Divide-and-conquer algorithms

Master Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0, b > 1$, and $d \geq 0$ then,

$$T(n) = \begin{cases} O(n^d) & \text{if } a/b^d < 1 \quad d > \log_b a \\ O(n^d \log n) & \text{if } a/b^d = 1 \quad d = \log_b a \\ O(n^{\log_b a}) & \text{if } a/b^d > 1 \quad d < \log_b a \end{cases}$$

Proof: Assume n is power of b (not too important). Notice that we have $\log_b n$ levels. Given a, the k^{th} level of the tree has work $a^k \times O\left(\frac{n}{b^k}\right)^d = O(n^d) \times \left(\frac{a}{b^d}\right)^k$

$\left(\frac{a}{b^d}\right)$ Ratio $< 1 \Rightarrow$ bounded, $O(n^d)$; $= 1 \Rightarrow \log n$ levels same; $> 1 \Rightarrow$ biggest term $n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^d \left(\frac{a^{\log_b n}}{(b^{\log_b n})^d}\right) = \frac{n^d}{n^d} a^{\log_b n} = n^{\log_b a}$

$n \log n$ sorting bound

In a length n array, there are $n!$ possible permutations. A correct sorting algorithm must generate $2^d \geq n!$ different true/false sequences. $d \geq \log_2 n! = \Omega(n \log n)$

Fast median finding

Modify quickselect (general algorithm for finding k^{th} largest)

$$\text{selection}(S, k) = \begin{cases} \text{selection}(S_L, k) & \text{if } k \leq |S_L| \\ v & \text{if } |S_L| < k \leq |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v| \end{cases}$$

Worst case $\Theta(n^2)$ if pivot is bad. Average case: $1/4$ to $3/4$ split good (in expectation, two splits until $3/4$'s problem size)

$T(n) \leq T(3n/4) + O(n)$ (time w/ $3/4$, time to split)

Matrix multiplication

$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$ — $O(n^3)$ naive algorithm for multiplication

$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$ $Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$ $XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$

$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$

$P_1 = A(F - H)$, $P_2 = (A+B)H$, $P_3 = (C+D)E$, $P_4 = D(G - E)$, $P_5 = (A+D)(E+H)$, $P_6 = (B-D)(G+H)$, $P_7 = (A-C)(E+F)$

$T(n) = 7T(n/2) + O(n^2) \Rightarrow O(n^{\log_2 7}) \approx O(n^{2.81})$

Fast Fourier Transform

Degree-d polynomial is uniquely characterized by $d+1$ points, so we can look at coefficients $a_0, a_1, \ldots a_d$ or $A(x_0), A(x_1) \ldots A(x_d)$. Multiplication $O(n)$ in point representation, and selection is $O(n)$ as well, need fast evaluation and interpolation

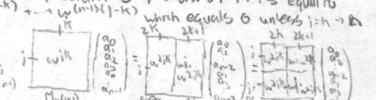
Evaluation

Trick with $\pm$ pairs only works at the top level. Use n roots of unity because ① n^{th} roots of unity are $\pm$ paired, $w^{n/2 + j} = -w^j$ ② Squaring them produces $(n/2)^{nd}$ roots of unity. At every level, express $A(x)$ as $A_e(x^2) + xA_o(x^2)$, eval A_e and A_o at even powers of w and compute $\forall j = 0 \ldots n-1$ $A(w^j) = A_e(w^{2j}) + w^j A_o(w^{2j})$

Interpolation

Matrix is just transformation to Fourier basis, invert using the operation $M_n(w)^{-1} = \frac{1}{n} M_n(w^{-1})$. This is because: Lemma The columns of M are orthogonal to each other. The inner product of any columns j and k of M is equal to $1 + w^{j-k} + w^{2(j-k)} + \ldots + w^{(n-1)(j-k)}$ which equals 0 unless $j = k \rightarrow n$

$$M(w) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{n-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(n-1)} \\ & & \vdots & & \\ 1 & w^j & w^{2j} & \cdots & w^{(n-1)j} \\ 1 & w^{(n-1)} & w^{2(n-1)} & \cdots & w^{(n-1)(n-1)} \end{bmatrix}$$



Complex Numbers

Consider the complex plane, where in polar coordinates, i is a $90°$ ccw rotation. Think of all numbers $z = r\cos\theta + ir\sin\theta$, $e^{i\theta} = \cos\theta + i\sin\theta$. Roots of unity are all of the solutions to $z^n = 1$ for some fixed n.

