Lecture 26
Final Exam Review

EECS 281: Data Structures & Algorithms

When/Where

- When: Monday December 11
  - 8:00am – 10:00am
- Locations are on Piazza
  - You will be assigned a room based on your uniqname
- Accommodation (extended time) starts at the same time and ends later
  - 3hr for 1.5x time
  - 4hr for 2x time

Policies

- Closed book and closed notes
- One “cheat sheet”, limited to 8.5"x11", (both sides), with your name on it
  - Writing it by hand will make you much better prepared
- No calculators or electronics of any kind
- Engineering Honor Code applies

Don't forget!

Bring your Mcard with you!

The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2023

FINAL EXAM
KEY 1
Monday, December 11, 2023
8:00AM – 10:00AM

Uniqname: _____ **Student ID:** _____

Name: _____

Uniqname of person to your left: _____

Uniqname of person to your right: _____

Honor Pledge:
"I have neither given nor received unauthorized aid on this examination,
nor have I concealed any violations of the Honor Code."

Signature: _____

Record your NAME, Uniqname and Student ID# LEGIBLY!

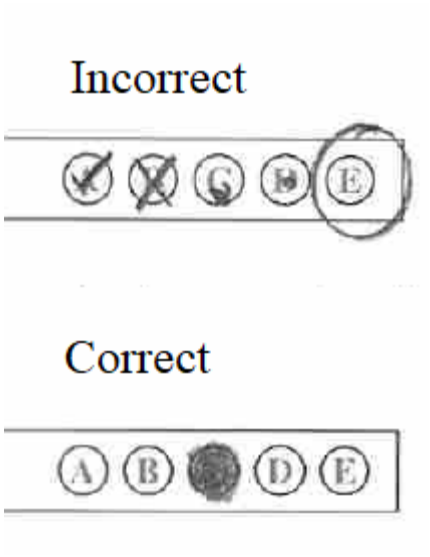
SIGN THE HONOR PLEDGE!
We won't post a final grade without it!

Multiple Choice Portion

- 24 questions, 2.5 points each
- 4-5 possible answers per question
- No deduction for being wrong
  - Make sure to answer all 24 questions
  - **ONE** answer per question
- **DO NOT wait until after time is called to <u>BUBBLE</u> in your answers**
  - **Do not just circle the letters to the left**

Filling in Bubbles

- Added to the instructions on the practice exam and actual exam
- Do NOT circle the letters next to the answers



NOTE

- Bring a #2 pencil, or #2 lead for mechanical pencils (also listed as “HB”)
  - #3 pencils are too hard, and don’t scan well
- Odd-numbered pages have room at the top to write your uniqname
  - This is a backup in case pages become separated between collecting and scanning

Answering Coding Questions

- If you decide you want a helper function, write it below the “given” function
- If you need a structure, write that inside the “given” function, below it, on the right, etc.
  - Some coding problems given in some semesters can ONLY be solved if you create a structure (or use a `pair<>`)
- Make it legible

Coding Questions – Lines

- How many lines of code is this?

```
if (x > 0) result = 0;
```

- 2 lines of code, same as this:

```
if (x > 0)
    result = 0;
```

Coding Questions – Lines

- How many lines of code is this?

```
if (x > 0) {
    result = 0;
    return result;
} // if
```

- 3 lines of code: the closing curly brace never counts as a “line of code”

Coding Questions – Lines

- How many lines of code is this?

```
if (x > 0)
    result = 0;
else
    result = x;
```

- 4 lines of code, the `else` statement counts as a line
- One line with ternary operator:

```
result = (x > 0) ? 0 : x;
```

Coding – Container of `struct`

- Once you create a structure, how can you easily add a member of that structure to a container? (OBTW: line count = 5)

```
struct WordCount {
    string word;
    int count;
};

vector<WordCount> vwc;
vwc.push_back({ "abc", 1});
```

