## Union Find Disjoint Set (UFDS)

rank(x): height of subtree below node x (exclusive)
- $\forall x$ rank(x) $\leq$ rank($\pi(x)$) with equality iff $x = \pi(x)$
- Any root node of rank(r) = k has $\geq 2^k$ nodes in subtree
- For n nodes, at most $\frac{n}{2^k}$ nodes of rank k
- [corollary] All nodes have rank $\leq \log n$ $\therefore$ find, union takes $\log n$

Path Compression: A sequence of n make-sets and m unions/find takes $O(n + m \cdot \alpha(m, n))$

Proof for $O((n+m)\log^* n)$ where $\log^*(2\uparrow i) = i$
- Group nodes by rank; consider $[0], [1], (1,2], \ldots (2\uparrow i, 2\uparrow(i+1)]$
- Largest nonempty interval is $i = \log^* n - 1$
- # of nodes in $(2\uparrow i, 2\uparrow(i+1)] \leq \frac{n}{2\uparrow i}$

## Linear Programming (LP)

Definitions:
- Vertex: x s.t. x is feasible and n of constraints are tight (n is the dimension of the problem i.e. $x \in \mathbb{R}^n$)
  - For every subset of n constraints, solve for point of intersection, check for feasibility.
- Unbounded: Objective value $p^* = \infty$
  - Primal unbounded $\Rightarrow$ dual infeasible and has no vertices
  - Can check for boundedness by using linear combinations of constraints
- Feasible: $p^* < \infty$ $\exists$ solution in feasible region
  - Primal feasible $\Rightarrow$ dual feasible and lower-bounded
  - Dual feasible $\Rightarrow$ primal upper-bounded

Facts:
- Feasible region of LP is always convex
- $\forall$ LP, $\exists$ x a solution that is a vertex
  $\Rightarrow$ find all vertices, get optimal value (motivation for Simplex)

## Duality

| Primal LP | Dual LP |
|---|---|
| $\max_x c^T x$ s.t. $Ax \leq b$, $x \geq 0$ | $\min_y b^T y$ s.t. $A^T y \geq c$, $y \geq 0$ |
| $\min_x c^T x$ s.t. $Ax \geq b$, $x \geq 0$ | $\max_y b^T y$ s.t. $A^T y \leq c$, $y \geq 0$ |

Duality Theorems:
- [weak] objective value of any feasible solution to primal $\leq$ objective value of any feasible solution to dual i.e. $p^* \leq d^*$
  - Solutions of dual upperbounds the primal
  - Solutions of primal lowerbounds the dual
- [strong] If $p^* < \infty$, then $p^* = d^*$
  - If LP has bounded optimum $p^*$, so does its dual and $p^* = d^*$

Interpretation of y (dual variables):
- Lagrangian multiplier for constraints: weights the importance placed on each constraints
- Think in terms of price per x

| Primal LP | Dual LP |
|---|---|
| $\max_x c_1 x_1 + \cdots + c_n x_n$ | $\min_y b_1 y_1 + \cdots + b_n y_m$ |
| $a_{i1} x_1 + \cdots + a_{in} x_n \leq b_i$ for $i \in I$ | $a_{1j} y_1 + \cdots + a_{mj} y_m \geq c_j$ for $j \in N$ |
| $a_{i1} x_1 + \cdots + a_{in} x_n = b_i$ for $i \in E$ | $a_{ij} y_i + \cdots + a_{mj} y_m \geq c_j$ for $j \in N$ |
| $x_j \geq 0$ for $j \in N$ | $y_i \geq 0$ for $i \in I$ |

## Set Cover

Given $U = \{1, 2, 3, \ldots, n\}$ and subsets $S_1, \ldots, S_m \subseteq U$ s.t. $\bigcup S_i = U$, find collection of minimal cardinality $I = \{i\}$ s.t. $\bigcup_{i \in I} S_i = U$.

Greedy: Repeatedly pick $S_i$ that covers maximum # of uncovered elements.

Theorem: If optimal solution of MIS uses k sets, Greedy uses at most $k \ln n$ sets. Greedy is most efficient approximation.

Key Idea: let $n_t$ be number of elements not covered after t iterations of greedy. Suffices to show $n_{k \ln n} < 1$
(Claim: $n_{t+1} < n_t(1 - \frac{1}{k})$.

## Max Flow

Algorithm:
- Start with zero flow
- Repeat: choose viable path from s to t, increase flow along edges as much as possible; change weights of residue graph respectively.