Fast multiplication of numbers

$\boxed{x} \rightarrow \boxed{x_1}\boxed{x_2} = x_1 \cdot 2^{n/2} + x_2$ $\quad x \cdot y = 2^n x_1 y_1 + 2^{n/2}(x_1 y_2 + x_2 y_1) + x_2 y_2$

Using Gauss' trick $[(a + ib) * (c + id) = (ac - bd) + i(ad + bc) = (a+b)(c+d) - ac - bd]$ we get the recurrence $T(n) = 3T(n/2) + O(n) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.6})$

Universal Hashing

Family is a set of hash functions $\mathcal{H}$ of $h: A \to B$ where for any $x \neq y \in A$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = \frac{1}{|B|}$. Good performance in expectation. You select the hash family, hash function and data are random

Parallel Algorithms

Sequential algorithms care about time (poly v, exp X), sometimes NP-complete

Parallel may have synch clock, enable parallel reads but not writes

Work - total number of instructions (ideally it is polynomial)

Depth - clock time in parallel execution (perhaps $\log n$ or $(\log n)^2$)

Brent's Principle: Solving problem with work W, depth D, ∞ P $\Rightarrow$ work $O(W)$, proc P, $D' = D + W/P$

Proof - Simulate each timestep with ceil$[W_t/P]$ steps of P processors which gets us $D' < D + W/P$

```
Sum(A[1...n])
If n = 1 return A[1]
for i = 1,...,n/2 do in parallel
A[i] = A[2i-1] + A[2i]
return Sum(A[1...n/2])
```

$W(n) = W(n/2) + n/2 \rightarrow O(n)$
$D(n) = D(n/2) + 2 \rightarrow O(\log n)$
(work same as best sequential)

Compact $(2\,0\,0\,1\,3\,0) \rightarrow (2\,1\,3)$

Find Root
for every node v do in parallel
if next[v] ≠ v next[v] = next[next[v]]
(perform log n times)

Boruvka's Algorithm

Initialize set of edges T and list of connected components C
while |C| > 1
for each c in C do in parallel
find the shortest edge out of C and add it to T $\quad (W = |E|, D = \log|V|)$
C = connected components of T $\quad (W = |E|\log|V|, D = \log|V|)$
Total: $W = O(|E|\log^2|V|)$, $D = O(\log^2|V|)$

Connected Components

function cc(V, E) returns array[V] of V
Initialize: for every node v do in parallel: leader[v] = coin(), ptr[v] = v
for all non-leader node v do in parallel:
choose an adjacent leader node, if one exists, and set ptr[v] = u (ptr a bunch of stars)
$V' = \{v : ptr[v] = v\}$ (the roots of the stars)
$E' = \{(u, v) : u \neq v$ in V', there is (a, b) in E such that $ptr[a] = u$, $ptr[b] = v\}$ (contract the graph)
label[V] = cc(V', E') (compute cc recursively on contracted graph)
return cc[v] = label[ptr[v]]

Graphs

Represent using adjacency list/matrix for sparse/dense

Average degree of graph $\frac{2|E|}{V}$, interested in sparse graphs

DAG - sources and sinks, can be linearized by topological sort: run DFS and keep track of post, reverse all edges and run DFS from highest post decreasing

Depth-first search explores the idea of connectivity

Marking each node with pre and post numbers, two intervals $[pre(u), post(u)]$, $[pre(v), post(v)]$ disjoint or contained by LIFO

Three types of edges in DFS tree (directed graphs): $[\,[\,]\,]$
① Forward edge - node → non-child desc ② Back edge - to ancestor in DFS tree $[\,[\,]\,]$
③ Cross edge - to node already visited $[\,]\,[\,]$

Strongly-connected graphs

Any two nodes u and v connected if path $u \to v$ and $v \to u$

Every graph is a DAG of SCC's, otherwise can merge

Biconnected graphs

A graph is biconnected if there is some edge $e \in E$ such that $C = \{e' \in E \mid e \sim e'\}$, either $e = e'$, or cycle w/ e, e'

Cannot share edges or more than one vertex (separating)

Shortest Paths

Breadth-First Search finds shortest paths for uniform weights

We can view Dijkstra's as BFS w/ alarm clocks at nodes yet to be visited and dummy nodes $\propto$ to distances

Using priority queue, insert - add element to set ($O(\log|V|)$)

Decrease-key - update key value ($O(\log|V|)$), Delete-min $\rightarrow O(\log|V|)$

Dijkstra's invariant is that every time a new node v is expanded, it is the least cost (otherwise contradicts vs status as lowest)

"At end of iteration of while loop, ① $\exists$ d s.t. all nodes in R at dist $\leq$ d and all outside $\geq$ d, ② $\forall u$, dist(u) = shortest path s.t. all intermediate $\in R$"

Running time: $|V|$ deletemin, $|V| + |E|$ insert/decreasekey $\rightarrow O((|E| + |V|)\log|V|)$

Priority Queue implementations

	deletemin	insert/decreasekey	$\lvert V\rvert \times$ deletemin + $(\lvert V\rvert + \lvert E\rvert) \times$ insert
Array	$O(\lvert V\rvert)$	$O(1)$	$O(\lvert V\rvert^2)$
Binary heap	$O(\log\lvert V\rvert)$	$O(\log\lvert V\rvert)$	$O((\lvert V\rvert + \lvert E\rvert)\log\lvert V\rvert)$
d-ary heap	$O\left(\frac{d\log\lvert V\rvert}{\log d}\right)$	$O\left(\frac{\log\lvert V\rvert}{\log d}\right)$	$O\left((\lvert V\rvert \cdot d + \lvert E\rvert)\frac{\log\lvert V\rvert}{\log d}\right)$
Fibonacci heap	$O(\log\lvert V\rvert)$	$O(1)$ (amort)	$O(\lvert V\rvert\log\lvert V\rvert + \lvert E\rvert)$

