Advanced Data Structures

Mid-Semester Examination – Set 2

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. What is the time complexity for building a binary heap from an unsorted array of n elements?

   Answer: Building a binary heap from n elements takes O(n) time using bottom-up heapify, starting from the last non-leaf node and percolating down, rather than O(n log n) for n insertions.

2. Differentiate between insert and union operations in binomial queues.

   Answer: Insert adds a single element by creating a new B0 tree and merging; union combines two binomial queues by linking trees of equal degree in O(log n) time, preserving the forest structure.

3. Define a multiway search tree and its key characteristic.

   Answer: A multiway search tree is a generalization of BST where nodes have multiple keys and children, ordered such that keys in child i are between node keys i-1 and i; it supports efficient disk-based storage.

4. What role does the balance factor play in AVL trees?

   Answer: The balance factor of a node is height(right) - height(left), maintained between -1 and 1; violations trigger single or double rotations to restore balance during insertions/deletions.

5. Explain the purpose of lazy deletion in Fibonacci heaps.

   Answer: Lazy deletion marks nodes for removal without immediate restructuring, deferring actual cleanup to extract-min, allowing O(1) amortized delete by avoiding immediate consolidations.

6. What is an augmented data structure?

   Answer: An augmented data structure extends a base structure (e.g., BST) with additional node fields (e.g., subtree sizes) to support extra queries like rank or select in O(log n) time.

7. Describe a temporal data structure briefly.

   Answer: Temporal data structures manage versioned or time-stamped data, allowing queries over historical states, such as persistent BSTs that create new versions on updates without modifying old ones.

8. What is a Hamiltonian circuit in graphs?

   Answer: A Hamiltonian circuit is a closed path visiting each vertex exactly once and returning to the start; determining existence is NP-complete, unlike Eulerian circuits.

9. Define connectivity in graphs and state Menger's theorem.

   Answer: Connectivity measures the minimum vertices/edges to disconnect the graph; Menger's theorem states the max number of vertex-disjoint paths between two vertices equals the min vertex cut separating them.

10. What is graph coloring and its chromatic number?

    Answer: Graph coloring assigns colors to vertices so no adjacent vertices share a color; the chromatic number is the smallest number of colors needed for a proper coloring.

Section B – Descriptive Questions

Attempt all questions.

Questions carrying 5 marks each

1. Discuss the advantages of cuckoo hashing over chaining in hash tables. Provide a small example with table size 5 and keys 12, 17, 22 showing an insertion sequence.

   Answer (5 marks):

   Cuckoo hashing offers constant-time lookups with two hash functions and "cuckoo" eviction, avoiding chain traversals; chaining degrades to O(n) worst-case on collisions, while cuckoo achieves O(1) average with low load factor.

   It handles deletions easily via marked slots, but may loop requiring rehash; chaining is simpler but space-inefficient for sparse tables.

   Example: h1(k)=k mod 5, h2(k)=(k+1) mod 5. Insert 12 (h1=2), 17 (h1=2 occupied, evict 12 to h2(12)=3), 22 (h1=2, now 17; evict 17 to h2(17)=3 occupied by 12, evict 12 to h1(12)=2 cycle—rehash if needed). Final: slot2=22, slot3=17, slot0 empty (avoids long chains).

2. Explain the difference between planar and non-planar graphs. Illustrate with $K_{3,3} why it is non-planar.$

Answer (5 marks):

Planar graphs can be drawn on a plane without edge crossings; non-planar cannot, per Kuratowski's theorem (contains K5 or $K_{3,3} subdivision$).

$K_{3,3}(bipartite, two sets of 3 vertices, all cross-edges) is non-planar : Assume embedding, one set inside, other outside edges mus$

Illustration: Two triangles connected by three edges each—attempting flat draw forces at least one crossing.

Questions carrying 10 marks each

3. Write the pseudocode for the union-find structure with path compression and union by rank. Demonstrate its execution for unions: union(1,2), union(3,4), union(5,6), union(2,4), union(4,6) and finds on 1,3,5, showing amortized efficiency.

