# Asymptotic Notation

$f(n) = O(g(n))$ Upper bound: $c > 0$ that $f(n) \le c \cdot g(n)$

$f(n) = \Omega(g(n))$ Lower bound: $c > 0$ that $f(n) \ge c \cdot g(n)$

$f(n) = \Theta(g(n))$ $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

LIMITS: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = o(g(n)) \\ c > 0 & f(n) = \Theta(g(n)) \\ \infty & f(n) = \Omega(g(n)) \end{cases}$

## Master Theorem

$T(n) = aT(\frac{n}{b}) + O(n^d)$

$$T(n) = \begin{cases} O(n^d) & d > \log_b a \\ O(n^d \log n) & d = \log_b a \\ O(n^{\log_b(a)}) & d < \log_b a \end{cases}$$

ALGORITHMS = FUN!

**Union find**

Find root node by going back up to parents

- connect smaller tree root to bigger tree root

$O(\log n)$ for find

# Divide and conquer

① Break problem into subproblems
② Recursively solve subproblems
③ Combine results

Proofs: APPLY INDUCTION
① Base case
② Assume algo correctly solves subproblems w/ recursion
③ Correct result when we combine

Ex: Mergesort:
① Break list down into several sublists until each sublist consists of a single element →
② Merge sublists to return sorted list
$T(n) = 2T(\frac{n}{2}) + O(n)$, $O(n\log n)$

[Add 1-2 more examples here]

* Karatsuba for fast multiplication

Ex: Finding majority element
① Divide into left and right halves and recursively find ME
② If ME same for each half, return
③ If diff ME, or one has ME and other doesn't: iterate over entire array and count total # of each ME

$T(n) = 2T(\frac{n}{2}) + O(n) \rightarrow O(n\log n)$

# FAST FOURIER TRANSFORM

Naive: $O(N^2)$ Fast: $O(n\log n)$

DFT: Multiply coefficients w/ root of unity matrix to evaluate polynomial @ roots of unity

$$\begin{bmatrix} P(1) \\ P(\omega_n) \\ P(\omega_n^2) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

$n^{th}$ root of unity: $\omega_n = e^{\frac{2\pi i}{n}}$

$(\omega_8)^2 = e^{i(\frac{2\pi}{8})2} = e^{i\frac{2\pi}{4}} = \omega_4$

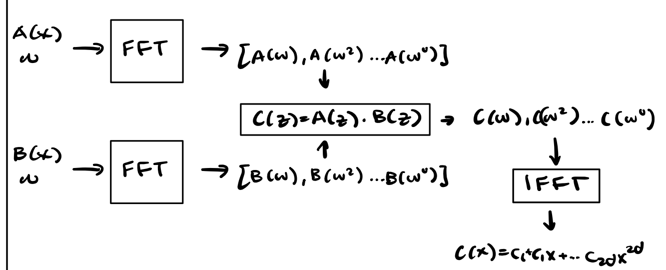
**Generator Fact:** $\omega_i = \omega_1^i$
magical: squares of $n^{th}$ roots = $(n/2)^{th}$ roots
$M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega^{-1})$

IFT: Get coefficients from evaluations

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & & \omega_n^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} P(1) \\ P(\omega_n) \\ P(\omega_n^2) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix}$$

Polynomial multiplication
- Given $A(x)$ and $B(x)$ of deg D
① Pick points $x_0, x_1 \cdots x_{n-1}$ $n \ge 2d+1$
② Evaluate $A(x_0), A(x_1) \cdots A(x_{n-1})$ and $B(x_0), B(x_1) \cdots B(x_{n-1})$ FFT
③ Multiply: $C(x_k) = A(x_k)B(x_k)$ for $k = 0 \cdots n-1$
④ Interpolate: Recover $C(x) = c_0 + c_1 x \cdots + c_{2d}x^{2d}$ IFFT

$A(x) = a_0 + a_1 x + \dots + a_d x^d$
$B(x) = b_0 + b_1 x + \dots + b_d x^d$




* Divide and conquer

**FFT:** Split into even and odd halves

```
def FFT(p=[p0,p1 ··· pn-1], n)
    if n==1 → return p
    yE = FFT([p0,p2 ··· ], n/2), yO = FFT([p1,p3, ··· ], n/2)
    y = [0]·n
    for j in range(n/2)
        y[j] = yE[j] + wn^j · yO[j]
        y[j+n/2] = yE[j] - wn^j · yO[j]
    return y
```

evaluate

$P(x) = p_0 + p_1 x + p_2 x^2 + \dots + p_{n-1} x^{n-1}$ (n coeffs)
$E(x^2) = p_0 + p_2 x^2 + p_4 x^4 + \dots$ (n/2 coeffs)
$O(x^2) = p_1 + p_3 x^3 + p_5 x^5 + \dots$ (n/2 coeffs)
$\boxed{P(x) = E(x^2) + O(x^2)}$ $T(n) = 2T(\frac{n}{2}) + O(n)$
* solve for $E(x^2)$ and $O(x^2)$

