CSE 421 Final Notesheet

Overview

- Complexity of algorithm: $T(N)$, which N is problelm input size.
- Polynomial Time: $T(N) \leq cN^k + d$.

Stable Matching

- Perfect Matching: everyone is matched monogamously.
- Stability: an unmatched pair is unstable if they prefer each other than its partner.
- Gale-Shapley Algorithm:
  Runtime: $O(n^2)$, man-optimal

```
while some man is free and did not propose to every woman:
        Choose such man m
        w = m prefers most and did not propsed to yet
        if w is free:
            assign w to m
        else if w refers m to her current partner:
            assign m and w to be engaged
```

Graph Traversal

- States of Node:
  - unvisited
  - visited/discovered
  - fully explored (itself and all neighbors are visited)
- Breadth First Search:
  - all non-tree edges join vertices on same or adjacent layers
  - BFS tree is mininum depth, which gives shortest paths

```
mark all node "unvisited"
    R <= {S}
    Layer L0 <- {S}

    while Li not empty:
        Li+1 = {}
        for each u in Li:
```

```
        for each neighbor v of u:
            if v is "unvisited":
                mark v "visited"
                add v to set R and to Li+1
        mark u "fully explored"
    i = i + 1
```

- Depth First Search:
    - No cross edges (edges join two subtrees)

```
DFS(u):
        mark u "visited" and add u to R
        for each edge {u, v}:
            if v is "unvisited":
                DFS(v)
        mark u "fully explored"
```

- Directed Acyclic Graphs (DAG): a directed, acyclic (contains no cycles) graph.
    - Every DAG has a vertex with in-degree 0.
- Topological Sort:
    - numbering of the vertices of G with distinct numbers from 1 to n so edges only go from lower number to higher numbered vertices.
    - Can be done with DFS
    - Total cost: $O(m+n)$

```
i = 1
    while There are nodes in G:
        mark any vertex with in-degree 0 i
        remove that vertex from G
        i = i + 1
```

Greedy Algorithm

- Interval Scheduling
    - Choose request with earliest finish time
- Interval Partitioning
    - Sort requests in increasing order of starting times
    - If compatable with last scheduled request, add to the same schedule
    - If not, schedule on new resourc

- Minimizing Lateness:
  - Earliest Deadline First: Greedy Schedule has no inversions
  - Swapping inverted jobs do not increase max lateness.
- Dijkstra's Algorithm:
  - Maintaining current best lengths of paths that only go through S to each of the vertices in G.
- Mininum Spanning Trees:
  - subgraph of mininum total weight which every pair of vertices connected in G are also connected in T.
  - Prim's Algorithm: choose cheapest edge from current tree to rest of the graph.
  - Kruscal's Algorithm: choose cheapest edge connecting two pieces of the graph that aren't yet connected.
    - For every cur of a graph, there is a mininum spanning tree connecting any cheapest edge crossing the cut.
    - Union-find disjoint sets

Divide and Conquer

- Reduce problem to sub-problems with same type, with size being a constant fraction of original problem.
- Closest pair in the plane: $O(nlog(n))$.
- Master Theorem: if $T(n) \leq a \cdot T(n/b) + c \cdot n^k$ for $n > b$, then:
  - if $a > b^k$, then $T(n) = \Theta(n^{log_b a})$.
  - if $a < b^k$, then $T(n) = \Theta(n^k)$.
  - if $a = b^k$, then $T(n) = \Theta(n^k log(n))$.
- Karatsuba's algorithm:
  - $T(n) = 3(T(n/2)) + cn$
  - $T(n) = O(n^{log_2 3})$

```
PolyMul(P, Q):
        # P, Q are length 2n vectors
        # Let P[i] be the coefficient of x^i in polynomial P
        # Let P0 be elements 0...k-1 of P
        # P1 be elements k...n-1
        if n == 1: return P[0]*Q[0]
        else:
            A = PolyMul(P0, Q0)
            B = PolyMul(P1, Q1)
            PSum = P0 + P1
            QSum = Q0 + Q1
            C = PolyMul(PSum, QSum)
            Mid = C - A - B
            R = A + Shift(Mid, n/2) + Shift(B, n)
```

```
        return R
```

- Interpolation & Evaluation:
  - evaluate polynomial at $1\omega^{-1}, \omega^{-2}, ..., \omega^{-(n-1)}$
  - Divide each answer by n to get $c_0...c_{n-1}$
  - $\omega = e^{2\pi i/n}$
  - $O(nlog(n))$ for both interpolation and evaluation.

Dynamic Programming:

- Useful when the sub-problems show up again and again in the solution.
- Memorization (Caching): Remember values from previous recursive calls, and use previous results if available.
- Examples:
  - Weighted Interval Scheduling
  - Segmented Least Squares
  - Knapsack Problem:
    - K(n, W) = K(n - 1, W) or K(n - 1, W - xn) + xn
  - RNA Secondary Structure:
    - $$OPT[1...j] = MAX(OPT(1...j-1), 1 + MAX_{k=1...j-5}(OPT(1...k-1) + OPT(k+1...j-1)))$$
  - Bellman-Ford
    - Shortest path with negative cost edges
    - Let Cost(s, t, i) be cost of minimum-length path from s to t using up to i hops
    - $Cost(v, t, i) = min(Cost(v, t, i-1), min_{(v,w)\in E}(c_{vw} + Cost(w, t, i-1)))$
    - O(m + n) time

Network Flow

- Ford-Fulkerson Method:
  - while residual graph has an argument path: argument
  - total O(mC) time, which $C = \sum_{(s,u)\in E} c(S, U)$
  - Partition of vertices: an s-t cut if source and sink on different pieces
  - It will terminate if c(e) is integer or rational.
- Max Flow/Min Cut Theorem: For any flow f, if Gf has no argumenting path. then there is some s-t cut (A, B), such that v(f) = c(A, B).
- Capacity Scaling: scale bits by bits

- Total time: $O(m^2 logU)$, which U is largest capacity,
- Edmonds-Karp Algorithm:
  - Use a shortest argumenting path (BFS in residual graph)
  - Total time: $O(nm^2)$

NP-Completeness

- Reduction: maps one problem to another with a black-box solution to the other problem.
- P: problems can be solved in polynomial time
- NP: a desicion problem is NP iff there is a polynomial time procedure to verify an instance of the problem itself.
- NP-hard: as hard as any problem in NP
- NP-Complete: iff both NP and NP-hard
- Cook-Levin: 3-SAT is NP-Complete
- Problems in NP-Complete:
  - 3-SAT
  - Independent Set
  - Clique
  - Vertex-Cover
  - Set-Cover: Smallest collection of subsets whose union equals the universe.
  - Hamiltonian-Cycle: cycle in graph which visits each vertex exactly once
  - Hamiltonian-Path: path in graph which visits each vertex exactly once
- Procedure to show a problem is NP-complete:
  - Show it is NP-hard:
    - Reduction is from NP-hard problem A
    - show the map f.
    - Argue that f is polynomial time
    - Argue corrections: two directions
  - Show it is NP:
    - State the certificate and why it works
    - show it is polynomial time to check