

**Disclaimer:** These rough notes for the instructor, if provided to students, come with no promises.

## 27.1 Chapter 1.

- Modular arithmetic. Can use equivalents in calculations for plus and times. Prove this for times:  $a = x \bmod n$ ,  $b = y \bmod n$ . That is  $a = x + in$  and  $b = y + jn$ , thus  $ab = xy + xjn + yin + ijn^2 = xy + n(xj + yin + ijn)$ , or  $ab = xy \bmod n$ .
- Modular exponentiation:  $\text{modexp}(a, b) \dots x = \text{modexp}(a, b/2)$ .  
Return  $x^2 * x^{b\%2}$ .
- Modular inverses from Euclid's theorem: for  $(a, b)$  there exist integer  $i, j$  such that  $\text{gcd}(a, b) = ax + by$ .  
Runtime for Euclid's? (Basic idea:  $\text{gcd}(a, b) = \text{gcd}(a, a \bmod b)$ . Use formula recursively.)
- Show that every  $a$  has an inverse mod a prime  $p$ .
- Fermat's little theorem. Can use equivalents in exponents. For a prime,  $a^{p-1} \bmod p = 1$ . Proof: There  $ai \neq aj \bmod p$  unless  $j = i$ . Thus,  $a^{p-1}(p-1)! = (p-1)! \bmod p$ .
- Prime number theorem: random number with  $b$  bits has probability  $\Omega(1/b)$  of being prime. probability  $x$  is prime is approximately  $1/\ln x$ .
- RSA: Private-key:  $(N = p, q, e)$ . Public-key  $(N, d = e^{-1} \bmod (p-1)(q-1))$ . Encode  $m$ :  $m^e \bmod N$ .  
Decode  $x$ :  $x^d \bmod N$ .
- Is identity since  $a^{(p-1)(q-1)} = 1 \bmod N$  for  $a$  not divisible by  $p$  or  $q$ . Euler's theorem.
- Hashing: universal hash function  $h() \rightarrow [1, n]$ , for all  $x, y$ :  $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = 1/n$  Example.

## 27.2 Chapter 2

- Karatsuba's algorithm:

$$(a_{top}2^{n/2} + a_{bot})(b_{top}2^{n/2} + b_{bot}) = 2^n a_{top}b_{top} + (a_{top}b_{bot} + a_{bot}b_{top})2^{n/2} + a_{bot}b_{bot}.$$

Four multiplies of  $n/2$  bit words.  $T(n) = 4T(n/2) + n = O(n^2)$ .

- But,  $(a_{top} + a_{bot})(b_{bot} + b_{top}) = a_{top}b_{top} + b_{bot}a_{bot} + (a_{top}b_{bot} + a_{bot}b_{top})$ . The paranthesized quantity is the middle quantity. Thus, the recurrence is  $T(n) = 3T(n/2) + n = O(n^{\log_2 3})$ .
- Master's theorem:  $T(n) = aT(n/b) + n^c$ . If  $c > \log_b a$ ,  $T(n) = O(n^c)$ , if  $c = \log_b a$ ,  $T(n) = O(n^c \log n)$ , and if  $c < \log_b a$  then  $T(n) = n^{\log_b a}$ .

- Recursion tree. Number of leaves is  $O(n^{\log_b a})$ . Depth is  $\log_b n$ . Cost at root is  $n^c$ . Cases are root dominates, all equal, leaves dominate.
- Mergesort. Sorting requires  $\Omega(n \log n)$  time.
- Matrix multiply. Can do this with 7 submatrix multiplies. Thus, recursion is  $T(n) = 7T(n/2) + O(n^2)$ . This is  $O(n^{\log_2 7})$ . Less than  $O(n^3)$ .
- FFT. What are the  $n$  roots of unity. Definition of the FFT:  $FFT(a_1, \dots, a_n)$  is  $A(\omega^i)$ . For multiplying polynomials:  $FFT^{-1}(FFT(a) \cdot FFT(b))$ .
- Algorithm: Take FFT of odd and even. Then combine. Each subproblem used for two outputs. Recurse. FFT procedure.  $T(n) = 2T(n/2) + O(n)$ .

### 27.3 Chapter 3

- Graph. Adjacency list representation: are  $(i, j)$  connected? Space? Matrix representation: fast lookup?  $O(n^2)$  space.
- Depth first search.
- Undirected graphs. Connected components. Properties: no cross edge. Only tree edges and backward edges.
- Directed graphs. Pre, post ordering.  $(u, v)$  is back edge  $[pre[u], post[u]]$  contained in  $[pre[v], post[v]]$ . The other way, it is forward or tree edge. If non-intersecting: cross edge, and  $post[v]$  is before  $pre[u]$ .
- Topological sort. Repeatedly find source. Reverse post ordering number: since for edge  $(u, v)$ ,  $post[u] \geq post[v]$ . Why?
- Strongly connected. Path from  $i$  to  $j$  and  $j$  to  $i$ . Partitions graph: if  $i$  in different components, than the union is strongly connected.
- Algorithm: dfs of reverse graph. Dfs ordering by reverse post ordering number: Each explore finds current sink component.

