CSE 421 Section 2

Graph Traversal and Algorithm Proofs

Administrivia

Announcements & Reminders

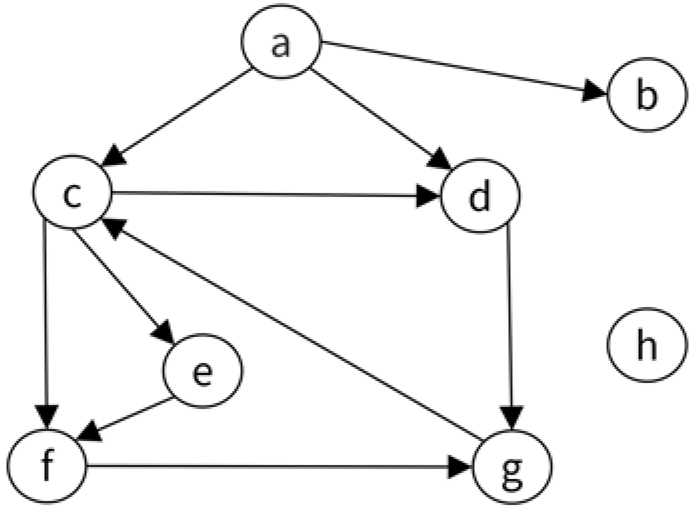
- **HW1**
  - Was due yesterday 4/9
  - Remember the **late days** policy
  - Total of up to **4 late problem days**
    - At most **1 late day per problem**
    - 25% penalty the next day, 50% if turned in on Saturday

- **HW2**
  - Due Wednesday 4/16 @ 11:59pm
  - Start early! Ask questions on Ed or Office Hours

Warmup: BFS/DFS

Problem 1 – BFS/DFS review

a) Run BFS and record the layer of each vertex, starting with a in layer 0.
b) Run DFS, record the start/end times, and classify the edges. When there are multiple choices for the next vertex, pick the alphabetically earliest one.

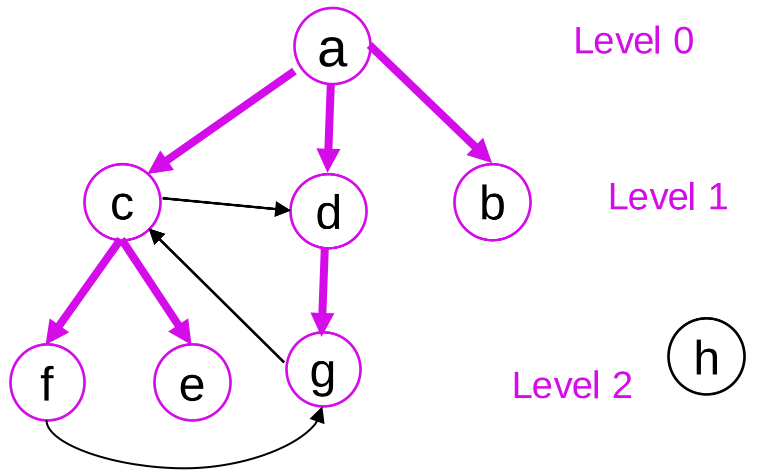


Feel free to work with the people around you!

Solutio
n

Problem 1 – BFS/DFS review

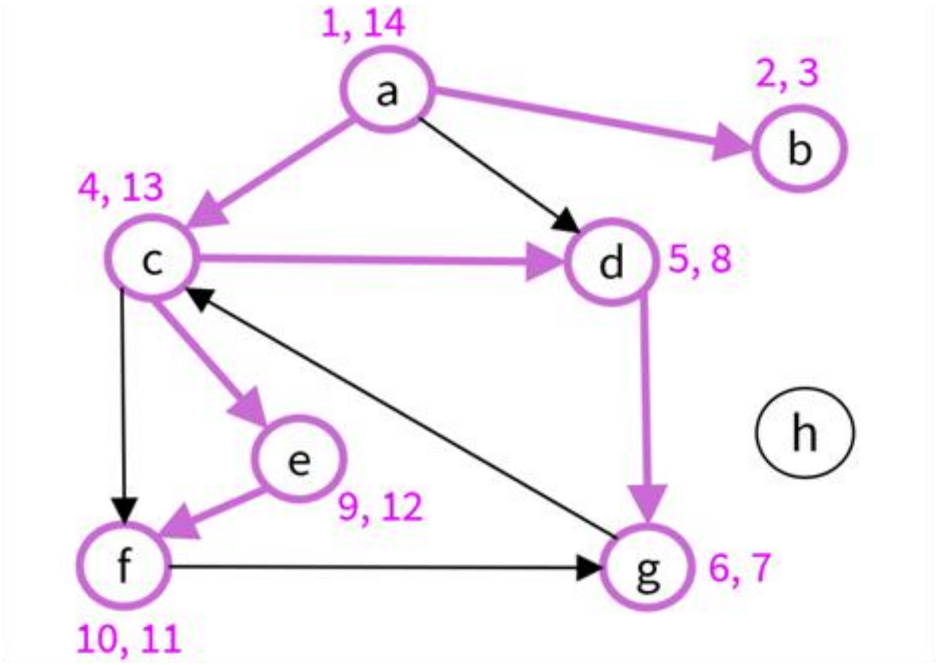
a) Run BFS and record the layer of each vertex, starting with a in layer 0.



