Design and Analysis of Algorithms

Backtracking

Reetinder Sidhu

Department of Computer Science and Engineering

Design and Analysis of Algorithms

Backtracking

Reetinder Sidhu

Department of Computer Science and Engineering

- Dynamic Programming
  - Computing a Binomial Coefficient
  - The Knapsack Problem
  - Memory Functions
  - Warshall's and Floyd's Algorithms
  - Optimal Binary Search Trees
- Limitations of Algorithmic Power
  - Lower-Bound Arguments
  - Decision Trees
  - P, NP, and NP-Complete, NP-Hard Problems
- Coping with the Limitations
  - **Backtracking**
  - Branch-and-Bound

Concepts covered

- Backtracking
  - Introduction
  - *N* Queens
  - Hamiltonian Circuit
  - Subset Sum
  - Algorithm

Tackling Difficult Combinatorial Problems

- There are two principal approaches to tackling difficult combinatorial problems (NP-hard problems):
  - Use a strategy that guarantees solving the problem exactly but doesn't guarantee to find a solution in polynomial time
  - Use an approximation algorithm that can find an approximate (sub-optimal) solution in polynomial time

Exact Solution Strategies

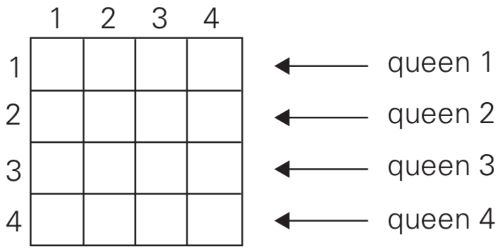
- Exhaustive search (brute force)
  - useful only for small instances
- Dynamic programming
  - applicable to some problems (e.g., the knapsack problem)
- Backtracking
  - eliminates some unnecessary cases from consideration
  - yields solutions in reasonable time for many instances but worst case is still exponential
- Branch-and-bound
  - further refines the backtracking idea for optimization problems

Introduction

- Construct the *state-space tree*
  - nodes: partial solutions
  - edges: choices in extending partial solutions
- Explore the state space tree using depth-first search
- “Prune” *nonpromising nodes*
  - DFS stops exploring subtrees rooted at nodes that cannot lead to a solution and backtracks to such a node’s parent to continue the search

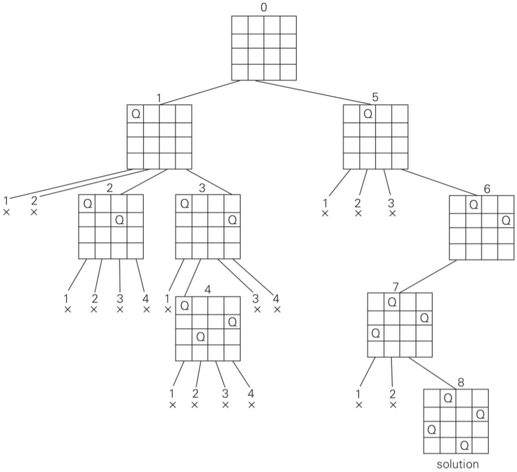
Example: *N*-Queens Problem

- Place N queens on an $N \times N$ chess board so that no two of them are in the same row, column, or diagonal



Backtracking

State-Space Tree of the 4-Queens Problem

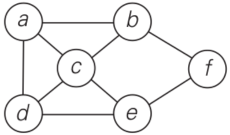


Example: Hamiltonian Circuit Problem

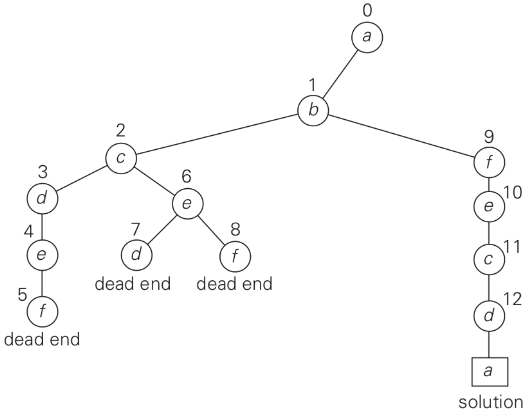
Hamiltonian Circuit

A Hamiltonian circuit is defined as a cycle that passes through all the vertices of the graph exactly once.

- Example graph:



- State-space tree for finding a Hamiltonian circuit (numbers above the nodes of indicate the order in which the nodes are generated):

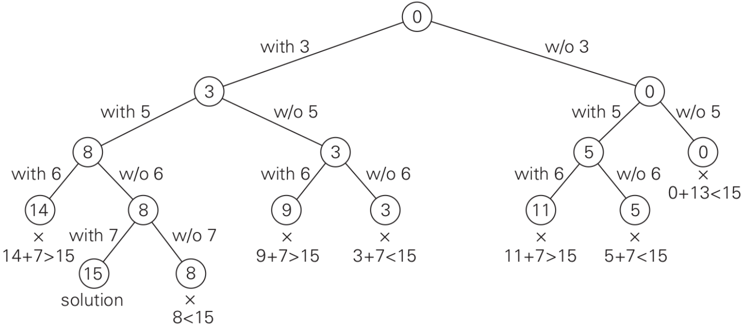


Example: Subset Sum Problem

Subset Sum Problem

Given set $A = \{a_1, \ldots, a_n\}$ of n positive integers, find a subset whose sum is equal to a given positive integer d

- State space tree for $A = \{3, 5, 6, 7\}$ and $d = 15$ (number in each node is the sum so far):



Algorithm

Backtrack Algorithm

```
procedure BACKTRACK(X[1 . . . i])
    ▷ Input: X[1 . . . i] specifies first i promising components of a solution
    ▷ Output: All the tuples representing the problem's solutions
    if X[1 . . . i] is a solution then
        write X[1 . . . i]
    else
        for each element x ∈ S_{i+1} consistent with X[1 . . . i] and the constraints do
            X[i + 1] ← x
            Backtrack (X[1 . . . i + 1])
```

- Output: n-tuples $(x_1, x_2, \dots, x_n)$
- Each $x_i \in S_i$, some finite linearly ordered set

Think About It

- Continue the backtracking search for a solution to the four-queens problem, to find the second solution to the problem
- Explain how the board's symmetry can be used to find the second solution to the four-queens problem