CSE 421 Section 5

Midterm Review

Announcements & Reminders

- **HW4** was due yesterday, 4/30
- **HW4.75** is due Wednesday, 5/7
- Your **midterm exam** is on **Monday, 5/5 in class**
    - A **practice midterm** is available on the class site

Preparation Tips

- Read the **practice midterm**. This will give you a good idea of the format, length, and type of questions asked. Work on it yourself before looking at solutions.
- Read the **midterm cover sheet** on the class site. This states our expectations for your solutions and some advice to keep in mind
- **Rewatch lectures** and refresh your understanding of all key concepts
- You may bring one sheet of double sided 8.5x11” paper containing notes.
    - Must write name, student number, and UW NetID
    - Must turn in with exam
- Review solutions to **homework and section problems!**

Midterm Tips

- **Come on time!**
- You have the full lecture period – **attempt everything** for partial credit!
- Write **short, direct solutions**. Use technical English where possible.
- Read each question fully. Most questions do not make you write the full algorithm, correctness and runtime – **don’t waste time** on unneeded parts!
- If you can’t find an algorithm with the optimal runtime, you will get partial credit for a weaker runtime algorithm.

Topics - General / Stable Matching

- Runtime, Big-O notation
- Direct Proofs, Proof By Contradiction, Induction / Strong Induction
- Stable Matching Problem
- Gale-Shapley – proposer optimality, receiver pessimality, runtime

Topics - Graphs

- Runtime, Big-O notation
- Direct Proofs, Proof By Contradiction, Induction / Strong Induction
- Cycles, trees, properties of trees
- Graph search (BFS, DFS), properties of BFS, DFS tree
- Finding connected components, odd cycles, etc.
- Directed graphs (topological sort), DFS on directed graphs

Topics - Greedy

- Interval Scheduling, Minimizing Lateness
- Prim's MST, Kruskal's MST, Dijkstra's Shortest Paths
- Ways to prove correctness:
    - **Greedy Stays Ahead**: Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithms
    - **Structural**: Discover a structure-based argument asserting that the greedy solution is at least as good as every possible solution.
    - **Exchange Argument**: We can gradually transform any solution into the one found by the greedy algorithm with each transform only improving or maintaining the value of the current solution.

Topics – Divide and Conquer

- Recurrences (Master Theorem)
- Binary Search, Merge-sort
- Approximation the Root of a Function
- Finding Closest Points
- Multiplication - Matrix, Integer, Polynomial
- Median, Selection, Quicksort

Topics – Dynamic Programming

- Writing recursive definition of problem
- Calculating runtime (subproblems * time per subproblem)
- Problems discussed:
    - Tribonacci
    - Edit Distance
    - Knapsack

Today's plan

Choose from these problems!

- Problem 1: Short answer
- Problem 2: Stable matching reduction*
- Problem 3: Graph algorithms
- Problem 4: Greedy algorithms*
- Problem 5: Divide and conquer*
- Problem 6: Dynamic programming

**the problem was an extra problem on a previous section handout*

Problems

1	2	3	4	5	6

Problem 1 – Short answer

- If p ranks r first and r ranks p first, then (p, r) must be in every stable matching.

Solution

Problem 1 – Short answer

- If p ranks r first and r ranks p first, then (p, r) must be in every stable matching.

True. If p and r were not matched, then they prefer each other over the current matches, so this is an instability.

