Gurumanie Singh Dhiman (COMS3110 HW5) – netID: dhiman

(1)
```
updateMST(G, w, T, n, a){

      T.remove(n)

      Let C1, C2 be the two connected components of T

      (Do BFS starting from one endpoint of n to label C1)
      // O(|V|) for DFS

      minEdge <- n

      minWeight <- w(n) + a

      for each edge e = (u, v) in G.E:   // O(|E|)

           cutEdge = (u in C1 & v in C2) OR (u in C2 & v in C1)

           if (cutEdge is True) & (e != n) & (w(e) < minWeight):
           // O(1) per edge which is O(|E|) total

                minWeight <- w(e)

                minEdge <- e

      T.add(minEdge)

      return T

}
```

The runtime for the above code is O(|V| + |E|) or O(n+m).

(2)(a)

Level	Problem Size	# of Nodes	Work/Node	Level Total
1	$\frac{n}{5^0}$	1	$c \cdot \frac{n}{5^0}$	$c \cdot \frac{n}{5^0}$
2	$\frac{n}{5^1}$	1	$c \cdot \frac{n}{5^1}$	$c \cdot \frac{n}{5^1}$
3	$\frac{n}{5^2}$	1	$c \cdot \frac{n}{5^2}$	$c \cdot \frac{n}{5^2}$
i	$\frac{n}{5^{(\text{i-1})}}$	1	$c \cdot \frac{n}{5^{(i-1)}}$	$c \cdot \frac{n}{5^{(i-1)}}$
l	1	1	1	1

Since problem size is $\frac{n}{5^{(i-1)}}$ and we stop when subproblem size reaches 1 then:

$$\frac{n}{5^{(i-1)}} = 1 \Rightarrow n = 5^{(i-1)} \Rightarrow \log_5 n = \log_5 5^{(i-1)} \Rightarrow \log_5 n = i - 1 \Rightarrow \log_5 n + 1 = i$$

Sum the level totals:

$$c \cdot \frac{n}{5^0} + c \cdot \frac{n}{5^1} + c \cdot \frac{n}{5^2} + c \cdot \frac{n}{5^{(i-1)}} = c\left(\frac{n}{5^0} + \frac{n}{5^1} + \frac{n}{5^2} + \frac{n}{5^{(i-1)}}\right) = cn\sum_{i=1}^{\mathrm{l}} \frac{1}{5^{i\text{-}1}} = cn \sum_{\mathrm{j}=0}^{\log_5 n} \left(\frac{1}{5}\right)^j$$

$$Using \sum_{j=0}^{k} r^j = \frac{1 - r^{k+1}}{1 - r} \Rightarrow \sum_{j=0}^{\log_5 n} \left(\frac{1}{5}\right)^j = \frac{1 - \left(\frac{1}{5}\right)^{\log_5 n+1}}{1 - \frac{1}{5}} \Rightarrow \frac{5}{4}\left(1 - \left(\frac{1}{5}\right)^{\log_5 \mathrm{n}+1}\right)$$

$$\Rightarrow Simplify \left(\frac{1}{5}\right)^{\log_5 n+1} \Rightarrow \left(\frac{1}{5}\right)^{\log_5 \mathrm{n}} \cdot \left(\frac{1}{5}\right) \Rightarrow \left(\frac{1}{5^{\log_5 n}}\right) \cdot \left(\frac{1}{5}\right) \Rightarrow \frac{1}{n} \cdot \frac{1}{5} \Rightarrow \frac{1}{5n}$$

$$Then \frac{5}{4}\left(1 - \frac{1}{5n}\right) \Rightarrow \frac{5}{4}cn - \frac{c}{4} \quad which\ is \quad O(n)$$

(b)