$G^f = (V, E^f)$ is residue graph of $G = (V, E)$

$$f(x) = \begin{cases} c_{uv} - f_{uv} & (u,v) \in E,\ f_{uv} < c_{uv} \\ f_{uv} & (u,v) \in E,\ f_{uv} > 0 \end{cases}$$

Definitions:
- s-t cut: (L, R) s.t. $L \cup R = V$, $s \in L$, $t \in R$
- Capacity of (L, R): $\sum_{u \to v : u \in L, v \in R} c_{u \to v}$ = capacity(L, R)

Theorems and Facts:
- [Max Flow Min Cut] Size of maximum flow in a network = $\min_{\text{cut}}$
- For any flow f and any cut (L, R), $f \leq$ capacity(L, R)
- If all capacities $\in \mathbb{Z}$, maximum flow is integral
- No flow of value k from s-t $\Rightarrow$ min s-t cut has capacity $< k$
- Flow of value k from s to t $\Rightarrow$ min s-t cut has capacity $\geq k$.

Techniques
- Phantom source, sink, nodes, edges
- Bipartite matching: phantom nodes s, t. Find max flow from s to t. $\exists$ Matching $\Leftrightarrow$ can send n units of flow
- Hall's Marriage lemma.

## LP Algorithms (simplex, Ellipsoid, Interior point)

```
SIMPLEX(v): # returns optimal vertex
    while obj(v') > obj(v) and v' neighbour of v:
        return SIMPLEX(v')
    return v
```

- Two vertices $\in \mathbb{R}^n$ are adjacent/neighbors if they share n-1 inequalities in common
- Number of neighbors of v in LP with n variables m constraints $\leq (m-n) \cdot n$
- $O(\binom{m+n}{n} mn)$ (worst case exponential time)
- Problems with simplex and how to resolve them:
  - Starting vertex (can find by FEASIBLE(A, b) LP)
  - Degeneracy (introduce perturbation)
  - Unboundedness ($p^* \to \infty$; halt and complain)

Ellipsoid and Interior point method are polynomial time.

## Last reports

- DON'T GET STUCK! Act on the fly pace
- Think DP (Encoding change of states), Greedy, Dn C, Graphs
- Check LPs: don't be tricked by direction of inequalities. Decompose fully. Don't forget $x_i \geq 0$ constraints.
- Is the problem asking for primal or dual problem?
- Huffman coding: ordering of leaf nodes doesn't matter.

## Zero Sum Games

For game matrix $G = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, expected payoff is
$= \sum_{i,j} G_{ij} \, PP[R=i, C=j]$ (R plays i, C plays j)

### Theorems and Facts

- Once a player's strategy is fixed, there is a optimal pure strategy for the other player
- [von Neumann] For $A \in \mathbb{R}^{m \times n}$

$$\max_{x \in X} \min_{y \in Y} x^T A y = \min_{y \in Y} \max_{x \in X} x^T A y$$

$$\max_{x \in X} \min_{y \in Y} \sum_{i,j} G_{ij} x_i y_i = \min_{y \in Y} \max_{x \in X} \sum_{i,j} G_{ij} x_i y_j$$

### Linear Programs

- Game #1: Row announces her strategy $p = [p_1 \; p_2]$

$$\max_{p_1, p_2, p_1 + p_2 = 1, p_1 \geq 0, p_2 \geq 0} \min(a p_1 + c p_2, \; b p_1 + d p_2)$$

$$\max_{p \text{ mixed}} \min_{q \text{ mixed}} p^T A q = \max_{p \text{ pure}} \min_{q \text{ pure}} p^T A q$$ (second player's optimal strategy is a pure one)

- Game #2: Column announces her strategy $q = [q_1 \; q_2]$

$$\min_{q_1, q_2, q_1 + q_2 = 1, q_1 \geq 0, q_2 \geq 0} \max(a q_1 + b q_2, \; c q_1 + d q_2)$$

$$\min_{q \text{ mixed}} \max_{p \text{ mixed}} p^T A q = \min_{q \text{ pure}} \max_{p \text{ pure}} p^T A q$$

NOTE: In min max P(r|c) row goes second ⇒ has more info

## Appendix Part I

### All pairs Shortest Path (APSP) $O(n^3)$

Let $d[i,j,k]$ denote length of shortest $i \to j$ path where intermediate vertices are in $\{1, \ldots, k\}$

- $d[i,j,0] = w(i,j)$
- $d[i,j,k] = \min(d[i,j,k-1], \; d[i,k,k-1] + d[k,j,k-1])$

```
APSP(G):
  for (i,j) ∈ E: d[i,j,0] ← w(i,j)
  for (i,j) ∉ E: d[i,j,0] ← ∞
  for k = 1,...,n:
    for i = 1,...,n:
      for j = 1,...,n:
        d[i,j,k] ← min(d[i,j,k-1], d[i,k,k-1] + d[k,j,k-1])
  return d
```