### 27.4 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.)

## 27.5 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.

## 27.6 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.  $O(n^2m)$ .
- 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.

## 27.7 Chapter 7

- Linear program.  $Ax \leq b, \max cx$ .
- Chocolate production. (In class, we did this as the ad buying problem.) 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.
- Tableau method. Rewrite coordinate system according to distance from each tight constraint. Now, the matrix can be rewritten in this basis, i.e.,  $y_i = b_i - a_i x_i$ . Solving for the  $x_i$ 's plug into  $Ax$ , to get the new matrix in terms of these variables. We also rewrite  $c$  in terms of the  $y_i$ 's as well. This in theory takes  $O(n^3)$  time. To look at each neighboring vertex takes  $O(mn^4)$  time.
- In fact, each iteration can be reduced to take time  $O(nm)$ .
- 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 the capacity of the cut. Since any cut upper bounds the flow, the maximum flow equals the minimum cut.
- Max flow/min-cut is an instance of duality. The dual for a linear program is  $yA \geq c$  and  $\min by$ . The value of the dual is the same as the value of the primal (when the primal has a bounded feasible solution).
- Other forms also work. An inequality corresponds to a positive variable in the dual. An equality corresponds to a unrestricted variable in the dual.

## 27.8 NP completeness

- NP: Problems where there is a polynomial time algorithm for checking a valid solution.
- Does  $G$  have a spanning tree of cost  $K$ ? There is an easy polynomial time algorithm that checks a solution. There is also a polynomial time algorithm that finds it.
- Does  $G$  have a tour that visits every city and has cost  $K$ ? There is a polynomial time algorithm to check whether a tour is indeed a tour and computes its cost.
- Abstractly, a language (i.e., the set of graphs with a low cost TSP tour, the set of 3Sat formulas that are satisfiable) is in  $NP$ , if there is a polynomial time computable function  $C(x, y)$  that for an instance  $x$ , if  $x \in L$  there exists a  $y$  where  $C(x, y)$  is “yes”, and if  $x \notin L$   $C(x, y)$  is “no.”
- A reduction for a problem  $A \leq_p B$ , is a polynomial time computable function where  $x \in A$  if and only if  $f(x) \in B$ .
- A language  $L$  is NP-complete if it is in NP (or a “search problem”, that is a solution is checkable in polynomial time) and for all languages  $A$  in  $NP$ ,  $A \leq_p B$ .
- Circuit-SAT is NP-complete. For a language  $A$  in NP with corresponding program  $C(x, y)$ , consider an instance  $x$ , specify the circuit that corresponds  $C(x, y)$  where  $y$  is now unspecified. This is an instance of circuit-SAT.
- Circuit-SAT can be reduced to 3-SAT (through SAT).
- SAT to 3-SAT. Large clauses to 3-CNF.  $(x_1 \vee \dots \vee x_k)$  can be rewritten as  $(x_1 \vee x_2 \vee y_1) \wedge (\bar{y}_1 \vee x_3 \vee x_4) \dots (y_k \vee x_{k-2} \vee x_k)$
- $3 - SAT \leq_p 3 - SAT_2$
- 3-SAT to Independent Set. Triangle for each clause. Edges from between literals.
- 3-SAT to 3-DM. A variable gadget that either matches boy 0 to girl 0 and boy 1 to girl 1, and leaves pets p1 and p3 free or vice versa.
- A clause gadget that consists of a boy-girl pair, and liking each pet that corresponds to the literal (p1 or p3 if positive, p2 or p4 otherwise.)
- Rudrata path. Ok, should take a look.

## 27.9 Chapter 9

not be examined on final. Backtrack. Kind of dynamic programming..but. Form list of subproblems, if possibly solvable, add to list. Choose from list, form more subproblems.

not be examined on final. Branch and bound. For optimization problem. Find a way to compute a lower bound on a subproblem. Don't add high cost subproblem to list.

- Approximation algorithms. Set cover. Found solution that is within  $O(\log n)$  of the optimal.
- Vertex cover. Factor 2, based on finding maximal matching and including both endpoints of matching in cover. The size of matching is a lower bound on the vertex cover size.
- TSP. Find MST and double it, if triangle inequality is obeyed. Factor of 2.

- Without triangle inequalities. Can't find approximation for any factor, can just choose large value of constant in reduction from Rudrata path.
- Knapsack. Round values to nearest multiple of  $\epsilon v_{max}/n$ . Find optimal with rounded values with dynamic program in time  $O(n^3/\epsilon)$ . Within additive  $n\epsilon v_{max}$  of optimal.
- Local optimization. Design a move to a neighboring solution (e.g., 2-change, 3-change, 4-change.. for TSP) and move as long as one improves.
- Simulated annealing. Occasionally go uphill. For example, go up  $\Delta$  with probability  $e^{-\Delta/T}$  where  $T$  is a parameter that goes down as algorithm progresses.