CS6515 Exam 2 Notes

General Graph Algorithms

Representation

- Adjacency List
  - Array of vertices that are pointers to linked lists of adjacent vertices
  - Each element represents an edge to another vertex
  - Operating on 1 vertex: O(1)
  - Operating on 1 edge: O(|E|)
  - Removing edges/producing subgraph: O(|V|+|E|)

BFS

- Priority queue based
- O(|V|+|E|)
- Access to: prev(u) giving vertex preceding u in shortest path from v
- Outputs: dist(u) set to shortest path between v and reachable vertex u, or infinity of not reachable
- Types of graphs: Unweighted/undirected or directed, with uniform edge length

DFS

- Stack based
- O(|V|+|E|)
- Previsit and postvisit Numbers:
  - Pre is first visiting, post is removing from the stack
  - End of branches have consecutive numbers
  - Ordering post numbers is a topological ordering for DAGs
- Edge types:
  - Tree edge: taken during DFS
  - Back edge: to ancestor
  - Forward edge: to non child descendent node
  - Cross edge: to non ancestor and non child and fully explored
- Longest path in DAGs can be found by updating distance of successors
- Finding a source vertex is O(|V|) and check in-degree = 0
- for v ∊ V, find all v with the same ccnum as s. These are reachable by s
- outputs connected components, topological sort on a DAG. You also have access to the pre and post arrays
  - Access to: pre[], post[], prev[], and visited[T/F] arrays shared between explore & DFS
- Types of graphs: Unweighted/Undirected graphs, directed graphs, in particular - Directional Acyclic Graph (DAG)
- Outputs:

- Undirect G = Vertices labelled by connected component number (ccnum)
- Directed G = list of subgraphs (1 for each subcomponent)

SCC

- Do DFS on G_r, reversed edge graph of G
- Then from highest post number (the sink vertex of G), find connected components and remove them
- Runtime: 2 DFS
- Metagraph of a directed graph is always a DAG
- Access to: strongly connected components via ccnum(u) of first DFS run, and all other DFS outputs/structures
- Outputs: metagraph that has to be a DAG (contains connected components from 2nd DFS run)
- Vertex with highest post number must lie in a source SCC

Explore Subroutine

- O(|V|+|E|)
- Find all nodes that can be visited from the starting node, returns an array that sets visit(u) to true for any visited node
- Run explore on the reverse graph to find sets of all nodes that can reach a node, v
- Access to:
    - previsited (prev[]) = arrays of vertices before a given vertices (but not used by Explore, needed for DFS)
    - ccnum[]

Dijkstra's

- Types of graphs: Weighted/undirected or directed graphs (no negative weights)
- BFS with priority queue
- O(|E|*Tdk +|V|*Tem) or O((|V|+|E|)* log |V|)
- Perform on a connected reverse graph, will get the distance to get to v
- Produces the shortest path tree rooted at v
    - Shortest path tree is a spanning tree of G such that the path distance from root v to any other vertex u in T is the shortest
- Nonnegative edges only
- Queue order take the least tentative distance from s
- Access to: prev(u) giving vertex preceding u in shortest path from v

Bellman-Ford

- Types of graphs: Weighted/directed or undirected (can have negative weights)
- Dijkstra but updates all edges V-1 times
- O(|V| * |E|)
- Detects negative cycles if no terminal point

- Update((u,v) in E):
  - dist(v) = min { dist (v), dist(u) + l(u,v)}

Floyd-Warshall
- Types of graphs: Weighted/directed or undirected (can have negative weights)
- Recursively finds the shortest path through k between I and j
- Set diagonal to infinity to detect cycles
- Access to: detect negative cycles by checking diagonals T[n,i,j]
- Recurrence: min {shortestPath(i,j,k-1), shortestPath(I,k, k-1)+shortestPath(k,j,k-1)}
  - where shortestPath(I,j,k) returns the shortest path from I to j using up to k as intermediate points
- O(|V|^3)

2-SAT
- CNF is a conjunction (AND) of disjunctions (OR)
- 2-CNF: each clause has 2 terms
- Graph reduction
  - X1, x1 bar, x2, x2 bar, … etc vertices
  - Clause A or B -> a bar to B edge and B bar to A edge
- If both x and bar x lie in the same SCC, the CNF is unsatisfiable
- Source SCC set to false, or sink SCC to true. Source is the complement of the sink
- Algorithm:
  - Construct G given CNF
  - Find the sink SCC
  - Set sink SCC to true and source to F
  - Remove S and S bar
  - Repeat until empty graph
- Runtime: O(|V|+|E|)

Minimum Spanning Tree

Kruskal
- Types of graphs: connected, undirected, weighted graphs
- Algorithm:
  - Edges are considered in order of least weight to most
  - Check if each end is in a different disjoint set with find
  - If yes, combine with union and add edge to MST
- Rank: height of the subtree hanging from the root node
- Find returns the root node of component that the vertex is in
- Union combines two components, attach the shorter one to the root of the longer one
  - If two equal rank, then increases rank by one