Problem 1 – Short answer

Running DFS on a directed acyclic graph may produce:

- ❑ Tree edges
- ❑ Back edges
- ❑ Forward edges
- ❑ Cross edges

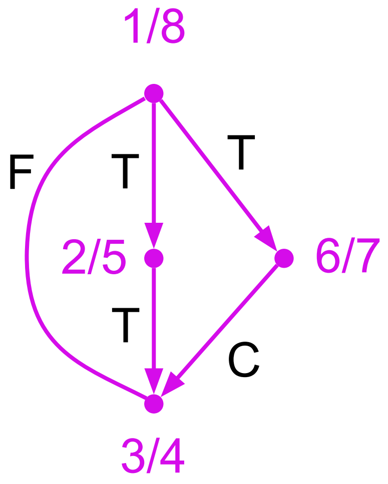
Solution

Problem 1 – Short answer

Running DFS on a directed acyclic graph may produce:

- ❑ Tree edges
- ❑ Back edges
- ❑ Forward edges
- ❑ Cross edges

All except back edges, since they create cycles.



Problem 1 – Short answer

- The recurrence $T(n) = 2T(n/3) + \Theta(n^2)$ simplifies to…?

Solution

Problem 1 – Short answer

The recurrence $T(n) = 2T(n/3) + \Theta(n^2)$ simplifies to...?

$\Theta(n^2)$. By master theorem, since $2 < 3^2$.

Problem 1 – Short answer

- Suppose G has positive, distinct edge costs. If T is an MST of G, then it is still an MST after replacing each edge cost c_e with c_e^2.

Solutio
n

Problem 1 – Short answer

- Suppose G has positive, distinct edge costs. If T is an MST of G, then it is still an MST after replacing each edge cost c_e with c_e^2.

True. Kruskal's (or Prim's) only depends on the relative order of edge costs. Furthermore, because costs are distinct, there is a unique MST, so Kruskal's algorithm found T before and will still find T now.

Problem 1 – Short answer

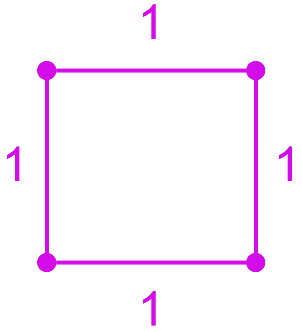
- Let $G = (V, E)$ be a weighted, undirected graph. Consider any cut $S \subseteq V$, and let e be an edge of minimum weight across the cut S. Then every MST contains e.

Solution

Problem 1 – Short answer

Let $G = (V, E)$ be a weighted, undirected graph. Consider any cut $S \subseteq V$, and let e be an edge of minimum weight across the cut S. Then every MST contains e.

False. The theorem requires edge weights be distinct. Consider:



Return to problem select

Problem 2 – Stable matching reduction

There are R riders, H horses with $2H < R < 3H$. Riders and horses have preferences for each other. Also, riders prefer the first 2 rounds. Horses prefer to ride every round.

Set up 3 rounds of rides, so that every rider will ride a horse exactly once, every horse does exactly 2 or 3 rides, and there are no unstable matches.

Solutio
n

Problem 2 – Stable matching reduction

There are R riders, H horses with $2H < R < 3H$. Riders and horses have preferences for each other. Also, riders prefer the first 2 rounds. Horses prefer to ride every round.

Set up 3 rounds of rides, so that every rider will ride a horse exactly once, every horse does exactly 2 or 3 rides, and there are no unstable matches.

For all horses h, create h_1, h_2, and h_3. Add $3H - R$ dummy riders. For preference lists:

- For real riders: original list with h_1 and h_2 replacing h, then original list with h_3's.
- For dummy riders: all h_3 (in any order), then everything else (in any order).
- For horse-in-rounds: original list, then dummy riders in any order.

Solution

Problem 2 – Stable matching reduction

For all horses h, create h_1, h_2, and h_3. Add $3H - R$ dummy riders. For preference lists:

- For real riders: original list with h_1 and h_2 replacing h, then original list with h_3's.
- For dummy riders: all h_3 (in any order), then everything else (in any order).
- For horse-in-rounds: original list, then dummy riders in any order.

Then:

- Every rider is matched because library returns perfect matching.
- Dummy matched to horse in round 1 or 2 is unstable.
- Horse and real rider who prefer each other is unstable.

