E|f^*|)$ ($|f^*|$ is the optimal flow value) under some circumstances like p728 Figure26.7 if we find an augmenting path arbitrarily, when $|f^*|$ is large, this complexity is poor. Therefore, Edmonds-Karp algorithm is used to improve the bound on Ford-Fulkerson. Edmonds-Karp algorithm choose the augmenting path as a shortest path from s to t (each edge has unit distance). We say an edge (u, v) is critical on an augmenting path if $c_f(p) = c_f(u, v)$. As p729 Theorem 26.8 proved, each augmenting path has at least one critical edge, and the total number of critical edges during entire execution is O(VE), so the bound of total time complexity is improved to O(E) (find the augmenting path with BFS) * O(VE) (at most O(VE) times) = $O(VE^2)$.

(3)

Reference : http://www.csie.ntnu.edu.tw/~u91029/Matching.html and discussion with b03902061

If we eliminate number one by one, we need to use N magic power. However, if we find any pair with two numbers differ by exactly one bit, the magic power used will decrease by one. Since two numbers in the same group can't be eliminated at once, the pair must contain two numbers in different groups. Our target is to find as much pair as possible, which is a maximum bipartite matching problem. Connect sorce s to every vertex in group 0 with capacity ∞, and connect every vertex in group 1 to sink t with capacity ∞. Check every combination of two numbers in different groups, if they differ by exactly one bit, connect number in group 0 to number in group 1. The problem becomes a maximum-flow problem, each pair will increase flow by 1. Applying Edmonds-Karp algorithm to find out max flow $|f^*|$, the minimum magic power used to eliminate all the N integers is N - $|f^*|$.

Problem 3:

(1) Since $\mu^*(G) = 0$, for any cycle c in G, $\mu(c) \geqq 0$, which means total weight = k $*\mu(c) \geqq 0$, there is no negative cycle.

(2)

(a) A shortest path with $k \geqq N$ must contains at least one cycle because the path will pass vertices total k + 1 times, but there are only N vertices, k + 1 > N, which means some vertices is passed more than one time, there must be a cycle. Since all cycle weight $\geqq 0$ in G, if we delete the cycle, we can get a path $\leqq$ shortest path, the equal must hold, so new path is also a shortest path. By repeating this process, we can finally get a shortest path with $0 \leqq k \leqq N - 1$

(b) As proved in (2)(a), we can find a $0 \leqq k' \leqq N - 1$ that $d_{k'}(v)$ is a shortest path. Since $d_N(v) \geqq$ shortest path = $d_{k'}(v)$, if we choose k = k',

$d_N(v) - d_{k'}(v) \geq 0$ $N - k' > 0$ $\frac{d_N(v) - d_{k'}(v)}{N-k'} \geq 0$, so

$\max\limits_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} \geq 0$

(3)

(a) We can find a path from s to v by go through d(u) and e, so d(v) ≦ d(u) + w(e). Also, we can find a path from s to u by go through d(v) and other edges in c except for e, so d(u) ≦ d(v) + (0 – w(e)). By d(v) ≦ d(u) + w(e) and d(v) ≧ d(u) + w(e), we can prove that equality must hold, so d(v) = d(u) + w(e)

(b) By (2)(a), choose a vertex u on c, we can find a 0 ≦ k ≦ N – 1 that $d_k(u)$ is a shortest path = d(u). By (3)(a), if there is an edge u→v in c, d(v) = d(u) + w(e), since the path to v with k + 1 edges by going through k edges of $d_k(u)$ and e has a weight of $d_k(u)$ + w(e) = d(u) + w(e) = d(v), $d_{k+1}(v)$ ≦ d(v), the equality must hold, so $d_{k+1}(v)$ = d(v). By repeating the process (N – K) times, we can find a vertex v' on c such that $d_N(v')$ = d(v').