**FFT input** $n$ = power of 2 & $d \le n-1$
① Coefficients of deg-D polynomial $A(x)$
② $n^{th}$ root of unity $\omega$ $\boxed{O(n\log n)}$

**FFT output** $A(x)$ eval @ n points

**IFFT input**
① d+1 points of $A(x)$
② $n^{th}$ root of unity → **OUTPUT** $A(x)$ coefficients

* **Cross correlation**
① REVERSE ONE OF THE VECTORS
② Encode into polynomial
③ Multiply w/ FFT

edge (a,b) Tree = part of forest

FORWARD: $pre(a) < pre(b) < post(b) < post(a)$ $\overset{a}{[}\,\overset{b}{[}\;\overset{b}{]}\,\overset{a}{]}$
↳ = leads to non-child descendant

BACK: $pre(b) < pre(a) < post(a) < post(b)$ $\overset{b}{[}\,\overset{a}{[}\;\overset{a}{]}\,\overset{b}{]}$
↳ = leads to ancestor

CROSS: $pre(b) < post(b) < pre(a) < post(a)$ $\overset{b}{[}\,\overset{b}{]}\;\overset{a}{[}\,\overset{a}{]}$

# DFS

```
def explore(G, v):
    visited(v) = True
    previsit(v)
    for each edge (v,u) in E:
        if not visited(u):
            explore(u)
    postvisit(v)
def dfs(G):
    for all v in V:
        if not visited(v):
            explore(v)
```

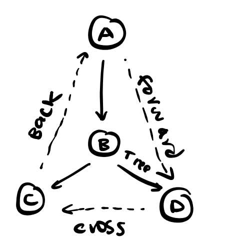
- $O(|V|+|E|)$
- Stack
- reachability, decompose → SCC

* Topological sort
if G is DAG and $(u,v) \in E$: $post(u) > post(v)$

# BFS

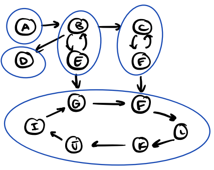
```
def bfs(G, s):
    for all u in V:
        dist(u) = infinity
    dist(s) = 0
    Q = [s]
    while Q is not empty:
        v = Q.eject()
        for each edge (v,u) in E:
            if dist(u) == ∞:
                Q.add(u)
                dist(u) = dist(v) + 1
```

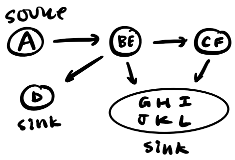
- $O(|V|+|E|)$
- Queue
- shortest path
- all vertices not connected to start vertex = ignored



## SCC: strongly connected component

vertices a, b are strongly connected if path a→b and path b→a




## KOSARAJU

* Find all SCC $O(|V|+|E|)$

1. Reverse G and run DFS → want $post^{rev}(v)$
2. Run DFS on G starting at vertex w/ highest post order in $G^{rev}$ (*unvisited)
   ↳ must belong in sink → traversed vertex part of current SCC
3. Repeat 2-3 until all SCC labeled

*** Every directed graph can be turned into DAG bc of SCCs**

## Directed Acyclic Graph

- no cycles
- ≥1 sink, ≥1 source
- can be TOPOLOGICALLY SORTED (every edge leads to vertex w/ lower post)

TOPOLOGICAL

DFS → look @ REVERSE post-order

a→b, a before b in order

## Shortest Paths

**Djikstra's**: greedily determines shortest path

BFS w/ priority queue for edge weights

NO NEGATIVE WEIGHTS!

input: graph w/ pos weights, start node

```
for all u ∈ V:
    dist(u) = ∞
    prev(u) = null
dist(s) = 0
H = makequeue(V)
while H is NOT empty:
    u = deletemin(H)
    for all edges (u,v) ∈ E:
        if dist(v) > dist(u) + l(u,v):
            dist(v) = dist(u) + l(u,v)
            prev(v) = u
            decreasekey(H,v)
```

$O((|V|+|E|)\log(|V|))$

## Bellman-Ford

```
for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0
repeat |V|-1 times:
    for all e ∈ E:
        update(e)   →   def update((u,v))
                            dist(v) = min{dist(v), dist(u) + l(u,v)}
```

update all edges |V|-1 times

$O(|V||E|)$

* works on negative edges
* relax all edges as many times as needed until all shortest paths found

## DAG shortest Path

```
def short_dag(G, l, s)
    linearize G
    for each u in V in linear order
        for all edges (u,v) in E:
            update (u,v)
```

* $O(|V|+|E|)$
* works for negative edges
* visits in topological order

## Greedy Algorithms

: At each timestep, choose (biggest, cheapest, earliest) CURRENT best option

EXCHANGE ARGUMENT

1. Assume optimal soln w/ sequence $[o_1, o_2 \cdots]$
2. Assume greedy soln w/ sequence $[g_1, g_2 \cdots]$

WLOG: 1st point of discrepancy between G and O @ index i

$g_i$ better/equivalent choice to $o_i$

↳ G equally or more optimal

## HORN FORMULA

want to satisfy all clauses

1. set all vars = false
2. while implication not satisfied: set right-hand var = True
3. If all pure neg clauses satisfied → return
4. else → return not satisfiable

