Final Review

10-607

Outline

- Propositional Logic
- Proof Techniques
    - Proof by Cases
    - Proof by Contradiction
    - Proof by Induction
- Computational Complexity
- Recursion
- Data Structures
    - Stacks and Queues
    - Trees
    - Graphs
- Algorithms
    - Perceptron
    - Variable Elimination

Propositional Logic

- Some important things to know:
  - Properties from the hand out
    - Distributivity: $a \vee (b \wedge c)$ is equivalent to $(a \vee b) \wedge (a \vee c)$
    - Associativity
    - Etc…
  - Know how to make truth tables show if statements are equivalent or not
  - Contrapositive

$$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$$

  - DeMorgan’s Law

$$\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$$

Any Logic Questions?

Proof Techniques

- Big ones we covered in class
  - Proof by cases
    - When there are a few cases, prove the statement under the assumption that each one is true.
    - This is a good choice in strategy when you can think of a few different cases that could happen in the statement.
  - Proof by contrapositive
    - Formulate the contrapositive of the statement and prove that instead.
  - Proof by contradiction
    - Assume that the opposite statement holds, then show there is a logical contradiction.
    - Know the difference between disproof by counter example, proof by contrapositive, and proof by contradiction! They are all very different!
  - Proof by induction
    - Prove the base case, assume true for $n = k$, show it is true for $n = k + 1$.
    - This is a good choice for statements of the form "$A(n)$ is true for $n >= n\_0$"

Proof Techniques (Disprove by Counterexample)

Which of the following can be disproved by counterexample?

a) Let a, b be odd integers. Then a + b is odd.

b) There exists odd integers a and b such that a + b is odd.

c) Both a) and b)

d) None of the above

Proof Techniques (Disprove by Counterexample)

Which of the following can be disproved by counterexample?

a) Let a, b be odd integers. Then a + b is odd.

The first statement implies that it is true for all possible odd integers, so a counterexample shows this is not true. A counterexample will not prove the existence of such odd integers for b)

b) There exists odd integers a and b such that a + b is odd.

c) Both a) and b)

d) None of the above

Proof Techniques (Proof by Contrapositive)

What is the contrapositive statement of : *If it is cloudy then it is raining.*

a) If it is not cloudy then it is not raining.

b) If it is not raining then it is not cloudy.

c) If it is raining then it is cloudy.

d) None of the above.

Proof Techniques (Proof by Contrapositive)

What is the contrapositive statement of : *If it is cloudy then it is raining.*

a) If it is not cloudy then it is not raining.

b) If it is not raining then it is not cloudy.

c) If it is raining then it is cloudy.

d) None of the above.

Proof Techniques (Proof by Contrapositive)

What is the contrapositive statement of : *If there exists an x such that f(x) = 3, then g(y) = 2 for all y*

a) If there exists an x such that f(x) ≠ 3, then g(y) ≠ 2 for all y.

b) If, for all x, f(x) ≠ 3, then there exists a y such that g(y) ≠ 2.

c) If, for all y, g(y) ≠ 2, then there exists an x such that g(x) ≠ 3.

d) If there exists a y such that g(y) ≠ 2, then f(x) ≠ 3 for all x.

Proof Techniques (Proof by Contrapositive)

What is the contrapositive statement of : *If there exists an x such that f(x) = 3, then g(y) = 2 for all y*

a) If there exists an x such that f(x) ≠ 3, then g(y) ≠ 2 for all y.

b) If, for all x, f(x) ≠ 3, then there exists a y such that g(y) ≠ 2.

c) If, for all y, g(y) ≠ 2, then there exists an x such that g(x) ≠ 3.

d) If there exists a y such that g(y) ≠ 2, then f(x) ≠ 3 for all x.

Note that usually “for all” turns into “there exists” and vice versa when negation is applied.

Proof Techniques (The Recipe for Induction)

- This recipe can be followed exactly.

Prove that A(n) is true for all n ≥ n'

<u>Base Case</u>
We first prove that A(n) is true for when n = n'
YOUR PROOF HERE

<u>Inductive Assumption</u>
Let n=k and assume that A(k) is true.

