CS 170 Midterm 2 Cheat Sheet

Recurrence Relations

Memoization is faster than bottom-up because you can discard some subproblems

Longest increasing subsequence: $L(j)$ is the longest increasing subsequence ending at j. The recurrence needs to explicitly know the last element in the subsequence so we can see if new element can be in it or not

$$L(j) = 1 + \max_{k:k<j, s[k]<s[j]} \{L(k)\}$$

Runs in $O(n^2)$, as there are $O(n)$ subproblems which each take $O(n)$ time to maximize over.

Knapsack with repetition: $K(w)$: Most value we can get from n items in bag of capacity w

$$K(w) = \max_i \{K(w - w_i) + v_i\}$$

Runs in $O(nw)$ time (w subproblems, each maximizing over n items).

Knapsack without repetition: $K(w, j)$: Most value we can get from items $[1, \ldots, j]$ bag of capacity w such that each item is used up to once.

$$K(w, j) = \max\{K(w - w_j, j - 1) + v_j, K(w, j - 1)\}$$

Runs in $O(nw)$ tyime (nw subproblems which each take $O(1)$ time).

Chain matrix multiplication: $C(i, j)$ is the optimal way to multiply matrices $A_i \ldots A_j$ using the least operations.

$$C(i, j) = \min_{i \le k < j} \{C(i, k) + C(k + 1, j) + m_{k-1} \cdot m_k \cdot m_{j+1}\}$$

Time complexity: $O(n^3)$ since we have n^2 subproblems which can be solved in $O(n)$ time.

All pairs shortest path (Floyd-Warshall): $d(i, j, k)$ is shortest path from i to j using vertices $\in [1, \ldots, k]$

$$d(i, j, k) = \min\{d(i, k, k - 1) + d(k, j, k - 1), d(i, j, k - 1)\}$$

Runs in $O(|V|^3)$ time since there are $O(|V|^3)$ subproblems which each take $O(1)$ time

Traveling salesman: Naiive solution takes $O(n!)$ to try every path. Let $C(S, i)$ be the shortest path from 1 to i including all vertices in S

$$C(S, i) = \min_{j \in S; i \neq j} \{C(i, S/\{j\}) + d_{ij}\}$$

Runs in $O(n^2 \cdot 2^n)$ time since there are $n \cdot 2^n$ subproblems which each take n time. Fastest algorithm with exact solution. (Held-Karp)

Union Find

Represent sets as directed trees - useful to see if two items are in same set - do they have same root?

Union find without path compression: (rank = tree height)

procedure MAKESET(x)
  $\pi(x) = x$
  $rank(x) = 0$

procedure FIND(x)
  while $x \neq \pi(x)$ **do** $x = \pi(x)$

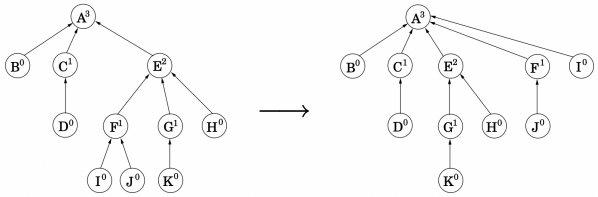
procedure UNION(x, y)
  $r_x = find(x)$
  $r_y = find(y)$
  if $r_x = r_y$ **then** return ▷ sets are same, do nothing
  if $rank(r_x) > rank(r_y)$ **then** $\pi(r_y) = r_x$
  else
    $\pi(r_x) = r_y$
    if $rank(r_x) = rank(r_y)$ **then**
      $rank(r_y) = rank(r_y) + 1$

1. For any x, rank(x) < rank($\pi(x)$)
2. Any node of rank k has at least 2^k descendants
3. If there are n elements overall, there can be up to $\frac{n}{2^k}$ nodes of rank k

Time complexity: $O(|E|log|V|)$ for union and find operations path compression: change FIND, (rank $\neq$ tree depth)

procedure FIND(x)
  if $x \neq \pi(x)$ **then** $\pi(x)$ = FIND($\pi(x)$)

Possible rank values $\in [0, \log n]$ Divide elements by rank into groups of $\{k + 1, \ldots 2^k\}$, so there are $\log^* n$ groups total.
$\log^* n$ = height of 2-power tower to get n, $\log^* 4 = 65536, \log^* 5 = 2^{65532}$
FIND on nodes in a given group takes n time in total - need to "pay"if parent is in same group, which can happen up to $\frac{n}{2^k}$ times, after that, never need to "pay"again. Subsequent FIND calls to given node take $O(1)$ time as parent has been set. Average time of $O(m \log^* n)$ + up to $O(n \log^* n)$ for m FIND operations



Horn SAT

1. While there is an implication that is not satisfied: set RHS to true (since $F \implies T$ is true)
2. Repeat until there are no unsatisfied clauses

If all pure negative clauses aren't satisfied, then not satisfiable
If a certain variable is true in in the greedy solution, then it must be true in all solutions, since we set the fewest variables possible to true.

