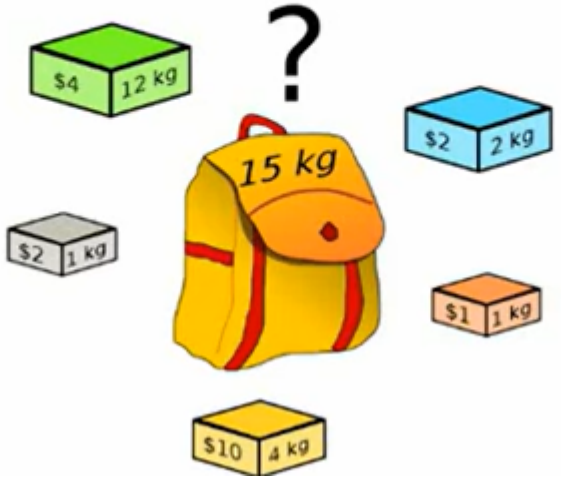
Solving 0/1 knapsack problem using Backtracking

0/1 knapsack problem

- A knapsack (kind of shoulder bag) with limited weight capacity.
- Few items each having some **weight and value**.
- Which items should be placed into the knapsack such that-
  - The **value or profit** obtained by putting the items into the **knapsack is maximum**.
  - And the **weight limit of the knapsack does not exceed**.



Note: An Item can be added any number of times.

(x,y) implies (weight, profit) eg (0,0)

Knapsack Weight: 6

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

0/1 knapsack problem

Weight, profit | 0,0

Knapsack

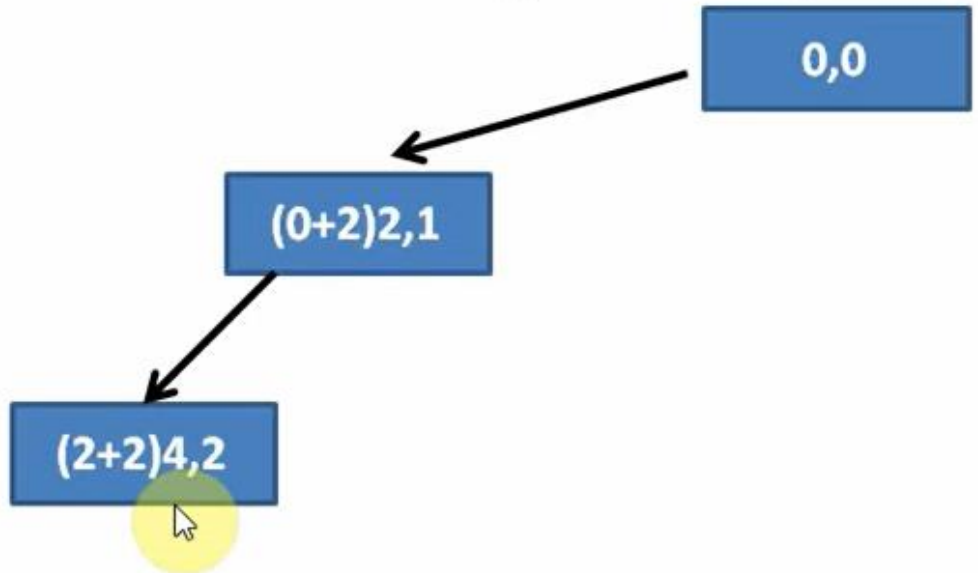
Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

0/1 knapsack problem

Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

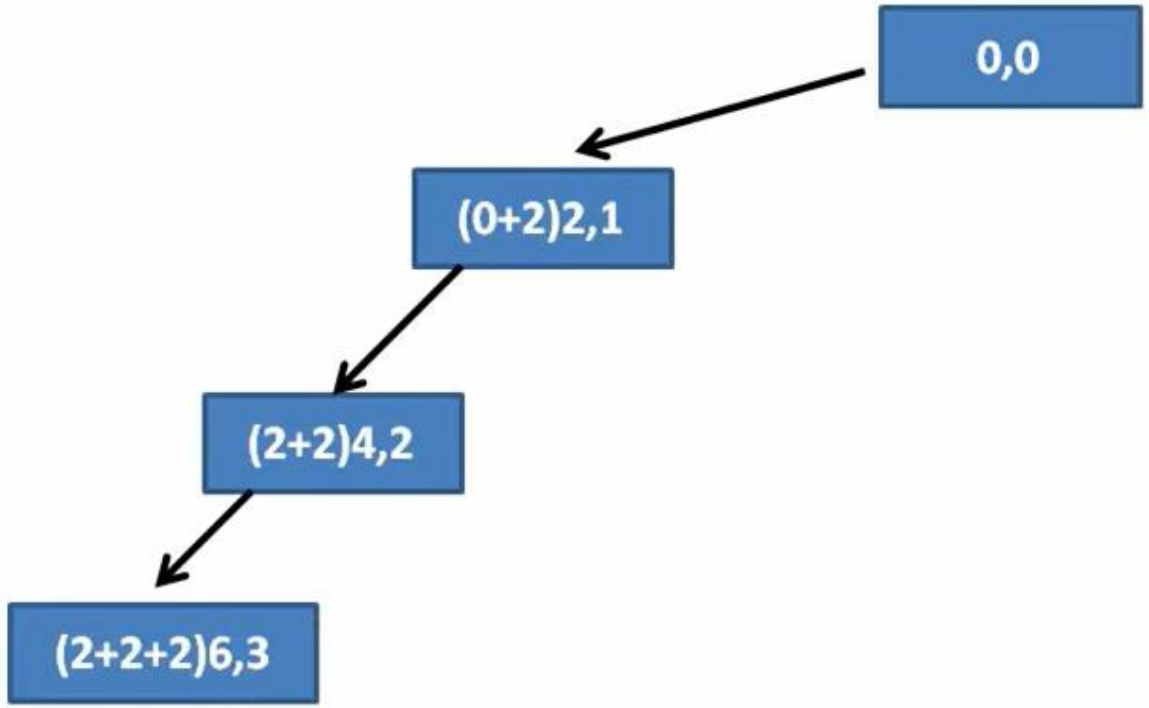
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

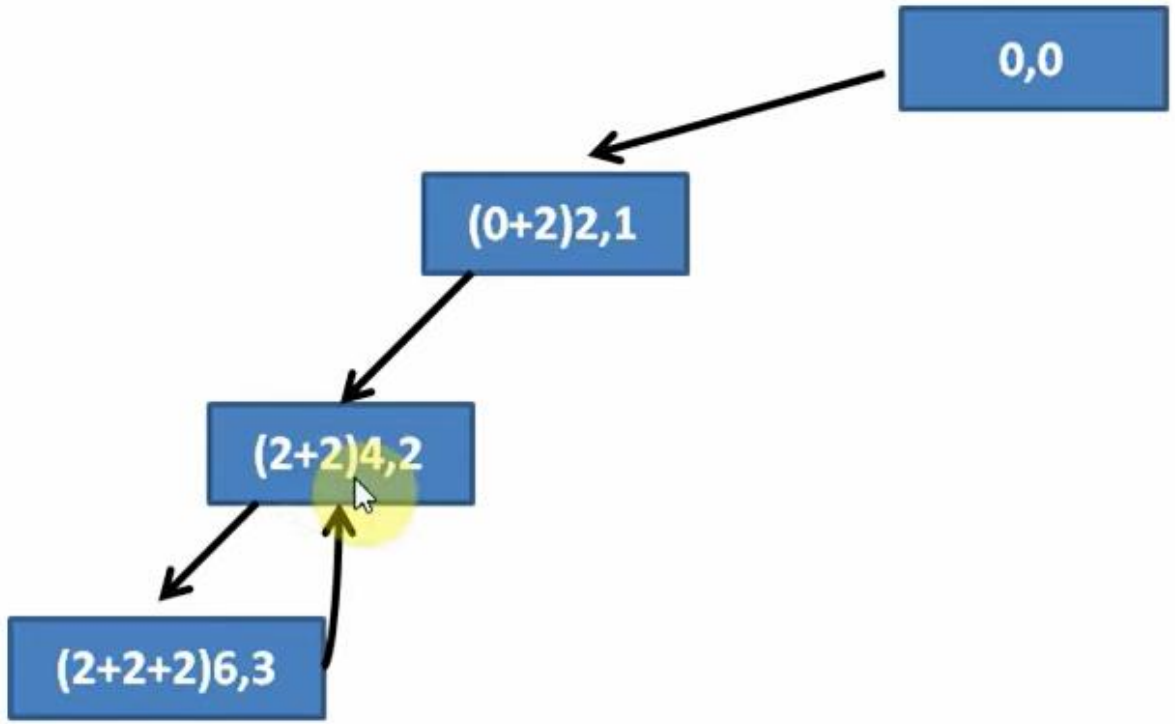
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

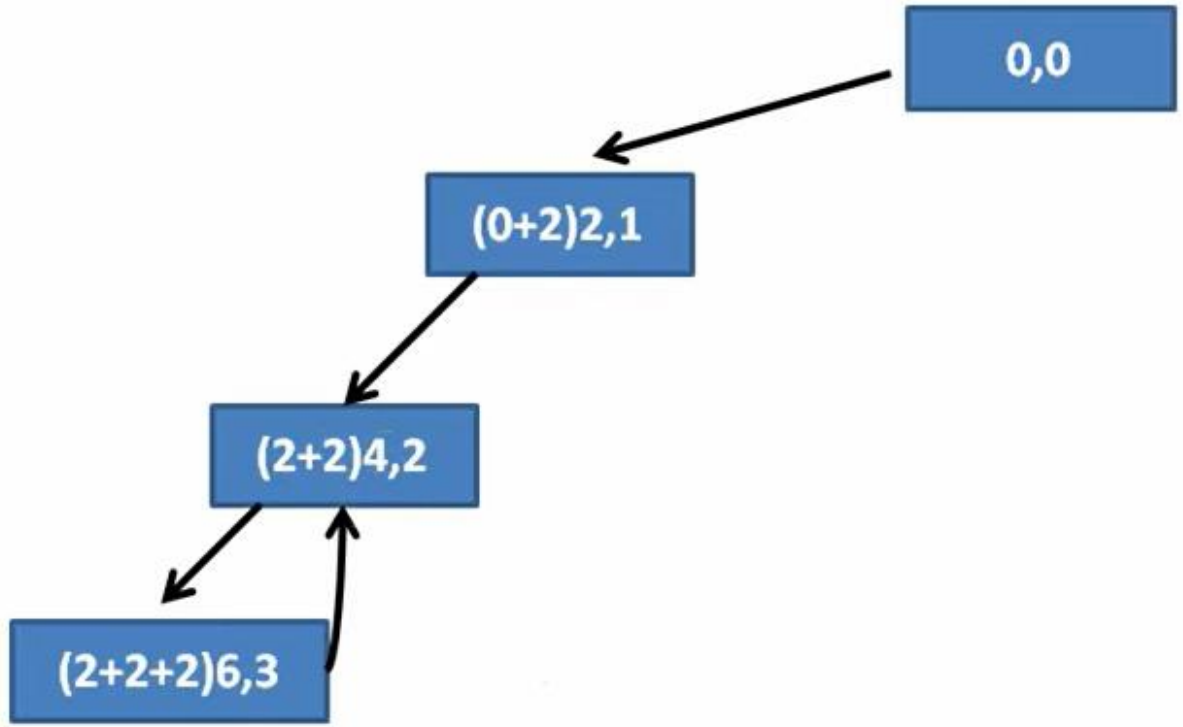
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