Negative Edges

Dijkstra's invariant is no longer upheld, now its possible that a nodes shortest distance found can change after expanded

Bellman-Ford ($O(|V| \cdot |E|)$)

Any path from S to any node is at most $|V| - 1$ long

Repeat the following $|V| - 1$ times: for all $e \in E$ update(e), averaging shortest paths found to nodes that it connects to $\quad$ dist(v) = max$\{$dist(v), dist(u) + $l_{u \to v}\}$

Negative cycles can be found by running Bellman-Ford one more time and checking for any further changes

Shortest Paths in DAGs

Run topological sort to linearize the DAG

Iterate through the nodes from left to right, and update all outgoing edges

When node is explored, optimal distance has been found

Greedy Algorithms

Minimum Spanning Trees

Goal: Find the set of edges $S \subseteq E$ in graph G that spans and is lightest

① Removing a cycle cannot possibly disconnect a graph

② A tree on n nodes has n-1 edges ③ Any connected G w/ $|E| = |V| - 1$ is a tree

④ An undirected graph is a tree if and only if $\exists$ unique path between all nodes

The cut property: Suppose edges X are part of some MST of $G = (V, E)$. Pick any subset of nodes S for which X does not cross between S and V-S, and let e be the lightest edge across this partition. Then $X \cup \{e\}$ is part of some MST.

If not, show by contradiction that adding lightest e and removing some heavier e' allows a better MST.

Kruskal's algorithm ($O(|E|\log|E|)$)

Represent algorithm's state as a collection of disjoint sets

Begin by sorting all edges in increasing order

Iterate through collection in order of increasing weight, and add edge (union(u, v)) if find(u) ≠ find(v)

Prim's algorithm

Instead of constantly connecting different sets together, always add the minimum edge to the current connected component until $|X| = |V| - 1$, pick edge not connecting elements seen together using perhaps a priority queue leading to similar runtime

Disjoint Sets

Each set has a root element serving as its representative

```
procedure makeset(x)        function find(x)
  π(x) = x                    while x ≠ π(x): x = π(x)
  rank(x) = 0                 return x
```

Union operation will merge smaller rank with larger (does not change rank). If identical rank, will increase rank of one by 1

① For any x, rank(x) < rank(π(x)) ② Any root node of rank k has at least 2^k nodes in its tree ③ If there are n elements, at most $\leq \frac{n}{2^k}$ rank k nodes so max rank is $\log n$ and all trees height $\leq \log n$.

Path Compression

Modification to find(x) → if x ≠ π(x): π(x) = find(π(x)) → return π(x)

No effect on union operations, finds do not change ranks

Divide ranks into intervals $\{1\}, \{2\}, \{3, 4\}, \{5, \ldots 16\}, \{17, \ldots, 2^{16}\} \ldots$ where there exist $\log^* n$ total groups

Give each node allowance 2^k dollars if in interval $\{k+1, \ldots, 2^k\}$ since nodes bounded by $\frac{n}{2^k}$, each level gets n dollars = $n\log^* n$ overall

Each node has sufficient dollars to reach top of current tree, then find takes care of rest ($\log^* n$ transitions between classes)

Huffman Encoding

Generate prefix-free encoding to minimize $\sum_{i=1}^{n} f_i \cdot$ (depth of i^{th} symbol) $= \sum_{i=1}^{2n-2}$ internal node cost

Two symbols with lowest frequencies must be at the bottom of the tree, else can swap with anything to ↓ cost

A greedy algorithm will behave correctly

Set cover

Given a set of nodes and possible subsets, choose smallest number of sets required to cover all nodes

Greedy: keep assigning set with largest uncovered until finished

Suppose optimal cover at k sets over n elements, greedy uses $k \ln n$

n_t - number of uncovered elements after t iterations (remaining elements covered by optimal k sets, must be one with n_t/k)

$n_{t+1} \leq n_t - n_t/k = n_t(1 - 1/k)$ so $n_t \leq n_0(1 - 1/k)^t$

Use inequality $1 - x \leq e^{-x}$, so $n_t \leq n_0(1 - 1/k)^t < n_0(e^{-1/k})^t = n e^{-t/k}$

At $t = k\ln n$, n_t strictly less than 1

Approximation Algorithms

Develop approximations for NP-complete problems that are possibly close to the optimal solution

Approximation ratio $\alpha_A = \max_I A(I)/OPT(I)$

Vertex Cover

Reduce to set cover, can be solved within factor of $O(\log n)$

Approximate using maximal matching, keep picking edges disjoint from others (solution is just vertices connected by these edges)

Factor 2 approximation - every edge must be covered by either vertex

k-Cluster

Given distance matrix, n points and integer k, find and assign k clusters such that minimum $\max_i \max_{x_a, x_b \in C_i} d(x_a, x_b)$

NP-hard, approximate by picking k means in following way: Pick first randomly, then repeatedly pick points that are farthest away from any of picked means

These k^+ points at least distance r from each other and any partition into k clusters must put two points in same cluster

TSP

2-Approximation possible for TSP if distance triangle inequality holds

Compute MST of the graph, which has dist smaller than TSP cost * 2

Write nodes in order of DFS (remove duplicates) and connect them directly ⇒ dist at most twice MST ≤ 2 * TSP cost

Can we approximate normal TSP?