For any 0 ≦ k ≦ N – 1, $d_k(v')$ ≧ $d_N(v')$ = d(v'), $d_N(v') - d_k(v') \leqq 0$ $N - k > 0$ $\frac{d_N(v') - d_k(v')}{N-k} \leqq 0$, the equality holds when $d_k(v')$ = d(v'), we can always find a k satisfying it by (2)(a), so $\max\limits_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} = 0$

(c) By (2)(b), for all v in V, $\max\limits_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} \geq 0$. By (3)(b), we can find a v make the equality holds. So $\min\limits_{v \in V} \max\limits_{0 \leq k \leq N-1} \frac{d_N(v) - d_k(v)}{N-k} = 0$

(4)

(a) A path of N edges will pass vertices total N + 1 times, but there are only N vertices, N + 1 > N, which means some vertices is passed more than one time, there must be a cycle. If we delete a cycle of length l and total weight W from $d_N(v)$, we can find another path to v with N – l edges and weight $d_N(v)$ – W, so $d_{N-l}(v)$ ≦ $d_N(v)$ – W, $d_N(v)$ - $d_{N-l}(v)$ ≧ W

(b) By (4)(a), every shortest path p contains at least one cycle. By (1), there is no negative-weight cycle in G, so the cycle is either positive-weight cycle or con-weight cycle. However, if p contains a positive cycle, by (4)(a), $\mathrm{d}_N(\mathrm{v}) - \mathrm{d}_{\mathrm{N-l}}(\mathrm{v}) \geqq \mathrm{W} > 0$, $\frac{d_N(v) - d_{N-l}(v)}{N-(N-l\,)} > 0$, which is a contradiction to $\max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k} = 0$. Therefore, every p contains at least one con-weight cycle, but doesn't contain any positive-weight cycle.

(5)

(a)

Case 1 :

$\mu^*(G) = \infty$

We can't find any path from s to v with exactly N edges because G doesn't contain any cycle, each vertex can only be passed one time, a path is at most N – 1 edges. So for all $v \in V$ $\max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k} = \infty$,

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

Case 2 :

$\mu^*(G)$ is finite

If we subtract the weight of every edge by $\mu^*(\mathrm{G})$ to get a new graph G', every cycle's mean weight will be subtracted by $\mu^*(\mathrm{G})$, so $\mu^*(\mathrm{G}')$ = 0, G' is a con-word graph. Also, $\frac{d_N(v) - d_k(v)}{N-k}\ in\ G' =$

$\frac{(d_N(v) - \mu^*(\mathrm{G}) * N) - (d_k(v) - \mu^*(\mathrm{G}) * k)}{N-k} = \frac{d_N(v) - d_k(v)}{N-k}\ in\ G$ - $\mu^*(\mathrm{G})$, each of $\frac{d_N(v) - d_k(v)}{N-k}$ in G' is subtracted by the same value $\mu^*(\mathrm{G})$, so v and k having extreme value are the same. As proved in (3)(c), in a con-word graph G', $\min_{v \in V'} \max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k} = 0$, taking same v and k in G ensures an extreme value too. $\frac{d_N(v) - d_k(v)}{N-k}\ in\ G = \frac{d_N(v) - d_k(v)}{N-k}\ in\ G' + \mu^*(\mathrm{G}) =$ $0 + \mu^*(\mathrm{G})$ = $\mu^*(\mathrm{G})$,

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

(b) As proved in (4)(b), for all v with $\max_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k} = 0$, every p contains at least one con-weight cycle, but doesn't contain any positive-

weight cycle. A con-weight cycle in G' is a minimum mean cycle in G, so if we can find out a v in G having minimum value of $\max\limits_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k}$, then find a shortest path p with exactly N edges, cycle in p is a minimum mean cycle we want.
First, use Bellman-ford algorithm to find all $d_k(v)$ ($0 \le k \le N$ *and* $v \in V$), here we adjust the definition of $d_k(v)$ to use exactly k edges, so the recurrence should be $d_k(v) = \min\{ d_{k-1}(u) + w(u, v)\}$, this step takes O(NM)
Then we can calculate $\max\limits_{0 \le k \le N-1} \frac{d_N(v) - d_k(v)}{N-k}$ of all vertices, whenever the minimum value updates, record the v, this step takes O(N^2)
Finally, find out the path of $d_N(v)$ by the table of Bellman-ford algorithm, for example, if $d_k(v) = d_{k-1}(u) + w(u, v)$, then we can know one possible choice of k-th edge is (u, v). With an array recording whether a vertex is visited or not, we can find a cycle in p, which is a minimum mean cycle in G. This step takes (find path) O(N^2) + (find cycle) O(N)
Total time complexity :
O(N^2 + NM)