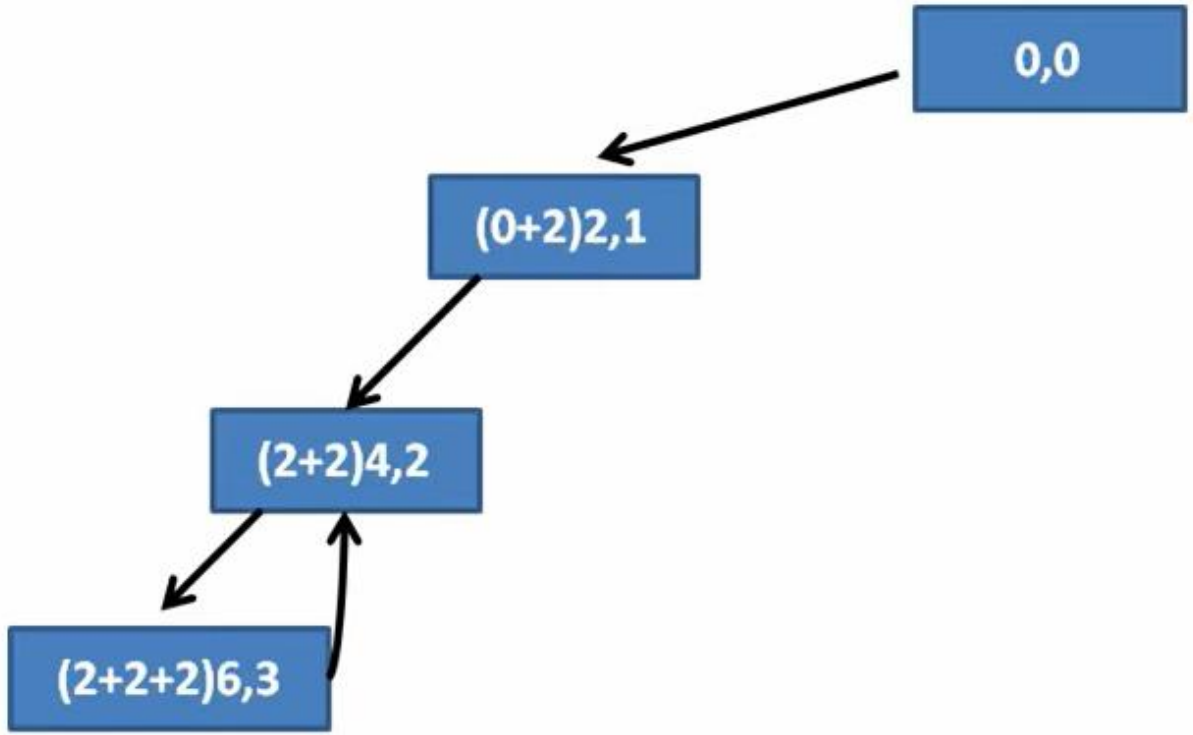
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

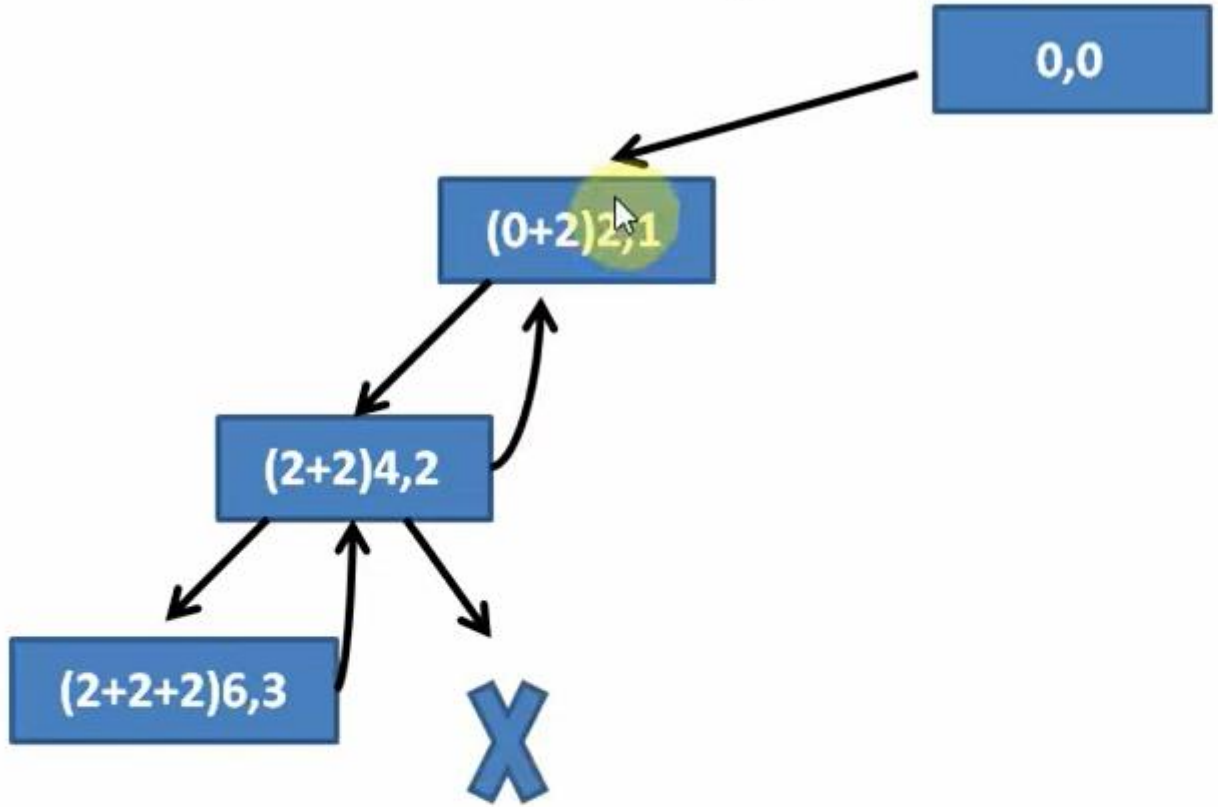
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

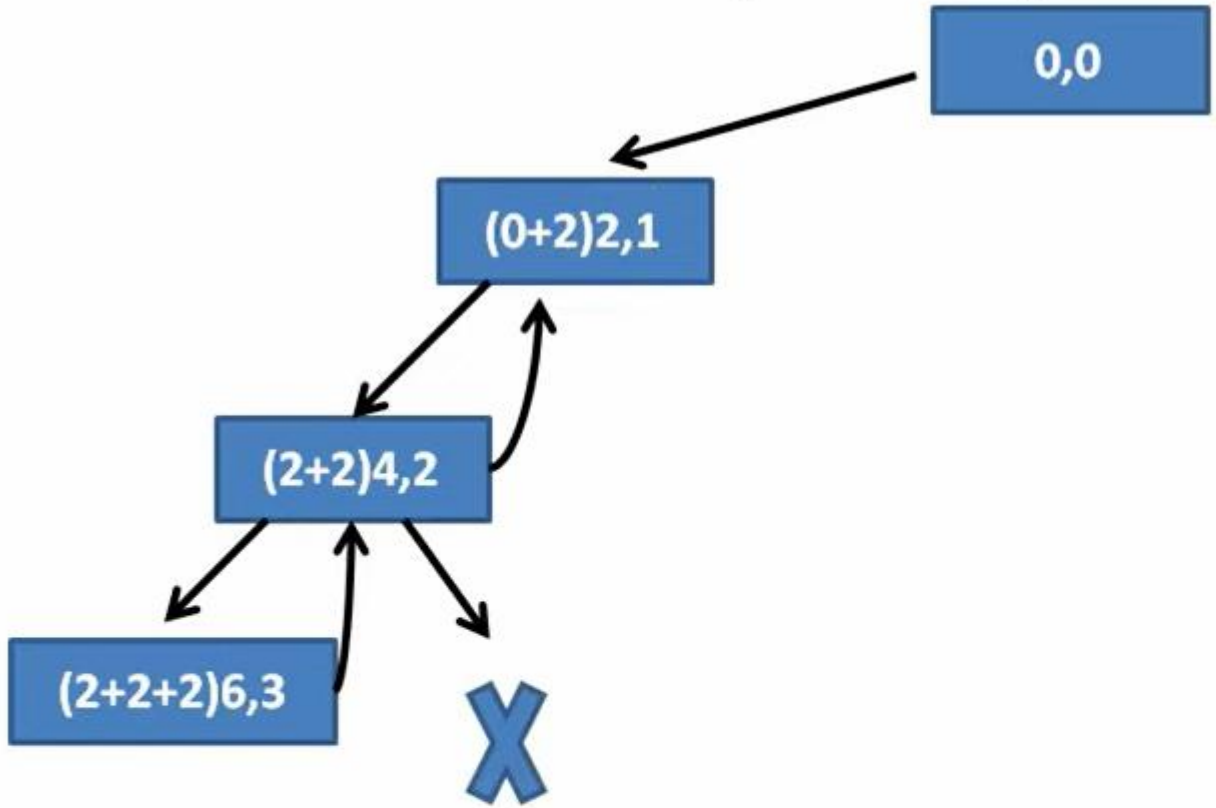
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

