$f = O(g)$: $f \leq g$ | $\frac{f}{g} \to 0$
$f = \Omega(g)$: $f \geq g$ | $\frac{f}{g} \to \infty$
$f = \Theta(g)$: $f = g$ | $\frac{f}{g} \to 1$

exponential dominates polynomial
polynomial dominates logarithm

Master Theorem:
$T[n] = a \cdot T(\frac{n}{b}) + O(n^d)$
CASE 1: $d < \log_b a$
$T(n) = O(n^{\log_b a})$
CASE 2: $d = \log_b a$
$T(n) = \Theta(n^d \log n)$
CASE 3: $d > \log_b a$
$T(n) = \Theta(n^d)$

$a^{\log b} = b^{\log a}$ $\log_2 8 = 3$ $2^3 = 8$

---

n^{th} roots of unity: Define $w = e^{(i\frac{2\pi}{n})}$
$e^{i\frac{2\pi}{8}(cw)}$ $1, w, w^2, \ldots, w^{n-1}$

Polynomial Multiplication
Input: $A(x)$, $B(x)$
Goal: $A(x) \cdot B(x)$
- represent polynomials using coefficients, or values
- degree n polynomial defined by n+1 points

Evaluation: Coeff → Values. Interpolation: Values → Coeff.
$C(x) = A \cdot B$ = degree 2n polynomial. need 2n+1 points to define.

MST
- picking lightest edge across cut will always work

Graphs
Adj Matrix: If n vertices
$A(i,j) = 1$ if $(i,j) \in E$, ow 0

Lists: $\forall$ vertex V, $L_v = \{$ list of neighbors $\}$

DFS - postorder = topo sort sorted by
$O(V+E)$
Tree Edge - part of DFS traversal
Cross - diff part of subtree
Back - to ancestor
Forward - to descendant
$u \to v$: FWD, BACK, CROSS
u v v u
v u u v
v v u u
For DAG, $u \to v \Rightarrow post(u) > post(v)$
Directed cycle $\Leftrightarrow$ Back Edges
Source: vertex with no incoming edges
Topo Sort: find src, remove and repeat
OR
order vertices in DAG in decreasing post value

SCC: $u \Rightarrow v$ iff $\exists$ $u \to v$, $v \to u$ path

To find SCC in directed graph $O(V+E)$
1) Run DFS on G reverse
2) Run DFS on G in decreasing post vertex
↳ start at U

Shortest Path:
If all edges same length: BFS $O(V+E)$
O.W.
Dijkstra's: $O((V+E)\log V)$
Put all vertices on PQ, (s, 0)
while PQ is not empty:
u = pop min dist off PQ
dist[u,] = c
for $\forall$ v, u → v
if (dist(v) > dist(u) + len(u,v))
PQ.decrease Dist(v, dist(u) + len(u,v))

Shortest path goes through every vertex at most once.
- Kruskals
1) sort edges
2) repeat until spanning tree
- add lightest edge that does not create a cycle.
$O(E \log E)$

Prim's
- Pick random node
- pick vertex whose edge has least weight
- find unvisited vertices, pick smallest edges that connect visited set with unseen vertices. $O(E + V\log V)$

To find source vertex / sink.
run DFS and get highest post number / topo sort

Bellman-Ford - $O(V \cdot E)$
V-1 iterations
Update vertices using known path length

Dijkstra's will not break if you add $|V|$ to each edge.

---

m bit # x n bit # = m+n bit number

$T(n) = aT(\frac{n}{b}) + f(n)$, $a \geq 1$, $b > 1$,
$f(n) = \Theta(n^c \log^k n)$,
$c = \log_b a$
$T(n) = \Theta(n^c \log^{k+1} n)$

Fourier Transform:
Degree d polynomial, defined by d+1 points
Find n^{th} roots of unity, $n > d+1$
e.g. $P(x) = 1 + x^3$
4^{th} roots of unity, $\{1, i, -1, -i\}$
$P(1) = 2$, $P(i) = 1 - i$, $P(-1) = 0$, $P(-i) = 1+$

FFT:
Run FFT on $A(x)$
- find roots of unity
- express A in even and odd terms
$A(x) = B(x^2) + xC(x^2)$,
B contains even, C contains odd
$A(x) = 4 + 2x + 3x^2 + x^3$
$B(x) = 4 + 3x$, $C(x) = 2 + x$
$B(x^2) = 4 + 3x^2$, $xC(x^2) = 2x + x^3$
Evaluate $A(x)$ at $x = \{1, i, -1, -i\}$, 4^{th} roots of unity.

