Assignment I

Aman Tiwari 160094

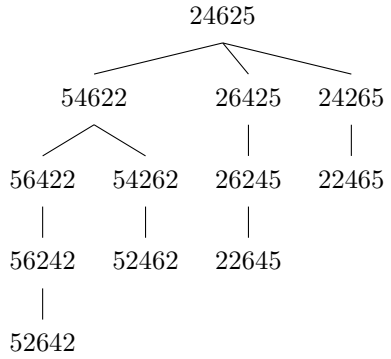
August 2018

1 Problem

Let N be a sequence of n digits . Let E be a set of some pairs of integers in the set 0,1,2,....,n-1.One *exchange operation* corresponds the swapping of i-th and j-th digit in N where $\{i, j\} \in E$. Design an algorithm to compute the shortest sequence of exchange operations which transforms N into the largest number (as an n-digit number) achievable this way.

2 Solution

We can model the problem as a graph problem . We can define the graph as $G = (V, E')$ where V represents the set of all the numbers which are possible after applying some valid *exchange operations* on the initial number and the edge $\{v_1, v_2\} \in E'$ iff it is possible to transform the number v_1 to v_2 by a single valid *exchange operation*.
Now in order to find the largest number achievable in shortest sequence of exchange operations, we can start from the initial number and perform a breadth first traversal of the graph. While doing the breadth first search we can store the largest number that we have seen so far and also the number of exchange operations that it took to reach that number. So after traversing the entire graph we will have the largest number possible and also the minimum number of steps to reach it.
Here is an example that describes working of the algorithm. Let the initial number be $N = 24625$ and $E = \{(0, 4), (1, 2), (2, 3)\}$.The breadth first tree of the graph looks like this :



Here the largest number achievable is 56422 and it is possible to achieve this number in a minimum of two *exchange operations*.

We can consider each vertex as a structure consisting of two members, a string which represents the number and an integer which represents the distance of the vertex from the initial vertex in the breadth first tree. Let the initial vertex be s , then $s.first = N$ and $s.second = 0$. As we traverse the graph we will create new vertices and initialize the member elements of the vertices to their default values. When the algorithm terminates the maximum number is stored in the variable *max* and the minimum possible *exchange operations* is stored in variable *minswaps*.

The algorithm is a simple breadth first search , the only difference is that we use an additional *convert* function to convert string into integer. The time complexity of the *convert* function is $\mathcal{O}(n)$. Hence time complexity of the algorithm will be $\mathcal{O}((|V| + |E'|) * n)$. Here the number of vertices in the graph is bounded by 10^n and the number of edges is bounded by $10^n * |E|$. Hence the time complexity is $\mathcal{O}(10^n * (1 + |E|) * n) = \mathcal{O}(10^n * |E| * n)$. We use an array of size 10^n to store the status of each vertex and the structure for storing vertices of the graph takes space proportional to the number of vertices which is bounded by 10^n so the space complexity is $\mathcal{O}(10^n)$.

Algorithm 1 Breadth First Search

```
1: function BFS(s,status,n)
2:     for i = 1 to 10^n do
3:         status[i] = "unvisited";
4:     max = NULL
5:     minswaps = ∞;
6:     s1 = s.first;
7:     num = convert(s1);
8:     status[num] = "currently visiting";
9:     initialize an empty queue Q;
10:    enqueue(s, Q);
11:    while Q ≠ φ do
12:        u = dequeue(Q);
13:        u1 = u.first;
14:        num1 = convert(u1);
15:        for (i, j) ∈ E do
16:            v1 = swap(u1, i, j);          ▷ Swaps i − th and j − th character
17:            num2 = convert(v1);
18:            if status[num2] = "unvisited" then
19:                status[num2] = "currently visiting";
20:                if num2 > max then
21:                    max = num2;
22:                    minswaps = u.second + 1;
23:                create and initialize a new vertex v;
24:                v.first = v1;
25:                v.second = u.second + 1;
26:                enqueue(v, Q);
27:        status[num1] = "visit complete";
```

Algorithm 2 Convert string to integer

```
1: function CONVERT(s)
2:     N = s.length;
3:     number = 0;
4:     c = 0;
5:     for i = N − 1 down to 0 do
6:         number = number + power(10, c) ∗ (s[i] −′ 0′);
7:         c = c + 1;
8:     return number;
```