Backtracking

COMP 215

Search Trees

- The next set of problems we will be looking at require search.
- A sequence of choices must be made – the sequence must satisfy some criterion, or maximize some utility function.
- Search is visualized as a search tree.
- Brute force approaches (depth first search, breadth first search) explore the entire tree.
- Let's look at *N*-Queens...

N-Queens

- The problem: position *N* queens on an *N*x*N* chess board, so that no two queens threaten each other.
- Observation: each row must contain exactly one queen.
- The brute force algorithm...

N-Queens Brute Force

```
//i = current row
//n = number of queens
//col[i]=position of queen in row i.

void queens (index i, int n, index[] col){
   if (i == n && noThreats()) {
      cout << col[1] through col[n]; //print solution
   } else {
      for (j = 1; j < n; j++) {
         col[i+1] = j;
         queens(i+1, n, col);
      }
   }
}
```

- Analysis: How many nodes in this search tree?

Backtracking

- Don't expand the search tree at a node if that node can't lead to a solution.
- Generic backtracking pseudocode:

```
void checknode (node v){
   if (promising(v)) {
      if (solution at v) {
         print solution;
      } else {
         for(each child u of v) {
            checknode(u);
         }
      }
   }
}
```

N-Queens W/Backtracking

```
//i = current row
//n = number of queens
//col[i]=position of queen in row i.

void queensBT (index i, int n, index[] col){
   if (noThreatsBT(col, i)) {
      if (i == n) {
         cout << col[1] through col[n];
      }
   } else {
      for (j = 1; j < n; j++) {
         col[i+1] = j;
         queensBT(i+1, n, col);
      }
   }
}
```

Checking for Threats

```
boolean noThreatsBT (index[] col, index i){
   boolean threat = false;
   for (int j = 1; j < i; j++) {
      if (col[i] == col[j])
         threat = true;
      if (abs(col[i] – col[j]) == i – j)
         threat = true;
   }
}
```

Analysis

- How much more efficient is the backtracking algorithm?
- Hard to say.
- Bound on nodes in search tree is: $1+n+n^2+...+n^n=\frac{n^{n+1}-1}{n-1}$
- If we take into account the fact that each column can have at most one queen, we can reduce that to:
  - $1 + (1 * n) + (1 * n * (n\text{-}1)) + (1 * n * (n\text{-}1) * (n\text{-}2)) + ... + n!$
- The book suggests a Monte-Carlo approach to analyzing backtracking algorithms.