We previously showed that budgeted TSP is NP-complete

Consider an approximation algorithm for TSP with constant α_A, show that we can solve Rudrata Path quickly

Knapsack

Given any $\epsilon > 0$, find a solution at least $(1 - \epsilon)$ times optimal, poly in $1/\epsilon$ and input

Procedure: ① Discard item with $w > W$ ② Let $v_{max} = \max_i v_i$ ③ Rescale values $\hat{v}_i = \lfloor v_i \cdot n/\epsilon v_{max} \rfloor$ ④ Run DP with $\{\hat{v}_i\}$ and output

$\hat{v}_i$ at most n/ϵ, so DP runs in time $O(n^3/\epsilon)$

$$\sum_{i \in S} \hat{v}_i = \sum_{i \in S} \left\lfloor v_i \cdot \frac{n}{\epsilon v_{max}} \right\rfloor \geq \sum_{i \in S} \left(v_i \cdot \frac{n}{\epsilon v_{max}} - 1\right) \geq K^* \cdot \frac{n}{\epsilon v_{max}} - n$$

Dynamic Programming

Identify a collection of subproblems, solve from bottom up to get larger ones
Implicit DAG contained in nodes (problems), edges (dependencies)

Shortest Paths in DAGs
DAGs can be linearized, so it's possible to construct states + dependencies
Initialize all dist to ∞ except s, and for each node in linear order
$dist(v) = \min_{(u,v) \in E} \{dist(u) + \ell(u,v)\}$
The structure is fundamental to DP (smaller dist necessary to get larger ones)

Longest Increasing Subsequence
Visualize by creating a DAG of all possible transitions
Subproblem: $L[i]$, length of longest increasing subsequence ending at i
for $i = 2$ to n: for $j = 1$ to $i-1$: (base case: $L[1] = 1$)
if $(a[j] < a[i])$: $L[i] = \max(L[j]+1, L[i])$; $prev[i] = j$ (keep track)
$L[i]$ takes $k \cdot d$ indegree to compute, overall $|E|$ or n^2 worst case computation

Optimal Gambling Strategy
Suppose you are in Vegas for d days with \$1, can gamble x dollars each day with payoff ⅓ success + ⅔ failure
Subproblem: $P[m, i]$: max proffit given that we have m \$ after ith day
Each day, try all possible outcomes and max (P happy) last day: $[1, c]$, Σ [illegible]

Minimum Edit Distance
Number of keystrokes required to go from one word to another
Subproblem: $E[i,j]$ = edit distance between $x[1 \ldots i]$, $y[1 \ldots j]$
For each subproblem, we decide $\{\frac{a}{-}\}, \{\frac{-}{a}\}$ or $\{\frac{a}{b}\}$ (deletion or identical)
$E[i,j] = \min\{1 + E(i-1,j), 1 + E(i,j-1), diff(i,j) + E(i-1,j-1)\}$
We use an $m \times n$ table where top row and left column are $(0, 1, \ldots, n)$
with $O(mn)$ total subproblems and $O(1)$ computation, $O(mn)$ total time
From underlying DAG, we can generalize to more complicated problems

Knapsack
Given finite capacity and items of value/weight (find best combo to take)
Knapsack with Repetition
There is an infinite number of objects, so all that matters is capacity
Subproblem: $K(w)$ – maximum achievable value with capacity w
At every step, $K(w) = \max_{i: w_i \le w} \{K(w - w_i) + v_i\}$ ($K(0)$ is taken to be 0)
Every problem requires up to $O(n)$ to compute, so $O(nw)$ time overall
Knapsack without Repetition
If items are unique, then we must keep an additional parameter
Subproblem: $K(w, j) = \max\{K(w - w_j, j-1) + v_j, K(w, j-1)\}$
where $K(w,j)$ = max value achievable using capacity w and items $1 \ldots j$
Filling out a $W \times n$ table, so $O(NW)$ – weakly polynomial

Shortest Paths
Finding shortest path (w/negative weights) using at most k edges
and between all pairs of vertices (Bellman-Ford → $|V|^2|E|$)
Floyd-Warshall algorithm
Gradually expand set of permissible nodes as intermediates
for all $(i,j) \in E$: $dist(i,j,0) = \ell(i,j)$ Subproblem: $d[i,j,k]$ – sh. path between i,j using $1 \ldots k$ as inter.
for $k = 1$ to n: for $i = 1$ to n: for $j = 1$ to n: $dist(i,j,k) = \min\{dist(i,k,k-1) + dist(k,j,k-1), dist(i,j,k-1)\}$
Using at most k edges, $dist(v, k) = \min_{(u,v) \in E} \{dist(u, k-1) + \ell(u,v)\}$