- linear, i.e.
$FFT(1,2,3,4) + FFT(-1,-2,-3,-4)$
$= [0,0,0,0]$

$n + (n-1) + \dots + 3 + 2 + 1 = \frac{(n+1)(n)}{2} \in \Theta(n^2)$

$n + \frac{n}{2} + \frac{n}{4} + \dots + 4 + 2 + 1 = \Theta(n)$ ($\log n$ terms)

$$\sum_{k=1}^{\log n} 2^{-k} \leq \sum_{k=1}^{\infty} 2^{-k} = 1$$

$$\sum_{i=1}^{\sqrt{n}} (n - i^2) = \sum_{i=1}^{\sqrt{n}} n - \sum_{i=1}^{\sqrt{n}} i^2 \simeq n\sqrt{n} - \frac{1}{3}n\sqrt{n} \in \Theta(n\sqrt{n})$$

$$\sum_{i=1}^{n} (2^i) = 2^{n+1} - 2 = \Theta(2^n)$$

Dijkstra's will work with no negative edges, no negative cycles.

```
dfs(G)
for all v ∈ V
    visited(v) = False
for all v ∈ V:
    if not visited(v): explore(v)
explore(G, v)
# Input: G = (V, E)
Output: visited(u) is set to true for all
        nodes u reachable from v
    visited(v) = true
    previsit(v)
    for each edge (v, u) ∈ E
        if not visited(u): explore(u)
    postvisit(v)

procedure previsit(v)
    pre[v] = clock
    clock = clock + 1

procedure postvisit(v)
    post[v] = clock
    clock = clock + 1
```

For DAG to be semi-connected, there must be a single path that goes through all vertices.
= when linearized, there is an edge between every consecutive pair of vertices

Want to prove/find 1 and only 1 src vertex when seeing if it is possible to reach every other vertex in the graph.

Any sorting or comparing values — can use merge/quick sort.

unweighted graph traversal - BFS

MST - look at cycles.

Set of consecutive integers: if N is odd, # 0's = # 1's; 0.w # 0's > # 1's

Sat should have same number of 1's and 0's in each "column"
missing number will fill in gap

Duality

max $4x_1 + 7x_2$

$x_1 + 2x_2 \le 10$ (Y_1)

$3x_1 + x_2 \le 14$ (Y_2)

$2x_1 + 3x_2 \le 11$ (Y_3)

$x_1, x_2 \ge 0$.

$(Y_1 + 3Y_2 + 2Y_3)x_1 + (2Y_1 + Y_2 + 3Y_3)x_2 \le 10Y_1 + 14Y_2 + 11Y_3$

Answer:

$Y_1 + 3Y_2 + 2Y_3 \ge 4$

$2Y_1 + Y_2 + 3Y_3 \ge 7$

$\min(10Y_1 + 14Y_2 + 11Y_3)$

Concave II

Convex ()

- Running time of DP $\le$ # edges in underlying DAG. F.
- 2 ways to implement DP algo. Bottom up, recursion w/ memoization. Both have asymptotically same time and space complexity. F.
- Possible to verify that flow f is a max flow in $O(|E|)$ time. T.
- All linear programs can be solved in polynomial time. T.
- If linear program has unbounded feasible region, it does not have optimum solution of finite value. F.

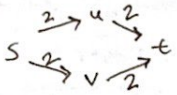
  $\min x+y$, $x, y \ge 0$.
- In successive iterations of Maxflow, total flow passing ~~thru~~ thru vertex may decrease. T.
- residual graph of max flow f can be strongly connected. F
  → If there is a path from s to t, flow is not maximal.
- Traveling Salesman uses exponential amount of memory, has exponential subproblems. T.
- value of edit distance between two strings of length n can be computed using $O(n)$ memory. T.
  → only need two columns

- maximum weight in a graph can be in a MST
- an implementation of a DP problem via memoization can yield asymptotic improvements in running time with respect to an iterative solution
- the Knapsack problem on n items where each $i \in \{1, \dots, n\}$, weight of the ith item is i can be solved in time polynomial in the input length.
- in a DP solution, the asymptotic space requirement may not always be as big as the number of unique subproblems
- in a DP solution, the asymptotic time complexity is always at least as big as the number of unique subproblems.
- there exists a polynomial time algorithm to find an appropriate minimum set cover of size at most $\log_2 n$ times the size of the optimal set cover
- Huffman encoding cannot always reduce the size of any document.

