CS124 Final Review

May 3, 2020

Complete List of Topics

1. DP ★ last — Dynamic Programming
2. Document similarity
3. Primality Testing — Number Theory, FLT
4. Cryptography — Euclidean Algo (and extended)
5. LP
6. Network Flow + Games — Games (writing and solving LP)
7. NPC
8. Local search
9. Approximation Algos

Review

Starting Toolkit

Approaches:

- Greedy, Divide + Conquer, DP, LP, Randomness

Ideas:

- Modelling (ex: Graphs), Reductions ★ (+ forms and NPC)
- Tradeoffs (time, space, approximation / error, prog runtime time)
- Importance of simplicity: hashing (bloom filter)

1 Notes about the Final

- Material covered in the first half of the class may be on the exam (e.g., dynamic programming), but the final will emphasize the second half of the course.
- These review notes are not comprehensive. Material that does not appear here (e.g., suffix trees, LCA, etc.) is still fair game.

2 Dynamic Programming

1. **Definition:** dimension of lookup table, what are the arguments? Say in words what the function returns. "Let *D[i, j]* …". Think: subproblems.
2. **Recursion:** give both a verbal and mathematical description of the function. Piece-wise function with the base cases!
3. **Analysis:** both of time and space! For time, remember:

   Number of possible inputs · Time to combine recursive calls

For space, start with the naïve space and see if you can improve it by using bottom-up approach instead of top-down.

2.1 Approach

1. Define your subproblem.
2. Define your recurrence relation
3. Define your base cases, and how you will fill up the subproblem matrix
4. Analyze the asymptotic runtime and memory usage.

2.2 Exercises

Exercise. *Suppose you're given a number diamond as follows:*

```
   1
  2 3
 4 5 6
  9 4
   0
```

What's the path from the top to bottom that maximzes the sum of numbers that you pass through? You start at the top end and at each step, you can only move downward. In this case the answer is $1 + 3 + 5 + 9 + 0 = 18$.

Space: Naive is $O(NK)$, can be improved to $O(K)$

Time: # possible inputs × Time to combine possible inputs $= O(N^2K)$ → maximal value K can take is N

Exercise. *How many permutations of* $\{1, 2, \ldots, N\}$ *have exactly* K *inversions? For example, for* $N = 3$ *and* $K = 1$ *the answer is 2:* $\{2, 1, 3\}$ *and* $\{1, 3, 2\}$.

1. **Definition:** let *X[i, j]* be the number of permutations of

   {i, i+1,..., N}

   that have exactly *j* inversions. We want: *X[1, K].*
2. **Recurrence:** at each step we decide where to place the smallest number in the set.

$$X[i, j] = \sum_{l=j-(N-i)}^{\min\{j,K\}} X[i+1, l].$$

Ex: in the set *{i+1,..., N}* we can add between 0 and $N - i$ new inversions depending on where we insert *i* into the set.

Base cases:

$X[N, 0] = 1$ $\quad X[N, j] = 0$ for $j = 1, \ldots, K$ $\quad X[N, i] = 0$ for $i \geq 0$

3 Document Similarity

3.1 Motivation

* most likely T/F question on final

Suppose we have many documents that we wish to compare for similarity in an efficient manner. We may have a search engine, and we don't want to display duplicate results. However, duplicate results (e.g., someone copied an article from another place) aren't exact duplicates, but rather exhibit high *s*imilarity. It's also okay if we don't have 100% accuracy.

3.2 Set Resemblance Problem

First, we discuss a problem that we can reduce document similarity to, as you will see in a moment.

1. Suppose that we have two sets of numbers A and B. Then their resemblance is defined as

$$\text{resemblance}(A, B) = R(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

2. Finding the exact resemblance is $O(n^2)$ if we compare the elements pairwise, and $O(n \log n)$ if we sort first and then compare. We wish to find an approximation of the resemblance in a much more efficient manner.