Traveling Salesman Problem
Visit all cities (nodes) exactly once and return home, min distance
Brute force solution tries all $(n-1)!$ possibilities: $O(n!)$
Subproblem: $C[S, j]$ – length of shortest path visiting each node in S exactly once, starting at 1 and ending at j
Recurrence: $C[S, j] = \min_{i \in S: i \ne j} C(S - \{j\}, i) + d_{ij}$
$C[\{1\}, 1] = 0$
for $s = 2$ to n:
for all subsets $S \subseteq \{1 \ldots n\}$ of size s and containing 1:
$C(S, 1) = \infty$
for all $j \in S$, $j \ne 1$:
$C(S, j) = \min\{C(S - \{j\}, i) + d_{ij} : i \in S, i \ne j\}$
return $\min_j C(\{1, \ldots, n\}, j) + d_{j1}$
There are $2^n \cdot n$ subproblems, each takes $O(n)$, so $O(n^2 2^n)$ runtime

Optimal Triangulation
Cut a convex polygon into triangles, minimizing total length + diff edges
Subproblem: $L(i,j)$ = cost of optimal triangulation of $P_i, P_{i+1} \ldots P_j$
All polygon edges participate in some triangle, can form smaller regions
Recurrence: $L(i,j) = \min\{d(i,j) + L(i+1,j), \ldots, d(i,k) + d(k,j) + L(i,k) + L(k,j)\}$

Independent Sets
Find the largest set of some tree where none are directly connected by edges
Pick any node as root (number of subtrees is linear)
Subproblem: $I(u)$ = size of largest independent set in subtree rooted at u
Recurrence: $I(u) = \max\{1 + \sum_{\text{grandchildren } w} I(w), \sum_{\text{children } w} I(w)\}$ (u in set / u not in set)
Each node is referenced a finite number of times, $O(|V| + |E|)$ runtime

Chain Matrix Multiplication
Given a sequence of matrices, find the most optimal way to multiply
Subproblem: $C(i,j)$ = minimum cost required to multiply $A_i \ldots A_j$
Recurrence: $C(i,j) = \min_{i \le k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$

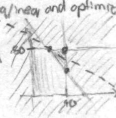
Linear Programming

Broad class of optimization tasks where constraints and objective are linear
Given a set of variables, assign values to satisfy eq/inequ and optimize objective

Profit Maximization

	Ruby Necklace		Diamond Ring		
	$20x$	+	$10y$	≤ 1200	grams gold
	0	+	$1 \cdot y$	≤ 80	diamonds
	$1 \cdot x$	+	0	≤ 40	rubys

$x, y \ge 0$
Maximize the profit $10x + 10y$



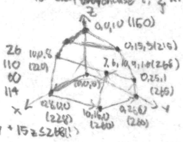
① Feasible region is convex (created by lines) ② Optimum of LP occurs at vertex
2 variables & m constraints ⇒ $\binom{m}{2}$ corners, actually more like m
Simplex is essentially a hill-climbing method on polygon vertices
It's possible for no optimum if (1) $x \le 1$, $x \ge 2$ infeasible (2) $\max x_1 + x_2$, $x_1, x_2 \ge 0$ feasible region unbounded
By geometry, all vertex neighbors lie on other half of hyperplane ⇒ optimality

Minimizing Shipping Costs
Given ① 100 warehouses $\{s_1, \ldots, s_{100}\}$ ② 100 cities $\{d_1, \ldots, d_{100}\}$ Variables x_{ij} – amount shipped from w_i to c_j
Minimize shipping cost $\sum_{w_i} \sum_{c_j} cost[i,j] \cdot x_{ij}$ for every city j, $\sum_i x_{ij} \ge d_j$ for every warehouse i, $\sum_j x_{ij} \le s_i$ $x_{ij} \ge 0$

Farms and Duality

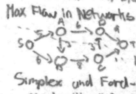
	wheat		corn		rice	
max	$10x$	+	$10y$	+	$15z$	
land	x	+	y	+	z	≤ 26
labor	$3x$	+	$4y$	+	$10z$	≤ 110
water	x	+	$2y$	+	$6z$	≤ 60
storage	$5x$	+	$4y$	+	$8z$	≤ 114

$x, y, z \ge 0$



Dual solution $(6, \frac{1}{2}, 0, \frac{1}{2})$ gets $10x + 10y + 15z \le 268$!

Max Flow in Networks

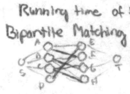


Given a directed graph representing capacities of edges, ship as much as possible from s to t.
LP: $\max \sum f_s$ s.t. $5 \ge f_{SA} \ge 0$, $1 \ge f_{CT} \ge 0$, $f_{SA} = f_{AD} + f_{AB}$, $f_{AD} + f_{CD} = f_{DT}$ (capacities and flow conservation)

Simplex and Ford-Fulkerson
start with all $f = 0$
while there is a path from s to T
find δ and add to all edges to get G^R
keep finding paths by BFS/DFS until ∃ no more paths
residual capacities $c^f = \begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$

Optimality
Picking any two cuts L and R, capacity – total possible from L to R, flow – actual flow
we will notice that $size(f) \le capacity(L, R)$
Max-flow min-cut theorem – The size of the maximum flow equals the capacity of the smallest (s,t)-cut.
Running time of this procedure is $O(|V| \cdot |E|^2)$ since # iterations $\le O(|V| \cdot |E|)$

Bipartite Matching



Given a graph of n nodes on both sides, is it possible to produce a feasible matching? Adding S and T, this reduces to the problem of max flow (check = 4). Integer solutions automatic!