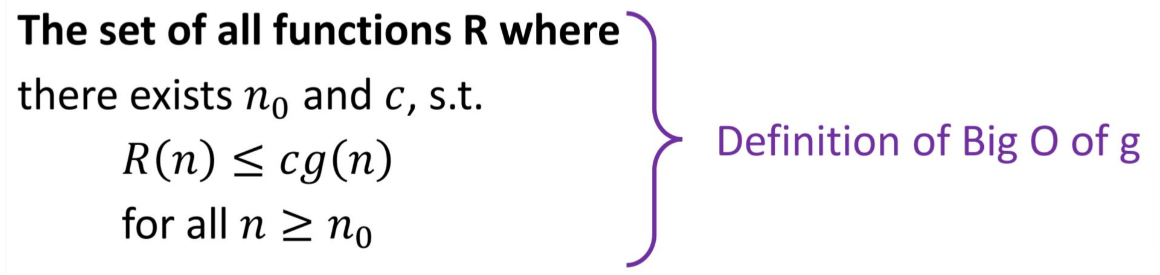
<u>Inductive Step</u>
We now prove that, where n = k + 1, that A(k + 1) is true.
YOUR PROOF HERE

Thus the statement holds true by induction.

Any Proof Questions?

Computational Complexity

- Know the definition of big-O:



The set of all functions R where
there exists n_0 and c, s.t.
$$R(n) \leq cg(n)$$
for all $n \geq n_0$

Definition of Big O of g

- Be able to analyze code complexity
  - Tips:
    - Try writing the number of operations in terms of math with sum notation.
    - Count the number of for loops/recursive calls this will often (but not always) correspond to the order of a polynomial. E.g. two for loops often means O(n^2)

- Be able to count number of operations

Computational Complexity

- Which of the following are true?

a) $n^2 + 2n + 3 \in \mathcal{O}(n^2)$

b) $4 + 3 \in \mathcal{O}(n^3)$

c) $\sum_{i=1}^{n} i \in \mathcal{O}(n^2)$

d) $\log\left(n^3\right) \in \mathcal{O}\left(\log\left(n^2\right)\right)$

e) $2^{2n} \in \mathcal{O}\left(2^n\right)$

f) $\sqrt{n} \in \mathcal{O}\left(\log(n)\right)$

Computational Complexity

- Which of the following are true?
  - [✓ a) $n^2 + 2n + 3 \in \mathcal{O}(n^2)$]
  - [✓ b) $4 + 3 \in \mathcal{O}(n^3)$]
  - [✓ c) $\sum_{i=1}^{n} i \in \mathcal{O}(n^2)$]
  - [✓ d) $\log\left(n^3\right) \in \mathcal{O}\left(\log\left(n^2\right)\right)$]

    Remember with logarithms you can take exponent out as coefficients.
  - e) $2^{2n} \in \mathcal{O}\left(2^n\right)$

    2^{2n} is the same as 2^n squared.
  - f) $\sqrt{n} \in \mathcal{O}\left(\log(n)\right)$

    Logarithm grows slower than any polynomial

Any Complexity Questions?

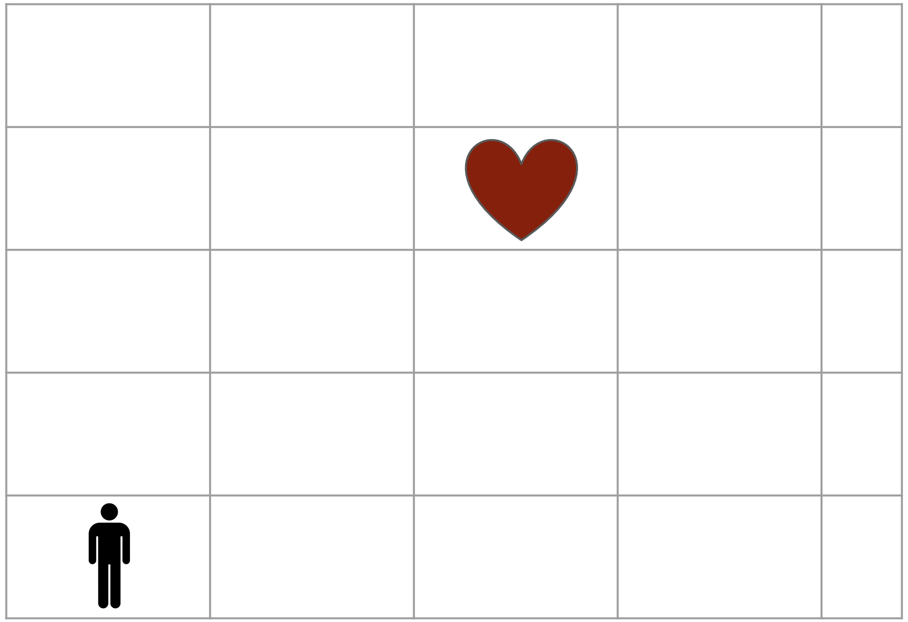
Recursion

- Strategy for writing recursive functions:
  - Identify what your base case is
  - Usually when you recursively call the function, the arguments that you pass are for a smaller/easier problem. How are you making the problem easier?

