COMP2007 Assignment 3 Report

Jiashu Wu 460108049 jiwu0083

Question 1

1.1 Dynamic Programming Algorithm

Sub-problem

OPT(i) = Optimal solution ending at i^{th} house.

Recurrence Relation

OPT(i) equals to the **maximum of the following two cases**

- Value(i) + OPT (i - 2) (OPT selects house i)
- OPT(i - 1) (OPT does not select house i)

Base Case

- OPT(0) = Value(0)
- OPT(1) = max{Value(0) , Value(1)}

Pseudo-code of this bottom-up dynamic programming algorithm

```
// Base Case
OPT[0] = Value[0]
OPT[1] = max(Value[0] , Value[1])
For i in 2 to n-1
    OPT[i] = max(Value[i] + OPT[i-2] , OPT[i-1])
// OPT[n-1] is the maximum value of the subset of houses,
// where no two chosen houses can be adjacent
```

1.2 Argue the correctness

I will prove the correctness of my algorithm using Prove by Induction.

Inductive Claim

The algorithm can correctly calculate OPT(i) for all i.

Base Case

For OPT(0), there is only one house. Since for all houses it has positive value, therefore the optimal solution is selecting this house, which is Value(0).

For OPT(1), there are only two houses, and we can't choose two houses at the same time since they are adjacent with each other, thus we can only choose one house from these two houses. Hence the optimal solution is to select the house with bigger value, which is max{Value(0) , Value(1)}.

Inductive Hypothesis

We assume our inductive claim holds for all the value i between 0 and j. (OPT[0] to OPT[j] are all correctly calculated)

Inductive Step

For i = j+1:

At this step of the algorithm, the optimal solution OPT(j+1) is the optimal choice of the following two cases:

Case 1: House j+1 is selected, therefore we can't choose the neighbour house j, hence the optimal solution OPT(j+1) is Value(j+1) plus OPT(j-1), the optimal choice ending at j-1th house. By the inductive hypothesis, OPT[j-1] was correctly calculated, hence the result of case 1 is correct.

Note: Prove Case 1 is correct using Prove by Contradiction: Let's suppose that Value(j+1) + OPT(j-1) is not the optimal solution in this case, so there must exist Value(j+1) + OPT'(j-1) which is optimal. But this contradicts with OPT(j-1) is optimal, therefore, Value(j+1) + OPT(j-1) is the correct optimal solution in this case.

Case 2: House j+1 is not selected, therefore we can still consider the neighbour house j, hence the optimal solution OPT(j+1) is OPT(j), the optimal choice ending at jth house. By the inductive hypothesis, OPT[j] was correctly calculated, hence the result of case 2 is correct.

Note: Prove Case 2 is correct using Prove by Contradiction: Let's suppose that OPT(j) is not the optimal solution in this case, so there must exist OPT'(j) which is optimal. But this contradicts with OPT(j) is optimal, therefore, OPT(j) is the correct optimal solution in this case.

Since at this step the algorithm pick the case with the maximum value, therefore, the solution is still optimal. Hence the algorithm can still correctly calculate OPT[j+1].

Corollary to Inductive Claim

The algorithm can correctly calculate OPT(i) for all i.

1.3 Prove the upper bound on the time complexity

Upper Bound Time Complexity

The upper bound time complexity of this algorithm is O(n), where n is the number of houses.

Proof

Since the algorithm used the memorization technique to store results of each sub-problem into an array, thus we don't need to calculate the optimal solution of each sub-problem multiple times, instead we only need to do it once, and looking up the result requires O(1) work. Since we need to calculate the optimal solution for n houses and each calculation takes O(1), hence the whole calculation takes O(n).

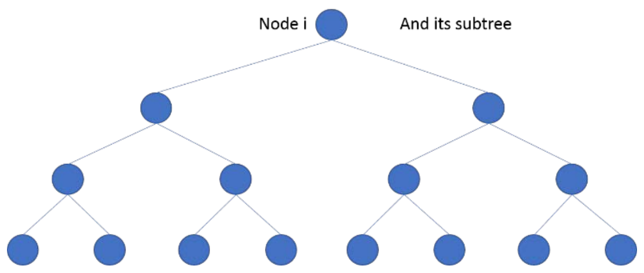
Hence, the whole algorithm requires O(n) work.