Level	Problem Size	# of Nodes	Work/Node	Level Total
1	$\frac{n}{2^0}$	16^0	$\left(\frac{\mathrm{n}}{2^0}\right)^3$	$16^0 \cdot \left(\frac{\mathrm{n}}{2^0}\right)^3$
2	$\frac{n}{2^1}$	16^1	$\left(\frac{\mathrm{n}}{2^1}\right)^3$	$16^1 \cdot \left(\frac{\mathrm{n}}{2^1}\right)^3$
3	$\frac{n}{2^2}$	16^2	$\left(\frac{\mathrm{n}}{2^2}\right)^3$	$16^2 \cdot \left(\frac{\mathrm{n}}{2^2}\right)^3$
i	$\frac{n}{2^{(i-1)}}$	$16^{(\mathrm{i}-1)}$	$\left(\frac{\mathrm{n}}{2^{\mathrm{i}\text{-}1}}\right)^3$	$16^{(\mathrm{i}\text{-}1)} \cdot \left(\frac{\mathrm{n}}{2^{\mathrm{i}\text{-}1}}\right)^3$
l	1	$16^{l-1} = n^4$	1	n^4

Since problem size is $\frac{n}{2^{(i-1)}}$ and we stop when subproblem size reaches 1 then:

$$\frac{n}{2^{(i-1)}} = 1 \Rightarrow n = 2^{(i-1)} \Rightarrow \log_2 n = \log_2 2^{(i-1)} \Rightarrow \log_2 n = i - 1 \Rightarrow \log_2 n + 1 = i$$

Sum the level totals: $\sum_{i=1}^{\log_2 n+1} n^3 \cdot 2^{(i-1)}$

Then using geometric sums: $n^3 \cdot \sum_{i=1}^{\log_2 n+1} 2^{(i-1)} \Rightarrow n^3 \cdot \left(2^{\log_2 n+1} - 1\right) \Rightarrow n^3 \cdot (2n - 1)$

$$\Rightarrow O(n^4)$$

If I skipped steps in 2(a) or 2(b) you can deduct marks as needed, typing latex is very difficult to do step-by-step and too tedious.

(3)(a) We start at (0,0) and collect credits which are random C[0][0]. We end at (i, j) which does not need to be the southeast end (M, N) but can be any N in the last M.

So, the recurrence relation is:

RecurrSolution (i, j) =

{C[0][0], if i = 0 and j = 0

{C[i][j] + RecurrSolution of the max credits for { (i-1, j) – 3 if i > 0 else

{ (i, j-1) – 1 if j > 0 else

{ (i-1, j-1) – 2 if i > 0 and j > 0

(b)
```
FindingExit(C){
    // create the game graph
    m = C.row(), n = C.col(), arr = 2D m*n array      // O(1)
    parentArr = 2D m*n array      // O(1)
    for i=0 to m-1      // O(m)
        for j=0 to n-1      // O(n)
            arr[i][j] = - infinity
    arr[0][0] = C[0][0]      // O(1)
    for i=0 to m-1      // O(m)
        for j=0 to n-1      // O(n)
            if i+1 < m:    // moving south
                arr[i+1][j] = max(arr[i+1][j], arr[i][j] +
                C[i+1][j] - 3)
                parentArr[i+1][j] = (i,j)
```

```
            if j+1 < n:    // moving east
                arr[i][j+1] = max(arr[i][j+1], arr[i][j] +
                C[i][j+1] − 1)
                parentArr[i][j+1] = (i,j)
            if i+1 < m and j+1 < n:  // moving southeast
                arr[i+1][j+1] = max(arr[i+1][j+1], arr[i][j]
                + C[i+1][j+1] − 2)
                parentArr[i+1][j+1] = (i,j)
    maxCredits = − infinity
    bestCol = −1
    for j in range(n):
        if arr[m−1][j] > maxCredits:
            maxCredits = arr[m−1][j]
            bestCol = j
    path = []
    i, j = m−1, best_col
    while (i, j) != Empty:
        path.append((i, j))
        i, j = parentArr[i][j] if parentArr[i][j] is not empty
    return reverse(path)
}
```

(c) Since there are two nested loops where it is M*N so it evaluates to 2(M*N) and in the end we have another check at the end which takes (N) time. Therefore, the worst case runtime for this algorithm evaluates to O(M*N).