* a → B
  not a ∨ B
* can only change False to TRUE

## Set cover (greedy approx)

* find min # of subsets that cover set $U = \{1, 2 \cdots n\}$

Pick set $S_i$ w/ largest # of uncovered components

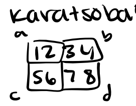
Repeat until all vertices covered

$k_g \le k_o \cdot \ln(n)+1$

## Algo design

1. ID algo method
2. Method → tools
   - D & C: FFT
   - graphs: Dijkstra's, graphs, MSTs
3. Pitfalls:
   - D & C: idea / proof runtime
   - graphs: building graph
   - greedy: proof w/ exchange

Karatsuba's




1. a × c
2. b × d
3. (a+b)(c+d)

$x \cdot y = 10^n ac + 10^{n/2}(ad+bc) + bd$

## Greedy set cover

while we haven't covered all sets

add set w/ largest # of elements

If optimal uses k sets, greedy uses k log n sets

proof by induction $n_{t+1} \le n_t(1-\frac{1}{k})$ $\forall t \ge 0$

## Minimum Spanning Tree

Goal: Given weighted undirected graph $G=(V,E)$ → find lightest weight tree that connects ALL vertices V

CUT PROPERTY: lightest edge across cut in some MST

: suppose edges X part of MST → let $(S, V \setminus S)$ be any cut for which edges in X do NOT cross the cut

e = lightest edge across cut

edges $X \cup \{e\}$ part of some MST

cycle property: largest edge on any cycle is NEVER in any MST

**Kruskal's**: Repeatedly add next lightest edge that does NOT produce cycle

* GREEDY
* use disjoint sets

$O(|E|\log|V|)$

union: log n

find: log n

```
for all u ∈ V:
    makeset(u)
X = {}
sort edges E by weight
for all edges {u,v} ∈ E        in increasing weight
    if find(u) ≠ find(v)
        add edge {u,v} to X
        union(u,v)
```

**Prims**: on each iteration: pick lightest edge between vertex in current subtree S and vertex outside S

$O(|E|\log|V|)$ * better on dense graphs, use PQ

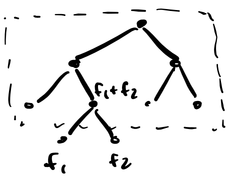
```
for all u ∈ V:
    cost(u) = ∞
    prev(u) = nil
start w/ any node v0
cost(v0) = 0
H = makequeue(V)
while H is NOT empty:
    v = deleteMin(H)
    for each {v,z} ∈ E:
        if cost(z) > w(v,z):
            cost(z) = w(v,z)
            prev(z) = v
            decreasekey(H,z)
```

## Huffman Encoding

Given characters/freqs: encode in binary for max efficiency

* 2 symbols w/ smallest freqs must be at bottom of optimal tree
* construct tree greedily
  - → continually find 2 symbols w/ smallest freqs
  - → make them children of new node

repeat until 1 node remaining




cost: $\sum_{i=1}^{n} f_i l_i$

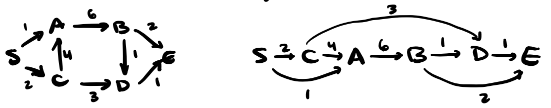
# of bits

# Dynamic Programming

* subproblems to help you build up to main soln
* top-down (recursion) & bottom-up (iteration)

## SHORTEST PATH IN DAG ($s \to t$)

$v \in V$: dist(v) = length of shortest path from s to v



order: topological order $O(n^2)$

base cases: dist(s) = 0, dist(v) = $\infty$ $v \neq s$ is source

* use precomputed dist(u) + length of edge (u,v)

subproblem: $\text{dist}(v) = \min_{(u,v) \in E} \{\text{dist}(u) + \ell(u,v)\}$

## LONGEST INCREASING SUBSEQUENCE

Think abt it as DAG →

subproblem: longest path in DAG length

$L(j) = 1 + \max\{L(i) : (i,j) \in E\}$

return $\max_j L(j)$

* note down prev to keep track of path

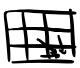
## EDIT DISTANCE * cost of best alignment

* slice the 2 strings for subproblems
* 3 possible cases: add, delete, substitute

x[i] — / — y[j]   x[i] / y[j]   subproblem is edit distance between i and j

$E(i,j) = \min\{1 + E(i-1,j), 1 + E(i,j-1), \text{diff}(i,j) + E(i-1,j-1)\}$

row by row, col by col ordering ✓



Base case: $E(i,0) = i$, $E(0,j) = j$ $O(mn)$

## KNAPSACK

knapsack w/ max capacity W & n items w/ weight $w_1 \cdots w_n$ & dollar $v_1 \cdots v_n$

Repetition: look @ smaller capacities & remove items

$K(w) = \max_{i, w_i \le w} \{K(w - w_i) + v_i\}$

Base case: $K(0) = 0$ $O(nW)$

ordering $1 \to W$ 1D array w $O(n)$ time

No repetition: capacity w, items $1 \cdots j$

$K(w,j) = \max\{\underbrace{K(w - w_j, j-1) + v_j}_{\text{used item } j}, \underbrace{K(w, j-1)}_{\text{did not use item } j}\}$