Question 2

2.1 Dynamic programming Algorithm

Sub-Problem

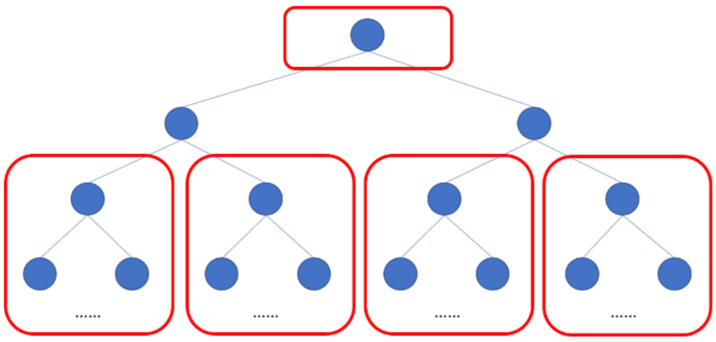
OPT(i) = Optimal solution of i^{th} node and its subtree.



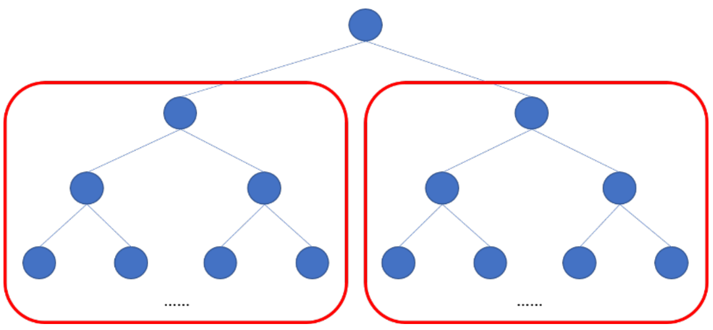
Recurrence Relation

OPT(i) equals to the **maximum of the following two cases**

- If node i is selected: OPT[i] = Value(i) + OPT[Left child of Left child of node i] + OPT[Right child of Left child of node i] + OPT[Left child of Right child of node i] + OPT[Right child of Right child of node i]

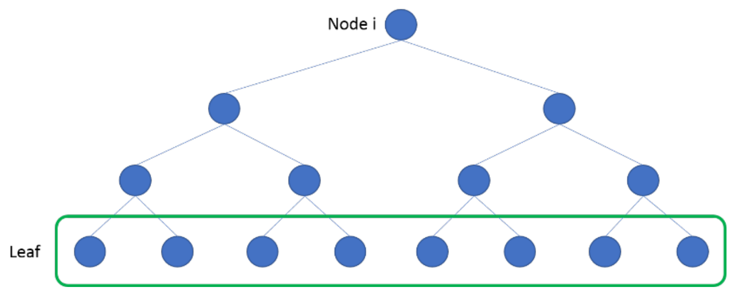


- If node i is not selected: OPT[i] = OPT[Left child of node i] + OPT[Right child of node i]



Base Case

- OPT[i] = Value(i) where node i is a leaf
- OPT[i] = 0 where i >=length(array)



2.2 Argue the correctness

I will prove the correctness of my algorithm using Prove by Induction.

Inductive Claim

The algorithm can correctly calculate the optimal solution of the subtree of any size less than or equal to j for all value of j.

Base Case

When node i is a leaf, its subtree has size 1 since the subtree only contains node i itself, thus OPT(i) = Value(i) since the only node which can be selected is the node i itself, and for all nodes it has positive value, therefore the optimal solution for every leaf node is its own value.

When i>= length(array), OPT(i) = 0. Since there is no node in the subtree, therefore there is no node can be selected, thus the optimal solution is 0.

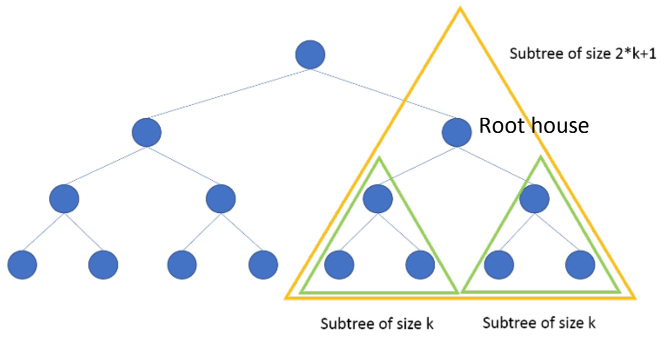
Inductive Hypothesis

We assume our inductive claim holds for all the subtrees which has size less than or equal to k.

