Homework 3
Colorado CSCI 5454

RITHIK KUMAR ATHIGANUR SENTHIL

September 14, 2022

People I studied with for this homework: Ashwin
Other external resources used: Youtube

Problem 1 (4 points)

In the Rod Cutting with Unique Lengths problem, we are given a rod of valuable material of integer length m. We can cut the rod into pieces of integer lenghts. We are given the profit from selling a piece of each length, v[i] for each i = 1, . . . , m. However, we can only sell at most one piece of a given length. For example, a piece of length 4 can be sold as one piece of length 4 or a piece of length 3 plus a piece of length 1. However, it can not be sold as two pieces of length 2, nor as four pieces of length 1. The problem: 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?

Question: Give a dynamic programming solution to Rod Cutting with Unique Lengths. Identify all of the components of your dynamic programming solution, including reconstruction. Briefly prove correctness.

Solution:
To solve the problem using Dynamic Programming, we will create a 2-D array R of length $[n+1][n+1]$ to store the profit values.
Input: Take input of an array A which contains the profit values of cutting the rod at that length which is $index + 1$.

Output: By taking only a unique length of cuts, we have find out the maximum profit we can gain.

Subproblem: $R[i][j]$ gives the maximum value for a rod of length i when we cut the rod for any length from $1, 2...., j$.
Computing final answer: Using the profits from array A, we can compute the value to

all the cells $R[i][j]$ and the final answer $R[n][n]$ which will give the final answer for maximum profit of length n.
Recurrence:

- **Basecase:** $R[0][j] = 0$ implies profit will be 0 for rod of length 0.
  $R[i][0] = 0$ implies profit will be 0 for profit value 0.

- **Inductive Case:**
  if $i \geq j$,

$$R[i][j] = max(R[i][j-1], R[i-j][j-1] + A[j])$$

  else,

$$R[i][j] = R[i][j-1]$$

Algorithm 1 Rod cutting problem with unique lengths

```
take input of a non-empty array 'A' of n integers
initialize an array for DP R[n+1][n+1]
for i in 0,1,2,....,n do
   set $R[i][0] = 0$
end for
for j in 0,1,2,....,n do
   set $R[0][j] = 0$
end for
for i in 1,2,....n do
   for j in 1,2,.....n do
      if $i \geq j$ then
         set $R[i][j] = max(R[i][j-1], R[i-j][j-1] + A[j])$
      else
         set $R[i][j] = R[i][j-1]$
      end if
   end for
end for
return $R[n][n]$ which is the maximum profit and call reconstruction function.
```

Reconstruction: To reconstruct the object, we need to create another array prev[] where,

$$prev[j] = i, \text{ if } R[i-j][j-1] + A[j] \geq R[i][j-1]$$

setting this to the prev array we can get the maximum profit value and see in which index this value is present in the value array and increase the count of that value in a map and we can trace back and get all the lengths of the rods and it's occurance which we have stored in the map.

Correctness: By using the base cases, we have already computed the maximum profit for the indexes $R[i-j][j-1]$ and $R[i][j-1]$ and using this we can find the maximum profit for $R[i][j]$ using induction, by solving the subproblems we would have correctly computed the maximum profit of $R[n][n]$.

Problem 2 (8 points)

0.1 Part a (8 points)

In a programming language of your choice, code up the algorithm for the knapsack problem (duplicates allowed). Include comments in your code highlighting where you use the "components of a DP algorithm". Attach the source code to the end of your PDF submission (for example, print the source file to PDF, then concatenate the two PDFs).
Your code does not need to reconstruct the optimal set (but see the bonus problem below).
Run your program on the following inputs and report the output:.
$v = [2, 3, 5, 8, 13, 21, 34, 55, 89, 144]$
$w = [2, 2, 3, 4, 7, 12, 20, 33, 54, 88]$
W = the first five digits of your student ID (usually a nine-digit number starting with 10 or 11).
The same values for v and w, but W = all nine digits of your student ID.

Solution:

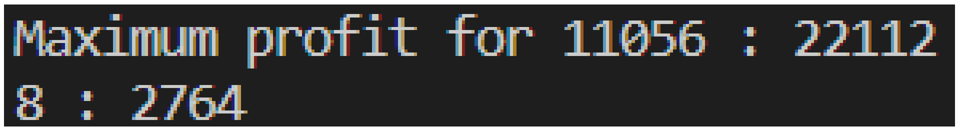
```
#include <bits/stdc++.h>
using namespace std;

void reconstruct(vector<int> v, vector<int> w, vector<long> prev, int W){
  map<int,int> m;
  //Initialize a map to store the number of times a item was chosen
  while(W != 0){
    m[v[prev[W]]]++; // find the index in the value array and increase it's count
    W-=w[prev[W]]; //decrease the weight to get the next index.
  }
  auto it = m.begin();
  while(it != m.end()){
    cout<<it->first<<" : "<<it->second<<endl;
    // print the items and the number if times it was taken into account.
    it++;
  }
}

void findUsingDP(vector<int> v, vector<int> w, int W, int n)
{
  vector<long> dp(W + 1, 0);
  /* Initialize a 1D arrray for DP and set the values to 0,
     With a size of W+1 */
  vector<long> prev(W + 1, 0);
  /* Initialize a 1D array for storing the indexes of values
```