### Shortest Path in DAG $O(|V|+|E|)$

Let $dp[i]$ denote length of shortest $S \to i$ path

$$dp[v] = \min_{u:(u,v) \in E} \{dp[u] + w(u,v)\}$$

```
SP_DAG(G,s):
  for dp[u] ← ∞
  dp[s] ← 0
  for v ∈ G in linearized order:
    dp[v] ← min_{u:(u,v)∈E} {dp[u] + w(u,v)}
return dp[E]
```

### HORN SAT

```
HORN-SAT(X):
  ∀i, x_i ← False
  while ∃ unsatisfied (x̄_i ∨ ... ∨ x_j):
    x_k ← True
  if every (x̄_i ∨ ... ∨ x_j) satisfied
    return (x_1, ..., x_n)
  else return "not satisfiable"
```

### Maximum Independent Set (Tree)

Find $S$ s.t. $u, v \in S, (u,v) \notin E$

Let $I(v)$ = size of largest independent set in subtree rooted at $v$.

$$I(v) = \max\left\{1 + \sum_{w \text{ grandchild}} I(w), \; \sum_{u \in \text{child}} I(u)\right\}$$

## Multiplicative Weight Updates

Problem: $n$ experts $E_1, \ldots, E_n$. On the $t^{th}$ day, expert $E_i$ incurs loss of $l_i^{(t)} \in \{0,1\}$. Pick $E_j$ and incur loss $l_j^{(t)}$. Want to minimize regret after $T$ days = Total loss − Total loss of best expert on hindsight.

At $t^{th}$ day, probability distribution $X^{(t)} = (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)})$

- Loss of algorithm on day $t = \sum_{i=1}^{n} x_i^{(t)} l_i^{(t)}$
- Total loss of algorithm over $T$ days $= \sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} l_i^{(t)}$
- $R_T = \sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} \cdot l_i^{(t)} - \min_{1 \leq i \leq n} \sum_{t=1}^{T} l_i^{(t)}$

$\therefore$ Exists algorithm s.t. $\lim_{T \to \infty} R = \lim_{T \to \infty} \frac{R_T}{T} = 0$.

### Key Ideas of Proof

- Fix $\varepsilon$; $\forall i$ set $w_i^{(0)} = 1$
- For each expert, assign weight $w_i^{(t)}$ (weight of expert $i$ on day $t$)
- On day $t$, $P(\text{pick } i) = x_i^{(t)} = \frac{w_i^{(t)}}{\sum w_i^{(t)}}$
- At end of day $t$, update weights via $w_i^{(t+1)} = w_i^{(t)} (1-\varepsilon)^{l_i^{(t)}}$

### Results:

- After $T$ days, regret $R_T \leq \varepsilon T + \frac{\ln n}{\varepsilon}$. Pick $\varepsilon = \sqrt{\frac{\ln n}{T}}$
- Average regret $R \in O\left(\sqrt{\frac{\ln n}{T}}\right)$

### Variants and Corollaries:

- Online algorithm
- Constant # of variables per expert (but $n$ lowers memory)
- # of time steps after runtime of algorithm
- $\varepsilon$ affects calculations of weights

If loses $\in [a,b]$, $R_T \leq (b-a)\left(\varepsilon T + \frac{\ln n}{\varepsilon}\right)$

## Appendix II

### Shortest path problem $O(k|V| + k|E|)$

Given directed $G = (V, E)$, positive or negative edge weights, $s, t \in V$, $k \in \mathbb{Z}$, output shortest path using at most $k$ edges.

$$dp(v,i) = \min\left\{\min_{u:(u,v) \in E} \{dp(u, i-1) + w(u,v)\}, \; dp(v, i-1)\right\}$$

```
SHORTEST-K-PATH(G, s, t, k):
  dist[v,0] ← ∞; dist[s,0] ← 0
  for i = 1,...,k:
    for v ∈ V:
      dist[v,i] ← min{min_{u:(u,v)∈E} {dist(u,i-1) + w(u,v)}, dist(v,i-1)}
  return dist[t,k]
```

Remark: For shortest path, return $dist[E, n-1]$

### Edit Distance $O(|s||t|)$

Edits allowed: insert one character, delete or substitute

$E[i,j]$ = edit distance of $s[1,\ldots,i]$, $t[1,\ldots,j]$

$$E[i,j] = \min\{1 + E[i-1,j], \; 1 + E[i,j-1], \; E[i-1,j-1] + diff(i,j)\}$$

```
EDIT_DISTANCE(s,t):
  for i = 0,...,m: E[i,0] ← i
  for j = 0,...,n: E[0,j] ← j
  for i = 1,...,m:
    for j = 1,...,n:
      E[i,j] ← min{E[i-1,j]+1, E[i,j-1]+1, E[i-1,j-1]+diff(i,j)}
return E[m,n].
```