Advanced Data Structures

Mid-Semester Examination – Set 3

Time: 3 hours **Maximum Marks: 100**

Instructions

- This question paper contains Section A of 20 marks and Section B of 80 marks.
- Section A consists of 10 compulsory questions of 2 marks each.
- Section B consists of 2 questions of 5 marks each and 7 questions of 10 marks each. Attempt all.
- Assume necessary data if not given.
- Diagrams must be drawn wherever required.

Section A – Short Answer Questions

Each question carries 2 marks. Answer all questions briefly.

1. Define a binomial tree and its recursive structure.

   Answer: A binomial tree of order k, denoted B$_k, isatreewithroothavingkchildren, eachabinomialtreeoforders0tok-1; ithas2^k nodesandsatisfiesheaporder.$

2. What is the significance of the degree field in Fibonacci heaps?

   Answer: The degree field tracks the number of children per node, enabling efficient linking during union by matching degrees and avoiding recounting subtrees.

3. Distinguish between weight-balanced and height-balanced search trees.

   Answer: Height-balanced trees (e.g., AVL) limit height differences to 1; weight-balanced (e.g., BB[ ]) maintain subtree size ratios within :(1-) for probabilistic balance.

4. Explain the role of sentinels in red-black trees.

   Answer: Sentinels are NIL nodes treated as black leaves, simplifying boundary checks in rotations and insertions by avoiding null pointer tests.

5. What is a splay operation in splay trees?

   Answer: A splay moves an accessed node to the root via 2-3 zig/zig-zig/zig-zag rotations, amortizing costs based on access frequency.

6. Define a succinct data structure for sequences.

   Answer: Succinct structures represent sequences (e.g., bit strings) in o(n) extra space beyond the information theoretic minimum, supporting rank/select queries in O(1) time.

7. What is a dictionary in the context of hashing?

   Answer: A dictionary is an abstract data type for dynamic sets supporting insert, delete, and search in average O(1) time, often implemented via hash tables.

8. Describe a cut-set in graph theory.

   Answer: A cut-set is a minimal set of edges whose removal increases the number of connected components, separating the graph into disconnected parts.

9. State Euler's formula for planar graphs.

   Answer: For a connected planar graph, V - E + F = 2, where V=vertices, E=edges, F=faces (including outer); it bounds edges E 3V - 6 for simple graphs.

10. What is vertex covering in graphs?

    Answer: A vertex cover is a set of vertices that includes at least one endpoint of every edge; the minimum vertex cover problem is NP-hard.

Section B – Descriptive Questions

Attempt all questions.

Questions carrying 5 marks each

1. Compare the merging process in binomial queues with that in binary heaps, highlighting time complexities.

   Answer (5 marks):

   Binomial queues merge by linking roots of equal-degree trees (O(log n) links), forming a new forest efficiently. Binary heaps merge by inserting one heap's elements into the other (O(n log n) worst-case), lacking native support.

   This makes binomial queues superior for repeated unions (e.g., in Prim's algorithm), while binary heaps excel in single-heap operations like heap sort.

   Time: Binomial union O(log n); binary O(n).

2. Discuss the applications of disjoint set structures in Kruskal's minimum spanning tree algorithm.

   Answer (5 marks):

   In Kruskal's, DSU tracks connected components during edge additions: union merges cycles (via find on endpoints), ensuring tree acyclicity. Path compression/union-by-rank yield near-O(1) per edge check.

   For graph with m edges, total O(m (n)) time, efficient for sparse graphs; avoids explicit cycle detection.

Questions carrying 10 marks each

3. Write pseudocode for the percolate-up operation in a min-heap and analyze its time complexity. Apply it to build a min-heap from array [9, 5, 3, 1, 7] step-by-step, drawing the heap after each insertion.

   Answer (10 marks):

   Pseudocode (Min-Heap Percolate-Up):

   ```
   Percolate-Up(A, i):  // A[1..n], i > 1
     while i > 1 and A[i] < A[parent(i)]:
       swap A[i], A[parent(i)]
       i = parent(i)  // parent(i) = i//2
   ```

   Time: O(log n), height traversals worst-case.

   Build Process: Start empty. Insert 9 (root). Insert 5: swap to [5,9]. Insert 3: percolate to root [3,9,5]. Insert 1: [1,3,5,9]. Insert 7: right of 3, no swap [1,3,5,9,7].

   Bottom-up build alternative O(n).

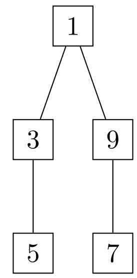

Figure 1: Final Min-Heap (array view: indices 1-5).

4. Explain augmented data structures with an example of order-statistic trees. Implement select(k) operation in pseudocode and discuss its utility in median finding.

   Answer (10 marks):

   Augmented DS: Base structure + node metadata for fast aggregate queries (e.g., BST + sizes for order stats).

   Example: Order-statistic tree: Each node stores subtree size; supports OS-Select(k) to find k-th smallest in O(log n).