Solutio
n

Problem 1 – BFS/DFS review

b) Run DFS, record the start/end times, and classify the edges. When there are multiple choices for the next vertex, pick the alphabetically earliest one.



Tree edges:

Back edges:

Forward edges:

Cross edges:

Proving algorithms correct

Problem 2 – Investigating algorithm proofs

The purpose of this problem is to help you:

- Figure out how to **start a proof** about algorithms.
- **Check the correctness** of proofs.

We will take a proof that you have already seen in lecture and critically analyze it to check correctness.

Problem 2 – Investigating algorithm proofs

Dive right in, and we'll summarize the takeaways afterwards.

a) Answer the questions embedded in the proof in your packet as you read. They are marked with ▷ and italics.

b) Discuss with people near you:
   - What is the general structure of a proof that an algorithm is correct?
   - How is the proof related to the pseudocode?

Feel free to work with
the people around you!

Solutio
n

Problem 2 – Investigating algorithm proofs

```
1: $R \leftarrow \{s\}$
2: while there is $\{u, v\} \in E$ with $u \in R$ and $v \notin R$ do
3:     Add $v$ to $R$.
4: return $R$
```

1. **Claim.** The algorithm terminates.
   a. ▷ *What quantity increases every iteration, but is bounded? Why does it increase? Refer to particular line(s) in the code.*

      $|R|$ increases every iteration, and is bounded by $|V|$.
      In every iteration, we add v to R (line 3) where $v \notin R$ (line 2),
      and never remove elements from R.

   a. ▷ *Conclude that there are a bounded number of iterations.*

      There can be at most $|V|$ iterations, by above.

Solutio
n

Problem 2 – Investigating algorithm proofs

```
1: $R \leftarrow \{s\}$
2: while there is $\{u, v\} \in E$ with $u \in R$ and $v \notin R$ do
3:     Add $v$ to $R$.
4: return $R$
```

2. **Claim.** At termination, $v \in R$ iff there exists a path from s to v.
   a. ($\Rightarrow$) **Claim.** If $v \in R$, then there exists a path from s to v.

      ▷ *Prove this fact. Refer to particular line(s) in the code.*

      By induction on iterations.
      BC: By (line 1), before the first iteration, $R = \{s\}$, and there is a path from s to s.
      IH: Before/after every iteration, if $v \in R$, then there exists a path from s to v.
      IS: Assume IH at start of this iteration.
      In current iteration, because $u \in R$ (line 2), there is a path from s to u by IH.
      Because $\{u, v\} \in E$ (line 2), we get a path from s to v.
      We only add v to R (line 3), so IH is true at end of this iteration.

Solutio
n

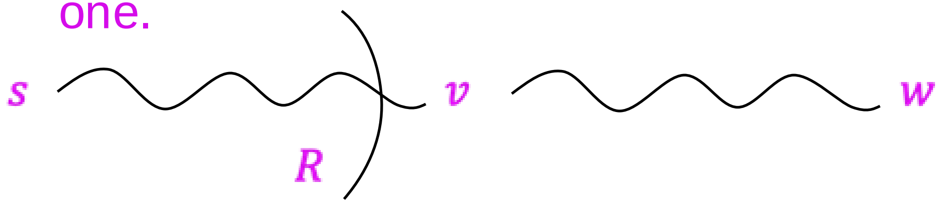
Problem 2 – Investigating algorithm proofs

```
1: $R \leftarrow \{s\}$
2: while there is $\{u, v\} \in E$ with $u \in R$ and $v \notin R$ do
3:     Add $v$ to $R$.
4: return $R$
```

- b. (⇐) **Claim.** If there exists a path from s to v, then $v \in R$.
  - i. Suppose for contradiction $w \notin R$, but there is a path from P from s to w.
  - ii. In this case, we may take v to be the first node on P such that $v \notin R$.

