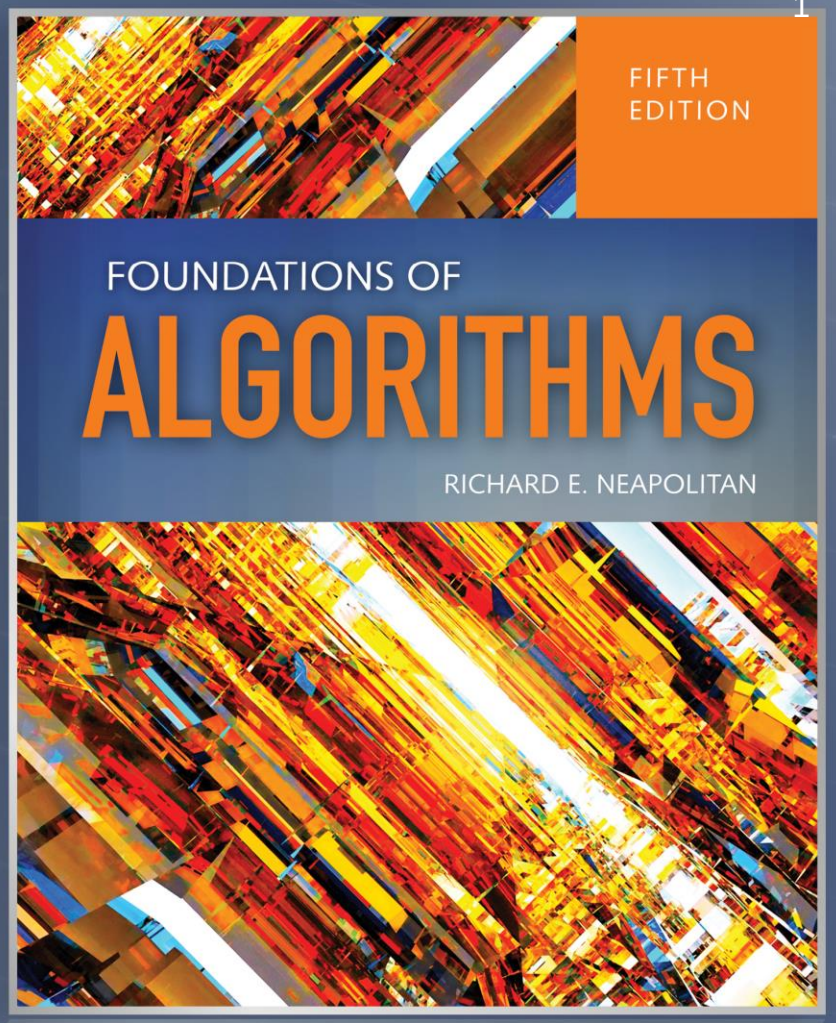


Backtracking

Chapter 5

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

Finding Your Way Thru a Maze

- Follow a path until a dead end is reached
- Go back until reaching a fork
- Pursue another path
- Suppose there were signs indicating path leads to dead end?
- Sign positioned near beginning of path – time savings enormous
- Sign positioned near end of path – very little time saved

Back Tracking

- Can be used to solve NP-Complete problems such as 0-1 Knapsack more efficiently

Backtracking vs Dynamic Programming

- Dynamic Programming – subsets of a solution are generated
- Backtracking – Technique for deciding that some subsets need not be generated
- Efficient for many large instances of a problem (but not all)