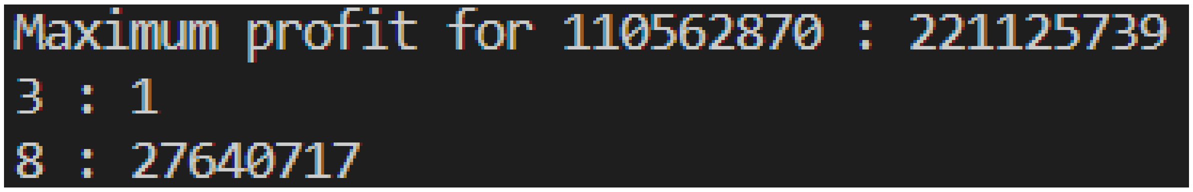
```
     for reconstruction*/
    for (long long i = 1; i <= W ; i++)
    {
      /* Loop starting from 1 to W and this loop evaluates the
         subproblem for a particular weight i*/
        for (int j = 0; j < n; j++)
        {
          /* Checking if the item's weight is less than the capacity */
            if (w[j] <= i)
            {
              //Recurrence:
              //Inductive Case.
                if(dp[i-w[j]] + v[j] > dp[i]){ // checking if the profit including
                    the present item is greater than the profit excluding that item
                  dp[i] = dp[i - w[j]] + v[j];
                  prev[i] = j;  // storing the value of the previous index taken
                }
            }
        }
    }
    cout << "Maximum profit for " << W <<" : " << dp[W] << endl;
    // Final answer for the maximum profit.
    reconstruct(v, w, prev, W);
}

int main()
{
  /* 1.) n is the size of the value(v) and the weight(w) array.
     2.) W is the total capacity that we go to.
     3.) v -> items values.
     4.) w -> weights of the items.*/
    int n = 10;
    int W = 11056;
    vector<int> v{2, 3, 5, 8, 13, 21, 34, 55, 89, 144};
    vector<int> w{2, 2, 3, 4, 7, 12, 20, 33, 54, 88};
    findUsingDP(v, w, W, n);
    /* call the findUsingDP function to calculate the maximum profit*/
    return 0;
}
```



```
Maximum profit for 11056 : 22112
8 : 2764
```

Maximum profit for the first 5 digits of the Student ID is 22112 **and it has taken the value** $8 \rightarrow 2764$ **times.**
Maximum profit for the complete 9 digit of the Student ID is 221125739 **and it**



```
Maximum profit for 110562870 : 221125739
3 : 1
8 : 27640717
```

has taken the value $3 \rightarrow 1$ **time and** $8 \rightarrow 27640717$ **times.**

0.2 Part b

Ensure your code can reconstruct the optimal set and report the optimal set for the inputs above.
Solution:
To reconstruct the object, we need to create another array prev[] where,

$$prev[j] = i, \text{ if } dp[i-j][j-1] + v[j] \geq dp[i][j-1]$$

setting this to the prev array we can get the maximum profit value and see in which index this value is present in the value array and increase the count of that value in the map and we can trace back and get all the lengths of the rods which we have stored in the map.

Problem 3 (6 points)

Following are flow instances with capacities in black. For each of the following functions f given in blue, why is it not a flow? Justify your answer briefly. Any labels not pictured are zero.

Solution:

For a given graph $G = (V, E)$ with a capacity function $c : E \rightarrow$ R $\geq 0, a flow function$f : V x V$\rightarrow$ should satisfy the following conditions:

$\rightarrow Skew-symmetric:\ \forall\ \{s,t\} \in$ V, $f(s,t) = -f(t,s)$.
$\rightarrow Net\ \ flow\ \ constraint : Each$ v $\notin\ \{s,t\},\ \sum_{u \in v} f(u,v) = 0$.
$\rightarrow Capacity\ \ constraint : \forall\ \{u,v\} \in$ V, $f(u,v) \leq c(u,v)$.

Part A

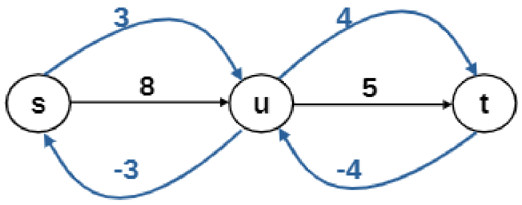


Figure 1: Graph

According to the net-flow constraint, whatever flows in must flow out, in the above graph the flow from $s \rightarrow u$ is 3, but the flow from $u \rightarrow t$ is 4 which defies the net-flow constraint, hence we can conclude that this is not a valid flow.

Part B

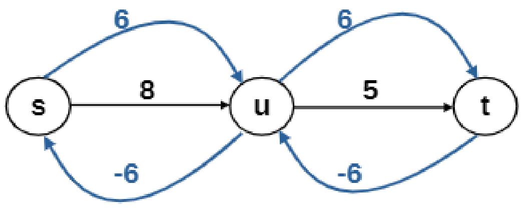


Figure 2: Graph

In the above graph, the capacity from $u \rightarrow t$ is 5, but the flow is 6 which defies the capacity constraint $f(u,t) \leq c(u,t)$, hence we can conclude that this is not a valid flow.

Part C

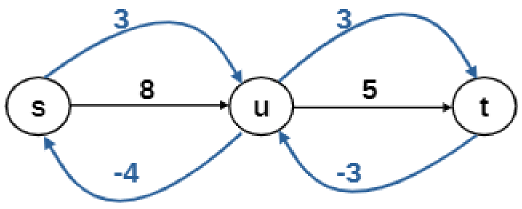


Figure 3: Graph

In the above graph the net flow for s is 3 + (-4) = -1, which defies the skew-symmetric constraint where the net flow must be 0 i.e $f(u,v) = -f(v,u)$ or $f(u,v) + f(v,u) = 0$, hence we can conclude that this is not a valid flow.