Lecture 12.1

- Anurag Sharma & Shymal Chandra

Backtracking Algorithms: knapsack problem

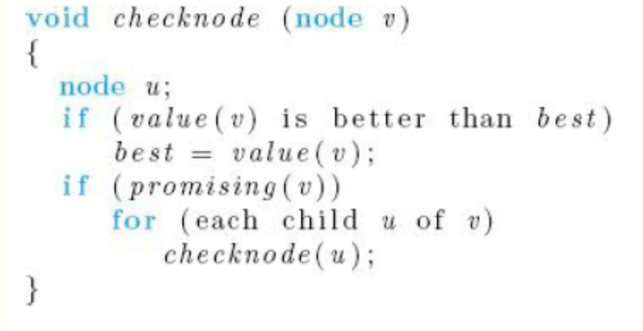
The 0-1 knapsack problem

The 0-1 knapsack problem

- Backtracking can be applied to solve the 0-1 Knapsack problem (knapsack problem - where a thief can steal items to maximize profit as well as remain within knapsack weight constraint)
- You can rank each item according to profit per weight (as in the greedy technique) and either select or not select each item
- The state space tree can be formed as follows:
  - Each node consists of a bound (maximum profit possible), the current total profit and the current total weight
  - Each level of the tree represents an item, starting with the highest ranked item (highest profit/weight)
  - From each node, a left node (child) indicates including that item and a right node indicates not including that item
  - The state space tree can be pruned by expanding at the nodes that give a higher bound (profit)

Cont.

- High level pseudocode:



```
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);
}
```

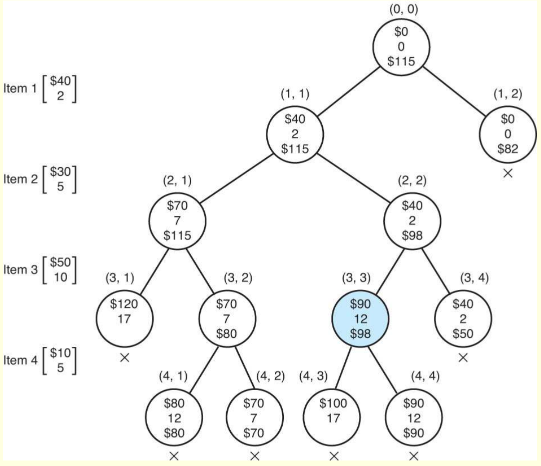
Example

Suppose that n = 4, W = 16, and we have the following:

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

cont

- Pruned state space tree after backtracking:


Item 1 [$40 2]
Item 2 [$30 5]
Item 3 [$50 10]
Item 4 [$10 5]
(0, 0) $0 0 $115
(1, 1) $40 2 $115
(1, 2) $0 0 $82 ×
(2, 1) $70 7 $115
(2, 2) $40 2 $98
(3, 1) $120 17 ×
(3, 2) $70 7 $80
(3, 3) $90 12 $98
(3, 4) $40 2 $50 ×
(4, 1) $80 12 $80 ×
(4, 2) $70 7 $70 ×
(4, 3) $100 17 ×
(4, 4) $90 12 $90 ×