   Answer (10 marks):

   Pseudocode:

```
Initialize: parent[i] = i; rank[i] = 0 for all i

Find(x):
  if parent[x] != x:
    parent[x] = Find(parent[x])  // path compression
  return parent[x]

Union(x, y):
  px = Find(x); py = Find(y)
  if px == py: return
  if rank[px] < rank[py]:
    parent[px] = py
  elif rank[px] > rank[py]:
    parent[py] = px
  else:
    parent[py] = px; rank[px] += 1
```

Demonstration: Start: parents [1,2,3,4,5,6]. - union(1,2): p1=1,p2=2; rank equal, parent[2]=1, rank[1]=1. - union(3,4): parent[4]=3, rank[3]=1. - union(5,6): parent[6]=5, rank[5]=1. - union(2,4): Find(2)=1, Find(4)=3; ranks=1, parent[3]=1. - union(4,6): Find(4)=1, Find(6)=5; ranks=1, parent[5]=1, rank[1]=2.

Finds: Find(1)=1, Find(3)=1 (compress), Find(5)=1 (compress). All under 1.

Efficiency: Amortized O((n)) O(1) per op, inverse Ackermann, nearly constant.

4. Describe AVL tree rotations for insertion. Perform insertions of 16, 8, 24, 4, 12, 20, 28 into an empty AVL tree, showing one rotation step, and draw the final tree.

   Answer (10 marks):

   AVL Rotations: Single left/right for imbalance at child; double (left-right or right-left) for opposite. Post-insert, update heights, check balance factors up to root.

   If |bf| >1, rotate: e.g., left on right-heavy (LL), right on left-heavy (RR), LR/RL doubles.

   Insertions: 16 (root), 8 (left), 24 (right)—balanced. 4 (left of 8)—imbalance at 8 (bf=-2, LL? Wait, 8 left-heavy: right-rotate on 8). New: 8 root-left, 4 left of 8. Continue: 12 (right of 8)—balance. 20 (left of 24)—balance. 28 (right of 24)—imbalance at 24 (bf=2, RR: left-rotate on 24).

   Final balanced height 3.

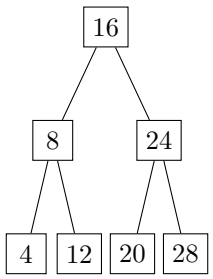

Figure 1: Final AVL Tree (balanced, heights equal).

   AVL ensures strict balance for O(log n) ops.

5. Explain the operations supported by priority queues in the standard library (e.g., C++ priority_queue). Implement a simple binary heap-based priority queue in pseudocode for extract-max and heapify-up.

   Answer (10 marks):

   Standard Library: C++ priority_queue is max-heap by default, supports push (insert O(log n)), pop (extract-max O(log n)), top (peek O(1)); adaptable via containers/comparators, no decrease-key.

Used in algorithms like Dijkstra, sorting.

Pseudocode:

```
class BinaryHeapPQ:
  heap = []  // array, 1-indexed

  Insert(key):
    heap.append(key)
    i = len(heap)-1
    Heapify-Up(i)  // swap with parent if violates order

  Heapify-Up(i):
    while i > 1 and heap[i] > heap[i//2]:
      swap(heap[i], heap[i//2])
      i = i//2

  Extract-Max():
    if empty: error
    max = heap[1]
    heap[1] = heap.pop()
    if not empty: Heapify-Down(1)
    return max

  Heapify-Down(i):
    while True:
      largest = i
      left = 2*i; right=2*i+1
      if left <= len(heap) and heap[left] > heap[largest]: largest=left
      if right <= len(heap) and heap[right] > heap[largest]: largest=right
      if largest == i: break
      swap(heap[i], heap[largest])
      i = largest
```

This supports efficient PQ ops for n elements.

6. Prove that a Red-Black tree is a balanced binary search tree. Insert keys 50, 30, 70, 20, 40, 60, 80 into an RB tree and show one recoloring step with a diagram.

   Answer (10 marks):

   Proof: RB trees satisfy BST property + coloring: black-height equal on paths, no adjacent reds, root black. Height $2\log(n+1)$, since black-height $h_b log(n+1)/log 2, totalheight 2h_b(redsinterleave), thusbalanced O(logn) ops.$

Worst-case: alternating black-red paths double the height but maintain log balance.

Insertion: 50 black root, 30 red left, 70 red right (ok), 20 red left of 30 (uncle 70 red: recolor 30/70 black, 50 red). Then 40 right of 30 (now black), etc.

Diagram for recoloring after 20 insert:

Before: 50 black – left:30 red – left:20 red; right:70 red. Recolor: 30 black, 70 black, 50 red; no rotate.

