Homework 2

Zhehao Wang 404380075 (Dis 1D)

Apr 10, 2016

1 Number of inversions remains unchanged for any permutation

Proof by induction on the number of position swaps in the permutation: we denote the original lists as $A = a_1...a_n$ and $B = b_1...b_n$. Each permutation consists of a number of position swaps for songs in both list A and B. We call a pair (a_i, a_j) *flipped* if it used to be (a_i, a_j) before the permutation, and becomes (a_j, a_i) after the permutation, and the songs a_i, a_j to be *involved* in the flip.

Base case: consider the case where only one pair of songs in both A and B have their positions swapped. Denote the pair as (a_i, a_j) in the original list A, and (b_m, b_n) in the original list B, we have $i < j$, $m < n$, $a_i = b_m$ and $a_j = b_n$.

The *flipped* pairs in A caused by this permutation include: (a_i, a_j), (a_p, a_j) and (a_i, a_p), where $i < p < j$. Similarly, *flipped* pair in B include: (b_m, b_n), (b_q, b_n) and (b_m, b_q), where $m < q < n$. Since $a_i = b_m$ and $a_j = b_n$, number of inversions is not changed by A and B both having the (a_i, a_j), (b_m, b_n) flips. Thus we consider each a_p and b_q *involved* in the flip, and the total number of inversions does not change if each *involved* a_p and b_q do not cause changes in the number of inversions. Case analysis on the position of each b_l in B where each $b_l = a_p$.

- If $l < m$, then list B used to have (b_l, b_m) and (b_l, b_n), list A used to have (a_i, a_p) and (a_p, a_j), number of inversions used to be 1. After the permutation, B's pairs involving b_l, b_m, b_n are not flipped, and A has (a_p, a_i), (a_j, a_p). Number of inversions is still 1.

- If $l > n$, the case is similar with above. The number of inversions before and after the permutation are both 1.

- If $m < l < n$, then B used to have (b_m, b_l) and (b_l, b_n), A used to have (a_i, a_p) and (a_p, a_j), and number of inversions used to be 0. After the permutation, B has (b_l, b_m) and (b_n, b_l), A has (a_p, a_i) and (a_j, a_p). The number of inversions is still 0.

Similar case analysis can be done for each a_k in list A where each $a_k = b_q$. Thus we have the number of inversions does not change when only one pair is swapped in the permutation.

Induction case: assume that the conclusion holds for any permutation involving n position swaps. For any permutation involving $n + 1$ position swaps, by the induction hypothesis, we know that the conclusion holds for its sub-permutation with one pair excluded. By applying the analysis of the base case on the results of the sub-permutation, we know that the conclusion holds for any permutations involving $n + 1$ position swaps as well.

2 Number of intersection and inversions

3 Celebrity iterative

4 Diameter of tree

(a)

Define the **height** of a rooted directed tree as the number of edges on the longest path from the root to a leaf. Algorithm is given in Alg 1.

Algorithm 1 Diameter of a rooted directed tree's underlying undirected tree, recursive

---

1: **function** FINDHEIGHTORDIAMETER(root, findHeight, prevRoot)
2:    **if** $degree(root) = 1$ **then**
3:        **return** 0
4:    $heights \leftarrow []$
5:    **for each** $\{n | n \in V, (n, root) \in E, n \neq prevRoot\}$ **do**
6:        $heights.push(1 + findHeightOrDiameter(n, true, root))$
7:    **if** $findHeight$ **then**
8:        **return** $max(heights)$
9:    **else**
10:        **return** $max(heights) + 2^{nd}highest(heights)$

This recursive algorithm takes in the root of a tree and produces the height of the tree, by each time removing the root and finding the maximum height among all resulting sub trees. The diameter of the tree would be the sum of the heights of two highest subtrees. Initial call to the algorithm should look like $findHeightOrDiameter(root, false, nil)$. This algorithm is $O(n)$, where n is the number of nodes in the tree, because each node in the tree will be visited exactly once.

(b)

The iterative version of the algorithm is given in Alg 2.

This algorithm is $O(n)$, where n is the number of nodes in the tree, because each node in the tree will be visited exactly once.

Algorithm 2 Diameter of a rooted directed tree's underlying undirected tree, iterative

---

1: **function** FINDHEIGHTORDIAMETER(root)
2:     $queue \leftarrow [root]$
3:     $height0 \leftarrow 0$
4:     $height1 \leftarrow 0$
5:     **while** True **do**
6:         $nodeCount \leftarrow queue.size()$
7:         **if** $nodeCount = 0$ **then**
8:             **return** $height0 + height1$
9:         $height \leftarrow height + 1$
10:        **while** $nodeCount > 0$ **do**
11:            $r \leftarrow queue.dequeue()$
12:            $r.visited \leftarrow true$
13:            **if** $degree(r) = 1$ **then**
14:                **if** $height > height0$ **then**
15:                    $height1 \leftarrow height0$
16:                    $height0 \leftarrow height$
17:                **else if** $height > height1$ **then**
18:                    $height1 \leftarrow height$
19:            **else**
20:                **for each** $\{n|n \in V, (n,r) \in E, n.visited = false\}$ **do**
21:                    $queue.enqueue(n)$
22:            $nodeCount \leftarrow nodeCount - 1$