Anthony Miller (milleant)
Taylor Fahlman (fahlmant)
Canvas Group #3

# Group Assignment #2

## Introduction

There are lockers in Dixon with Tennis balls in some of them. All of the lockers are locked and can be opened with keys provided to you. A key can only open a locker with the same number as the key. Once a locker is opened you can open the locker to the left or right of the current opened locker. Given a set of keys and a number of tennis balls within the lockers what is the minimum number of lockers that must be opened to get all of the tennis balls?

*Example:*

T = Tennis Ball
K = Key

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| | T | | T | T | T | | |
| k | | K | | | K | | K |

*Optimal solution: 5*
*SubOptimal solution: 6, 7*

## Algorithm 1

---

***Pseudocode***

```
for i from 0 to m
   combinations = all possible combinations of numbers from 0 to m

for each item in combinations
  #All lockers with keys start as opened
   for i from 0 to T
      for j from 0 to N

          if(lockers[j] == open):
             distances[i] = abs(tennisballs[i] - j)
         minimum_distance = min(distances)
         lockers from i to j of minimum_distance = open
      lengths[i] = sum # of open lockers
return min(lengths)
```

The pseudocode first finds all possible combinations of keys. Then for each combination of key, each tennis balls distance from each key is checked. The minimum distance between the first tennis ball is found, and all lockers between the ball and that key are ‘opened’. This continues for each tennis ball. Once each tennis ball has been found, the minimum open lockers is found from the collection of final open lockers.

***Analysis***

Our algorithm has a running time of O(TN2^m). This is due to finding all combinations of keys, then looping over each tennis ball and each locker. The best case would still be TN2^m, because the looping of tennis balls and lockers, as well as the combinations of keys happen every time.

***Solutions***

dp_set1.txt
Test 1: 11
Test 2: 14
Test 3: 7
Test 4: 15
Test 5: 18

Test 6: 1
Test 7: 15
Test 8: 8

## Algorithm 2

---

***Pseudocode***

```
# DP Table with each starting value set to higher than the maximum number of lockers
for each key
DP <- (n + 1) : 0 to m

# Base case
if M[0] < T[0]
   DP[0] = 0
else
   DP[0] = T[0] - M[0] + 1

for i <- 1 to m
   for j <- 0 to i
      leastopened = calc_least(M[i], M[j], T)
      if DP[j] + least < DP[i]
         DP[i] = DP[j] + least

return DP[m - 1]


calc_least(mi, mj, T)
   tmp_tennis = [ ]
   distances = [ ]
   for i <- 0 to len(T)
      if T[i] >= mj and T[i] <= mi
         append(tmp_tennis, T[i])
      for j <- len(tmp_tennis) to 0
         right_key = 0
         left_key = 0
         for x <- 0 to j
```

```
            if(x == 0)
                right_key += distance from closest tennis ball to left most key
            else
                right_key += distance between previous tennisball and current tennisball

        for y <- k to len(tmp_tennis)
            if(y ==0)
                left_key += distance from closest tennis ball to right most key
            else
                right_key += distance between previous tennisball and current tennisball
        distance[j] = right_key + left_key
    return min(distance)
```

This pseudocode is broken up into two functions. The first handles the base case of the DP table, and fills in each other value of the DP, then returns the last value of the DP table which should be minimum number of lockers to open. The calc_least takes two keys for locker mj and mi. Then it finds all the tennis balls that exist between between the two lockers. Once that's found, it loops through each subsequence of tennis balls, and calculates the minimum number of lockers to be opened to get all tennis balls between two keys. It then returns that minimum value. The calc_least function is run m^2 times, comparing each key with every other key.

**Analysis**

Our algorithm has a running time of $O(TM^2)$. This is due to the algorithm checking the smallest number of lockers that needs to be opened between each possible subset of keys and comparing the current value to the previous value stored within the DP table. Each possible subset would be $M^2$ and within each subset we compare each tennis ball to the keys within the subset leading to $T$ times. The best runtime would still be $TM^2$ since we have to iterate through each subset and tennis ball in each subset to find the optimal solution. In other words we have to compute the entire DP table before finding the optimal number of lockers to be opened.

**Solutions**

dp_set2.txt
alg2: Test 1 = 103
alg2: Test 2 = 27
alg2: Test 3 = 80

alg2: Test 4 = 31
alg2: Test 5 = 106
alg2: Test 6 = 32
alg2: Test 7 = 87
alg2: Test 8 = 80

**CONCLUSION**

The first algorithm takes an extremely long time to run or large numbers of keys since each possible iteration has to be checked. This makes it impossible to run large numbers of key due to the memory requirements alone to store every possible iteration. The second algorithm is much more efficient since we only need to store m^2 values to find the minimum length instead of 2^m.

The main difference between the 1st and 2nd algorithm is the 1st algorithm compares the distance from each tennis ball to every possible key iteration and the 2nd is only comparing the distance of each tennis ball between a subset of keys. The 2nd algorithm then decides to either use the current key or not based on the previous computed value of min distance between the key subset and tennis balls.

Run times for 1st would be 2^m for each possible combination and T * N for checking the path to each tennis ball through each locker, this end up being $O(2^m * TN)$. The 2nd algorithm loops over each subset of keys m^2 and checks the distance between each tennis ball and the 2 keys within the subset. This equates to T times for each subset or $O(TM^2)$