$O(nW)$: 2D array w/ constant time

## ALL PAIRS SHORTEST PATHS

need distance between all pairs of vertices

* use intermediate nodes
  - ↳ check if intermediate node gives shorter path

$\text{dist}(i,j,k) = \min(\text{dist}(i,k,k-1) + \text{dist}(k,j,k-1), \text{dist}(i,j,k-1))$

i,j: distance between i & j $O(|V|^3)$

k is intermediate

## TRAVELING SALESMAN PROBLEM

Tour that starts + ends @ 1, visits each city once, has minimum total length

* Look at subset of cities S and city $j \in S$

$C(S,j)$ = length of shortest path visiting all cities in S once and starts at 1, ends at j

* need to specify second to last city

$C(S,j) = \min_{i \in S, i \neq j} C(S - \{j\}, i) + d_{ij}$

end @ i and find distance from $i \to j$

$C(S,1) = \infty$ (path cannot start & end @ 1)

$2^n \cdot n$ subproblems, linear $O(n)$ time → $O(n^2 2^n)$

## INDEPENDENT SETS IN TREES

independent set of Graph $G = (V,E)$ if no edges between them

$I(u)$ = size of largest indep set hanging from u

## Linear Programming: constraints + optimal criteria

* optimum usually at vertex of feasible region
* FIGURE OUT VARIABLES FIRST

no optimum:

1. LP bounds are too tight (infeasible)
2. constraints are so loose (unbounded)

| Primal | Dual |
|---|---|
| $\max c^T x$ | $\min y^T b$ |
| $Ax \le b$ | $y^T A \ge c^T$ |
| $x \ge 0$ | $y \ge 0$ |

PRIMAL

$\max x_1 + 6x_2$

constraints:
$x_1 \le 200$
$x_2 \le 300$
$x_1 + x_2 \le 400$
$x_1, x_2 \ge 0$

→ $y_1$, $y_2$, $y_3$ →

DUAL

$\min 200y_1 + 300y_2 + 400y_3$
$y_1 + y_3 \ge 1$
$y_2 + y_3 \ge 6$
$y_1, y_2, y_3 \ge 0$

primal feasible | dual feasible | objective value — duality gap = 0

dual feasible ≥ primal feasible

duality thm: the 2 optima values coincide

### TRANSFORMATIONS

| | |
|---|---|
| $\max c^T x = \min -c^T x$<br>$\min c^T x = \max -c^T x$ | $ax \le b \to ax + s = b$, $s \ge 0$ |
| $ax = b \to ax \le b$, $ax \ge b$ | $x \in R \to x = x_+ - x_-$, $x_+, x_- \ge 0$ |

weak duality: all feasible solns x to primal LP ≤ all feasible solns y to dual LP

strong duality: primal LP opt = dual LP opt

### SIMPLEX

*DOES NOT visit opt*

1. Find feasible region from constraints
2. start at vertex & plug into objective
3. hill climb on vertices until vertex has NO better neighbors $V_a < V_b > V_c$

if chosen vertex not optimal: object

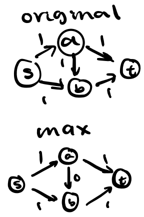
### MATRIX NOTATION

$\max x_1 + 6x_2 \to c = \begin{pmatrix}1\\6\end{pmatrix}$ $x = \begin{pmatrix}x_1\\x_2\end{pmatrix}$

constraints:
$x_1 \le 200$
$x_2 \le 300$
$x_1 + x_2 \le 400$
$x_1, x_2 \ge 0$

$\to \underbrace{\begin{pmatrix}1 & 0\\0 & 1\\1 & 1\end{pmatrix}}_{A} \underbrace{\begin{pmatrix}x_1\\x_2\end{pmatrix}}_{x} \le \underbrace{\begin{pmatrix}200\\300\\400\end{pmatrix}}_{b}$

## MAX FLOW from s to t



original

max

1. Doesn't violate edge capacities: $0 \le f_e \le c_e$ for $e \in E$
2. For nodes u (except s, t): amount of flow entering = leaving
   - ↳ conservation of flow $\sum_{(w,u) \in E} f_{wu} = \sum_{(u,z) \in E} f_{uz}$

$\text{size}(f) = \sum_{(s,u) \in E} f_{su}$ Goal: find max flow val($f^*$)

residual graph finds "leftover" flow

$r(u,v)$: $c_{uv} - f_{uv}$ if $(u,v) \in E$ and $f_{uv} < c_{uv}$

$r(v,u)$: $f_{vu}$ if $(v,u) \in E$ and $f_{vu} > 0$

↳ flow we've already assigned

- distinct edge capacities $\neq$ unique max flow
- $\neq$ unique min cut
- min cut same after multiplied w/ factor
- possible to have directed cycle