Coding Questions – Libraries

- Each coding question will tell you what you can or cannot use from the C and STL libraries
- The function header that we give you will not be part of the line limit
  - If you add a struct or helper function, those lines will count
  - If this is a reasonable way to solve the question, it is already factored in the line limit

Coding Questions – Integers

- For loop variables, use whatever type makes sense (`size_t`, `int`, etc.)
  - You don't have to worry about implicit conversions on loop variables
- If we pass a `vector<int>` to your function and you need to keep a copy of one or more of those values, use an `int` variable, or a container of `int`
  - Stay consistent with data

Study Materials

- Two practice exams posted on Canvas
- Lecture slides and recordings
- In-class exercises
- Laboratory materials
- Study group
- Another idea: programming interview questions

Topics

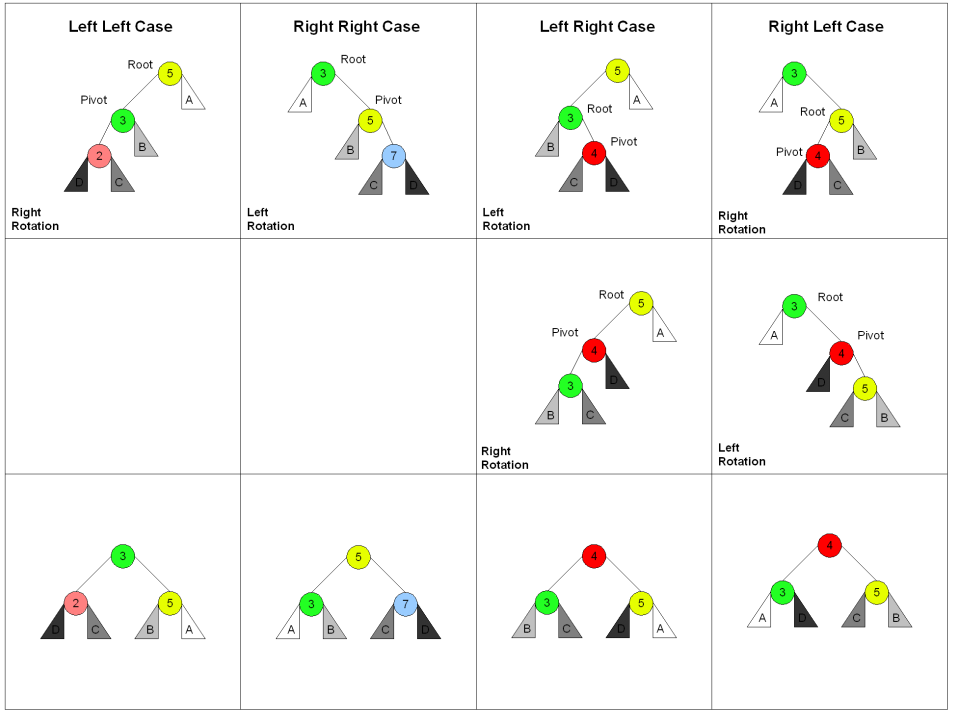
- Cumulative, but we will focus on material that has not yet been tested:
  - Trees (general, binary, search, AVL)
  - Hashing (and collision resolution)
  - Graphs and graph algorithms
  - Algorithm Families, DP, Knapsack
  - Backtracking, Branch and Bound
  - Shortest Paths, but NOT Floyd
  - NOT Strings and Sequences
  - NOT Computational Geometry

Trees

- General trees
- Binary trees
- BSTs
- AVL trees

Exercise

- Insert these keys into an AVL tree, rebalancing when necessary
- 4, 2, 3, 5, 7, 8, 9, 13, 12, 10



