Backtracking (2)

By: Aminul Islam

Based on Chapter 5 of Foundations of Algorithms

Objectives

Objectives

- Describe the backtrack programming technique
- Determine when the backtracking technique is an appropriate approach to solving a problem
- Define a state space tree for a given problem
- Define when a node in a state space tree for a given problem is promising/non-promising
- Create an algorithm to prune a state space tree
- Create an algorithm to apply the backtracking technique to solve a given problem

Sum of Subsets Problem

Sum of Subsets Problem

- Let $S = \{s_1, s_2, \ldots, s_n\}$

Sum of Subsets Problem

- Let $S = \{s_1, s_2, \ldots, s_n\}$
- Let W be a positive integer

Sum of Subsets Problem

- Let $S = \{s_1, s_2, \ldots, s_n\}$
- Let W be a positive integer
- Find every $S' \subseteq S$ such that $\sum_{s \in S'} s = W$

Sum of Subsets Problem

- Let $S = \{s_1, s_2, \ldots, s_n\}$
- Let W be a positive integer
- Find every $S' \subseteq S$ such that $\sum_{s \in S'} s = W$
- There are n positive integer values (weights) and a given positive integer W (constraint).

Sum of Subsets Problem

- Let $S = \{s_1, s_2, \ldots, s_n\}$
- Let W be a positive integer
- Find every $S' \subseteq S$ such that $\sum_{s \in S'} s = W$
- There are n positive integer values (weights) and a given positive integer W (constraint).
- The goal is to find all subsets of integers that sum to W.

Sum of Subsets Problem (Brute force Algorithm)

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases $=$

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases $=$ $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases = $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
- $= \sum_{i=1}^{4} \binom{4}{i}$

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases = $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
- $= \sum_{i=1}^{4} \binom{4}{i}$
- If there are n integers in the set then, number of possible solutions would be

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases = $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
- $=\sum_{i=1}^{4} \binom{4}{i}$
- If there are n integers in the set then, number of possible solutions would be
- $\sum_{i=1}^{n} \binom{n}{i}$

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases = $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
- $= \sum_{i=1}^{4} \binom{4}{i}$
- If there are n integers in the set then, number of possible solutions would be
- $\sum_{i=1}^{n} \binom{n}{i}$
- Time complexity using Brute force algorithm would be

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases = $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
- $= \sum_{i=1}^{4} \binom{4}{i}$
- If there are n integers in the set then, number of possible solutions would be
- $\sum_{i=1}^{n} \binom{n}{i}$
- Time complexity using Brute force algorithm would be
- $T(n) = 2^n - 1$

Sum of Subsets Problem (Brute force Algorithm)

- Let $S = \{1, 2, 3, 4\}$ and $W = 6$
- Number of possible solutions/cases = $\binom{4}{4} + \binom{4}{3} + \binom{4}{2} + \binom{4}{1}$
- $=\sum_{i=1}^{4} \binom{4}{i}$
- If there are n integers in the set then, number of possible solutions would be
- $\sum_{i=1}^{n} \binom{n}{i}$
- Time complexity using Brute force algorithm would be
- $T(n) = 2^n - 1$
- Order of complexity $= 2^n$

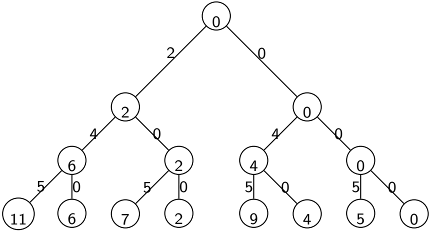
Example of Sum of Subsets Problem

Example of Sum of Subsets Problem

- $n = 3$
- $W = 6$
- $w_1 = 2$
- $w_2 = 4$
- $w_3 = 5$

Example of Sum of Subsets Problem

- $n = 3$
- $W = 6$
- $w_1 = 2$
- $w_2 = 4$
- $w_3 = 5$



Prune the Tree

Prune the Tree

- Sort weights in ascending order before search

Prune the Tree

- Sort weights in ascending order before search
- Let *weight* be the sum of weights that have been included up to a node at level i:

Prune the Tree

- Sort weights in ascending order before search
- Let *weight* be the sum of weights that have been included up to a node at level i:
  - node at level i is promising if *weight* $+w_{i+1} <= W$

Prune the Tree

- Sort weights in ascending order before search
- Let *weight* be the sum of weights that have been included up to a node at level i:
  - node at level i is promising if *weight* $+w_{i+1} <= W$
- Let *total* be the total weight of the remaining weights at a given node.

