Homework 3
Colorado CSCI 5454

Sai Siddhi Akhilesh Appala

November 30, 2022

People I studied with for this homework: None

Problem 1

Question
In Rod Cutting Problem with Unique Lengths, what is the total profit we can get from cutting up the rod and selling the pieces? And to achieve that profit, how many pieces should we cut, and of what lengths? Give a DP solution and identify all the components.

solution
Subproblem: Now we are considering a 2-d array to optimize the previous algorithm without duplicating. Specifically, let K[n, w] be the maximum value one can obtain from a knapsack of size w using only items from the subset 1, . . . , n.

proof
To find the optimal solution with the items 1, . . . , n, and capacity w, If we don't, then the optimal solution uses only items 1, . . . , n 1, so its value is C[n-1,w]. we are only allowed to use items 1, . . . , n1 because we just used item n. So the remaining value is C[n-1,w-wn], and our total value is Vn+C[n-1,w-wn]. Note this is only possible if $wn > w$, as otherwise, item n cannot fit. Since these are the only two possibilities (or only one possibility if $wn < w$), and the recurrence chooses the best of both, it is optimal.

Reconstruction If we are cutting the rod of current length m into rod length l, then we will make prev[l][m] is 1. To reconstruct, we will check the array from length L, we will check if it is included by checking for 1 in that column and if we find it add it to the list. when we find subtract that particular l from the Length MaxLen and repeat the process until max Len becomes zero

Recurrence Relation:

- Base Case: C[0,n] = 0, C[i,0]=0
- Inductive Case: P[i,k] = max{C[i,n-1], C[i-n,n-1]+v[n] $\forall i > n$ }

Algorithm 1 Rod cutting - without duplicate

```
Input: Rod of length L, Integer array V of length L.
lets create an array of arr[L + 1][L + 1] for MaxProfitValue
lets create an array of prev[L + 1][L + 1] for traceback(reconstruction)
initialize arr[0][0...L]=0 and arr[0....L][0]=0 set MaxLen = L
for Iterate i from 1 to L  do
   for Iterate j from 1 to L do
      if l¡m then
         arr[i][j]=arr[i-1][j]
      else
         arr[i][j]=max(arr[i-1][j],arr[i-1][i-j]+V[j]
         if arr[i-1][j] ¡ arr[i-1][i-j]+V[j then
            prev[i][j]=I
         end if
      end if
   end for
end for// Below for reconstruction
for  i = L; i >= 0 ;i– do
   if arr[i][MaxLen] == 1 then
      add I to the list
   end if
   MaxLen=MaxLen-i
end for
Return arr[L][L]
```

Problem 2

```
public class knapsackProblem {
  private static int returnMax(int i, int j) {
    return (i > j) ? i : j; // compares and returns the max
  }
```

```
  private static int returnAfterKnapsack(int W, int len, int[] values, int[]
      weights) {
    System.out.println("max value for " + W + " and " + len + " is ");
    int maxValue[] = new int[W + 1]; // initiates values with W+1
    maxValue[0] = 0; // and covers base case for 0.
    for (int i = 0; i <= W; i++) {
      for (int j = 0; j < len; j++) {
        if (weights[j] <= i) {
          maxValue[i] = returnMax(maxValue[i], maxValue[i - weights[j]] +
            values[j]); // the inductive case which will evaluate the max value.
          // System.out.println("reconstruction path " + maxValue[i] );
        }
      }
    }
    return maxValue[W]; // final solution
  }
  public static void main(String[] args) {
    int W = 11054;
    int values[] = {2, 3, 5, 8, 13, 21, 34, 55, 89, 144};
    int weights[] = {2, 2, 3, 4, 7, 12, 20, 33, 54, 88};
    int len = values.length;
    System.out.println(returnAfterKnapsack(W, len, values, weights));
  }
}
```