3. Suppose that we have functions $\{\pi_1, \pi_2, \pi_3, \ldots\}$, where each π_k is a different random permutation from the 64-bit numbers to the 64-bit numbers (or whatever the space of numbers you are using). We define $\pi_k(A) = \{\pi_k(a) : a \in A\}$. Then we find that

$$\Pr[\min\{\pi_k(A)\} = \min\{\pi_k(B)\}] = \frac{|A \cap B|}{|A \cup B|} = R(A, B)$$

   This is because every element in $A \cup B$ has an equal probability of mapping to the minimum element after the permutation is applied.

4. Thus, the more random permutations we use, the more confidence we have about the resemblance of A and B.

3.3 Document Similarity

hash overlapping phrases → #s

1. In order to turn the document similarity problem into the set resemblance problem, we must first transform a document into a set of numbers. We do this by the process of *s*hingling, where we hash overlapping pieces of the document (e.g., every four consecutive words), which gives us a set of numbers. Let this set be called S_D.
2. Now for every document, all we need to do is store is a *s*ketch of the document, which would be an ordered list of numbers $(\min\{\pi_1(S_D)\}, \min\{\pi_2(S_D)\}, \min\{\pi_3(S_D)\}, \ldots, \min\{\pi_{100}(S_D)\})$. The larger the sketch, the more confidence you'll have about the resemblance, but the more memory and time it'll take to process each document.
3. Now to compare the similarity of two documents, we just need to compare their sketches. We'll set a threshold for the number of matches that need to occur in order for two documents to be considered similar. For example, we can say that two documents whose sketches match at 90 out of 100 numbers are considered similar. As you can see, this process of comparing the similarity of two documents is quite efficient!
4. The probability $p(r)$ that at least 90 out of 100 entries in the sketch match stays very low until r approaches 0.9 and then grows very quickly to 1. This property of the function means that we won't make mistakes too often!

4 Primality Testing

4.1 Review

1. *Fermat's Little Theorem*: For all primes p, $a^{p-1} = 1 \mod p$ for all integers a that are not multiples of p.
2. A composite number n is said to be a-pseudoprime if $\gcd(a, n) = 1$ and $a^{n-1} = 1 \mod n$. *Carmichael numbers* are composite numbers n such that n is a-pseudoprime for all a that do not share a factor with n.
3. *Miller-Rabin Primality Test:* Suppose we have an odd number n, where $n - 1 = 2^t u$, and we want to check if it is prime. Then we proceed as follows:
   (a) Randomly generate a positive integer $a < n$.
   (b) Calculate $a^u, a^{2u}, a^{4u}, \ldots, a^{2^t u} = a^{n-1} \mod n$.
   (c) Check if $a^{n-1} = 1 \mod n$. If it is not, n is composite. If it is, let $x = a^{n-1} = a^{2^t u}$ (already calculated). ↳ last step of Rabin-Miller is FLT.
   (d) There are three possibilities for x:
      i. $x = 1$: then if $x = a^u$ end, otherwise let $x = x^{1/2}$ and repeat 4.

* only composite #s have a non-trivial square root
$x \neq \pm 1 \mod n$ and $x^2 = 1 \mod n$

ii. $x = -1$: then the algorithm is indecisive and ends.

iii. $x \notin \{-1, 1\}$: then n must be composite and a is a witness

(e) Repeat the above steps several times until we get bored or n is determined to be composite. If n is not found to be composite, then return that it is (probably) prime.

It turns out that for any composite number n, the probability that a randomly chosen a will reveal it as composite (i.e., be a witness to the compositeness of n) is 3/4. So by iterating enough times, the probability of error can be reduced below the probability that your computer crashes during the execution.

if rabin-miller says n is composite ⇒ it is composite

4.2 Exercises

Exercise. *If n is a 2-pseudoprime, then so is $m = 2^n - 1$*

That *n* is a 2-pseudoprime means:

gcd(2, n) = 1 and $2^{n-1} = 1 \bmod n$.