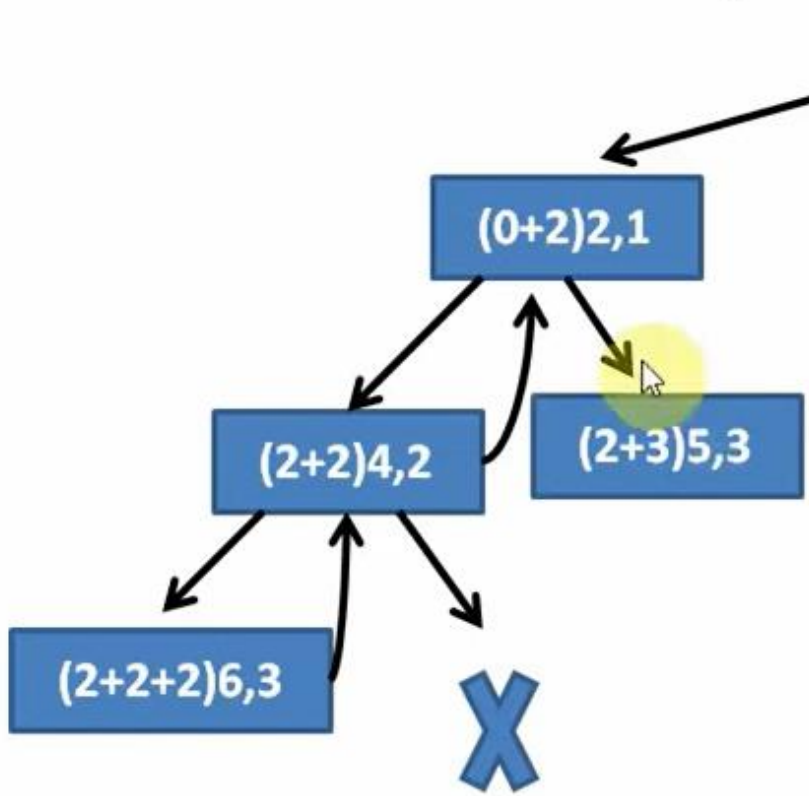
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

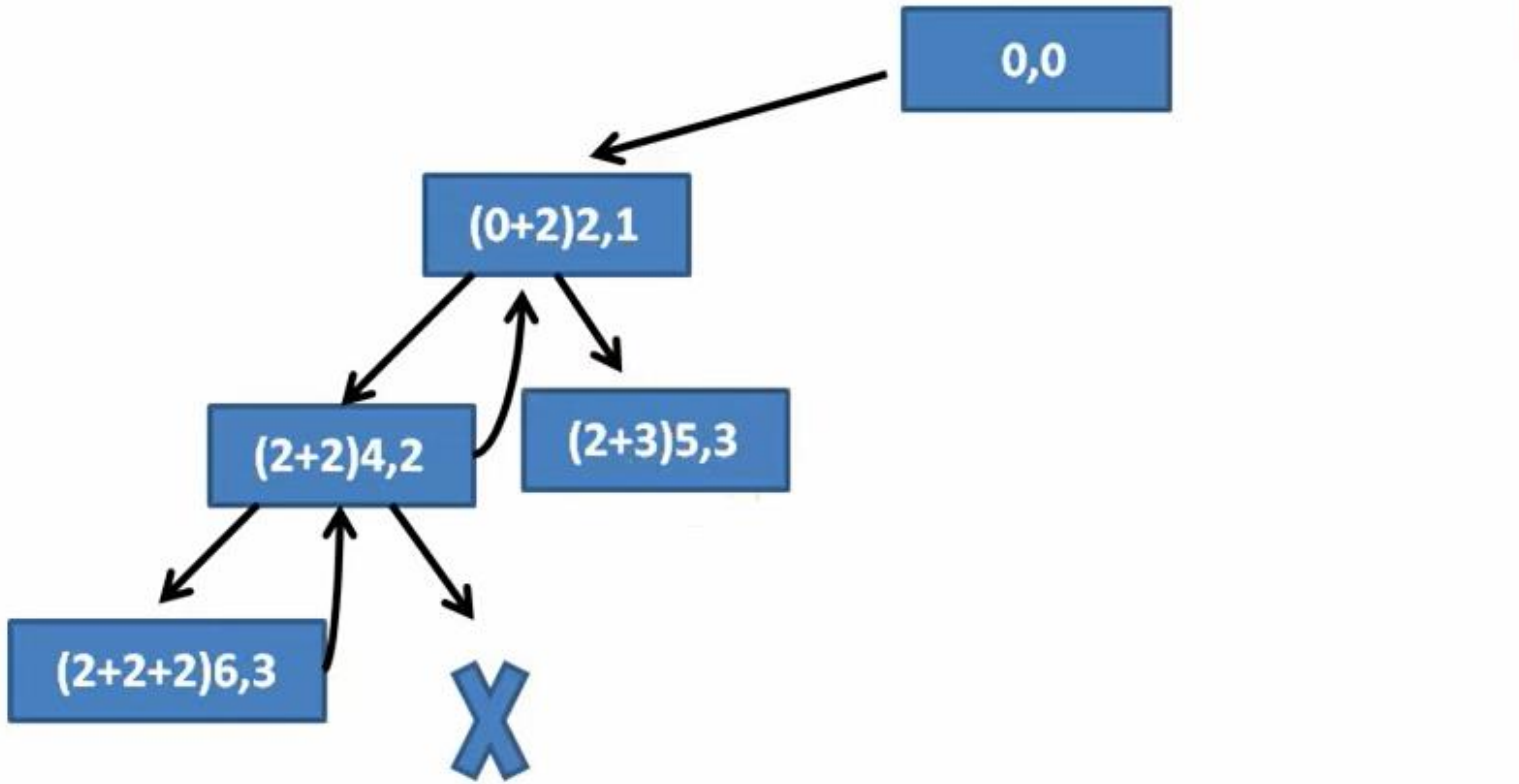
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

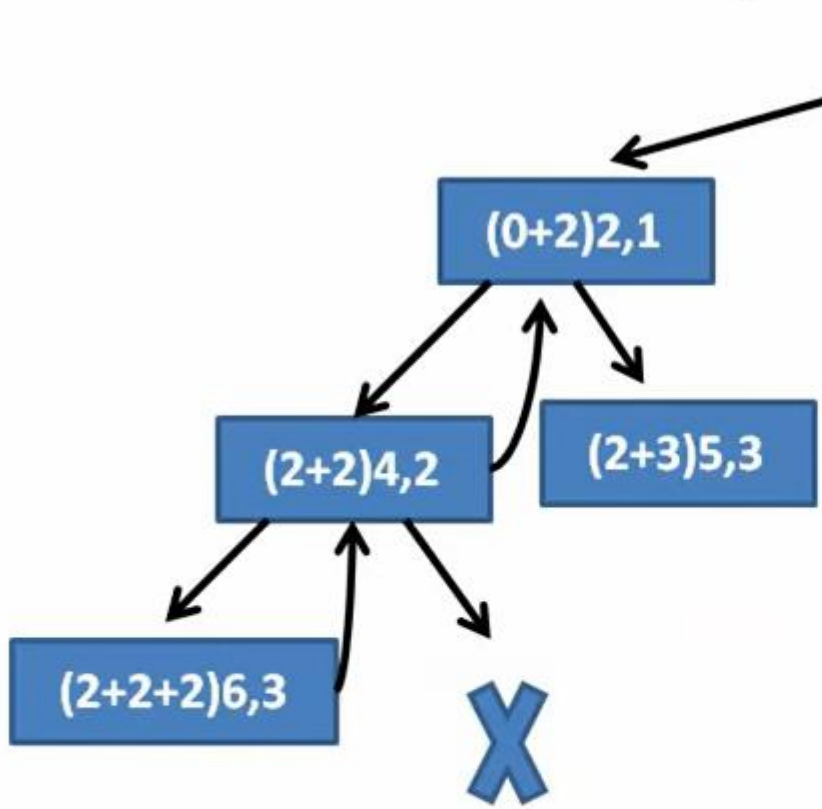
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

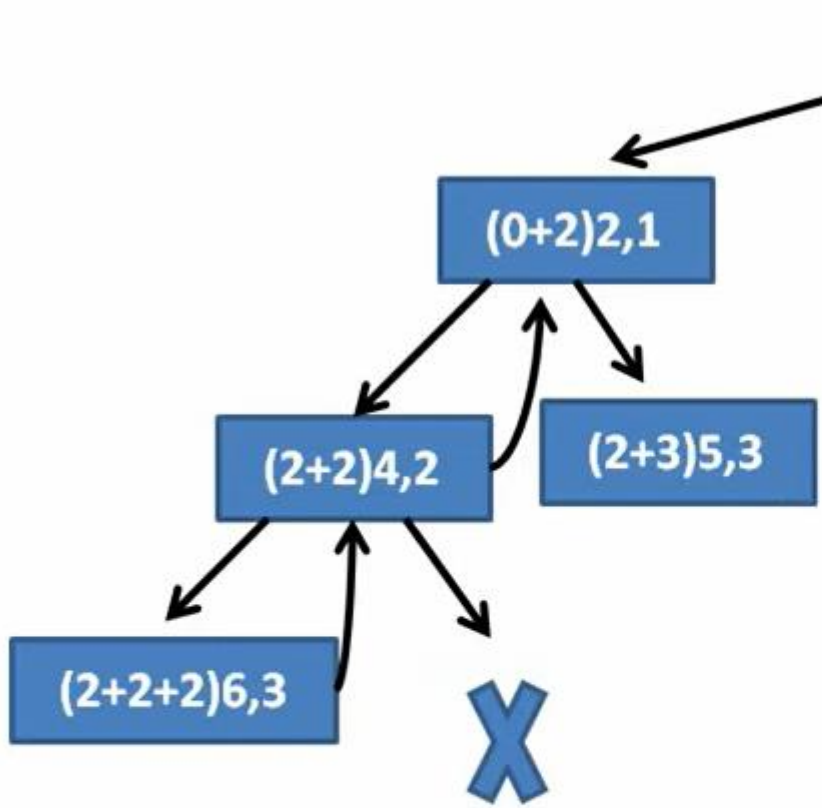
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

