6.046 Exam 2 Cheat Sheet

Early lecture trivia

Matrix Multiplication Two $n \times n$ matrices can be multiplied in $O(n^{2.376})$ time.

Lecture 8: Greedy Algorithms and MST

Kruskal's Algorithm Pick the lowest-weight edge such that no cycles are formed at each step.

Prim's Algorithm Pick a starting vertex and grow from there, on any vertex yet spanned.

Lecture 9: Gradient Descent

placeholder

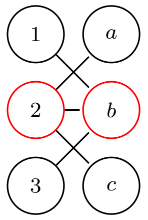
Lecture 10: Maxflow-Mincut

placeholder

Lecture 11: Vertex Cover and Matching

Vertex Cover A set of vertices in a graph that touch all edges in a graph.

For instance:

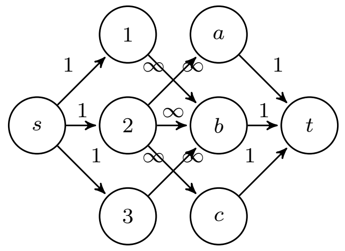


Goal: Given an undirected graph, find a vertex cover of minimum size.

This problem is NP-Complete. We know that these are all solvable within $O(e^x)$ time, but not that they don't have a polynomial time algorithm solution.

Bipartite Graphs $G = (L \cup R, E \subseteq L \times R)$ for disjoint sets of vertices L and R.

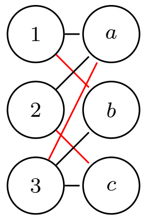
A helpful way to think about this is to sandwich the bipartite graph between a start and end node (source and sink). Furthermore, we change the undirected graph to make it directed, such that each edge is directed from L to R, with infinite capacities (a la Maxflow).



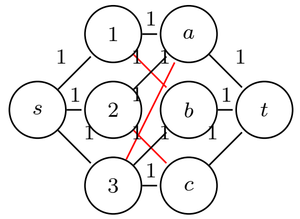
The size of the mincut of G' is the size of the Minimum VC of G.

Matching : A set of edges of a graph that share no endpoints.

Let the below graph be G. A matching is highlighted.



As before, set up an s and a t node. This time, set all capacities to 1, rather than ∞. We call the below graph G''.



Goal: Given a graph, find maximum matching $|M| \leq \frac{|V|}{2}$.

Konig's Theorem (1931) : In bipartite graphs, the size of the maximum matching is the same as the size of the minimum vertex cover.

Alternating Path : A path in a bipartite graph where every second edge is used in the matching.

Augmenting path : An alternating path where both the first and the last vertices are unmatched.

If you flip what you consider matched and unmatched in an augmenting path, then you get a matching of bigger size. Not sure what I meant by this when I wrote it, but I'm keeping it in the cheat sheet for the exam.

Lecture 12: Linear Programming

Linear Program Minimize or maximize a linear objective function subject to linear inequalities.

Standard LP Form $\max \vec{c} \cdot \vec{x}$ subject to constraints $A\vec{x} \leq \vec{b}$ and $\vec{x} \geq 0$.

Simplex Algorithm $\vec{x}$ walks from vertex to vertex in direction of $\vec{c}$. Worst-case exponential.

Interior-Point Method $\vec{x}$ moves inside polytope vaguely $\vec{c}$-wards. $O(n^{3.5} \cdot L^2)$, where n is the number of variables and L is the number of bits required to describe the constraints to the LP.

Strong LP Duality A maximization problem in standard from can be transformed into the **dual** as follows: $\min \vec{b} \cdot \vec{y}$ s.t. $A^\top \cdot \vec{y} \geq \vec{c}$ and $\vec{y} \geq 0$.

The dual has the same solution to the optimization problem as the other LP, but no other points overlap.

Weak LP Duality $\vec{c} \cdot \vec{x} \leq OPT_{\max} \leq OPT_{\min} \leq \vec{b} \cdot \vec{y}$

Lecture 13: Game Theory and the MinMax Theorem

Stable Strategy A set of choices for a given game from which no agent increases expected value by diverging from.

Nash Equilibrium A stable state for a given game.