Backtracking Technique

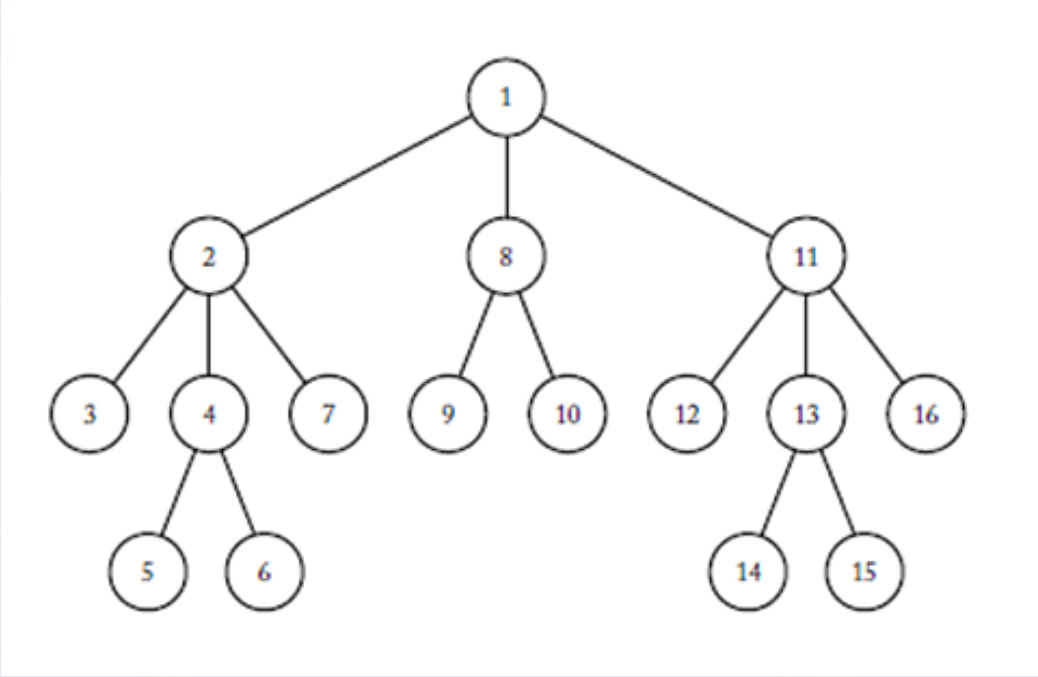
- Solve problems in which a sequence of objects is chosen from a set
- Sequence satisfies some criterion
- Modified DFS of a rooted tree
- Pre-order traversal
- General-purpose algorithm does not specify order children visited – we will use left to right

Procedure called by passing root at the top level


```
void
depth_first_tree_search(node v)
{
        node u;
        visit v
        for( each child u of v)

        depth_first_tree_searach
(u);
}
```

Figure 5.1



Backtracking Procedure

- After determining a node cannot lead to a solution, backtrack to the node's parent and proceed with the search on the next child
- Non-promising node: when the node is visited, it is determined the node cannot lead to a solution
- Promising node: may lead to a solution
- Backtracking
  - DFS of state space tree
  - Pruning state space tree: if a node is determined to be non-promising, back track to its parent