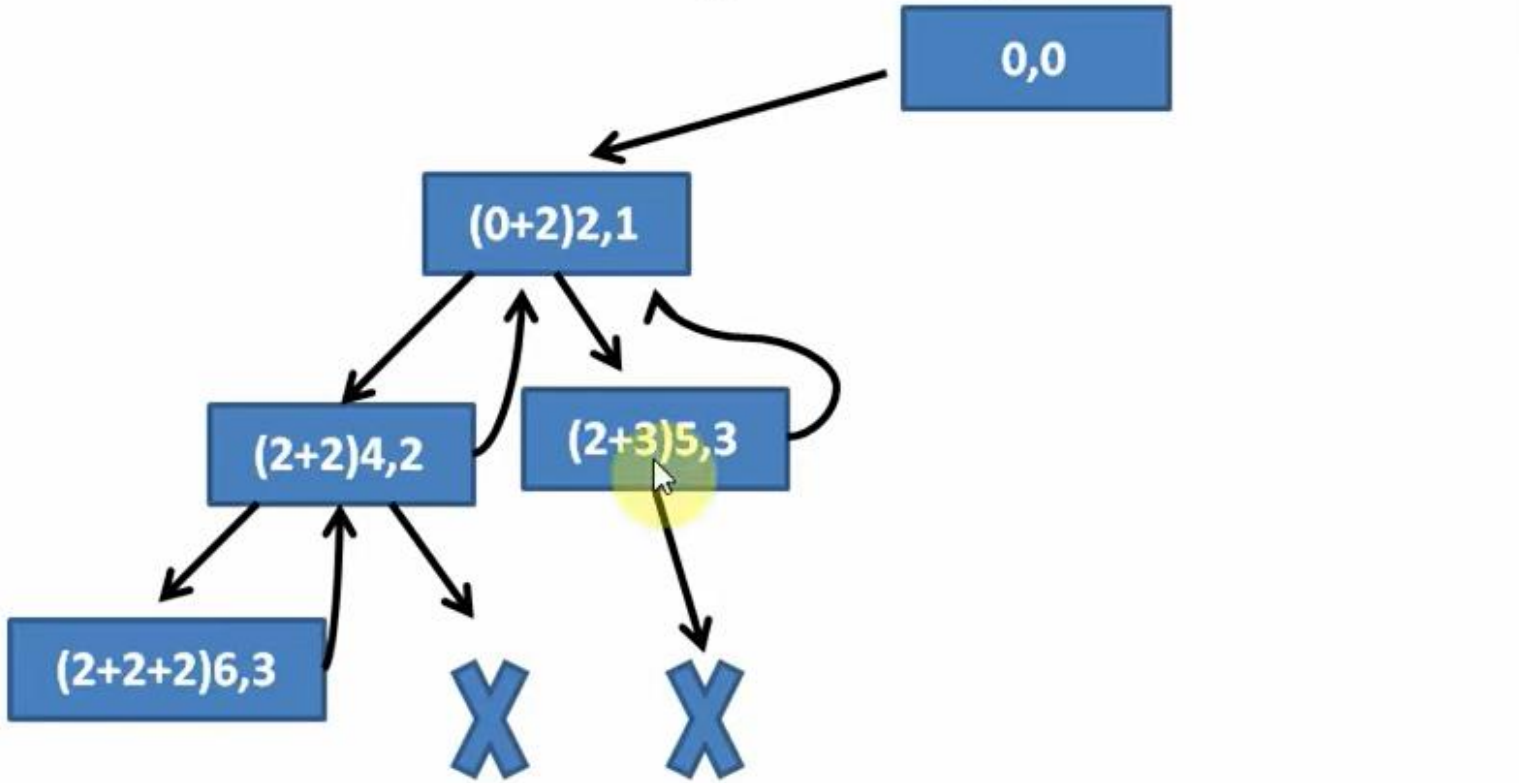
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

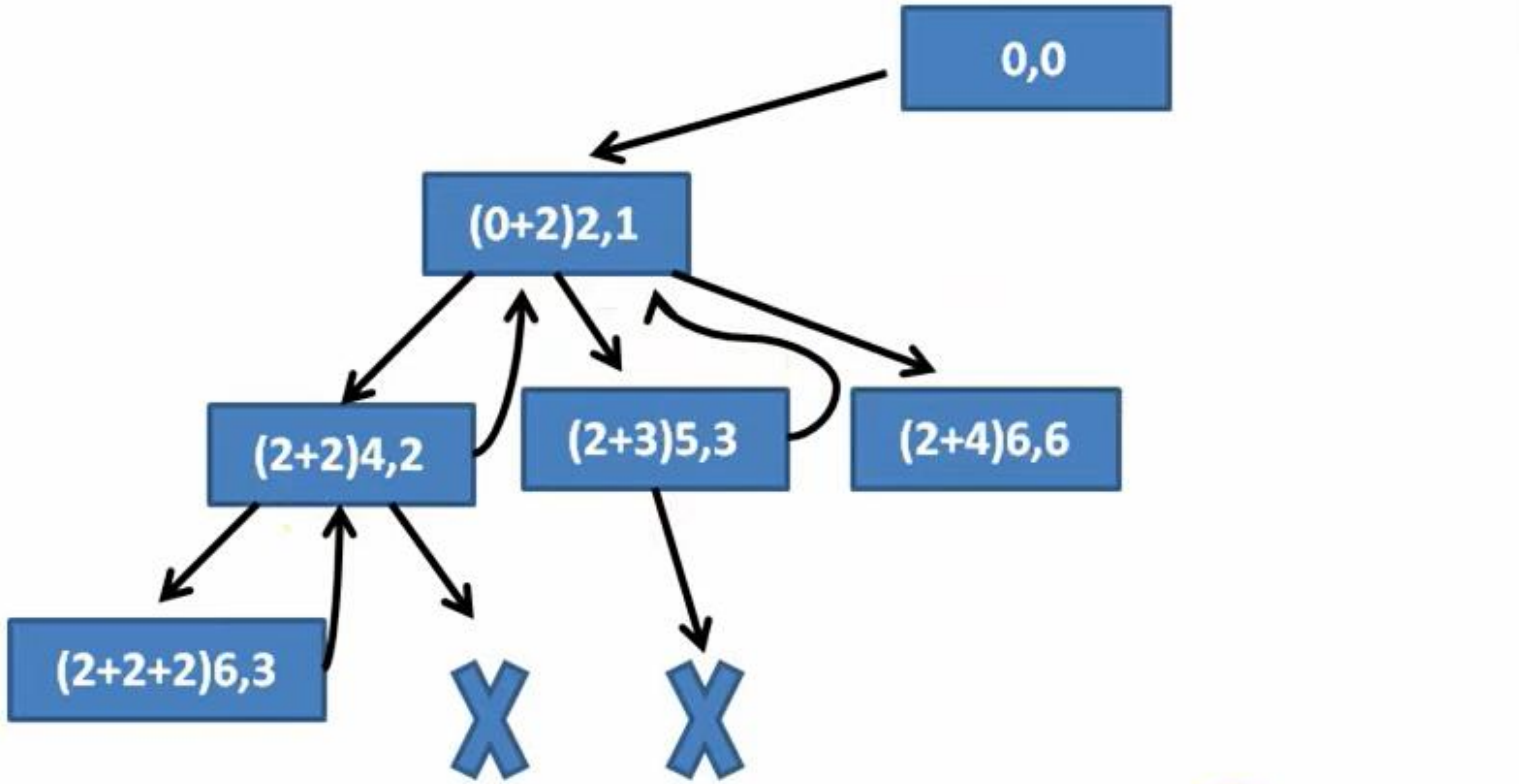
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

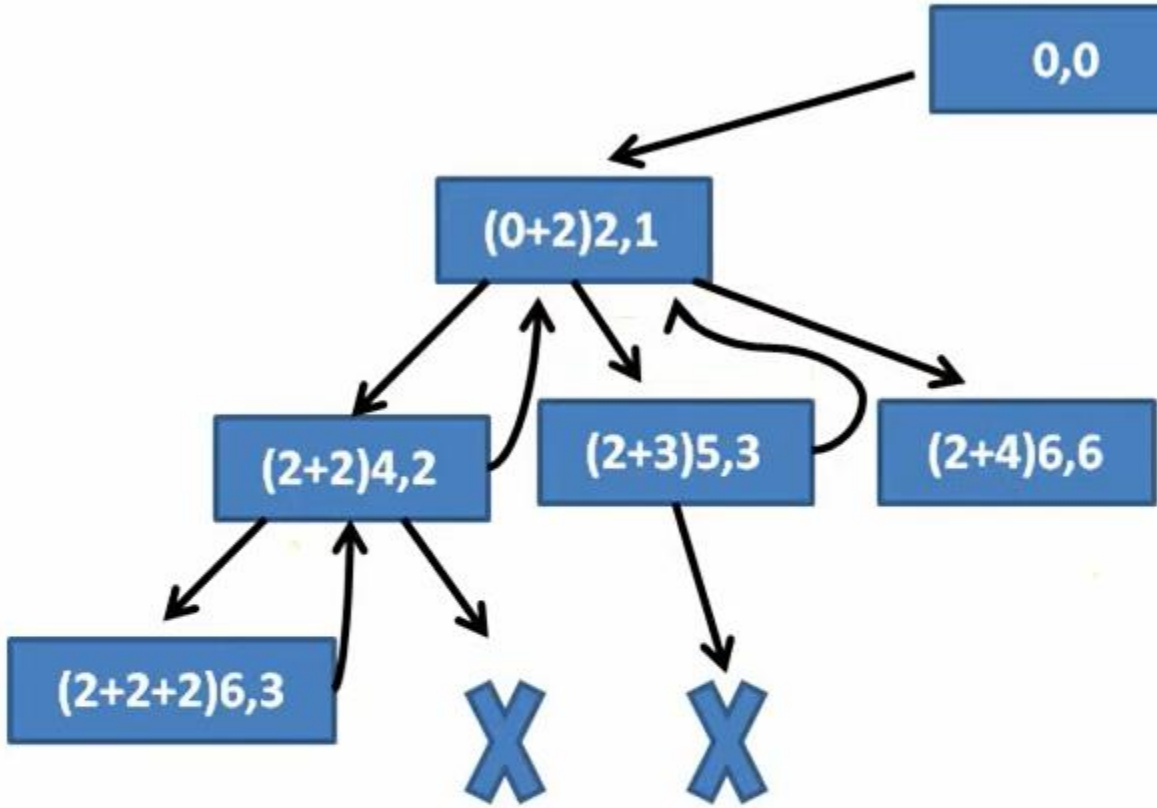
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