LP Notes
Can be (1) min/max problem (2) constraints either eq/inequalities (3) var usually non neg., also can unrestrict
① max problem can be turned into min problem, multiply obj by −1
② a) Turn inequality $\sum_{i=1}^n a_i x_i \le b$ into equation $\sum_{i=1}^n a_i x_i + s = b$, $s \ge 0$ using slack var
b) Change equality constraint $ax = b$ into inequalities $ax \ge b$, $ax \le b$
③ represent x using $x^+, x^- \ge 0$ and replacing all x with $x^+ - x^-$
This allows us to reduce all LP into "standard form": var nonneg, equations, min obj

Duality
Primal LP: $\max c^T x$, $Ax \le b$, $x \ge 0$ Dual LP: $\min y^T b$, $y^T A \ge c^T$, $y \ge 0$
Theorem: If an LP has a bounded optimum, then so does its dual, and the two optimum values coincide.

$\min 26\alpha + 110\beta + 60\gamma + 114\delta$
$\alpha + 3\beta + \gamma + 5\delta \ge 10$
$\alpha + 4\beta + 2\gamma + 4\delta \ge 10$
$\alpha + 10\beta + 6\gamma + 8\delta \ge 15$
$\alpha, \beta, \gamma, \delta \ge 0$

$\alpha = 6$, $\beta = \frac{1}{2}$, $\gamma = 0$, $\delta = \frac{1}{2}$, opt: 268

Zero-Sum Games

		Column		
		r	p	s
Row	r	0	−1	1
	p	1	0	−1
	s	−1	1	0

It is in Row and Col's interests to play a mixed strategy
Expected payoff $= \sum_{i,j} G_{ij} \cdot Prob[Row\ i, Col\ j] = \sum_{i,j} G_{ij} x_i y_j$
Suppose Row plays randomly, Col will have 0 payoff no matter what!
Same with Col's strategy.

Consider two scenarios: 1) Row announces, Col resp 2) Col announces, Row resp

$G=$	m	t
e	3	−1
s	−2	1

① $x = (x_1, x_2)$ Row will pick x that maximizes $\min\{3x_1 - 2x_2, -x_1 + x_2\}$
writing z as $\max z$, $z \le 3x_1 - 2x_2$, $z \le -x_1 + x_2$ to get $\max z$: $-3x_1 + 2x_2 + z \le 0$, $x_1 - x_2 + z \le 0$, $x_1 + x_2 = 1$, $x_1, x_2 \ge 0$
② $y = (y_1, y_2)$ Col will pick y that minimizes $\max\{3y_1 - y_2, -2y_1 + y_2\}$ where $\min z$: $-3y_1 + y_2 + z \ge 0$, $2y_1 - y_2 + z \ge 0$, $y_1 + y_2 = 1$, $y_1, y_2 \ge 0$
These LP's are dual to each other and we find that
$\max_x \min_y \sum_{i,j} G_{ij} x_i y_j = \min_y \max_x \sum_{i,j} G_{ij} x_i y_j$

Proving the Max-Min Theorem
Idea: can write min max as LP, so primal and dual must have the same value

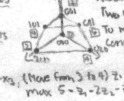
The Experts Problem
cost c_i^t (← time, ← expert) Want: small regret $= \frac{1}{T}$(our cost − how well best exp)

Experts	1	2	3	4	5	6	7	8
$t=1$	0.4	0.1	0.5	0.3	0.3	0.2	0.2	0.0
$t=2$	0.4	0.1	0.8	0.4	0.2	0.7	0.2	0.8
$t=3$	0.2	0.9	0.3	0.5	0.1	0.0	0.3	0.7

Note①: cannot do well compared to expert each day
Note②: To achieve, you need randomness

