170 Final Review

Sherry Fan

November 2021

Contents

1 Big O, Arithmetic

1.1 Arithmetic

1.1.1 Karatsuba

$$x = x_h|x_l, y = y_h|y_l$$
$$xy = x_h y_h 10^n + (x_h y_l + x_l y_h)10^{n/2} + x_l y_l$$
$$A = x_h y_h, B = x_l y_l, D = (x_l + x_h)(y_l + y_h)$$
$$xy = A10^n + (D - A - B)10^{n/2} + B$$

- Recurrence: T(n) = 3T(n / 2) + O(n)
- Runtime: $O(n^{\log_2 3})$

$$\begin{aligned}
&cn + \frac{3}{2}cn + (\frac{3}{2})^2 cn + ... + (\frac{3}{2})^k cn \\
&= cn(\frac{3}{2} + (\frac{3}{2})^2 + ... + (\frac{3}{2})^k) \\
&= cn(\frac{3}{2} + (\frac{3}{2})^2 + ... + (\frac{3}{2})^{\log_2 n}) \\
&= cn\frac{(3/2)^{\log_2 n+1} - 1}{3/2 - 1} \\
&\approx cn(3/2)^{\log_2 n} \\
&= cn\frac{3^{\log_2 n}}{2^{\log_2 n}} \\
&= c * 3^{\log_2 n} \\
&= c * (2^{\log_2 3})^{\log_2 n} \\
&= c * (2^{\log_2 n})^{\log_2 3} \\
&= c * n^{\log_2 3}
\end{aligned}$$

2 Divide-and-Conquer

2.1 Fibonnaci

2.1.1 Iteration

```
def fib_iter(n):
    a, b = 0, 1
    for i = 2 ... n:
        a, b = b, a + b
    return b
```

Runtime: $\Theta(n)$ flops, $\Theta(n^2)$ total

2.1.2 Fast Matrix Powering

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix}$$

$$A^n \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix}$$

- Repeated squaring to compute A_n
- Multiplication algo runs in $M(n)$

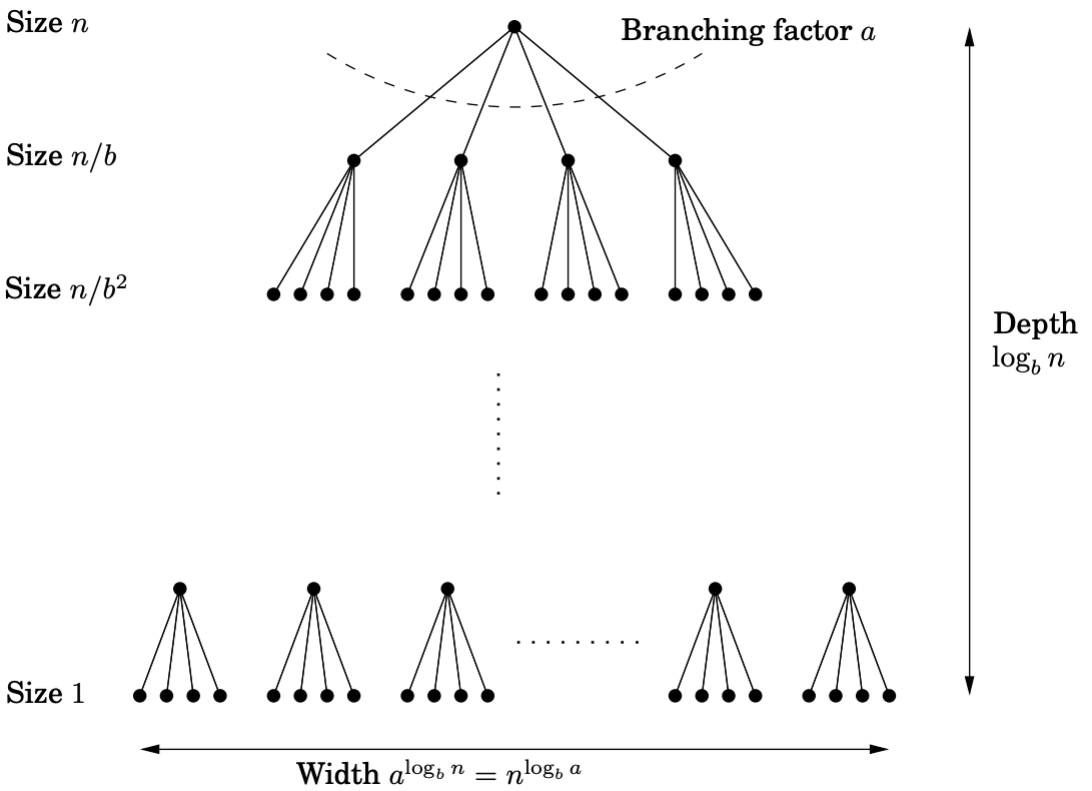
Runtime: $\Theta(\log n)$ flops, $\Theta(M(n) \log n)$ total

2.2 Asymptotic Notation

- $f = O(g)$ $(\leq)$: $\exists c > 0, \forall n f(n) \leq cg(n)$
- $f = o(g)$ $(<)$: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- $f = \Omega(g)$ $(\geq)$: $g = O(f)$
- $f = \omega(g)$ $(>)$: $g = o(f)$
- $f = \Theta(g)$ $(=)$: $f = O(g), f = \Omega(g)$

2.3 Master Theorem

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



$$cn^d + ac(\frac{n}{b})^d + a^2c(\frac{n}{b^2})^d + ... + a^{\log_b n}c(\frac{n}{b^{\log_b n}})^d$$

$$= cn^d(1 + \frac{a}{b^d} + (\frac{a}{b^d})^2 + ... + (\frac{a}{b^d})^{\log_b n})$$

- Case 1: $\frac{a}{b^d} < 1$, first term dominates $\rightarrow \Theta(n^d)$
- Case 2: $\frac{a}{b^d} = 1$, $\rightarrow \Theta(n^d \log n)$
- Case 3: $\frac{a}{b^d} > 1$, last term dominates $\rightarrow \Theta(n^{\log_b a})$

$$cn^d * (\frac{a}{b^d})^{\log_b n}$$

$$= cn^d * \frac{a^{\log_b n}}{b^{d \log_b n}}$$

$$= c * a^{\log_b n}$$

$$= c * (b^{\log_b a})^{\log_b n}$$

$$= c * (b^{\log_b n})^{\log_b a}$$

$$= c * n^{\log_b a}$$

2.4 Mergesort

2.4.1 Recursive Mergesort

```
def MS(A[1...n]):
    B = MS(A[1... n/2])
    C = MS(A[n/2 + 1...n])
    return merge(B, C)
```

Recurrence: T(n) = 2T(n / 2) + O(n)
Runtime: O(n log n)

2.4.2 Iterative Mergesort

```
Q = [queue of size-1 lists]
while Q.size() > 1:
    L1, L2 = Q.pop(), Q.pop()
    L3 = merge(L1, L2)
    Q.push(L3)
return Q[0]
```

Runtime: O(n log n)

- $\log n$ phases when all lists are of size k
- each phase examines each item once, during merge: $O(n)$
- overall is $O(n \log n)$

2.5 Sorting Lower Bound

- comparisons are binary tree, each leaf is a permutation
- runtime proportional to number of comparisons (depth of tree)
- $n!$ possible permutations, 2^d leaves
- $2^d \geq n!$, meaning $d \geq \Theta(n \log n)$

$$n! = 1 * 2 * 3 * ... * n \leq n * n * n * ... * n = n^n$$

$$n! = 1 * 2 * 3 * ... * n \geq (n/2) * (n/2) * ... * (n/2) = (n/2)^{(n/2)}$$

2.6 Selection

```
def quickselect(A, k):
    p = randomly selected pivot
    L = {everything < p}
    R = {everything > p}
    if k = |L| + 1:
        return p
    elif k <= |L|:
        return quickselect(L, k)
    else:
        return quickselect(R, k - |L| - 1)
```

2.6.1 Deterministic QuickSelect

1. Split A into arrays of size 5, find the median of each.
2. Compute the median of these n / 5 medians.
3. Use the median of medians as new pivot.

Runtime: $O(n)$

- median of medians is larger than n / 10 other medians, each of which is larger than 2 items in its size-5 subarray
- finding median of n / 5 size-5 arrays takes O(n) time
- recursively calling median on size n / 5 array takes T(n / 5)

$$T(n) = T(7/10n) + T(n/5) + cn$$

- induction: $T(n) \leq Bn$, choose $B \geq \max(10c, T(1))$

$$T(1) \leq B$$

$$T(n) = T(7/10n) + T(n/5) + cn$$

$$T(n) \leq 7/10Bn + Bn/5 + cn \leq Bn$$

$$cn \leq 1/10Bn$$

$$B \geq 10c$$

3 FFT

3.1 Polynomial Interpolation

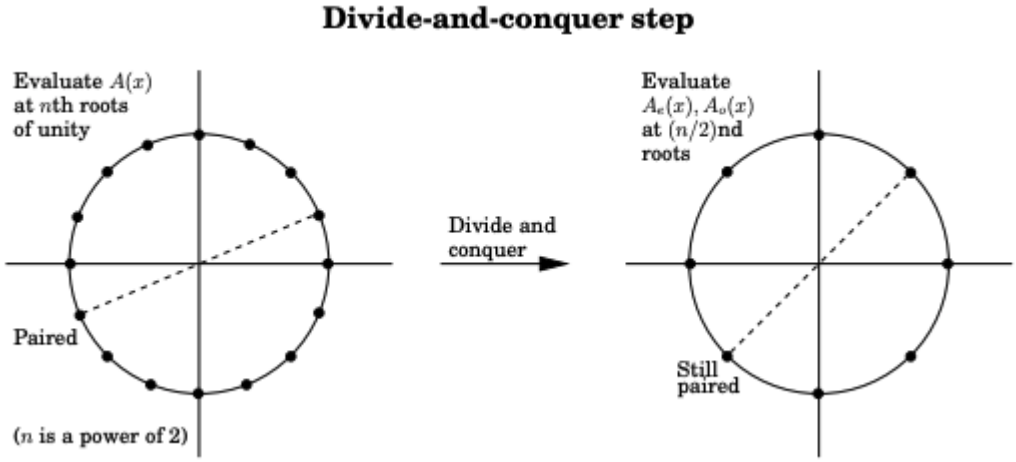
- degree < N polynomial fully determined by evaluation on N distinct points
- Vandermonde matrix:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^{N-1} \\ 1 & x_1 & x_1^2 & ... & x_1^{N-1} \\ ... & ... & ... & ... & ... \\ 1 & x_{N-1} & x_{N-1}^2 & ... & x_{N-1}^{N-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ ... \\ c_{N-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ ... \\ y_{N-1} \end{bmatrix}$$

$$Vc = y, c = V^{-1}y$$

3.2 Roots of Unity

- Nth roots of unity: $2^{2\pi i/N}$



- DFT matrix: $F_{ij} = \omega^{ij}$

$$\begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega_n & \omega_n^2 & ... & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & ... & \omega_n^{2(n-1)} \\ ... & ... & ... & ... & ... \\ 1 & \omega_{n-1} & \omega_{n-1}^2 & ... & \omega_n^{(n-1)(n-1)} \end{bmatrix}$$