    ▷ *Why is this allowed?*

    If there are multiple of any object, we can always look at the first one.



Common technique!

Solution

Problem 2 – Investigating algorithm proofs

```
1: $R \leftarrow \{s\}$
2: while there is $\{u, v\} \in E$ with $u \in R$ and $v \notin R$ do
3:     Add $v$ to $R$.
4: return $R$
```

- iii. The predecessor u of v in P satisfies $u \in R$ and $\{u, v\} \in E$.

  ▷ *Why does the predecessor exist? Refer to particular line(s) in the code.*

  Why is $u \in R$ and $\{u, v\} \in E$?

  Predecessor exists because $v \neq s$, as $v \notin R$ but $s \in R$.

  To show $s \in R$ at loop end, formally induction on iterations:

  BC: $R = \{s\}$ before first iteration by (line 1).

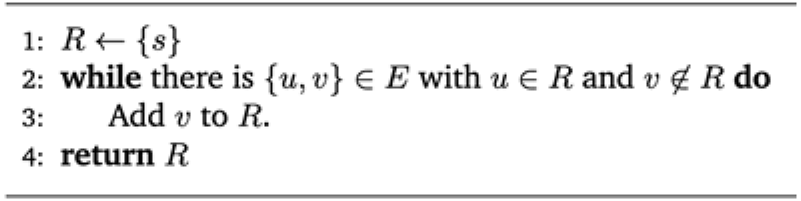
  IH: $s \in R$ before/after every iteration.

  IS: We never remove vertices from R, so IH is true at end of iteration.

  Lastly, $u \in R$ because v was first vertex not in R, and $\{u, v\} \in E$ because u is the predecessor of v.

Solutio
n

Problem 2 – Investigating algorithm proofs



```
1: $R \leftarrow \{s\}$
2: while there is $\{u, v\} \in E$ with $u \in R$ and $v \notin R$ do
3:     Add $v$ to $R$.
4: return $R$
```

iii. This is a contradiction.

▷ *What is the contradiction? Refer to particular line(s) in the code.*

We found u and v such that $u \in R$, $v \notin R$, and $\{u, v\} \in E$.
Contradicts loop exit condition in (line 2).

▷ *Is it possible to directly prove this claim by induction on iterations, as you did for the ⇒ direction, instead of contradiction?*

No, the claim "If there exists a path from s to v, then $v \in R$." is not true until the loop exits.

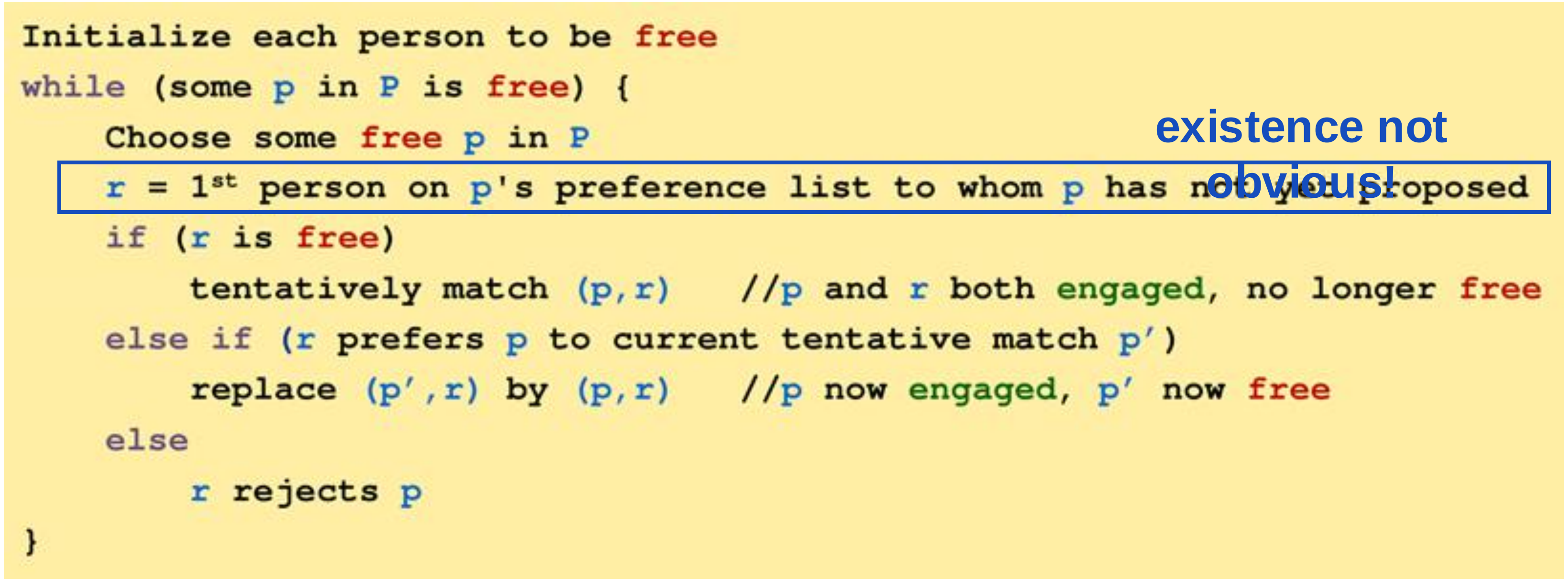
Problem 2 – Investigating algorithm proofs

