REVIEW

David Kauchak
CS 140 – Spring 2023

1

Admin

Final
- posted on Gradescope
- due Wednesday (4/10) at 11:59pm (seniors: 4/4 at noon)
- time-limited (3 hours – with some flexibility to scan, etc.)
- You may use:
  - the book
  - your notes
  - the class notes
  - the assignments
  - ONLY these things
- Do NOT discuss it with anyone until after Wednesday 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 (or memoize)

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
- Linear programming (very light coverage)

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 heaps

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

[**5 points**] A full binary tree is a tree where every node is either a leaf or has two children. (Note this is different than a complete binary tree where all levels are full.) Prove using induction that in a full binary tree the number of internal nodes, I, is equal to the number of leaves, L, minus 1, i.e., $I = L - 1$.

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

All-pairs shortest paths

V * Bellman-Ford: $O(V^2E)$

Floyd-Warshall: $\theta(V^3)$

Johnson's: $O(V^2 \log V + V E)$

19

Floyd-Warshll: Recursive relationship

${d_{ij}}^k$ = shortest path from vertex i to vertex j using only vertices $\{1, 2, \ldots, k\}$

Two options:
1) Vertex k+1 doesn't give us a shorter path
2) Vertex k+1 does give us a shorter path

$${d_{ij}}^{k+1} = \min({d_{ij}}^k, {d_{i(k+1)}}^k + {d_{(k+1)j}}^k)$$

Pick whichever is shorter

20

Floyd-Warshall

Calculate d_{ij}^{k} for increasing k, i.e. k = 1 to V

Floyd-Warshall(G = (V,E,W)):
d^0 = W // initialize with edge weights
for k = 1 to V
for i = 1 to V
for j = 1 to V

$$\text{dijk} = \min({d_{ij}}^{k-1}, {d_{ik}}^{k-1} + {d_{kj}}^{k-1})$$

return d^V

21

Johnson's algorithm

Create G' with one extra node s with 0 weight edges to all nodes
run Bellman-Ford(G',s)

if no negative-weight cycle
reweight edges in G with h(v)=shortest path from s to v

$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

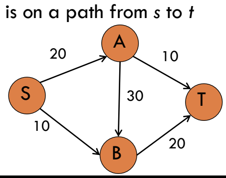
run Dijkstra's from every vertex
reweight shortest paths based on G

22

Flow graph/networks

Flow network

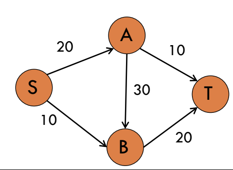
- directed, weighted graph (V, E)
- positive edge weights indicating the "capacity" (generally, assume integers)
- contains a single source $s \in$ V with no incoming edges
- contains a single sink/target $t \in$ V with no outgoing edges
- every vertex is on a path from s to t



23

Max flow problem

Given a flow network: *what is the maximum flow we can send from s to t that meet the flow constraints?*



24

Network flow properties

If one of these is true then all are true (i.e. each implies the the others):

f is a maximum flow

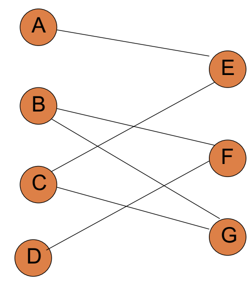
G_f (residual graph) has no paths from s to t

$|f|$ = minimum capacity cut

25

Application: bipartite graph matching

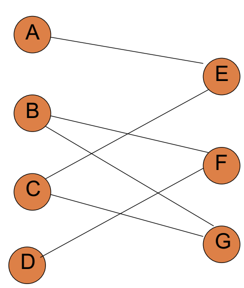
Bipartite graph – a graph where every vertex can be partitioned into two sets X and Y such that all edges connect a vertex $u \in X$ and a vertex $v \in Y$



26

Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

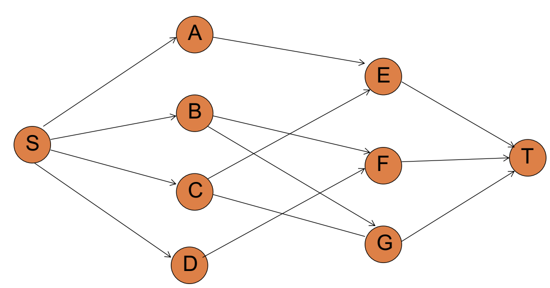


27

Application: bipartite graph matching

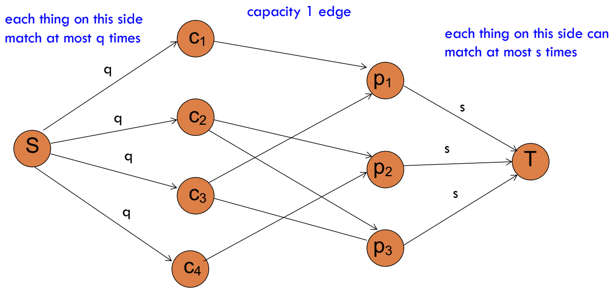
Setup as a flow problem:

all edge weights are 1



28

Matching generalized



29

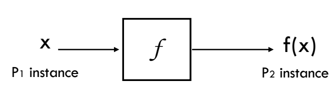
NP-Complete

A problem is NP-Complete if

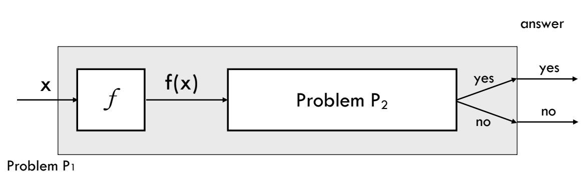
1. It is in NP (verifiable in polynomial time)
2. It is NP-Hard (there exists a polynomial-time reduction from all known NP-Hard problems)
   - (We can show this by showing a reduction from just one NP-Hard problem)

30

NP-Complete reduction proofs



Allow us to solve P_1 problems if we have a solver for P_2



31

How many slides this semester?

32