Prune the Tree

- Sort weights in ascending order before search
- Let *weight* be the sum of weights that have been included up to a node at level i:
  - node at level i is promising if *weight* $+w_{i+1} <= W$
- Let *total* be the total weight of the remaining weights at a given node.
- A node is promising if *weight* $+$ *total* $>= W$

Example of Sum of Subsets Problem using backtracking

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

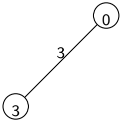
- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

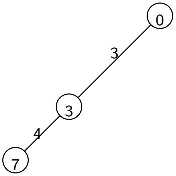
- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$



Promising:
- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

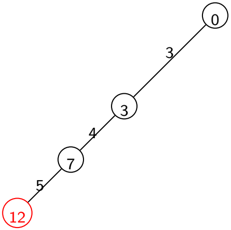


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

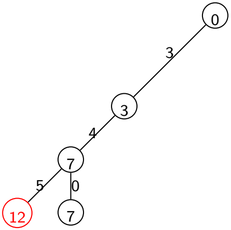


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

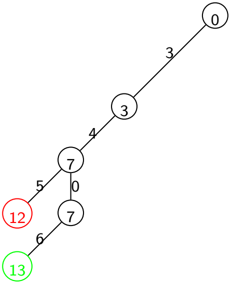
- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$



Promising:
- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

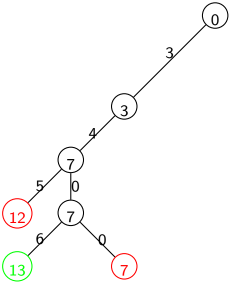


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

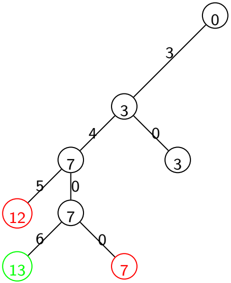


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

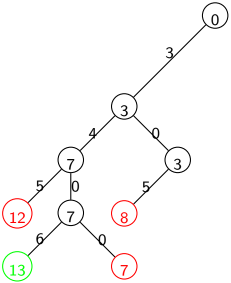


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

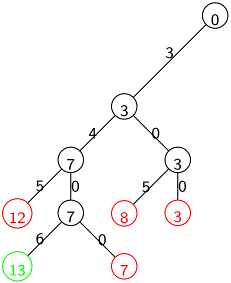


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

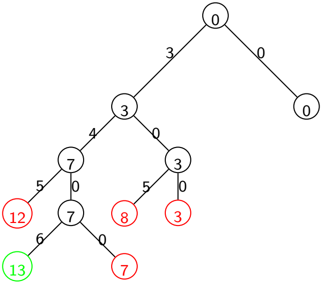


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

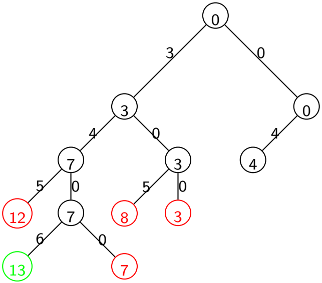


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

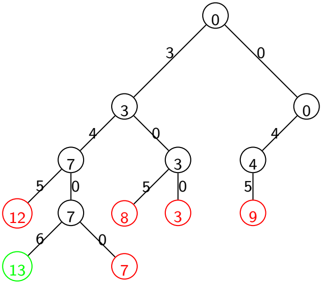


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

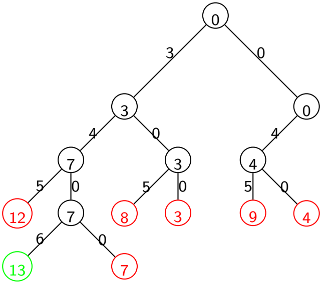
- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$



Promising:
- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$

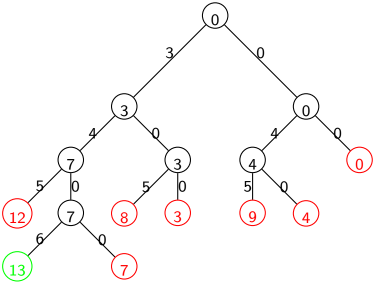


Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Example of Sum of Subsets Problem using backtracking

