Assignment 2 **Wen Han Xu**
CS711008Z ID: 201818018670073
Date: 2018/10/30

1 Money Robbing

A robber is planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

1. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.
2. What if all houses are arranged in a circle?

1.1 Algorithm Description

Let w_i denote the amount of money stashed in the i-th house on the street. As the alarm will be triggered when two adjacent houses were broken into on the same night, this problem can be converted into finding the discontinued sub-sequence with the largest weight.

1.1.1 Optimal Substructure

Let $d[i]$ denote the maximum amount of money the robber have stolen until he reached the i-th house on the street. When deciding whether to break into the i-th house, the robber look back on $d[i-1]$ and $d[i-2]$. If the robber decides to break into the i-th house, then $d[i]$ cannot be calculated based on $d[i-1]$, in this circumstance, $d[i] = d[i-2] + w_i$. If the robber decides not to break in, then $d[i] = d[i-1]$. Thus, the DP equation of this problem is as below.

$$d(i) = \begin{cases} max\{d(i-2) + w_i,\ d(i-1)\} & 3 \leq i \leq n \\ max\{w_1,\ w_2\} & i = 2 \\ w_1 & i = 1 \end{cases}$$

1.1.2 Pseudo-Code

Algorithm 1 Money Robbing

```
1: procedure RobinHood(w_1, w_2, ..., w_n)
2:     for i ← 1 to n do
3:         d[i] = 0
4:     d[1] = w_1
5:     d[2] = max(w_1, w_2)
6:     for i ← 3 to n do
7:         d[i] = max{d[i − 2] + w_i, d[i − 1]}
```

The final answer to this problem can be inspected by picking $max(d[n-1],\ d[n])$.

1.2 Proof of Correctness

We use the "cut-and-paste" proof.

Suppose for $w_1, w_2, ..., w_k$, there is another optimal solution $d'[k]$, then based on $d'[k]$ we could construct a better solution to $d'[k+1]$ and $d'[k+2]$, which would probably leads to a better final solution $d'[n]$, resulting in a contradiction. Thus, the optimal sub-structure of this problem is correct.

1.3 Time Complexity

Performing Algorithm 1 requires looping from 1 to n, in each loop a constant time operation is performed, thus the overall time complexity of Algorithm 1 is $O(n)$.

1.4 All Houses In A Circle

To solve the problem when all houses are arranged in a circle, we need to consider the decision of the final sub-problem. At the n-th house, the decision of whether to pick it or not depends on the decision of $n-1$th and 1st house. In the case when the last house is picked, the 1st house can not be picked, and we can consider this a new problem where the array(street) starts from the 2nd house. We use 2 DP arrays to help solve this problem. $d_1(k)$ is the optimal solution that starts from the 1st element but ends at the $n-1$th element; $d_2(k)$ is the optimal solution that starts from the 2nd element but ends at the nth element.

$$d_1(i) = \begin{cases} max\{d_1(i-2) + w_i,\ d_1(i-1)\} & 3 \le i \le n-1 \\ max\{w_1,\ w_2\} & i = 2 \\ w_1 & i = 1 \end{cases}$$

$$d_2(i) = \begin{cases} max\{d(i-2) + w_i,\ d(i-1)\} & 3 \le i \le n \\ w_2 & i = 2 \\ 0 & i = 1 \end{cases}$$

The final answer is $max\{d_1(n-1),\ d_2(n)\}$.

2 Node selection

You are given a binary tree, and each node in the tree has a positive integer weight. If you select a node, then its children and parent nodes cannot be selected. Your task is to find a set of nodes, which has the maximum sum of weight.

2.1 Algorithm Description

Assume there are n nodes in the given binary tree, each node is marked in a top-bottom and left-right order. For instance, the root node is marked 1, and its 2 children nodes are marked 2&3. Let w_i denote the weight of node i.

2.1.1 Optimal Substructure

Let $d(k)$ denote the maximum subset weight at $node-k$, as no adjacent node can be picked into the subset, $d(k)$ can either be calculated from its children $d(2k)$ and $d(2k+1)$, or from its grandchildren nodes $d(4k), d(4k+1), d(4k+2)$ and $d(4k+3)$.

