ECE358

Algorithm: takes input of size n & generates an output.
↳ 3 things matter: correctness, time complexity, space complexity

$\log^k n = (\log n)^k$

$\log^{(k)} n = \underbrace{\log\log\ldots\log n}_{k \text{ logs}}$

$\log^* n = \{\min\{i \geq 0, \log^{(i)} n \leq 1\}$

Big O/Ω, Θ notation: $O(g(n)) = \{f(n) : \exists c>0, n_0>0 \text{ s.t. } 0 \leq f(n) \leq c g(n)\ \forall n \geq n_0\}$

$\Omega(g(n)) = \{f(n) : \exists c>0, n_0 \geq 0 \text{ s.t. } 0 \leq c g(n) \leq f(n)\ \forall n \geq 0\}$

$\Theta(g(n)) \Rightarrow f(n) \in \Omega(g(n)),\ f(n) \in O(g(n))$

Cookbook: Transistivity: { if $f(n) \in O(g(n))$ & $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$
if $f(n) \in \Omega(g(n))$ & $g(n) \in \Omega(h(n))$, then $f(n) \in \Omega(h(n))$
" $\Theta(g(n))$ " " $\in \Theta(h(n))$, " " $\in \Theta(h(n))$

Bin Search ($\log n$)

Symmetry: $f(n) \in O(g(n))$ iff $g(n)\ \Omega(f(n))$

Misc: $n^a \in O(n^b)\ a \leq b$ $\quad \log_a(n) \in O(\log_b n)\ \forall a, b$
$c^n \in O(d^n)$ iff $c \leq d$ $\quad \log(n!) \in O(n \log n)$

If $f(n) \in O(f'(n))$ & $g(n) \in (g'(n))$ then $f(n) \cdot g(n) \in O(f' \cdot g')$
$f(n) + g(n) \in O(\max(f'(n), g'(n)))$

limit method $\displaystyle \lim_{n\to\infty} \frac{f(n)}{g(n)} = ?$
- $0 \to f \in O(g(n))$
- $\infty \to f \in \Omega(g(n))$
- $c>0 \to f \in \Theta(g(n))$

Summations $\displaystyle \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)$ $\quad \displaystyle \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$

$\displaystyle \sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$ $\quad \displaystyle \sum_{k=0}^{\infty} x^k, |x|<1 = \frac{1}{1-x}$ $\quad \displaystyle \sum_{k=0}^{n} a_i - a_{i-1} = a_n - a_0$

Proof Methods for $(P \to Q) \longleftrightarrow ((\neg P) \vee Q) \longleftrightarrow (\neg Q) \to (\neg P)$
Direct Proof: Assume P is true, show Q is also true using basic Logic.
Contrapositive Proof: Assume ¬Q is true, show ¬P "
Proof by Contradiction: Assume $P \wedge (\neg Q)$, find a contradiction of R, a predicate
Disproval by Counter Example: Disprove a $\forall x$ statement w/ 1 example for $P \wedge \neg(Q)$
Induction: Base: prove that $P \to Q$ for base case (usually n=0, etc.)
Hypothesis: assume $P \to Q$ for n
Inductive Step: prove $P \to Q$ for n+1 given the hypothesis
Strong Induction: Induction but hypothesis assumes that $P \to Q$ from n = base ... n

Combinatorial Arguments

rule of products, if you need to choose k & ℓ, $\overbrace{\binom{m}{k}\binom{n}{\ell}}^{ROP}$, corresponds to AND

Rule of Sum if there are # of cases, add possibilities $\sum_{i=0} \binom{m}{i}$, corresponds to OR

Ex to prove $(n-k)\binom{n}{k} = (k+1)\binom{n}{k+1} = n\binom{n-1}{k}$

a) Setup senario: pick k comittee members & 1 pres from n people.

b) show all of these corresponding eq's are equal to scenario

↳ (1) Pick k members first, then pick 1 president from n: $\binom{n}{k} \times \binom{n-k}{1} = \binom{n}{k}(n-k)$

(2) Pick $k+1$ members first, then the pres from those $k+1$ $\binom{k+1}{1}\binom{n}{k+1}$

(3) Pick president from n members, then k from rest $n\binom{n-1}{k}$

Graphs / Trees

$G = (V, E)$ → V: set of nodes/vertices E: set of edges $\{e = (u,v),\ u, v \in V\}$

$|V|$ # of vertices in G $|E|$: # of edges

Types: undirected: directed: connected $|E| \geq |V|-1$ ∃ path btw all vertices disconnected

Def^n's: Path → sequences of edges exist from a node to some other node in G.