- Other things to know
  - Memoization. This is something that should be added onto the regular recursive approach.

Recursion: A Practice Problem

Suppose you are in a grid environment where you can either move right or up. How many ways are there to get to the heart at some arbitrary (x, y) coordinate (assume x >= 0 and y >= 0).



From “Cracking the Coding Interview”

Recursion: A Practice Problem

- What is a good base case for this problem?
- How would you break the problem down to make it easier?
- How could you memoize your recursive solution?

Recursion: A Practice Problem

- What is a good base case for this problem?

  When you are on top of the goal. Depending on your solution also if you have moved past the goal.

- How would you break the problem down to make it easier?

  After you go right or up you are left with the same problem but with a close goal target. E.g. if you go one step up now it is the same problem but the goal is at (x, y - 1)

- How could you memoize your recursive solution?

  Keep a dictionary mapping (x, y) coordinate of the goal to the number of ways to get there. Look this up every call of the function to see if you already have the answer.

Solution at end of slides

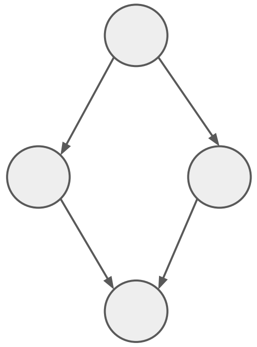
Any Recursion Questions?

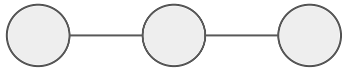
Data Structures

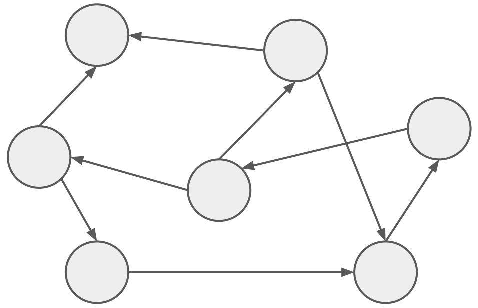
- What to know:
  - Stacks, queues, trees, graphs definition and the ways that they can be implemented
    - For graph: adjacency matrix vs using a class-based implementation.
  - BFS and DFS
    - What the order things will be expanded, what data structures to use.
  - Properties of graphs
    - Directed vs undirected: Whether there is an arrow on the edges. What does each mean for how we implement a graph?
    - acyclic vs cyclic: acyclic means that there are no loops
    - Tree vs graph (maybe wasn’t covered?): Tree is a DAG where each node can only have one parent.

Data Structures (Graphs)

- Which graphs are DAGs?

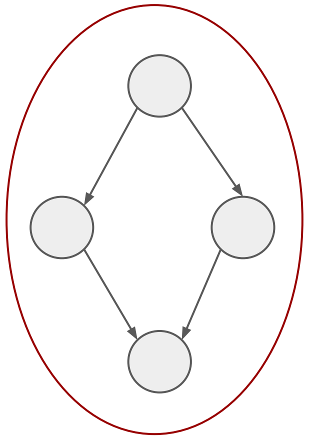

(a)

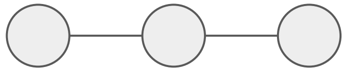

(b)

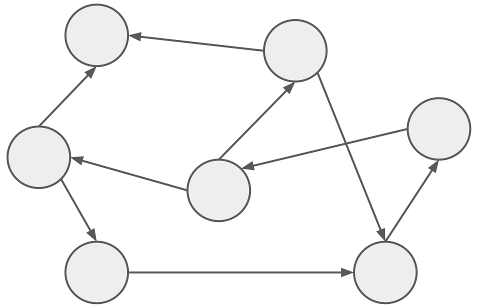

(c)

Data Structures (Graphs)

- Which graphs are DAGs?


(a)


(b)


(c)

Any Data Structures Questions?

Algorithms (Bayes Nets)

- Things to know
  - From a picture of a Bayes net, be able to tell what probability tables are being stored.
  - Know how to get the joint probability from those probability tables.
  - Compute the number of entries of a table/how many operations are needed.
  - Basic idea of Variable Elimination

- Might be good to brush up on basic probability. In particular
  - The Chain Rule for Probability
  - Marginalization

Algorithms (Bayes Nets: Variable Elimination)

How can we rearrange the terms in the following expression to use as few operations as possible?

$$\sum_d \sum_a \sum_c P(A=a|D=d)P(D=d|B=b)P(B=b)P(C=c|B=b)$$

Algorithms (Bayes Nets: Variable Elimination)

How can we rearrange the terms in the following expression to use as few operations as possible?

