Assignment#2 Report

Team name: Yunfan Li, Haichao Zhang, Jingyuan Xu

Course: CS325-001

**Algorithm 1:**

Pseudocode:

1. N, M, T

2. key[M], ball[T], box[N], newKey[M]

3. index 🡨 1

4. recursive (level, state) // level means current key, state means whether we use

// this key or not

5. if level != M+1

6. if state = 1

7. newKey[index] = key[level]

8. index 🡨 index + 1

9. res 🡨 recursive (level+1, 1)

10. index 🡨 index - 1

11. res 🡨 recursive (level+1, 0)

12. index 🡨 index - 1

13. else

14. res 🡨 recursive (level+1, 1)

15. index 🡨 index - 1

16. res 🡨 recursive (level+1, 0)

17. index 🡨 index - 1

18. else //if level = M+1

19. front 🡨 calculate how many lockers should be unlocked before

the first key

20. back 🡨 calculate how many lockers should be unlocked after

the last key

21. for i 🡨 1 to (index-1)

22. k 🡨0

23. for j 🡨 newKey[i-1]+1 to newKey[i]

24. openRight[N], openLeft[N]

25. if box[i] = 1

26. openRight[k] 🡨 j - newKey[i-1]

27. openLeft[k] 🡨 newKey[i] - j +1

28. k 🡨 k+1

29. tmpMin 🡨 INFINITY

30. for j 🡨 1 to (k-2)

31. if tmpMin > openRight[k] + openLeft[k+1]

32. tmpMin = openRight[k] + openLeft[k+1]

33. min 🡨 min + tmpMin

34. res 🡨 front + min + back

35. return res

Time Analysis:

For the recursive process, we will find out all the possibility of keys combinations. So all sum of keys combination is 2^M. For every single keys combination, we search for every pair of keys, from left key plus 1 position to right key, calculate how many doors need to be opened in order to get all the balls between these two keys. This part of for-loop is N. So the total running time will be O(N2^M). The big-Omega appears when we do not have any balls in these lockers, so it is Omega(2^M).

Solutions:

Test1: 11

Test2: 14

Test3: 7

Test4: 14

Test5: 18

Test6: 1

Test7: 15

Test8: 8

**Algorithm 2:**

Pseudocode:

1. if do not have balls before the first key

2. d[1][0] 🡨 0

3. d[1][1] 🡨 1

4. else //have balls before the first key

5. d[1][0] 🡨 INFINITY

6. d[1][1] 🡨 key[1] - ball[1] + 1

7. for i 🡨 2 to M

8. if have ball between key[i-1] and key[i]

9. d[i][0] 🡨 min (d[i-1][0]+(minimum of doors need to be opened

between key[i-1] and key[i])+1, d[i-1][1]+ (minimum of

doors need to be opened between key[i-1] and key[i]))

10. for j 🡨 key[i-1]+1 to key[i]

11. for k 🡨 j-1 down to key[i-1]

12. if box[k] = 1

13. break;

14. if k = key[i-1]

15. leftNotOpen 🡨 0

16. leftOpen 🡨 0

17. else

18. leftNotOpen 🡨 k – key[i-1] + 1

19. leftOpen 🡨 k –key[i-1]

20. for k 🡨 j to key[i]

21. if box[k] = 1

22. break;

23. if k = key[i]+1

24. r 🡨 1

25. else

26. r 🡨 key[i] – k +1

27. sum\_leftNotOpen = min(sum\_leftNotOpen, r+leftNotOpen)

28. sum\_leftOpen = min(sum\_leftOpen, r+leftOpen)

29. d[i][1] = min(d[i-1][0]+sum\_leftNotOpen, d[i-1][1]+sum\_leftOpen)

30.if have balls after the last key

31. res 🡨 min(d[M][0]+ball[T]-key[M]+1, d[M][1]+ball[T]-key[M])

32 else

33. res 🡨 min(d[M][0], d[M][1])

Time Analysis:

For the most outer for-loop is M, and inner for-loops is NM, so the total running time is O(NM^2). The big-Omega appears when there is no balls in these lockers, so it is Omega(M).

Solutions:

Test1: 96

Test2: 22

Test3: 68

Test4: 31

Test5: 103

Test6: 30

Test7: 87

Test8: 83