|  |  |
| --- | --- |
| **Mathematical shorthands**  Given values find a set S of such that | |
| **Greedy** | |
| **Inductive proof concepts**: Show that greedy is optimal for a set of sub-cases, show that large problems can be broken down into those sub-cases. Often, proof is done by contradiction (suppose optimal < greedy …) | |
| **Dijkstra’s algo**: Initialize set of explored nodes and array storing shortest path cost to those nodes . Repeatedly choose an unexplored node where , add to and set | |
| **Cashier’s algo:** Proveoptimality for sub-cases, then extend to larger cases | |
| **Huffman encoding**: Understand bottom-up construction of tree starting with the least occurring nodes, combining as you go. Repeat until all nodes have been combined under a single tree | |
| **Divide & conquer** | |
| **Master’s Theorem**: Applies to recurrences of the form , where are constants and is an asymptotically growing function (Note: Unless otherwise stated, log is base 2) 1. If for some , then  2. If with , then – note that  3. If for some , and then | |
| **Median of medians**: Recursive calling between 2 functions – *SELECT()* and *CHOOSEPIVOT(). SELECT()* performs sorting and selection if array is below a certain size. Otherwise, it calls *CHOOSEPIVOT()*, which splits the array into groups and chooses the median out of each of them. Overall complexity is | |
| **Dynamic programming**  **Iterative approach**: Usually involves 2D arrays and using solutions to smaller problems to solve larger ones  **Recursive approach**: Backtrack with memorization  Both involve identifying an optimal substructure and coming up with a recursive formulation | |
| **Knapsack problems**: 0-1, unbounded | |
| **Flow** | |
| **Ford-fulkerson (greedy)**: Given graph and flow , residual graph has the same nodes as G. For any edge , include edge with capacity and with capacity if . Continuously try to push more flow from to in residual graph until is unreachable from  Complexity is where =optimal flow value. DFS is but in a connected graph each iteration improves flow by at least 1. | |
| **Dummy nodes** can be introduced for cases of multiple sources & sinks. Applicable to problems like bipartite matching and max circulation | |
| **Edmonds-karp**: Uses BFS to find augmenting path, runs in | |
| **Max-flow, min-cut**: Max flow for a graph is equivalent to min-cut | |
| **Computational limits** | |
| **Reduction**: We can reduce problem X to problem Y by doing a transformation from X to Y, and using a solver for Y to find a solution, then transforming the solution back to a version in X.  - X reduces to Y is denoted as (transitive  **Optimization**: Minimize or maximize certain metrics **Search**: Answer a question with evidence **Decision**: Give a boolean answer to a question  *Optimization>Search>Decision*  **P problems** are solvable in polynomial time  **NP problems** have solutions that can be *verifiable* in polynomial time *(but we cannot verify the absence of a solution in polynomial time)* **NP-complete** if all other NP problems that can be reduced to them  **NP-hard** problems are at least as hard as NP-complete problems, and are not necessarily in NP |  |
| **Linear programming** | |
| **General form**: Maximize/minimize a certain metric while conforming to constraints. Inequalities in constraints can be converted to equalities using *slack variables*. Solution is guaranteed to lie at a point on the feasible region, if one exists | |
| **ILP (Integer Linear Programming)**: Restricted to integers/binary instead of real numbers | |
| **Duality**: Drawing on the concepts from linear algebra, the *dual* of an LP can be derived from the *primal* as such:  - Each variable in the primal becomes a constraint in the dual  - Each constraint in the primal becomes a variable in the dual  - Objective direction is inversed, maximum in the primal becomes minimum in the dual  **Symmetry and asymmetry**    **Strong duality**if both the primal and the dual have optimal solutions. If and are solutions to the primal and dual, then | |
| **Approximation** | |
| **k-approximation**: For maximization, for any problem instance , the algo produces a solution such that . For minimization, | |
| **Relaxation** in the context of ILPs means allowing real numbers instead of integer values. Will give lower minima and higher maxima in feasible region. | |
| **TSP approximation**: (For TSP problem instances where the triangle inequality holds) - Find an MST of the graph, choose an arbitrary vertex as root, and return a preorder walk (2-approximation) | |
| **Heuristics and randomization** | |
| **Local search**: Define a neighbourhood around a solution, involving slight permutations. A *k-opt* neighbourhood might involve making *k* adjustments to the current solution to form a new solution. This can be done multiple times to make incremental improvements | |
| **Max-cut local search**: Arbitrarily partition vertices into 2 sets. Loop through all vertices. If a vertex can be switched to the other side to increase crossing edges, do so. Repeat until no improvements can be made. | |
| **Randomized algos** | |
| **Finding median**: Upon randomly sampling containing elements from an input array, we can sort with complexity, and guarantee an element between to with error probability . Note that as n grows larger, error probability decreases | |
| **Las Vegas** solutions have a *deterministic* output and *variable* runtime  **Monte Carlo** solutions have a *variable* output and *deterministic* runtime | |
| **Bernoulli trial**: For a Bernoulli distribution with probability of success , expected tries for first success is | |
| **Law of total expectation**: | |