Show that *m* is composite. This follows from the fact that *n* is composite.

$n = ab$

$m = 2^n - 1$

$m = 2^{ab} - 1 = (2a)^b - 1 = (2^a - 1)[(2^a)^{b-1} + (2^a)^{b-2} + \cdots + (2^a) + 1].$

To show that m is a 2-pseudoprime, we need to show:

$2^{m-1} = 1 \pmod m \iff 2^{m-1} = 0 \pmod m \iff m \mid 2^{m-1} \iff 2^n - 1 \mid 2^{m-1} - 1$

It suffices to show that $n \mid m-1$. Why? If $m-1 = kn$, then

$2^{m-1} - 1 = 2^{nk} - 1 = (2^n)^k - 1 = (2^n - 1)((2^n)^{k-1} + \cdots + 2^n + 1)$

$2^n - 1 \mid 2^{m-1} - 1$

By assumption, *n* is a 2-pseudoprime, which by definition means that

$n \mid 2^n - 1$

Hence,

$m - 1 = 2(2^{n-1} - 1)$

and so $n \mid m-1$ as required.

$m = 2^n - 1 \rightarrow$ 2-pseudoprime

hint: $x^k - 1 = (x-1)(x^{k-1} + x^{k-2} + \cdots + x + 1)$

Exercise. *Show that any prime p passes the Miller-Rabin test.*

What does it mean for *p* to not pass Rabin-Miller test?

It means that *p* has a non-trivial square root of 1; i.e. there **exists a witness** *a* such that

$a^{2^i u} \neq \pm 1 \pmod p$

$a^{2^{i+1} u} = (a^{2^i u})^2 = 1 \pmod p$

Let $x = a^{2^i u} \pmod p$

Then, $x^2 = 1 \pmod p; \quad x \neq \pm 1 \pmod p$

key:

$p \mid (x^2 - 1) = (x-1)(x+1)$

$p \mid ab \Rightarrow p \mid a$ or $p \mid b$

so $p \mid x-1$ or $p \mid x+1$

hence $x = 1$ or $x = -1 \bmod p$

contradiction!

∴ any prime p must pass.

Exercise. *(Challenge) Suppose n is product of distinct primes. If $p - 1 | n - 1$ for each prime p that divides n, then n is either a prime or a Carmichael number.*

5 Cryptography

5.1 Review

1. RSA is an encryption algorithm loosely based on the assumption that factoring is difficult. In order to efficiently implement RSA, many of the algorithms previously discussed in this course (Extended Euclidean algorithm, repeated squaring, Miller-Rabin primality testing) are used.

   - Set-up: Alice uses Miller-Rabin to find a prime numbers p, q and sets $n = pq$. She then chooses $e < n$ and publishes a public key (e, n), and she privately calculates a secret key d such that $ed = 1 \mod (p-1)(q-1)$.
   - Encryption: $E(x, n, e) = x^e \mod n$
   - Decryption: $D(c, d) = c^d \mod n = x^{ed \mod (p-1)(q-1)} \mod n = x$

5.2 Exercises

Exercise. *Prove that RSA is multiplicative: if C_1 is the encryption of M_1 and C_2 is the encryption of M_2 , then C_1C_2 is the encryption of M_1M_2.*

If C_1 is the encryption of M_2, and C_2 is the encryption of M_2, then:

$$C_1 = M_1^e \pmod n$$

$$C_2 = M_2^e \pmod n$$

Multiply:

$$C_1C_2 = M_1^e M_2^e = (M_1M_2)^e \pmod n$$

Exercise. *Let (n, e) be a public key of RSA. Show that if we have an efficient algorithm A which decrypts 1 percent of the messages, we can build an efficient randomized algorithm that decrypts every message with high probability.*

Hint: if you need to decrypt C, pick a random M_0, compute its encryption C_0 and try decrypting CC_0.

