Here is a list of topics for the final.

1. Binary search trees (INSERT, DELETE, SEARCH, MIN, MAX, PRED, SUCC)
   - binary search tree property
   - tree walks

2. Red-black trees
   - red-black tree invariants

3. Augmented search trees
   - augment every node $x$ with some additional information $f(x)$; maintain $f(x)$ under INSERT and DELETE in the same asymptotic bounds.
   - Sufficient if $f(x)$ can be computed using only $f(left(x))$ and $f(right(x))$.
   - implement SELECT(i), RANK(x) by augmenting every node $x$ with size($x$)
   - Interval tree

4. Dynamic programming
   - optimal substructure, overlapping subproblems
   - recursive formulation
   - running time without storing table
   - running time with dynamic programming (storing table)
   - examples: rod cutting, matrix chain multiplication, subset sum, weighted subset sum (0-1 knapsack),

5. Greedy algorithms
   - Correctness proof
   - examples: interval scheduling, fractional knapsack, Dijkstra

6. Graph algorithms
   - Basic definitions, graph representation (adjacency list, adjacency matrix)
   - Graph traversal: BFS
     - $G$ directed or undirected
– find connected components (G undirected), check bipartiteness (G undirected), compute shortest paths (G un-weighted, all edges have weight 1)

• Graph traversal: DFS
  – G directed or undirected
  – find cycles (back edges), topological sort (G directed, acyclic)

• Shortest paths: SP
  – Dijkstra’s SSSP algorithm
    * G directed or undirected, weighted, non-negative weights
  – SP on DAGs
  – SSSP on graphs with negative edge weights

Practice problems

1. Short answers:
   (a) Given a binary search tree with n elements, its height can be as small as \( \Omega( \quad ) \) and as large as \( O( \quad ) \).

   (b) Is it true that the height of a red-black tree with n elements is \( \Theta(\log n) \)?

   (c) How long does it take, in the worst case, to build a red-black tree of n elements?

   (d) Is it true that in the worst case, a red-black tree insertion requires \( O(1) \) rotations?

   (e) Is it true that walking a red-black tree with n nodes in in-order takes \( \Theta(n \log n) \) time?

   (f) You have a binary search tree \( T \) with n elements. Is it true that the predecessor of an element in \( T \) can be found in \( O(1) \) time?

   (g) As a function of \(|V|\), what is the minimum number of edges in a connected undirected graph with \(|V|\) vertices?

   (h) How many edges are in a complete graph with \(|V|\) vertices?

   (i) You have an undirected, connected, weighted graph \( G \) and a source vertex \( s \). Is it true that BFS computes shortest paths from \( s \) to every other vertex?

   (j) Is it true that Dijkstra’s SSSP algorithm works only on graphs with non-negative edge weights?
(k) Let $p = u \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_k \rightarrow v$ be the shortest path from $u$ to $v$ in a graph $G$. Let $v_i$ and $v_j$ be two vertices on $p$ such that $1 \leq i < j \leq k$. Is it true that the subpath $v_i \rightarrow v_{i+1} \ldots \rightarrow v_j$ in $p$ is the shortest path in $G$ from $v_i$ to $v_j$?

(l) Can a shortest path contain cycles?

(m) Is it true that running Dijkstra’s algorithm on a graph $G = (V, E)$ takes $O(|E| \cdot \log |V|)$?

(n) You have a complete graph with $|V|$ vertices. How long does it take to run Dijkstra’s algorithm on this graph?

(o) How fast can you compute shortest paths between two arbitrary nodes in a graph with negative edge weights?

(p) How fast can one identify negative cycles in a graph?

2. In this problem we consider a data structure $D$ for maintaining a set of integers under the normal INSERT, DELETE, and FIND operations, as well as a COUNT operation, defined as follows:

- $\text{INSERT}(D, x)$: Insert $x$ in $D$.
- $\text{DELETE}(D, x)$: Delete $x$ from $D$.
- $\text{FIND}(D, x)$: Return pointer to $x$ in $D$.
- $\text{COUNT}(D, x)$: Return number of elements larger than $x$ in $D$.

Describe how to modify a standard red-black tree in order to implement $D$ such that INSERT, DELETE, FIND, and COUNT are supported in $O(\log n)$ time.