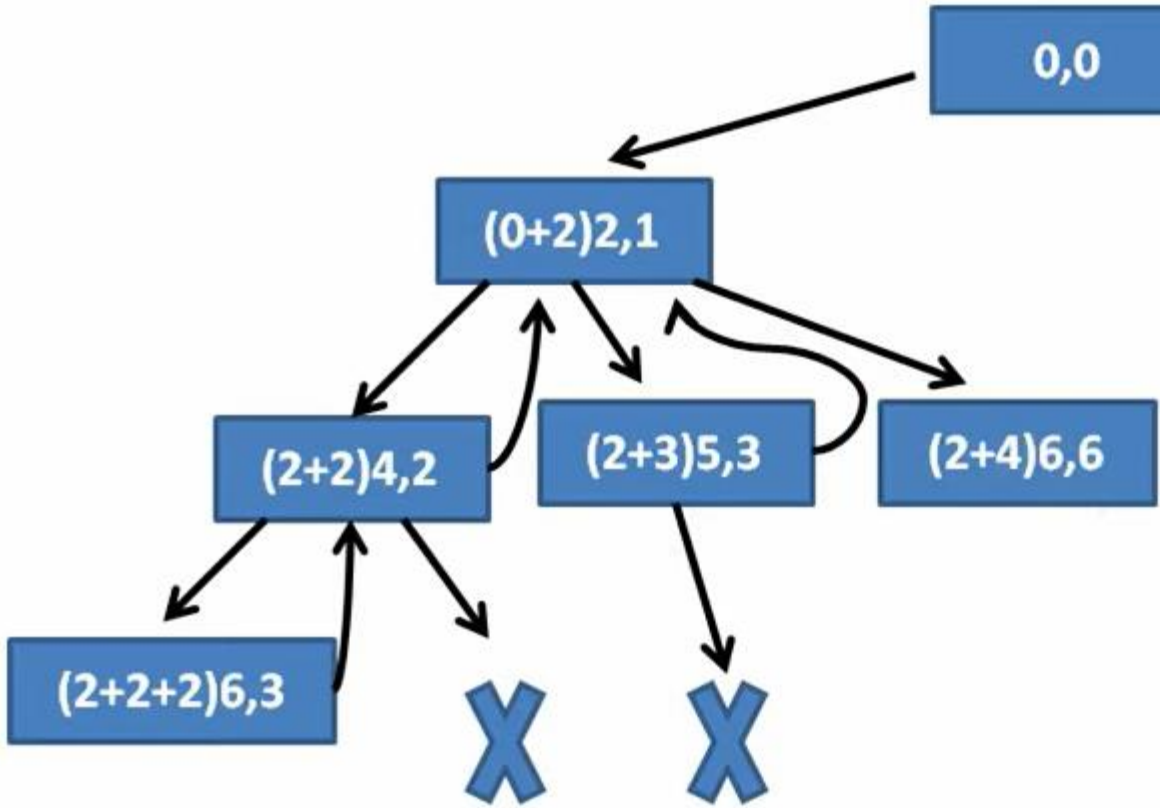
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

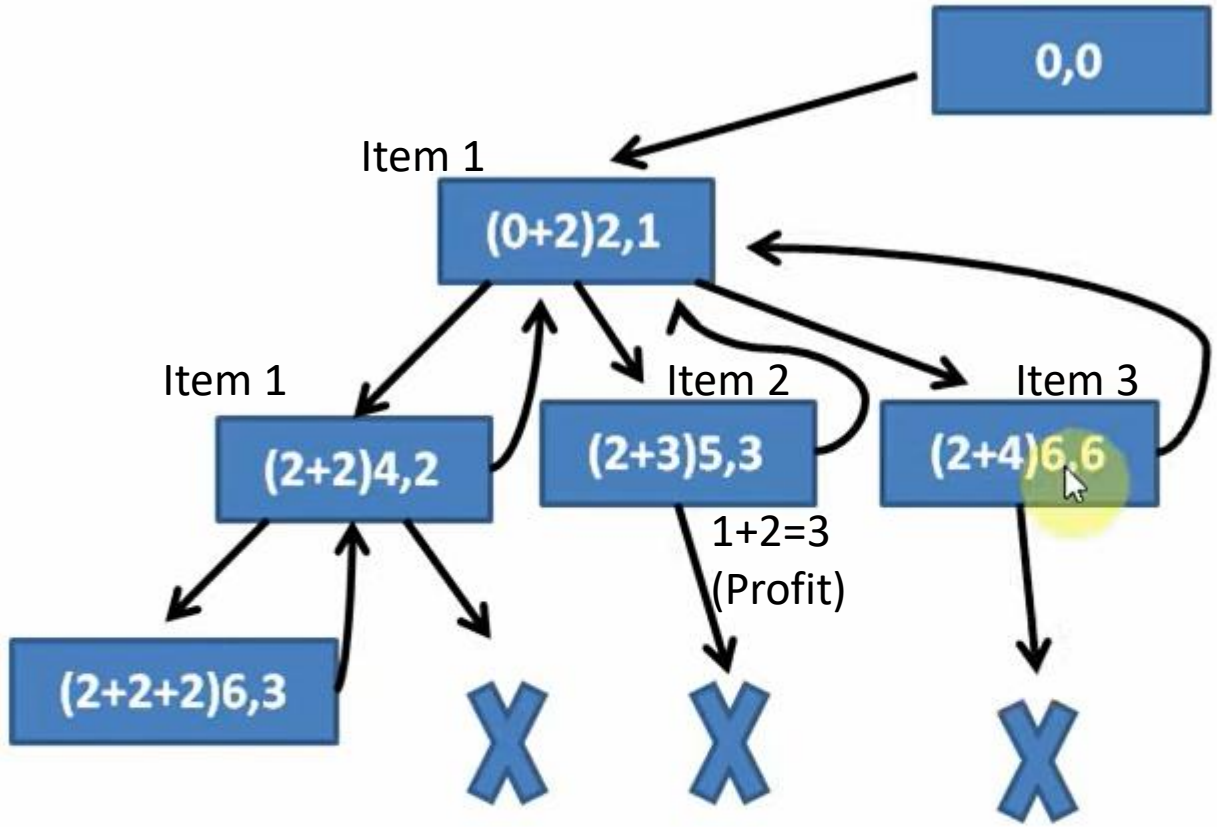
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

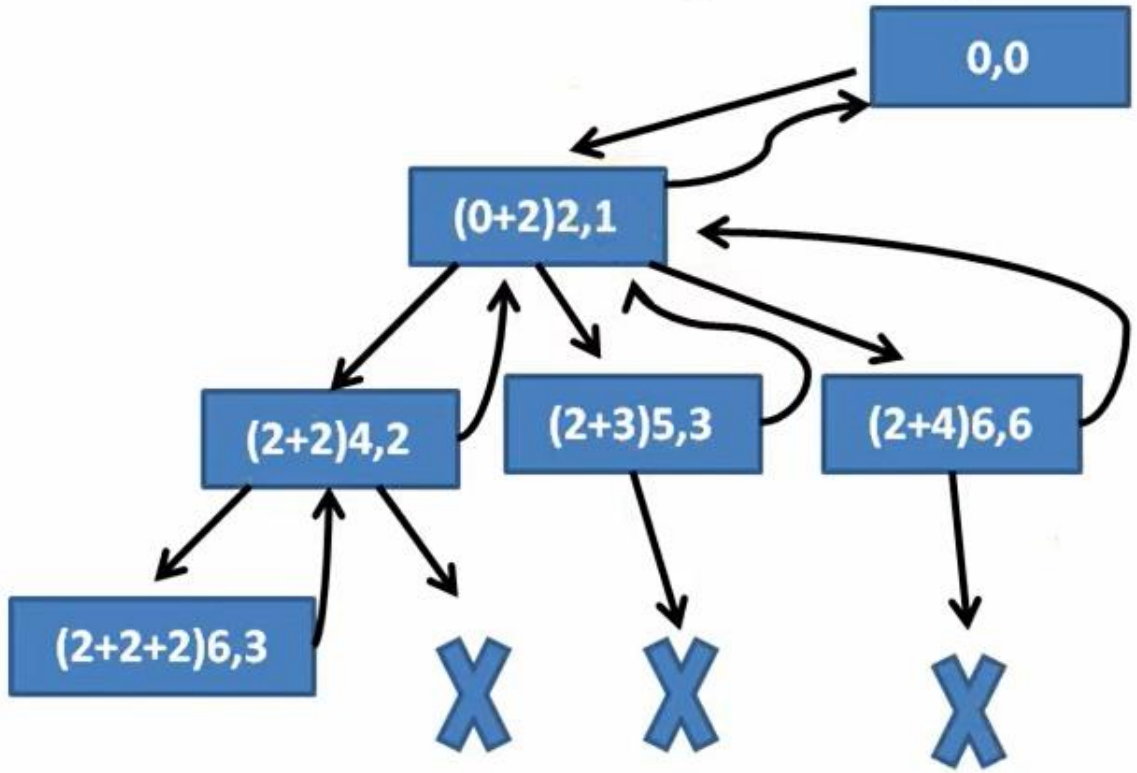
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