Nash's Theorem "In every finite game, (finite # players and finite # strategies for each), $\exists$ a collection of randomized strategies (one per player) s.t. no player can improve their expected payout by unilaterally changing their strategy."

You can prove the probabilities for a Nash equilibrium by assigning variable probabilities (x_1, x_2, etc.) for one player, and setting it up as a linear program with $\mathbb{E}[option] \geq z$, for each option, and the sum of the probabilities is 1.

By strong LP duality, the solutions to each of those LPs are the opposite of one another ($val_{P1} = -val_{P2}$), in a Nash Equilibrium.

Lecture 14: Dynamic Programming

Memoizing Implement recursive algorithm using look-up table. If the entry is null, it hasn't been solved before.

Iterating Subproblems to be solved in an order which never requires the solving of subproblems that haven't been solved before. For instance, solving problems for all leaves and then working up the tree.

Catalan Numbers The number of binary trees with n nodes is the n^{th} Catalan number $C(n) = \frac{1}{n+1}\binom{2n}{n} \approx \frac{4^n}{n^{3/4}\sqrt{\pi}}$

Lecture 15: All-Pairs Shortest Path

Single-Source Shortest Path (SSSP) Given a directed graph $G(V, E)$, vertex $s \in V$, and edge weights $w : E \to \mathbb{R}$, find $S(s, v)$, the shortest-weighted path from s to $v \in V$, for all v.

Algorithm	Time	Conditions
BFS	$O(V+E)$	$w(e) = 1\, \forall e \in E$
Dijkstra	$O(E + V \log V)$	$w(e) \geq 0\, \forall e \in E$
Bellman-Ford	$O(VE)$	general
Toposort + BF	$O(V+E)$	acyclic graph

All-Pairs Shortest Path given $G(V, E, w)$, find $S(u, v)\ \forall u, v \in V$. There's another general algorithm for this specific problem that runs in $O(VE + V^2 \log V)$.

Floyd-Warshall Algorithm

```
$C = w(u, v)$
for $k = 1..n$
    for $u \in V$
        for $v \in V$
            if $c_{uv} > c_{uk} + c_{kv}$: // Relaxation
                $c_{uv} = c_{uk} + c_{k_v}$
```

Johnson's Algorithm Better than Floyd-Warshall for sparse graphs.

1. Find function $h : V \to \mathbb{R}$ such that $w_h(u,v) = w(u,v) + h(u) - h(v) \geq 0 \, \forall u,v \in V$, or determine that a negative-weight cycle exists.
2. Run Dijkstra's algorithm on (V, E, w_h), for every source vertex to get $S_h(u,v) \, \forall u,v \in V$.
3. $S(u,v) = S_h(u,v) + h(u) - h(v)$

You find h by solving a system of difference constraints (yay LPs!) for$h(v) - h(u) \leq w(u,v) \, \forall u,v \in V$.

Lecture 16: NP-Completeness

Optimization Problem Find the best-case solution to the problem. Ex for MST: Find the cheapest spanning tree. **Search Problem** Find a solution that satisfies some search constraint. Ex for MST: given $G = (V, E, w)$, and a budget L, find a spanning tree of cost $\leq L$ (or report that none exists). **Decision Problem** Does a solution exist, given constraints? Ex for MST: Given G, budget L, is there a spanning tree of cost $\leq L$?

For any given algorithm, decision is easier than search is "easier" than optimization, since you can solve search given optimization, and decision given search. So it's best to write an algorithm for optimization, and to prove hardness for decision.

Polynomial-time (P) A decision problem Π is in P if there is an algorithm A with running time $|x|^c$ for some constant c such that $A(x)$ outputs the right answer (yes/no).

Non-deterministic polynomial time (NP) . These are problems that can be verified in polynomial time. Given a solution yes/no to the decision problem, and a polynomial length certificate C that this is the right answer, there exists a polynomial time verification algorithm V such that $V(x,c) = \text{yes} \Leftrightarrow x$ is a yes instance.

Reductions Create a mapping between an unknown problem Π_1 and a known problem Π_2 whose runtime is known. If there is such a reduction, we say $\Pi_1 \leq_P \Pi_2$ if that reduction takes P time to run.