3.3 FFT: Algorithm

$$p_0 + p_1 z + p_2 z^2 + ... + p_{N-1} z^{N-1}$$

$$= (p_0 + p_2 z^2 + p_4 (z^2)^2 + ... + p_{N-2} (z^2)^{N/2-1}) + ...z(p_1 + p_3 z^2 + p_5 (z^2)^2 + ... + p_{N-1} (z^2)^{N/2-1})$$

$$= P_e(z^2) + zP_o(z^2)$$

Recurrence: T(N) = 2T(N / 2) + O(N) = O(N log N)

- Evaluate degree-N polynomial on N points = evaluate two degree-N/2 polynomials on N/2 points + linear work to recombine

```
def FFT(A, ω):
    if ω = 1:
        return A(1)
    A_e(ω) = FFT(A_e, n/2)
    A_o(ω) = FFT(A_o, n/2)
    for k = 0 ... n/2 - 1:
        A(w_n^k) = A_e(w_{n/2}^k) + w_n^k A_o(w_{n/2}^k)
        A(w_n^{k+n/2}) = A_e(w_{n/2}^k) - w_n^k A_o(w_{n/2}^k)
```

3.4 Polynomial Multplication w/ FFT

1. Use FFT to compute $\hat{a} = Fa$
2. Use FFT to compute $\hat{b} = Fb$
3. Pointwise multiply to get $\hat{c}_i = \hat{a}_i * \hat{b_i}$
4. $c = F^{-1}\hat{c}$ (coefficients of C(x) = A(x)B(x))
   (a) $F^{-1}\hat{c} = \frac{1}{N}\bar{F}\hat{c} = \frac{1}{N}\overline{F\bar{\hat{c}}}$

$$(F * \frac{1}{N}\bar{F})_{ij}$$
$$= \frac{1}{N}\sum_{k=0}^{N-1} \omega^{ik}\omega^{-kj}$$
$$= \frac{1}{N}\sum_{k=0}^{N-1} \omega^{k(i-j)}$$

when $i \neq j$:

$$= \frac{1}{N}\frac{1-(\omega^{i-j})^N}{1-\omega^{i-j}} = 0$$

when $i = j$:

$$= \frac{1}{N}\sum_{k=0}^{N-1} 1 = 1$$

3.5 Cross-Corr

Inputs: $x = (x_0, x_1, ..., x_{m-1})$, $y = (y_0, y_1, ..., y_{n-1})$
Output: all $n - m + 1$ shifted dot products c_k

$$x(z) = x_{m-1}z^0 + x_{m-2}z^1 + ... + x_{m-i}z^{i-1} + ... + x_0 z^{m-1}$$
$$y(z) = y_0 z^0 + y_1 z^1 + ... + y_j z^j + ... + y_{n-1}z^{n-1}$$
$$Q(z) = (x * y)(z), q_{m-1+k} = c_k$$

4 Graph Decomposition

4.1 Graph Representations

- Adjacency matrix: $A_{ij} = 1$ if $(i, j) \in E$, else 0
  - Weighted: $A_{ij} = w_{ij}$ if $(i, j) \in E$, else ∞
- Adjacency list: array of linked lists, $A[i]$ = all j s.t. $(i, j) \in E$

	Matrix	List
space	n^2 bits	$O(m+n)$ words
edge exists	O(1)	$O(deg(v))$
get neighbors	$\Theta(N)$	$\Theta(dev(v))$

4.2 DFS

```
def dfs(G):
    clock = 1
    visited = [False] * n
    pre, post = int[n], int[n]
    for all v in v:
        if not visited[v]:
            explore(v)

def explore(v):
    visited[v] = true
    pre[v] = clock++
    for each edge (v, u) in E:
        if not visited[v]:
            explore(u)
    post[v] = clock++
```

- Claim: `explore(v)` visits all vertices reachable from v
  Suppose FSOC some vertex v_r not visited on path $v_1, v_2, ..., v_r$. Then some first vertex v_i not visited. `explore(`v_{i-1}`)` is called, so inside of loop:
  Case 1: `explore(`v_i`)` is called since v_i is not visited. Then first line of `explore` sets visited[v_i] to true. Contradiction.
  Case 2: `explore(`v_i`)` not called since v_i already visited. Contradiction.
- DFS runs in $O(V+E)$ time

$$\Theta(n) + \sum_{v \in V} 1 + c * deg(v) = \Theta(n+m)$$

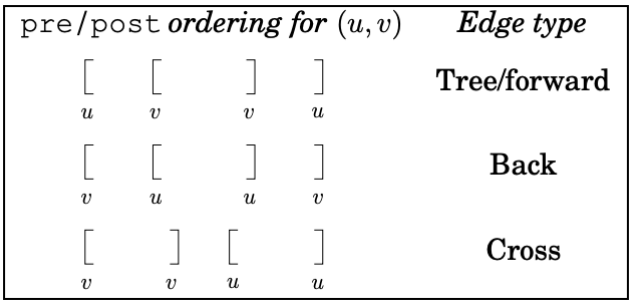
- Identifying connected components in undirected graph:

```
def previsit(v):
    ccnum[v] = cc

def dfs(G):
    ccnum = 0
    visited = [False] * n
    for all v in v:
        if not visited[v]:
            explore(v)
            ccnum++
```

4.2.1 Edge Types

- tree: traversed during DFS
- forward: ancestor to descendant, not a tree edge
- back: descendant to ancestor
- cross: all other edges



- Claim 1: pre, post intervals are either nested or disjoint (must finish exploring children before returning to parent)
- Claim 2: post(u) < post(v) $\iff$ (u, v) is back edge
  $\Longleftarrow$: Suppose (u, v) is a back edge. Then v is ancestor of u in DFS tree so it will return only after the call to `explore(u)` terminates. Thus, higher post.
  $\Longrightarrow$: FSOC, suppose $post(u) < post(v)$ but (u, v) not a back edge. Then possible cases:
  1. v u u v: back edge, but we assumed (u, v) is not back edge. Contradiction.
  2. u v u v: not possible, must be nested or disjoint.
  3. u u v v: during call to `explore(u)`, do not take the edge (u, v). But v is not visited so should traverse that edge. Contradiction.
- Claim 3: back edge $\iff$ cycle
  $\Longrightarrow$: Suppose back edge (u, v). Then take the path in DFS tree $v \to u$ and edge (u, v) to get a cycle.
  $\Longleftarrow$: Suppose some cycle $v_1, v_2, ..., v_r$. Let v_i have highest post in cycle. Then edge (v_{i-1}, v_i) where $post(v_{i-1}) < post(v_i)$, back edge.

5 Paths in Graphs

5.1 Topological Sort

Input: DAG
Algo: reverse postorder
Output: ordering of V s.t. $(u, v) \in E \implies u$ before v

- Claim: highest post $\implies$ source
  FSOC, suppose v has highest post but not a source. Then has incoming edge (u, v). Possible cases:
  1. u u v v: During `explore(u)`, edge (u, v) is not taken. But v is not visited yet, so should take edge. Contradiction.
  2. v u u v: (u, v) is back edge. Not possible in DAG, contradiction.

5.2 SCCs

Def: strongly connected = u has path to v and vice versa
Def: SCC = maximal subset of strongly connected vertices
Algo: Reverse graph, and DFS in decreasing postorder
Runtime: O(m + n)

- Claim: SCC graph is a DAG
  FSOC, suppose not a DAG. Then some cycle. But all vertices in cycle have path to each other, so component is not maximal. Contradiction.

- Claim: u has highest post $\implies$ u in source SCC.
  FSOC, supposed u has highest post but is not in source SCC. Then some other SCC V pointing into u's SCC U.

  1. Suppose DFS visits U first. U cannot have any edges to V (otherwise part of the same SCC), so u finishes before any vertices in v. Then $post(u) < post(v)$, $\forall v \in V$. Contradiction.
  2. Suppose DFS visits V first. Some $v \in V$ has edge to $w \in U$. Then there is a path $v \to w \to u$. u is descendant of v in DFS tree, so u returns before v and has lower post. Contradiction.

5.3 BFS

```
def BFS(G, s):
    dist[1...n] = ∞
    prev[1...n] = null
    vis[1...n] = False
    Q = queue()
    dist[s] = 0
    vis[s] = True
    Q.push(s)
    while Q.size() > 0:
        u = Q.pop()
        for (u, v) in E:
            if !vis[v]:
                vis[v] = True
                prev[v] = u
                dist[v] = dist[u] + 1
                Q.push(v)
    return dist, prev
```

Runtime: O(m + n)

- Lemma 1: $\forall v \in V$, $dist[v] \geq \delta(s, v)$
  Induction on number of pushes.
  Base case: Only s on queue. $dist[s] = 0 \geq \delta(s, s) = 0$.
  Inductive step: Suppose holds after k pushes. Consider $k + 1$st vertex pushed onto Q, v.

$$dist[v] = dist[u] + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

- Lemma 2: $Q = [v_1, ..., v_r] \implies$ (1) $\forall i$, $dist[v_i] \leq dist[v_{i+1}]$, (2) $dist[v_r] \leq dist[v_1] + 1$

- Thm: BFS finds shortest path from s.
  FSOC, suppose some $dist[v] > \delta(s, v)$. Take the v with minimal $\delta(s, v)$.
  Consider shortest path $s \to ... \to v_r \to v$. $dist[v_r] = \delta(s, v_r)$ since v is first wrong vertex (minimal $\delta(s, v)$). When processing v_r in BFS, would assign $dist[v] = dist[v_r] + 1$, unless $dist[v]$ already set by earlier vertex.
  If $dist[v] = dist[v_r] + 1$, $dist[v] = \delta(s, v)$ is set correctly. If $dist[v] = dist[v_i] + 1$ for some earlier v_i, then $dist[v_i] \leq dist[v_r]$ so $dist[v] \leq dist[v_r] + 1 \leq \delta(s, v)$.
  Thus in any case $dist[v] \leq \delta(s, v)$. Contradiction.

