Lecture 11.1

- Anurag Sharma & Shymal Chandra

Backtracking Algorithms

Backtracking Technique

- Backtracking is used to solve problems in which a sequence of objects is selected from a specified set so that the sequence satisfies some criterion
- Often the goal is to find any feasible solution rather than an optimal solution – example, when solving a maze that could have many possible solutions
- Backtracking is a modified **depth-first search** of a state-space tree
- What is depth-first search? A preorder traversal of a tree!

Cont.

- A **state space tree** of a problem is a tree that contains nodes indicating the object chosen or the direction chosen. A path from the root of the tree to a leaf (node with no children) is a candidate solution
- Backtracking is a procedure whereby, after determining a node can lead to nothing but dead ends, we go back (backtrack) to parent node and search on the next child
- A node is **nonpromising,** if it is determined that it cannot possibly lead to a solution and promising otherwise

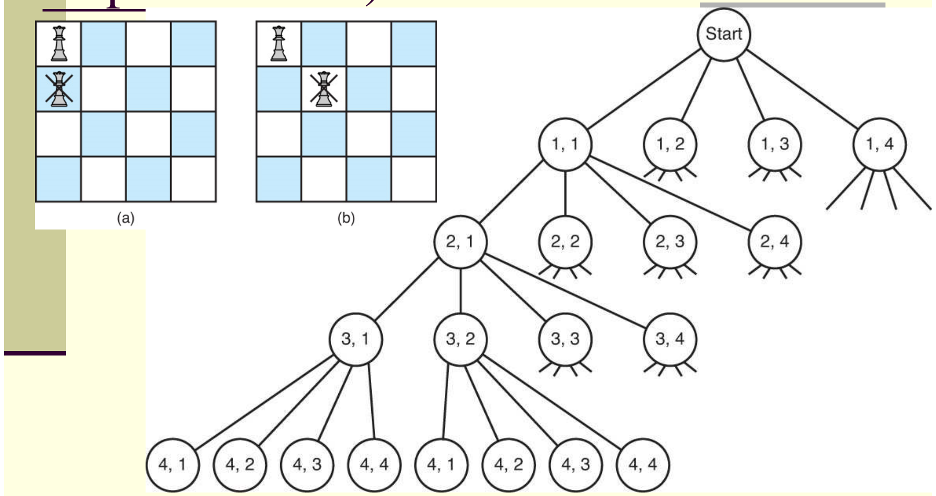
Cont.

- **Pruning** a state space tree is doing a depth-first search and checking whether each node is promising or not; if not promising then backtrack to parent node
- Pruning helps shorten the entire state space tree
- The subtree consisting of the visited nodes is called **pruned state space tree**

Example (The n-queen problem)

- The idea in the n-Queens problem is to place n queens on an n x n chess board, such that none of the queens can attack another queen
- Remember that queens can move horizontally, vertically, or diagonally any distance
- We will illustrate backtracking using n = 4 i.e. placing 4 Queens on a 4 x 4 chess board such that no queen can attack any other

4-queen problem (4x4x4x4=256 possibilities)



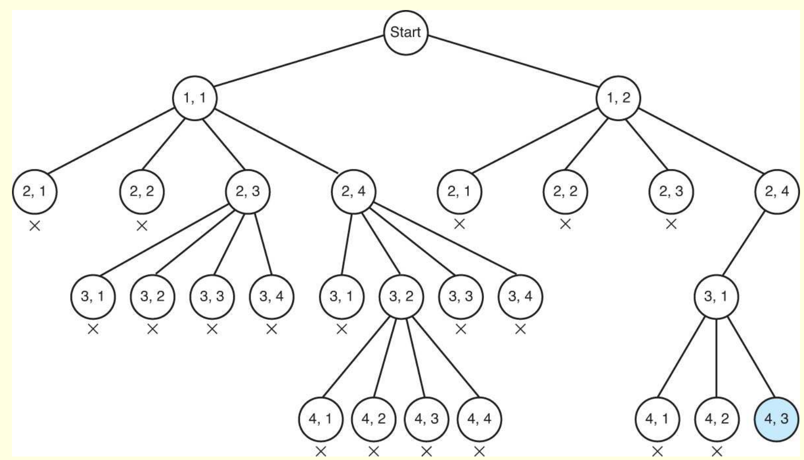
Use backtracking approach

Backtracking is the procedure whereby, after determining that a node can lead to nothing but dead ends, we go back ("backtrack") to the node's parent and proceed with the search on the next child. 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***. To summarize, 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***. A general algorithm for the backtracking approach is as follows:

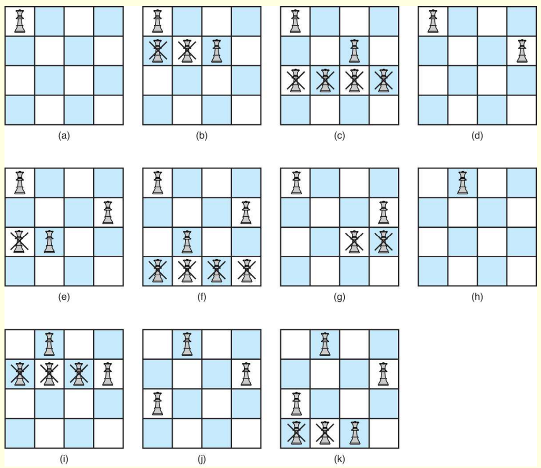
```
void checknode (node v)
{
  node u;

  if (promising(v))
     if (there is a solution at v)
        write the solution;
     else
        for (each child u of v)
           checknode(u);
}
```


4-queen problem (promising solutions)



Solution for 4-queen problem



Efficiency of backtracking

- N-queen problem with backtracking has efficiency of O(n!).

• **Table 5.1** An illustration of how much checking is saved by backtracking in the n-Queens problem *

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}

Any better solution? [optional]

- https://link.springer.com/chapter/10.1007/978-3-642-35101-3_21

Table 1. Comparative test results on no problem specific information extraction

N	CMA-ES [25]	DE [25]	GA	NSGA II	ICHEA
4	456 NFC (SR = 1.00)	134 NFC (SR = 1.00)	367 NFC (SR = 1.00)	93 NFC (SR= 1.00)	39 NFC (SR = 1.00)
5	656 NFC (SR = 1.00)	254 NFC (SR = 1.00)	750 NFC (SR = 1.00)	217 NFC (SR = 1.00)	37 NFC (SR = 1.00)
6	22,013 NFC (SR = 1.00)	1,11,136 NFC (SR = 0.65)	30,086 NFC (SR = 0.75)	694 NFC (SR = 1.00)	51 NFC (SR = 1.00)
7	9,964 NFC (SR = 1.00)	24,338 NFC (SR = 0.95)	1,400 NFC (SR = 1.00)	2631 NFC (SR = 1.00)	34 NFC (SR = 1.00)
8	84,962 NFC (SR = 1.00)	7,576 NFC (SR = 0.75)	3,786 NFC (SR = 0.80)	1273 NFC (SR = 1.00)	41 NFC (SR = 1.00)
9	133,628 NFC (SR = 1.00)	19,296 NFC (SR = 0.50)	18,333 NFC (SR = 0.80)	27,852 NFC (SR = 1.00)	72 NFC (SR = 1.00)
10	263,572 NFC (SR = 0.95)	286,208 NFC (SR = 0.30)	3,300 NFC (SR = 0.30)	1,737 NFC (SR = 1.00)	83 NFC (SR = 1.00)
11	284,382 NFC (SR = 0.95)	68,255 NFC (SR = 0.10)	15,550 NFC (SR = 0.40)	SR = 0.00	132 NFC (SR = 1.00)
12	295,740 NFC (SR = 0.75)	99,120 NFC (SR = 0.25)	23,000 NFC (SR = 0.70)	SR = 0.00	122 NFC (SR = 1.00)
13	376,631 NFC (SR = 0.85)	95,485 NFC (SR = 0.15)	3,400 NFC (SR = 0.10)	SR = 0.00	293 NFC (SR = 1.00)
14	450,654 NFC (SR = 0.85)	160,475 NFC (SR = 0.10)	47,350 NFC (SR = 0.40)	SR = 0.00	308 NFC (SR = 1.00)
15	627,391 NFC (SR = 0.50)	223,425 NFC (SR = 0.10)	95,625 NFC (SR = 0.40)	SR = 0.00	381 NFC (SR = 1.00)

Hamiltonian Circuits Problem (optional)

- A Hamiltonian circuit (tour) of a graph is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex. The problem is to find all the Hamiltonian circuits in a graph
- A state space tree for this problem is as follows: Put the starting vertex at level 0 in the tree; the zeroth vertex on the path. At level 1, create a child node for the root node for each remaining vertex that is adjacent to the first vertex. At each node in level 2, create a child node for each of the adjacent vertices that are not in the path from the root to this vertex, and so on…

Cont.

- In order to backtrack in this state space tree: The i^{th} vertex on the path must be adjacent to the $(i - 1)^{st}$ vertex on the path
- The $(n - 1)^{st}$ vertex must be adjacent to the 0th vertex
- The i^{th} vertex cannot be one of the first $i - 1$ vertices