CS207 Design and Analysis of Algorithms

Sajith Gopalan

Indian Institute of Technology Guwahati

sajith@iitg.ac.in

March 22, 2022

Backtracking

Backtracking

- Useful for constraint satisfaction problems (CSPs)
- (A CSP has a set of variables, each with a domain of values, and a set of constaints on these variables specified through relations on them. We need to find values for all variables, so that all constraints are satisfied. E.g., Sudoku)
- Incrementally builds candidates to the solutions
- Abandons a candidate ("backtracks") as soon as it becomes clear that the candidate cannot be completed into a valid solution

The n-Queens Problem: Algorithm

- The recursive algorithm "nQueen(k, n)" given in the next slide assumes that $Q_1, \ldots, Q_{k-1}$ are already placed in columns $col[1], \ldots, col[k-1]$ respectively, and prints out all solutions given these placements
- Invocation "nQueen$(1, n)$" prints out all solutions to the n-Queens problem
- Valid(k, i) verifies whether placing Q_k in the i-th column is valid given the current placements of $Q_1, \ldots, Q_{k-1}$

The n-Queens Problem: Algorithm

Algorithm 1 nQueen(k, n)

```
1:  for i = 1 to n do
2:      if Valid(k, i) then
3:          col[k] = i;
4:          if k = n then
5:              print (col[1], ..., col[n]);
6:          else
7:              nQueen(k + 1, n)
8:          end if
9:      end if
10: end for
11: return
```

Validating a placement: Algorithm

Algorithm 2 Valid(k, i)

```
1: for j = 1 to k − 1 do
2:     if |col[j] = i| OR |col[j] − 1| = |j − k| then
3:         return false
4:     end if
5: end for
6: return true
```

Example (Compare with the previous lecture)

nQueen(1,4)
  (1)
  nQueen(2,4)
    (1,3)
    nQueen(3,4)
    (1,4)
    nQueen(3,4)
      (1,4,2)
      nQueen(4,4)
  (2)
  nQueen(2,4)
etc.

The tree perspective

Every permutation of $\{1, \ldots, n\}$ is a candidate solution. But most of the candidate solutions are invalid; as you have seen, for most of them, invalidity can be proved by examining a short prefix.

Visualize a tree. Let the root of the tree correspond to the set of all $n!$ permutations. Let the leftmost child of the root correspond to (1), or in other words to the $(n-1)!$ permutations in which 1 appears in the first place. The second leftmost child of the root correspond to (2), and so on. When we examine a partial candidate solution, we are considering a path in the tree with root as one of its end points. For example, when examining $(1, 4, 2)$, we consider the path ()-(1)-(1, 4)-(1, 4, 2).

The tree perspective

The algorithms attempts a Depth First Search of this tree. At each internal node, it examines whether going further deep from this node might give a full solution. When this is ruled out, the algorithm prunes the subtree rooted at that node without searching it, and turns back from that node. Because of this, the actual number of nodes visited would be a lot less than $n!$