5.4 Dijkstra's

```
def dijkstra(G, s)
    heap H
    dist[1...n] = ∞
    prev[1...n] = null
    for v in V:
        H.insert(v, ∞)
    dist[s] = 0
    H.decreaseKey(s, 0)
```

```
while H.size() > 0:
    u = H.delMin()
    for (u, v) in E:
        if dist[u] + w(u, v) < dist[v]:
            prev[v] = u
            H.decreaseKey(v, dist[u] + w(u, v))
            dist[v] = dist[u] + w(u, v)
return dist, prev
```

Runtime: $\Theta(n + m + n * T_{ins} + m * T_{decK} + n * T_{del})$
$O((m + n)logn)$ (binary heap)

- Lemma: any non-infinite `dist` value corresponds to some $s \to u$ path.
- Claim: $\forall u\ dist[u] = \delta(s, u)$
  Let A be all vertices popped of queue. Induct on $|A|$.
  Base case: $|A| = 1$. Only s popped off; set `dist` to 0 at beginning $dist[s] = \delta(s, s)$.
  Inductive step: Suppose true for $|A| = k$. Consider $|A| = k + 1$, when v is popped off. Let u be most recent vertex to set $dist[v]$.
  Consider the path $s \to u \to v$, versus some other path $s \to x \to y \to v$, where $x \in A$, $y \notin A$ (this is the first time it leaves A).

$$\begin{aligned} l(s \to x \to y \to v) &\geq l(s \to x) + w(x, y) \\ &= \delta(s, x) + w(x, y) \\ &= dist[x] + w(x, y) \\ &\geq dist[y] \\ &\geq dist[v] \\ &\geq l(s \to u \to v) \end{aligned}$$

6 MSTs

6.1 Trees

Def: tree = connected, acyclic, $|E| = |V| - 1$ (any 2 implies the 3rd)
Def: spanning tree = subgraph of G w/ all vertices is a tree

- Property 1: removing cycle edge cannot disconnect graph
- Property 2: tree on n nodes has $n - 1$ edges
  Build the tree one edge at a time, starting w/ n nodes. Each iteraction, add edge btwn diff connected components. Each edge decreases connected components by 1. After $n - 1$ edges, single connected component left.
- Property 3: connected, undirected graph w/ $|E| = |V| - 1$ is tree.
  Connected, acyclic, $|E| = |V| - 1 \implies$ tree. Must show acyclic.
  While cycle exists, remove edge from G to get acyclic G'. By Property 1, still connected. G' is a tree so has $|V| - 1$ edges. Then no edges were removed from G; $G = G'$ and G is tree.
- Property 4: Tree iff unique path between nodes.
  $\implies$: Contraposition. More than one path between some $(u, v) \implies$ cycle, so not a tree.
  $\impliedby$: Path between any 2 nodes $\implies$ connected. Unique means acyclic, since cycle has 2 paths between nodes.

6.2 MST Algos

Meta-alg:

```
def MST(G):
    X = {}
    while |X| < n - 1:
        pick  S ⊂ V such that no edge in X crosses cut (S, V\S)
        (a, b) = min-weight edge crossing cut
        X.add((a, b))
    return X
```

- Claim: At all points in time, ∃ MST T containing X.
  Induction on $|X|$.
  Base case: $|X| = 0$. Any set contains the empty set.
  Inductive step: Suppose holds for $|X| = k$. Consider next edge added, $e = (a, b)$. Suppose $X \subseteq T$ before the addition of e but $(a, b) \notin T$.
  T must have other edge $e' = (u, v)$ crossing cut. $w(e') \leq w(e)$ (otherwise, could swap to form lighter MST); since e is minimal $w(e) = w(e')$. Then swapping e, e' forms valid MST T'; thus $X \subseteq T'$.

6.2.1 Prim's

```
def Prim(G, s):
    X = {}
    S = {s}
    repeat n - 1 times:
        (a, b) = min weight edge crossing (S, V\S)
        S.add(a)
        X.add(e)
    return X
```

```
def Prim(G, s):
    X = {}
    H = heap()
    for i = 1...n:
        H.insert(i, ∞)
    from[1...n] = null
    H.decKey(s, 0)
    while H.size() > 0:
        u = H.delMin()
        if u != s:
            X.add((from[u], u))
        for (u, v) in E:
            if v.key > w(u, v):
                H.decKey(v, w(u, v))
                from[v] = u
    return X
```

Runtime: same as Dijkstra's

6.2.2 Kruskal's

```
def Kruksal(G):
    E.sort(key=w(u, v))
    X = {}
    UF = unionfind()
    for e =(a, b) in E:
        if UF.find(a) != UF.find(b):
```

```
            X.add(e)
            UF.union(a, b)
    return X
```

Runtime: $2m * T_{find} + (n-1) * T_{union} + m \log n$

7 Greedy Algos

7.1 Scheduling

Input: collection of jobs in intervals $[x_i, y_i]$
Goal: schedule max num jobs w/o time conflicts
Algo: pick earliest end time
Runtime: $\Theta(n \log n)$
Correctness: Suppose FSOC greedy soln $G = g_i$ not optimal. Then some optimal jobs $S = s_j$. Pick optimal S w/ maximal overlap with G.
Cannot be the case that $s_i = g_i$, $\forall i$ and that s_{i+1}... has more jobs left (since greedy always takes jobs while available), would also take s_{i+1}.
Sort by end times. Must be some first $s_i \neq g_i$.
g_i ends before s_i, so doesn't overlap with any of s_{i+1}.... Also doesn't overlap with any previous jobs since equal between optimal/greedy.
Thus, swap g_i for s_i in S. Still optimal. Contradiction since assumed S has max overlap with G.

7.2 Huffman

Input: alphabet Σ, frequencies f_i
Def: prefix-free = no char encoding is prefix of another
Algo: repeatedly pair lowest-freq chars
Runtime: $\Theta(n \log n)$ with binary heap

7.3 Set Cover

Input: subsets $S_1, ..., S_m$ of $[n]$ Goal: min size subcollection to cover $[n]$

- Thm: Greedy (pick largest left) uses $\leq OPT * \lceil \ln n \rceil$ sets.
  Let A_t be A (the set to cover) after t sets have been picked.
  OPT uses k sets: some set must $|A_t|/k$. Greedy will pick a set covering at least $|A_t|/k$ items.

$$|A_{t+1}| \leq |A_t| * (1 - \frac{1}{k})$$

$$|A_{t+1}| \leq |A_0| * (1 - \frac{1}{k})^{t+1}$$

$$|A_t| \leq n * (1 - \frac{1}{k})^t$$

  Algo is done when $A_t \leq 1$.

$$n * (1 - \frac{1}{k})^t \leq n * (e^{-\frac{1}{k}})^t \leq 1$$

$$e^{-\frac{t}{k}} \leq \frac{1}{n}$$

$$\frac{-t}{k} \leq -\ln n$$

$$\frac{t}{k} \geq \ln n$$

$$t \geq k \ln n$$

8 Union Find

8.1 Disjoint Forest

Represent sets as collections of rooted trees.
Def: path compression = during find, point all child pointers to root.
Def: union by rank = higher-height tree always becomes new root during union.

```
class UnionFind:
    p[1...n]
    r[1...n]

    def makeSet(x):
        p[x] = x
        r[x] = 0

    def find(x):
        if x = p[x]:
            return x
        p[x] = find(p[x])
        return p[x]

    def union(x, y):
        px = find(X)
        py = find(y)
        if px == py:
            return
        if r[px] > r[py]:
            swap(px, py)
        p[px] = py
        if r[px] == r[py]:
            r[py]++
```

- Claim 1: any root node x has $\geq 2^{r[x]}$ items in subtree. Induction on number of operations. Note `find` doesn't change size of trees, so only focus on `union`.
  Base case: single elem with rank 0. $1 \geq 2^0$.
  Inductive step: suppose true after k `unions`. Consider next `union(x, y)`. If height doesn't increase, then trivially true (tree becomes larger, same root rank). Otherwise, if 2 equal-rank roots unioned, new root has rank $r[x] + 1$. By inductive hypothesis, $r[x] = r[y] \geq 2^{r[x]}$. Then new size $\geq 2^{r[x]} + 2^{r[x]} = 2^{r[x]+1}$.

- Claim 2: ranks strictly increase following parent ptrs.

- Claim 3: nodes of rank k $\leq \frac{n}{2^k}$.
  Let elems of rank k be $x_1, ..., x_Q$.
  We claim exists disjoint subsets of n items $S_1, ..., S_Q$ s.t. $\forall i |S_i| \geq 2^k$.

$$Q * 2^k \leq |\bigcup_{i=1}^{Q} S_i| \leq n$$

$$Q \leq \frac{n}{2^k}$$

  Define S_i = items in x_i's tree when $r[x_i]$ became k. Rank only increases during union.
  Consider two sets S_i, S_j. WLOG, assume x_i became rank k first and consider some elt in x_i's subtree z. $z \notin S_j$, since for $z \in S_j$ x_i or some parent of it must be in x_j's subtree. But by claim 2, ranks strictly increase on path to root, and $r[x_i] = k$, so cannot be subtree in x_j when $r[x_j] = k$.
  By claim 1, $|S_i| \geq 2^{r[x_i]} = 2^k$.

- Claim: Disjoint forest with path compression on m `find`, n `union/makeSet` takes $O((m+n)\log^* n)$ time.

Split nodes by rank: $[0,1),[1,2),[2,4),[4,16),...[k,2^k)$. Number of groups: $\log^* n$.
Total runtime is number of parent pointers (u,v) followed. Split into 3 types:

1. v is root.
   Once per find, $O(m)$ total.
2. v not root, u and v in different groups.
   $\log^* n$ possible groups and parent ranks strictly increase, so $\log^* n$ pointers of this type per `find`. m total finds, $O(m \log^* n)$.
3. v not root, u and v in same group.

$$\sum_{u=1}^{n} \text{total ptrs from } u \text{ of type 3}$$

$$= \sum_{u=1}^{n} \sum_{t=0}^{\log^* n} \text{total ptrs from } u \text{ of type 3 while in group } t$$

Rank of $p[u]$ increases each time case 3 occurs. While in group t, can follow at most $2^t - t \leq 2^t$ ptrs. Total ptrs followed (k_u = final group of u):

$$2^0 + 2^1 + 2^2 + 2^4 + 2^{16} + ... + 2^{k_u} = O(2^{k_u})$$

$$\sum_{u=1}^{n} \text{total ptrs from } u \text{ of type 3} = \sum_{u=1}^{n} 2 * 2^{k_u}$$

Divide by final group:

$$\sum_{g=0}^{k} \sum_{u \in g_i} 2 * 2^{k_u}$$

$$\sum_{g=0}^{k} |g_i| * 2 * 2^{k_u}$$

Total items of rank $\geq k_u$: by claim 3,

$$\leq \frac{n}{2_u^k} + \frac{n}{2^{k_u+1}} + ...$$

$$\leq 2 * \frac{n}{2_u^k}$$

$$\sum_{g=0}^{k} \sum_{u \in g_i} 2 * 2^{k_u} = \sum_{g=0}^{k} \sum_{u \in g_i} 4n$$

$$= O(n \log^* n)$$

9 DP

9.1 DAG SSSP

- Subproblem: $f(t)$ = length of SP from s to t
- Soln: $f(t)$, $\forall t \in V$
- Base cases: $f(s) = 0$, $f(t) = \infty$ if $indegree(t) = 0$
- Recurrence: $\min_{a:(a,t)\in E} f(a) + w(a,t)$
- Order: topological
- Time: O(V + E)
- Space: O(V)