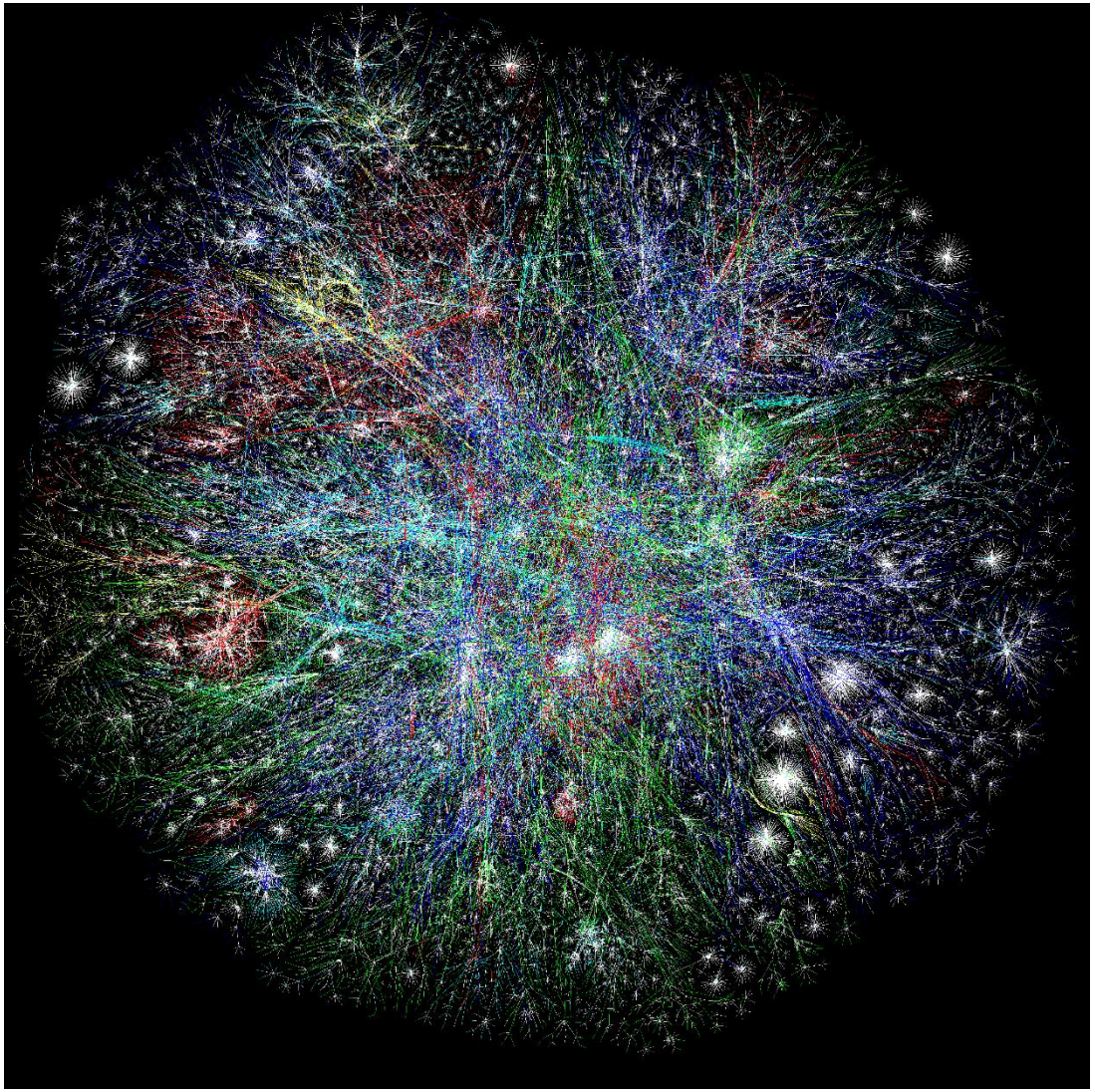
Hashing

- Hash functions
- Hash tables
- Collision resolution
  - Separate Chaining
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

Hashing

- What is the complexity of inserting into a hash table with *m* slots that stores *n* **unique** keys?
- Separate chaining
  - Best, worst, average
- Linear probing
  - Best, worst, average

Graphs



Graphs

- Graphs in general
  - Terminology and types
  - Adjacency matrix vs. adjacency list
- Search
  - DFS, BFS
- MST
  - Prim, Kruskal
- Shortest path
  - Dijkstra~~, Floyd-Warshall~~

