Midterm Review

- Shivani
Format

• Question 1 : Short Answers 4 Problems
• Question 2 : Mechanical Execution
• Question 3 : Design/ Analysis (D/Q)
• Question 4 : Design/ Analysis (Graph/Shortest Path)
General Tips

• Read the entire Midterm first
• Understand that Questions 1 and 2 are intended to be "very doable". Don't lose points on these!!!
• For Questions 1 and 2, it is VERY IMPORTANT to be able to execute basic algorithms (SCC-finding, Dijkstra, etc.) WITHOUT MAKING SILLY MISTAKES. Please spend some time actually practicing "mechanical execution”
• On algorithm design problems, remember what your tools are: Master Theorem, Dijkstra, etc. Think about what fits or what is possible within constraints given (e.g., big-O time complexity bounds).
• Once you've written down pseudocode, test it on a small example and see whether your algorithm is well-defined, whether it actually works as you imagined, etc. A good algorithm designer always looks for holes/flaws in his/her algorithm design!
• Make cheat-sheet early
• Go through previous years Midterm (try to solve at least one of the Midterms in 80 minutes).
• Go through Previous year Quizzes, Things to know, Solving recurrence relation and big-O from Additional resources.
Question 1 : Short Answers

• Tips :
  • Big-O, Time Complexity, Short Mechanical Problems, Conceptual Problems
  • Review of previous years’ short-answers would help
Question 2 : Mechanical Problem

• Tips :
  • Execute all the algorithms, concept taught on class on small graphs
  • DFS, BFS
  • Topological Ordering
  • SCC
  • Shortest Path (Dijkstra, Bellman-Ford)
  • Dijkstra might not give correct shortest path for graph with negative edges, why? Bellman-Ford with negative edges.
Question 3 : Design/Analysis (D/Q)

• Tips :
  • Understand Divide and Conquer
  • Go through examples covered in class step by step
  • Try to write pseudocode without peeking in notes
  • In class, we often noted a "natural ordering" that could guide the "divide" into subproblems. We also noted that, depending on the time budget, preprocessing (e.g., a one-time sort) could help.
  • Time Complexity (recurrence relation)
  • Master Theorem
Question 4: Design/Analysis (Graph, Shortest-Path)

• Tip:
  • Go through HW-2 problems (4-5).
  • Please attend office hours if you are having any difficulty with HW-2.
Cheat Sheet Item (Class input)

- Masters Theorem
- Pseudocodes (also implement algorithms on small graph rather than writing pseudocode on cheat-sheet)
- Edge types – pre and post label
- SCC basic idea
- Time complexities of algorithm
- Limit theorem
- Orders (e.g., log(n) is order polynomial))
- Stirling’s approximation
Big-O

• $f(n) = O(g(n))$ if there exist positive constants $c$ and $n_0$ such that $0 \leq f(n) \leq c(g(n))$ for all $n \geq n_0$.

• Alternate definition:
  • $f(n) = O(g(n)) \implies 0 \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

• Tip: Go through HW-1 solutions.

• Examples:
  • $\log(n)$, $n$, $e^n$
  • $2^{\sqrt{\log(n) \log(n)}}$, $n^3$
Master Theorem

• Time complexity for recurrence relation
  • Recurrence relation
    • \( T(n) = aT(n/b) + O(n^d) \)
  • Closed form solution
    • \( T(n) = O(n^d) \) if \( d > \log_b a \)
    • \( T(n) = O(n^d \log(n)) \) if \( d = \log_b a \)
    • \( T(n) = O(n^{\log_b a}) \) if \( d < \log_b a \)

• Examples:
  • \( T(n) = T(n/2) + O(n) \)
  • \( T(n) = 16T(n/2) + O(\log(n)) \)

• How to find \( a, b \) and \( d \) in a algorithm?
DFS

• Edges are explored out of the most recently discovered vertex \( v \) that still has unexplored edges
• Returns three arrays of size \( |V| \) containing pre number, post number, and cc.
• Time Complexity.
• Example:
DFS in Directed Graphs (from class notes)

- A is *root* of tree; all other nodes are A’s *descendants*
- E has *descendants* F, G, H (E is an *ancestor* of G)
- C is the *parent* of D
- H is a *child* of E

**TERMINOLOGY**
DFS in Directed Graphs  (from class notes)

- **Tree** edges = part of the DFS forest
- **Forward** edges = from a node to a non-child descendant
- **Back** edges = to an ancestor in the tree
- **Cross** edges = to neither descendant nor ancestor

Relation to pre label and post label
DFS in Directed Graphs

• Examples
Topological Ordering (from class notes)

• “Inductive” thinking
  • – Any DAG always has a vertex with indegree = 0 (“source”)
  • – Give this vertex the next label, delete vertex and its edges,... \( \rightarrow \) keep doing this to get a topological ordering

• DFS-based algorithm: Run DFS, then perform tasks (label vertices) in decreasing order of post numbers
  • – Because in a DAG, every edge leads to a vertex with lower post number

• Examples

How many topological ordering does this graph have?
Strongly Connected Components

• Note: Undirected case connected components found by DFS

• Definition: \( u \) is connected to \( w \) iff there is a path from \( u \) to \( w \) and a path from \( w \) to \( u \) \textbf{(directed case!)}

• With this definition, we partition \( V \) into \textbf{strongly connected components}

• Algorithm:
  
  Run DFS on \( G^R \)
  
  for \( v \in V \) in decreasing order of post numbers
  
  if not visited\([v]\)
    
    explore\((G,v)\)
  
  output nodes seen as a SCC
Strongly Connected Components

• Examples:
Breadth-First Search (from class notes)

- Suppose we’ve found all nodes at distance $\leq d$
- A node is at distance $d+1$ if:
  - it is adjacent to a node at distance $d$
  - it hasn’t been seen yet
- Example:
Dijkstra Algorithm

• BFS treats all the edge to have same *weight*.

• The usage of the concept of alarm clock in the breadth-first search performed on the graph with dummy nodes results in Dijkstra’s algorithm.

• Examples:

Source: wiki
Bellman-Ford

• Dijkstra might fail when edges have negative weight.
  • Example where Dijkstra fails
• Go through todays lecture notes again!
• Idea : Successive approximation
• Examples :
  • Same example as Dijkstra
  • Second example of Dijkstra with negative edge.
DQ

• Divide (into sub parts)
• Conquer (solve for each part separately)
• Combine separate solutions

• Examples:
  • Given a set of $S$ containing $n$ real numbers, and a real number $x$. We seek an algorithm to determine whether two elements of $S$ exist whose sum is exactly $x$. (Ref: Skienna)
    a) Assume that $S$ is sorted. Give an $O(n)$ algorithm for the problem.
    b) Assume that $S$ is unsorted. Give an $O(n\log(n))$ algorithm for the problem.

• Runtime analysis
Graph/Shortest Path

• Can you tweak any known algorithm
• Store extra information within algorithm
• HW-2 Problems

• Example:
  • Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes $s$ and $t$ such that the distance between $s$ and $t$ is strictly greater than $n/2$. Show that there must exist some node $v$, not equal to either $s$ or $t$, such that deleting $v$ from $G$ destroys all $s$-$t$ paths. (In other words, the graph obtained from $G$ by deleting $v$ contains no path from $s$ to $t$.)
  
  Give an algorithm with running time $O(m + n)$ to find such a node $v$. 
Questions?