Inductive Step

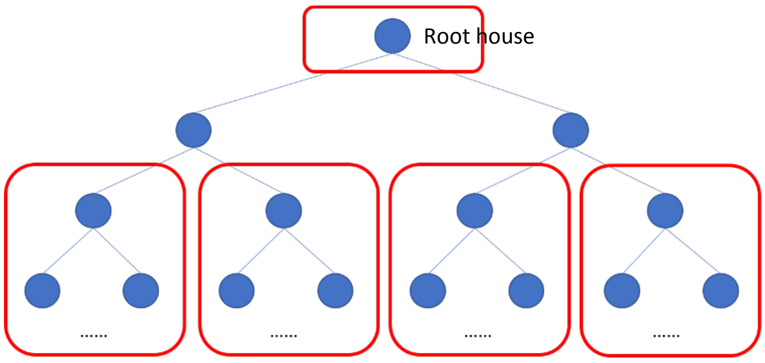
For subtree of size 2*k + 1:

We have two cases:



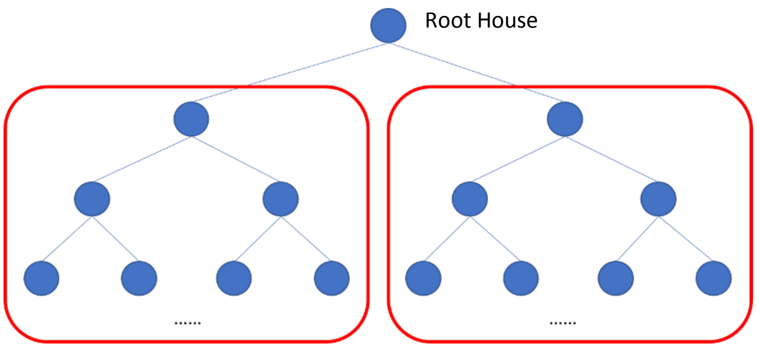
Case 1: The root house is selected, therefore we can’t choose both left and right child house, but we can choose the children of both left and right child, therefore the optimal solution

OPT(root) is the value of root of this subtree plus the optimal solution of L-L child, L-R child, R-L child and R-R child. Since for these four grandchildren, their subtree has size less than k, thus by the inductive hypothesis, their optimal value is correct. Therefore, the result of case 1 is correct.



Note: Prove Case 1 is correct using Prove by Contradiction: Let's suppose that OPT(root) is not the optimal solution in this case, so there must exist at least one OPT'(one of the grandchild) which is optimal. But this contradicts with OPT(one of the grandchild) is optimal, therefore, OPT(root) is the correct optimal solution in this case.

Case 2: The root house is not selected, therefore we can still consider the left and right child house, hence the optimal solution OPT(root) is OPT(left child) + OPT(right child). Since for these two children, their subtree has size less than k, thus by inductive hypothesis, their optimal value is correct. Therefore, the result of case 2 is correct.



Note: Prove Case 2 is correct using Prove by Contradiction: Let's suppose that OPT(root) is not the optimal solution in this case, so there must exist at least one OPT'(one of the child) which is optimal. But this contradicts with OPT(one of the child) is optimal, therefore, OPT(root) is the correct optimal solution in this case.

Since at this step, the algorithm pick the case with the maximum value, therefore the solution is still optimal. Hence the algorithm can still correctly calculate the optimal solution of subtree with size 2*k+1.

<u>Corollary to Inductive Claim</u>

The algorithm can correctly calculate the optimal solution for subtree of any size less than or equal to j for all value of j.

2.3 Prove the upper bound on the time complexity

Upper Bound Time Complexity

The upper bound time complexity is O(n), where n is number of house value.

Proof

Loading the value n takes O(1) time.

Loading all the house values and store them into an array takes O(n) since we have n house values and each storing operation takes O(1) time.

Since the algorithm used the memorization technique to store results of each sub-problem into an array, thus we don’t need to calculate the optimal solution of each sub-problem multiple times, instead we only need to do it once, and looking up the result requires O(1) work. Since we need to calculate the optimal solution for n houses and each calculation takes O(1), hence the whole calculation takes O(n).

Hence, the whole algorithm requires O(1) + O(n) + O(n) = O(n) work.