Graph Terminology

- edge
- vertex/node
- weight
- directed / undirected
- simple graph (no loops or parallel edges)
- adjacency matrix
- adjacency list

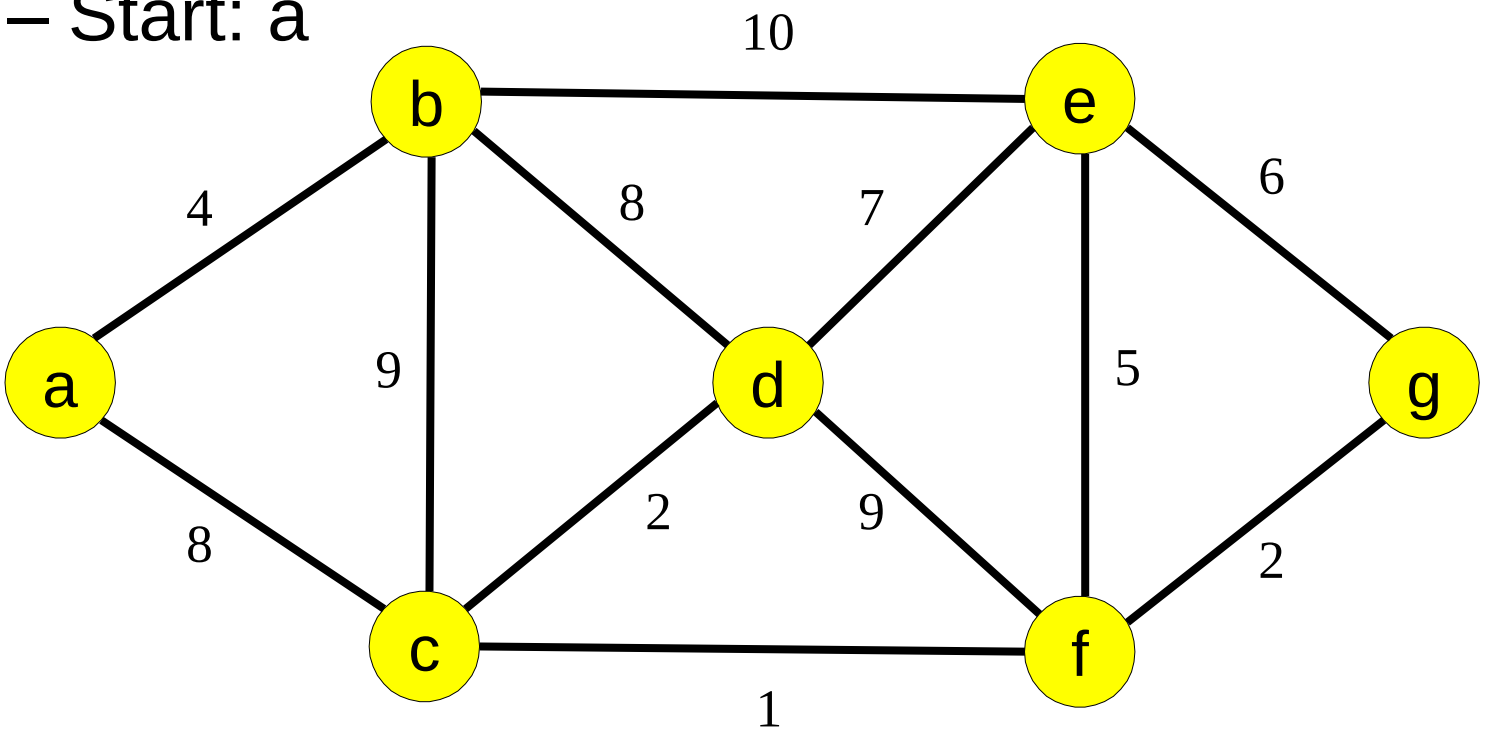
- sparse graph
  - |E| << |V|^2, adjacency list
- dense graph
  - |E| ~= |V|^2, adjacency matrix
- simple path
- connected graph
- cycle
- spanning tree / MST
- Hamiltonian cycle