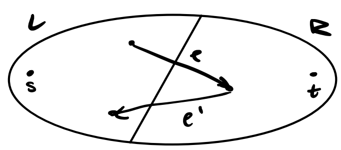
### FORD-FULKERSON

1. Add back edges w/ capacity 0 & copy forward edges into residual graph G'
2. Find valid path from $s \to T$ in residual and push flow = bottleneck
   - ↳ DFS
3. update capacities: subtract flow you pushed from forward edge; add flow you pushed to back edge
4. Repeat 2 and 4 until no path from $s \to t$ is found

$O(f^*(E+V))$

Edmond's Karp: use BFS → $O(V \cdot E^2)$

### MAX-FLOW MIN-CUT: size of max flow = capacity of min-cut



size(f) = capacity(L, R)

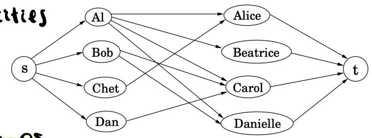
$L \to R$: full capacity ($f_e = c_e$)

$R \to L$: zero flow ($f_{e'} = 0$)

# BIPARTITE MATCHING

① create s node w/ outgoing edges to 1 group
t node w/ incoming edges to other group
② give each edge capacity = 1
③ Perfect matching iff network flow = # of pairs

* if all edge capacities are integers → optimal flow is integral




## Zero sum games

row = player A moves
col = player B moves
values = A's reward for each move
$\boxed{\text{fair}: z=0}$
col player payoff: $M = -M^T$

strategy = vector probabilities of specific moves

$$\sum_{i,j} G_{ij} \cdot \text{Prob}[\text{Row plays } i, \text{col plays } j] = \sum_{i,j} G_{ij} x_i y_j$$

row wants to MAXIMIZE, col wants to MINIMIZE

Best strategy: both players play completely randomly → expected payoff = 0

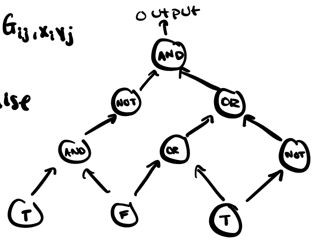
If both play optimally → it doesn't hurt to announce
① A goes first → B minimize A reward → A = max that min
② B goes first → A maximize A reward → B = min of A's moves

| | B1 | B2 |
|---|---|---|
| A1 | a | c |
| A2 | b | d |

A's strat = $[x_1, x_2]$ → max min $\{ax_1 + bx_2, cx_1 + dx_2\}$
B's strat = $[y_1, y_2]$ → min max $\{ay_1 + cy_2, by_1 + dy_2\}$
↳ form of duality

$\boxed{\text{Min-Max Thm}}$ $\max_x \min_y \sum_{i,j} G_{ij} x_i y_j = \min_y \max_x \sum_{i,j} G_{ij} x_i y_j$

**circuit evaluation** = DAG w/ logic gates
- input gates: indegree zero w/ value True / False
- AND gates & OR gates have indegree 2
- NOT gates have indegree 1




Translate into LP
① create var $x_g$ for each gate g

- T: $x_g = 1$
- F: $x_g = 0$
- OR: $x_g \geq x_h$, $x_g \geq x_{h'}$, $x_g \leq x_h + x_{h'}$
- AND: $x_g \leq x_h$, $x_g \leq x_{h'}$, $x_g \geq x_h + x_{h'} - 1$
- NOT: $x_g = 1 - x_h$

② Look at answer $x_o$ from output gate

* all problems solved in polynomial time can REDUCE to LP

## P and NP complete

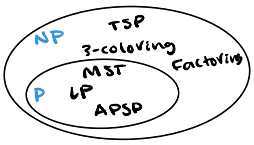
P = "efficiently" solvable in polynomial time $O(n^k)$
NP = SOLUTIONS can be verified in polynomial time
NP-HARD: if all problems in NP reduce to this one
NP-complete: if problem in NP & NP-HARD

3-coloring problem ∈ NP
verify ( input $G=(V,E)$ , soln $c: V \to \{R,G,B\}$ ) ∀ edges $(u,v) \in E$ check $c(u) \neq c(v)$

Factorization ∈ NP
verify ( input N , soln p & q ) ① check $p,q > 1$ ② check $p \cdot q = N$ ← $O(N^2)$

Rudrata cycle / Hamiltonian cycle ∈ NP
Find cycle visiting each node exactly once

Traveling Salesman Problem ← prolly not in NP
Min-TSP → tour w/ min weight $O(n^2 2^n)$
Search-TSP → find tour w/ total weight ≤ budget
↳ NP
Decision-TSP ∈ NP → does there exist tour w/ weight ≤ B
↳ returns yes/no




## DYNAMIC PROG

① subproblems
② recurrence relation
③ ordering

## independent sets

$$I[v] = \max\{1 + \underbrace{\sum I(w)}_{\text{grand children}}, \sum_{\text{children } u} I(u)\}$$
v in set ; v not in set