9.2 Bellman-Ford

- Subproblem: $f(v, i)$ = shortest path to v using at most i edges
- Soln: $f(v, n-1)$, $\forall v \in V$
- Base cases: $f(v, 0) = \infty$, $\forall v \neq s$; $f(s, 0) = 0$
- Recurrence: $f(v, i) = max_{u:(u,v)\in E} f(v, i-1), f(u, i-1) + w(u, v)$
- Order: $i = 1...n-1$
- Time: O(VE)
- Space: O(V)

```
def BF(G, s):
    T[1...n] = ∞
    T[s] = 0
    for k = 1...n - 1:
        for e = (u, v) in E:
            T[v] = min(T[v], T[u] + w(u, v))
    return T
```

9.2.1 Negative Cycles

- Claim: $\exists$ negative cycle $\iff$ $\exists v$ s.t. $f(v, n) < f(v, n-1)$.
  $\Longleftarrow$: If such v exists, means repeating vertices creates a shorter path. Must be negative cycle.
  $\Longrightarrow$: Contraposition, $f(v, n-1) \leq f(v, n)$, $\forall v$ $\implies$ all cycles nonnegative.
  Consider some cycle $v_1, v_2, ..., v_r$.

$$f(v_{i+1}, n-1) \leq f(v_{i+1}, n) \leq f(v_i, n-1) + w(v_i, v_{i+1})$$

$$\sum_{i=1}^{r} f(v_{i+1}, n-1) \leq \sum_{i=1}^{r} f(v_i, n-1) + w(v_i, v_{i+1})$$

$$0 \leq \sum_{i=1}^{r} w(v_i, v_{i+1})$$

9.3 APSP

- Subproblem: $f(i, j, k)$ = shortest path $i \to j$ using vertices $1...k$
- Soln: $f(i, j, n)$, $\forall i, j \in V$
- Base cases: $f(i, j, 0) = w(i, j)$; $f(i, i, 0) = 0$, $\forall i \in V$
- Recurrence: $f(i, j, k) = \max(f(i, j, k-1), f(i, k, k-1) + f(k, j, k-1))$
- Order: $k = 1...n$
- Time: $O(n^3)$
- Space: $O(n^2)$

9.4 LIS

- Subproblem: $f(i, last)$ = LIS from $i...n$ using items greater than $A[last]$
- Soln: $\max_i f(i+1, i)$
- Base cases: $f(n+1, last) = 0$
- Recurrence:
$$\begin{cases} f(i+1, last) & A[i] \leq A[last] \\ \max(f(i+1, last), f(i+1, i)) & A[i] > A[last] \end{cases}$$
- Order: $k = n...1$
- Time: $O(n^2)$
- Space: $O(n)$

9.5 Edit Distance

- Subproblem: $f(i, j)$ = edit distance from $x[1...i]$, $y[1...j]$
- Soln: $f(m, n)$
- Base cases: $f(i, 0) = i$, $f(0, j) = j$
- Recurrence: $f(i, j) = max(1 + f(i-1, j), 1 + f(i, j-1), f(i-1, j-1) + diff(i, j))$
- Order: $i = 1...n, j = 1...n$
- Time: $O(n^2)$
- Space: $O(n)$

9.6 Independent Sets in Trees

- Subproblem: $f(u)$ = largest independent set rooted at u
- Soln: $max_v f(v)$
- Base cases: $f(v) = 1$ for all leaves v
- Recurrence:
$$f(v) = \max(\sum_{c \in child(v))} f(c), 1 + \sum_{g \in grancdhild(v)} f(g))$$
- Order: reverse level-order
- Time: $O(n + m)$
- Space: $O(n)$

9.7 Knapsack

Given array $A[1...n]$ of (w_i, v_i). Find max value with total weight $\leq W$.

- Subproblem: $f(i, w)$ = max value using items $1...n$ up to weight w
- Soln: $f(n, W)$
- Base cases: $f(0, w) = 0$; $f(i, 0) = 0$
- Recurrence:
$$\begin{cases} f(i-1, w) & w_i > w \\ \max(f(i-1, w), f(i-1, w-w_i) + v_i) & \text{otherwise} \end{cases}$$
- Order: increasing i, W
- Time: $O(nW)$
- Space: $O(W)$

9.8 TSP

n locations, visit all starting and returning to location 1. Minimize total dist.

- Subproblem: $f(i, S)$ = shortest path $1 \to i$ traversing all vertices in S
- Soln: $\min_i f(i, V) + w(i, 1)$
- Base cases: $f(0, w) = 0$; $f(i, 0) = 0$
- Recurrence: $f(i, S) = \min_{j \in S} f(j, S - \{j\}) + w(j, i)$
- Order: increasing $|S|$
- Time: $O(n^2 2^n)$
- Space: $O(n2^n)$

9.9 Chain Matmul

Given n matrices with dimensions $s_1, s_2, ...s_{n+1}$ (A_i is $s_i \times s_{i-1}$), find parentheisization using fewest flops.

- Subproblem: $f(i, j)$ = best way to multiply $A_i ... A_j$
- Soln: $f(1, n)$
- Base cases: $f(i, i) = 0$
- Recurrence: $f(i, j) = \max_{i \le k < j} f(i, k) + f(k + 1, j) + s_i s_{k+1} s_{j+1}$
- Order:

```
for i = 1...n:
    for j = i+1...n:
```

- Time: $O(n^3)$
- Space: $O(n^2)$

10 Linear Programming

$$\max c^T x$$
$$Ax \le b$$

- Example:

$$\max 4x + 5y$$
$$2x \le 10; x + 3y \le 6; 4x + 2y \le 6$$
$$x \ge 0; y \ge 0$$

- Thm: Optimal point is always vtx.

10.1 Duality

$$z_1(2x \leq 10)$$
$$z_2(x + 3y \leq 6)$$
$$z_3(4x + 2y \leq 6)$$
$$4x + 5y \leq x(2z_1 + z_2 + 4z_3) + y(3z_2 + 2z_3) \leq 10z_1 + 6z_2 + 6z_3$$

Creates a new linear program to upper bound original.

$$\max 10z_1 + 6z_2 + 6z_3$$
$$z_1, z_2, z_3 \geq 0$$
$$2z_1 + z_2 + 4z_3 \geq 4; 3z_2 + 2z_3 \geq 5$$

- Thm (weak duality): OPT(primal) $\leq$ OPT(dual)
- Thm (strong duality): OPT(primal) = OPT(dual) (provided primal is bounded, feasible)
- Primal:
$$\max c^T x$$
$$Ax \leq b$$
$$x \geq 0$$
- Dual:
$$\min y^T b$$
$$y^T A \geq c^T$$
$$y \geq 0$$

10.1.1 Duality with Equalities

I = inequalities, E = equalities over n vars. N = nonnegative vars.
Primal:

$$\max c_1 x_1 + ... + c_n x_n$$
$$a_{i1} x_1 + ... + a_{in} x_n \leq b_i, i \in I$$
$$a_{i1} x_1 + ... + a_{in} x_n \leq b_i, i \in E$$
$$x_j \geq 0, j \in N$$

Dual:

$$\min b_1 y_1 + ... + b_m y_m$$
$$a_{i1} y_1 + ... + a_{mj} x_m \geq c_j, j \in N$$
$$a_{i1} y_1 + ... + a_{mj} x_m = c_j, j \notin N$$
$$y_j \geq 0, i \in I$$

10.2 Converting LPs

- Max to min: negate the objective
- Unrestrained to nonnegative: $x = x^+ - x^-$
- Inequality to equality: $\sum_{j=1}^n a_{ij} x_j \leq b_j \implies \sum_{j=1}^n a_{ij} x_j + s_j = b_j, b_j \geq 0$
- $\leq, <$ to $\geq, >$: negate
- Equality to inequality: $\sum_{j=1}^n a_{ij} x_j = b_j \implies \sum_{j=1}^n a_{ij} x_j \leq b_j, \sum_{j=1}^n a_{ij} x_j \geq b_j$

11 Network Flow

Given capacitated directed graph with source s, sink t: find max rate of flow.

11.1 Flow LP

$$\forall e, f_e \geq 0$$

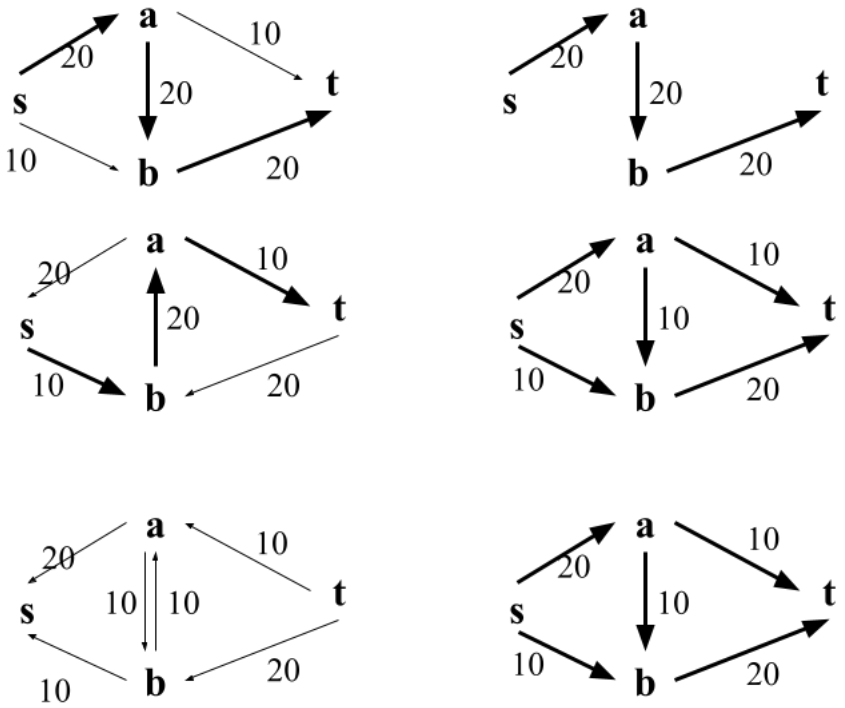
$$\forall e, f_e \leq c_e$$

$$\forall v, \sum_{e=(u,v)} f_e = \sum_{e=(v,w)} f_e$$

$$\max \sum_{e=(s,u)} f_e$$

11.2 Ford-Fulkerson

Find path, push bottleneck amount along path, add equivalent backwards flow in residual graph. Keep augmenting $s-t$ paths until none left.
(Residual, left; flow, right)



Runtime: $O((m+n)F)$. where F is max flow

- Lemma 1: Any $s-t$ flow decomposes into $\leq m$ cycles, paths.
  Induction on m.
  Base case: Single edge $s \to t$. Clearly 1 path. Inductive step: Consider flow on m edges. Do DFS from s. 3 cases:

  1. $s-t$ path. Then find bottleneck edge, subtract from each edge on path. At least one edge has 0 capacity and can be removed. Then $m-1$ edges left, by inductive hypothesis, decomposes into $m-1$ paths/cycles (+ 1 for path just found).
  2. stuck in cycle. Then repeat procedure above. Still m total paths/cycles.
  3. stuck at some vertex v but not in cycle. Not possible due to conservation: inflow to v means some outflow from v.

