CSC721 Algorithms Fall 2017

Algorithm Design Strategies

1. Divide and conquer / Balancing

   - Usually applicable when;
     - A problem can be split into sub-problems which are smaller instances of the same type of problem as the original, and
     - the solutions to the sub-problems can be re-assembled into a solution to the original problem.
   - Example: mergesort
   - Failed example: minimal spanning tree
   - Implementation: usually recursive, otherwise requires explicitly managing a stack of postponed sub-problems.
   - Analysis: usually done using recurrence equations
   - Balancing: more efficient divide-and-conquer methods are obtained when the sizes of the sub-problems are (approximately) equal.
   - Example: unbalanced mergesort

2. Greedy algorithms

   - Usually applicable when;
     - By examining local information, an action can be chosen which makes certain progress towards a solution.
     - A mathematical proof is often required to ensure correctness.
   - Example: Kruskal's algorithm (minimum spanning tree)
   - Failed example: 0-1 Knapsack problem
   - Implementation: often iterative
   - Analysis: usually by simply counting operations

3. Backtracking

   - Usually applicable when;
     - Local information can be considered, but we have no set of rules which enable us to choose an action that is certain to make progress towards a solution. We must guess at the correct choice of actions, and our guess may be incorrect.
   - When a series of guesses fails to lead to a solution, we must systematically reverse the affects of our previous guesses and explore new guesses, potentially traversing the entire search space.
   - Computationally expensive. avoid backtracking unless no viable alternative is available
   - Implementation: recursive or stack-based
   - Analysis: usually using recurrence equations
   - Example: 0-1 knapsack problem

4. Dynamic Programming

   - Usually applicable when;
     - A problem can be split into sub-problems which are smaller instances of the same type of problem as the original. (similar to divide-and-conquer).
     - The solutions to the sub-problems can be used to find a solution to the original problem, but it is often not known exactly which sub-problems need to be solved.
     - There must be a simple method to index the sub-problems using a few (non-negative and small) integer parameters so that solutions to sub-problems can be contained in a table.
     - The sub-problems must overlap to achieve any gains in efficiency.
   - Key idea: Solutions to sub-problems are stored in a table. Whenever a solution to a sub-problem is needed, the table is consulted to check if the needed solution is already available. I.e., If the same sub-problem arises more than once, its solution is only computed once; the solution is stored in a table and may be reused as often as needed.
   - When applicable, dynamic programming can be miraculously effective. It enables polynomial time solutions in cases where obvious algorithms would take exponential time.
   - Implementation: either recursive (on-demand style) or iterative (pre-ordered style), depending on condition (4).
   - In some cases, the order of computations can be arranged so that the solution to each sub-problem is computed and stored in a table prior to needing the solution to that sub-problem.
   - Example: optimal order for matrix multiplication

5. Monte Carlo Algorithms

   - Usually applicable when;
     - A desired solution can be described as the statistical average of some random process.
   - Monte Carlo takes a fixed amount of time and space, but gambles with the accuracy of the solution found.
   - Example: Programming contest problem "Selling Cells"
   - Analysis: based on probabilistic reasoning

6. Las Vegas Algorithms

   - Usually applicable when;
     - A random choice (or several random choices) at some step can be used to avoid the worst-case performance of a deterministic algorithm.
   - Las Vegas gets the correct solution, but gambles with the amount of resources (time or storage space) needed.
   - Example: Randomized quicksort
   - Analysis: based on probabilistic reasoning