Simple path ⇒ path that does not revisit nodes: e.g. not

cycle is a path that starts & ends at the same node. directions matter

Complete graph: $\forall u, v \in V,\ \exists\ e = (u, v)$. IE is a clique

edge → simple path of length 1

TREES: an undirected, connected, acyclic graph.

↳ Forest: ≥ 2 Trees

↳ N-ary: any node has at most N children.

↳ depth of a node: path length from root

↳ height of a tree: largest depth

Graph / tree equivalent propositions

equivalent

- ↳ ① G is a tree (undirected, acyclic, connected
- ↳ ② Any 2 nodes are connected by a unique path
- ↳ ③ G is connected, but if any edge is removed from E, $G(V, E')$ is disconnected
- ↳ ④ G is connected & $|E| = M-1$
- ↳ ⑤ G is acyclic and $|E| = M-1$
- ↳ ⑥ G is acyclic, but if any edge is added to E, $G(V,E')$ contains a cycle

Recurrences

MASTER THEOREM: Textbook Page 94 & 97

SUBSTITUTION ① guess sol^n ② use induction to find consts & solve

↳ ex. $f(n) = \begin{cases} 1 \\ 2(T(n/2)) + n \end{cases}$

Guess $T(n) = n \log n + n$

Base: $T(1) = 1 \log 1 + 1 = 1$

Hypothesis: $T(k) = k \log k + k \quad \forall k < n$

Then $T(n/2) = \frac{n}{2} \log \frac{n}{2} + \frac{n}{2}$

Induction: $T(n) = 2T(n/2) + n$

$= n \log n - n \log 2 + 2n = n \log n + n$

Other examples on Pg 84 - 86 of CLRS

RECURRENCE TREE: CLRS 44, $O(f(n))$ = cost per level × # of levels, use for sub.

Heaps (CLRS-151) (Notes Pg 20)

MAX-HEAPIFY: $O(\log n)$ (CLRS 154)

BUILD MAX-HEAP: $O(n)$ (CLRS 157)

HEAPSORT: $O(n \log n)$ (CLRS 160)

HEAP-MAX: $O(1)$

HEAP-EXTRACT-MAX: $O(\log n)$ (CLRS 163)

HEAP-INCREASE KEY: $O(\log n)$ (CLRS 164), HEAP-INSERT comp sorts are $\Omega(n\log n)$

Quick sort

Worst case : $\Theta(n^2)$ Best / Average-case : $\Theta(n\log n)$

↳ In-Place Standard Partition : CLRS 171 Random. Partition CLRS 179

Linear Sorts

↳ Counting Sort : $O(k+n)$ CLRS 195

↳ Radix Sort $O((k+n)d)$ CLRS 198

Order Statistics

- Minima is 1st order, Maxima (nth order for input n)
- Median : ODD n : $\frac{n+1}{2}$th order EVEN upper : $\frac{n}{2}+1$ lower $\frac{n}{2}-1$: AVG $O(n)$ BEST $O(n^2)$

↳ Found an order? ⇒ Randomized-Select CLRS 216

Binary Search Tree CLRS 287

↳ value of all nodes in left subtree ≤ Parent value

↳ value of all nodes in right subtree ≥ parent value.

↳ Functions : Preorder P, L, R; Inorder L, P, R; Postorder R, L, P — CLRS 287-8

Search (root, x) $\Theta(h)$ 290

Successor (~~root~~) $O(h)$ 292

Predecessor (x) $O(h)$ 292

Delete (x) $O(h)$ Insert $O(h)$

↑ CLRS 294 ↑ 296

Dynamic Programming

↳ ① Define sub-problems & find # of subproblems

↳ use english, the soln

↳ ② Guess part of soln

↳ ③ express the optimal problem as a combination of optimal soln to subproblems

↳ ④ memoize / fill dictionary

↳ ⑤ Solve the original problem ⇒ O(# subproblems × time/subproblem)

GREEDY ALGORITHMS

1.) Cast the optimization problem as one in which we make a choice and are left with 1 subproblem to solve
2) Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe
3) Demonstrate optimal substructure by showing that, having made the greedy choice, what remains is a subproblem w/ the property that if we combine OPT to the subproblem w/ the greedy choice, we arrive at OPT to the full problem

E.g. Activity Selection $S_{ij} = \{a_k \in A : f_i \le s_k < f_k \le s_j\}$ $A_{ij} = A_{ik} \cup$

① The problem reduces to 1 sub: S_{ij} !

