Algorithm Design Patterns

EECS 281 Handout

Algorithmic Design Techniques (Patterns)
Each design pattern describes a solution strategy:
- Brute Force
- Greedy
- Divide-and-conquer (top-down strategy)
- Dynamic programming (bottom-up strategy)
  - Memoization (top-down strategy)
- Backtracking (branch-and-bound strategy)

More than one way of solving a given problem. Choose carefully considering the cost/complexity!

Brief Introduction
- Brute Force:
  - Enumerate all possible solutions and pick the best one
  - May be very expensive to evaluate
  - But is easy to implement!
  - In some cases, may be the only solution
- Greedy:
  - Selectively explore the “solution space”
  - Start from a well-defined starting condition
  - Iteratively make additional choices by identifying the decision that achieves the best cost from all of the current possible choices (i.e. local optimum)
  - Not guaranteed to pick the optimal solution; but with smart heuristics, can be often close
  - For some problems, greedy produces optimal answer; i.e. global optimal can be reached by a series of locally optimal choices
  - Examples: Dijkstra’s shortest path, Kruskal’s MST, Prim’s MST, Huffman’s coding algorithm (text compression), Boyer-Moore’s pattern matching
Divide-and-Conquer: (Top-Down Strategy)

- Divide: problem into subproblems, recursively solve each subproblem (except, of course, base case)
- Conquer: merge solutions to subproblems into a solution to the original problem
- Hint: typically the main routine contains two recursive calls
- Be aware of inefficient applications of this strategy: e.g. recursive implementation of Fibonacci numbers
- Examples: quicksort, mergesort
- Is binary search a divide-and-conquer strategy?

Dynamic Programming: (Bottom-Up Strategy)

- Combine solutions to subproblems bottom-up --- avoid unnecessary duplication of effort
- Practical way of thinking about this strategy: Re-write a recursive algorithm as a non-recursive algorithm that systematically records the answers to subproblems in a table
- An optimal solution to the global problem must be a composition of optimal subproblem solutions
- Often an alternative to an exponential brute-force approach
- Example: Floyd’s all-pairs shortest path algorithm
- Examples (to be discussed): computing Fibonacci numbers, computing binomial coefficient, similarity testing for text (longest common subsequence)

Memoization: (Top-Down Strategy)

- Variation of dynamic programming with a top-down strategy
- IDEA: memoize the natural, but inefficient, recursive algorithm
- Maintain a table with subproblem solutions, but implement a control structure for filing the table is more like a recursive algorithm
- When a subproblem is subsequently encountered, the value stored in the table is used
- A little more overhead than bottom-up DP for recursion and maintaining a table
- Note: An added advantage of solving only those subproblems that are definitely required
Backtracking:

- View all possible decisions as exploring a decision tree
- Brute-force: Explore the entire tree
- Backtracking: Do the same, but if you detect that the remaining search will not produce an optimal (or desirable) solution, then abort, i.e. backtrack before exploring any further.
- Elimination of a large number of possibilities is called pruning; often we use heuristics to prune a subtree
- Exploring the search tree can be done
  - Depth-first search
  - Breadth-first search
- Example: arranging furniture in a room!