"Normal" Euclids Algorithm

Finds gcd(a, b) recursively. Avoids factoring.

```
gcd: Euclid (a, b)
     if b == 0 return a
     return Euclid (b, a mod b)
```

$a \geq b \geq 0$

Correctness: gcd (a, b) = gcd (b, a mod b).

Example: find gcd (270, 192).

$270 = 192 \cdot 1 + 78$

$192 = 78 \cdot 2 + 36$

$78 = 36 \cdot 2 + 6$

$36 = 6 \cdot 6 + 0$ stop

GCD

But we need the extended version.

$a \bmod b \leq a/2$

Euclid (a, b)

Euclid (a', b')

where $a' \leq a/2$

Extended Euclidean Algorithm

Also find x, y st $ax + by = d$, $d = \gcd(a, b)$

gives GCD, returns x and y

2 steps: I) Do Euclidean Alg. as before.

II) Build up.

$270 = 192 \cdot 1 + 78$

$192 = 78 \cdot 2 + 36$

$78 = 36 \cdot 2 + 6$

$36 = 6 \cdot 6 + 0$

gcd

$6 = 78 - (192 - 78 \cdot 2) \cdot 2 = 5 \cdot 78 - 2 \cdot 192$

$6 = 78 - 36 \cdot 2$

x a y b

$6 = 5 \cdot (270 - 192 \cdot 1) - 2 \cdot 192 = 5 \cdot 270 - 7 \cdot 192$

$6 = 5 \cdot 270 - 7 \cdot 192$ ✓

Creating the keys (Bob's)

Bob does:

1) Find primes p, q Compute $n = pq$

2) Choose $e < n$ st e is coprime with $\varphi(n) = (p-1)(q-1)$

Euler's totient function

$\gcd((p-1), (q-1), e) = 1$

3) Find d st $de \equiv 1 \bmod \varphi(n)$ with the extended Euclidean alg. finds x, y st $ex + \varphi(n) y = \underbrace{\gcd(e, \varphi(n))}_{1}$