Exchange Argument!

Compare greedy w/ optimal.
Find first index where they differ
Show that greedy element $\ge$ optimal

All instances of LP may not have exactly one optimum



Max flow can be reduced to LP.
A reduced to B, B reduced to C, A reduced to C

If graph $G(V, E)$ has multiple sources and sinks,
$T = \{t \mid t \text{ is a sink}\}$, $S = \{s \mid s \text{ is a source}\}$
Add a new source s^* with edge (s^*, s) from S^* to every node $s \in S$ with capacity $c(s^*, s) = d_s$, $d_s = \sum_{(s,u) \in E} c(s, u)$.
Add a new sink t^* with an edge (t, t^*) from every node $t \in T$ to t^* with capacity $c(t, t^*) = d_t$, $d_t = \sum_{(v,t) \in E} c(v, t)$.
→ Ford Fulkerson

- Cross edges occur in DFS tree only if graph is directed
- cost of every TSP tour in a graph is always greater than the cost of a MST
- Dijkstra's doesn't work on every DAG with negative edge weights

- linear program w/ integral optimal value may not have unique solution
- Min Vertex Cover is polynomial time solvable on a tree
- simplex is exponential
- Bipartite matching → Independent Set
- Independent Set → Max Flow iff P = NP
- Factoring → Rudrata Path
- MST is an NP problem

rank - height of subtree (log n) makeset(x) find(x) union(x,y)
makeset - O(1)
make root of shorter tree point to root of taller tree in union.
↳ height ↑ iff trees are same height
↑ tree, ↑ rank
- $\forall x$, rank(x) < rank(π(x)) (no pc)
- Any root node of rank k has $\geq 2^k$ nodes in its tree.
- node of rank k has $\geq 2^k$ descendants.
- n element tree $\Rightarrow \leq \frac{n}{2^k}$ nodes of rank k.
$\Rightarrow$ max rank = log n.
Path Compression → all pointers in path point to root

log*: # logs needed to bring n down to 1

Huffman - pick nodes with smallest 2 freq and combine; O(n log n)
cost - sum of frequencies of all leaves and internal nodes.

Horn Formulas - set all false unless have to be true
- follow clauses (+) and set vars
- see if they match with negative clauses O(n)

Set Cover
- B contains n elements, optimal cover consists of k sets.
- greedy uses at most k ln n sets
Key: n_t remaining (uncovered) after t iterations.
$\exists$ some set w/ $\geq \frac{n_t}{k}$
$\therefore n_{t+1} \leq n_t - \frac{n_t}{k} = n_t(1-\frac{1}{k})$
$\Rightarrow n_t \leq n_0(1-\frac{1}{k})^t$
$\therefore n_t \leq n_0(1-\frac{1}{k})^t < n_0(e^{-\frac{1}{k}})^t$
$= ne^{-\frac{t}{k}}$
when $t = k \ln n$, $n_t < ne^{-\ln n} = 1$
$\Rightarrow$ no uncovered elements.

Longest Increasing Subsequence
DAG: edges iff i<j, $a_i < a_j$
$\Rightarrow a_i, a_j$ consecutive elements.
For j = 1, 2, ..., n: O(n²)
L(j) = 1 + max{L(i) : (i,j) ∈ E}
return max; L(j)

Edit Distance
Edits: insertions, deletions, substitutions.
x[i] - x[i]
- y[j] y[j] O(mn)
E(i,j) = min{1 + E(i-1,j), 1 + E(i,j-1), diff(i,j) + E(i-1,j-1)};
diff(i,j) = { 0 if x[i] = y[j]; 1 otherwise }

Knapsack. (Repetition) n items, capacity W
K(w) = max value achievable with a knapsack of capacity w. O(nW)
If optimal solution includes w_i, remove.
$K(w) = \max_{i: w_i \leq w} \{K(w-w_i) + v_i\}$

w/out Repetition
K(w,j) = max value with capacity w and items 1, ..., j O(nW)
K(w,j) = max{K(w-w_j, j-1) + v_j, K(w,j-1)}