- $n = 4$
- $W = 13$
- $w_1 = 3$
- $w_2 = 4$
- $w_3 = 5$
- $w_4 = 6$



Promising:

- (current node) weight + next node weight $<= W$
- (current node) weight + total of remaining nodes weight $>= W$

Backtracking Alg. for Sum of Subsets Problem

Backtracking Alg. for Sum of Subsets Problem

```
void sumOfsubset(index i, int weight, int total){
    if (promising(i))
       if (weight==W)
          cout<< include[1] through include[i];
       else {
          include[i+1]=‘‘yes’’;
          sumOfsubset (i+1, weight+w[i+1], total-w[i+1]);
          include[i+1]=‘‘no’’;
          sumOfsubset (i+1, weight, total-w[i+1]);
       }
}
                // Call the function with sumOfsubset(0,0, total)
                //total:  initially sum of weights of all items

bool promising (index i){
    return (weight + total ≥ W) && (weight + w[i+1]≤W);
}
```

0-1 Knapsack Problem Using Backtracking

0-1 Knapsack Problem Using Backtracking

- It can be solved similar to the Sum of subsets problem

0-1 Knapsack Problem Using Backtracking

- It can be solved similar to the Sum of subsets problem
- Their state space trees are also similar

0-1 Knapsack Problem Using Backtracking

- It can be solved similar to the Sum of subsets problem
- Their state space trees are also similar
  - Go to the left from root to include first item and to the right to exclude the first item and so on . . .

0-1 Knapsack Problem Using Backtracking

- It can be solved similar to the Sum of subsets problem
- Their state space trees are also similar
  - Go to the left from root to include first item and to the right to exclude the first item and so on . . .
- The difference is that this is an "optimization" problem!

0-1 Knapsack Problem Using Backtracking

- It can be solved similar to the Sum of subsets problem
- Their state space trees are also similar
  - Go to the left from root to include first item and to the right to exclude the first item and so on ...
- The difference is that this is an "optimization" problem!
  - We do not know the optimal solution until the search is over

General Backtracking Algorithm for Optimization Problems

General Backtracking Algorithm for Optimization Problems

```
void checkNode (node v)
{
    node u;
    if (value(v) is better than best)
      best = value(v);
    if (promising(v))
      for (each child u of v)
      checkNode(u);
}
```

0-1 Knapsack Problem Using Backtracking

0-1 Knapsack Problem Using Backtracking

Promising cases:

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)
- We use greedy method as an idea to limit the backtracking tree

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)
- We use greedy method as an idea to limit the backtracking tree
  - Order items based on profit per unit (p_i/w_i)

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)
- We use greedy method as an idea to limit the backtracking tree
  - Order items based on profit per unit (p_i/w_i)
  - We define "upper bound" of profit as the case that "fractional knapsack" is applied

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)
- We use greedy method as an idea to limit the backtracking tree
    - Order items based on profit per unit (p_i/w_i)
    - We define "upper bound" of profit as the case that "fractional knapsack" is applied
    - profit: is the sum of profits of items selected so far

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)
- We use greedy method as an idea to limit the backtracking tree
  - Order items based on profit per unit (p_i/w_i)
  - We define "upper bound" of profit as the case that "fractional knapsack" is applied
  - profit: is the sum of profits of items selected so far
  - maxprofit: is the maximum profit we have found so far.

0-1 Knapsack Problem Using Backtracking

Promising cases:

- Sum of the weights up to the node $< W$ (i.e, knapsack capacity)
- We use greedy method as an idea to limit the backtracking tree
    - Order items based on profit per unit (p_i/w_i)
    - We define "upper bound" of profit as the case that "fractional knapsack" is applied
    - profit: is the sum of profits of items selected so far
    - maxprofit: is the maximum profit we have found so far.
- A node at level i is promising if:
  (upper bound)$_i >$ maxprofit

Example of 0-1 Knapsack Problem using backtracking

Example of 0-1 Knapsack Problem using backtracking

$n = 4,\ W = 16$

Item i	P_i	W_i	P_i / W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:

- (current node) bound $>$ maxprofit
- (current node) weight $< W$

Example of 0-1 Knapsack Problem using backtracking

$n = 4,\ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $=$ \$0

Example of 0-1 Knapsack Problem using backtracking

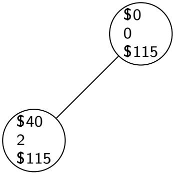
$n = 4, \ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:

- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $= \$40$



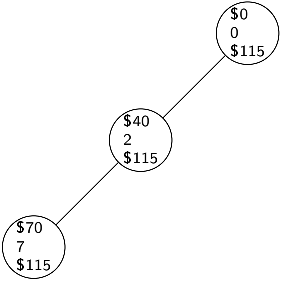
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \ W = 16$

Item i	P_i	W_i	P_i / W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $= \$70$



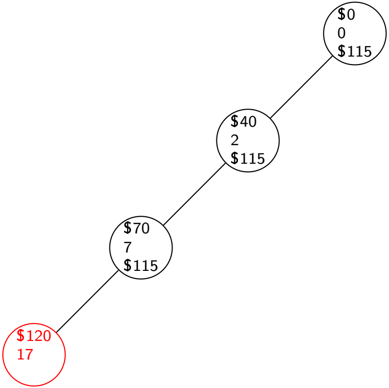
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \; W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $=$ \$70



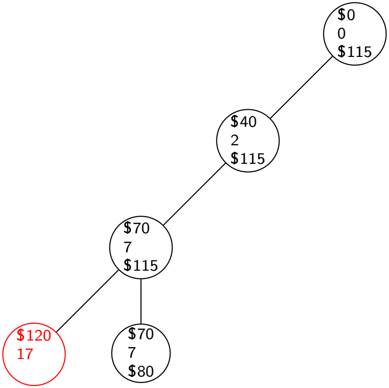
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $=$ \$70



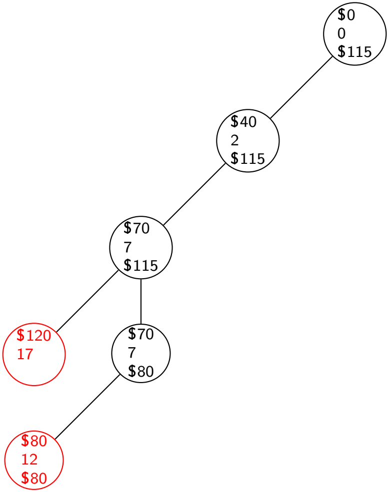
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $=$ \$80



Example of 0-1 Knapsack Problem using backtracking

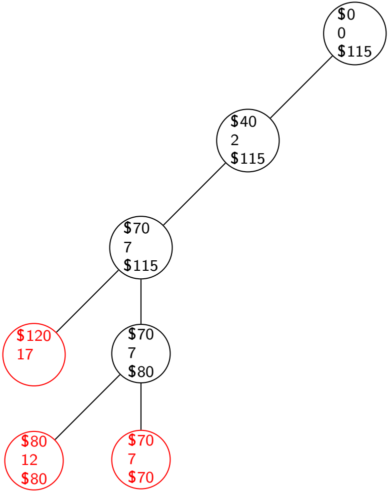
$n = 4, \; W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:

- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $=$ \$80



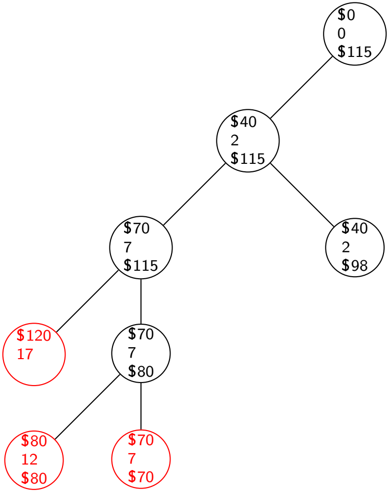
Example of 0-1 Knapsack Problem using backtracking

$n = 4,\ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $<$ W

maxprofit $=$ \$80



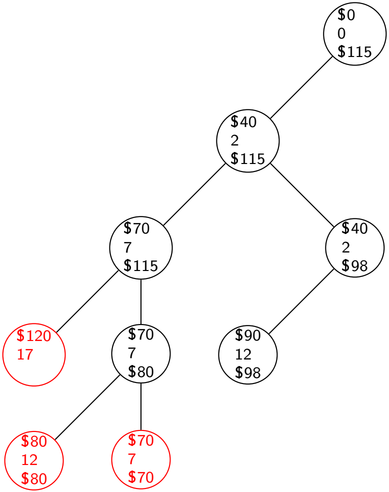
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \; W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $= \$90$



