Group Assignment #2

Kyle Nichols, Alex Bailey, Justin Pham

**Psuedocode**

The following is pseudocode for the enumerative and dynamic programming algorithms.

*Algorithm 1:*

result = 0;

for i=1 to end of locker size

if found a ball == 1 //found

for b = 0 to the found ball location

find the leftKeyLocation = 1

break;

end

for b = found ball location to the end of locker size

find the rightKeyLocation = 1

break;

end

if both key found

for j = left key location to the right key location

if ball found == 1

leftCost = ball location - left key location

rightCost = right key location - ball location

end

end

for k = left key location to the right key location

sumleft = 0, sumRight = 0

if ball found = 1

distanceKeyToBall = ball found  - leftKeyLocation

distanceBallToKey = rightKeyLocation - ball found

if distanceBallToKey > distanceKeyToBall

sumLeft = distanceKeyToBall

else if distanceBallToKey < distanceKeyToBall

sumRight = distanceBallToKey

break;

else if distanceBallToKey == distaanceKeyToBall

sumLeft = distanceKeyToBall

end

end

sum = sumLeft + sumRight

end

result += min(sum, min(leftCost, rightCost))

update the balls to be opened and keys used

else if key left found == 1 and key right found == 0

temp = 0

for j = key left location to locker size

if ball found = 1

temp = ball found location - left key location

end

end

result += temp;

update keys and balls to be used

else if key left found == 0 and key right found == 1

result += key right location - ball found location

update the balls and keys to be used

end

end

The algorithm runs through the locker size, finds the first ball, finds the keys, comparing which keys is most suitable to use. There are 3 cases, one is using both keys, second is using left key or third using the right key relatives to the balls within the range of both keys. In addition there are two more cases, which are begin and end cases where there is only one key available either left or right relative to the balls.

*Algorithm 2:*

MinLockers(balls[1,...,n], keys[1,...,n], n, B, K):

int d[B]

d[0] = 0;

for i = 1 to N

if balls[i] == 0 // There is no ball

d[i] = d[i-1]

else if balls[i] == 1 and keys[i] == 0 // There is a ball, but we don't have the key

distance = Dist(balls, keys, i, n) // How many lockers need to be opened to get to the key

d[i] = 1 + distance + d[i-distance]

else if balls[i] == 1 and keys[i] == 1 // There is a ball and we have the key

d[i] = 1 + d[i-1]

return d[n]

Dist(balls[1,...,n], keys[1,...,n], i, n):

// Search above the ball location

for j = i+1 to n-1

if keys[j] == 1

distance = j - i

if distance < minDistance

minDistance = distance

firstLoop = true

secondLoop = false

// Search below the ball location

for k = i-1 to 0

if keys[k] == 1

distance = i - k

if distance < minDistance

minDistance = distance

firstLoop = false

secondLoop = true

if firstLoop

for k = i to i+minDistance

keys[k-1] = 1

keys[k+1] = 1

else if secondLoop

for j = i-minDistance to i

keys[j-1] = 1

keys[j+1] = 1

return minDistance

The algorithm runs through the lockers, from 0 to N, and at each one checks the condition: there is no ball, the is a ball but no key, and there is a ball and a key for the locker. Depending on the choice, it will add to d[i] the appropriate value. There is also a function that finds the minimum distance between the ball and the closest key.

**Analysis**

The following is a runtime analysis for the enumerative and dynamic programming algorithms.

*Algorithm 1:*

The algorithm will loop through the number of lockers, n, looking for the ball then it will throught the lockers on the left and right, looking for a key. Since the algorithm will have to look through the lockers in every case, this will give the algorithm an upper and lower bound of **n^m.**

The run time from the first loop through n locker is: n

Within the first loop we have:

Two loops to find the keys, one from the location of the ball and from the ball to the location of the key: n + n

Using the range of the keys to decided which keys to use for each ball. There are three calculation for loops from location of the first key to the location of the second key.  m + m + m

*Algorithm 2:*

The lower bound for the algorithm is likely Ω(M), as the best case scenario is that your all the tennis balls are matched 1:1 with a given key. The worst case runtime is when all of the loops are run through completely. In our implementation of this, there is an outer loop that runs iteratively from the first locker to the last, and an inner set of for loops will do the same thing (though likely never will), followed by a smaller loop. This gives N(N+cN), with c being some constant. Ultimately, our upperbound is Ο(N2 + cN2) and Θ(N2).

**Solutions**

The following is a list of solutions for the input sequences provided for the enumerative and dynamic programming algorithms.

*Algorithm 1:*

dp\_set1: 11, 14, 7, 14, 18, 1, 15, 8

dp\_set2: 99, 22, 67, 31, 103, 30, 87, 83

dp\_set3: 401, 493, 389, 202, 377, 392, 268, 246

*Algorithm 2:*

|  |  |
| --- | --- |
| Test | Output |
| 1 | 191 |
| 2 | 23 |
| 3 | 89 |
| 4 | 31 |
| 5 | 146 |
| 6 | 33 |
| 7 | 87 |
| 8 | 85 |