CS3230 Tutorial 5 (Decrease and Conquer) Sample Solution

March 6, 2017

1 Question 1

The largest number of key comparisons will be, in particular, for strictly increasing or decreasing arrays:

$$C_{max}(n) = \sum_{i=1}^{n-1}(\lfloor \log_2 i \rfloor + 1) = \sum_{i=1}^{n-1} \lfloor \log_2 i \rfloor + \sum_{i=1}^{n-1} 1 \in \Theta(n \log n) + \Theta(n) = \Theta(n \log n)$$

It is the number of key moves, however, that will dominate the number of key comparisons in the worst case of strictly decreasing arrays. The number of key moves will be exactly the same as for the classic insertion sort, placing the algorithm's worst-case efficiency in $\Theta(n^2)$.

2 Question 2, Question 3, Question 4

The vertices and their corresponding binary values are:

$v_0 : 000$	$v_4 : 100$
$v_1 : 001$	$v_5 : 101$
$v_2 : 010$	$v_6 : 110$
$v_3 : 011$	$v_7 : 111$

The cube is illustrated in Figure 1.

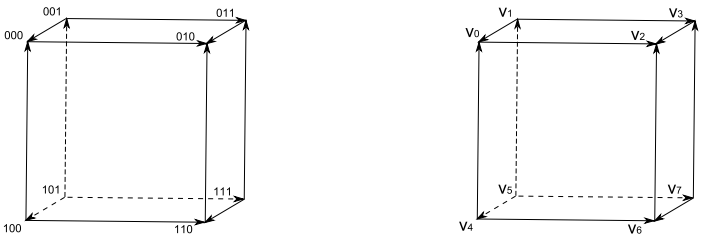

Figure 1: The Cube.

The DFS and BFS traversal are given in Figure 2.

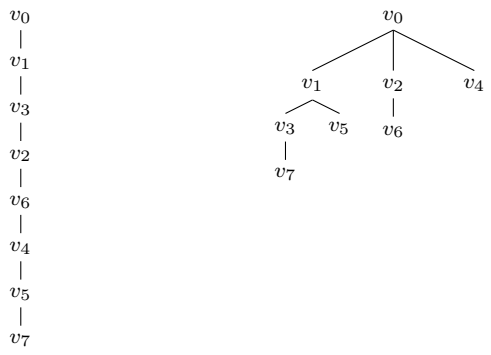

Figure 2: DFS(v_0) and BFS(v_0).

One possible solution to the topological sort is:

$$v_5, v_7, v_4, v_6, v_1, v_3, v_0, v_2$$

3 Question 5

Figure 3 is the maze and a graph representing it:

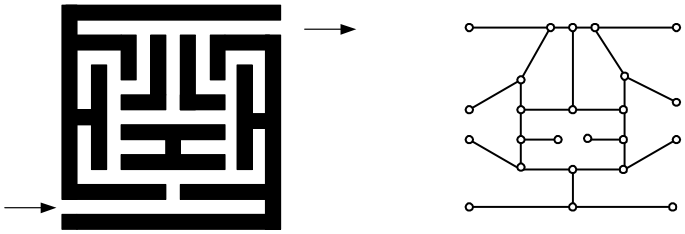

Figure 3: Model the maze with a graph.

Note that in the graph each vertex represents either a dead-end or a T-shape junction. We can label the vertices and perform the DFS/BFS traversal respectively (details omitted here).

(**Further Discussion**) DFS is much more convenient for going through a maze than BFS. When DFS moves to a next vertex, it is connected to a current vertex by an edge (i.e., "closely nearby" in the physical maze), which is not generally the case for BFS. In fact, DFS can be considered a generalization of an ancient right-hand rule for maze traversal: go through the maze in such a way that your right hand is always touching a wall.

4 Question 6

The problem can be solved by a recursive algorithm based on the decrease-by-1 strategy. Indeed, by asking just one question, we can eliminate the number of people who can be a celebrity by 1, solve the problem for the remaining group of $n-1$ people recursively, and then verify the returned solution by asking no more than two questions. Here is a more detailed description of this algorithm:

If $n = 1$, return that one person as a celebrity. If $n > 1$, proceed as follows:

Step 1 Select two people from the group given, say, A and B, and ask A whether A knows B. If A knows B, remove A from the remaining people who can be a celebrity; if A doesn't know B, remove B from this group. (*1 question needed*)

Step 2 Solve the problem recursively for the remaining group of $n-1$ people who can be a celebrity.

Step 3 There are 4 possibilities after Step 2:

1. Step 2 returns "no celebrity" among the group of $n-1$ people, the larger group of n people cannot contain a celebrity either.
2. Step 2 identified A as a celebrity. Just ask whether B knows A: return A as a celebrity if the answer is yes and "no celebrity" otherwise. (*1 question needed in this case*)
3. Step 2 identified B as a celebrity. Just ask whether B knows A: return B as a celebrity if the answer is no and "no celebrity" otherwise. (*1 question needed in this case*)
4. Step 2 identified as a celebrity a person other than either A or B, say, C. Now ask whether C knows the person removed in Step 1 and, if the answer is no, whether the person removed in Step 1 knows C. If the answer to the second question is yes," return C as a celebrity and "no celebrity" otherwise. (*2 questions needed in this case*)

The recurrence for $Q(n)$, the number of questions needed in the worst case, is as follows: $Q(n) = Q(n-1) + 3$ for $n > 2$, $Q(2) = 2$, $Q(1) = 0$.

Its solution is $Q(n) = 3n - 4$ for $n > 1$ and $Q(1) = 0$.

5 Question 7

Given the fact that

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

we can develop the following simple algorithm:

```
Combination (k, A[i...n])
{
  // base case
  if(k==1)
    return n possible combinations: {Ai,A(i+1),...,An}
  else if(k==n-i+1)
    return the single combination: Ai A(i+1) ... An

  // if A1 is selected, we select k-1 elements from the n-1 remaining elements
  S1 <- Combination(k-1, A[(i+1)...n])
  R1 <- append A1 as prefix to each element of S1

  // if A1 is not selected, we select k elements from the n-1 remaining elements
  R2 <- Combination(k, A[(i+1)...n])

  return (R1 union R2)
}
```

To start, simply call `Combination(k, A[1...n])` where `A` is the array of $\{1, 2, \ldots, n\}$.

Any bugs and typos, please report to Roger Zimmermann (rogerz@comp.nus.edu.sg).