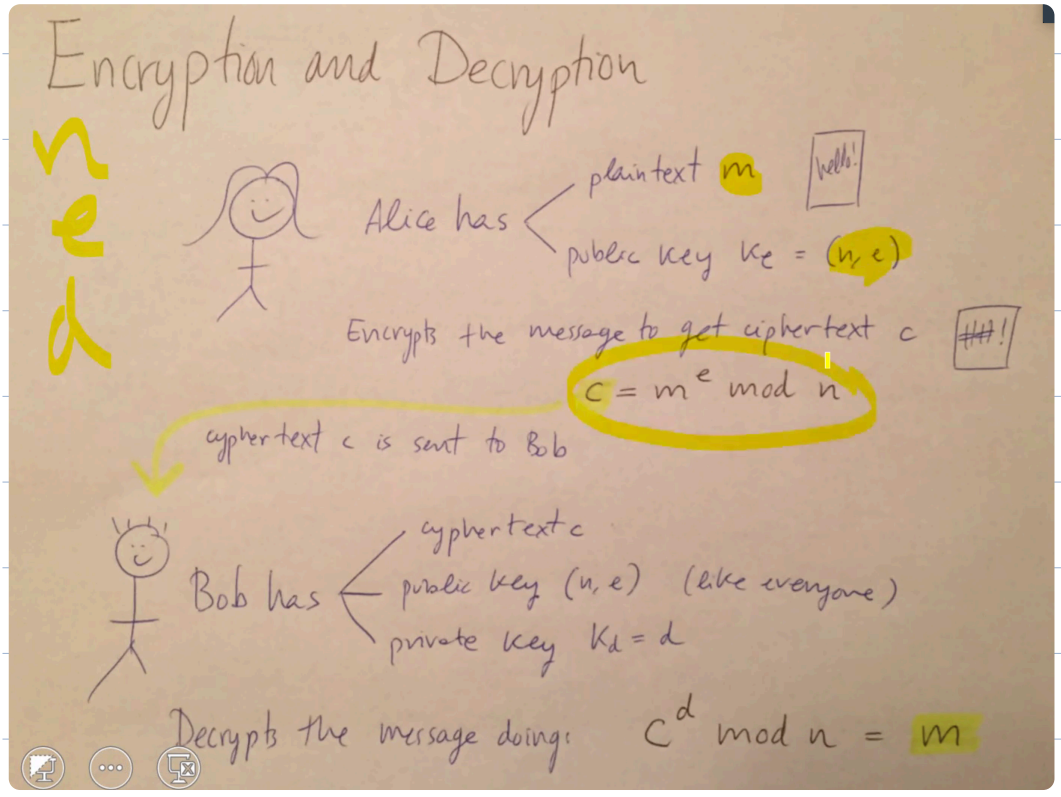
take modulo $\varphi(n)$

$ex + 0 \equiv 1 \bmod \varphi(n)$

4) Public key $K_e = (n, e)$.

$d = e^{-1} \bmod (p-1)(q-1)$

Private key $K_d = d$. more formally, (p, q, d)



Alice sends $e(x) = x^e \bmod n$

Bob computes $d(e(x)) = (e(x))^d \bmod n$

$\longrightarrow$ $d(e(x)) = x$ by construction

6 Linear Programming

6.1 Review

1. *Simplex Algorithm*: The geometric interpretation is that the set of constraints is represented as a polytope and the algorithm starts from a vertex then repeatedly looks for a vertex that is adjacent and has better objective value (its a hill-climbing algorithm). Variations of the simplex algorithm are *not* polynomial, but performs well in practice.
2. Standard form required by the simplex algorithm: *minimization, nonnegative variables and equality constraints.*

multiply obj by -1

$X = X^+ - X^-$

$X^+ \geq 0,\ X^- \geq 0$

$\sum a_i x_i \geq b$ to $\sum a_i x_i - s = b$

↳ slack variable $\sum a_i x_i \leq b$ to $\sum a_i x_i + s = b$

$s \geq 0$

6.2 Exercises

Exercise. *Suppose that we have a general linear program with n variables and m constraints and suppose we convert it into standard form. Give an upper bound on the number of variables and constraints in the resulting linear program.*

Exercise. *Given a weighted, directed graph $G = (V, E)$, with weight function w mapping edges to positive real-valued weights, a source vertex s, and a destination vertex t. Show how to compute the value $d[t]$, which is the weight of a shortest path from s to t, by linear programming.*

games

	m	t
r	3	-1
s	-2	1

row: maximin the minimum

$$\max_x \min_j \sum_{i=1}^{m} G_{ij} x_i$$

$$\max z$$

$$\sum_i x_i = 1 \quad \text{probability constr.}$$

$$z - 3x_1 + 2x_2 \le 0$$

$$z + x_1 - x_2 \le 0$$

column: minimax the max

$$\max w$$

$$\sum_i y_i = 1$$

$$w - 3y_1 + y_2 \ge 0$$

$$w + 2y_1 - y_2 \ge 0$$

dual LPs

$$\max_x \min_y \sum x_i y_j G_{ij} = \min_y \max_x \sum x_i y_j G_{ij}$$

7 Network Flows and Games

max flow min cut!

7.1 Review

1. The way that simplex algorithm works for Max-Flow problem: it nds a path from S to T (say, by DFS or BFS) in the *residual graph* and moves flow along this path of total value equal to the minimum capacity of an edge on the path.

2. A *cut* is a set of nodes containing S but not T. The *capacity* of a cut is the sum of the capacities of the edges going out of this set.