Linear Programs

Vertex of the feasible region: all n inequalities are tight. An adjacent vertex shares values at $n-1$ constraints and differs by only 1.
Simplex algorithm:

1. Check if any variables can be increased without worsening objective
2. Choose a variable x_i with positive coefficient in the objective and increase it until all inequalities are tight
3. Change variables s.t. new point is the origin by defining $y_i = b_i - a_i \cdot x$ for constraint $a_i \cdot x \leq b_i$ (the one that we hit increasing x_i)
4. Repeat from the origin in new coordinate system until our solution is optimal

Simplex runs in $O(mn)$ time per iteration, but may take exponentially many iterations, making it exponential time.
Interior point and ellipsoid algorithms run in polynomial time.
Primal and dual: Primal $\leq$ dual (weak duality) and primal = dual (strong duality) (# of vars in primal = # of constraints in dual)

$$\max_x c^T x \text{ s.t. } \begin{cases} a_i^T x \leq b_i \text{ for } i \in I \\ a_i^T x = b_i \text{ for } i \in E \\ x_j \geq 0 \text{ for } j \in N \end{cases}$$

$$\min_y b^T y \text{ s.t. } \begin{cases} a_i^T y \geq c_i \text{ for } j \in N \\ a_i^T y = c_i \text{ for } j \notin N \\ y_i \geq 0 \text{ for } i \in I \end{cases}$$

The dual serves as a "certificate of optimality", as the dual variables y_i give us multipliers for the constraints on the primal which results in a tight bound.

Multiplicative Weights

Expert i incurs a loss on day t of $\ell_i^{(t)} \in [0,1]$. Minimize regret on day T

$$R_T = \sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} \ell_i^{(t)} - \min_i \sum_{t=1}^{T} \ell_i^{(t)}$$

Multiplicative weights algorithm:

- At timestep t, weight expert i with $w_i^{(t)}$. Start with $w_i^{(0)} = 1 \forall i$
- Create probability vector $x^{(t)}$ based on $w^{(t)}$ s.t. $x_i^{(t)} = \frac{w_i^{(t)}}{\sum_j w_j^{(t)}}$
- After each timestep, update w based on the experts' performance s.t. for some pre-chosen ϵ, $w_i^{(t+1)} = w_i^{(t)} \cdot (1-\epsilon)^{\ell_i^{(t)}}$
- Bounds regret very well: after T steps, $R_T \leq T\epsilon + \frac{\ln n}{\epsilon}$ where n is # of experts. $\epsilon = \sqrt{\frac{\ln n}{T}} \implies R_T \leq 2\sqrt{T \ln n}$

Zero Sum Games

Pure strategy: deterministic
Mixed strategy: probability distribution over outcomes
If one person announces their strategy first, the other person can always create a deterministic strategy (no assumption that either player plays optimally)
Payoff matrix gives payoffs for row player by default - Payoff matrix for Column is simply the matrix with every weight multiplied by -1
For row strategy r, column strategy c, and payoff matrix P:

$$\text{Payoff} = \sum_{i,j} r_i c_j P_{ij}$$

Mixed strategies do at least as well as pure strategies: if pure strategy is optimal, then mixed strategy will become pure.
Optimal strategy - maxmin - $\max_i \min_j \{P_{ij}\}$ (Row knows Column will try to minimize the payoff, so they pick the row with the highest min value). If both players play optimally, can announce strategies beforehand with no disadvantage
Can formulate as an LP to find row's strategy:

$$\max_{x,t} t \text{ s.t. } \begin{cases} \forall j, t \leq p^{(j)T} x \\ \sum_i x_i = 1 \\ \forall i, x_i \geq 0 \end{cases}$$

The optimal is where each row has the same expected payoff
Column's optimal strategy is simply the dual of this (still calculates Row's payout, multiply by -1 to find Column's payout)
Simple form to find Column's strategy:

$$\min_{y,t} t \text{ s.t. } \begin{cases} \forall i, p_i^T y \leq t \\ \sum_i y_i = 1 \\ \forall i, y_i \geq 0 \end{cases}$$

The payout of a game is unique, but the optimal strategies may not be unique

Set Cover

Greedy approximation: $C(n,m)$ for items $1 \ldots n$ and sets $s_1 \ldots s_m$
If the real optimal solution has k sets:

- Let n_t be number of uncovered elems after iteration t.
- If k sets cover n_t elements in the optimum then one set must cover at least $\frac{n_t}{k}$ elements
- Thus, $n_{t+1} \leq n_t - \frac{n_t}{k}$
- So $n_t \leq n(1-\frac{1}{k})^t$
- Greedy stops adding sets when $n_t < 1$ (all elems covered) which we can approximate as $k' \leq k \ln n$ sets

Max Flow

Conservation of flow: $\forall v \notin \{s, t\}$ flow in = flow out
Max flow as LP: Maximize flow out of s such that flow is conserved, flow is nonnegative, flow satisfies capacity constraints

$$\max \sum_{(s,u)\in E} f_{su} \text{ all outward flow from } s$$
$$\forall e \in E, 0 \leq f_e \leq c_e$$
$$\sum_{(w,u)\in E/\{s,t\}} f_{wu} = \sum_{(u,z)\in E/\{s,t\}} f_{uz}$$

Algorithm:

1. Initialize the residual graph G' by copying all forward edges and adding back edges w/ capacity 0 wherever there is a forward edge.
2. Find a valid path from s to t in the residual graph and push as much flow as possible along that path.
3. Subtract flow pushed along all forward edges and add the corresponding amount to the capacity of the back edge.
4. Repeat until the residual graph doesn't have any more $s \to t$ paths.

Runs in $O(|E|)$ time per iteration, with up to $O(|V||E|)$ iterations in Edmonds-Karp (uses BFS and picks shortest path each time) so total runtime is $O(|V||E|^2)$.
Ford-Fulkerson can use any path-finding algorithm, so $O(f)$ iterations depending on algorithm used, or $O(|E|f)$ time. If DFS used, may repeatedly push flow back and forth through the same edge a large amount of times ("DFS gets unlucky")
Properties of min cut:

- Separates vertices into S, set of vertices reachable in the final residual graph from s, and T, set of vertices in the residual graph reachable from t.
- All forward edges across cut are fully saturated
- Backward edges across cut have zero flow

Reducing the capacity of any edge in a min cut will reduce the max flow, but increasing the capacity of an edge will only increase flow if that edge is part of every min cut (bottleneck edge).