1 General problem solving methods

- Brute force generate and test
  - Generate all potential solutions and test for which ones are actual solutions
  - Example: We could do sorting by generating all possible permutations and to see which one is correctly ordered (extremely inefficient)
- Divide-and-conquer
  - Recursively, break the problem into smaller pieces, solve them, and put them back together
  - Merge-sort and Quicksort were classic examples of this
- Heuristics (rule of thumb)
- Dynamic Programming

1.1 Heuristics

The objective of a heuristic is to produce a solution in a **reasonable time frame** that is **good enough** for solving the problem at hand. This solution **may not be the best** of all the solutions to this problem, or it may simply **approximate the exact solution**. But it is still valuable because finding it **does not require a prohibitively long time**. Heuristics may produce results by themselves, or they may be used in conjunction with optimization algorithms to improve their efficiency

- Generally, meant to mean something that gives better decisions, than the nave methods, but still not necessarily optimal.
- Two common types (the term is over-loaded)
- Decisions within a procedure that gives optimal answers, but are designed to make it go **faster**
- Decisions within a procedure that might not give optimal answers, but are designed to give good answers that are impractical to obtain otherwise
- Examples: A star, pivot finding in quicksort
- Used on problems when the exact methods are too slow, e.g.
- Timetabling and scheduling, genetic algorithms, metaheuristics (simulated annealing, tabu search, etc, etc), approximate greedy methods.

1.1.1 Greedy algorithms

A common heuristic is to be **greedy**. Take the decision that looks best in the short term, Prims algorithm for constructing a Minimal Spanning Tree is a **greedy algorithm**: it just adds the shortest edge without worrying about the overall structure, without looking ahead. It makes a **locally optimal** choice at each step.

1.1.2 Greedy algorithm: optimal

Sometimes greedy algorithms can still give optimal answers. E.g. Prims algorithm for constructing a Minimal Spanning Tree is a greedy algorithm: It just adds the shortest edge without worrying about the **overall structure**, **without looking ahead**. It makes a locally optimal choice at each step. But it turns out that this is sufficient for the final answer to be optimal

1.2 Dynamic programming (DP)

DP is a general method that can be suitable when the optimal solutions satisfy a *decomposition property*, The general idea is roughly:

- **Splitting** an optimal solution into sub-solutions corresponds to splitting the problem into sub-problems and the sub-solutions are optimal for the sub-problems
- So optimal solutions can be built out of optimal solutions of (smaller) sub-problems
- So solve small sub-problems first and build up towards the full solution
- Generally split up a problem to it's smallest parts, and keep solving from bottom down.

Reference section

placeholder