CS345: Algorithms II
Assignment 1 Solution

Ayush Bansal
Roll No. 160177

August 22, 2018

I Problem 7(b)

1.1 Problem Statement

We are given a sequence N made up of n digits. We can perform exchange operations on the digits of this sequence and the list of allowed exchange operations (by position in the sequence) is given to us in the form of a set E, E is a set of some pair of integers from the set $0, 1, \ldots, n-1$. One exchange operation corresponds to swapping of i^{th} and j^{th} digit in N where $\{i, j\} \in E$.
I have to design an algorithm to compute the shortest sequence of exchange operations which transforms N into the largest achievable number (as an n-digit number).

1.2 Brief Outline of Solution

So, think of this as a graph problem whose vertex set will be a set of n-digit numbers which will the possible permutations of the original sequence N we can get by applying operations from exchange set E. The edge-set of the above graph will be the exchange operations which leads from one permutation to another.
Finally, I will use the **BFS** algorithm on this graph to find the vertex with max value and the shortest path leading up to it.

1.3 Data Structures

Firstly, I will define a structure called exchange which will contain 2 integers from the set $0, 1, \ldots, n-1$ denoting the position corresponding to the exchange operation to be done.

```
typedef struct Exchange {
  tuple e;
} exchange;
```

Secondly, lets look at the node structure, it will consist of the value of the node, it will a string (since sequence of digits can be large and thus, integer won't be able to hold it). It will also contain a tuple value which basically is the representation of an exchange operation, this exchange operation will give us the original number that this number was generated from.

```
typedef struct Node {
  string value;
} node;
```

Thirdly, there will be a hash table named **swaps** whose keys will be nodes and the values will be the exchange tuple that led to the creation of that node.
Next, since I am going to perform a BFS operation, I will be creating a hash table named **status** which will have nodes as keys and the values of the table will be one of the three which are **visited**, **currently-visiting**.
Finally, there will be a stack data structure used for reverse-traversal from destination vertex to initial vertex to print the exchange operations in ascending order (from initial vertex to final).

1.4 Algorithm

I will do a **BFS** on the graph, but wait where is the graph, the graph will be created as the BFS algorithm is executed. I will take the initial sequence as the initial vertex and then apply each possible exchange operation from the set E to get a new vertex from this one.
Firstly creating the function for swapping digits and giving a new node.

Algorithm 1 Swap digits to generate new vertex

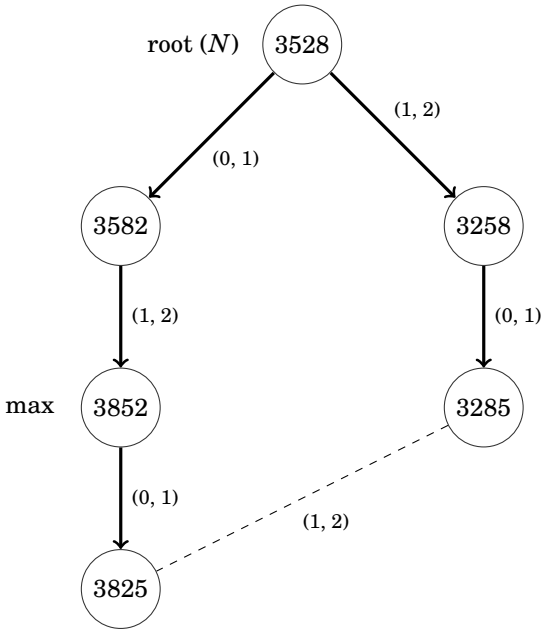
```
1: procedure SWAPDIGITS(v, e)                    ▷ Takes initial vertex and exchange tuple
2:     v_new ←new-node()
3:     v_new.value ←Swap(v.value,e)              ▷ this function will give final string after swapping
4:     return v_new
5: end procedure
```

