Lecture 25
Final Exam Review

EECS 281: Data Structures & Algorithms

Time and Location

- When: Friday 18 December, 8:00am (sharp) - 10:00am (120min)
- Accommodation (extended time) exam starts at 8:00am
- Location: check CTools announcements, rooms assigned by uniqname

Policies

- Closed book and closed notes
- One “cheat sheet”, limited to 8.5"x11", double-sided, hand-written by you, with your name on it
- No calculators or electronics of any kind
- Engineering Honor Code applies

Don't forget!

The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2015

FINAL EXAM
Multiple-Choice Portion, KEY 1
Friday December 18, 2015
8:00AM – 10:00AM (120 minutes)

Record your NAME, Student ID# and Exam KEY # on the Scantron form. There will be a penalty for incorrectly filling out the form.

Name: ____________________ **Uniqname**: ____________________

Student ID: ______________________________________

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! on the written portion

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

Study Materials

- Practice questions posted on CTools
- Lecture slides and recordings
- In-class exercises
- Discussion materials
- Study group
- Another idea: programming interview questions

Topics

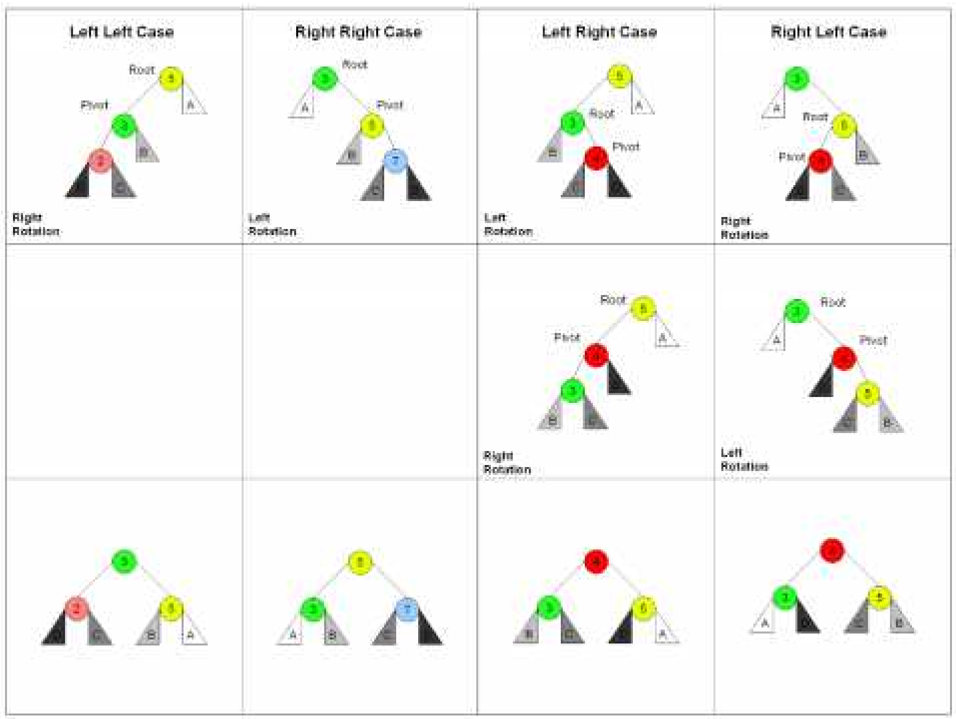
- Cumulative, but we will focus on material that has not yet been tested
- Trees (general, binary, search, AVL)
- Hashing (and collision resolution)
- Graphs and graph algorithms
- Algorithm Families

Trees

- General trees
- Binary trees
- BSTs
- AVL trees

Exercise

- Insert these keys into an AVL tree, rebalancing when necessary
- 4, 2, -1, 5, 7, 8, 9, 13, 12, 10



Exercise

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

Hashing

- Separate chaining
  - Best, worst, average
  - $O(1)$, $O(n)$, $O(n/m) \approx O(1)$
- Linear probing
  - Best, worst, average
  - $O(1)$, $O(n)$, $O(n/m) \approx O(1)$

