EE4371 Data Structures and Algorithms

Knapsack Problem

Sanjana S Prabhu -EE17B072

December 9, 2019

1 Introduction

Given a set of N objects of positive integer weights wi, we have to find the subset of these objects that maximizes:

$$\sum_{i=1}^{N} x_i \log_{100} x_i \tag{1}$$

subject to the condition:

$$\sum_{i=1}^{N} \frac{w_i x_i}{\sqrt{k}} <= 10000 \tag{2}$$

where $x_i = 0$ if w_i is not used and 1 if it is used. I have used dynamic programming to do the same, and the pseudocode is illustrated in the next section.

2 Algorithm and Pseudocode

2.1 Algorithm

The following steps were followed:

- For each value of maximum weight from 1 to the maximum weight given multiplied by root of total number of objects(condition in which all objects were selected), I called the knapsack function.
- The knapsack function generates a table and outputs the weights to be selected for the particular value of total weight, as well as returns the total number of weights selected, in turn maximising the value of the weights in the knapsack, and storing these weights and the total value in the array, known as answer array.

- Now for each of these total weights, we have a total maximum value possible, and all the weights used, stored in rows of the two-dimensional array, that we create everytime we call the knapsack function.
- For each of these rows, we have to check if the sum of the weights divided by the root of the total number of weights selected, is less than 10000, which is the condition required.
- Starting from the maximum value of total weight(approximately 10000*sqrt(908) for this problem), we perform our iteration.
- First we pass the total weight through the checkpath function, which using the knapsack array, returns the pointer to an array, whose first element is the total number of weights required to create that particular sum, while generating the maximum value possible, and its other elements are the weights.
- Next we check if this combination obeys condition (2).
- If yes, then we break from the loop and this is the combination of weights, that optimises (1), while satisfying (2).

2.2 Pseudocode

```
 k=pow(n,0.5)*10000;
 for i=1 to n:
     for j=0 to k:
         if w[i]>j:
             m[i,j]=m[i-1,j]
         else:
             m[i,j]=max(m[i-1,j],m[i-1,j-w[i]]+w[i])
int *ans;
int index;
for(int j=k;j>=0;j--){// start checking from maximum
total weight if the condition is satisfied
ans=checkpath(wt_size,j);// to find the total number of
weights selected for that particular value of total
weight

if((10000*pow(ans[0],0.5))>j)// limiting condition
{
   index=j;
   break;// once the condition is satisfied, we can
   stop, since this will be the optimal value
}
}
```

```
function checkpath (int size, int j){
  int k=1;
  while( size>0 & j>0)
  {
   if(value[size-1][j]==value[size][j])
    size--;

    else
    {
  path[k]=size;
     k+=1;
     j=j-wt[size];
     size--;
    }
  }
  path[0]=k-1
  return path
}
```

3 The code

The above pseudocode is written as a code, in the above file(ee17b072.c).
The output is as follows:

3.1 Output

```
Enter the filename to be opened
input1.txt
Total value is 367.757812.
Number of elements in sack is 292.
Selected weights are:
2       3       4       5       6       7       8       9       11      12      13      14      15      16      17      18      19      20   2
1       22      23      25      26      28      29      31      33      35      37      38      39      43      44      45      47      49   5
1       53      54      56      62      65      66      68      72      78      81      84      86      87      94      98      100     105  1
07      109     111     112     115     116     117     127     130     133     136     147     154     160     161     164     174     178  1
85      187     196     198     199     200     205     216     218     221     227     229     233     236     242     248     250     252  2
57      258     261     262     270     271     274     280     281     282     300     305     309     314     316     323     326     330  3
31      333     340     342     344     345     350     351     354     365     377     378     379     381     382     391     397     399  4
00      404     406     408     411     415     417     424     437     441     448     451     460     472     474     478     479     482  4
89      498     507     516     526     534     538     543     552     553     556     562     572     576     582     594     602     629  6
43      644     645     647     656     659     665     669     675     678     683     687     696     705     710     715     732     735  7
40      742     743     749     750     754     756     766     768     773     795     807     815     827     829     830     842     843  8
45      848     850     865     871     872     873     874     877     885     891     904     913     917     925     933     938     943  9
48      949     955     967     970     982     983     992     994     999     1003    1008    1009    1011    1013    1023    1028    1031 1
036     1040    1045    1048    1050    1052    1058    1062    1063    1065    1078    1088    1091    1099    1107    1113    1121    1135 1
139     1169    1183    1184    1203    1216    1217    1220    1223    1227    1248    1260    1266    1269    1272    1300    1305    1310 1
318     1339    1342    1348    1362    1366    1371    1381    1386    1388    1395    1400    1413    1416    1427    1436    1449    1451 1
454     1462    1490    1500
```

Figure 1:

4 Time-complexity and conditions to be checked

- First we perform the dynamic knapsack solution for all values from 1 to the maximum W(10000*sqrt(908)). This is O(nW).
- Then we check if any of these satisfy condition (2), we call the check-path function(of time-complexity O(max(n,W))), W times. The time complexity of this step comes out to be $O(n^2 + W^2)$.
- Since these two loops are performed seperately, the total time-complexity is $O(n^2 + W^2 + nW)$.