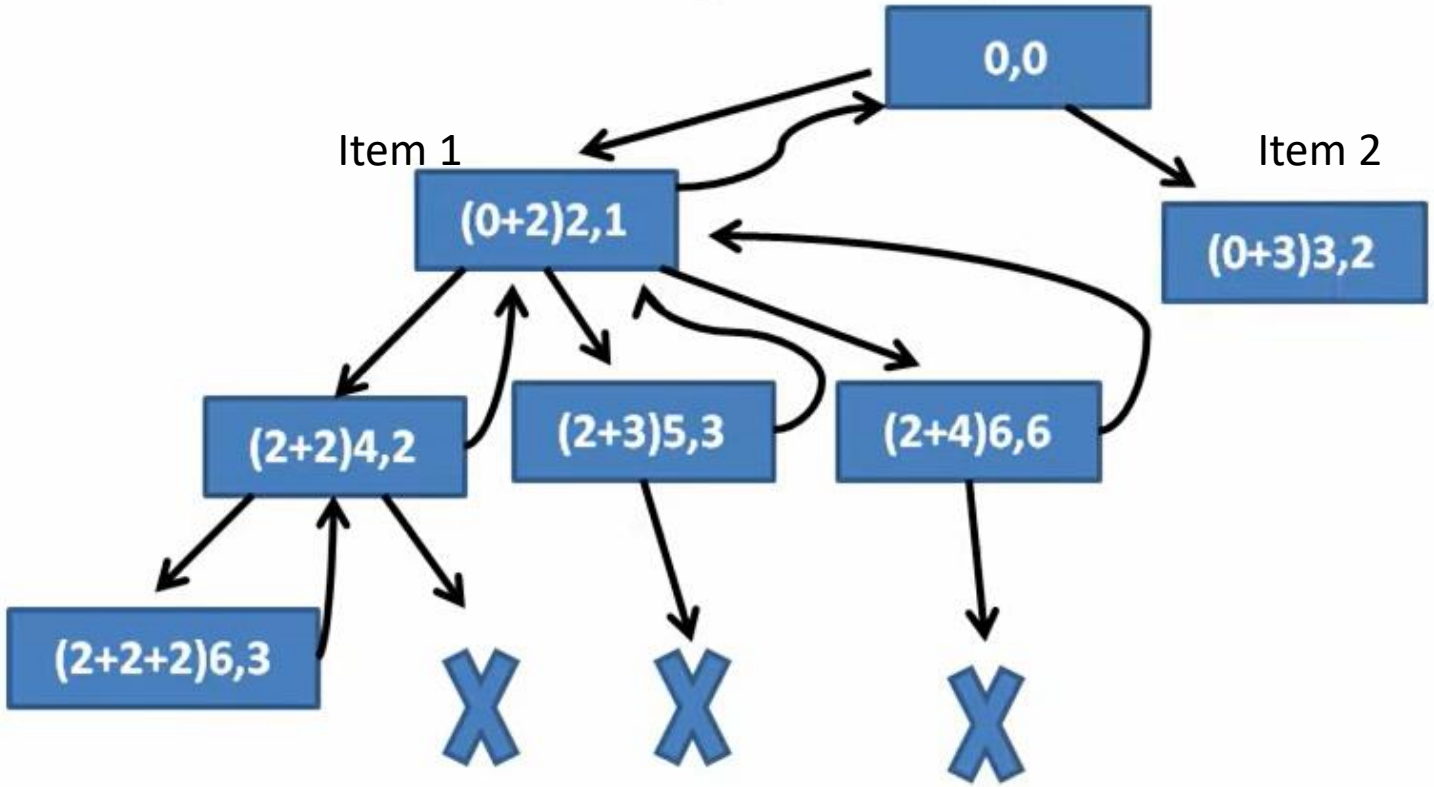
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

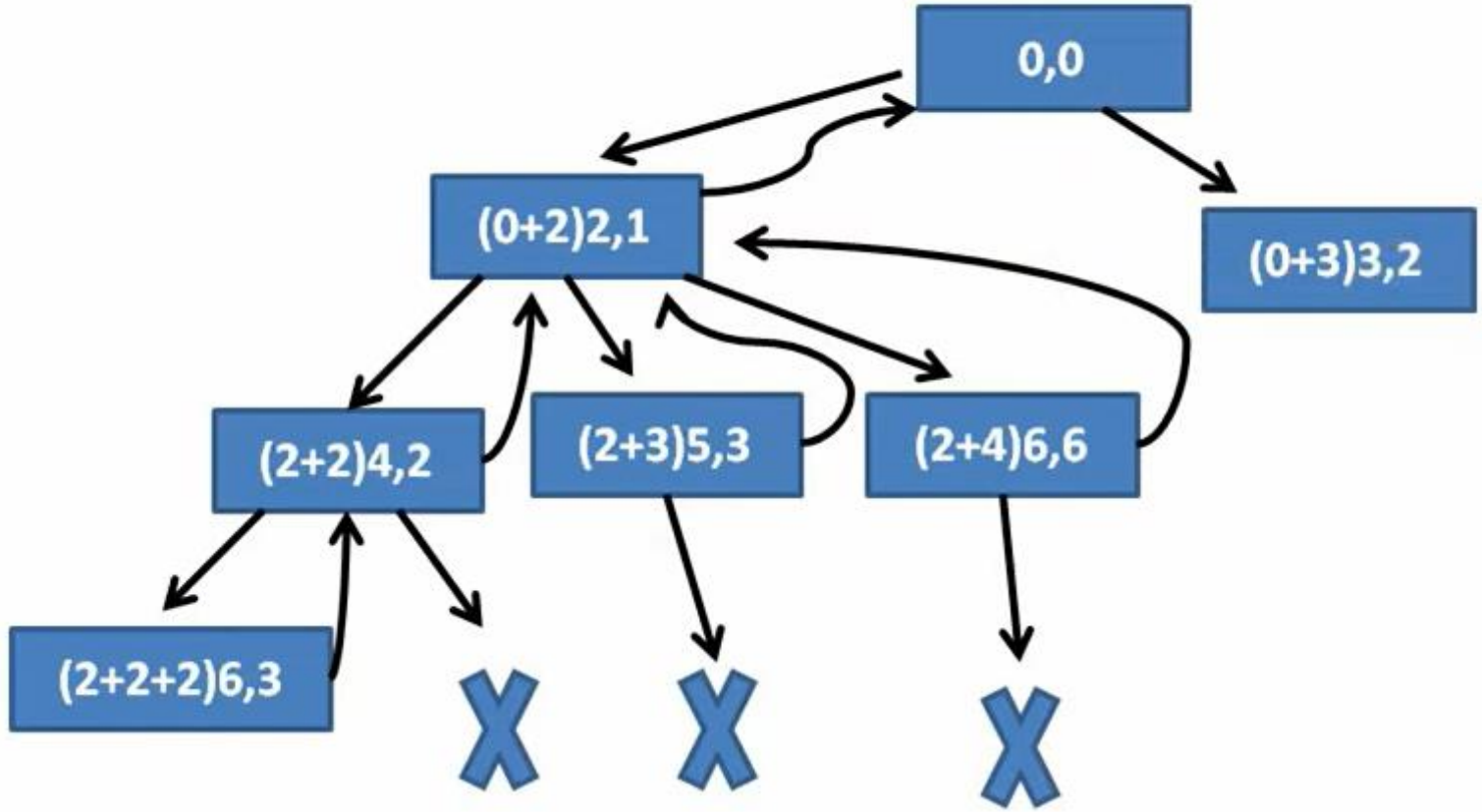
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

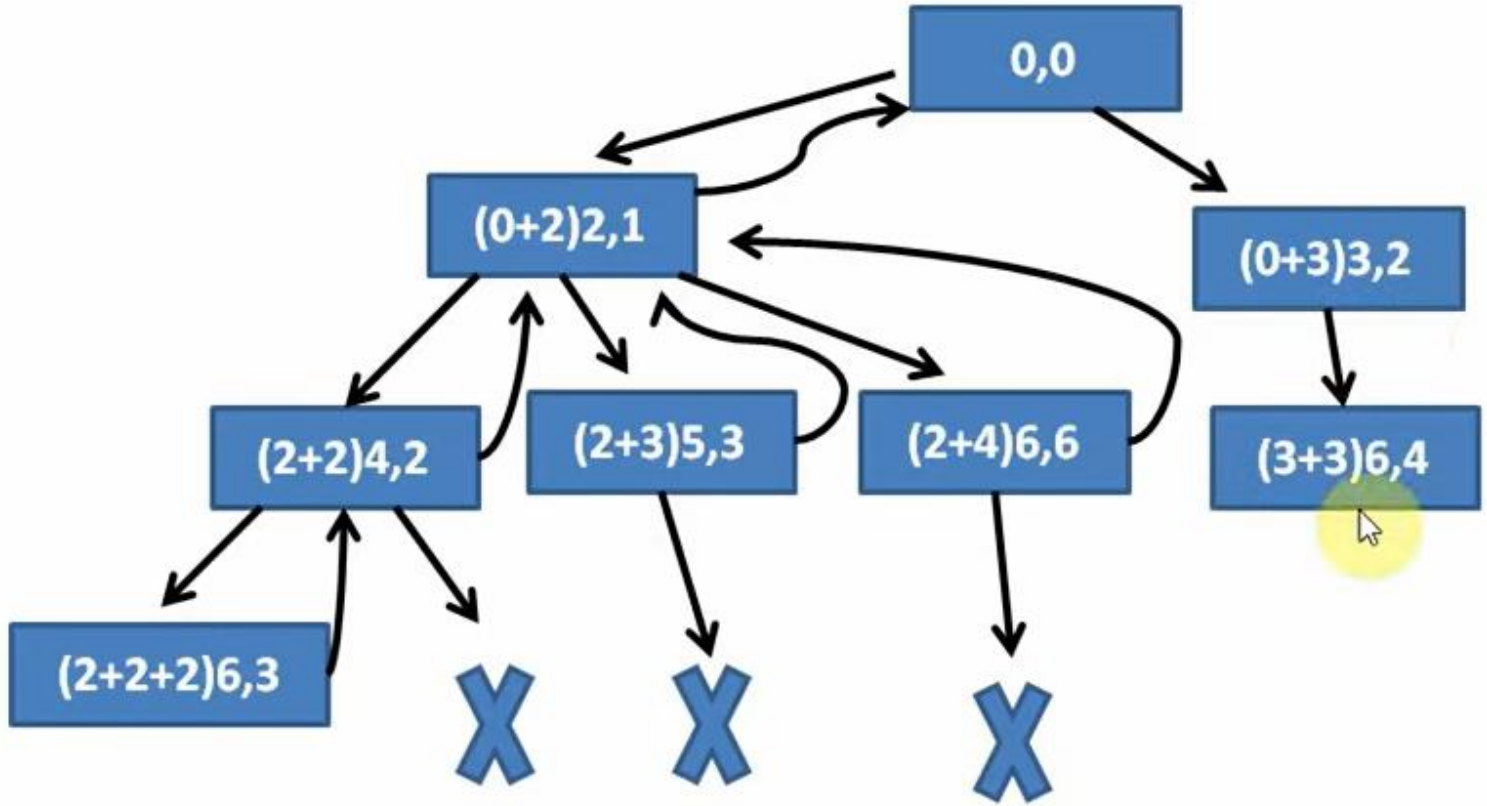
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