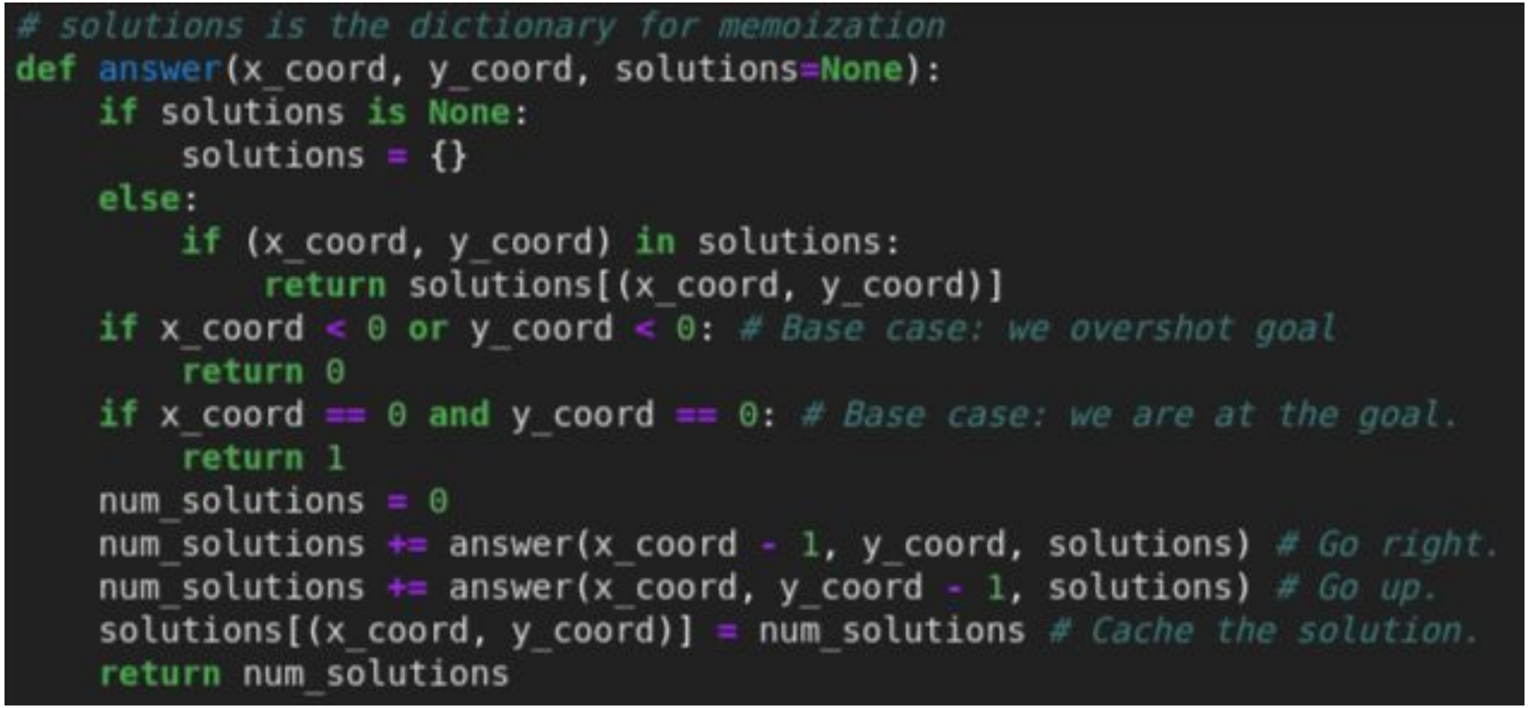
$$\sum_d \sum_a \sum_c P(A = a|D = d)P(D = d|B = b)P(B = b)P(C = c|B = b)$$

$$P(B = b) \sum_d P(D = d|B = b) \sum_a P(A = a|D = d) \sum_c P(C = c|B = b)$$

Perceptron Algorithm

- Things to know
  - Be able to do the perceptron algorithm by hand
  - Understand definition of margin

Questions?



```python
# solutions is the dictionary for memoization
def answer(x_coord, y_coord, solutions=None):
    if solutions is None:
        solutions = {}
    else:
        if (x_coord, y_coord) in solutions:
            return solutions[(x_coord, y_coord)]
    if x_coord < 0 or y_coord < 0: # Base case: we overshot goal
        return 0
    if x_coord == 0 and y_coord == 0: # Base case: we are at the goal.
        return 1
    num_solutions = 0
    num_solutions += answer(x_coord - 1, y_coord, solutions) # Go right.
    num_solutions += answer(x_coord, y_coord - 1, solutions) # Go up.
    solutions[(x_coord, y_coord)] = num_solutions # Cache the solution.
    return num_solutions
```