Multiplicative Weights
Each expert has weight w_i^t ($w_i^1 = 1$)
Repeat for T steps: choose expert i w prob $\frac{w_i^t}{W^t}$ ($W^t = \sum_i w_i^t$), update $w_i^{t+1} = w_i^t (1-\epsilon)^{c_i^t}$
Theorem: MW guarantees regret $\le 2\sqrt{\frac{\ln n}{T}}$
① $(1-\epsilon)^x \le 1 - \epsilon x$
② $-x - x^2 \le \ln(1-x)$
i^* = the best expert, with smallest $\sum_t c_{i^*}^t = OPT$
$-\sum w_i^t \epsilon c_i^t = -W^t \sum \frac{w_i^t}{W^t} \epsilon c_i$
we know that $W^T \ge w_{i^*}^T = (1-\epsilon)^{OPT}$
Recurrence: $W^{t+1} = \sum w_i^{t+1} = \sum w_i^t (1-\epsilon)^{c_i^t} \le \sum w_i^t (1 - \epsilon c_i^t) = W^t - \epsilon W^t \bar{c}^t$
From $(1-\epsilon)^{OPT} \le W^T \le n \prod_{t=1}^T (1 - \epsilon \bar{c}^t)$
$\ln(1-\epsilon) \cdot OPT \le \ln n + \sum_t \ln(1 - \epsilon \bar{c}^t)$
$(-\epsilon - \epsilon^2) OPT \le \ln n - \sum_t \epsilon \bar{c}^t$
$(1+\epsilon) OPT \ge -\frac{\ln n}{\epsilon} + \sum_t \bar{c}^t = -\frac{\ln n}{\epsilon} + V$
$V - OPT \le \frac{\ln n}{\epsilon} + \epsilon OPT \le \frac{\ln n}{\epsilon} + \epsilon T$ (choose $\epsilon = \sqrt{\frac{\ln n}{T}}$)
$\frac{V - OPT}{T} \le \frac{\ln n}{\epsilon T} + \epsilon \Rightarrow \frac{V - OPT}{T} \le 2\sqrt{\frac{\ln n}{T}}$
First, we know that max min ≤ min max, so we can write max min = min max − δ for $\delta \ge 0$
Goal: Show that $\delta = 0$
Idea: R and C can play by Multiplicative weights, with $0 \le a_{ij} \le 1$ and experts columns for C and rows R
Expert costs: (for C) $c_j^t = col_j(A) p^t$ ← prob of R at t (for R) $c_i^t = 1 - row_i(A) q^t$ ← prob of C at t
Both achieve a value $V = \frac{1}{T} \sum_{t=1}^T \sum_{i,j} a_{ij} p_i^t q_j^t$ (this will be $O(\frac{1}{\sqrt{T}})$ close to best expert of each)
Allow them to play for a long time, then R: $\hat{x}_i = \frac{1}{T} \sum_{t=1}^T p_i^t$, C: $\hat{y}_j = \frac{1}{T} \sum_{t=1}^T q_j^t$
$\max_x \min_j \{col_j(A) x\} \ge \min_j \{col_j(A) \hat{x}\} \ge V - 2\sqrt{\frac{\ln m}{T}} \ge (\max_i \{row_i(A) \hat{y}\} - 2\sqrt{\frac{\ln n}{T}}) - 2\sqrt{\frac{\ln m}{T}}$ (# columns)
$\max_x \min_j \{col_j(A) x\} \ge \min_y \max_i \{row_i(A) y\} - 2\frac{\sqrt{\ln m} + \sqrt{\ln n}}{\sqrt{T}}$, so $\delta \le 2\frac{\sqrt{\ln m} + \sqrt{\ln n}}{\sqrt{T}}$ for all $T > 0 \Rightarrow \delta \to 0$

Simplex Algorithm
Start from vertex (say, (0,0,0)) and repeat: If vertex is optimum, stop. otherwise, find better adjacent vertex
$\max x_1 + 2x_2 + 3x_3$ $n = 3$ variables
$x_1 + x_2 + x_3 \le 2$ ①
$x_3 \le 1$ ② $m = 5$ constraints
$x_1 \ge 0$ ③
$x_2 \ge 0$ ④
$x_3 \ge 0$ ⑤



Recall: A point in n-dimensional space is the intersection of n constraints
Two vertices are adjacent if they share all but one plane/constraint
To move, we increase one variable until we hit the next plane
consider each new point as a zero, terminate when all obj [illegible]

(Move up 3) $y_1 = x_1$, $y_2 = x_2$, $y_3 = 1 - x_3$; $\max 3 + y_1 + 2y_2 - 3y_3$
$y_1 + y_2 - y_3 \le 1$ ①
$y_3 \ge 0$ ②
$y_1 \ge 0$ ③
$y_2 \ge 0$ ④
$y_3 \le 1$ ⑤

(Move from 3 to 4) $z_1 = y_1$, $z_2 = 1 - y_1 - y_2 + y_3$, $z_3 = y_3$; $\max 5 - z_1 - 2z_2 - z_3$
$z_2 \ge 0$ ①
$z_3 \ge 0$ ②
$z_1 \ge 0$ ③
$z_1 + z_2 - z_3 \le 1$ ④
$z_3 \le 0$ ⑤
(1, 2, 1) are the multipliers and get us
$x_1 + x_2 + x_3 \le 2$ ×2
$x_3 \le 1$ ×1
$-x_1 \le 0$ ×1
$x_1 + 2x_2 + 3x_3 \le 5$

Degeneracy

Perhaps $4 > n$ planes intersect
Perturbation changes each b_i by $\pm\epsilon_i$

NP-Complete Problems

P – problems for which we can find solution efficiently in polynomial time
NP – problems for which we can recognize/check the solution efficiently
Search problem is specified by algorithm C that takes two inputs, an instance I and proposed solution S, and runs in time polynomial in $|I|$. S is a solution to I iff $C(I, S)$ = true.

