|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Helpful Notes / Walkthrough** | **Purpose** | **Constraints** | **Runtime** | **Simplex Hand Instructions** |
| **BFS** | N / A | Searching / Traversal | None |  | 1. Convert the problem into a slack matrix / tableau 2. While has a negative number:    1. Find the variable with the most negative coefficient in    2. Find such that is minimized    3. Do elementary row operations (using only ) to reduce the column of to a 1-hot vector (hot in ) 3. When has no negative numbers:    1. Convert the yielded matrix back into equation form, ignoring the non-basic variables, and report the values of   **Matrix form of slack:**  Isolate for constants / zeros of each equation, convert to matrix form, objective on |
| **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) | | | | |

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 |