- Lemma 2: Suppose f is non-maximal flow. Then $f^* - f$ is feasible flow in G_f.
- Thm: Ford-Fulkerson always finds correct path.
  FF stops when no more paths. Suppose FSOC final flow f when no more paths is not maximal. Then $f^* - f$ is feasible in G_f. $f^* - f$ decomposes into paths, cycles, but cycles contribute 0 net flow from s to t. Means that there is still some $s - t$ path in G_f. Contradiction.

11.3 Max-flow Min-cut

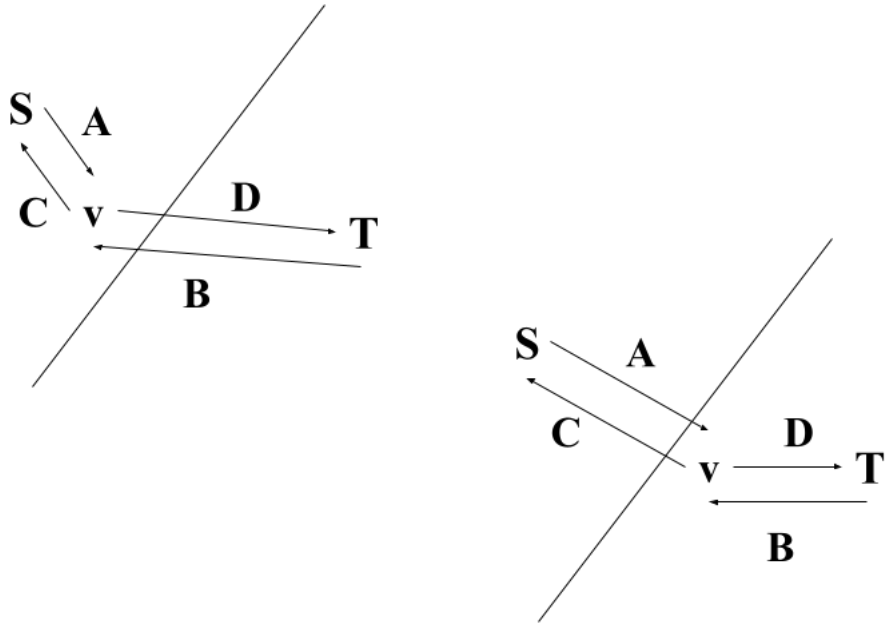
Def: cut = partition of V into nonempty $(S, V - S)$
Def: $s - t$ cut = cut where $s \in S$, $t \in T$

$$val(f^*) = \sum_{e=(s,u)} f_e - \sum_{e=(v,s)} f_e$$

$$u(S) = \sum_{e \in S \times T} u_e$$

$$f(S) = \sum_{a \in S, b \in T} f_{ab} - f_{ba}$$

- Lemma: $\forall S, f(S) = val(f)$
  Induction on $|S|$.
  Base case: $|S| = 1$ (only contians s). True by defn of flow.
  Inductive step: Consider some $v \in S$. Let A = flow into v from vertices in S, B = flow into v from vertices in T, C = flow out of v to vertices in S, D = flow out of v to vertices in T. By conservation, $A + B = C + D$.
  Move v to T side of cut. Ignore edges not involving v (no change). Old flow $D - B$ is equal to new flow $A - C$. $f(S) = f(S - \{v\}) = val(f)$ by inductive hypothesis.



- Thm (max-flow/min-cut): $val(f^*) = u(S^*)$, S = vertices reachable from v in G_f
  1. $val(f^*) \leq u(S^*)$
     Show that $\forall f, S$, $val(f) \leq u(S)$. Then clearly true for f^*, S^*.

$$val(f) = f(S) = \sum_{a \in S, b \in T} f_{ab} - f_{ba} \leq \sum_{a \in S, b \in T} f_{ab} \leq \sum_{a \in S, b \in T} u_{ab} = u(S)$$

2. $val(f^*) \geq u(s^*)$
   Show that $\exists f, S$ s.t. $val(f) = u(S)$. $val(f^*) \geq val(f) = u(S) \geq u(S^*)$.
   Let f be max flow. Consider G_f, and define S = all vertices reachable from s in G_f.
   Any edge from $S \to T$ must have 0 leftover capacity (otherwise some vtx in T reachable from S, so should have been included in S). Thus flow from $S \to T = u(S)$.
   Any edge from $T \to S$ must have 0 flow. Otherwise, would be back edge with nonzero capacity from S to T in G_f, contradicting defn of S. Thus, flow from $T \to S = 0$.

$$val(f) = f(S) = u(S) - 0 = u(S)$$

12 Zero-Sum Games

12.1 Strategies

1. Pure: always pick same action
2. Mixed: fixed probability distribution

	L	R
T	5	-3
B	-1	1

$$\max_x \min(5x_1 - x_2, -3x_1 + x_2)$$

$$z = \min(5x_1 - x_2, -3x_1 + x_2)$$

$$\max z$$

$$z \leq 5x_1 - x_2; z \leq -3x_1 + x_2$$

$$x_1, x_2 \geq 0; x_1 + x_2 = 1$$

$$x = (1/5, 4/5) \implies 5(1/5) - 1(4/5) = 1/5, -3(1/5) + 1(4/5) = 1/5$$

$$\min_y \max(5y_1 - 3y_2, -y_1 + y_2)$$

$$w = \max(5y_1 - 3y_2, -y_1 + y_2)$$

$$\min w$$

$$w \geq 5y_1 - 3y_2; w \geq -y_1 + y_2$$

$$y_1, y_2 \geq 0; y_1 + y_2 = 1$$

$$y = (2/5, 3/5) \implies 5(2/5) - 3(3/5) = 1/5, -1(2/5) + 1(3/5) = 1/5$$

12.2 General ZSG

$$\max_x \min_y \sum_{j=1}^{n} (\sum_{i=1}^{m} x_i U(i,j)) y_j$$

- Thm: best response is always pure strategy.
- Thm: LPs of two players are duals, and always same value (equilibrium).

13 Multiplicative Weights

n experts. Expert i incurs loss $l_i^{(t)}$ on day t. Pick expert i w.p. $x_i^{(t)}$. Total loss: $L = \sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} l_i^{(t)}$. Regret $= L - \min_{i \in [n]} \sum_{t=1} l_i^{(t)}$ – best fixed expert.

- Strategy 1: Choose same expert every day.
  Worst case: chosen expert has loss 1, some other expert has loss 0 every day. R = T.
- Strategy 2: Choose majority opinion.
  Suppose each expert has loss 1 except 1 expert always correct. R = T.
- Strategy 3: Pick uniform random expert.

$$E[R] \leq (1 - 1/n)T$$

$$E[L] = \sum_{t=1}^{T} \sum_{i=1}^{n} \frac{1}{n} l_i^{(t)}$$

$$= \sum_{t=1}^{T} \frac{1}{n} l_{i^*}^{(t)} + \sum_{j \neq i*} \frac{1}{n} l_j^{(t)}$$

$$\leq \sum_{t=1}^{T} l_{i^*}^{(t)} + \sum_{j \neq i^*} \frac{1}{n} l_j^{(t)}$$

$$\leq L^* + \frac{n-1}{n} T$$

$$E[R] = E[L] - L^* \leq \frac{n-1}{n} T$$

- Strategy 4: Pick best expert so far.
  Can still achieve $E[R] \approx (1 - 1/n)T$.

experts	l_1	total loss	l_2	total loss	l_3	total loss	l_4
1	0	0	1	1	0	1	0
2	1/3	1/3	0	1/3	1	4/3	0
3	2/3	2/3	0	2/3	0	2/3	1

Any particular expert wrong T/n times. Total loss $= T$. $E[R] = T - T/n$.

13.1 Hedge Algorithm

$$\epsilon \in (0, 1/2]$$

$$w_i^{(1)} = 1$$

$$x_i^{(t)} = \frac{w_i^{(t)}}{\sum_{j=1}^{n} w_j^{(t)}}$$

$$w_i^{(t+1)} = w_i^{(t)} * (1 - \epsilon)^{l_i^{(t)}}$$

- Lemma 1: $W_{t+1} \geq (1 - \epsilon)^{L^*}$

$$W_{t+1} \geq w_{i^*}^{(t+1)}$$

$$= \prod_{t=1}^{T} (1 - \epsilon)^{l_i^{(t)}}$$

$$= (1 - \epsilon)^{\sum_{t=1}^{T} l_i^{(t)}} = (1 - \epsilon)^{L^*}$$

- Lemma 2: $W_{t+1} \leq n * \prod_{t=1}^{T}(1-\epsilon L_t)$
  Show $W_{t+1} \leq (1-\epsilon L_t)W_t$. Then induction (and $W_1 = n$).

$$W_{t+1} = \sum_{i=1}^{n} w_i^{(t)} * (1-\epsilon)^{l_i^{(t)}}$$

$$\leq \sum_{i=1}^{n} w_i^{(t)} * (1-\epsilon l_i^{(t)})$$

$$= \sum_{i=1}^{n} x_i^{(t)} W_t * (1-\epsilon l_i^{(t)})$$

$$= W_t(\sum_{i=1}^{n} x_i^{(t)} - \epsilon \sum_{i=1}^{n} x_i^{(t)} l_i^{(t)})$$

$$= W_t(1-\epsilon L_t)$$

- Thm: $E[R] \leq \epsilon T + \frac{\ln n}{\epsilon}$; $\epsilon = \sqrt{\frac{\ln n}{T}} \implies E[R] \leq 2\sqrt{T \ln n}$

$$(1-\epsilon)^{L^*} \leq n * \prod_{t=1}^{T}(1-\epsilon L_t)$$

$$L^* \ln(1-\epsilon) \leq \ln n + \sum_{t=1}^{T} \ln(1-\epsilon L_t)$$

$$L^*(-\epsilon - \epsilon^2) \leq \ln n + \sum_{t=1}^{T} -\epsilon L_t$$

$$L^*(-1-\epsilon) \leq \frac{\ln n}{\epsilon} + \sum_{t=1}^{T} L_t$$

$$L - L^* \leq \frac{\ln n}{\epsilon} + \epsilon L^*$$

$$R \leq \frac{\ln n}{\epsilon} + \epsilon T$$

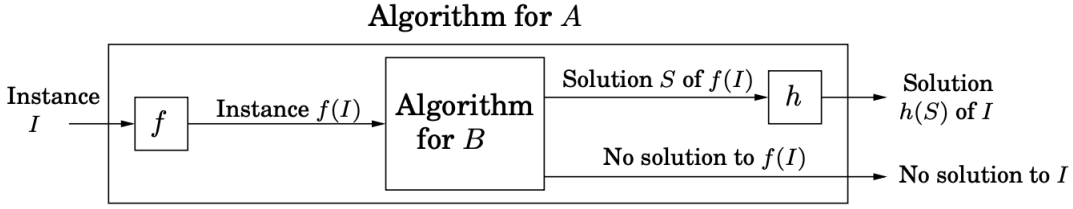
13.1.1 General Losses

Normalize $l'^{(t)}_i = \frac{l_i^{(t)} - a}{b-a}$.
$E[R] \leq (b-a)\epsilon T + \frac{\ln n}{\epsilon}$

14 Reductions

Computational problems A, B. $A \to B$ (A reduces to B) means:

1. Efficient algorithm for B $\implies$ efficient algorithm for A.
2. No efficient algorithm for A $\implies$ no efficient algorithm for B.



