Algorithms 08

CS201

Kaustuv Nag

n Queens Problem

Problem Statement

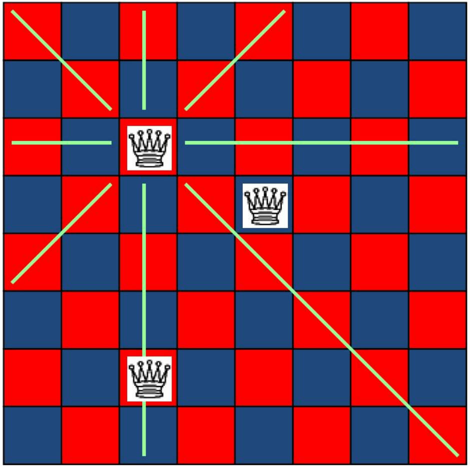
- **Input:** n queens and an $n \times n$ chess board.
- **Input:** A way to place all n queens on the board such that no queens threaten another queen.

Issues to be Checked

- How do queens work?
- How do we formulate this problem?
- How do we represent a solution?

n Queens Problem

How do queens work?



n Queens Problem

First Idea

- Consider every placement of each queen one at a time.

Size of Search Space

- How many placements are there?
  -
$$\binom{n^2}{n} \tag{1}$$
  - when $n = 8$
$$\binom{n^2}{n} = \binom{64}{8} = 4,426,165,368$$

n Queens Problem

Second Idea

- Do not place two queens in the same row.

Size of Search Space

- Now, how many positions need to be checked?
  - we can put one queen at any of the n cells in a row. Therefore, n^n possibilities are there.
  - when $n = 8$

$$n^n = 8^8 = 16,777,216$$

n Queens Problem

Third Idea

- Do not place two queens in the same row or in the same column.

Size of Search Space

- How many placements are there?
  - Generates all permutations of $\{1, 2, \ldots, n\}$. Therefore, $n!$ placements are there.
  - when $n = 8$

$$n! = 8! = 40,320$$

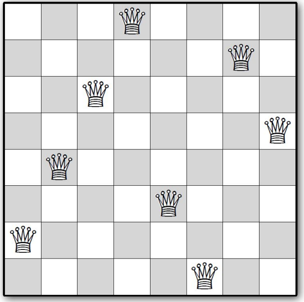
Shrinking the Search Space

- We applied explicit constraints to shrink our search space from $\binom{n^2}{n}$ to n^n to $n!$

n Queens Problem

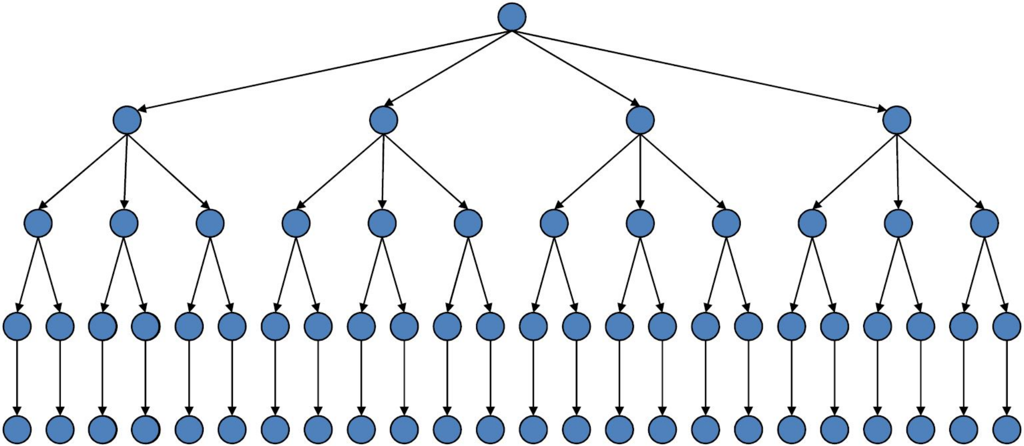
Representation of a Solution

- Let us represent possible solutions to the n-queens problem using an array $Q[1..n]$, where $Q[i]$ indicates which square in row i contains a queen.
- Thus, $Q[1..8] = [4,7,3,8,2,5,1,6]$ represnts the following solution:



Four Queens Problem

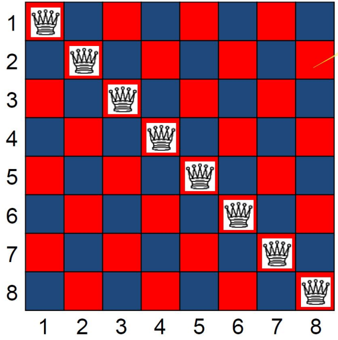
The Complete Recursion Tree Without Backtracking



n Queens Problem

Observation

- Two queens are placed at positions (i,j) and (k,l). They are on the same diagonal only if $i-j=k-l$ or $i+j=k+l$.