Graph Problems

- Traveling salesperson
- Graph coloring
- Knapsack
- N-Queens

MSTs

- Do Prim's and Kruskal's on this graph
  - Start: a



Prim's Algorithm

Repeat until every k_v is true:

1. **From the set of vertices for which k_v is false, select the vertex v having the smallest tentative distance d_v**
2. **Set k_v to true**
3. **For each vertex w adjacent to v for which k_w is false, test whether d_w is greater than d_v. If it is, set d_w to d_v and set p_w to v.**

- $O(V^2)$ or $O(E \log V)$ with heaps

Kruskal's Algorithm

- Greedy MST algorithm for edge-weighted, connected, *undirected* graph
  - Presort all edges: $O(E \log E) = O(E \log V)$ time
  - Try inserting in order of increasing weight
  - Some edges will be discarded so as not to create cycles
- Initial two edges may be disjoint
  - We are growing a forest (union of disjoint trees)

Dijkstra's Algorithm

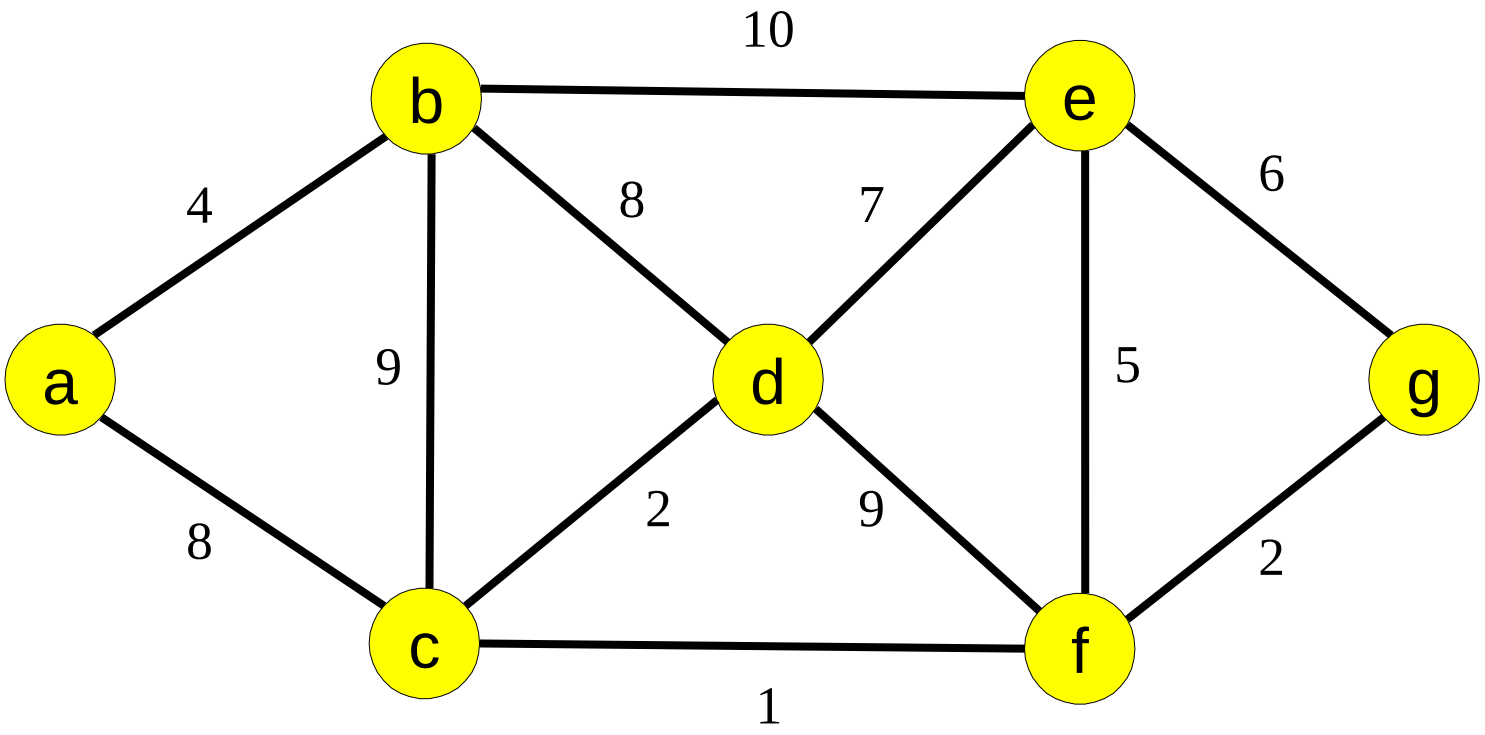
- Greedy algorithm for solving shortest path problem
- Assume non-negative weights
- Find shortest path from v_s to each other vertex