Imagine S_{ij} is non empty after greedy selection.

a_m: activity having the earliest finish time. i.e. $f_m = \min\{f_k : a_k \in S_{ij}\}$

→ Need to show $S_{im} = \emptyset$ leaving only S_{mj} as a subproblem

Proof: Suppose S_{im} is non-empty ∴ there exists some a_k w/ $f_i \le s_k \le f_k \le s_m < f_m$
↳ but $f_m < f_k$! ∴ S_{im} is empty.

The new sub-problem is S_{mj} (size reduced.)

② Greedy Choice.

Let A_{ij} be opt. solution (OPT) for S_{ij} & a_k is the first activity in A_{ij}

⇒ $a_k = a_m$, we are done (OPT agrees w/ greedy first activity)

⇒ $a_k \ne a_m$, we construct $A'_{ij} = A_{ij} - \{a_k, a_m\}$

But A_{ij}' is the same size as OPT, → compatible as $(f_m \le f_k)$

③ In $A_{im} \cup \{a_m\} A_{mj}$ ⇒ $A_{im} = \emptyset$
⇒ A_{mj} must be OPT_{mj}, else can be improved.

Amortized ANALYSIS → averaged analysis

Aggregate analysis: ~~Find~~ worst case. Operating time for n success. operations, divide by n.

Accounting analysis: find a fixed cost on operation s.t. you will never be in deficit for cost.

ELEMENTARY GRAPH ALGORITHMS CLRS 589

↳ representation Adj - list. Space $\Theta(V+E)$ Time $\Theta(\text{degree}(u))$ to find adj of u

$\Theta(\text{degree}(u))$ to find if $(u,v) \in E$

Adj - matrix Space. $\Theta(V^2)$ Time. $\Theta(V)$ to find adj edges

$\Theta(1)$ to find if $(u,v) \in E$

BFS CLRS 595 $O(V+E)$

↳ source outwards wave use FIFO QUEUE to manage the outw. spread. $(O(V+E))$

DFS CLRS 604 $\Theta(V+E)$

↳ source along 1 path, output a start time & finish time.

↳ white (undiscovered), black (finished), grey (discovery)

609 ↳ Edges: tree edge (edges used for discovery) Forward edge (children of a visited node)

back edge (parent of a visited node) Cross edge (all others)

↳ Major Properties CLRS 606

↳ Paranthesis Theorem.

(1) (u_d, u_f) disjoint from (v_d, v_f), neither is a descendant of each other

(2) $(u_d, u_f) \subseteq (v.d, v.f)$, u is a descendant of v or (3), vice versa.

↳ White path Theorem.

↳ v is a descendant of u iff at $u.d$, path of only white nodes to v exists

TOPOLOGICAL SORT CLRS 613 $\Theta(V+E)$
↳ linear ordering of nodes such that ~~dag~~ for nodes in a dag always follow their
↳ predecessors.
↳ call DFS(G), and list nodes in reverse order of finishing time.
(while they finish)

Strongly Connected Components. CLRS 617
↳ a subgraph s.t. $\forall u,v$ in V', there is a direct path from $u \to v$, $v \to u$

MINIMUM SPANNING TREES
↳ on an undirected graph $G=(V,E)$, where edges have weight $w(u,v)$
the MST is the tree $T \subseteq E$ s.t. T spans all vertices, & total weight is minimal

GENERIC ALGORITHM
$A \leftarrow \emptyset$
While A not spanning tree
do find safe edge (u,v)
$A \leftarrow A \cup \{(u,v)\}$
return A

cut $(S, V-S)$ is a partition of nodes into disjoint sets, $S \cap (V-S) = \emptyset$
$S \cup (V-S) = V$
edge crosses cut iff $u \in S$ & $v \in V-S$ or v.v.
cut respects ~~A~~ if no edge in A crosses the cut.
edge is light edge crossing cut iff it is the minimum over all edges crossing cut.
↳ always safe.
PRIM'S ALGORITHM → CLRS 634 Bin Min Heap $O(E \lg V)$ Fib Heap $(E + V \lg V)$
KRUSKAL'S ALGORITHM → CLRS 631 $O(E \lg V)$

Single source shortest path
- How to find a shortest route btwn nodes, with weighted edges
  - ↳ Not necessarily unique, graph part formula

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if path exists from } u, v \\ \infty & \text{else} \end{cases}$$

↳ breaks down with negative weight cycle.

