CSCC73 – Algorithm Design and Analysis

Fall 2018 Finals Notes – Anna Bretscher

Dijkstra's Algorithm O(E + V log V)
Greedy, uses a vertex set
Finds the shortest path from a source node (s) to every other node (t)

Prim's Algorithm O(E · log V)
Greedy, uses a min heap
Finds a Minimum Spanning Tree
- graph must be connected, otherwise only 1 of several MSTs are found
- better for dense graphs due to nature of fibbonaci heap

Kruskal's Algorithm O(E · log V)
Greedy, uses disjoint sets
Finds a Minimum Spanning Tree
- if graph is non-connected, finds minimum spanning forest
- better for sparse graphs

Tree theorem: Need to prove that an MST is a tree
Cut Theorem: Need to prove that greedily choosing the smallest edge at each iteration of the algorithm, will result in an MST.

Huffman Encoding
Greedy, tree data structure to store encoding
Finds the optimal smallest bit representation of strings by sorting by ascending usage rate and constructing intermediate trees

Bellman-Ford's algorithm O(VE)
Dynamic Programming, uses vertex set
Finds the shortest path from a single node (s) to every other node (t)
- Can detect negative cycles in graph
- Can deal with negative weight edges and negative cycles

Floyd-Warshall's Algorithm O(V^3)
Dynamic Programming, uses vertex set
Finds all-pairs shortest path in a graph
- shortest path length for all u, v in graph
- doesn't store the details of the path length
- can deal with negative weights
- cannot deal with negative cycles

Ford-Fulkerson's Algorithm O(Ef)
Greedy, done by running augmenting paths on residual graph
Finds the maximum network flow in a graph

Min-cut is max flow – the minimum cut of a graph is equivalent to its max flow

Hall's Theorem
Hall's Marriage Theorem states for any G bipartite graph containing the partition V1, V2 st. |V1| == |V2| there is a perfect pairing iff
Neighbourhood(any subset X) > |any subset X|

That is, any subset of the smaller bipartite graph has a neighbourhood (edges to other nodes) larger than the number of elements

Johnson's Algorithm
Finds all-pairs shortest paths
Add new vertex s, add edges from G to s, becoming G'. Run Bellman-Ford on G' using s as source finding a positive weighted edge for every pair. Remove added vertex and run Dijkstra's on every vertex
- Needs fibbonaci heap to run better than F-W
- Worse on dense graphs
- Can deal with negative weight edges and negative cycles

Simplex Method/Tableau
1. Set up slack variables s1, s2.. sn
2. Transform into standard form, z - nx1 - mx2 ... = 0
3. Set up matrix such that
4. Use row-reduction to solve, when all x1..xn are >= 0, then we have our answer

$$\begin{array}{c|c|cccccc|c} & z & x_1 & x_2 & s_1 & s_2 & s_3 & b \\ \hline z & 1 & 0 & 0 & 1.6 & 2.2 & 0 & 544 \\ \hline s_1 & 0 & 1 & 1 & 0.4 & -0.2 & 0 & 16 \\ s_2 & 0 & 0 & 1 & -0.2 & 0.6 & 0 & 72 \\ s_3 & 0 & 0 & 0 & -0.4 & 0.2 & 1 & 19 \end{array}$$

Dual LP
Create coefficients and use them to find a new LP
Solve that LP to find the constraints on the primal LP
Related that Primal is Maximization, Dual is Minimization
Is extended from same proof that Max Flow = Min Cut

Algorithms and Applications

Greedy Algorithms

Greedy Stays Ahead and Exchange Argument – usually O(n log n) due to some well-ordered sorting that exposes optimal sol'n

Scheduling Problem – O(n log n)
- **Q:** Schedule max amount of jobs to be completed
- **A:** Sort the jobs by earliest end time and take greedily take each end time that doesn't conflict – exchange argument

Interval Partitioning – O(n log n)
- **Q:** Schedule min amount of rooms to accommodate all meetings
- **A:** Sort lecs by start time, if can be scheduled do it, otherwise open new room – greedy stays ahead

Shortest Paths in a Graph – O(E + V log V)
- **Q:** Find the shortest path between two nodes u, v
- **A:** Run Dijkstra's algorithm, greedily taking the shortest edge not in the solution at each step – greedy stays ahead

Minimum Spanning Tree – O(E · log V)
- **Q:** Find a tree which visits every node but has the least total path weight
- **A:** Use Prims or Kruskal's algorithms, greedily taking by the cut theorem the upcoming edge – exchange argument

Huffman Encoding
- **Q:** Finding a minimal bit representation of any string – the optimal prefix codes
- **A:** Using Huffman encoding, we greedily take lowest priority characters and construct a tree from them – exchange argument

Knapsack Problem – Fractional is greedy, simply choose the best (ratio) item continuously, then continue to next item, etc.

Divide and Conquer

Divide into subproblems and recombine into answer – using recurrence relations and bounding runtime Master Theorem

Counting Inversions – divide into two until base case, merge step counts from 1..n on both arrays, counting inversions at each step
Closest Pairs – divide into two parts until base case, find smallest distance d_1, create zone of d_1 around division, find smallest d_2 in that zone, the smallest pair in this subproblem is $\min(d_1, d_2)$
Integer Multiplication – convert to binary, into n/2 high and n/2 low bits, becomes a shift – recursive multiply theorem
Matrix Multiplication – divide a/b into n/2 blocks, conquer by multiplying the 8 n/2 blocks recursively, combine by adding **Strassen's Al.**

Dynamic Programming

Divide into optimality of previously solved (memoized) subproblems – proven using star (description), dagger (recurrence)

Weighted Interval Scheduling – scheduled jobs have weights, we take **max(vj + M[pj], M[j − 1])**
Knapsack Problem 0-1 – thief must choose to take or leave the item, having recurrence of **max(Opt(i − 1,w), vi + Opt(i − 1,w − wi))**
Sequence Alignment– define gap penalty, can model as a DAG and use **Hirschberg's Algorithm**
All-Pairs / Shortest Path –
- Floyd Warshall's Algorithm – distance array, SP between nodes,
- Bellman-Ford's Algorithm – finds the optimal edges by selecting shorter paths and working upwards

Optimal BSTs
Longest Common Subsequence - Given two strings X,Y, find the longest common subsequence of characters **O(N M)**
Longest Increasing Subsequence - Given an unsorted array A, find the longest increasing subsequence of characters **$O(N^2)$**
Maximum Independent Set - Find set in a path G, with largest weight and no adjacent nodes **O(N)**

Network Flow

Finding the maximum flow of a network through setting up network, applying concepts like Residual Graph, Augmented Paths

Bipartite Graph Matching – create source to all V_1, sink to all V_2 with capacities of 1, find if there exists a max flow equal to $|V_1|$
Disjoint Paths – create source to all V_1 and sink to all V_2 all edges w/cap 1, then the number of disjoint paths is max flow to sink
Network Connectivity – disconnect if set of edges are used in all s,t paths
- **Menger's Theorem -** The max number of edge-disjoint s, t-paths is equal to the minimum number of edges whose removal disconnects t from s

Circulation Problem - we create S to all suppliers with pos capacity equal to their demand, and sink T to all retailers' w/edges equal

Linear Programming

Optimal answer to objective function subject to some linear constraints. Know Simplex Method, Tableau, and concept of Duality

Max Flow can be a LP problem
Vertex Cover/MIS can be a LP problem
Shortest paths can be a LP problem

Slack Variables are used to convert from <= and >= to ==