Dijkstra's Algorithm

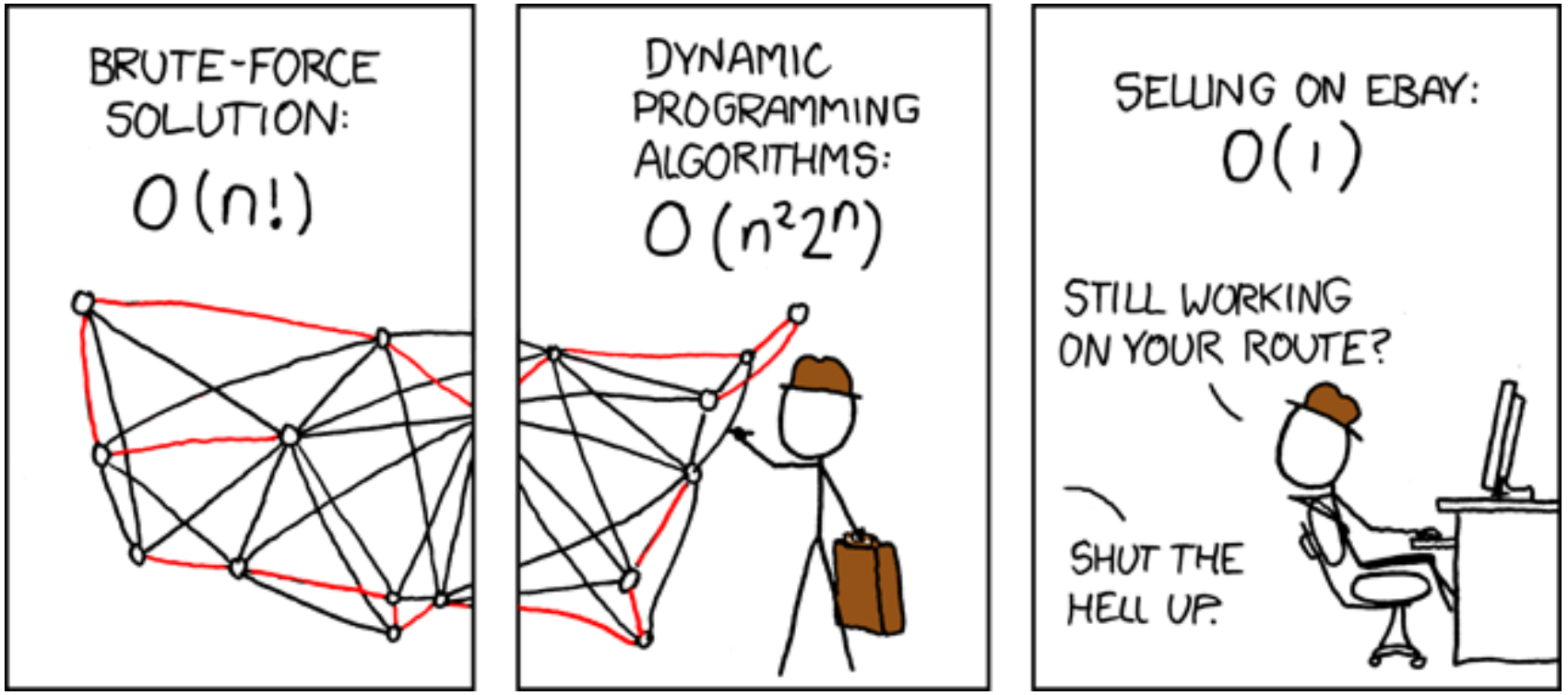
- For each vertex v, need to know:
  - k_v: Is the shortest path from v_s to v known? (initially false for all $v \in V$)
  - d_v: What is the length of the shortest path from v_s to v? (initially ∞ for all $v \in V$, except $v_s = 0$)
  - p_v: What vertex precedes (is parent of) v on the shortest path from v_s to v? (initially unknown for all $v \in V$)
- $O(V^2)$, or $O(E \log V)$ with heaps