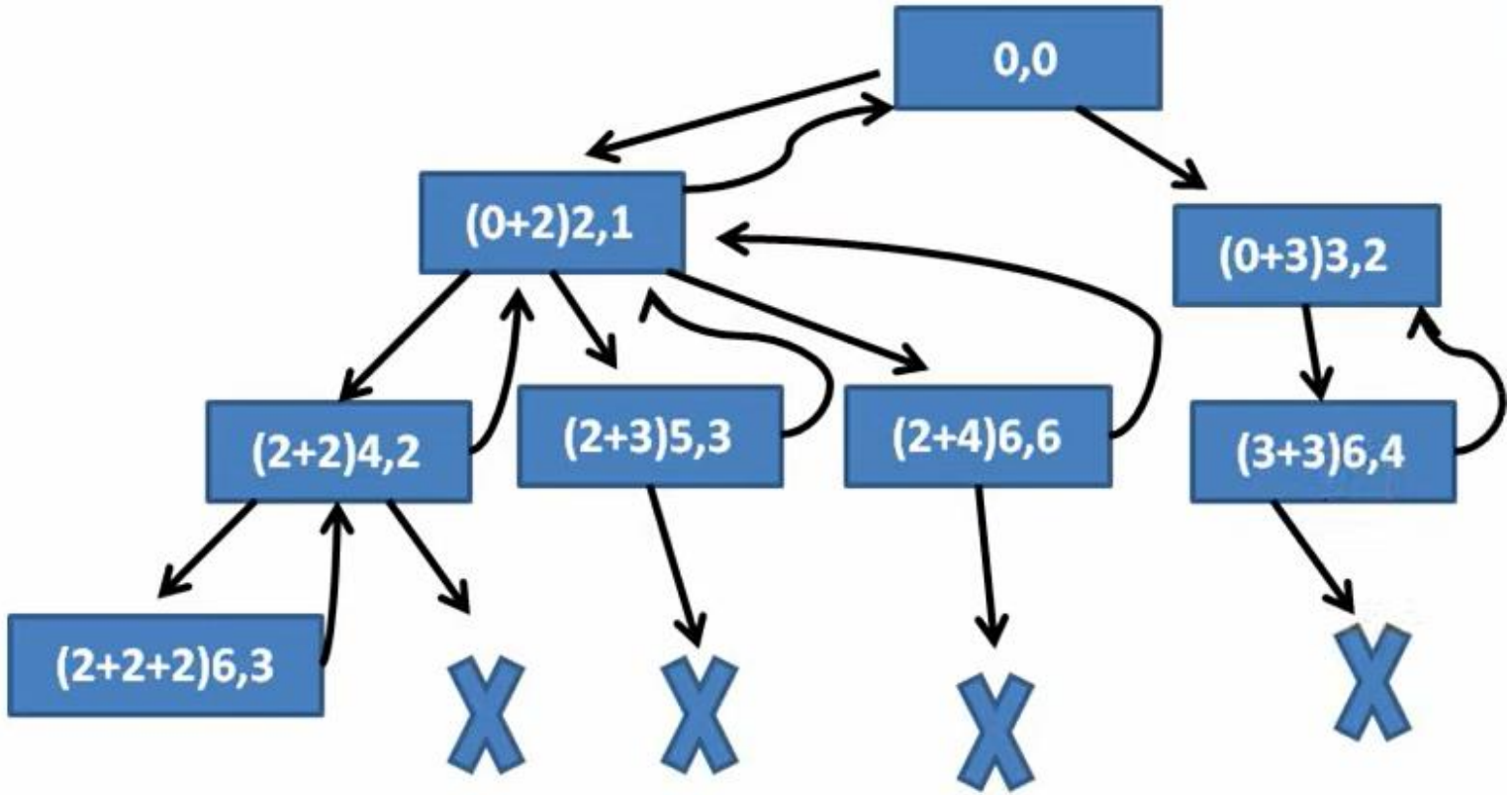
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

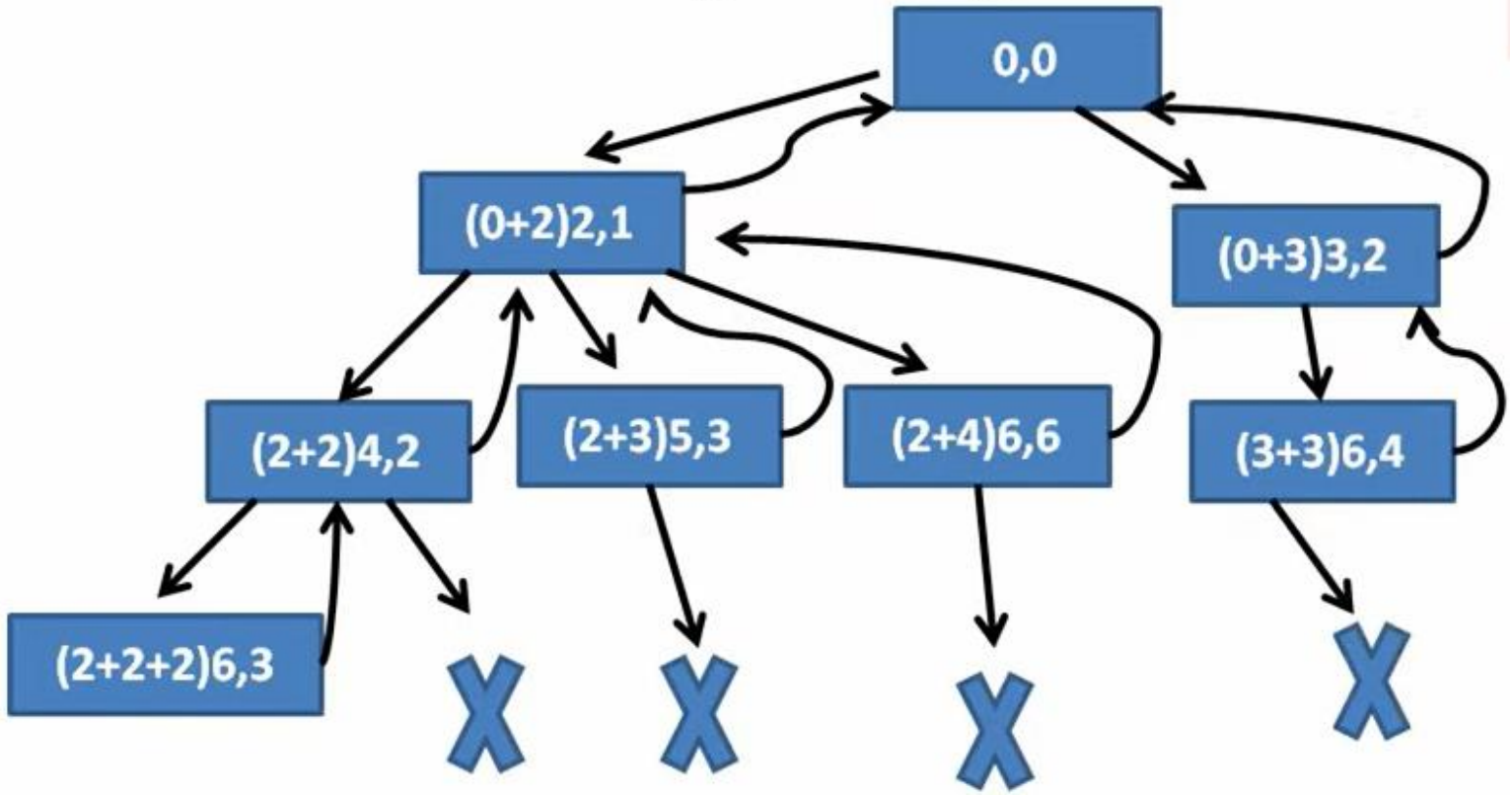
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

