Module 4

BackTracking Algorithms

Topics

- What is Backtracking
- *N*-Queens Problem
- Sum of Subsets

Backtracking

- Backtracking is a methodical way of trying out various sequences of decisions, until you find one that “works”

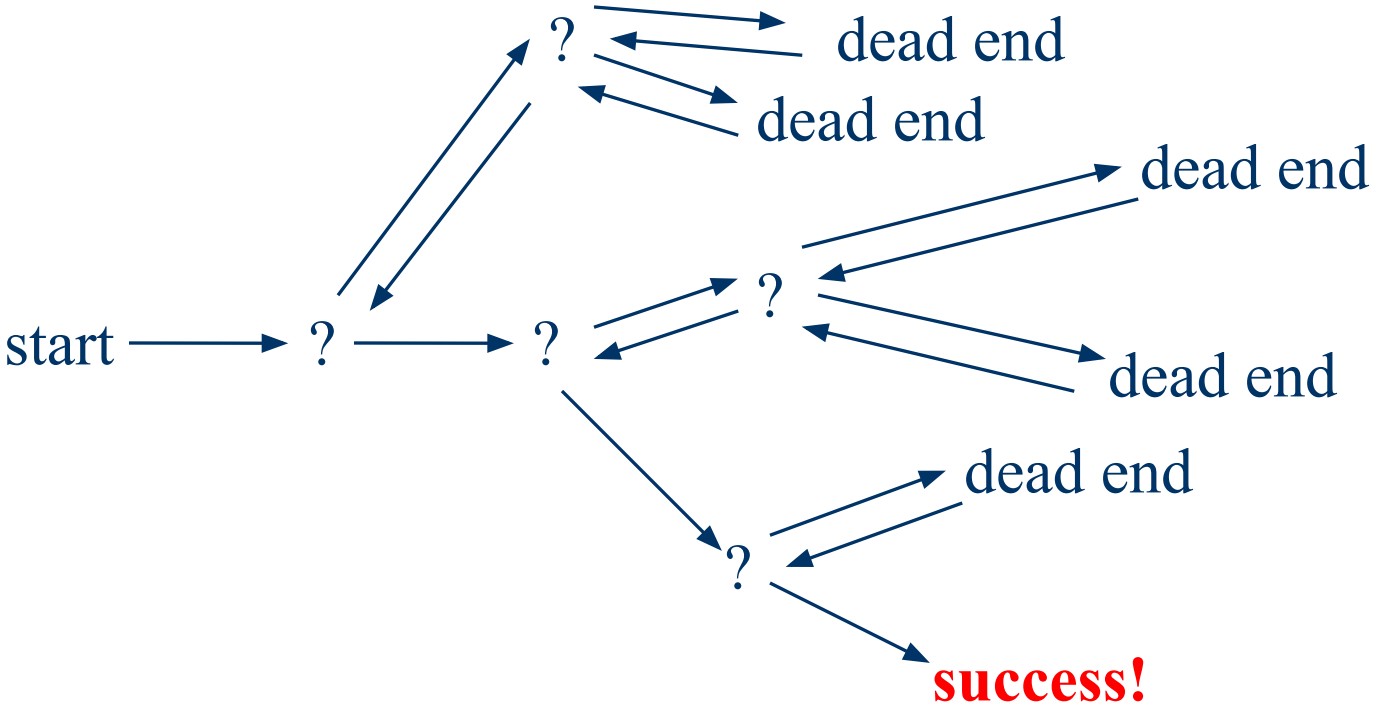
Introduction

- Backtracking is a modified depth-first search of a tree.
- Backtracking involves only a tree search.
- Backtracking is the procedure whereby, after determining that a node can lead to nothing but dead nodes, we go back (“backtrack”) to the node’s parent and proceed with the search on the next child.

Introduction …

- We call a node nonpromising if when visiting the node we determine that it cannot possibly lead to a solution. Otherwise, we call it promising.
- In summary, backtracking consists of
  - Doing a depth-first search of a state space tree,
  - Checking whether each node is promising, and, if it is nonpromising, backtracking to the node's parent.
- This is called pruning the state space tree, and the subtree consisting of the visited nodes is called the pruned state space tree.