Chain Matrix Multiplication
Multiplying m×n, n×p takes mnp multiplications
Matrices as leaves, interior nodes as products
C(i,j) = min cost of multiplying $A_i \times A_{i+1} \times \dots \times A_j$
O(n³)
$= \min_{i \leq k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$

Shortest Reliable Path
dist(v,i) = len of shortest path from s→v that uses i edges
dist(v,i) = $\min_{(u,v) \in E}$ (dist(u, i-1) + l(u,v))

All Pairs Shortest Path Floyd-Warshall
u→v, increase intermediate nodes.
calculating beginning from no intermediate edges
dist(i,j,k) = len shortest path from i→j where only nodes {1,2,..,k} are used.
= min{dist(i,k,k-1) + dist(k,j,k-1), dist(i,j,k-1)} redundant
O(|V|³)

Traveling Salesman
Portion of travels - have to pick second to last, i
C(S,j) = len shortest path visiting each node in S once, starting at 1 and ending at j.
$= \min_{i \in S, i \neq j} C(S-\{j\}, i) + d_{ij}$
O(n²2ⁿ).

Independent Sets in Trees
Graph = tree.
I(u) = size of largest independent set of subtree hanging from u
I(u) = max{1 + Σ I(w) (grandchildren w of u), Σ I(w) (children w of u)}
O(|V|+|E|)

Max Flow
- does not violate edge capacities
- flow is conserved O(V·E) - Ford Fulkerson
size(f) = $\sum_{(s,u) \in E} f_{su}$ O(V·E²) - Karp Edmonds

Path Compression - during each find, when a series of parent pointers is followed up to root, change all pointers so they point directly to root.

LP doesn't have strict inequality
- don't use parameters as variables
→ don't optimize.

CMM - different ways of grouping things together.

log 65536 = 16
log* 65536 = 4

passed in k ...
value = k
input size = log k

Search Problem - specified by (C, I, S),
I: instance
S: proposed solution

TSP: Input: n vertices
$\frac{n(n-1)}{2}$ edge lengths
budget b
Output: tour (cycle that passes thru every vertex once)
OR no soln

Optimization $\leftrightarrow$ Search

Rudrata Cycle: Input: Graph
Output: whether $\exists$ cycle that visits each vertex exactly once.

Path: same, output path (not cycle)

Balanced Cut: partition input: graph w/ n vertices, budget b
vertices into sets S, T: $|S|, |T| \geq \frac{n}{3}$ and there are $\leq b$ edges between S and $\bar{T}$

ILP: Input: A, $m \times n$ matrix
b, m vector
Find nonnegative integer vector x
satisfying $Ax \leq b$

ZOE (zero one equations):
Input: A, $m \times n$ matrix w/ 0-1 entries
Find x, vector of 0's and 1's satisfying $Ax = 1$

3D Matching: n boys, n girls, n pets
Find n disjoint triples (b, g, p)

Independent Set - Input: graph, integer g
Find g vertices that are independent/don't touch same edge

Vertex Cover: graph g, budget b
find b vertices that touch every edge

Set Cover: Input: set E and subsets $S_1, \ldots, S_n$, budget b
Find b subsets so union is E

Vertex Cover: set E = all edges, subsets are edges adjacent to each vertex.

Clique: Input: graph, goal g
Find g vertices st all possible edges between them are present (complete)

Longest Path: Input: Graph G, ZO edge weights, vertices s, t; goal g
Find path from s to t with weight $\geq g$

Knapsack: weights $w_1, \ldots, w_n$; values $v_1, \ldots, v_n$
weight capacity W, goal g
Find set of items whose weight $\leq W$ total value $\geq g$

SubsetSum: Input: set of integers, goal g
Find subset of integers which add to g.

all NP-complete
If $A \rightarrow B$, B is at least as hard as A

3SAT
$(\_ \vee \_ \vee \_)(\_ \vee \_ \vee \_)$

Reductions
Rudrata Path $\rightarrow$ Rudrata Cycle
Add a vertex x between s, t. Edges are $s-x$, $x-t$

3SAT $\rightarrow$ Independent Set
Triangle between 3 vars in 1 clause
Edges between $x, \bar{x}$ (opposites)
$\rightarrow$ can only choose one of 3, in Δ
goal g = # clauses in 3SAT