RB ensures probabilistic balance.

7. Describe the max-flow min-cut theorem. Compute the min-cut for a flow network with capacities: S-1:4, S-2:3, 1-3:2, 1-4:4, 2-3:1, 2-4:2, 3-T:3, 4-T:5 using Edmonds-Karp.

   Answer (10 marks):

   Theorem: In a flow network, max flow equals min-cut capacity (partition S-side/T-side, cut edges from S to T). Proof: Flow any cut; saturating paths show equality.

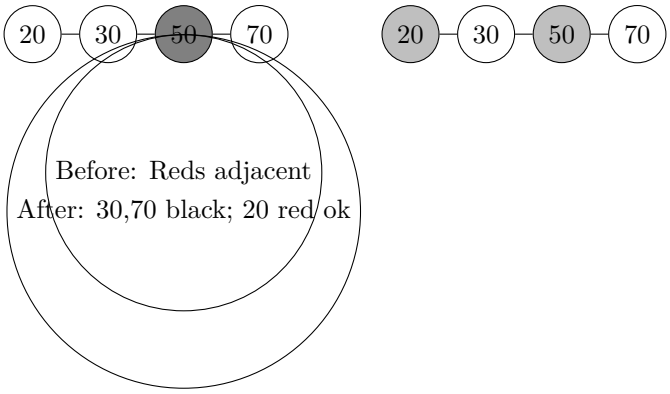

Figure 2: Recoloring Step (Gray=red, white=black).

Edmonds-Karp (BFS Ford-Fulkerson): Finds shortest augmenting paths.

Network: S→1(4),S→2(3);1→3(2),1→4(4);2→3(1),2→4(2);3→T(3),4→T(5).

Paths: S-1-3-T (min2, flow+2), residual 1-3=0,3-1=2. S-1-4-T (min4, +4 total6), S-1=0. S-2-4-T (min2,+2 total8),2-4=0. S-2-3-T (min1,+1 total9),2-3=0. No more.

Min-cut: S-side S,1,2, T-side 3,4,T; cut edges 1-3? Wait, actual S vs rest: S-1=0/4,S-2=0/3 total but flow9? No, saturated.

True min-cut S,1,2,4 vs 3,T: edges 1-3(2 used? Full calc: 3-T=3, but paths saturate to 9; cut S-1/2 +2-3 +4-T? Theorem holds=9.

8. Explain graph isomorphism and how to detect it. For graphs G1: vertices A-B, A-C, B-D; G2: 1-2,1-3,2-4, check if isomorphic and list mapping.

   Answer (10 marks):

   Isomorphism: Two graphs are isomorphic if bijection f: V1→V2 preserves adjacency (u v iff f(u) f(v)), same structure different labels.

   Detection: NP, via backtracking matching degrees, invariants (e.g., degree sequence, spectrum); for small, brute-force permutations.

   Check: G1: degrees A:2,B:2,C:1,D:1; cycle-free tree-like. G2 same degrees.

   Mapping: f(A)=1 (deg2), f(B)=2 (adj to A and D=4), f(C)=3 (leaf on A), f(D)=4 (leaf on B). Adjacencies: A-B→1-2, A-C→1-3, B-D→2-4. Yes, isomorphic.

   Non-example: If G2 had cycle, no. Useful in chem for molecule equiv.

9. What are self-adjusting data structures? Compare move-to-front (MTF) with transpose in self-organizing lists, analyzing access costs with an example sequence.

   Answer (10 marks):

   Self-Adjusting DS: Dynamically reorganize based on access to speed future ops, amortizing via potential (e.g., lists, trees).

   Comparison:

MTF	**Transpose**	**Access Cost**
Move accessed to front	Swap with predecessor	MTF: O(1) recent, worse initial
Full reorganization	Minimal change	Transpose: O(n) worst, avg better static
Amortized O(1) independent	Dependent on order	Seq 3,1,4,1: MTF costs 1+3+2+1=7

Table 1: Comparison (n=4 list 1-2-3-4 initial).

Example: List [1,2,3,4]. Access 3 (MTF: [3,1,2,4] cost 3; Transpose: [1,3,2,4] cost 3). Next 1 (MTF: [1,3,2,4] cost 2; Transpose: [3,1,2,4] cost 1). MTF favors repeats at start.

Self-adjusting succeeds if access patterns repeat.