Backtracking Procedure

- Pruned State Space Tree: sub-tree consisting of visited nodes

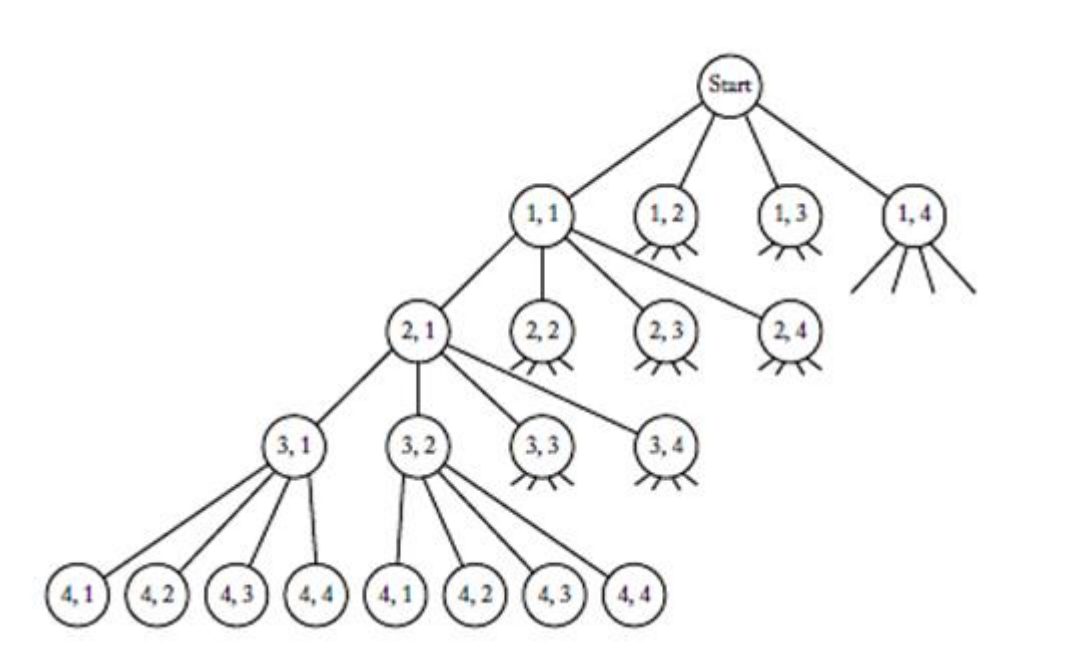
N-Queen Problem

- Goal: position n queens on a n x n board such that no two queens threaten each other
  - No two queens may be in the same row, column, or diagonal
- Sequence: n positions where queens are placed
- Set: n^2 positions on the board
- Criterion: no two queens threaten each other

e.g. 4-Queen Problem Solution

	Q		
			Q
Q			
		Q	

Figure 5.2



4-Queen Problem

- Assign each queen a different row
- Check which column combinations yield solutions
- Each queen can be in any one of four columns: 4 x 4 x 4 x 4 = 256

Construct State-Space Tree – Candidate Solutions

- Root – start node
- Column choices for first queen stored at level-1 nodes
- Column choices for second queen stored at level-2 nodes
- Etc.
- Path from root to a leaf is candidate solution
- Check each candidate solution in sequence starting with left-most path

Construct State-Space Tree – Candidate Solutions

- Example Figure 5.2 – no point in checking any candidate solutions from 2,1

Pruning

- DFS of state space tree
- Check to see if each node is promising
- If a node is non-promising, backtrack to node's parent
- Pruned state space tree – subtree consisting of visited nodes
- Promising function – application dependent
- Promising function n-queen problem: returns false if a node and any of the node's ancestors place queens in the same column or diagonal

checknode

- Backtracking algorithm for the n-Queens Problem
- State space tree implicit – tree not actually created
  - Values in current branch under investigation kept track of
- Algorithm inefficient
- Checks node promising after passing it to procedure
- Activation records checked and pushed onto stack for non-promising nodes

expand

- Improves efficiency
- Check to see if node is promising before passing the node

Algorithm 5.1 Backtracking Algorithm for N-Queens Problem

- All solutions to N-Queens problem
- Promising function:
  - 2 queens same row? col(i) == col(k)
  - 2 queens same diagonal? col(i) – col(k) == i – k || col(i) – col(k) == k - i

Analysis of queens theoretically difficult

- Upper bound on number of nodes checked in pruned state space tree by counting number of nodes in entire state space tree:
  - 1 node level 0
  - 2 nodes level 1
  - n^2 nodes level 2 . . .
  - N^n nodes level n

Total Number of nodes

$$1+n+n^2+n^3+\ldots+n^n=\frac{n^{n+1}-1}{n-1}$$

Let n = 8

- Number of nodes in state space tree = 19173961
- Purpose of backtracking is to avoid checking many of these nodes
- Difficult to theoretically analyze savings by backtracking

