EE4371 Data Structures and Algorithms Final Q1 2019

Rishhanth Maanav V
EE16B036

1. Introduction

In this problem, we solve the modified Knapsack problem defined as follows,

$$\max_{x_1,x_2,\ldots x_n} \sum_{i=1}^{N} x_i \log_{100} w_i$$

$$\text{s.t } \frac{\sum_{i=1}^{N} x_i w_i}{\sqrt{k}} \leq 10000$$

where k is the number of items chosen and each of $x_i \in \{0,1\}$. The weights w_i are positive integers.

2. Algorithm

Algorithm 2 Procedure Knapsack

```
1:  Input array of weights arr_w $w_i$, no of weights N, constraint W
2:  Initialize obj = 0        ▷ total sum of log of weights until step k
3:  Initialize sumw = 0  ▷ total sum of weights until step k
4:  Init k=0
5:  Set ChosenW = $\Phi$                ▷ set of chosen nodes
6:  Sort array of weights arr_w in descending order
7:  while k <N do
8:      for w in arr_w do
9:          if w + sum $\leq \sqrt{k+1} * W$ and index not in ChosenW then
10:             Add w to ChosenW
11:             sumw := sumw + w
12:             obj := obj + $log_{100} w$
13:             k := k+1
14:             print w
15:             break inner loop
16:     if no w in arr_w satisfies the if condt. for k then
17:         break outer loop
18: Output maximised sum of weights obj
```

3. Pseudo-Code

```
Procedure knapsack(int arr_w[], int N, int W){
    # sort weights array
    quicksort(arr_w)
    Initialize objec=0
    Initialize chosen[N] = {0}
    # to store chosen weights
    Initialize sumw = 0
    # iterate through all weights
    for k from 0 to n-1
    {   # if flag to check if weight found
        initialise flag = 0;
        for i from 0 to n-1 {
        {
            # checking if weight is chosen
            if  chosen[i]==0
            # checking constraint
                if (arr_w[i]+sumw <= sqrt(k+1)*W)
                {
                    # constraint valid
                    # found a weight for k
                    flag=1;
                    # update summation w_i
                    sumw:=sumw + arr_w[i];
                    # print the chosen weight
                    print(arr_w[i])
                    #index is marked chosen
                    chosen[i]=1;
                    # update value
                    objec:=objec+log100(arr_w[i])
                    break;
                }
        }
    # break outer loop if no new weight is found
        if (check==0)
            print(k); # print k
        }
    }

    return objec
}
```

4. Number of Conditions and Time Complexity

The algorithm checks conditions for satisfying constraints and to breaking the outer loop over number of items chosen. In the inner loop, we have constraint checking statement once and the breaking condition in the outer while loop once. The nested loop runs for the worst case as $O(N^2)$ with the inner loop being $O(N)$ and the outer loop being $O(N)$.

The time complexity of the algorithm in the worst case when all weights are checked is at every step k $O(N^2)$. For the best case when all weights satisfy the constraints at every step k, it is $O(N)$.