Backtracking (animation)



N-Queens Problem

- Try to place N queens on an N * N board such that none of the queens can attack another queen.
- Remember that queens can move horizontally, vertically, or diagonally any distance.

Let's consider the 4 queen example…

N-Queens Problem

```
Algorithm NQueens ( k, n) //Prints all Solution to the n-queens problem
{
    for i := 1 to n do
    {
        if Place (k, i) then
        {
            x[k] := i;
            if ( k = n) then write ( x [1 : n]
            else NQueens ( k+1, n);
        }
    }
}

Algorithm Place (k, i)
{
    for j := 1 to k-1 do
        if (( x[ j ] = // in the same column
           or (Abs( x [ j ] - i) =Abs ( j – k ))) // or in the same diagonal
        then return false;
        return true;
}
```

N-Queens Problem

Monte Carlo simulation

To decide what move to make, simulate thousands of random games that start from the current position, compute for each possible move the percentage of the games that started with that move and that ended up as wins.

Choose the move that had the best winning percentage. This seems like a stupid approach, but if enough games are played, it works reasonably well.

Sum-of-Subsets problem …

Given an array of non-negative integers and an integer sum. We have to tell whether there exists any subset in an array whose sum is equal to the given integer sum.

-Possible tree organizations for the two different formulations of the problem

Variable tuple size formulation

* Edges labeled such that an edge from a level i node to a level i + 1 node represents a value for xi

* Each node partitions the solution space into subsolution spaces

* Solution space is defined by the path from root node to any node in the tree

Sum-of-Subsets problem …

- **Fixed tuple size formulation**

∗ Edges labeled such that an edge from a level i node to a level i + 1 node represents a value for xi which is

either 0 or 1

∗ Solution space is defined by all paths from root node to a leaf node

∗ Left subtree defines all subsets containing w1; right subtree defines all subsets not containing w1

∗ 2n leaf nodes representing all possible tuples

Example

- Say that our weight values are 5, 3, 2, 4, 1
- W is 8
- We could have
  - 5 + 3
  - 5 + 2 + 1
  - 4 + 3 + 1
- We want to find a sequence of values that satisfies the criteria of adding up to W

Tree Space

- Visualize a tree in which the children of the root indicate whether or not value has been picked (left is picked, right is not picked)

- Weight is the sum of the weights that have been included at level i
- Let *weight* be the sum of the weights that have been included up to a node at level i. Then, a node at the ith level is nonpromising if

  weight + $w_{i+1} > W$

Sum-of-Subsets problem …

- Example: Show the pruned state space tree when backtracking is used with n = 4, W = 13, and $w_1 = 3$, $w_2 = 4$, $w_3 = 5$, and $w_4 = 6$. Identify those nonpromising nodes.

```
procedure SUMOFSUB(s, k, r)
 //find all subsets of W(1:n) that sum to M. The values of//
 //X(j), 1 ≤ j < k have already been determined. s = Σ_{j=1}^{k-1} W(j)X(j)//
 //and r = Σ_{j=k}^{n} W(j) The W(j)s are in nondecreasing order.//
 //It is assumed that W(1) ≤ M and Σ_{i=1}^{n} W(i) ≥ M.//
1  global integer M, n; global real W(1:n); global boolean X(1:n)
2  real r, s; integer k, j
   //generate left child. Note that s + W(k) ≤ M because B_{k-1} = true//
3  X(k) ← 1
4  if s + W(k) = M   //subset found//
5       then print (X(j), j ← 1 to k)
        //there is no recursive call here as W(j) > 0, 1 ≤ j ≤ n//
6       else
7         if s + W(k) + W(k + 1) ≤ M then   //B_k = true//
8              call SUMOFSUB(s + W(k), k + 1, r - W(k))
9           endif
10 endif
   //generate right child and evaluate B_k//
11 if s + r - W(k) ≥ M and s + W(k + 1) ≤ M   //B_k = true//
12      then X(k) ← 0
13           call SUMOFSUB(s, k + 1, r - W(k))
14 endif
15  end SUMOFSUB
```

Algorithm 7.6 Recursive backtracking algorithm for sum of subsets problem