Illustrate backtracking savings by executing code and counting nodes checked

- Algorithm1 – DFS of state space tree without backtracking
- Algorithm2 – checks no two queens same row or same column

Table 5.1

n	Number of Nodes Checked by Algorithm 1†	Number of Candidate Solutions Checked by Algorithm 2‡	Number of Nodes Checked by Backtracking	Number of Nodes Found Promising by Backtracking
4	341	24	61	17
8	19,173,961	40,320	15,721	2057
12	9.73×10^{12}	4.79×10^{8}	1.01×10^{7}	8.56×10^{5}
14	1.20×10^{16}	8.72×10^{10}	3.78×10^{8}	2.74×10^{7}

*Entries indicate numbers of checks required to find all solutions.

†Algorithm 1 does a depth-first search of the state space tree without backtracking.

‡Algorithm 2 generates the $n!$ candidate solutions that place each queen in a different row and column.

Sum-of-Subsets Problem

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

$$\sum_{s \in S'} s = W$$

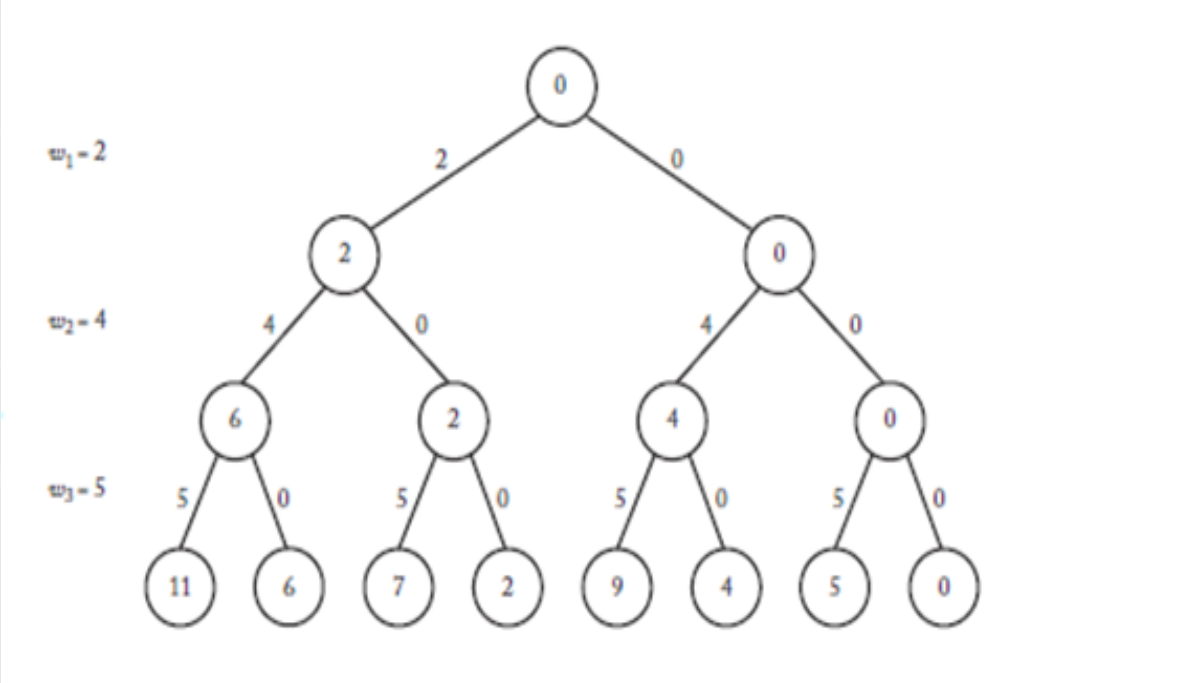
Example

- S = {$w_1 = 5$, $w_2 = 6$, $w_3 = 10$, $w_4 = 11$, $w_5 = 16$} and W=21
- Solutions:
  - {w_1 , w_2 , w_3 } : 5 + 6 + 10 = 21
  - {w_1 , w_5 } : 5 + 16 = 21
  - {w_3 , w_4 }: 10 + 11 = 21

