Lecture 13

Fibonacci- recursive implementation

In this implementation, there is a lot of redundant computation.

So we use memoization:

* record a value the first time it is computed
* and look it up when needed

Here we are relying on Table lookup of which memoization is a special case.

**OPTIMAL SUBSTRUCTURE:**

You can get globally optimal solution from locally optimal solutions of the sub problems.

**FOR THE 0-1 KNAPSACK PROBLEMS:**

A – collection of objects, for each object in A we have a value

We want to find the subset of A which has the maximum value subject to the weight constraint

if size of A is n then the number of possible subsets is 2n

Is there an optimal substructure? – using the decision tree(important):

Weights are = [5,3,2]

Values are = [9,7,8]

Maximum weight is 5.

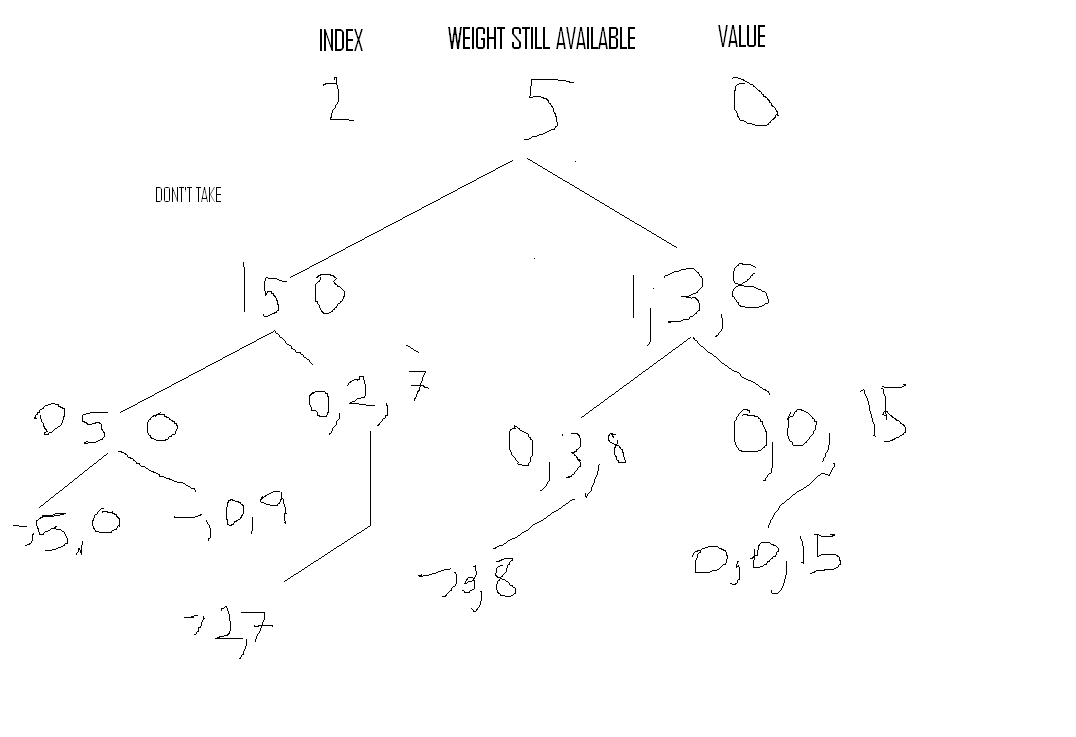
We will start at the end of the list.

To be systematic, always build the tree depth first left first.

After we go to the final node on the left beyond which we can’t go, we will BACKTRACK – back to tree where we have already been

In case while including an item the weight constraint gets violated then that branch will not exist.

As in example, tree 3 does not have a right branch. as the right branch, taking the item 0, is not feasible due to the weight constraint.



Once the tree has been made, then the winner can be selected.

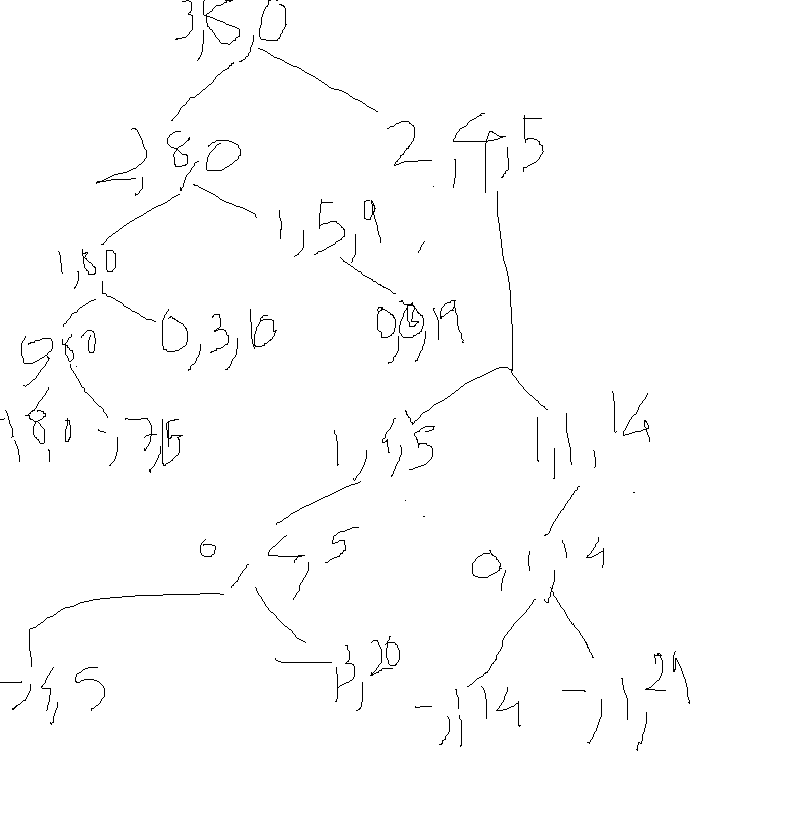
this solution is also exponential.

weights = 1,5,3,4

values = 15,10,9,5

maximum weight is: 8 kg

tree is:

The max is 29.

The maximum value is 29.

For a vector with 30 elements, it will take approximately 17 million calls. 17000000.

With the print statement turned on it is obvious that like Fibonacci it is doing lot of the same work again.

Solution (It is the same!):

similar to Fibonacci we save the value that we try to return the value In the memo if it is there(Each index and available weight pair has a unique solution, so keys to the memo are ordered pairs of index and available weight.) if it is not there then we simply generate it as in MaxVal and then store it away in the memo and return it so that next time there is no need to compute it again.

With 8 elements, 50 calls instead of 85 calls.

with 30 elements, 1885 calls instead of 17 million calls.