3. Getting the *dual* of a maximization problem: (1) Change all inequality constraints into $\leq$ constraints, negating both sides of an equation if necessary; (2) transpose the coefcient matrix; (3) invert maximization to minimization; (4) interchange the roles of the right-hand side and the objective function (5) introduce a nonnegative variable for each inequality, and an unrestricted one for each equality (6) for each nonnegative variable introduce a $\geq$ constraint, and for each unrestricted variable introduce an equality constraint.

7.2 Exercises

Exercise. *The edge connectivity of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how the edge connectivity of an undirected graph $G = (V, E)$ can be determined by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.*

8 NP-completeness

8.1 Review

1. P is the class of all yes/no problems which can be solved in time which is polynomial in n, the size of the input.

2. P is closed under polynomial-time reductions:

(a) A **reduction** R from Problem A to Problem B is an algorithm which takes an input x for A and transforms it into an input $R(x)$ for B, such that the answer to B with input $R(x)$ is the answer to A with input x.

(b) The algorithm for A is just the algorithm for B *composed* with the reduction R.

Problem A is *at least as hard as* Problem B if B reduces to it (in polynomial time): If we can solve A, then we can solve B. We write:

$$A \geq_{\mathrm{P}} B \quad \text{or simply} \quad A \geq P. \tag{1}$$

3. NP is the class of yes/no problems such that if the answer if yes, then there is a **short** (polynomial-length) **certificate** that can be checked in polynomial time to prove that the answer is correct.

   Examples: Compositeness, 3-SAT are in NP.

   Not-satisfiable-3-SAT is not in NP.

4. **NP-complete**: In NP, and all other problems in NP reduce to it. That is, A is NP-complete if

   $$A \in \mathrm{NP} \quad \text{and} \quad A \geq_{\mathrm{P}} B \ \forall B \in \mathrm{NP}.$$

   NP-hard: All problems in NP reduce to it, but it is not necessarily in NP.

5. NP-complete problems from lecture: circuit SAT, 3-SAT, integer LP, independent set, vertex cover, clique.

8.2 Example: reduction from 3-SAT to 3-COLOR

3-COLOR: Is there a valid 3-coloring of a graph G?

1. Root vertex X. Form triangle with vertices X, T, F.
2. For every variable a in the 3-SAT problem, form triangle with vertices $X, a, \bar{a}$. Notice that any valid 3-coloring of this graph so far will give a truth assignment to each variable.
3. $a \vee b \vee c$ looks like the following figure:

Exercise. *Why does this work?*

8.3 Exercises

8.3.1 NP exercises

Exercise. *Subgraph isomorphism: Consider graphs $G = (V, E)$, $H = (V_2, E_2)$. Does G contain a subgraph (V_1, E_1) which is isomorphic to H?*

(An isomorphism of graphs (V_1, E_1) and (V_2, E_2) is a 1-1 function $f : V_1 \to V_2$ such that the map $(u, v) \mapsto (f(u), f(v))$ is a 1-1 function $E_1 \to E_2$.)

Show that subgraph isomorphism is NP-complete.

Exercise. *3-EXACT-COVER: Given a finite set X of size $3q$ and a collection $\mathcal{C}$ of 3-element subsets of X, does $\mathcal{C}$ contain an* **exact cover** *(i.e. a partition) of X?*

Given that 3-EXACT-COVER is NP-complete, show that 4-EXACT-COVER is NP-complete.

8.3.2 Vertex cover and clique exercises

Exercise. *Vertex cover: Give an algorithm to find the optimal vertex cover for a tree in linear time.*

Exercise. *Maximum clique: Given a graph $G = (V, E)$, we define a new graph $G^{(k)} = (V^{(k)}, E^{(k)})$ as follows: We let*

$$V^{(k)} = V^k = \{\bar{v} = (v_1, \ldots, v_k) : v_i \in V \ \forall i\}.$$

We then say $\bar{v} \sim \bar{v}'$ in $G^{(k)}$ (i.e. that $(\bar{v}, \bar{v}') \in E^{(k)}$) if $v_i \sim v_i'$ in G (i.e. if $(v_i, v_i') \in E$) or $v_i = v_i'$ for each i.

