Final Review

Problem 1

Ex: $\overset{A}{1}, \overset{B}{3}, \overset{A}{100}, \overset{B}{2}$ A scores 101, B scores 5

Let $a_1, a_2, \ldots, a_n$ be the card values.

Idea: ~~OPT(i, j) After our move, what~~

☆IMPORTANT☆ ① $OPT(i,j)$ = value of game for first player on game with cards $a_i, \ldots, a_j$.

$= \max\{\text{value of taking } a_i, \text{ value of taking } a_j\}$

$= \max\{a_i + \sum_{k=i+1}^{j} a_k - OPT(i+1, j),\ a_j + \sum_{k=i}^{j-1} a_k - OPT(i, j-1)\}$

② $= \sum_{k=i}^{j} a_k - \min\{OPT(i+1, j), OPT(i, j-1)\}$

③ Base cases: $OPT(i,i) = v_i$ for $i = 1, \ldots, n$

④ Fill out cells in increasing order of substring length (i.e. # of cards in game)

⑤ Return $(OPT(1,n), \sum_{k=1} v_k - OPT(1,n))$

runtime: $O(n^2)$ entries, each of which takes $O(n)$ time to compute, so $O(n^3)$

better runtime: Pre-compute $\sum_{k=i}^{j} a_k$ for all (i, j) in $O(n^2)$ time. Then, we have $O(n^2)$ entries each of which takes $O(1)$ time to compute, so $O(n^2)$ total.

Problem 2

Idea: In any parenthesization, there must be a last operation, and any operation can be last.

~~OPT(~~

Let our expression be $a_1 o_1 a_2 o_2 a_3 \dots o_{n-1} a_n$. Then, let $OPT(i,j,v)$ = True if we can parenthesize $a_i o_i \cdots o_{j-1} a_j$ to evaluate to something congruent to v mod m, False otherwise.

$$= \bigvee_{i \le k \le j-1} [a_i o_i \cdots o_{j-1} a_j \text{ can evaluate to } v \text{ if } o_k \text{ is the last operation}]$$

$$= \bigvee_{i \le k \le j-1} \Big[\bigvee_{\substack{(u,w):\ 0 \le u,w \le m-1 \\ \text{and } u\, o_k\, w \equiv v \bmod m}} [OPT(i,k,u) \wedge OPT(k+1,j,w)] \Big]$$

Base cases: $OPT(i,i,v)$ = True if $a_i \equiv v$ mod m, False otherwise for $i = 1, \dots, n$

Fill out OPT in increasing order of $j-i$.

Return $OPT(i,j,0)$

runtime: $O(n^2 m)$ entries, each of which takes $O(nm)$ time to compute, so $O(n^3 m^2)$ total.

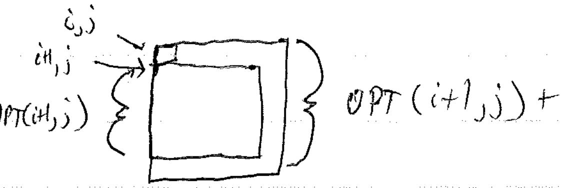
Problem 3

Idea: Let $A[1..n][1..n]$ be the table. Then, let

$$OPT(i,j) = \text{side length of largest square of all 1's w/ upper left corner at } (i,j)$$

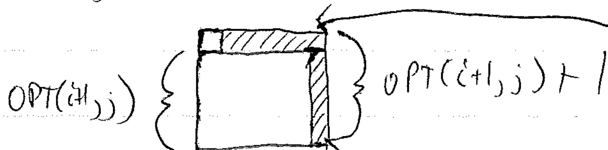
$$= \begin{cases} \min\{OPT(i+1,j), OPT(i,j+1), OPT(i+1,j+1)\} + 1 & \text{if } A[i][j]=1 \\ 0 & \text{otherwise} \end{cases}$$

To show correctness, note that if $\min\{\cdot,\cdot,\cdot\} = OPT(i+1,j)$, then the biggest square at (i,j) has side length at most $OPT(i+1,j)+1$.



← If this is all 1's, this contradicts the maximality of $OPT(i+1,j)$.

Also in this case, the biggest square at (i,j) has side length at least $OPT(i+1,j)+1$.



all 1's because $OPT(i,j+1) \geq OPT(i+1,j)$

all 1's because $OPT(i+1,j+1) \geq OPT(i+1,j)$

Hence, if $\min\{\cdot,\cdot,\cdot\} = OPT(i+1,j)$, $OPT(i,j) = OPT(i+1,j)+1$ satisfies the specification.

