Name: Aisha Muhammad Nawaz

RollNo: 20L-0921

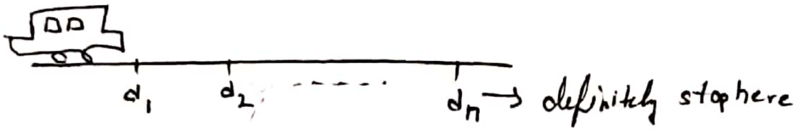
Section: BSCS 4A

DAA (Dr. Maryam Bashir)

Assignment /HW#7

Dynamic Programming

Problem 1



α miles a day → amount of loss = $(500-\alpha)^2$

Aim: Total loss minimize

(a).

For subproblem, we will find minimum Loss from optimal values of D_K (where K varies from $0 \le K < i$) + $(500-(d_i-d_k))^2$ (cost of travelling in one day) and store in D_i.

K are the stops that might have been stopped on the night before.

(b).

$$D[i] = \min_{0 \le K < i} \left(D[K] + (500 - (d_i - d_k))^2 \right)$$

(c).

```
int Findminloss ( int d[ ], int n)
{   int D = new int[n];   int sum = -∞
    for (int i=0 ; i < n ; i++)
    {
            min=-∞;  sum=-∞
        for(int K=0; K < i ; k++)
        {
            sum = D[K] + (500-( D[i]-D[k]))^2
            if ( min > sum)
            { min = sum; }
        }
        D [i] = min;
    }
    return D [n-1];
}
```

(d). Create a 2-D array instead of a 1-D (array D[i]) and use D[i][j] and store value of K for each cell which minimize D[i].

Problem #2

(1). for each index in the array we will find longest increasing with decreasing subsequence and add it to bigger problem later on. We will create two array one to store increasing and the other to store decreasing subsequence for each index i in the array.

(2).

```
if (A[i] > A[j] && List[i] < List[j]+1)
  List[i] = List[j]+1;
```

(3).

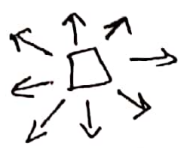
```
int SubsequenceFinder (int A[], int n) {
  int List[n], List2[n];
  for (int i=0; i<n; i++) //initializing.
    { List[i] = 1;
      List2[i] = 1; }
  for (int i=1; i<n; i++)
    for (int j=0; j<i; j++) {
      if (A[i] > A[j] && List[i] < List[j]+1)
        List[i] = List[j]+1;

  for (int i=n-2; i>=0; i--)
    for (int j=n-1; j>i; j--)
      if (A[i] > A[j] && List2[i] < List2[j]+1)
        List2[i] = List2[j]+1;

  int Ans;
  Ans = List[0] + List2[0] - 1;
  for (int i=1; i<n; i++)
    if (List[i] + List2[i] - 1 > ans)
      ans = List[i] + List2[i] - 1;
  return Ans;
}
```

(4). $O(n^2)$

Problem #3

Algo (per block): for each index in the m*m array find max length, if we go right, left, top, bottom, diagonally left, diagonally right

(a).



maximum [i,j] = max { maximum [i][j+1] // horizontally right
maximum [i][j-1] // horizontally left
maximum [i-1][j] // vertically up
maximum [i+1][j] // vertically down
maximum [i-1][j-1] // diagonally left backwards
maximum [i+1][j+1] // diagonally right forward
maximum [i+1][j-1] // diagonally right back
maximum [i-1][j+1] // diagonally right forward } + A [i,j]

(b). O(m*m) or $O(m^2)$

(c).

$O(m*m*m)$ or $O(m^3)$

Problem #4

(a). $D_{i,j} = D_{i-1,j} + (d_{j+i+3})$ = 3rd digit after previous sum ended index

* j means starting at index

Check if prime (D[i,j])
where j varies from 1 to 3 (for 3 consecutive) (for 3 consecutive)

(b). O(dn) where d is the number of digits of the number and n is the number itself.

(c).
```
bool checkprime conditions (int n) {
int sum=0; int i=1; int k=0;
for (int d=n%10; d!=0; d=d/10) { // This loop checks
    sum = sum + d;                 conditions
    if (i%3==0)                    for 3 consecutive.
        D[0][k]=sum;
        k++;
    if(checkprime(sum)==false)
        return false;
        sum = 0;
```

```
Sum = 0        int K = 0;
for (int i = 0; d = n%10 ; d != 0; d = d%10, i++)   // this loop checks conditions for 4 consecutive
{
        if (i %4 == 0) {
    Sum = sum + D[0][K] + d;

        if (checkprime (Sum) == false)
            return false;   // returns false if even one condition false.
        else
                D[1][K] = sum;
                Sum = 0;  K++;
    }
        }

for (int i = 0; d = n%10; d != 0; d = d%10; i++)    Sum = 0; K = 0;
{                                    // This loop checks conditions for 5 consecut
    if ( i %5 == 0) {
    Sum = Sum + D[1][K] + d;
        if ( checkprime (Sum) == false)
            return false;
        else
        { D[2][K] = sum;
              K++;
            Sum = 0;
        }
    }
  }
return true;
}

bool checkprime (int n) {   // This function check if number n is prime.
                            returns true if it is otherwise, false.
    for (int i = 2 ; i < n; i++) {
        if (n % i == 0)
            return false;
    }
  return true;
}
```

Problem #5

$$a = C[i - s[j]][j]$$
$$b = C[i][j-1]$$
$$C[i][j] = a + b;$$

(or)

$$C[i][j] = sum \begin{bmatrix} C[i - C[j]][j] \\ C[i][j-1] \end{bmatrix}$$

(E).

O(mn) where m2 is amount and m is number of coins more available.

(F).

```
// code similar to binary knapsack problem   // this array is for coins.
int CountNoOfWay( int m, int m2, int C[])
// m2 is value/Amount // m is no of coins.
    int Values[m2+1][m];   // creating 2-D array that will be filled in bottom up manner.
// m2+1 because there is a base case counted of amount = 0
for(int i=0; i<m; i++)
    Values[0][i] = 1;   // intializing base case as 1 when amount is zero, no coins selected. (one way)
for(int i=1; i<m2+1; i++)   // filling up values array.
{
    for(int j=0; j<m; j++)   // one by one increasing the amount upto total m
    {
        if(i - C[j] >= 0)   // if the coin choosen is equal to or greater than the coin we are currently on
            a = Values[i - C[j]][j];   // save the values in a
                // for this same amount number of coins choosen excluding the currently one.
        else
            a = 0;
        if(j >= 1)   // Check to see if amount is greater than 1
            b = Values[i][j-1]
                // for this same amount number of coins choosen except the current one.
        else
            b = 0;
        Values[i][j] = a + b;
    }
}
return Values[m2][m-1];
```