14.1 Max Bipartite Matching

Def: Matching = subset of edges $e \in L \times R$ where no two edges share common vertex
Reduce to max flow. Add directed edges with capacity 1:

- (s, l), $\forall l \in L$
- (l, r), $\forall e = \{l, r\} \in L \times R$
- (r, t), $\forall r \in R$

Proof:

1. If matching exists, maps to some integral flow in modified graph where $|M| = val(f_M)$.
   Consider any matching. Push 1 unit $s \to l$ for each l that in the matching, 1 unit $r \to t$ for each r in matching. Between (l, r) push 1 unit for matched pairs.
   A valid flow, since each l only has 1 unit incoming from s, one unit outgoing to match; each r has 1 unit incoming from match and 1 unit outgoing to t. Total outgoing flow from s is 1 per pair. Thus $|M| = val(f_M)$.
2. If integral flow in modified graph exists, maps to matching solution where $val(f) = |M_f|$.
   Pair up items corresponding to (l, r) with 1 unit flow to form matching. Valid matching since each l only has 1 unit incoming, so can only go to one r. $val(f)$ = outgoing units from s = number of l that are matched = $|M_f|$.

Runtime: $O(mn)$

14.2 CVP

Given boolean ckt C with n input bits x_i, compute output bit o.
Algo: topologically sort, then compute from left to right.
Proof sketch: any $A \in P$ has efficient reduction to CVP.
$\exists$ poly-time algo alg_A for A. Given instance a, uses $\leq T = poly(n)$ mem/time.
Make T ckts, one per timestep. Output for C_t = inputs for C_{t+1}. Each ckt represents mem state of alg_A at time t.

14.2.1 CVP $\to$ LP

Introduce one var per gate z_g and input bit z_i.

- $\forall g$, add constraint $0 \leq z_g \leq 1$
- $\forall$ input gates $x_j = b$, add constraint $z_j = b$
- for OR gates $g_1 \wedge g_2 = g$: $z_g \leq z_{g_1}; z_g \leq z_{g_2}; z_g \geq z_{g_1} + z_{g_2} - 1$
- for AND gates $g_1 \vee g_2 = g$: $z_g \geq z_{g_1}; z_g \geq z_{g_2}; z_g \leq z_{g_1} + z_{g_2}$
- for NOT gates $g = g_1$: $z_g = 1 - z_{g_1}$

14.3 Matmul $\leftrightarrow$ Mat Inversion

14.3.1 Matmul $\to$ Inversion

To get $A \times B$, create $M = \begin{bmatrix} I & -A & 0 \\ 0 & I & -B \\ 0 & 0 & I \end{bmatrix}$. Then $M^{-1} \begin{bmatrix} I & A & AB \\ 0 & I & B \\ 0 & 0 & I \end{bmatrix}$

14.3.2 Inversion → Matmul

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \times \begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}$$

Let $Y = CA^{-1}$, $S = D - CA^{-1}B$.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{bmatrix} \times \begin{bmatrix} I & 0 \\ -Y & I \end{bmatrix}$$

To invert $n \times n$ matrices, must recurse on two $n/2 \times n/2$ matrices (A^{-1}, S^{-1}) and constant number of matrix multiply/add/subtract. Suppose matmul takes $O(n^w)$ time:

$$T(n) = 2T(n/2) + O(n^w) = O(n^w)$$

15 Search Problems

15.1 Decision vs Search

`decide(R)`: given x, does $\exists w$ s.t. $(x, w) \in R$.
`search(R)`: given s, return w s.t. $(x, w) \in R$.
`decide` → `search`: just run search and see if solution if outputted
`search` → `decide`: learn w bit by bit.
Def: verifier = some algo given x, w

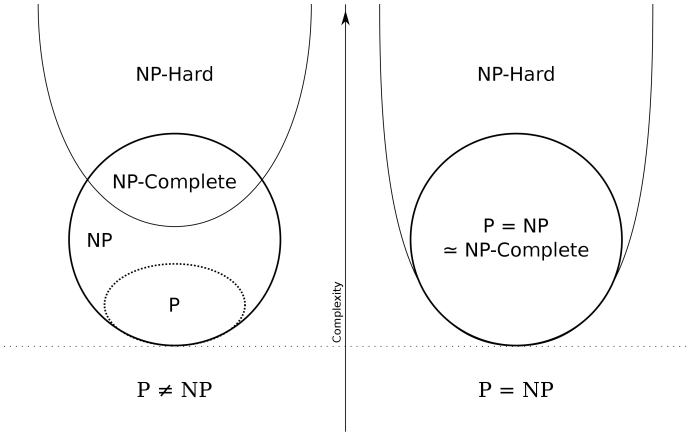
$$V_R(x, w) = \begin{cases} 1 & (x, w) \in R \\ 0 & (x, w) \notin R \end{cases}$$

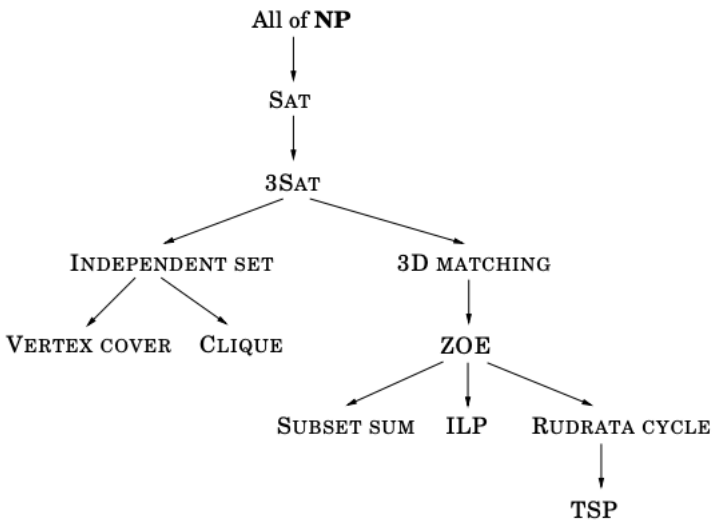
15.2 Complexity Classes

Boolean problems, specified by set of inputs that return True.
P: set of all problems decidable in poly time
NP: exists poly-time verifier

1. $x \in L$: $\exists$ poly-size w s.t. $V_L(x, w) =$ true.
2. $x \notin L$: $\forall w$, $V_L(x, w) =$ false.

NP-hard: $\forall B \in NP$, $B \to R$ (may not be in NP itself).
NP-complete: NP-hard and $R \in NP$





15.3 CSAT

x = some ckt with AND, OR, NOT
$(x, w) \in R$ if output bit is 1
Given ckt C, does $\exists$ input s.t. $C(w) = 1$.

1. CSAT $\in$ NP
   Just evaluate in topological order.

2. CSAT $\in$ NP-hard.
   Take verifier and create ckt C_b s.t. $C_B(w) = V_B(x, w)$. Feed C_B into CSAT.

15.4 CSAT $\rightarrow$ SAT

1. SAT $\in$ NP
   Just evaluate formula (linear time).

2. CSAT $\in$ NP-hard.
   Create one var per wire (input bit/gate output). Find satisfying assignment.

   (a) z = True $\rightarrow (z)$
   (b) z = False $\rightarrow (\bar{z})$
   (c) z = x OR y $\rightarrow (z \vee \bar{x}) \wedge (z \vee \bar{y}) \wedge (\bar{z} \vee x \vee y)$
   (d) z = x AND y $\rightarrow (\bar{z} \vee x) \wedge (\bar{z} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$
   (e) z = NOT x $\rightarrow (z \vee x) \wedge (\bar{z} \vee \bar{x})$

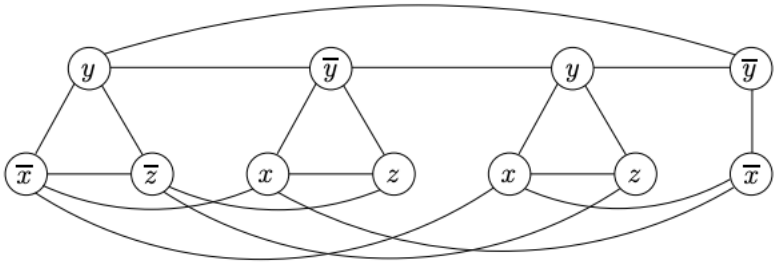
   Force output o to be true: (o).

15.5 3SAT $\rightarrow$ Indep Set

Given undirected G, int k: exists indep set of size $\geq k$?
Def: indep set = $S \subset V$ s.t. no 2 elements share an edge.

Create one clique per clause and edges between variables in same clause, as well as edges btwn var

and negation. Ensures one variable set to true per clause, and no variable and negation both set to true.



15.6 Indep Set → Vertex Cover

Does there exist vertex cover of G of size $\leq k$?
Def: vtx cover = $\forall e \in E$, $\exists v \in S$ s.t. e touches v

Claim: S is vtx cover $\iff$ $V - S$ is indep set.
$\implies$: FSOC, suppose S is vtx cover but $V - S$ is not indep set. Then some $u, v \in V - S$ shares an edge. But then neither u, v are in S, so edge (u, v) not covered. Contradiction.
$\impliedby$: FSOC, suppose $V - S$ is indep set but S is not vtx cover. Then some edge (u, v) not covered by S, which means both $u, v \in V - S$. But then $V - S$ is not indep set. Contradiction.

Reduction: Indep set of size k → vertex cover of size $n - k$.

15.7 Indep Set → Clique

Given k, G, exists clique of size k?
Def: clique = complete subgraph
Reduction: take edge complemenet of graph. See if exists independent set of size k.

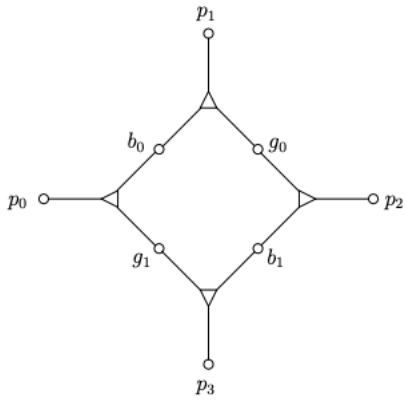
15.8 3SAT → 3D-Matching

Given n individuals in each of B, C, D and compatible triples T, exists subset of T covering all individuals once each?

15.8.1 3SAT → 3SAT$_{L\leq 2}$

Def: 3SAT$_{L\leq 2}$ = 3SAT where each literal only appears at most 2 times.
Reduction: Remove any var only appearing positively/negatively – assign it.
Suppose some var appears as x or $\bar{x}$ $t \geq 3$ times. Create $x_1, x_2, ..., x_t$. Replace ith occurence with x_i. Then add constraint: $(x_1 \vee \bar{x_2}) \wedge (x_1 \vee \bar{x_2}) \wedge ... \wedge (x_n \vee \bar{x_1})$.