Similar arguments hold in the other cases.

Base cases are $i=n$ or $j=n$, where $OPT(i,j)=1$ if $A[i][j]=1$ or 0 otherwise.

Fill out table in rows from right to left.

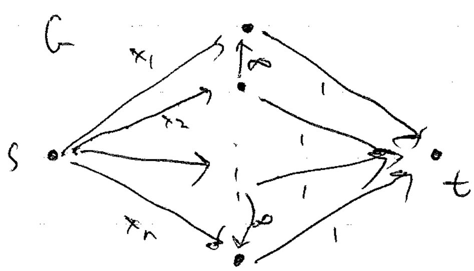
Return maximum over all entries of OPT.

run-time: $O(n^2)$, since we have $O(n^2)$ entries that each take $O(1)$ time to fill. $O(n^2)$ time to compute maximum at end.

Problem 4

Idea: This is matching, where each starting coupon is matched to its final form after trades. Reduce to bipartite matching.

Alternate idea: Each flow represents a sequence ~~of~~ ~~trades~~.



Let $x_1, x_2, \ldots, x_n$ be your starting counts of each coupon.

Vertices:

- s and t
- One per coupon type, $v_1, \ldots, v_n$

Edges:

- $s \to v_i$ of capacity x_i for all i
- $v_i \to t$ of capacity 1 for all i
- $v_i \to v_j$ of capacity ∞ if one can trade a coupon of type i for one of type j.

The $s \to v_i$ edges force x_i units of flow to start at each vertex v_i. The $v_i \to v_j$ edges allow any unit of flow to travel from its starting vertex to any vertex corresponding to a coupon that vertex could trade for. Since each vertex has a capacity 1 edge to t, there is a max flow of n iff there is a way to swap coupons so that exactly one unit of flow ends at each v_i.

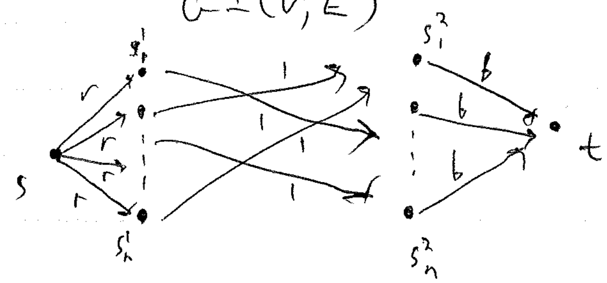
Ford-Fulkerson runs in time $O(n^3)$, $O(|E||V|)$ algorithm also in $O(n^3)$.

run-time: $|V| = n+2$, $|E| \leq n^2 + 2n$, max flow $= n$

Problem 5

Idea: We want to match sensors to backup sensors, so this is matching. Use max flow.

$G = (V, E)$



Vertices:

- s and t
- s_i^1 and s_i^2 for each sensor s_i

Edges:

- $s \to s_i^1$ of capacity r for all i
- $s_i^2 \to t$ of capacity b for all i
- $s_i^1 \to s_j^2$ if s_j can be in the backup set of s_i

$\iff i \neq j$ and $d((x_i, y_i), (x_j, y_j)) \leq d$

Return "possible" if max s-t flow is rn; "impossible" otherwise.

run-time: $O(n^2)$ to create graph

For computing max flow,

$|V| = 2n + 2$

$|E| \leq n^2 + 2n$

max flow value $= rn$,

so Ford-Fulkerson gives $O(rn^3)$

and $O(|E| \cdot |V|)$ algorithm gives $O(n^3)$.

Hence, $O(n^3)$ is best time.

Problem 6

Given $G=(V,E)$, ask

(a) Reduce from Hamiltonian Path. ~~Ask~~ the oracle whether there is a simple path with at least ~~n edges~~ $|V|$ edges.

(b) Reduce from 3-SAT. Given an instance with m clauses, ask the oracle whether it is possible to satisfy m clauses.

(c) Reduce from decision Independent set. Given (G, k), create a new graph G' on the same vertices where any two vertices in G' have an edge between them iff they do not in G. Ask the oracle whether G' has a subset of $\leq k$ vertices with at least $k(k-1)/2$ edges between them.

Problem 7

Decision version: Given $G = (V,E)$ and K, determine whether there exists $S \subseteq V$ such that $|\Gamma(S) \setminus S| \geq k$.

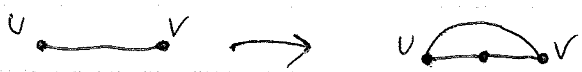
Call the decision problem Decision Max Fringe, or DMF.