The structure of an algorithm proof

- Validity:
  - Make sure that every line is possible, may require ad-hoc methods.
- Termination (if you have while-loops):
  - Find a **measure of progress** that increases/decreases every iteration, but is **bounded** by the problem setup.
- Correctness:
  - Start with “We will show that the output matches the desired result,” then unwrap definitions.

Problem 2 – Investigating algorithm proofs

Validity was obvious in our case, but not all algorithms.



```
Initialize each person to be free
while (some p in P is free) {
    Choose some free p in P
    r = 1st person on p's preference list to whom p has not yet proposed
    if (r is free)
        tentatively match (p,r)    //p and r both engaged, no longer free
    else if (r prefers p to current tentative match p′)
        replace (p′,r) by (p,r)    //p now engaged, p′ now free
    else
        r rejects p
}
```

existence not obvious!

Problem 2 – Investigating algorithm proofs

How to solve HW problems

1. Think of an algorithm (we'll give more tips for this next section)
2. Prove its **validity**, **termination**, and **correctness** as in the last slide.
    - Write the same level of detail as in sections/lectures.
3. Just for yourself, **expand the details like we did today** to check that it's correct.
    - If you can't, your TAs probably can't either (and will mark your proof as wrong or incomplete).
        - Go back to step 1 or step 2 and try again.

Remember, **if your algorithm is wrong, it's impossible to prove it to be correct!**

Applications of graph algorithms

Tips for algorithms with graphs

Ask yourself the following questions:

1. What are the vertices?
2. What are the edges?
3. What am I looking for in the graph?

Problem 3 – Judging books

You have a large collection of books and want to arrange them by color. You wish to put only books of a single color on any given shelf. Every pair of books is either "same color" or "not same color", and this relation is an equivalence relation (reflexive, symmetric, and transitive).

Input: A list of books, and a list of pairs that are the same color
Expected output: The best upper bound on the number of shelves you will need

Example input: books u, v, w, x, and pairs (u, v), (v, w)
Output: 2

Problem 3 – Judging books

Input: A list of books, and a list of pairs that are the same color
Expected output: The best upper bound on the number of shelves you will need

1. What are the vertices?
2. What are the edges?
3. What am I looking for in the graph?

Feel free to work with the people around you.

Solutio
n

Problem 3 – Judging books

Input: A list of books, and a list of pairs that are the same color

Expected output: The best upper bound on the number of shelves you will need

1. What are the vertices? book
s
2. What are the edges? pairs that are the same
color
3. What am I looking for in the graph? number of connected
components

Problem 3 – Judging books

You have a large collection of books and want to arrange them by color. You wish to put only books of a single color on any given shelf. Every pair of books is either “same color” or “not same color”, and this relation is an equivalence relation (reflexive, symmetric, and transitive).

Input: A list of books, and a list of pairs that are the same color
Expected output: The best upper bound on the number of shelves you will need

Now write the algorithm (two sentences) and think about the proof.
Feel free to work with the people around you.

Solutio
n

Problem 3 – Judging books

Algorithm:

1. Construct a graph $G = (V, E)$ where V is the list of books and E is the list of pairs.
2. Use BFS or DFS to compute the number of connected components and output it.

Proof:

- Termination is clear, and all lines are valid.
- We will show that the number of connected components is the best upper bound on the number of shelves needed.

Purely 311-type claim, no algorithm needed — can rely on knowledge that B/DFS computes # conn. comp.