15.8.2 $\text{3SAT}_{L\leq 2} \rightarrow$ 3DM



Create one gadget per literal. Possible triples in gadget: (b_0, p_1, g_0) and (b_1, p_3, g_1) (top/bottom) or (b_0, p_0, g_1) and (b_1, g_0, p_2) (left/right). Let taking top/bottom indicate True, left/right indicate False.
Introduce another b_c, g_c per clause. Connect to p_{x0} or p_{x2} if positive literal, p_{x1} or p_{x3} if negative. (Each variable only appears as positive/negative at most 2x, so will not "use up" all p.
Suppose m clauses, n variables. $4n$ (# of p) $- 2n$ (# of b, g for variables) $- m$ (# of b, g for clauses) $= 2n - m$. So create $2n - m$ dummy b, g that can match any p.

15.9 3DM $\rightarrow$ ZOE

Given $A \in \{0,1\}^{m \times n}$, exists $x \in \{0,1\}^n$ s.t. $Ax = 1$?
Reduction: Create $3n \times m$ matrix. m variables: $x_i = 1$ if ith triple is chosen.
Let $a_{ij} = 1$ if ith individual in jth triple. Chooses triples such that each individual is in exactly 1 triple.

15.10 ILP $\rightarrow$ ZOE

LP with integrality constraints:

$$c^T x \geq k$$

$$Ax \leq b$$

$$\forall i \in S, x_i \in \mathbb{Z}$$

Reduction: $k = -\infty$, $b = 1$. Add constraint that all $x_i \in \{0,1\}$

15.11 Rudtradata Path $\rightarrow$ Rudrata Cycle

Exists $s - t$ path passing through each vtx exactly once? For $s - t$ path, add edges $(x, s), (x, t)$.

1. Rudrata path has soln $\implies$ modified graph has Rudrata cycle.
   In modified graph, take (x, s) then $s - t$ path through all other vertices, then (t, x). Visits all vertices and returns to start, so valid Rudrata cycle.

2. Modified graph has Rudrata cycle $\implies$ Rudrata path has soln.
   Remove $(x, s), (x, t)$ from the graph. Rest of cycle visits all vertices between s and t, so is valid Rudrata path.

15.12 SAT $\rightarrow$ 3SAT

Suppose some clause with more than 3 literals: $(a_1 \vee a_2 \vee ... \vee a_k)$. Introduce $y_1 ... y_{k-3}$ and add clauses $(a_1 \vee a_2 \vee y_1) \wedge (\bar{y_1} \vee a_3 \vee y_2) \wedge (\bar{y_2} \vee a_4 \vee y_3) ... (\overline{y_{k-3}} \vee a_{k-1} \vee a_k)$.
If new clauses satisfied, then at least one a_i is true $\rightarrow$ original clause satisfied.
If at least one a_i true, then set $y_1, ..., y_{i-2}$ to true, rest to false.

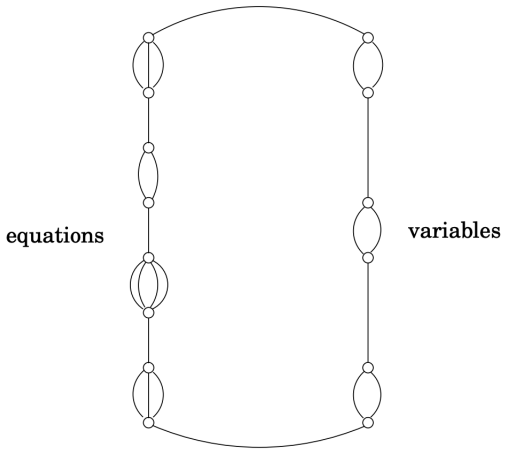
15.13 ZOE → Subset Sum

Given integers, exists subset summing up to k?
Reduction: Interpret each column as binary integer for subset sum, in base $n + 1$ (to avoid carry).

15.14 ZOE → Rudrata Cycle

15.14.1 ZOE → Rudrata Cycle with Paired Edges

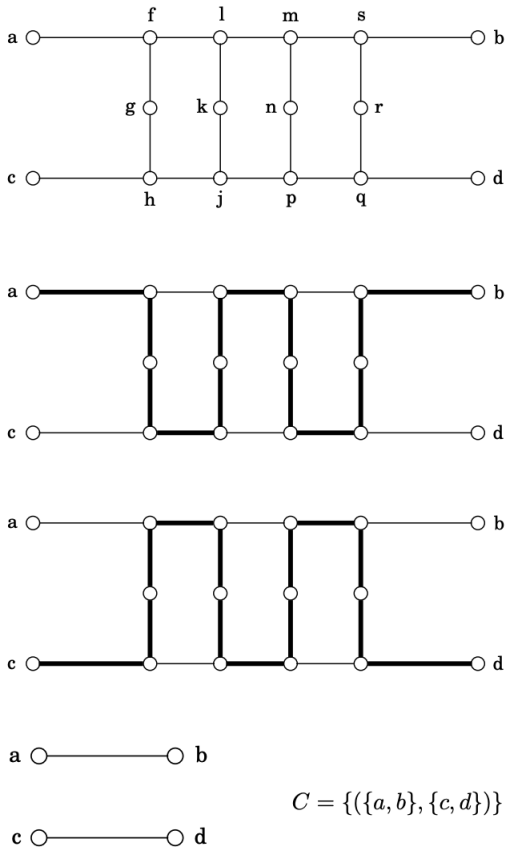
Def: Rudrata cycle with edge pairs, where can only use exactly one edge in a pair.
Reduction: Equations on left, variables on right. Each set of parallel edges must choose one value for variable, and one variable to set to 1 in each equation.



To enforce consistency of variable value and equations, add edge pairs ($x_i = 0$, equation chooses x_i).

15.14.2 Rudrata Cycle with Paired Edges → Rudrata Cycle

Reduction: Add gadget per pair. Ensures only one edge is chosen, since only 2 ways to complete cycle.



15.15 Rudrata Cycle → TSP

Reduction: Let edges in original graph have weight 1. Edges not in original graph have dist $1 + \alpha$. If TSP has value n, Rudrata cycle exists. Otherwise, uses at least one edge not in original graph: $n - 1 + 1 + \alpha = n + \alpha$.

16 NP-Completeness Strategies

16.1 Exact Methods

- backtracking: end recursion early if soln definitely fails
- branch and bound: cut off recursive subtrees if any completion cannot be optimal

16.2 Heuristics

- local search: start with random soln, iterative adjustments until local optimum
- simulated annealing: allow small probability of worse move

16.3 Approx Algos

Def: approx ratio α = cost $\leq \alpha * OPT$ (min), cost $\geq OPT/\alpha$ (max)

16.3.1 Vtx Cover

minimize $S \subseteq V$ such that every edge touches at least 1 vtx

- Strat 1: special case of set cover (vertices = sets, edges = universe) $\rightarrow \ln m = 2 \ln n$ approx.

- Strat 2: greedy, take both endpoints of uncovered edges until done. $|S| \leq 2 * OPT$
  Proof: S is a 2-approx.
  Consider edges e not covered, which forced addition to S. Call this M.
  M is maximal matching. Edges don't have common vertices (since taking both endpoints of a edge also covers any neighbors). Also maximal, since M is a vertex cover so adding any edge causes edges to share vertices.
  Any vtx cover must take at least 1 vtx from each edge in matching, so $|S| \geq |M|$. $OPT \geq |M|$.
  Greedy algo takes $|S| = 2|M|$.

$$OPT \leq |S| = 2|M| \leq 2|OPT|$$

- Strat 3: ILP
  For any edge $(u, v), x_u + x_v \geq 1$.
  All $x_i \in \{0, 1\}$.
  $\min \sum_{u \in V} x_u$
  Relax to normal LP, and solve. Round up/down. Guaranteed to take at least 1 vtx per constraint, so valid vtx cover.
  At most double the opt of the LP, so final soln $|S| \leq 2 * OPT(LP)$.
  $OPT(LP) \leq OPT(ILP) = OPT(VC)$.

$$|S| \leq 2 * OPT(LP) \leq 2 * OPT(VC)$$

16.3.2 Metric TSP

OPT $\geq$ OPT with last edge removed $\geq$ spanning tree $\geq$ MST.
compute MST T^* and use shortcut DFS tour. DFS tour is at most 2x MST, which is at most 2x OPT. Assumes triangle inequality (shortcut DFS is shorter than MST tour).

16.3.3 General TSP

No efficiently computable approx for general TSP unless P = NP. Otherwise, use Rudrata cycle reduction where $\alpha = cn$, where c is approximation ratio. If cycle exists, has cost at most cn. Otherwise, has cost $n + cn \geq cn$.

16.3.4 Clustering

Partition into k clusters of smallest possible diameter.
Algo: Repeatedly choose furthest point to be next center.
Claim: Greedy is 2-approx.
Proof: Let x be furthest point from all centers, with dist r from its center. Every other point has dist $\leq 2r$ from its center, so diameter of any cluster $\leq 2r$.
$u_1, u_2, ..., u_k, x$ are all at least dist r from each other. 2 of them must be in same cluster, so optimal diameter is at least r.

16.3.5 Knapsack

Algo: Rescale values by $\hat{v_i} = \lfloor v_i * \frac{n}{\epsilon v_{max}} \rfloor$.
Runtime: $O(n^3/\epsilon)$ Proof: $(1 - \epsilon)$ approximation.
Optimal subset S with value K^* has rescaled value

$$\sum_{i \in S} \hat{v_i} \geq \sum_{i \in S} v_i * \frac{n}{\epsilon v_{max}} - 1 \geq K^* \frac{n}{\epsilon v_{max}} - n$$

So rescaled optimal $\hat{S}$ has at least that value.

$$\sum_{i \in \hat{S}} v_i \geq \sum_{i \in \hat{S}} \hat{v_i} * \frac{n}{\epsilon v_{max}} \geq (K^* \frac{n}{\epsilon v_{max}} - n) * \frac{n}{\epsilon v_{max}} = K^* - \epsilon v_{max} \geq K^*(1 - \epsilon)$$

17 Randomized Algos

Def: Las Vegas = always correct, runtime is random
Def: Monte Carlo = always fast, correctness is random

17.1 Quicksort

Comparison-based sort. Partition around random pivot and recurse on each half.
Claim 1: Runtime proportional to number of comparisons.
Claim 2: Never compare same elts twice.

$$T = C * \sum_{i<j} X_{ij}$$

X_{ij} is indicator for if i, j ever compared.

$$E[T] = E[\sum_{i<j} X_{ij}] = \sum_{i<j} E[X_{ij}] = \sum_{i<j} P(X_{ij} = 1)$$

When choosing pivot $<$ i or $>$ j, nothing happens. If i $\leq$ pivot $\leq$ j, then either compare i, j or don't. In this case, $X_{ij} = 1$ iff pivot is i or j. Thus, $P(X_{ij} = 1) = \frac{2}{j-i+1}$.