SAT $\rightarrow$ 3SAT
for clauses w/ >3 literals
$(a_1 \vee a_2 \vee a_3 \ldots \vee a_n)$
$\hookrightarrow (a_1 \vee a_2 \vee y_1)(\bar{y}_1 \vee a_3 \vee y_2)$
$(\bar{y}_2 \vee a_4 \vee y_3) \ldots (\bar{y}_{k-3} \vee a_{k-1} \vee a_k)$

Independent Set $\rightarrow$ Vertex Cover
- a set of nodes S is a vertex cover of G (S touches every edge) iff $V - S$ are an independent set of G
- look for vertex cover of G with $|V| - g$ nodes, if exists
$\hookrightarrow$ take all nodes not in it

Independent Set $\rightarrow$ Clique
complement $G = (V, E)$: $\bar{G} = (V, \bar{E})$
$\bar{E}$ is all unordered pairs of vertices that are not in E.
- a set of nodes S is an independent set of G iff S is a clique of $\bar{G}$

Independent Set $(G, g) \rightarrow$ Clique $(\bar{G}, g)$

ZOE $\rightarrow$ SUBSET SUM
columns A of ZOE = binary integers (digit to bottom)
look for subset of columns that add to $111\ldots111$
Use base $n+1$ to make up for carry.
e.g.
$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$
18 5 4 8
(1 0 1 1)

LP formulation of vertex cover,
for two vertices i, j,
$x_i + x_j \geq 1$

LP solution should be lower bound on optimal solution

Approximation algorithm says you need to choose all vertices w/ value greater than $1/2$

Approximation Ratio:
$\alpha_A = \max_I \frac{A(I)}{OPT(I)}$

Matching - subset of edges that have no vertices in common
$\rightarrow$ any vertex cover of a graph must be at least as large as the number of edges in any matching in G.
$\rightarrow$ matching provides lower bound

Let S be a set that contains both endpoints of each edge in a maximal matching G.
$\rightarrow$ upper bound $\leq 2|M|$

Undirected longest path: NP complete

Ordering of vertices, N
$\forall e \in E : (u, v)$

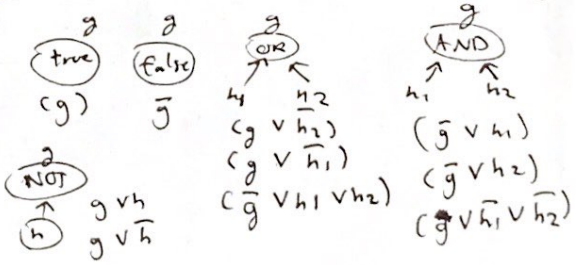
$N(u) > N(v)$	$N(u) > N(u)$
topologically sorted DAG	reverse sorted DAG

NP-hard - at least as hard as problems in NP

Search - looking for something
Optimization - best way
Decision - trying to decide whether something is true

Circuit SAT
- AND gates join clauses $\rightarrow$ output
- each clause is OR of literals
- literal is an unknown input gate or the NOT of one

Circuit SAT $\rightarrow$ SAT
- $\forall g \in$ gate, var g



- # of scc in an n vertex DAG $\geq n$
- every directed graph can be decomposed into a DAG of scc
- cross edges occur in DFS tree only if graph is directed
- running DFS from a node in a sink scc yields a scc of the graph
- MST never contains the heaviest edge in every cycle
- some MST will contain the lightest edge in every cut

Carmichael numbers

Miller Rabin looks for these.

Primality - ∃ only false positive, believing carmichael number is prime.

If there is a number a relatively prime to n s.t. $a^{n-1} \not\equiv 1 \bmod n$, n is composite.

a is Fermat witness for compositeness of n.

If ∃ 1 valid witness, at least ½ of potential witnesses are valid

FLT

If p is prime and $\gcd(a,p) = 1$, $a^{p-1} = 1 \bmod p$

If p is prime, always gives correct result.

Check n

Cases:

1) n is prime
2) n is composite (not prime)
   a) has Fermat witness i.e. $a^{n-1} \not\equiv 1 \bmod n$ → just run again
   b) has no Fermat witness
      ↓
      Carmichael numbers.

MR

If x, n are positive integers s.t. $x^2 \equiv 1 \bmod n$ but $x \not\equiv \pm 1 \bmod n$, n is composite.

One way

1) efficient - solve in poly time.
2) bijective