Algorithm 2 Generating largest value sequence in minimum number of swaps

```
1: procedure PRINTPATH(N, E)                     ▷ Takes original sequence and exchange operations set
2:     v_0 ←New-Node()                           ▷ by default value and e in new-node are both NULL
3:     v_0.value ← N
4:     Max ← v_0                                 ▷ Keeping track of max number
5:     status[v_0]← currently-visiting
6:     Queue← New-Queue()
7:     Enqueue(v_0,Queue)
8:     while Queue is not empty do
9:         u ← Dequeue(Queue)
10:        for all ex ∈ E do                     ▷ Get new vertex by exchange operations
11:            v ←swapDigits(u,e)                ▷ Generate a new vertex
12:            if status[v] is NULL then         ▷ Key not found in status table
13:                status[v]← currently-visiting
14:                swaps[v]← ex                  ▷ Exchange used to get this vertex
15:                if v.value > Max.value then   ▷ Comparison is created especially for strings
16:                    Max ← v
17:                end if
18:                Enqueue(v,Queue)
19:            end if
20:        end for
21:        status[u]← visited
22:    end while
23:    stack←New-Stack()                         ▷ Building stack to contain exchanges
24:    cur ← Max
25:    while cur.value ≠ v_0.value do
26:        Push(stack, swaps[cur])
27:        cur ←swapDigits(cur, swaps[cur])      ▷ Getting the previous Vertex
28:    end while
29:    while stack is not empty do               ▷ Printing the final sequence of exchanges
30:        e ←Pop(stack)
31:        Print(e)
32:    end while
33: end procedure
```

1.5 Example

For the example lets take $N = 3528$ and $E = \{(0,1),(1,2)\}$.
So, number of digits will be equal to 4, the **BFS** tree for this example:



BFS Tree

So, the final answer given by the algorithm will be $(0,1),(1,2)$, i.e. 2 exchanges in this order.

1.6 Proof of Correctness

I'll do the proof for this algorithm by 2 claims.

Claim 1.1. *The above algorithm will give the path to the largest achievable number from the sequence N under the restriction of exchange operations provided by E.*

Proof. I will prove the above claim by contradiction.
Suppose the largest achievable number has the value x, the permutation of digits in the number x are from the result of doing a sequence of exchange operations from the set E (as given in the problem statement).
Since, we are considering x is not the number we gain from the algorithm, so 2 cases are possible.
Case I:

x is not the largest achievable number which is a contradiction to our assumption as we already assumed that x is the largest achievable number.

Case II:

The Max value which we get from the algorithm is not the largest achievable number which means the value x is either larger than Max or Max is not achievable by the set of exchange operations.
If we assume that Max is not achievable by the given set of exchange operations, then it is a contradiction since the new vertex is generated from the old vertex by performing only the accepted exchange operations given by set E (line 10 of Algorithm 2).
If we assume that x is larger than Max than it is also a contradiction, as if it was larger then the value of Max would have been replaced by that of x at line 16 of Algorithm 2.

Thus, by above 2 cases of contradiction, it is proved that the number we finally get is actually the largest achievable number. □

Claim 1.2. *The above algorithm will give the minimum number of exchange operations to reach the largest achievable number.*

Proof. The setup of the graph is such that each vertex is a possible permutation of the sequence and each edge will be an exchange operation from one vertex to other from the given set E.
So, the number of exchanges to reach the final number will be equal to the number of edges required to be travelled to reach the final node (Max).
The Algorithm implements the **BFS** algorithm to calculate the shortest path from the initial node to the Max node and since we get the shortest path, we will get the minimum number of exchanges possible, if there would have been less number of exchanges possible to reach the final node then the BFS algorithm would have given that as the shortest path.
Hence, the sequence of exchanges generated by the algorithm is the minimum number to reach the Max node. □

1.7 Time Complexity Analysis

The main portion of the code is the **BFS** algorithm and the graph is generated while the algorithm is in play.
Since, the time complexity of **BFS** algorithm is $O(V+E^{'})$, the maximum number of vertices possible is equal to the maximum number of permutations possible, thus, $V \leq 10^n$.
Also, the max number of edges will be equal to half the product of number of elements in exchange operation set and number of vertices since every edge is created by application of exchange on a vertex.
Thus, time complexity is $O(|E| \times 10^n)$.