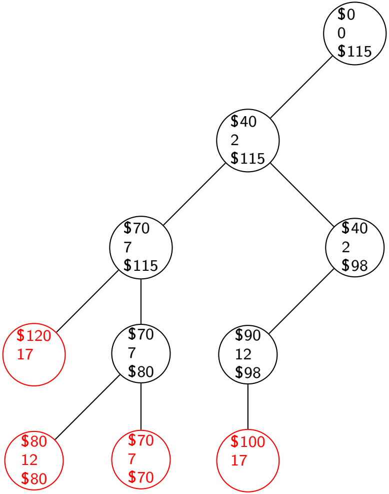
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \; W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $<$ W

maxprofit $=$ \$90



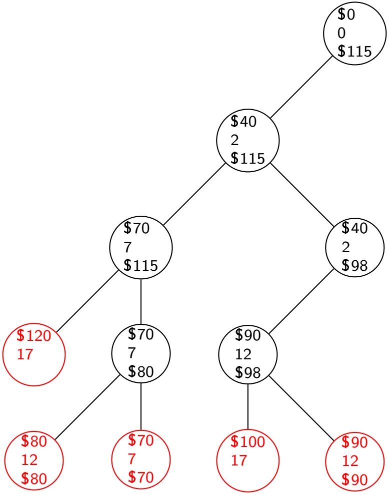
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $<$ W

maxprofit $=$ \$90



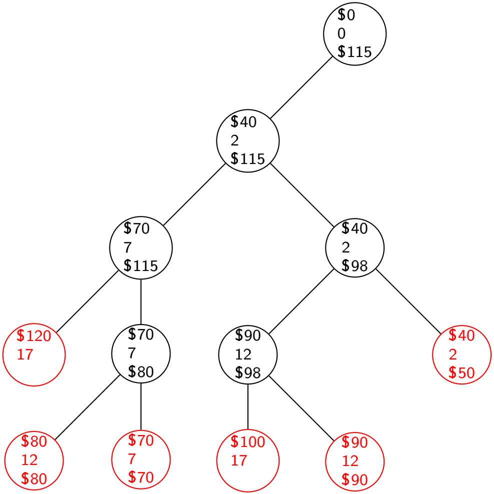
Example of 0-1 Knapsack Problem using backtracking

$n = 4, \ W = 16$

Item i	P_i	W_i	P_i/W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $<$ W

maxprofit $=$ \$90



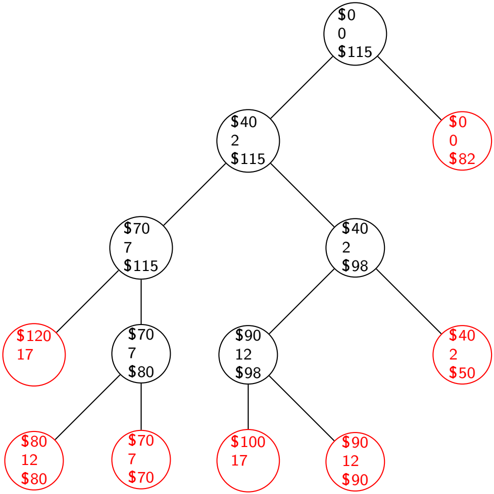
Example of 0-1 Knapsack Problem using backtracking

$n = 4,\ W = 16$

Item i	P_i	W_i	P_i / W_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

Promising:
- (current node) bound $>$ maxprofit
- (current node) weight $< W$

maxprofit $=$ \$90



0-1 Knapsack Problem using backtracking

```
void knapsack (index i,
               int profit, int weight)
{
  if (weight <= W && profit > maxprofit){
     maxprofit = profit;
     numbest = i;
     bestset = include;
  }


  if (promising(i)){
     include[i + 1] = "yes";
     knapsack(i + 1, profit + p[i + 1], weight + w[i + 1]);
     include[i + 1] = "no";
     knapsack(i + 1, profit, weight);
  }
}

bool promising (index i)
{
  index j, k;
  int totweight;
  float bound;

  if (weight >= W)
     return false;
  else{
     j = i + 1;
     bound = profit;
     totweight = weight;
     while (j <= n && totweight + w[j] < = W){

        totweight = totweight + w[j];
        bound = bound + p[j];
        j++;
     }
     k = j;
     if (k <=n)
        bound = bound + (W - totweight) * p[k]/w[k];

     return bound > maxprofit;
  }
}
```