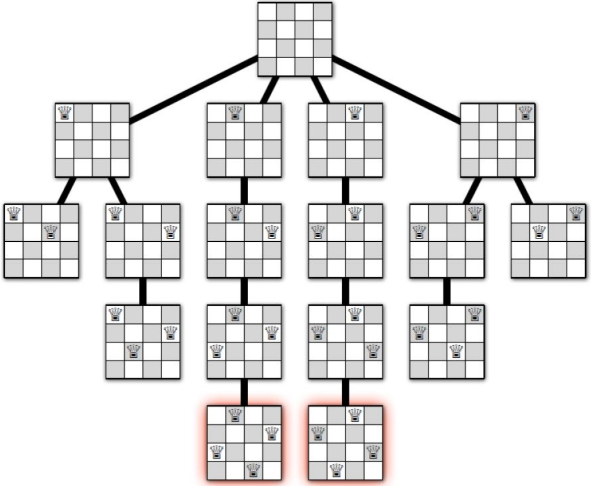
n Queens Problem

Backtracking Algorithm

```
RecursiveNQueens(Q[1..n], r):
if r == n + 1
    print Q
else
    for j = 1 to n
        legal = True
        for i = 1 to r - 1
            if (Q[i] == j) or (Q[i] == j + r - i) or (Q[i] == j - r + i)
                legal = False
        if legal
            Q[r] = j
            RecursiveNQueens(Q[1..n], r + 1)
```

Four Queens Problem

The Complete Recursion Tree With Backtracking



Backtracking

When?

- Suppose we have to make a series of decisions, among various choices, where
  - We do not have enough information to know what to choose.
  - Each decision leads to a new set of choices.
  - Some sequence of choices (possibly more than one) may be a solution to our problem.

What?

- Backtracking is
  - A general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems.
  - Incrementally builds candidates to the solutions.
  - Abandons a candidate (“backtracks”) as soon as it determines that the candidate cannot possibly be completed to a valid solution.

Representation of a Solution and Solution Space

- We can represent the solution space for the problem using a state space tree.
  - A path from a root to a leaf represents a candidate solution.

Subset Sum Problem

Problem Statement

- **Input:** A set X of positive integers and target integer T.
- **Input:** Is there a subset of elements in X that add up to T?

Examples

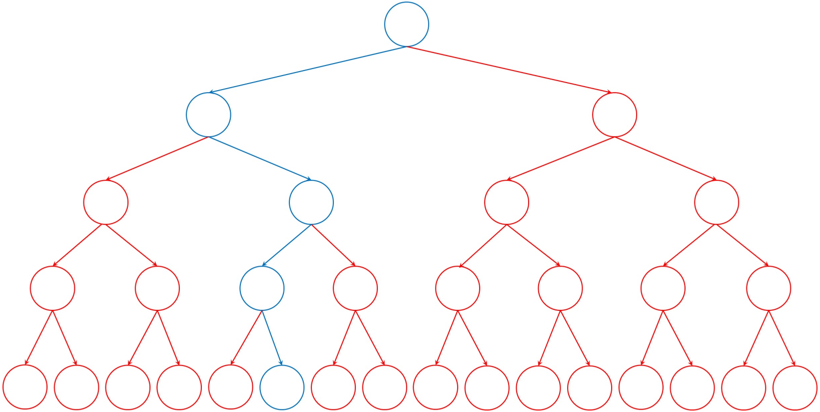
- If $X = \{8, 6, 7, 5, 3, 10, 9\}$ and $T = 15$, the answer is *true* as $T = 8+7$, $T = 7+5+3$, $T = 6+9$, and $T = 5+10$.
- if $X = \{11, 6, 5, 1, 7, 13, 12\}$ and $T = 15$, the answer is *false*.