$DMF \in NP$:

Let witness be a subset of V. In quadratic time, we can compute $\Gamma(S) \setminus S$ and check if it has size $\geq k$. This is a poly-time verification procedure.

DMF is NP-hard: Assume that G is connected.

Reduce from decision Vertex Cover. Given (G, k), create G' by replacing each edge in G with the following gadget.



where $|V| = n$
$|E| = m$

Ask the oracle whether $(G', n+m-k)$ is a "yes" instance of DMF, and return same answer as the oracle. This reduction runs in linear time.

Correctness:

Want to show

G has VC of size $\leq k \iff \exists S$ in G' such that $|\Gamma(S) \setminus S| \geq n+m-k$

$\Rightarrow$: Let S be the VC. Every vertex not in S must be in $\Gamma(S)$ since if it has an incident edge, the vertex across that edge must be in S. Hence, $|\Gamma(S) \setminus S| = \cancel{n-|S| \geq n-k}$
$n+m-|S| \geq n+m-K$

$\Leftarrow$: Let S be such that $|\Gamma(S)\setminus S| \geq n+m-K$. WLOG, assume that S contains only vertices in V. We can assume this because in each gadget, ~~[illegible]~~ ~~switching~~ switching the middle vertex for either u or v cannot decrease $|\Gamma(S)\setminus S|$. Then, for each middle vertex not in $\Gamma(S)$, add a corresponding u or v to S. This also does not decrease $|\Gamma(S)\setminus S|$.

Then, S is a vertex cover of G since for every edge, at least one incident vertex is in S. Since $|\Gamma(S)\setminus S| \geq n+m-K$ in G, $|S| \leq K$.

$OMF \leq_p DMF$:

Use standard binary search to find best value K of $|\Gamma(S)\setminus S|$.

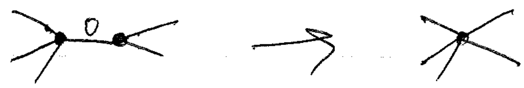
~~Let $V = \{v_1, v_2, \ldots, v_n\}$ and set $K_? = K+n+1$~~

For each vertex, add ~~a lot of~~ $n+1$ vertices connected to only that vertex to force that vertex to be part of S, and ask the oracle whether we can achieve $K+n+1$ on the new graph. If yes, leave the extra vertices in, and if no, take them out. Adjust K to take into account all the extra vertices we leave in.

Problem 8

(a) pseudopolynomial = polynomial in size of input (in bits) + all values in input

Let $G = (V, E)$, and fuse together all vertices connected by 0-weight edges beforehand. Let $w(u,v)$ be weight of



edge between u and v.

Let

$$OPT(v, x, w) = \begin{cases} \text{True} & \text{if } \exists \text{ a path from } s \text{ to } v \text{ of length } w \text{ that uses at most } x \text{ edges,} \\ \text{False} & \text{otherwise} \end{cases}$$

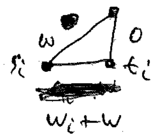
$$= \bigvee_{u:(u,v)\in E} OPT(u, x-1, w - w(u,v))$$

Rest of algorithm left as exercise.

run-time: $O(n\ell^2)$ entries, each of which takes $O(n)$ time to compute, so $O(n^2\ell^2)$ total.

(6)

Idea: Reduce from Subset Sum. Suppose we are given $w_1, \dots, w_n$, and we want to determine if a subset of these sums to w. Then for each w_i, create the gadget



For all $1 \le i \le n-1$, connect t_i to s_{i+1} with an edge of weight 0, and set $s=s_1$, $t=t_n$. Ask the oracle whether there is an $s-t$ path of weight $(n+1)w$, and return the same answer as the oracle. This runs in linear time.

Correctness:

$\exists$ $s-t$ path of weight $(n+1)w \iff \exists$ subset of $w_1 \dots w_n$ that sums to w

$\Rightarrow$: To get from s to t, we must pass through all s_i and t_i. Getting from s_i to t_i adds at least w to the total, so any $s-t$ path has weight at least nw. If we ever take two nonzero edges in the same gadget, we must take at least three nonzero edges, which adds a minimum of $2w$ to the total weight, so total weight is at least $(n+2)w$, which is too large.

Hence, we take exactly one nonzero edge in each gadget, and $\sum w_i = w$ where the sum is over all gadgets i where we take the edge of weight w_i+w.

$\Leftarrow$: For each i, take a path of weight w_i+w from s_i to t_i if w_i is in the set that sums to w, or the path of weight w otherwise.