Shortest Path

- Do Dijkstra's algorithm starting at node a



Algorithm Families



Algorithm Families

- Brute-Force
- Greedy
- Divide-and-Conquer
- Dynamic Programming
- Backtracking
- Branch-and-Bound

Algorithm Families

- Brute force
  - “Simple” straight-forward way
  - Usually inefficient
- Greedy
  - Decisions never reconsidered
  - Are never optimal?

Algorithm Families

- Divide-and-Conquer
  - Divide in 2 or more sub-problems
  - Non-overlapping sub-problems
  - Usually recursive
- Dynamic Programming
  - Divide into multiple overlapping sub-problems
  - Remember partial solutions and reuse
  - “Memoization”

Algorithm Families

- Backtracking
  - Systematically check all possible solutions
  - Prune those that don't satisfy constraints
  - Stop when constraints are satisfied
- Branch-and-Bound
  - For optimization problems, e.g., best solution
  - Can't stop early

Dynamic Programming

- Write two versions of this function, using top-down and bottom-up DP
- What family best describes the implementation below?

```
1  // returns the nth Fibonacci number
2  uint64_t fib(unsigned int n) {
3      if (n <= 1)
4          return n;
5      return fib(n - 1) + fib(n - 2);
6  } // fib()
```

Dynamic Programming: How To

- Look at the coding question problem titles, one will say “Dynamic Programming”
- Look at the memory requirements
  - What does this tell you about the memo?
  - Figure out what the memo looks like
  - Figure out the relationship between memo items
- Look at the time requirements
  - What does this tell you about loops?

What You Have Learned

- At the beginning of the semester
  - You mean we *don't get starter files?!!!*
- At the end of the semester
  - Design, implement, test, debug and optimize your own complex programs
  - Mathematically analyze code complexity and compare different design choices
- You've come a long way!

What's Next

- (Almost) all the upper level CS/MDE courses!
  - 367 Introduction to Autonomous Robotics, 388 Introduction to Computer Security, 390 Programming Paradigms, 440 Design of a Search Engine, 441 Mobile App Development for Entrepreneurs, 442 Computer Vision, 445 Introduction to Machine Learning, 449 Conversational AI, 471 Applied Parallel Programming with GPUs, 475 Introduction to Cryptography, 476 Data Mining, 477 Introduction to Algorithms, 480 Social Computing Systems, 481 Software Engineering, 482 Introduction to Operating Systems, 483 Compiler Construction, 484 Database Management Systems, 485 Web Systems, 486 Information Retrieval and Web Search, 487 Introduction to NLP, 489 Computer Networks, 490 Programming Languages, 491 Introduction to Distributed Systems, 492 Introduction to Artificial Intelligence, 493 User Interface Development, 494 Computer Game Design and Development, 495 Software Development for Accessibility, 497 Human-Centered Software Design and Development
- Learning on your own
  - Evaluate libraries written by others (e.g., Boost) and decide when to borrow, and when to create your own

Thank you!