Observations

- Considering an arbitrary element $x \in X$, there is a subset of X that sums to T if and only if one of the following statements is true:
  - There is a subset of X that includes $x \in X$ and whose sum is T.
  - There is a subset of X that excludes $x \in X$ and whose sum is T.

Subset Sum Problem

Binary State Space Tree



Subset Sum Problem

Backtracking Algorithm

```
SubsetSum(X, T):
if T == 0
    return True
else if T < 0 or X is empty
    return False
else
    x = any element of X
    return SubsetSum(X \ {x}, T) OR SubsetSum(X \ {x}, T - x)
```

0/1 Knapsack Problem

- Consider a collection of N indivisible objects, labelled by the integers $i = 1, 2, \ldots, N$.
- The ith object is associated a positive real number w_i, the "weight" of the object, and real number v_i, the "value" of the object.
- Need to form a loading of the objects by selecting from among the N objects a subcollection which has a maximum total value but which does not exceed a total weight of say W units.
- Mathematically: find $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T \in \{0, 1\}^N$, such that

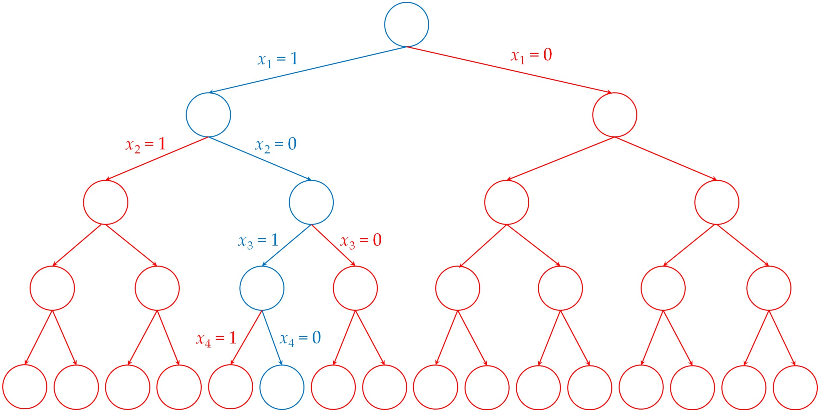
$$\textit{maximize} \left\{ \sum_{i=1}^{N} x_i v_i \right\}$$

$$\text{such that} \sum_{i=1}^{N} x_i w_i \leq W$$

0/1 Knapsack Problem

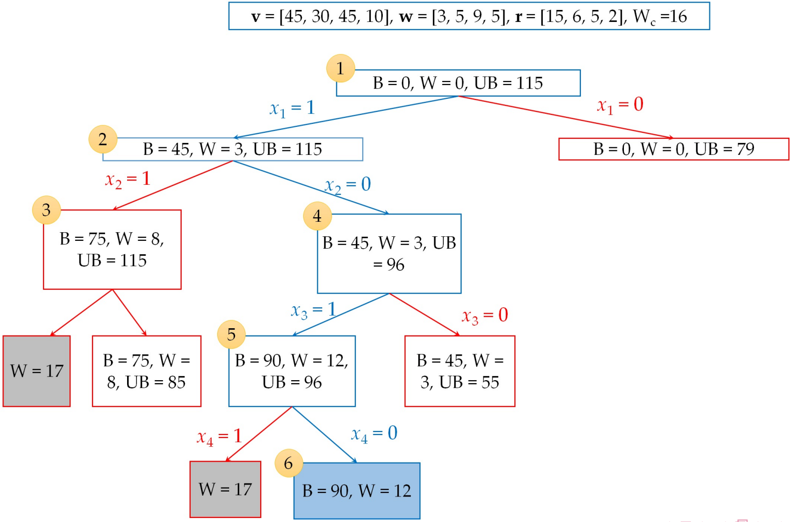
Example: Complete Recursion Tree (State Space Diagram)

- $\mathbf{v} = (45, 30, 45, 10)$, $\mathbf{w} = (3, 5, 9, 5)$, $W = 16$



0/1 Knapsack Problem

Example



Branch and Bound

- Can be used to solve optimization problems especially in discrete and combinatorial optimization without an exhaustive search in the average case.
- It is efficient in the average case because many branches can be terminated very early.
- A very large tree may be generated in the worst case.

White Board

White Board