## SSSP (single source shortest path)

shortest path to every vertex v
* source locked

$$d(v,i) = \min_{u,v \in E} \{\underbrace{dist(u,i-1) + \ell(u,v)}_{\text{go through}}, \underbrace{dist(v,i-1)}_{\text{go straight}}\}$$

ordering $i = 0$
$dist(s,0) = \emptyset$

## string shuffling

$$DP(m,n) = (z_{m+n} == x_m) \wedge f(m-1,n) == \text{true}$$
or
$$= (z_{m+n} == y_n) \wedge f(m,n-1) == \text{true}$$

$DP(0,0) = T$

## Egg drop

n floors, k eggs, min # of drops to find ℓ

$$f(n,k) = \min_{x:1\to n} [\max(\underbrace{1+f(x-1,k-1)}_{\text{breaks}}, \underbrace{1+f(n-x,k)}_{\text{doesn't break}})]$$
n: floor, k: egg, x: any floor

## vertex cover

: set of vertices that covers every edge w/ min weight

$$c(v) = \min\{w(v) + \sum_{x:v\to x} c(x), \sum_{x:v\to x} w(x) + \sum_{\text{grand child } v\to x\to y} c(y)\}$$

include v, call on child ; don't have v → NEED children, call on grandchildren

## Egg drop revisited

max floors to find ℓ w/ x drops, m eggs

$$M(x,k) = M(x-1,k) + M(k-1,k-1) + 1$$

## knightmare

$$f(h,v,w) = \sum_{\substack{v:u,v,w \\ \text{are valid}}} f(h-1,u,v)$$

* look at prev rows & valid patterns: v = h−1 row config, w = h row config

## Matrix multiplication

$$C(i,j) = \min_{i\leq k<j} \{C(i,k) + C(k+1,j) + \underbrace{m_{i-1} \cdot m_k \cdot m_j}_{\text{cost to multiply submatrices}}\}$$
$(A_i \times \cdots \times A_k)$ & $(A_{k+1} \times \cdots \times A_j)$

## common patterns

① input: $x_1\ x_2 \cdots x_n$
subproblem: $x_1\ x_2 \cdots x_i$
$\boxed{x_1\ x_2\ x_3}$ $x_4\ x_5\ x_6$
linear # of subproblems
→ LIS
$L(j) = 1 + \max_{\substack{i<j \\ a_i < a_j}} L(i)$

② input: $x_1\ x_2 \cdots x_n$
: $y_1\ y_2 \cdots y_m$
subproblem: $x_1\ x_2 \cdots x_i$
$y_1\ y_2 \cdots y_j$
$\boxed{x_1\ x_2\ x_3}$ $x_4\ x_5\ x_6$
$\boxed{y_1\ y_2\ y_3\ y_4}$ $y_5\ y_6$
# of subproblems $O(mn)$
→ Edit distance
$E(i,j) = \min\{1 + E(i-1,j), 1 + E(i,j-1), \text{diff}(i,j) + E(i-1,j-1)\}$

③ input: $x_1\ x_2 \cdots x_n$
subproblem: $x_i\ x_{i+1} \cdots x_j$
$x_1\ x_2$ $\boxed{x_3\ x_4\ x_5}$ $x_6$
# of subproblems: $O(n^2)$
→ matrix multiplication
$C(i,j) = \min_{i\leq k<j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$

④ input: rooted tree
subproblem: rooted subtree
# of subproblems n nodes: $O(n)$
→ indep set on trees
$I[v] = \max\{1 + \underbrace{\sum I(w)}_{\text{grand children}}, \sum_{\text{children } u} I(u)\}$
v in set ; v not in set

⑤ Encode state: store info abt state representing subproblem
→ knightmare, TSP

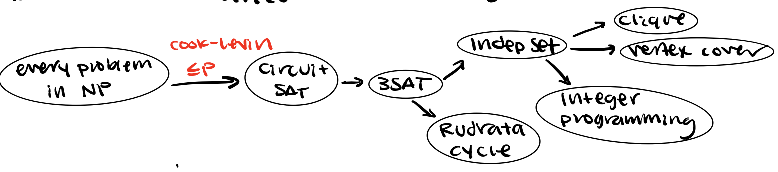
⑥ Diff statuses: info abt whether current state includes/excludes something
→ indep set, k-indep set

# Reductions

Proving A reduces to B

If A true on original → B true on modified

B true on modified → A true on original



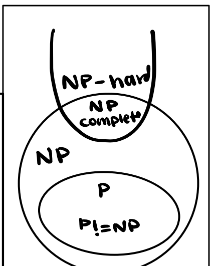

NP-complete: all other problems in NP can be reduced to L

To show problem A = NP-complete

① A ∈ NP (verification algo in P)

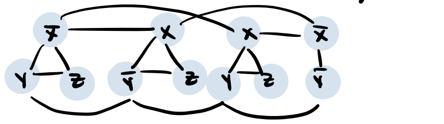
② NP-complete B reduces to A

* if any single NP-complete problem solved in P → every problem in NP in P




## 3 SAT → Indep set

$\ell = (\bar{x} \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$



- construct edges between clauses & between $(a, \bar{a})$
- indep set of size m = # of clauses iff ℓ satisfiable

