CS3230
Design and Analysis of Algorithms
github.com/securespider

01. Stable matching

Both sides rank each other and goal is to pair each up **Constraints** No **rogue couples** - matched partner likes someone more than current partner that also likes them back

Gale-Shapley Algo

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

Invariant

- If woman not on boy list, she has a better current fav
- Boy choice is strictly worsening
- Girl choice cannot worsen (weakly increasing)

02. Algorithm Analysis

Algorithm Finite sequence of well-defined instruction to solve computational problem

- Optimize running time
- Runtime is machine and input dependent

Word-RAM model

- Word is basic unit of memory (few bytes)
- Runtime $\approx$ number of instructions taken
- Ram can be accessed in the same time irrespective of location
- Every operation involves constant number of words and cycles by CPU

Problem vs Algorithm

- Problem does not specify approach but algorithm defines ways/methods to solve prob

Polynomial time

- There exist constants $c, d \in \mathbb{R}$ on every input of size N such that its running time is bounded by cN^d
- Brute Force - Checks all possible solution and typically takes exponential time 2^N
- Generally efficient because c and d are low

Worst-Case Analysis

- Bound on largest possible running of algorithm on input
- Captures efficiency in practice

vs Average case running time

- Hard to accurately model real instances by random distributions
- Tuned for certain distributions

Asymptotic Order of Growth

- Compare for asymptotically large values of input sizes (rate of growth)
- Suppress constant factor and lower-order terms

Bounds

Upper Bound T(n) = O(f(n)): Exist constants $c, n_o \geq 0$ such that $\forall n \geq n_0$, we have $T(n) \leq cf(n)$

Lower Bound T(n) = ω(f(n)): Exist constants $c, n_o \geq 0$ such that $\forall n \geq n_0$, we have $T(n) \geq cf(n)$

Tight Bound T(n) = θ(f(n)): T(n) is both O(f(n)) and ω(f(n))

Properties

Transitivity If f = O(g) and g = O(h) then f = O(h)

Additivity If f = O(h) and g = O(h) then f + g = O(h)

Bounds for some functions

- Polynomials - Just take the highest power
- Logarithms - Base doesn't matter and always smaller than polynomial
- Exponential - Always grows faster than polynomial

Common runtimes

- O(n) - List traversal or merging lists
- nlogn - sorting, intervals, divide and conquer algos (recursive)
- n^2 - Finding pairs of elements
- n^k - Enumerate subsets of k nodes, set disjoints
- Exponential - All subsets

03. Graph

- Describes relationships between nodes/entities

Representation

Adjacency Matrix NxN matrix where $A_uv =$¡weight¿ if (u,v) is an edge

- Good for big graphs/a lot of edges(dense)
- O(1) time for checking edge
- Space is O(n^2)
- Identifying all edges take O(n^2) time

Adjacency list List of linked lists representing nodes that are attached to each node

- Good for sparse graphs/trees
- O(deg(u)) for checking edge
- Space is V+E
- Identifying all edges takes O(V+E) time

Path and Connectivity

- **Path** - Sequence P$\in \{v_0, v..v_k\}$ of nodes such that each adjacent pair is joined by an edge

  Simple All nodes are distinct

  Cycle Path such that $v_0 = v_k, k > 2$ and first k-1 nodes are distinct

- **Connected** - There is a path for all pairs of nodes
- **Tree** - Undirected connected graph that does not contain a cycle
  - Any 2 implies Tree
  - G is connected
  - G does not contain cycle
  - G has n-1 edges
- **Rooted tree** - Arrangement of tree such that root is top and nodes attached to parent is in different level

Connectivity problems

s-t connectivity Is there path between any 2 points s, t

s-t shortest path prob Length of shortest path btw s,t

Breadth First Search

Algorithm

- Explore outward from some node, s in all directions adding nodes one layer at a time
- Algorithm
  1. Layer$_0$=s
  2. L_1=neighbours of L_0
  3. L_n = neighbours of L_{n-1} not in all prev layers

Properties

- Induction theorem: $\forall i, L_i$ consist of nodes at dist(s) = i
- Let T be a BFS tree of $G = (V, E)$, and let (x, y) be an edge of G. Then the level of x and y differ by at most 1.
- BFS runs in O(V+# E) given adjacency list

Bipartite graph

- Independent sets such that no nodes within the set are connected to each other but are connected to nodes of other sets
- Colour each node red or blue and every edge has strictly one red and blue end each
- Bipartite graphs cannot contain odd length cycles

Algo for Bipartiteness

- Let G be a connected graph and $L_0..L_k$ are layers produced by BFS
  1. Edge between 2 nodes in same layer
     - Graph is bipartite as we can colour alternate layers as red and blue
  2. No edge btw nodes in same layer
     - Graph cannot be bipartite as there is odd length cycle
     - Let x,y be the 2 points in same level and has edge btw them
     - Cycle: s-x-y-s (s-x == y-s) Odd length

Directed graph

- Asymmetric edges: (u,v) (edge from u to v) does not imply (v,u)
- Directed reachability problem: Find all nodes reachable from node s

Strong Connectivity

Mutually reachable Path from u to v and vice versa

Strongly connected Every pair of nodes are mutually reachable

- Strongly connected iff every node reachable from s and s reachable from every node

Algorithm Pick any node and run bfs on G and G^{rev} (reverse orientation of every edge in G)

- G is strongly connected if all nodes reached in both BFS executions
- Runs in O(m+n) time

Directed Acyclic Graph

- Directed graph that has no directed cycle
- **Precedence constraints** - Edge (v_i, v_j) means v_i must preceed v_j
- **Topological order** - For every edge (v_i, v_j) we have $i < j$
  - If G has a topological order, then G is a DAG (Proof by contra)
  - If G is a DAG then G has a node without incoming edges (Proof contra)

04. Greedy algorithms

- Consider jobs in some order and immediately take the job that is compatible with the previous jobs
- Has some heuristic to sort the jobs

Scheduling intervals

- Maximize number of jobs run
- Heuristic: end time, start time, interval size, fewest conflict

Interval Partitioning

- Minimize number of classrooms to run lectures concurrently
- Heuristic: start time
- Assign lecture to first compatible classroom

Minimize lateness scheduling

- Heuristic: Earliest deadline, Shortest processing time first, Smallest slack ($d_j - t_j$)
- Start with an optimal solution and inductively reduce number of inversions until it reaches the greedy solution

Inversions

- Pair of jobs i and j where dl(i)¡dl(j) but j schedule before i
- Greedy solution does not have any inversions

Greedy analysis Strategies

1. Each step in greedy solution is at least as good as other solutions
2. Exchange argument: Transforming solution to greedy algorithm without hurting its quality
3. Structural: Every solution has a certain value and greedy algos can be in this bound

Optimal offline caching

- Eviction strategy that minimises number of cache misses
- Note that the full sequence of requests is known a priori
  - vs Online: Request not known in advance
- Heuristic: Farthest in future
  - Evict item in cache that is not requested until farthest in the future