Return to problem select

Problem 3 – Graph modeling

- Given $(a_1, b_1), \ldots, (a_n, b_n)$, the person living in unit a_i is moving to b_i. Some people may be new arrivals (a_i = null) or moving out (b_i = null). Give an algorithm that returns a valid moving order (every unit is vacated before someone moves in), or “not possible” and a minimal list of pairs that explains why.

Solution

Problem 3 – Graph modeling

- Given $(a_1, b_1), \ldots, (a_n, b_n)$, the person living in unit a_i is moving to b_i. Some people may be new arrivals ($a_i = \text{null}$) or moving out ($b_i = \text{null}$). Give an algorithm that returns a valid moving order (every unit is vacated before someone moves in), or "not possible" and a minimal list of pairs that explains why.

$(2, \text{null}) \rightarrow (1, 2) \rightarrow (\text{null}, 1)$

$A \rightarrow B$ iff A must happen before B

$(3, 4) \quad (4, 3)$

Solution

Problem 3 – Graph modeling

- Given $(a_1, b_1), \ldots, (a_n, b_n)$, the person living in unit a_i is moving to b_i. Some people may be new arrivals ($a_i = \text{null}$) or moving out ($b_i = \text{null}$). Give an algorithm that returns a valid moving order (every unit is vacated before someone moves in), or "not possible" and a minimal list of pairs that explains why.

$$(2, \text{null}) \rightarrow (1, 2) \rightarrow (\text{null}, 1)$$

$$\begin{tikzcd} (3,4) \arrow[r, dash, bend left] \arrow[r, dash, bend right] & (4,3) \end{tikzcd}$$

1. Check for cycles with B/DFS.
   a. If there is a cycle, not possible.

Solution

Problem 3 – Graph modeling

- Given $(a_1, b_1), \ldots, (a_n, b_n)$, the person living in unit a_i is moving to b_i. Some people may be new arrivals ($a_i = \text{null}$) or moving out ($b_i = \text{null}$). Give an algorithm that returns a valid moving order (every unit is vacated before someone moves in), or "not possible" and a minimal list of pairs that explains why.

$(2, \text{null}) \rightarrow (1, 2) \rightarrow (\text{null}, 1)$

$(3, 5) \longrightarrow (4, 3)$

1. Check for cycles with B/DFS.
   a. If there is a cycle, not possible.
   b. If there is no cycle, topo sort.

Return to problem select

Problem 4 – Greedy algorithms

- Given a set $\mathcal{X}$ of integer intervals $[a, b] \subseteq \mathbb{Z}$, find the smallest set $\mathcal{Y} \subseteq \mathcal{X}$ such that every point in any interval of $\mathcal{X}$ belongs to some interval of $\mathcal{Y}$ (i.e. $\mathcal{Y}$ covers $\mathcal{X}$).

Solutio
n

Problem 4 – Greedy algorithms

- Given a set $\mathcal{X}$ of integer intervals $[a, b] \subseteq \mathbb{Z}$, find the smallest set $\mathcal{Y} \subseteq \mathcal{X}$ such that every point in any interval of $\mathcal{X}$ belongs to some interval of $\mathcal{Y}$ (i.e. $\mathcal{Y}$ covers $\mathcal{X}$).

Repeatedly pick the interval with the largest end point that covers the smallest yet-uncovered point.

(For implementation details, see solutions tonight. Naively finding the "smallest yet-uncovered point" is technically correct but slow.)

Solution

Problem 4 – Greedy algorithms

- Repeatedly pick the interval with the largest end point that covers the smallest yet-uncovered point.

