|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Helpful Notes / Walkthrough** | **Purpose** | **Constraints** | **Runtime** | **Simplex Hand Instructions** |
| **BFS** | N / A | Searching / Traversal | None |  | Arrange the program in slack form (see entry in format table).  Repeat until only has non-basic (new) variables on the RHS:   1. Find entering variable : the basic variable in with most coefficient 2. Find leaving variable :    1. Pick the non-basic LHS constraint with the largest ratio    2. LHS of the above equation 3. Rearrange (on LHS) in terms of (on RHS), eubstitute the new equality for into all other mentions of   **Residual Graph Conversion**  (min cut = max flow, )  B - A  A  A/B |
| **DFS** | Can be done recursively |
| **Krskl** | Keep picking the cheapest possible valid edge in the ENTIRE graph | Finding MST |  |
| **Prm** | Start at the cheapest edge  Keep picking the cheapest edge leaving the tree |  |
| **Dijk** | Keep PQ of vertices ordered by  Let where (until empty)  Run for each | Single-Source Shortest Path | Only positive edges |
| **BllFd** | Do the following times:  For each edge , run  Can detect negative cycles (run one more time, if values change, negative cycle) | No negative cycles |  |
| **FldWar** | Nodes numbered  Semantic: is where uses only nodes  Computational: | All-Pairs Shortest Paths |  |
| **John** | Reweight edges to eliminate negative edges.  Then run Dijkstra for every node ( times). |  |
| **FdFk** | While we can find augmentations:  Send the maximum possible flow over the augmentation  Update edge capacities on the residual graph | Finding Max-Flow | Source has no parents  Sink has no children | maximum flow |
| **EdKp** | Exactly the same as FF, but use BFS to find augmentations (prefer paths with less edges) |  |
| **First/Second** | **First** | **Second** | | **How?** | |
| Informal (Max)/  Standard | Max:  Sub to: | Max:  Sub to: | | Replace all variables with no non-negativity constraints:  Replace equalities with two inequalities:  Flip all constraints (exclude ): multiply by -1 | |
| Standard/  Slack | Max:  Sub to: |  | | Let basic variables be  Let non-basic variables be (the LHS of the new equations)  Set = objective function | |
| Standard / Matrix | Max:  Sub to: | Max:  Such that: , | | , | |
| Primal / Dual | Max:  Such that: | Min:  Such that: | | Dual provides an upper bound for optimal primal value | |
| **Farkas Lemma** |  | | | | |
| **Weak Duality** | optimal solutions to primal and dual, respectively , i.e. (Optimal Primal Value Optimal Dual Value (provided both exist))  (Dual Feasible)(Primal Bounded)  (Primal Feasible)(Dual Bounded)  (Primal Feasible Bounded) (Dual Feasible Bounded) | | | | |
| **Strong Duality** | (Primal Feasible and Bounded) (Optimal Primal Value = Optimal Dual Value) | | | | |

|  |  |  |
| --- | --- | --- |
| **Master Theorem (for , consider )** | | |
|  |  |  |

|  |  |
| --- | --- |
| **Dynamic Programming Template** | |
| Lookup Table – where we store our solutions | = -dimensional array that stores solutions to sub-problems |
| Semantic Array – English description of lookup table | = value of optimal solution for sub-problem with  = the answer to the original problem (where ) |
| Computational Array – Mathematical description of lookup table | = the direct solution to the base case  = operation that uses sub-solutions to solve |
| Correctness – show that semantic = computational | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Greedy Template** | | | | |
| Generic Algorithm | | Sort the input by some attribute. Loop over it. If the current element can be added to the solution, add it. | | |
| Feasibility Proof | | Show that the generated solution doesn’t violate the constraints of the problem | | |
| **Correctness** | Let be some optimal solution, be the greedy solution, be some “cost” function | | | |
| **“Stays Ahead”** | | | **Internal Swapping** | **Swapping Mismatches** |
| Use induction to show:  Since , must be optimal! | | | Let violate ’s ordering rule  Show does not increase | Let and  Show does not increase |
| Induction proves continual switching transforms at no further cost | |

* Proving Problem A in NP:
  + Potential solutions to instances of A have size polynomial to input size(s)
  + Potential solutions to instances of A can be verified as correct in time polynomial to input size(s)
* Proving Problem A in NPC (Reduction):
  + Prove Problem A in NP
  + Prove that some NPC problem reduces in poly-time to A:
    - Prove that inputs of the NPC problem can be converted to the inputs of A (in polynomial time)
    - Prove that by running A on the converted inputs, we solve the NPC problem
* Proving Problem A is Self-Reducible
  + Let A be an optimization or search problem (i.e. we have to find the answer, not just say there is one)
  + Derive A’s corresponding decision problem D (e.g. want to maximize does the problem have solution of size ?)
  + Use D’s oracle a polynomial number of times to solve A (may have to try with many possibilities, may have to use DP)

|  |  |  |
| --- | --- | --- |
| **Examples of Known NPC Problems (for reductions)** | | |
| **Search:** find a satisfying solution, output null if none exist | | **Decision:** does there exist a satisfying solution? |
| **Name (NPC version)** | **Input** | **Satisfying Condition** |
| 3SAT (search) | Variables  Clauses |  |
| Minimum Vertex Cover (optimization) | Undirected Graph | is minimized  (every edge is touched by node in ) |
| Maximum Independent Set (optimization) | is maximized  (no edges connecting nodes of ) |
| -Coloring |  |
| -Clique |  |
| Hamiltonian Cycle | (cycle that visits every node) |
| Hamiltonian Path | such that no node appears twice |
| TSP (search) | Weighted Undirected Graph | such that is minimal |
| Subset Sum | Set target |  |
| 0-1 Knapsack (optimization) | Items , Capacity  weight, value of | such that:  maximized, |

* Approximation Ratios
  + Maximization -approximation: produces a solution that has at the very least times the optimal value
  + Minimization -approximation: produces a solution that has at the very least times the optimal value
  + Proof of ratio: think about the general solution produced by the approximation and compare it to the optimal (HARD: requires thinking)

|  |  |  |  |
| --- | --- | --- | --- |
| **Useful Definitions / Trivia** | | | |
|  |  |  |  |
|  |  | Smallest edge in any cut belongs in the MST |  |
| Optimization Search |  | Min-Cut = Max Flow |  |

Linear Programming Cheat Sheet

LP Problem Definition:

* Inputs: (-dimensional Objective (Linear) Function **F** to maximize or minimize), (Collection of (Linear) Constraints **C**)
* Outputs: (amounts such that **F** is maximized/minimized and no is violated)

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | **Example / Notes** |
| Linear Function | Some -dimensional hyperplane |  |
| Linear Constraint | Linear equality () OR Linear inequality: | Equality:  Inequalities: |
| Feasible Solution | Any solution where no constraint is violated | satisfies |
| Feasible Region | Region that contains all feasible solutions (convex) | It’s the polygon bounded by the constraints |
| Convex | convex |  |
| Polytope | Some -dimensional shape with flat faces | Polygons = 2D polytopes, Polyhedrons = 3D polytopes |
| Vertex | -dimensional vertex =  Intersection of )-many -planes | Corners of a prism |
| Degenerate Vertex | Vertex defined by more than -many -planes | Tip of a square pyramid (4 vertices) – can cause infinite looping in simplex |
| Infeasible Program | Program with no feasible solutions |  |
| Unbounded Program | When the objective function can be maximized or minimized infinitely | Use logic / fix a few variables to show that the objective function can be increased forever without violating any constraints |