REVIEW

David Kauchak
CS 140 – Spring 2024

1

Admin

Final
- posted on Gradescope
- due Wednesday (5/8) at 11:59pm (seniors: 5/2 at noon)
- time-limited (3 hours to take – 3.5 hours upload back to Gradescope)
- You may use:
  - the book
  - your notes
  - the class notes
  - the assignments
  - ONLY these things
- Do NOT discuss it with anyone until after Wednesday (5/8) at 11:59pm

2

Test taking advice

- Read the questions carefully!
- Don't spend too much time on any problem
  - if you get stuck, move on and come back
- When you finish answering a question, reread the question and make sure that you answered everything the question asked
- Think about how you might be able to reuse an existing algorithm/approach
- Show your work (I can't give you partial credit if I can't figure out what went wrong)
- Don't rely on the book/notes for conceptual things
  - Do rely on the notes for a run-time you may not remember, etc.

3

High-level approaches

Algorithm tools
- Divide and conquer
  - assume that we have a solver, but that can only solve sub-problems
  - define the current problem with respect to smaller problems
  - Key: sub-problems should be non-overlapping
- Dynamic programming
  - Same as above
  - Key difference: sub-problems are overlapping
  - Once you have this recursive relationship:
    - figure out the data structure to store sub-problem solutions
    - work from bottom up

4

High-level approaches

Algorithm tools cont.

- Greedy
  - Same idea: most greedy problems can be solve using dynamic programming (but generally slower)
  - Key difference: Can decide between overlapping sub-problems without having to calculate them (i.e., we can make a local decision)
- Flow
  - Min-capacity cut
  - Bottleneck edge
  - Matching problems
  - Numerical maximization/minimization problems

5

Data structures

A data structure

- Stores data
- Supports access to/questions about data efficiently
  - the different bias towards different actions
- No single best data structure

Fast access/lookup?

- If keys are sequential: array
- If keys are non-sequential or non-numerical: hashtable
- Guaranteed run-time/ordered: balanced binary search tree

6

Data structures

Min/max?

- heap
- binomial heap

Fast insert/delete at positions?

- linked list

Others

- stacks/queues
- extensible data structures
- balanced BSTs
- disjoint sets

7

Graphs

Graph types

- directed/undirected
- weighted/unweighted
- trees, DAGs
- cyclic
- connected

Algorithms

- connectedness
- contains a cycle
- traversal
  - dfs
  - bfs

8

Graphs

Graph algorithms cont.

- minimum spanning trees (Prim's, Kruskal's)
- shortest paths
  - single source (BFS, Dijskstra's, Bellman-Ford)
  - all pairs (Johnson's, Floyd-Warshall)
- topological sort
- flow

9

Other topics…

Analysis tools

- recurrences (master method, recurrence trees)
- big-O
- amortized analysis

NP-completeness

- proving NP-completeness
- reductions

10

Proofs: general

Be clear and concise

Make sure you state assumptions and justify each step

Make sure when you're done you've shown what you need to show

11

Proof by induction

1. State what you're trying to prove
   We show that XXX using proof by induction
2. Prove base case
3. State the inductive hypothesis
4. Inductive proof
   a. State what you want to show (may include a variable change, e.g., k in instead of n)
   b. Show a step by step derivation from the left hand side resulting in the right hand side. Give justifications for steps that are non-trivial

12

Prof by induction: structural

Use induction to prove that the number of degree-2 nodes in a non-empty binary tree is 1 less than the number of leaves. Recall that the degree of a vertex in a tree is the number of children that it has.

1. State what you're trying to prove
   We show that XXX using proof by induction
2. Prove base case
3. State the inductive hypothesis
4. Inductive proof
   a. State what you want to show (may include a variable change, e.g., k in instead of n)
   b. Show a step by step derivation from the left hand side resulting in the right hand side. Give justifications for steps that are non-trivial

13

Other (important) places we saw proofs

Recurrences (substitution method)

Big O (needed find constants c n_0)

Greedy algorithm correctness (proof by contradiction or stays ahead—induction —)

Proof of algorithm correctness (MSTs, Flow)

NP-completeness (proving correctness of reductions)

14

Recurrences

Three ways to solve:

- Substitution
- Recurrence tree (may still have to use substitution to verify)
- Master method

15

Recurrences

$$T(n) = 2T(n/3) + d$$

$$T(n) = aT(n/b) + f(n)$$

if $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(n/b) \le cf(n)$ for $c < 1$
then $T(n) = \Theta(f(n))$

$$T(n) = T(n-1) + \log n$$

16

Dynamic programming

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to subproblems

AND

the subproblems are overlapping

Local decisions result in *different* subproblems. Not obvious how to make the first choice.

17

DP advice

Write the recursive definition

- What is the input/output to the problem?
- What would a solution look like? What are the options for picking the first component of a solution?
- Assume you have a solver for subproblems. How can you combine the first decision with answer to subproblem.

Define DP structure: what are subproblems indexed by?

State how to fill in the table (including base cases and where the answer is)

18