## Indep set → vertex cover

Graph $G = (V, E)$

S is vertex cover ⟷ V \ S = indep set

Reduction: find indep set → take set of other vertices

## SAT → 3SAT

clauses w/ 3 literals = same

> 3 literals: add more vars & split the clause

$(a \vee b \vee c \vee d) \to (a \vee b \vee x_1)$
$\wedge (\neg x_1 \vee c \vee x_2)$
$\wedge (\neg x_2 \vee d \vee e) \ldots$

## Rudrata Path → Rudrata cycle

For s → t path, add edges (x, s), (x, t) → enforces cycle

## Indep set → clique

An indep set S of graph G = clique C of graph $\bar{G}$ ← complement

Reduction: <G, k> ↦ <$\bar{G}$, k>
(↑ iset, ↑ clique)

## 3SAT → set cover

n+m elements: one for each literal & clause

For each literal $x_i$: create 2 sets

① $S_{i_1} = \{$clauses satisfied by $x_i\} \cup \{x_i\}$

② $S_{i_2} = \{$clauses satisfied by $\bar{x_i}\} \cup \{x_i\}$

Select set cover of size n to indicate all m clauses satisfied

## Rudrata cycle → TSP

Edges w/ weight 1

Edges NOT in OG graph w/ weight 1+α

If TSP has value n: cycle ✓

Else: uses at least 1 edge not in original graph

## Bipartite matching → max flow

Goal: Find max # of matching pairs in bipartite graph

① Add source s and sink t

connect s to all v in L, all v in R to t ] T

w=1 to all edges s→L, R→T

② Run max-flow alg

③ Take all edges saturated between L & R

## NP completeness

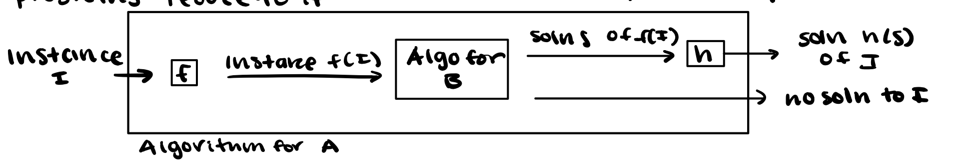
A → B (difficulty)

NP-Hard = if all problems in NP reduce to this one

NP-complete: in NP AND NP-HARD

* SEARCH PROBLEMS = NP
* SEARCH problems in polynomial time = P
* NP-complete if all other search problems reduce to it

P = NP → efficient method to prove any theorem




① 3-SAT: Given set of clauses (1-3 literals) $(\bar{x} \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(\bar{x} \vee y)$ → find satisfying truth assignment

② Independent set: Find g pairwise non-adjacent vertices in graph G

③ Vertex cover: w/ graph G, budget b: find b vertices st endpoint of every edge in cover set

④ Integer LP: feasible linear soln w/ system of linear inequalities

⑤ Rudrata Path: Given vertices s and t in graph G, find path starting @ s → t and going through each vertex exactly once

⑥ Set cover: Given set of elements E, subsets $S_1 \cdots S_m$ w/ budget b: select b subsets to cover E

* can turn optimization problem to search problem bc they reduce to each other

# Approximation Algorithms

Find approximately opt soln to optimization problem A

approximation ratio: $\alpha_A = \max_I \frac{A(I)}{OPT(I)}$ ← as small as possible

minimization: α > 1

maximization: α < 1

## Karger contraction (approximates min cut)

$O(rn^2)$ r = # of times repeat 1-2

① Pick edge uniformly at random & contract along that edge (combine vertices along edge into metanode)


② Repeat until 2 meta vertices remaining: meta vertices = cut

$P[\text{fail to find min-cut}] \leq \left(1 - \frac{1}{\binom{n}{2}}\right)^r \leq e^{-r/\binom{n}{2}}$

## vertex cover (2 approx)

matching = edges w/ no vertices in common

↳ maximal by repeatedly picking disjoint edges

① Find maximal matching M

② S = set w/ both endpoints of edge in M

③ cover has 2|M| vertices

↳ ANY vertex cover has at least size |M|

↳ approx factor = 2

## TSP 2-approx O(n log n)

① compute MST*

② use each edge twice & follow shape of MST → tour w/ length at

| NP-complete | Easy in P |
|---|---|
| 3SAT | 2SAT, HORNSAT |
| TSP | MST |
| Longest path | Shortest path |
| 3D matching | Bipartite matching |
| 3 coloring | |
| Knapsack | Unary knapsack |
| Independent set | Independent set on trees |
| Integer LP | LP |
| Rudrata Path (vertices) | Euler Path (edges) |
| Balanced cut | minimum cut |

YOU CAN DO THIS!!!