- Path compression: links every node in the path to the root, happens during find
- Works with negative edges
- Runtime: O(|E| * log |V|), dominated by the sort
- Data structure: disjoint-set

Prim's
- Algorithm:
  - From starting vertex, find lightest edge to the rest of the graph, forming S.
  - At each iteration, find lightest edge from S to X -S, add to S
- Works with negative edges
- Same runtime as Dijkstra's, O((|V|+|E|)* log |V|) or O(|E| * log |V|)
- Data structure: Binary heap
- Outputs: A minimum spanning tree defined by the array prev[]

MST Properties
- Cut is a partition of the vertices into two disjoint subsets
- Cut-set of a cut is the set of edges that have one endpoint in each subset of the partition
- Cut property: *For any cut C of the graph, if the weight of an edge e in the cut-set of C is strictly smaller than the weights of all other edges of the cut-set of C, then this edge belongs to all MSTs of the graph. if more than one edge is of minimum weight across a cut, then each such edge is contained in some minimum spanning tree.*
- Cycle property: *For any cycle C in the graph, if the weight of an edge e of C is larger than any of the individual weights of all other edges of C, then this edge cannot belong to an MST*
- Unique if there is no cycle or one cycle or if every edge has a different weight

Flow Networks

Invariants
- **Capacity Constraint:** An arc's flow cannot exceed its capacity; f(u,v) <= c(u,v)
- **Flow Conservation Constraint**: The total net flow entering a node *v* is zero for all nodes in the network except the source and the sink, s and t.
- **Non-deficient flows**: The net flow *entering* the node *v* is non-negative, except for the source, which "produces" flow; x_f(v) > 0 for all v in {V - s}
- **Skew symmetry constraint:** The flow on an arc from *u* to *v* is equivalent to the negation of the flow on the arc from *v* to *u*, that is: $f(u, v) = -f(v, u)$

- **Values:** The flow leaving from *s* must be equal to the flow arriving at *t*
- *F is the collection of flows on G*
- *f_e is the flow across a single edge*
- *size(f) is the sum of flow leaving s or entering t*
- *F* is the max flow*
- *Flow correctness needs to check if flow is valid*
  - *Check capacity and flow conservation constraint*

Residual Graph
- Take flow, subtract from capacity, put into original edge
- Create new reverse edge with flow
- Min-cut is the strongly connected component of the final residual graph starting from the source

Ford-Fulkerson
- Let f(u,v) <- 0 for all edges in G
- Define residual graph, G_f
- While there is an augment path *p* from *s* to *t* in G_f, such that c_f(u,v) > 0 for all edges (u,v) in p:
  - Find c_f(p) = min {c_f(u, v}: (u,v) in p}
  - For each edge (u,v) in p
    - f(u,v) <- f(u,v)+c_f(p)
    - f(v,u) <- f(v,u)-c_f(p)
- When capacities are integers, runtime is O(|E|*f), f is the maximum flow in the graph

Edmonds-Karp
- Same as Ford-Fulkerson but used BFS to find the augmenting path
- Runtime is O(|V|*|E|^2)

Feasible Flow
- Constructing G′ from G
  - Original edges are grounded to difference between demand and cap
  - S′ edge weights are the demand into vertices
  - T′ edge weights are the demand out of vertices
  - Infinite edge from t to s
- Saturating flow is when the edges out of S′ and into T′ are fully capacitated
- G has feasible flow if G′ has saturating flow

Number Theory

Extended Euclidean Algorithm
- Bezout's Identity: s * a + t * b = gcd (a, b)

- Table: a(mod base) b q r s1 s2 s3 t1 t2 t3
- First row: s1 = 1, s2 = 0, t1= 0, t2= 1
- S3 = s1 - q * s2 , t3 = t1 - q* t2
- Next row = b -> a, r-> b,s2-> s1, s3->s2, t2-> t1, t3-> t2

RSA Cryptography

- Choose primes p and q, let N = pq
- Find e where gcd(e, (p-1)(q-1)) = 1
- Let d = e^-1 mod (p-1)(q-1)
- Public key = (N, e)
- Encryption: y = m^e mod N
- Decrypt y: m = y^d mod N

Other Number Theory Formulas

- Fermat's Little Theorem:
  - a^p = a (mod p) when p is prime
- Euler's Formula:
  - For distinct primes, p and q, and any a /= 0 (mod pq), a^(p-1)(q-1) = 1 (mod pq)
- Multiplicative inverse exists if the modulus and number are relatively prime, gcd = 1

Graph Properties and Other Formulas

- Max edges = n * (n-1) / 2 for undirected, n*(n-1) for directed
- Adding an edge to a spanning tree creates a cycle
- Serial exponentiation: a^b^c = a^(b^c)
- When the graph is connected, |E| >= |V-1|, so O(|V|+|E|) simplifies to O(|E|)