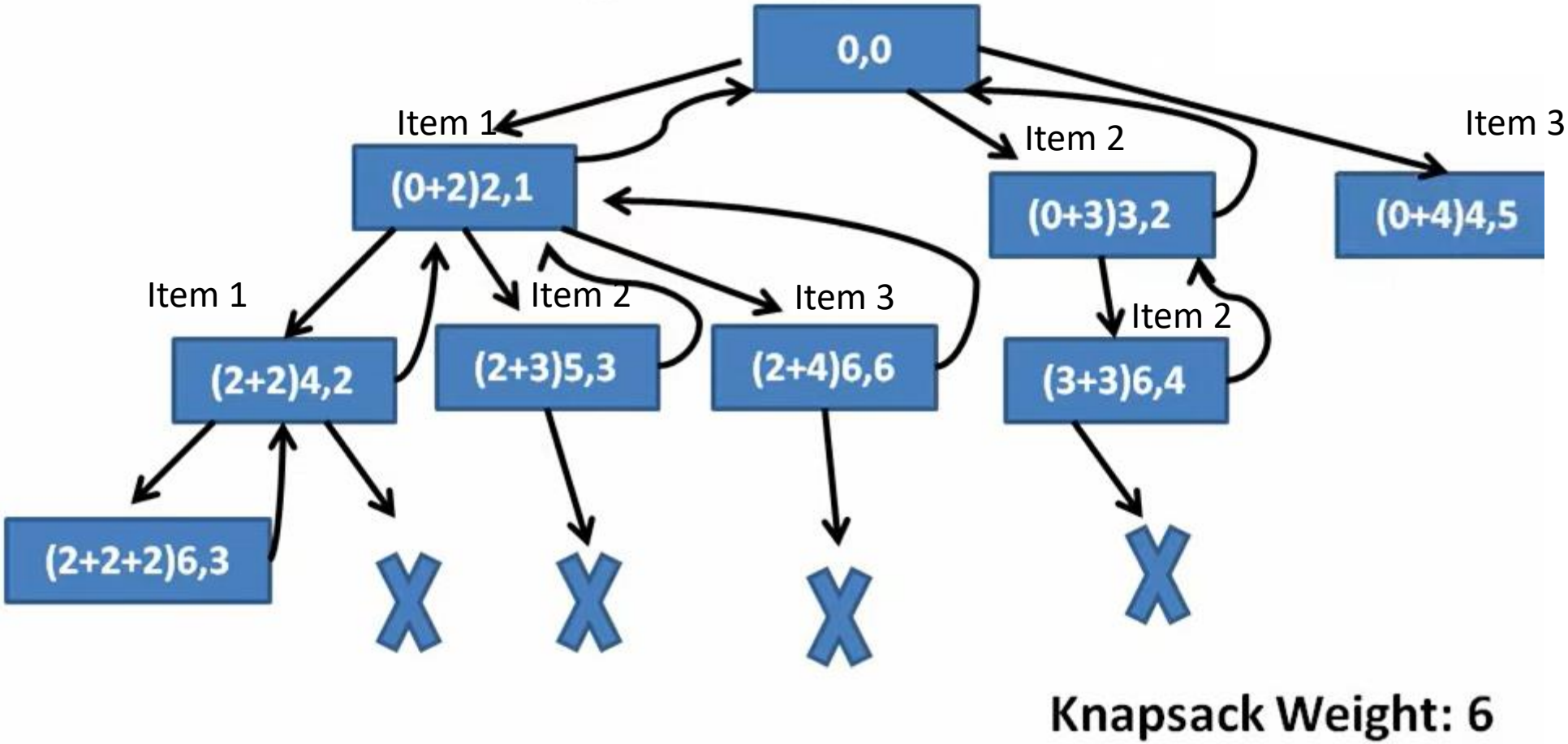
0/1 knapsack problem



Knapsack

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

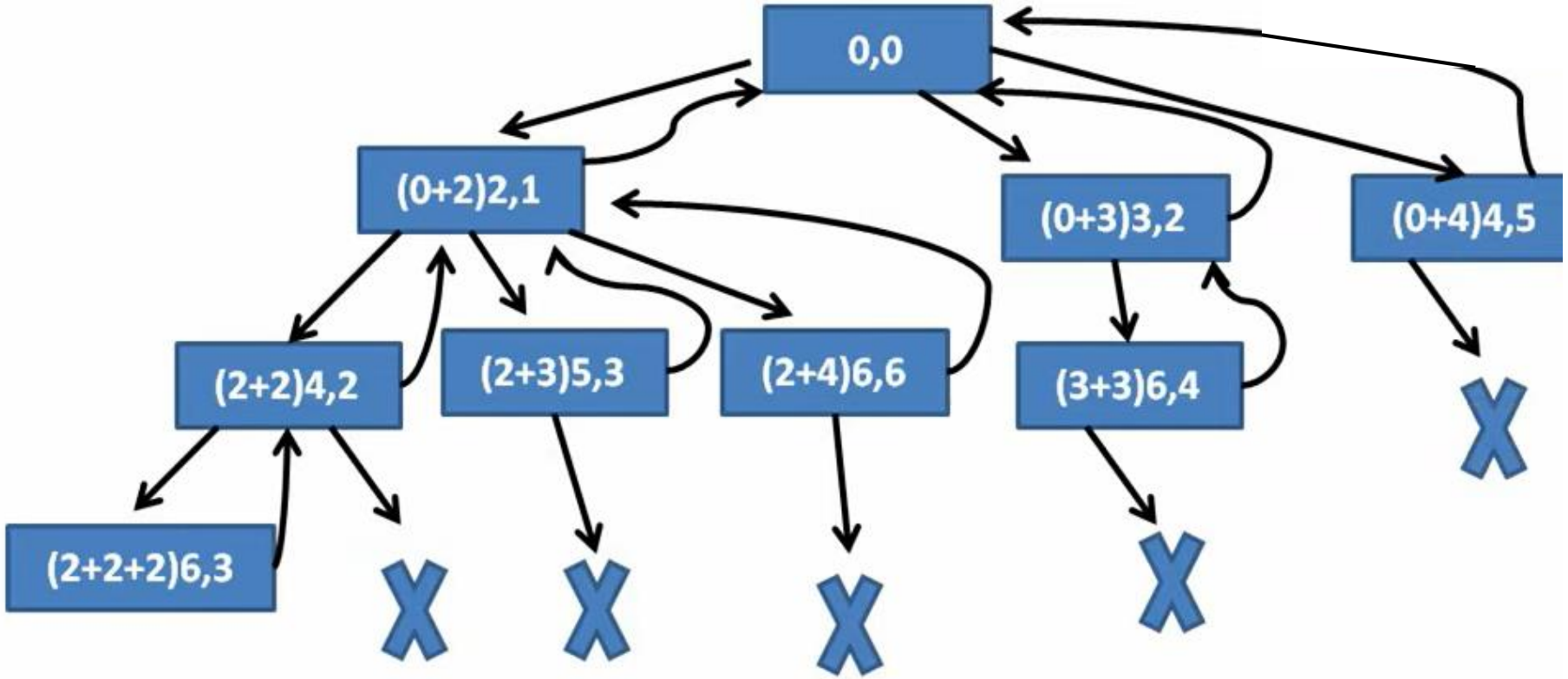
0/1 knapsack problem



Knapsack Weight: 6

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

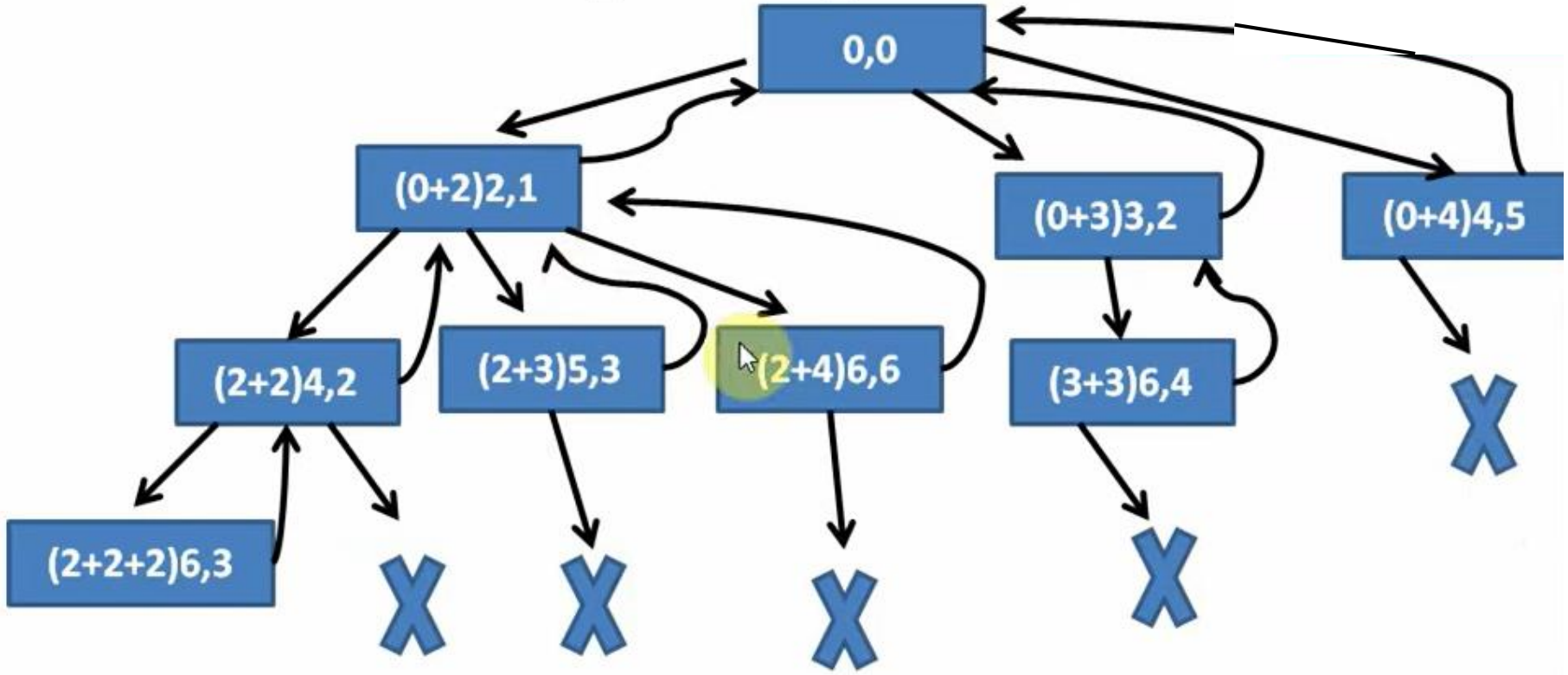
0/1 knapsack problem



Knapsack Weight: 6

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

0/1 knapsack problem



Knapsack Weight: 6

Item No.	Item1	Item2	Item3
weight	2	3	4
Profit (or) Value	1	2	5

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
void knapsackBacktrack(const vector<int>& weights, const
vector<int>& values, int W, int index, int currentWeight, int
currentValue, int& maxValue) {
    // Base case: all items considered
    if (index == weights.size()) {
        maxValue = max(maxValue, currentValue);
        return;
    }
knapsackBacktrack(weights, values, W, index + 1,
currentWeight, currentValue, maxValue);

if (currentWeight + weights[index] <= W) {
        knapsackBacktrack(weights, values, W, index + 1,
currentWeight + weights[index], currentValue +
values[index], maxValue);
    }
}
```

```
int knapsack(const vector<int>& weights, const vector<int>&
values, int W) {
    int maxValue = 0;
    knapsackBacktrack(weights, values, W, 0, 0, 0, maxValue);
    return maxValue;
}

int main() {
    vector<int> weights = {1, 3, 4, 5};
    vector<int> values = {1, 4, 5, 7};
    int W = 7;

    int result = knapsack(weights, values, W);
    cout << "Maximum value in knapsack = " << result << endl;
    return 0;
}
```

Time Complexity: $O(2^N)$
Auxiliary Space: O(N)