# Hashing

* want to pick hash function at random from class of functions (hash family)

* universal hash function: for any 2 data items → $P(\text{collision}) = \frac{1}{n}$ if hash function randomly drawn

① choose table size $n$ to be some PRIME NUMBER that's larger than # of items

② Assume size of domain: $N = n^k$

③ Data item = $k$ tuple of ints mod $N$

$\mathcal{H} = \{h_a : a \in \{0, \dots, n-1\}^k\}$

universal hash family

## Universal hash family

$\forall x, y \in X$ where $x \neq y$

$P_{h \sim \mathrm{unif}(\mathcal{H})}[h(x) = h(y)] \leq \frac{1}{R}$


## Pairwise independent

$\forall x \neq y \in X$, $\forall a, b \in \{1 \cdots R\}$

$P_{h \sim \mathrm{unif}(\mathcal{H})}[h(x) = a, h(y) = b] = \frac{1}{R^2}$

* resulting hash values should be independent

## Examples

① 1-var: $\mathcal{H} = \{h_a : a \in \{0, 1, \dots, n-1\}\}$

$h_a(x) = a \cdot x \bmod n$

② m-vars: $\mathcal{H} = \{h_a : a \in \{0, 1 \dots n-1\}^m\}$

$h_a(\vec{x}) = \sum_{i=1}^{m} a_i \cdot x_i \bmod n$

# Streaming

① cannot store all data ② Read stream once

Algos that work in sublinear space to handle indefinite sequence of data

### Frequency moments

$F_0 = \sum_i m_i^0$ = # of distinct elements

$F_1 = \sum_i m_i^1$ = total # of elements ("heavy hitters")

$F_2 = \sum_i m_i^2 \approx$ variance

# Probability Review

union bound: $P[A \vee B] \leq P[A] + P[B]$

if $A, B$ indep: $P[A \wedge B] = P[A] \cdot P[B]$

$E[X] = \sum P[X = v] \cdot v$

Linearity of expectation: $E[X+Y] = E[X] + E[Y]$

$E[vX] = v \cdot EX$

Independent rand vars: $E[XY] = E[X] \cdot E[Y]$

$\mathrm{Var}\,X = E(X - E[X])^2$

## RESERVOIR SAMPLING

① maintain reservoir that holds current choice of rand sample

② When next element arrives, algo places $s$ in reservoir

↳ discard sample that was alr in it

* Reservoir sampling outputs uniformly random element of the stream

* At end of ith iteration: for $j < i$

$\boxed{P[\text{reservoir} = s[j]] = \frac{1}{i}}$

* sample $t$ elements w/ replacement

↳ $t$ parallel executions

* sample $t$ distinct elements of stream w/o replacement

↳ reservoir of size $t$

Markov's Inequality: $P[X \geq t] \leq \frac{E[X]}{t}$

Chebyshev's Inequality: $P[|X - E[X]| > t\sqrt{\mathrm{Var}\,X}] \leq \frac{1}{t^2}$

Chernoff/Hoeffding Bound: rand vars $\{0,1\}$, $p = E[X_i]$, $\epsilon > 0$

$$P\left[\left|\frac{1}{t} \cdot \sum_{i=1}^{t} X_i - p\right| \geq \epsilon\right] \leq 2e^{-2\epsilon^2 t}$$

$P[\text{estimate has error} \geq \epsilon] \leq \delta$

↳ $t = \left[\frac{1}{2\epsilon^2} \log_e\left(\frac{2}{\delta}\right)\right]$

## Counting distinct elements

① Pick hash function $h: \Sigma \to [0,1]$

② compute minimum hash value $\alpha = \min_i h(w_i)$ by going over stream

③ Output $\frac{1}{\alpha}$

Runs in $O(\log n)$

$E[\min_i h(w_i)] = \frac{1}{k+1}$

Hash values have uniformly random number in $[0,1]$

$P(\text{large} \mid N \leq k) = \frac{k}{B} = \frac{1}{4}$ if $B = 4k$

$P(\text{large} \mid N \geq 2k) \geq \frac{2k}{B}\left(1 - \frac{k}{B}\right) = \frac{3}{8}$ if $B = 4k$

# Zero sum Games LP

max $z$

$M_{1,1} \cdot p_1 + \dots + M_{n,1} \cdot p_n \geq z$

$M_{1,2} \cdot p_1 + \dots + M_{n,2} \cdot p_n \geq z$

$\vdots$

$M_{1,m} \cdot p_1 + \dots + M_{n,m} \cdot p_n \geq z$

$p_1 + p_2 + \dots + p_n = 1$

$p_1, p_2 \cdots p_n \geq 0$

## Heavy Hitters

majority element $a$: $f_a > \frac{n}{2}$

$\ell = 2\log n$  $B = 20$

① initialize $\ell \times B$ array $M$ to all zeros

② initialize $L$ to empty list

③ Pick $\ell$ random functions $h_1 \cdots h_\ell$ where $h_i : \Sigma \to \{1 \cdots B\}$

④ while not end of stream

- read label $x$ from stream
- for $i = 1$ to $\ell$
  * $M[i, h_i(x)]$++
- if $\min_{i=1 \dots \ell} M[i, h_i(x)] > .3n$ → add $x$ to $L$ (if not present)

⑤ return $L$

$P[h(a) = h(b)] = \frac{1}{B}$

$P[\text{good estimate } n \text{ times in a row}] \geq 1 - \frac{1}{n}$