Data Structures and Algorithms

Week 14: Course Review

Subodh Sharma and Rahul Garg

{svs,rahulgarg}@iitd.ac.in

Lists

- **Linked List:**
  - Insert@Head:
    - Create new node n; Set whatever is being pointed by head to n->next
  - Delete@Head:
    - Store current head in n; move head to head->next; remove n
- **Doubly Linked List:**
  - Insert@Head:
    - Create new node n; n->prev = Null; n->next = Head; Head->prev = n
  - Delete@Head: Let the node be n
    - Head = n->next; n->next->prev = n-> prev; n->prev->next = n->next

Lists

- **Problems: Reverse a doubly linked list in groups of size k**
    - Traverse each group of size k:
        - **Swap** next and prev pointers of each node in the group
    - Adjust the prev of the first node of the next group (if exists) to point to the first node of the reversed group
    - Adjust the next of the last node of previous group (if exists) to the last node of the reversed group
    - Update the current to the first node of the next group

Lists

- **Problems: Split a circular list in two halves each being circular**
  - Find the middle of the list and break the list maintaining circularity

- More problems:
  - Sort a doubly linked list using Merge sort
  - Leader election in a circular list of size n wherein we keep removing every m^{th} person in the list. The last one who remains is the leader

Stacks

- Invariants:
  - Last-in-first-out
  - Can’t pop on an empty stack, and can’t insert in a full stack (if stack has size limitation)
  - Only top element is accessible — Can’t access middle or bottom elements
- **Problem:** Prefix expression evaluation (*+12-34)
  - Start from the last; Push if an operand and pop top two if an operator and push the result back on stack!
- **Problem:** Balanced parenthesis — Open brackets are closed by the same type of brackets AND Open brackets are closed in the correct order
  - If an opening bracket, push onto the stack
  - If a closing bracket, check if empty or top of the stack is a matching opening bracket.
  - At termination of input processing, stack should be empty

Queues

- Invariants:
  - First-in-first-out: Means we have to keep two pointers — front and rear
  - Capacity invariants similar to stacks
- **Problems:**
  - Implementing a stack using two queues
    - Push: Enq in Q1 and Pop: Copy everything but the last element from Q1 to Q2; remove the last element of Q1; swap the names of Q1 and Q2
  - Maximum in a sliding window of size k
  - Perform a BFS over a tree or a graph
    - Population Infection: Given a 2D grid where each cell (representing a person) can either be healthy, recovering, or infected. Every minute any healthy person that is adjacent to an infected person can become infected. Calculate the minimum time in which in the population there is no healthy person left.

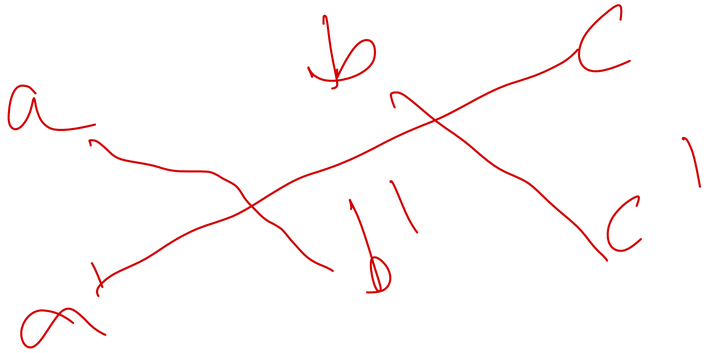
Hashing and Dictionaries

- Purpose of hashing — quick retrieval of data; quick check for integrity
  - Mapping of arbitrary sized input to fixed size input
  - Techniques to handle collision: Chaining, Open addressing — linear probing, ..
- Dictionaries are key-value pairs — keys are immutable
  - Lookups, insertions and deletions in O(1) for average case
- Understand the trade-offs of collision resolution techniques
- Check the use of these concepts in real-world applications:
  - Database indexing, Symbol tables, Network routing, Secure communication(?)

Graphs — ADTs and Traversals

- Directed and undirected graphs
- Weighted vs unweighted graphs
- Graph representations:
  - Adjacency list : space-efficient for sparse graphs and slower lookup time
  - Adjacency matrix: Faster lookups, efficient for dense graphs and space inefficient
- Graph traversals
  - DFS and BFS: How are they used to find connected components in a graph?
  - DFS is more memory efficient, BFS has simplicity

Graphs —SCCs



- SCCs in directed graphs
  - Tarjan's alg: How does it work? What is the role of low-link values? What is the significance of stack in the algorithm? What is the invariant preserved in the Alg.?
    - What is the time complexity?
      - How can it be improved with a different choice of a data structure?
    - What is its behaviour on DAGs?
    - What are some interesting applications?
      - Find **critical** or **bridge** edges. Removal of such edges increases the SCCs in a graph! (SAMPLE EXAM QUESTION)
      - Boolean satisfiability for a 2-SAT problem (SAMPLE EXAM QUESTION)
        - 2SAT is solvable if the literal and its negation doesn't exist in the same SCC
      - Optimal Road networks: cities connected by one-way roads, determine minimum number of new roads required to make them strongly connected?

① critical edges occur in all MST ≤ n-1

(V+E) log(V)
+E+VlogV)

Graphs — TopoSort

- Applicable to DAGs
- Preserve the dependency ordering on vertices in the sorted list
- Via DFS, Kahn’s algorithm
- Time complexity?
- Applications:
  - Chemical and molecular simulations respecting some dependencies
  - Circuit design
  - Deadlock detection(?)

Graphs —Shortest Paths

- SSSPs
  - Dijkstra’s algorithm- Time complexity with various data structure choices
    - Greedy algorithm,
  - Bellman-ford algorithm — can handle negative edge weights and detect negative weight cycles!
- APSPs
  - Warshall’s algorithm
    - Time complexity

Minimum Spanning Trees

- Prim's Alg — Start expanding from a node into a single growing tree with one edge at a time
  - Efficient for dense graphs
  - Uses heaps to extract minimum element
  - O(E + V log V)
- Kruskal's Alg — Sort the edges and choose min-weighted edge at each step; explicit checks for cycles
  - Good for sparse graphs; Time complexity: O(E log V) or O (E log E);
  - Uses disjoin-set data structure to keep track of connected components