1. Is an upper bound
2. All upper bounds are at least this big

Solutio
n

Problem 3 – Judging books

1. The number of connected components is an upper bound.
   a. Give each connected component its own shelf.
   b. Because "same color" is transitive, if there is a path from u to v, then u and v are the same color (formally by induction).
   c. Thus, it was valid to put them on the same shelf.
2. All upper bounds are at least the number of connected components.
   a. The input is compatible with the situation where every connected component has books of different colors.
   b. Thus, every upper bound must output at least the number of connected components.

Problem 4 – Water jugs

You have a 5-gallon jug and 3-gallon jug, which start out empty. Your goal is to have **4 gallons of water in the 5-gallon jug** and **0 gallons of water in the 3-gallon jug**. Unfortunately, you are only allowed the following operations:

- Fill any of your jugs completely.
- Pour one of your jugs into the other, until the first jug is empty or the second is full.
- Empty out all the water in a jug.

a) Describe a method to reach the goal. (No need to use any general algorithm yet, just solve the puzzle however you like.)

Feel free to work with the people around you.

Solutio
n

Problem 4 – Water jugs

a) Describe a method to reach the goal. (No need to use any general algorithm yet, just solve the puzzle however you like.)

Let (m, n) denote m gallons in the 5-gallon jug and n gallons in the 3-gallon jug.

(0, 0) → (5, 0) → (2, 3) → (2, 0) → (0, 2) → (5, 2) → (4, 3) → (4, 0)

OR

(0, 0) → (0, 3) → (3, 0) → (3, 3) → (5, 1) → (0, 1) → (1, 0) → (1, 3) → (4, 0)

or other solutions.

Problem 4 – Water jugs

b) Solve with a graph algorithm and state the running time (no proof during section):

Input: Jug sizes a and b, with target amounts x and y, respectively.
Expected output: The minimum number of steps to reach the target amount, or "unreachable".

1. What are the vertices?

2. What are the edges?

3. What am I looking for in the graph?

Feel free to work with the people around you.

Solutio
n

Problem 4 – Water jugs

b) Solve with a graph algorithm and state the running time (no proof during section):

Input: Jug sizes a and b, with target amounts x and y, respectively.
Expected output: The minimum number of steps to reach the target amount, or "unreachable".

1. What are the vertices? pairs (m, n) of the current amounts in each jug
2. What are the edges? all allowed steps in the problem
3. What am I looking for in the graph? length of shortest path $(0, 0)$ to (x, y) or "unreachable"

Solutio
n

Problem 4 – Water jugs

Algorithm:

1. Construct a graph $G = (V, E)$ where V is a list of all pairs (m, n) for $0 \leq m \leq a$ and $0 \leq n \leq b$, and E is all possible transitions according to the rules.
2. Use BFS to compute the shortest path from $(0, 0)$ to (x, y) and output it, or "unreachable" if BFS terminates before reaching (x, y).

Running time:

BFS runs in $\Theta(|V| + |E|)$, where $|V| = (a + 1)(b + 1)$ and $|E| \leq 6|V|$ (since there are 3 types of transitions from every state, which you can do with either jug). Thus, $\Theta(ab)$.

Extra: Can you tweak this solution to be a bit faster?

Solutio
n

Problem 4 – Water jugs

Slight improvement!
Notice that all transitions result in one of the jugs being full or empty.

Algorithm:

1. Construct a graph $G = (V, E)$ where V is a list of all pairs $(0, n), (a, n), (m, 0)$, (m, b) for $0 \leq m \leq a$ and $0 \leq n \leq b$, and E is all possible transitions according to the rules.
2. Use BFS to compute the shortest path from $(0, 0)$ to (x, y) and output it, or "unreachable" if BFS terminates before reaching (x, y) or (x, y) is not in the graph.

Running time:

BFS runs in $\Theta(|V| + |E|)$, where $|V| = 2(a + 1) + 2(b + 1)$ and $|E| \leq 6|V|$ (since there are 3 types of transitions from every state, which you can do with either jug).
Thus, $\Theta(a + b)$.

Last note on applying graph algorithms

In section today, we saw two examples of **directly calling a graph algorithm**, as if it were a library function.

Sometimes, in other problems, you might need to **reimplement a graph algorithm** and tweak a line or two to solve the problem.

Summary

- For algorithms, prove **validity, termination, and correctness**.
  - Make sure that you can expand all details in your head.
  - Find a **measure of progress** to prove termination.
  - Start with "We will show that the output matches the desired result," then expand definitions, use **observations that are true in every iteration**, and use
  - Model problems using graphs, then apply or tweak algorithms like BFS/DFS/etc.

Thanks for coming to section this week!