Example 5.3

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

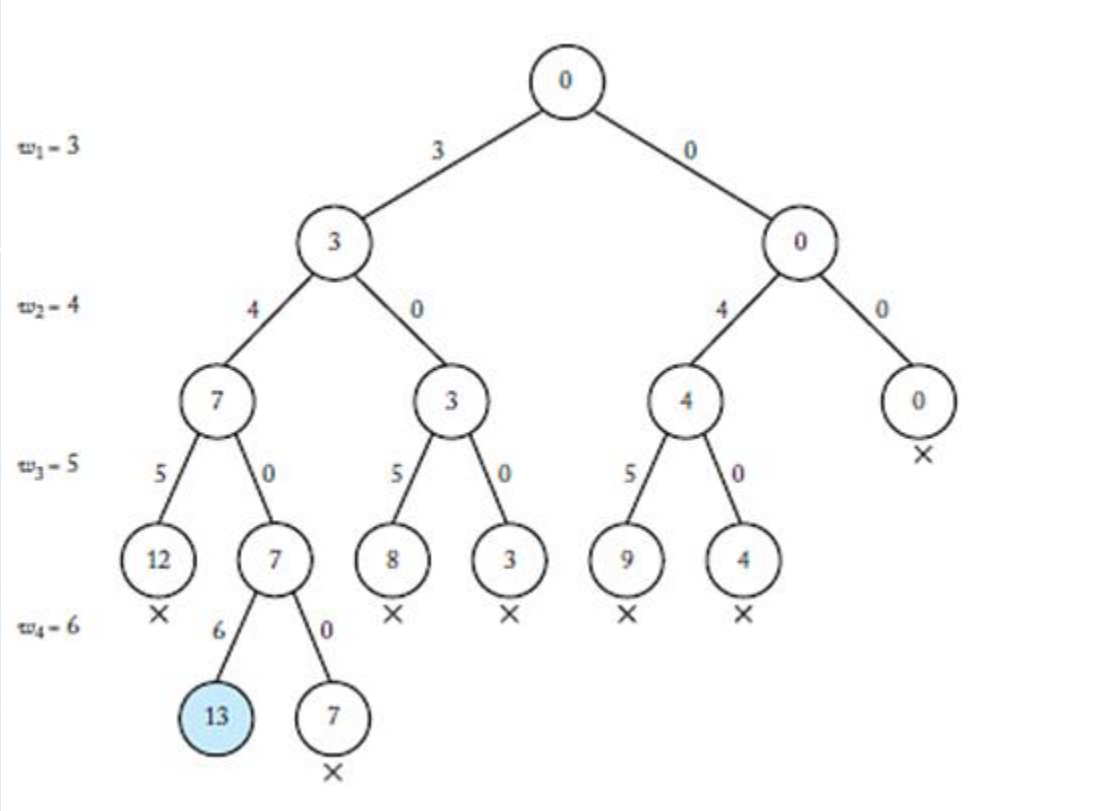
Figure 5.8 state space tree



Prune the Tree

- Sort weights in non-decreasing 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 non-promising if *weight* + $w_{i+1} > W$
- Let *total* be the total weight of the remaining weights at a given node.
  - A node is non-promising if *weight* + *total* < W

Figure 5.9



Algorithm 5.4 Backtracking Algorithm for Sum-of-Subsets

- Total number of nodes in the state space searched by Algorithm 5.4
- $1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$
- Worst case could be better – i.e. for every instance only a small portion of state space tree is search
- Not the case
- For each n, it is possible to construct an instance for which algorithm visits exponentially large number of nodes – even if only seeking one solution

0-1 Knapsack Problem

- Optimization problem
- State space tree similar to Sum-of-Subsets problem
- Best solution so far

Figure 5.14