Search Problems

David Cline
Things to think about

• Characteristics of the world
  • Deterministic vs. non-deterministic
  • Observable vs. partially observable
  • Explorable vs. non-explorable
  • State space
    • How to represent individual states?
    • How big is the state space?
Well-Defined Problems

• Initial State
  • beginning state of the world
• Transition Model
  • When action taken, how does world change?
  • What actions are applicable in a state?
• Goal Test
  • Is the world in a goal state?
• Path Cost
  • What is the cost of the sequence of actions?
  • Optimal solution finds best cost path
Algorithm Characteristics

• Completeness
  • Will algorithm always find a solution?

• Optimality
  • Will algorithm find the best solution?

• Time and space complexity (Big-O)
Example: Rod Cutting

• Given a rod of length n centimeters and a table of prices for different rod lengths, cut the rod so that it maximizes overall price.

• In particular, find the maximum profit that can be made from a rod of length n

• (Q) For this problem, define
  • initial state
  • transition model
  • goal test
  • path cost
Basic Recursive Implementation

```c
int cutRod_bruteForce(Prices p, int n)
    if n==0
        return 0
    q = -infinity
    for i=1 to n
        q = max(q, p[i] + cutRod_bruteForce(p, n-i))
    return q

// First Call:
maxProfit = cutRod_bruteForce(prices, rodLength)
```

• (Q) How big is the state space (how many size sets)?
• (Q) How often is each state made during the search?
Increasing Cut Sizes

• Optimize by making each cut at least as long as the last. Each set of sizes is only generated once.

```c
int cutRod_increasingCut(Prices p, int n, int minCut)
    if n<minCut
        return 0
    q = -infinity
    for i=minCut to n
        q = max(q, p[i] + cutRod_increasingCut(p,n-i,i));
    return q

// First Call:
maxProfit = cutRod_increasingCut(prices, rodLength)
```

• (Q) Do we have to generate every single state?
Backtracking

- Use a loose bound on the best possible solution down a given branch of the recursion tree:

```c
int maxProfitPerUnit = max(p[i]/i)
int bestSoFar = -infinity

int cutRod_backtrack(Prices p, int n, int minCut, int qIn)
    if n<minCut
        return 0
    if bestSoFar > (qIn + n*maxProfitPerUnit) // backtrack
        return 0
    q = -infinity
    for i=minCut to n
        q = max(q, p[i] + cutRod_backtrack(p, n-i, i, qIn+p[i]));
    return q

// First Call
maxProfit = cutRod_backtrack(prices, rodLength, 1, 0)
```
What about partial solutions?

• Backtracking proceeds as follows
  • start with length n
  • cut off a piece of length k
  • recursively generate every possible set of sizes of length n-k, and choose the best one.

(Q) What if we already knew the best set for length n-k?
Memoized

• Keep track of calls already made in a table, and use table whenever possible. This is called “memoization”

```java
int[] memos = {-1, -1,...}

int cutRod_memoized(Prices p, int n)
    if n==0
        return 0
    if memos[n] >= 0
        return memos[n]
    for i=1 to n
        q = max(q, p[i] + cutRod_memoized(p, n-i))
    memos[n] = q
    return q

// First Call:
maxProfit = cutRod_memoized(prices, rodLength)
```
Bottom Up

• We can specifically build subproblems from small to large, so that we always have table values:

```java
int cutRod_bottomUp(Prices p, int n)
    int[] memos = new int[n+1];
    memos[0] = 0
    for j = 1 to n
        q = -infinity
        for i = 1 to j
            q = max(q, p[i] + memos[j-i])
        memos[j] = q
    return memos[n]
```
N-Queens

• Given an N×N chess board
  • Place N queens on the board so that they cannot attack each other

(Q) What are?
• Initial state?
• Transition model?
• Goal test?
• Path cost?
Algorithm 1

• Each placement looks over whole board

• (Q) How many possibilities for the first placement?
• (Q) How many possibilities for N placements?
• (Q) Can we do better?
Algorithm 2

• Placements put on successive rows

```java
boolean PlaceQueens2(Board B, int y)
{
    for (x=0; x<N; x++) {
        if (no placed queen can attack B[y][j]) {
            B[y][x] = queen;
            if (PlaceQueens2(B)) return true;
            B[y][x] = blank;
        }
    }
    return false;
}
```

• (Q) What's missing from the code above?
• (Q) Trace algorithm for 4x4 case
Sudoku

• 9x9 grid
• Some cells filled in
• Fill in other cells so that
  • 1-9 on rows
  • 1-9 on columns
  • 1-9 on subgrids
Sudoku

• (Q) Write pseudocode for simple Sudoku solver
• (Q) Can we do better?