$$\sum_{i<j} \frac{1}{j-i+1}$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j-i+1}$$

$$= (\frac{1}{2} + \frac{1}{3} + ...\frac{1}{n}) + (\frac{1}{2} + \frac{1}{3} + ...\frac{1}{n-1}) + ... + \frac{1}{2}$$

$$= (n-1)\frac{1}{2} + (n-2)\frac{1}{3} + ... + 1\frac{1}{n}$$

$$= n * (\frac{1}{2} + \frac{1}{3} + ...\frac{1}{n})$$

$$\leq n \ln n$$

17.2 Frievald's

Verify matmul $C = A * B$ faster than $O(n^3)$.
Algo: Pick $x_1, ..., x_T \in \{0,1\}^n$ randomly. If all $ABx_i = Cx_i$, return true; otherwise return false.
Runtime: $O(n^2 T)$
P(incorrect) = $\frac{1}{2^T}$
Set T = $\log_2 p$, where P(failure) = p
Proof: $P(ABx = Cx) \leq 1/2$ when $AB \neq C$
Let $D = AB - C$. If $D \neq 0$, some $d_{ij} \neq 0$.
Suppose some $Dx = 0$. Let x' be x but flipped on the jth bit.

$$Dx = \sum_{k=1}^{n} x_k d_k = \sum_{k \neq j} x_k d_k + x_j d_j = 0$$

$$Dx' = \sum_{k \neq j} x_k d_k + x'_j d_j = 0$$

If x_j originally 0, then $\sum_{k \neq j} x_k d_k = 0$. $x'_j = 1$, so $Dx' \neq 0$.
If x_j originally 1, then $\sum_{k \neq j} x_k d_k = -x_j d_j \neq 0$. $Dx' = \sum_{k \neq j} x_k d_k \neq 0$.
Pair $\{0,1\}^n$ into 2^{n-1} pairs. P(pick x that makes Dx nonzero) $\leq \frac{1}{2}$.

17.3 Karger's

Global min cut.
Algo: Pick random edge, merge vertices and edges. Repeat contraction n - 2 times and output final 2 vertices.
Runtime: Repeat $\binom{n}{2}\log_2\frac{1}{p}$ times. $O(mn^2\log(1/p))$ total.
Claim: P(returns particular min cut) $\geq \frac{1}{\binom{n}{2}}$.
Suppose k edges crossing $(S, V \setminus S)$. Algo outputs S iff never contracts edge crossing cut.
In first round, P(contract edge crossing cut) = k / m.
Consider the cut with a single vertex $(\{u\}, V - \{u\})$. Size of cut is $deg(u)$.

$$k \leq deg(u)$$

$$\sum_{u\in V} k \leq \sum_{u\in V} deg(u) \leq 2m$$

$$nk \leq 2m \rightarrow m \geq \frac{nk}{2}$$

Thus, in ith round, P(contract edge crossing cut) $\leq \frac{2}{n-i+1}$.
P(never contract edge crossing cut) =

$$(1-\frac{2}{n})*(1-\frac{2}{n-1})*(1-\frac{2}{n-2})*...*(1-\frac{2}{4})*(1-\frac{1}{3})$$

$$=\frac{n-2}{n}*\frac{n-3}{n-1}*\frac{n-4}{n-2}*...*\frac{3}{5}*\frac{2}{4}*\frac{1}{3}$$

$$=\frac{2}{n(n-1)}=\frac{1}{\binom{n}{2}}$$

18 Hashing

Def: static = query only
Def: dynamic = insert, delete, query

18.1 Hashing with Chaining

Pick random $h : \{0, ..., U-1\} \rightarrow \{0, ..., m-1\}$.
Array of linked lists. A[i] = DLL storing all (k, v) where h(k) = i.
Claim: $E[T_{op}] = O(T_h + \frac{n}{m}) \rightarrow O(1)$ if $m \geq n$ and $T_h = O(1)$

$$E[T_{op}] \leq E[T_h + C * \text{size of linked list containing x}] = T_h + C * E[\text{size of LL with x}]$$

Let database have keys $j_1, ..., j_n$. Define $z_i = 1$ if $h(j_i) = h(x)$.

$$E[\text{size of LL with x}] = E[\sum_{i=1}^{n} z_i] = \sum_{i=1}^{n} P(z_i = 1)$$

$$=\sum_{i=1}^{n}\sum_{t=1}^{m} P(h(j_i) = t \wedge h(x) = t)$$

$$=\sum_{i=1}^{n}\sum_{t=1}^{m}\frac{1}{m^2}=\frac{n}{m}$$

18.2 Universality

Def: k-wise independent = $\forall x_1 \neq x_2 \neq ... \neq x_k; \forall y_1, ..., y_k; P(h(x_1) = y_1 \wedge h(x_2) = y_2 \wedge ... \wedge h(x_k) = y_k = \frac{1}{m^k}$
Def: universal = 2-wise independent, $\forall x_1 \neq x_2, P(h(x_1) = h(x_2)) = \frac{1}{m}$.
Let U = m = p a prime. $\mathbb{H} = \{ax + b \mod p : a, b \in \{0, ..., p-1\}\}$ is universal.

18.3 Static Hashing

Def: perfect hashing $= \forall k_1, k_2; h(k_1) \neq h(k_2)$
Claim: If use $m \approx n^2$ and universal hash, P(collision) $\leq 1/2$.
Let $Z_{ab} = 1$ if $h(j_a) = h(j_b)$. Let $C =$ num collisions.

$$E[C] = \sum_{a<b} Z_{ab} = \sum_{a<b} P(h(j_a) = h(j_b)) = \sum_{a<b} \frac{1}{m} = \frac{\binom{n}{2}}{m}$$

Choose $m = n^2$, then $E[C] \leq \frac{1}{2}$.

$$P(C \geq 1) \leq \frac{E[C]}{1} \leq 1/2$$

In expectation, repeat 2x to get perfect hash.

19 Streaming

19.1 Morris'

`init()`: `X = 0`
`incr()`: `X += 1` w.p. $\frac{1}{2^X}$
`query()`: `return 2**X - 1`
Claim: $X_n =$ value of X after n increments. $E[2^{X_n}] = n + 1$.
Proof by induction on n.
Base case: $X_0 = 0$, so $2^{X_0} = 0 + 1$. Suppose holds true for n. Consider $E[2^{X_{n+1}}]$.

$$E[2^{X_{n+1}}] = \sum_{j=0}^{\infty} E[2^{X_{n+1}} | X_n = j] P(X_n = j)$$

$$= \sum_{j=0}^{\infty} (2^{j+1} \frac{1}{2^j} + 2^j (1 - \frac{1}{2^j})) P(X_n = j)$$

$$= \sum_{j=0}^{\infty} (2 + 2^j - 1) P(X_n = j)$$

$$= \sum_{j=0}^{\infty} P(X_n = j) + \sum_{j=0}^{\infty} 2^j P(X_n = j)$$

$$= 1 + E[2^{X_n}] = 1 + (n + 1)$$

19.1.1 Error Bound

Thm (Chebychev's): $P(|X - E[X]| > \lambda) < \frac{E[(X-E[X])^2]}{\lambda^2} = \frac{var(X)}{\lambda^2}$
Proof: Apply Markov's inequality.

$$P(|X - E[X]| > \lambda) = P((|X - E[X]|)^2 > \lambda^2) \leq \frac{E[(X - E[X])^2]}{\lambda^2}$$

Claim: $E[(X - E[X])^2] = E[X^2] - E[X]^2$

$$E[(X - \mu)^2] = E[X^2 - 2X\mu + \mu^2] = E[X^2] - 2E[X]E[X] + E[X]^2 = E[X^2] - E[X]^2$$

Claim: $E[2^{2X_n}] = \frac{3}{2}n^2 + \frac{3}{2}n + 1$.
Base case: $n = 0$, $E[2^{2X_0}] = 1 = 0 + 0 + 1$
Suppose holds true for n.

$$E[2^{2X_{n+1}}] = \sum_{j=0}^{\infty} E[2^{2X_{n+1}} | 2^{X_n} = j] P(2^{X_n} = j)$$

$$= \sum_{j=0}^{\infty} (4j^2 \frac{1}{j} + j^2(1 - \frac{1}{j}))P(2^{X_n} = j)$$

$$= \sum_{j=0}^{\infty} (j^2 + 3j)P(2^{X_n} = j)$$

$$= E[2^{2X_n}] + 3E[2^{X_n}]$$

$$= \frac{3}{2}n^2 + \frac{3}{2}n + 1 + 3n + 3$$

$$= \frac{3}{2}(n+1)^2 + \frac{3}{2}(n+1) + 1$$

Let $Z = 2^X - 1$. $var(Z) \leq \frac{n^2}{2}$.
Run R copies of algorithm, take $Z = \frac{1}{R}\sum_{i=1}^{R} Z_i$. Same expectation, variance becomes $\frac{n^2}{2R}$. Set $R \approx \frac{1}{\epsilon^2 p}$.

$$P(|Z - N| \geq \epsilon N) < \frac{var(Z)}{\epsilon^2 N^2} < p$$

Mem usage is $O(\epsilon^{-2}p^{-1} \log\log n/(\epsilon p))$ w.p. $1 - p$.
Alternatively, change base of exponent. If increment w.p. $\frac{1}{(1+a)^X}$, $E[\frac{(1+a)^X - 1}{a}] = N$ and $var(Z) \leq aN(N-1)/2$. Choose $a \approx \epsilon^2 p$. Mem usage about $O(\log\log N + \log\frac{1}{\epsilon} + \log\frac{1}{p})$

19.2 Frievald's

stream of m items with repeats from universe $[n]$. Count number distinct elts.
`init()`: $X \to 1$, pick random $h : [n] \to [0, 1]$.
`update(i)`: $X \to \min(X, h(i))$
`query()`: $\frac{1}{X} - 1$
Claim: $E[X] = \frac{1}{t+1}$ where unique integers are $i_1, ..., i_t$.

$$E[X] = \int_0^{\infty} P(X > \lambda)d\lambda$$

$$= \int_0^1 P(h(i_1) > \lambda \wedge ... \wedge h(i_t) > \lambda)$$

$$= \int_0^1 (1 - \lambda)^t d\lambda$$

$$= \frac{1}{t+1}$$

19.2.1 Error Bound

Claim: $E[X^2] = \frac{2}{(t+1)(t+2)}$

$$E[X^2] = \int_0^{\infty} P(X^2 > \lambda)d\lambda$$

$$= \int_0^1 P(X > \sqrt{\lambda})d\lambda$$

$$= \int_0^1 (1 - \sqrt{\lambda})^t d\lambda$$

$$= 2\int_0^1 u^t(1 - u)du$$

$$= \frac{2}{(t+1)(t+2)}$$

Let $s = \frac{1}{\epsilon^2 p}$ and average s copies to get final output Z. Same expectation, $Var(Z) < \frac{1}{s(t+1)^2}$.

$$P(|Z - \frac{1}{t+1}| \geq \frac{\epsilon}{t+1}) < \frac{(t+1)^2}{\epsilon^2}\frac{1}{s(t+1)^2} \leq p$$