RELAX: (u, v, w)

if $v.d > u.d + w(u,v)$
    $v.d = u.d + w(u,v)$
    $v.predecessor = u$

Dijkstra's CLRS 658

Initialize $\delta[v] = \infty$, $S = \emptyset$, $Q = V$ ← in a priority queue.
while $Q \neq 0$
    $u = extract\ min\ (Q)$ $O(E \log V)$
    $S = S \cup \{u\}$
    relax all of u's adj

CLRS 655 DAGS → Top Sort → Initialize d, ∀ nodes in top order do
    ∀ v adj to u, relax

BELLMAN FORD CLRS 651 $O(VE)$
↳ detects negative weight cycles

Difference constraints CLRS 667
↳ setup graph w/ variable as node
    ↳ for $a - b \leq C$, $e \in (b, a)$ $w(e) = C$
↳ run Bellman Ford: if true, solved BF gives values, if False, impossible

Σ = alphabet

Regular Languages

↳ $\emptyset \in L$, $\varepsilon \in L \; \forall L$

↳ $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$

Σ^* set of all strings

↳ $L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$

↳ $\overline{L} = \{x \mid x \notin L \;\&\; x \in \Sigma^*\}$

↳ $L_1 - L_2 = \{x \mid x \in L_1 \text{ but } x \notin L_2\}$

→ $L_1 L_2 \quad \{xy \mid x \in L_1,\ y \in L_2\}$

→ $L^n = LLLL \ldots L \quad L^0 = \{\varepsilon\}$

→ $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$

$L^+ = L^* - L^0$

Regex: ① ε is a r.e. ② $a \; \forall a \in \Sigma$ is an re

③ if R&S are regex's, R+S is a regex (R or S)

④ if R&S are regex, RS is a regex

⑤ if R is a regex, R^* is a regex

⑥ if R is a regex, (R) is a regex

NP Completeness

A problem R is NP-complete if:

$R \in NP$ i.e., A soln can be verified in polynomial time

$R \in$ NP-HARD, i.e. $\forall R' \in NPC \;\; \underbrace{R' \leq_p R}_{\text{polytime reduction}}$

To solve: a) write a verifier function. told to return T or F given a solution.

b) pick a known NPC problem L' and show that $L' \leq_p L$

E.g. HWS A: Vertex Cover B: $\{G : G$ is a undirected graph, V even, w/ vertex cover $\frac{V}{2}\}$

a) Verifier fn: (V, E, C) ← certificate (set of nodes)

if $|C|$!= $|V|/2$

return false

$\forall e \in E$

if (u_e or $v_e \notin C$) return false.

return true.

b) K can be broken into $k < |V|/2$, $k = \frac{|V|}{2}$, $k > \frac{|V|}{2}$

Case 1 reduction fn: in: $G=(V,E)$, out: $G=(V,E)$
$|V|/2 = k$ reduces directly, $A \Leftrightarrow B$

Case 2 reduction fn in: $G=(V,E)$, out $G'=(V \cup \{v_i\}_{1 \ldots 2k-|V|}, E)$
$|V|/2 < k$ If VC exists of size k, new graph will have a VC of K. thus both are satisfied by the same condition, $A \Leftrightarrow B$

FORMAL Case 3 : $k < |V|/2$

IN: $G=(V,E)$. out: $G'=(V \cup \{v_i\}_{1 \ldots |V|-2k}, E \cup E_1 \cup \ldots \cup E_{|V|-2k})$
↳ where $E_i = \{e=(v_i, u), \forall u \in V\}$
Clearly $(|V|+1)(|V|-2k)$ $O(1)$ operations, runs in poly time.

$A \Rightarrow B$

If a vertex cover of size k exists in G, a vertex cover of size $V-k$ must exist in the augmented graph G', as all $|V|-2k$ nodes belong to the vertex cover by construction
∵ the graph has $2|V|-2k$ nodes, B is satisfied. ∴ $A \Rightarrow B$

$B \Rightarrow A$

If a vertex cover of size $|V|-k$ exists in G', we can subtract all the nodes added, which must be in the VC by construction, and A will have a VC of size k as required.
Thus A is also true. ∴ $B \Rightarrow A$

∴ $A \Rightarrow B$, $B \Rightarrow A$: Thus $A \Leftrightarrow B$, & B is NP hard.

Since $B \in NP$ & $B \in$ NP-HARD, $B \in NPC$.