

Disclaimer: *These rough notes provided to students, come with no promises.*

27.1 Chapter 4

- Breadth first search. Time: $O(m)$, $O(n)$.
- Shortest path: positive edge weights. Label nodes: source is 0, all others are infinite. Set S to null. Repeatedly find smallest node u in $V - S$, add it to S , and update outgoing arcs (u, v) : if $d(v) < d(u) + l(u, v)$ set $d(v) = d(u) + l(u, v)$.
- Induction: correct distance to each node for all paths through S to node. True at beginning. At each step, node u has correct distance since it is the shortest through S and any path not through S must be at least as long. Moreover, paths through u are updated.
- Graphs with negative edges. Bellman-Ford. Set source to 0. Update all edges n times. If a node changes in the n th iteration there is a negative cycle.
- Proof: after t iterations $d(v)$ is at most the length of any path that uses t edges.
- Heap operations: $O(\log n)$ for decrease-key, find-min.
- Shortest path in dag. In topological order update edges.
- Applications of shortest path. For example, schedule. (Longest path.)

27.2 Chapter 5

- Greedy algorithms: MST. Tree has $n - 1$ edges. Adding an edge and remove another on the resulting cycle leaves a tree.
- Algorithm: Repeat. Sort edges. Add edge if no cycle.
- Cut property: lightest edge in some MST. Proof of cut property: if some MST T does not contain lightest edge, add lightest (u, v) , follow path from u to v remove edge that is on cut. Remains a tree is no more expensive.
- Prims. Dijkstra's with modified update rule. If $d(v) \geq l(u, v)$ then $d(v) = l(u, v)$.
- Data structures: Union-find. Represent as rooted trees (i.e., outpointer for each node.) Root is "set". To see if in same set, compare roots.

- Union make one root point to other. Union by rank. Point to larger rank one. If ranks tie, increase rank by one of new (arbitrary) root. $O(\log n)$ depth.
- Path compression. When one finds, point each guy that one jumped over on way to root at root. $O(\log \log n)$ amortized. Probably don't need to know the proof.
- Huffman coding. Characters, frequencies, find binary tree of minimum cost, where cost is frequency of a character times its depth (or the sum over the non-root nodes of the frequencies of the nodes) Algorithm: find two lowest frequency nodes. Put them together in tree. Make new character which has frequency that is the sum of the two. The cost is the cost of the new tree plus the old.
- Horn formula: implications with positive literals, and all negative clauses. Only set a clause to true if one must. Induction: at any point in time, every true variable must be true in any satisfying assignment. (By construction.)
- Set Cover: greedy. Choose largest set. Must cover $1/k$ of remaining nodes. Thus, after t sets, only $(1 - 1/k)^t n$ are uncovered. When $t > k \ln n$ this is less than 1.

27.3 Chapter 6

- DAG: Longest path. Note: Subproblem: Longest to node. Update rule. Longest over incoming edges.
- Weighted interval scheduling: given a set of intervals, (s_i, e_i) and weights w_i , find max weight set of nonoverlapping intervals.
- Assume sorted by e_i . Subproblem: $C(i)$ is cost of max weight subset that ends before e_i . $C(0) = 0$. Update rule: $C(i) = \max(C(e_i), w_i + C(j))$ where j is the largest integer where $e_j < s_i$.
- LCS. Subproblem: $C(i)$ length of longest sequence ending at i . Update rule:
- Knapsack with repetition, (w_i, v_i) . Subproblem: $C(w)$ is highest value of weight w . Update rule: $C(w) = \max_i(C(w - w_i) + v_i)$.
- Knapsack without repetition. $C(w, i)$ highest value subset of first i items of weight w . Update rule: $C(w, i) = \max(C(w, i - 1), C(w - w_i, i - 1) + v_i)$.
- Edit distance. $M(i, j)$ -best alignment that aligns string $x_1 \dots, x_i$ and y_1, \dots, y_j . Update rule: $M(i, j) = \min(M(i, j - 1) + 1, M(i - 1, j) + 1, M(i - 1, j - 1) + \text{diff}(x_i, y_j))$.
- Dynamic programming of tree structures. Matrix association: a_1, \dots, a_n . (Note $n - 1$ matrices, i has dimension $a_i \times a_{i+1}$.) Subproblem: $M(i, j)$ best solution for matrix i to j . Update Rule: $M(i, j) = \min_k(M(i, k) + M(k, j) + a_k * a_{k+1} * a_{k+2})$.
- All pairs shortest paths: $C(i, j, k)$ shortest i to j path that only uses nodes numbered less than k . Update rule. $C(i, j, k) = \min(C(i, j, k - 1), C(i, k, k - 1) + C(k, j, k - 1))$. $O(n^3)$. Contrast with length version. $O(n^2 m)$.
- Traveling salesman problem: $C(S, j)$ best path that starts at 1 and ends at j and hits S . $C(S, j) = \min_{i \in S} C(S - j, i) + l(i, j)$.
- Dynamic programming on tree. Independent set. $I(v, b)$ largest independent set in subtree that uses v or does not use v depending on whether b is true or false.

27.4 Chapter 7

- Linear program. $Ax \leq b$, $\max cx$.
- Chocolate production. Inequality for each resource. Variable for each type of resource. Maximize profit.
- Bandwidth allocation. Variable for each path. Constraint for each edge, sum over path variables that use edge.
- Vertex solution is optimal.
- Simplex algorithm. Find improvement along each edge. When no improvement, we are done.
- Maximum flow algorithm: route along residual path. (An arc is residual if it has positive capacity remaining or its inverse has positive flow, the residual capacity is the natural thing, i.e., $c(e) - f(e)$ or $f(e^r)$, where e^r is the reverse edge.)
- Eventually, can't. There must be a cut in the residual graph. The flow saturates this cut. The value of the flow is the capacity of the cut. Since any cut upper bounds the flow, the maximum flow equals the minimum cut.