Proof sketch: (greedy stays ahead)

- We output $[a_1, b_1], \dots, [a_k, b_k]$ and suppose $[o_1, p_1], \dots, [o_l, p_l]$ is valid and sorted.
- Can prove by induction that $b_i \geq p_i$ for all i (explain why this is enough).
  - After selecting $[a_1, b_1], \dots, [a_{i-1}, b_{i-1}]$ the smallest uncovered point is larger than b_{i-1} and hence not covered by $[o_1, p_1], \dots, [o_{i-1}, p_{i-1}]$ by induction.
  - If $[o_i, p_i]$ does not cover it, by sortedness, other solution is invalid.
  - If $[o_i, p_i]$ does cover it, then $b_i \geq p_i$ because that was our greedy criterion.

Return to problem select

Problem 5 – Divide and conquer

- $A[1..n]$ is a mountain if there is a peak i such that
$$A[1] < \cdots < A[i-1] < A[i] \text{ and } A[i] > A[i+1] > \cdots > A[n].$$
The peak may be at 1 or n. Given a mountain, find the peak in $O(\log n)$ time.

Solutio
n

Problem 5 – Divide and conquer

- $A[1..n]$ is a mountain if there is a peak i such that

$$A[1] < \cdots < A[i-1] < A[i] \text{ and } A[i] > A[i+1] > \cdots > A[n].$$

The peak may be at 1 or n. Given a mountain, find the peak in $O(\log n)$ time.

function peakFinder(i, j)

(base case omitted for slide brevity)

1. $m \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$
2. **if** $A[m+1]$ exists and $m+1 \le j$ and $A[m] < A[m+1]$
   a. **return** peakFinder($m+1, j$)
3. **else if** $A[m-1]$ exists and $i \le m-1$ and $A[m-1] > A[m]$
   a. **return** peakFinder($i, m-1$)
4. **else return** m

(checking for edge cases)

Solutio
n

Problem 5 – Divide and conquer

function peakFinder(i, j)

1. $m \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$
2. **if** $A[m+1]$ exists and $m+1 \leq j$ and $A[m] < A[m+1]$
   a. **return** peakFinder($m+1, j$)
3. **else if** $A[m-1]$ exists and $i \leq m-1$ and $A[m-1] > A[m]$
   a. **return** peakFinder($i, m-1$)
4. **else return** m

Induction on k:

For all i and j with $j - i = k$, **if** $\boldsymbol{A[i..j]}$ **contains the peak**, peakFinder(i, j) finds it.

(crucial point!)

Solution

Problem 5 – Divide and conquer

- Induction on k:

  For all i and j with $j - i = k$, **if $A[i..j]$ contains the peak**, peakFinder(i, j) finds it.

Three cases for where the peak is:

1. The peak is in $A[m+1..j]$.
   - We end up in the first if branch (explain why).
   - Can apply IH to peakFinder$(m+1, j)$ because the peak is in $A[m+1..j]$!
2. The peak is in $A[i..m-1]$. Similar.
3. The peak is $A[m]$.
   - We end up in the else branch (explain why).

Return to problem select

Problem 6 – Dynamic programming

- Compute the maximum reward going from $(1, 1)$ to (m, n) on a grid, where you gain $R[i, j]$ whenever passing through (i, j). Starting/ending count as passing through. $R[i, j]$ may be negative (penalty) or $-\infty$ (impassible).

Solution

Problem 6 – Dynamic programming

- Compute the maximum reward going from $(1,1)$ to (m,n) on a grid, where you gain $R[i,j]$ whenever passing through (i,j). Starting/ending count as passing through. $R[i,j]$ may be negative (penalty) or $-\infty$ (impassible).

$$\mathrm{OPT}(i,j) = R[i,j] + \max\big(\mathrm{OPT}(i-1,j), \mathrm{OPT}(i,j-1)\big) \qquad i,j > 2$$

$$\mathrm{OPT}(1,1) = R[1,1]$$
$$\mathrm{OPT}(1,j) = R[1,j] + \mathrm{OPT}(1,j-1) \qquad j > 2$$
$$\mathrm{OPT}(i,1) = R[i,1] + \mathrm{OPT}(i-1,1) \qquad i > 2$$

Return to problem select