AOA Final

Kasiviswanathan Srikant Iyer 5222-2519

January 17, 2022

1 Question 1

Dynamic programming [35]

1.1 Psudocode

Algorithm 1 Knapsack with weight and time

Data: Get values of mac weight W, max time T, and the corresponding value, weight and time of each item as v[i],wt[i],ti[i].
Result: $i[]$ which gives the items the thief should pick for maximum profit
$n \leftarrow length.v[]$
$K[n+1][W+1][T+1] \leftarrow 0$

for `i= 0...n` **do**
  for `w= 0...W` **do**
    for `t= 0...T` **do**
      if *i or w or t is 0* **then**
        $K[i][w][t] \leftarrow 0$; ▷ Set first x, y, z axis to 0
      else
        if $wt[i-1] <= w$ *And* $ti[i-1] <= t$ **then**
          $K[i][w][t] = Max(v[i-1] + K[i-1][w - wt[i-1]][t - ti[i-1]], K[i-1][w][t])$;
        else
          $K[i][w][t] = K[i-1][w][t]$;
        end
      end
    end
  end
end
BestValue= $K[n][W][T]$;
while $i \geq 1$ *and* $BestValue \geq 1$ **do**
  if $BestValue$ *is* $K[i-1][w][t]$ **then**
    Continue; ▷ *because that value has not been used*
  else
    $i[x] = i$; ▷ *Gets the optimaly selected values*
    $x \leftarrow x + 1$;
    $BestValue \leftarrow BestValue - v[i-1]$;
    $w \leftarrow w - wt[i-1]$;
    $t \leftarrow t - ti[i-1]$;
  end
end
return i;

1.2 Proof of termination

The Loop is designed to run till the Max value of Weight and Max value of Time allotted for the robber. Once the 3D matrix has traversed all the points items with its weight and time combinations, it will terminate along with the 3 for loops.

1.3 Proof by Induction

Hypothesis: Compute all values in the 3D matrix in the increasing order as per the loops. The value of the current element is based on the previous elements and it is assumed the previous values are correct.

Base case for the induction is K[0][w][t]=K[i][0][t]=K[i][w][0]=0 which is true by base case of algorithm where all values in the K matrix are set to 0.

If our inductive hypothesis is true, while computing K[i][w][t] if the weight and time are within the allotted time and weight, the item is stolen, else it is left for another object. Since we take the greater of K[i-1][w][t] and K[i-1][w-1][t-1] which are based on out inductive hypothesis, out algorithm gets the most efficient value for K[i][w][t].

1.4 Complexity

The complexity of the above algorithm will be equal to the number of values it needs to iterate in the matrix to compute the best value. So as there are n values and W is the max weight and T is the max time, and we iterate for every value if weight below W and every value of time below T, the complexity of this algorithm will be O(nWT).

2 Question 2

Network flow [35]

2.1 Psudocode

Algorithm 2 Modified Ford–Fulkerson

Data: Undirected graph $G(V, E)$, vertices S and T
Result: Least number of edges to separate S and T
for *edge*(u, v) *in* E **do**
  $(u, v).flow = 0$; ▷ Setting flow to 0
end
DFS (S); ▷ Run Depth First search to get to T
  while *Path p exsits in* $G : S \to T$ **do**
    if $(u,v) \in E$ **then**
      (u, v).*flow*= (u, v).*flow*+1; ▷ *Increase the number of min connections*
    else
      (v, u).*flow*= (v, u).*flow* - 1; ▷ *Decrease the number of min connections*
    end
  end

2.2 Proof by Contradiction

Claim: Ford Fulkerson's algorithm gives us the maximum flow through the network. This maximum flow is equal to the capacity of the minimum cut.

Assuming the contradiction : The algorithm does not give us the maximum flow through the network.

Proof: It is claimed that the algorithm used to solve the problem specification gives us maximum flow of the network. Assuming the contradiction: The flow generated by the algorithm used does not give the maximum flow of the network. The maximum flow of the network is, however, generated through a network flow path that the algorithm above has analysed. If the maximum flow algorithm has already analyzed the path, it is a contradiction to our claim. Therefore, the Ford Fulkerson Maximum Flow - Minimum Cut algorithm has yielded the maximum flow required.

2.3 Complexity

- Depth first search or breadth first search is used to find the various augmenting paths in the network from vertices S to T.
- The value of flow increases by one at each iteration, as the edges of the graph are given a weight of one. Thus, the algorithm depends on the maximum flow found.
- The algorithm has a running time of $O(E|flow*|)$.
- $flow*$ is the maximum flow found and E is the number of edges in the graph.

3 Question 3

Complexity [30]

3.1 Solution

To prove that the solution to the above question is in NP Complete, we need to satisfy the below 4 points.

1. Show a non deterministic polynomial time algorithm that solves our program; that is our program $\in NP$.
2. Any known NP-Complete problem can be reduced to our problem
3. The reduction works in polynomial time
4. Our problem has a solution only if the NP Complete problem has an solution.

3.2 Proof

1. SetPartition $\in NP$ Randomly guess 2 partition and check if they have equal sums.
2. Reduction of SubSet-Sum to Set- Partition: In the SubSet-Sum we are given a set of values S and a target x. Our goal is to find a SubSet from S such that the sum of its digits gives us x. So to make this into a Set Partition problem, we need to get the sum of the set S and feed the value S $\cup$ Sum(S)-2x.
3. The above task clearly works in linear time as the only additional step is to add up the set.
4. If SetPartition (S $\cup$ Sum(S)-2x) is true, then SubSetSum(S,X) is also true as by design the set partition was done for (2(Sum(S)) - 2x).

3.3 Conclusion

1. For any set of numbers in the set S that add to make x, the remaining numbers in X sum to S-x. So if there exists a partition of (2Sum(S)-2x) into such that each partition sums to S-x.
2. For such a value whee the sum of each partition is S-x, one of the sets contains S-2x. By removing the set of numbers whose sum is x. All these belong in set S.