CS 721: Advanced Algorithms & Analysis
Homework – 3

Chakradhar Reddy Donuri
E949F496

1Q. (20 points) Edit distance: Given two strings x[1; : : : ;m] and y[1; : : : ; n] of length m and n respectively, a natural way of measuring distance between them is the extent to which they can be aligned or matched up. Edit distance attempts to measure this alignment by applying minimum number of edits to the first string x so that the after the edits x is transformed to y. Edits are applied on x in the form of insertion, deletion and substitution. For example, one possible way of aligning the string x = SNOWY with string y = SUNNY is writing them as S _NOWY and SUNN _ Y . Now In position 2 we can inset U, in position 4 we can substitute O with N and position 5 we van delete W. These edit operations will transform the first string to the second string. If cost each of the insertion, deletion and substitution operation is 1 then the edit distance is 3. You will write a dynamic programming solution to find edit distance automatically. Let D(i; j) is the solution of the subproblem of finding edit distance between string x[1; : : : ; i] and y[1; : : : ; j]. Then our goal is to find D(m; n). First write D(i; j) in terms of solution of smaller subproblem (optimal substructure) and give a dynamic programming solution to find minimum edit distance between x[1; : : : ;m] and
y[1; : : : ; n]. What is the running time of your solution?

Answer :-

D(i,j) = min {1+D(i-1, j), 1 + D(i, j-1), diff (i, j) + D (i-1,j-1)}

Where diff (i, j) = 1, **if x [i] ≠ y[j]**
diff(i, j) = 0, **otherwise**

```
for i= 0, 1, 2, ….. m {
        D( i, 0) = i
        }
for j= 1, 2, …. n {
        D( 0, j) = j
        }
for i= 1, 2, …. m {
         for j = 1, 2, …. n {
                D( i, j ) = min {D (i-1, j) + 1, D (i, j-1) + 1,D( i-1, j-1) + diff (i,j) }
                }
        }
return D( m, n)
```

Therefor run time of the above is given as = **Θ(mn)**

2Q. (10 points) Knapsack with repetition: During a robbery, a burglar finds more loot than he expected and he has to decide what to take. His bag (\knapsack") can hold a total weight of at most W pounds. There are n items to choose from, of weight w1; : : : wn of dollar value v1; : : : ; vn respectively. What is the maximum dollar value of items he can _t into his bag? Denote by K(w), the maximum dollar value items(s) of weight at most w that can be _t into the knapsack. Our goal is to find K(W). First write K(w) in terms of solution of smaller subproblem (optimal substructure) and give a dynamic programming solution to find K(W). What is the running time of your solution?

Answer:-

- Let us consider Knapsack with Repetition
- In this case we can shrink the original problem to smaller knapsack capacities $w \leq W$
  K(w) = maximum value achievable with a knapsack of capacity w.
- we express this in terms of smaller subproblems if the optimal solution to K(w) includes item ' i ', then removing this item from the knapsack leaves an optimal solution to $K(w - w_i)$
- In other words, K(w) is simply $K(w - w_i) + v_i$, for some i.
  $K(w) = \max I : w_i \leq w \{K (w - w_i) + v_i\}$
- where as usual our convention is that the maximum over an empty set is 0
  Algorithm:

```
K(0) = 0
 for w=1 to W
        K(w)=0
         for i=1 to n
                if (wi <=w)
                        K(w)=max { k(w), k( w-wi) + vi}
 return k(W)
```

- This algorithm fills in a one-dimensional table of length W + 1, in left-to-right order. Each entry can take up to O(n) time to compute, so the overall running time is **O(nW).**

3Q. (20 points) Suppose you are managing construction of billboards on the 21st St. N. which runs from east to west on a straight line. There are n possible sites for billboard construction given in the array x1; : : : ; n], where 0 _ x[1] < x[2] < _ _ _ < x[n] specifies the distance of each possible billboard location from the west side end of 21st St. N. There is also an array p[1; : : : n] which contains the payment information, i.e., if you place a billboard at location x[j] you receive payment p[j].
Restrictions imposed by the Sedgwick county requires that any pair of billboard must be more than 3 miles apart. You would like place the billboard at a subset of sites so as to maximize your revenue, subject to Sedgwick county's placement restriction. For example, if n = 4, x = [3; 4; 8; 9] and p = [5; 6; 5; 1] then optimal solution will place billboards at x[2] and x[3] with revenue p[2] + p[3] = 6 + 5 = 11.
Suppose you are also given the array prev [1; : : : ; n] where prev [j] stores the location index of the previous billboard site (to the west of x[j]) that satis_es Sedgwick county's billboard

restriction, i.e., prev[j] = max fi : i < j and x[i] < x[j] − 3g. Let R(j) denote the optimal revenue obtained by placing billboards at a subset of locations x[1]; x[2]; :::; x[j] satisfying Sedgwick county's restriction. Then our goal is to find R(n). First, write R(j) in terms of solution of smaller subproblem (optimal substructure) and give a dynamic programming solution to _nd R(n) that takes input x[1; : : : ; n]; p[1; : : : ; n] and prev[1; : : : ; n]. What is the running time of your solution? For convenience assume prev[1] = 0 and R(0) = 0
Answer:-

R(j) = max {R (j-1), R(prev [j]+ p[j] }

where R(j-1) means don't place a billboard at x[j] and R(prev [j]) + p[j] means place the billboard at x [j]

```
R[0] = 0
 for j=1 to n {
          R(j) = max { R(j-1), R( prev[j] + p[j] }
          }
 return R(n)
```

Therefor run time of above is given as **Θ(n)**