Show that if m is the size of the maximum clique in G, then m^k is the size of the maximum clique in $G^{(k)}$.

9 Local search

9.1 The basic idea

1. Can be thought of as "hill climbing."
2. Importance of setting up the problem: Need to define the notion of **neighbourhood** so that the state space is nice, and doesn't have lots of local optima.
3. Choosing a starting point is important. Also, how we move to neighbours can be important — e.g. whether we always move to the best neighbour or just choose one that does better, and also how we choose to break ties.

9.2 Variations

1. Metropolis rule: random neighbours, with higher likelihood of moving to better neighbours.
2. Simulated annealing: Metropolis with decreasing randomness (cooling) over time.
3. Tabu search: adds memory to hill climbing to prevent cycling.
4. Parallel search, genetic search.

10 Approximation algorithms

10.1 Review

1. Vertex cover: Want to find minimal subset $S \subseteq V$ such that every $e \in E$ has at least one endpoint in S.

   To do this, repeatedly choose an edge, throw both endpoints in the cover, delete endpoints and all adjacent edges from graph, continue. This gives a 2-approximation.

2. Max cut: see homework problem.

   (a) Randomized algorithm: Assign each vertex to a random set. This gives a 2-approximation in expectation.

   (b) Deterministic algorithm: Start with some fixed assignment. As long as it is possible to improve the cut by moving some vertex to a different set, do so. This gives a 2-approximation.

3. Euclidean traveling salesman: Find MST, create "pseudo-tour," take shortcuts to get tour. This gives a 2-approximation.

4. MAX-SAT: Asks for the maximum number of clauses which can be satisfied by any assignment.

5. LP relaxation and randomized rounding.

10.2 Example: k-center clustering

Suppose we are given a set of cities represented by points in the plane, $P = \{p_1, \ldots, p_n\}$. We will choose k of these cities in which to build a hospital. We want to minimize the maximum distance that you have to drive to get to a hospital from any of the cities p_i. That is, for any subset $S \subset P$, we define the cost of S to be

$$\max_{1 \le i \le n} \operatorname{dist}(p_i, S) \quad \text{where} \quad \operatorname{dist}(p_i, S) = \min_{s \in S} \operatorname{dist}(p_i, s).$$

This problem is known to be NP-hard, but we will find a 2-approximation. The basic idea: Start with one hospital in some arbitrary location. Calculate distances from all other cities — find the "bottleneck city," and add a hospital there. Now update distances and repeat.

Exercise. *Come up with a precise description of this algorithm, and prove that it runs in time $O(nk)$.*

Exercise. *Prove that this gives a 2-approximation.*

CLRS 21.1-6

calculate max flow, src if ≥ 2

CLRS 21.2-11

choose source

run max flow to every sink

return min (max flows)

CLRS 21.2-12

there must $\exists$ a cycle

CLRS 21.2-13

minimum cut w/ smallest # of edges

→ increase capacity by $\frac{1}{|E|+1}$

since integral flow, flow won't change

CLRS 29.2-3

CLRS 29.2-7

CLRS 35.1-3

Tricks for Tricky DP

Palindrome Partition

$$P(s) = \begin{cases} \text{if } s \text{ not a palindrome}: \min_{i \in \text{len}(s)} \left(P(s[0:i]) + P(s[i:]) + 1 \right) \\ \text{else}: \text{return } 0 \end{cases}$$

Mobile Keypad

$$D(p, n) = \sum_{p' = \{\text{moves}\}} D(p', n-1)$$

base case: $D(p, 0) = 0$

Painter's Partition

insert $k-1$ partitions and make sth equal

$$P(i, k) = \min_{j \in (i, n)} \left(\max \left(P(j, k-1), \sum_{l=i}^{j} A_l \right) \right)$$

base case: $P(i, k=1) = \sum_{l=i}^{n} A_l$