3. With the new inside acquired from your algorithms class, you decided to research the relation between people’s birthday and IQ. For this purpose you maintain a database of the birthdays (just the day, not the date) and IQs of all your friends. You would like to be quickly able to answer questions of the form:

- $\text{INSERT}(b, iq)$ inserts a person with birthday $b$ and IQ $iq$. You can assume that both $b$ and $iq$ are positive integers.
- $\text{AVERAGE}(b1, b2)$ returns the average IQ among people born between days $b1$ and $b2$ inclusively (assume $b1 < b2$).

You decide to create your data structure by augmenting a red-black tree.

(a) Describe the data contained in each node of your augmented RBT.

(b) Describe briefly your implementation of INSERT operation (write a description in English, do not write pseudo-code).
(c) Describe your implementation of AVERAGE operation. Argue briefly why it is correct, and analyze its running time (in terms of the total number \( n \) of friends inserted so far).

Note: it may be easier to use a diagram.

4. A pharmacist has \( W \) pills and \( n \) empty bottles. Let \( \{p_1, p_2, ..., p_n\} \) denote the number of pills that each bottle can hold.

a) Describe a greedy algorithm, which, given \( W \) and \( \{p_1, p_2, ..., p_n\} \), determines the fewest number of bottles needed to store the pills. Prove that your algorithm is correct (that is, prove that the *first* bottle chosen by your algorithm will be in some optimal solution).

b) How would you modify your algorithm if each bottle also has an associated cost \( c_i \), and you want to minimize the total cost of the bottles used to store all the pills?

Give a recursive formulation of this problem (formula is enough). You do not need to prove correctness.

(Hint: Let \( \text{MinPill}[i, j] \) be the minimum cost obtainable when storing \( j \) pills using bottles among 1 through \( i \). Thinking of the 0-1 KNAPSACK PROBLEM formulation may help. )

c) Describe briefly how you would design an algorithm for it using dynamic programming and analyse its running time.

5. [22.1-5] The square of a directed graph \( G = (V, E) \) is a graph \( G^2 = (V, E^2) \) such that \((u, w) \in E^2\) if and only if, for some \( v \in V \), both \((u, v) \in E\) and \((v, w) \in E\). Thus, \( G^2 \) contains an edge between \( u \) and \( w \) whenever \( G \) contains a path with exactly two edges between \( u \) and \( w \). Describe efficient algorithms for computing \( G^2 \) from \( G \) from both the adjacency-list and the adjacency-matrix representation of \( G \). Analyze the running time of your algorithms.
Dijkstra’s SSSP algorithm works on general graphs with non-negative weights. The running time of Dijkstra’s algorithm is $O(|E| + |V| \times \text{INSERT} + |V| \times \text{DELETE-MIN} + |E| \times \text{CHANGE-KEY})$. Assuming the graph is connected and the priority queue is implemented as a heap the running time is $O(|E| \log |V|)$. The running time can be improved to $O(|E| + |V| \log |V|)$ using improved versions of priority queue (for instance the Fibonacci heap, which supports INSERT and CHANGE-KEY in $O(1)$ time amortized, and DELETE-MIN in $O(\log n)$ amortized). While Dijkstra’s algorithms gives the best known upper bounds for general SSSP with general non-negative weights and linear space, improved algorithms are known for special classes of graphs. In the following problems you will investigate several examples and derive improved bounds for computing SSSP.

6. **Shortest path for Directed Acyclic Graphs (DAGs):** Let $G = (V, E)$ be a DAG and let $s$ be a vertex in $G$. Find a linear time $O(|V| + |E|)$ algorithm for computing SSSP(s). What vertices are reachable from $s$? Sketch a proof that your algorithm is correct. Does your algorithm need the constraint that the edge weights are non-negative?

7. Consider a directed weighted graph with non-negative weights and $V$ vertices arranged on a rectangular grid. Each vertex has an edge to its southern, eastern and southeastern neighbours (if existing). The northwest-most vertex is called the root. The figure below shows an example graph with $V=12$ vertices and the root drawn in black:

Assume that the graph is represented such that each vertex can access all its neighbours in constant time.

(a) How long would it take Dijkstra’s algorithm to find the length of the shortest path from the root to all other vertices?

(b) Describe an algorithm that finds the length of the shortest paths from the root to all other vertices in $O(V)$ time.

(c) Describe an efficient algorithm for solving the all-pair-shortest-paths problem on the graph (it is enough to find the length of each shortest path).