---

results for 5 digits :

```
int W = 11054;
int values[] = {2, 3, 5, 8, 13, 21, 34, 55, 89, 144};
int weights[] = {2, 2, 3, 4, 7, 12, 20, 33, 54, 88};
int len = values.length;
System.out.println(returnAfterKnapsack(W, len, values, weights));
```

```
DAA — -zsh — 80×24
(base) ssakhilesha@SS-Akhilesh-MacBook-Pro DAA % javac knapsackProblem.java
(base) ssakhilesha@SS-Akhilesh-MacBook-Pro DAA % java knapsackProblem.java
max value for 11054 and 10 is
22107
(base) ssakhilesha@SS-Akhilesh-MacBook-Pro DAA %
```

Results for 9 digits:

```
int W = 110541585;
int values[] = {2, 3, 5, 8, 13, 21, 34, 55, 89, 144};
int weights[] = {2, 2, 3, 4, 7, 12, 20, 33, 54, 88};
int len = values.length;
System.out.println(returnAfterKnapsack(W, len, values, weights));
```

```
DAA — -zsh — 80×24
(base) ssakhilesha@SS-Akhilesh-MacBook-Pro DAA % javac knapsackProblem.java
(base) ssakhilesha@SS-Akhilesh-MacBook-Pro DAA % java knapsackProblem.java
max value for 110541585 and 10 is
221083168
```

part b with reconstruction

```
import java.io.*;
import java.util.*;
public class knapsackProblem
{
 private static int returnMax(int i, int j)
 {
   return (i > j) ? i : j; // comapres and returns the max
 }

 private static void reconstruction(int set[], int W , int wt[]){
    int weight=W;
        while(weight>0)
        {
            System.out.println(wt[set[weight]] + "\n");
           weight=weight-wt[set[weight]];
        }
        System.out.println("\n");
 }
 private static void returnAfterKnapsack(int W, int len, int[] values, int[]
     weights){
 int set[]=new int[W+1];
//   System.out.println("max value for " + W + " and " + len +" is" );
  int maxValue[] = new int[W + 1]; // initiates values with W+1 and covers base
      case for 0.
  int D[]=new int[W+1];
  for(int i=0;i<=W;i++){
    D[i]=-1;
  }
```

```
 maxValue[0]=0;   // base case for length 0, then max value will be 0
 for(int i = 0; i <= W; i++){
   int k=0,max_wt=0;
  for(int j = 0; j < len; j++){
   if(weights[j] <= i){
       maxValue[i] = returnMax(maxValue[i], maxValue[i - weights[j]] +
          values[j]); // the inductive case which will evaluate the max value.
       max_wt = j;
   // System.out.println("reconstruction path " + maxValue[i] );
   }
  }
  set[i]=max_wt;
 }

 System.out.println("max value for " + W + " and " + len +" is " + maxValue[W]
    +"\n"); // final solution
 reconstruction(set,W, weights);
}
public static void main(String[] args)
{
 int W = 11054;
 int values[] = {2, 3, 5, 8, 13, 21, 34, 55, 89, 144};
 int weights[] = {2, 2, 3, 4, 7, 12, 20, 33, 54, 88};
 int len = values.length;
 returnAfterKnapsack(W, len, values, weights);
}
}
```

Problem 3

Part a

This graph is not a flow as it is failing for Net flow constraint, From the definition, Net flow constraint states that the total incoming flow of an intermediate node must be 0. Here we have 3 and -4 which sums to -1. not 0, so not a flow

part b

This graph is not a flow as it is failing for capacity constraint. By definition, capacity constraint states that, for every edge e, the max flow for that edge must be less than the capacity for that edge. $f(u, v)c(u, v)$. here in the given graph the capacity of an edge from u to t is 5 but the flow is 6 which is not possible.hence its not a flow

part c

This graph is not a flow as it is failing the skew-symmetry constraint. From definition For all u,v: we have f(u,v) = -f(v,u). here in the given graph the edge $s \rightarrow u$ has 3 while edge $u \rightarrow s$ has -4 and as both are not equal, the Skew symmetry constraint is failing. hence its not a flow