   Pseudocode (OS-Select):

```
OS-Select(x, k):  // x node, k rank
  r = size(x.left) + 1
  if k == r: return x.key
  if k < r: return OS-Select(x.left, k)
  else: return OS-Select(x.right, k - r)
```

   Utility: Median = OS-Select((n+1)/2); enables dynamic order stats without sorting.

   In databases, quick percentile queries.

5. Describe temporal data structures and persistent binary search trees. Outline how persistence is achieved via path copying, with a diagram showing versions after two updates.

   Answer (10 marks):

   Temporal DS: Handle evolving data over time, querying past/current states without recomputation.

   Persistent BST: Immutable updates create new root/version, sharing unchanged paths.

   Achievement: On insert, copy path from root to insertion point (O(log n) nodes), update new copy; old root unchanged.

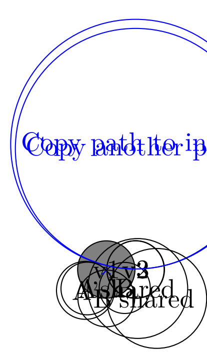

Figure 2: Versions: v1 initial, v2 update on B, v3 on A (shared nodes dotted implied).

Space O(log n) per update; used in version control.

6. Detail the insert operation in cuckoo hashing, including eviction chains. Simulate insertions of 10, 23, 37, 4 into a table of size 8 with h1(k)=k mod 8, h2(k)=(2k+1) mod 8, and resolve any cycle.

   Answer (10 marks):

   Insert in Cuckoo: Hash to h1; if occupied, evict to h2 of evicted, alternate until empty slot or max trials (then rehash).

   Handles two tables implicitly via hashes.

   Simulation: - 10: h1=2, empty → slot2=10. - 23: h1=7, empty → slot7=23. - 37: h1=5, empty → slot5=37. - 4: h1=4, empty → slot4=4. No evictions.

   If cycle (e.g., add 12: h1=4 occupied by4, evict4 to h2(4)=1 mod8=1 empty; now slot4=12, slot1=4).

   No cycle here; load low. Worst: long chain O(log n) amortized, but rare loops trigger rehash.

   Efficient for static sets.

7. Explain graph partitioning and its relation to min-cut problems. For a graph with vertices 1,2,3,4,5 and edges (1-2:1,2-3:2,3-4:1,4-5:3,1-5:4), find a balanced partition minimizing edge cut.

   Answer (10 marks):

   Graph Partitioning: Divide vertices into subsets of given sizes minimizing crossing edges; NP-hard, approx via Kernighan-Lin or spectral.

   Relation: Min-cut finds global min edges between S/T; partitioning adds balance constraint.

   Example: n=5, aim |A|=|B|=2/3? Say 2-3 split.

   Possible: A=1,2, B=3,4,5: cut 2-3:2. A=1,5, B=2,3,4: cut 1-2:1 +4-5:3=4 worse. A=2,3, B=1,4,5: cut 1-2:1 +3-4:1=2 same.

   Min cut 2; balanced via heuristics.

   Used in VLSI design for load balance.

8. Describe the concept of covering and partitioning in graphs. Prove that every graph has a vertex partition into independent sets equal to its chromatic number.

   Answer (10 marks):

   Covering: Edge/vertex cover includes all edges/vertices incident. Partitioning: Disjoint sets union to whole graph.

   Vertex Partition: Graph colors partition vertices into (G) independent sets (no intra-edges).

   Proof: Proper -coloring assigns colors 1 to , each color class independent (no adjacent same color). Classes disjoint, cover V. Converse: If partition into k independents, color each class i, needs k colors.

   Thus, (G) = min k for such partition.

   E.g., bipartite =2, two independents.

   Applications: Scheduling, register allocation.

9. Analyze the time complexity of heap sort. Write pseudocode for heap sort on an array and trace its execution on [4, 1, 3, 2, 16, 9, 10, 14, 8, 7], showing the heapify phase.

   Answer (10 marks):

   Time Complexity: Build O(n), n extracts O(n log n), total O(n log n) worst/avg.

   Stable? No.

   Pseudocode:

```
Heap-Sort(A):
  Build-Max-Heap(A)  // O(n)
  for i = n downto 2:
    swap A[1], A[i]
```

```
    Max-Heapify(A, 1, i-1)  // O(log i)

Max-Heapify(A, i, n):  // percolate down
  largest = i
  l=2*i; r=2*i+1
  if l<=n and A[l]>A[largest]: largest=l
  if r<=n and A[r]>A[largest]: largest=r
  if largest != i:
    swap A[i], A[largest]
    Max-Heapify(A, largest, n)
```

Trace Build: Array [4,1,3,2,16,9,10,14,8,7]. After build-max: [16,14,10,8,7,9,3,2,4,1] (heapified bottom-up).

Then extracts sort descending, swap root to end, heapify.

Final sorted ascending via reverses.