Hamiltonian/Rudrata Cycle
Input: Graph G, Solution: A cycle that visits every vertex exactly once
Rudrata Path
Input: Graph G, s, t, Solution: A path $s \to t$ that visits every vertex
Factorization
Input: An n-bit number N, Solution: two numbers $p, q > 1 \to p \cdot q = N$
3SAT
Input: A 3-CNF boolean formula, Solution: An assignment $x_1 \ldots x_n \in \{0,1\}$ satisfying
Maximum Independent Set ∉ NP
Input: Graph G, Solution: The largest independent set (no edges inside)
Independent Set
Input: Graph G, number k, Solution: An independent set of size ≥ k
Reachability
Input: Graph G, s, t Solution: A path from s to t
Minimum Spanning Tree
Input: Graph, costs $\{c_e\}$, Solution: Minimum cost spanning tree
Break RSA
Input: Encrypted message m, public key pk, Output: Decryption of m
Three-dimensional Matching
Input: Relationship Graph of 3 parties Solution: n valid triples
Longest Path
Input: Graph G, non-negative edges, s, t, g Solution: Path $s \to t$, total weight ≥ g
Independent Set/Vertex Cover
Input: Graph G Solution: set of vertices (no internal edges) / V: all edges covered
Knapsack/Subset Sum
Input: Set of items (weight = value) with one each, capacity W
Solution: A subset of these items such that $\sum w_i = W$
Balanced Cut
Input: Graph G, n vertices, budget b
Solution: subsets S, T where $|S|, |T| \ge n/3$, at most b edges between S and T
Integer Linear Programming
Input: Set of linear inequalities $Ax \le b$, objective function and goal $c \cdot x \ge g$
Solution: Find a non-negative integer that satisfies above
Zero-one equations (ZOE): Find vector x of 0's and 1's satisfying $Ax = 1$

Reductions
search problem $A \le_p$ search problem B: a way to use an algorithm for B to solve A (reduction in polynomial time)
Reductions compose ($A \to B \to C$)

Rudrata Path → Rudrata Cycle
Given an instance $(G = (V,E), s, t)$ of the Rudrata ~~Path~~ problem and an algorithm for Rudrata Cycle, solve Rudrata Path by connecting s, t to a new node x to create cycle. Find cycle, then delete edges.

Rudrata Cycle → Rudrata Path
For each pair of nodes u, v with (u,v) in G, run Rudrata Path on $(G = (V, E - (u,v)), u, v)$. If path, return path + (u,v) to complete

3SAT → SAT
Special case, just run SAT solver given 3SAT clauses

SAT → 3SAT
replace input clauses with > 3 literals $(a_1 \vee a_2 \ldots \vee a_k)$ with $(a_1 \vee a_2 \vee y_1) \wedge (\bar{y}_1 \vee a_3 \vee y_2) \ldots (\bar{y}_{k-3} \vee a_{k-1} \vee a_k)$, at least one must be true

3SAT → 3SAT Constrained
Allowing variables to appear no more than 3 times, replace all x appearing > 3 times by $x_1, x_2 \ldots x_k$ for each k appearance and write $(\bar{x}_1 \vee x_2)(\bar{x}_2 \vee x_3) \ldots (\bar{x}_k \vee x_1)$ → all must have same value

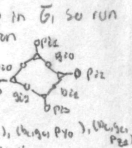
3SAT → Independent Set
Write each clause as a triangle, connecting all instances of $x, \bar{x}$
$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_4 \vee \bar{x}_3) \wedge (x_5 \vee x_6 \vee \bar{x}_2)$ →

If satisfiable, there exists an independent set and vice-versa.

Independent Set → Vertex Cover
Given an instance of independent set (G, g), find a vertex cover of size $|V| - g$ and return the complement
Reverse can be done from vertex cover to independent set

Independent Set → Clique
An independent set in G will be a clique in $\bar{G}$, so run clique algorithm on $\bar{G}$ to obtain solution

3SAT → 3D Matching
Write each variable x_i as the gadget (Note: two possible states)



For each clause, introduce boy and girl b_c, g_c
and for $(x \vee \bar{y} \vee z)$, introduce (b_c, g_c, p_{x1}), (b_c, g_c, p_{y0}), (b_c, g_c, p_{z1})
By earlier reduction, we can further ensure that no variable appears twice, then add $2n - m$ boy-girl couples who love all animals to finish the reduction

3D Matching → ZOE
Given a 3D matching, make x in $Ax = 1$ the triangles used and the columns if that the person is involved in a triangle
write as $Ax = 1$ and find a solution if possible

ZOE → Subset Sum
Look at columns of A as numbers and x as a vector of whether or not they are used to sum to (vec of 1's).
Account for carry by looking at numbers as base $(n+1)$ – # of cols

ZOE → Integer Linear Programming
Write each ZOE equality as two inequalities and write inequalities $x_i \le 1$, $-x_i \le 0$ to constrain domain

ZOE → Rudrata Cycle
Start by reducing to Rudrata Cycle with Paired Edges