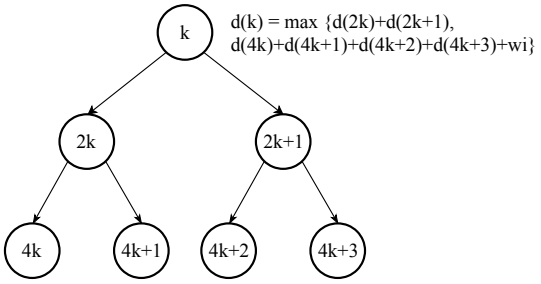

Figure 1: Optimal Substructure

The DP equation of this problem is as below. (For convenience if the index exceeds the total number of nodes in the binary tree, d(k) returns zero, e.g. d(n+1)=0)

$$d(i) = max\{d(2i) + d(2i+1),\ d(4i) + d(4i+1) + d(4i+2) + d(4i+3) + w_i\}$$

2.1.2 Pseudo-Code

Algorithm 2 NodeSelection

procedure NODESELECTION($w_1, w_2, ..., w_n$)
  for $i \leftarrow 1$ *to* n **do**
    $d[i] = 0$
  for $i \leftarrow 2^{\lfloor \log n \rfloor}$ *to* n **do**
    $d[i] = w_i$
  for $k \leftarrow \lfloor \log n \rfloor - 1$ *to* 1 **do**
    for $i \leftarrow 2^{\lfloor k \rfloor}$ *to* $2^{\lfloor k+1 \rfloor}$ **do**
      $d(i) = max\{d(2i) + d(2i+1),\ d(4i) + d(4i+1) + d(4i+2) + d(4i+3) + w_i\}$

After execution of Algorithm 2 the final answer to this problem is in $d(1)$.

2.2 Proof of Correctness

We use the "cut-and-paste" proof to proof the correctness of the optimal substructure.

Suppose for node $4k$ and $d(4k)$, there is a better optimal solution named $d'(4k)$. Then based on $d'(4k)$, we could construct a better solution $d'(2k)$ and $d'(k)$ for node $2k$ and k, and finally leading to a better solution to the final problem, which is contradictory to its definition. Thus, the optimal substructure is correct.

2.3 Time Complexity

Algorithm 2 consists of 3 for-loops. The first 2 loops take $O(n)$ time. The last loop iterates over n nodes in the binary tree, which also takes $O(n)$ time. Thus, the overall time complexity of Algorithm 2 is $O(n)$.

3 Decoding

A message containing letters from A-Z is being encoded to numbers using the following mapping: Given an encoded message containing digits, determine the total number of ways to decode it.

For example, given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12). The number of ways decoding 2" is 2.

3.1 Algorithm Description

We present the encoded message (length n) as a sequence $(a_1, a_2, a_3, ..., a_n)$.

3.1.1 Optimal Substructure

Let $d[i, j]$ be the number of decoding of the sub-sequence $(a_i, a_i + 1, ...a_j)$. It's easy to notice that its 2 sub-problems $(a_i, ...a_k)$ and $(a_k, ..., a_j)$ (for every k $i \leq k \leq j$)) are independent. Thus we can divide the original problem of range $[i, j]$ to sub-problems with smaller size. When merging the sub-problems, we need to find the specific k that maximizes $d[i, k] + d[k, j]$.

The DP equation of this problem is as below.

$$d(i,j) = \begin{cases} \max\limits_{i \leq k \leq j} \{d[i,k] + d[k,j]\} & \\ 1 & i = j-1, a_i a_{i+1} \leq 26 \\ 0 & i = j-1, a_i a_{i+1} > 26 \\ 0 & i = j \end{cases}$$

The final answer to this problem is $d(1, n) + 1$.

3.1.2 Pseudo-Code

Algorithm 3 Decoding

```
1: procedure DECODING(a_1, a_2, a_3, ..., a_n)
2:     Initialize d[i, j] with 0
3:     for i ← 1 to n − 1 do
4:         if a_i a_{i+1} ≤ 26 then
5:             d[i, i + 1] = 1
6:     for i ← 1 to n do
7:         for k ← 2 to n − i do
8:             d[i, i + k] = max_{i≤k≤j} {d[i, k] + d[k, j]}
```

3.2 Proof of Correctness

As the 2 sub-problems $(a_i, ...a_k)$ and $(a_k, ..., a_j)$ (for every k $i \leq k \leq j$)) are independent, we can divide $d[i, j]$ into smaller problems. For $d[i, j]$, if there exists a better optimal value $d'[i, j]$, then we may construct a better optimal solution to the final problem. Thus, Algorithm 3 is correct.

3.3 Time Complexity

The time complexity of Algorithm 3 is $O(n^2)$.