- Same as insert

Graphs

Graphs

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

Graph Terminology

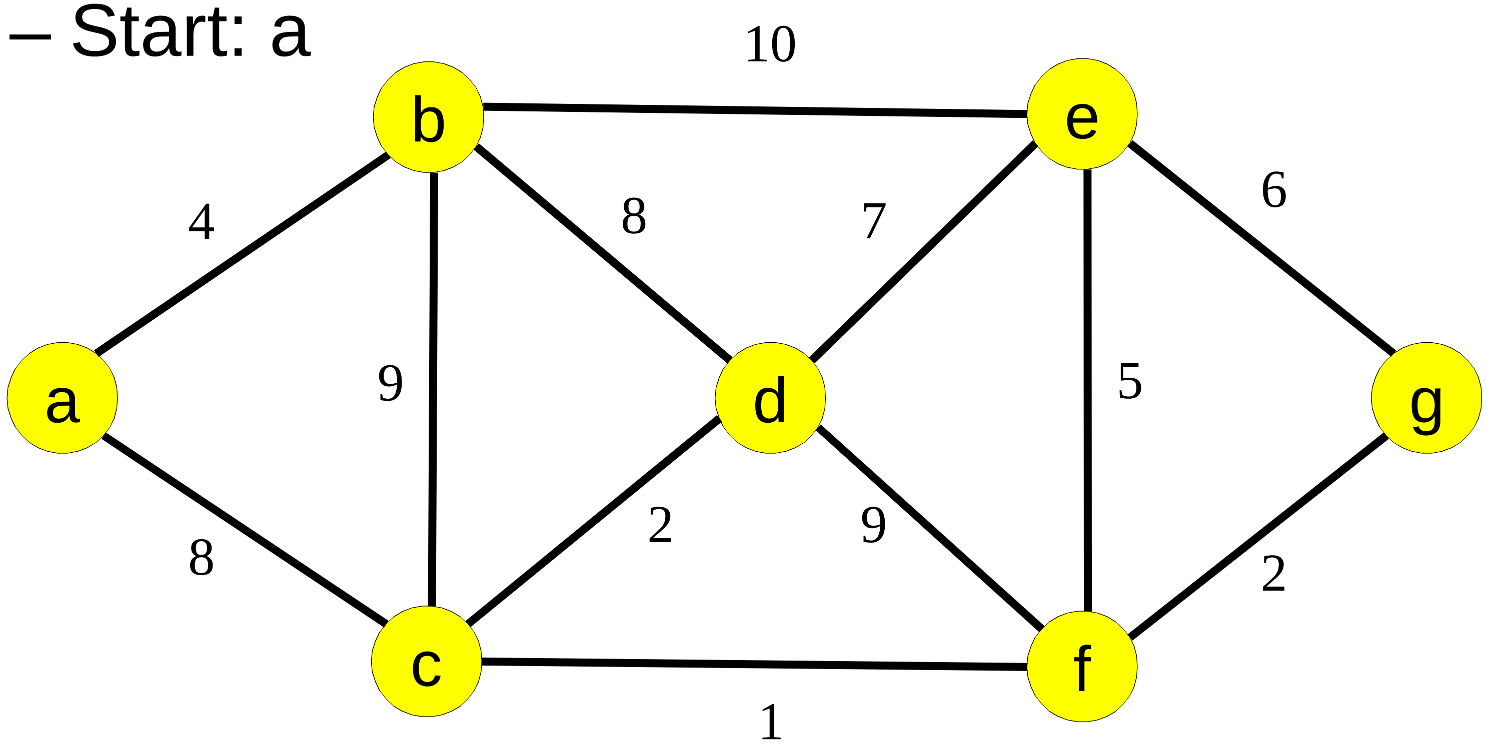
- edge
- vertex/node
- degree
- weight
- directed / undirected
- simple graph (no loops or parallel edges)
- adjacency matrix
- adjacency list
- sparse graph
  - |E| << |V|^2, adjacency list
- dense graph
  - |E| ~= |V|^2, adjacency matrix
- simple path
- connected graph
- cycle
- spanning tree / MST
- Hamiltonian cycle

Graph Problems

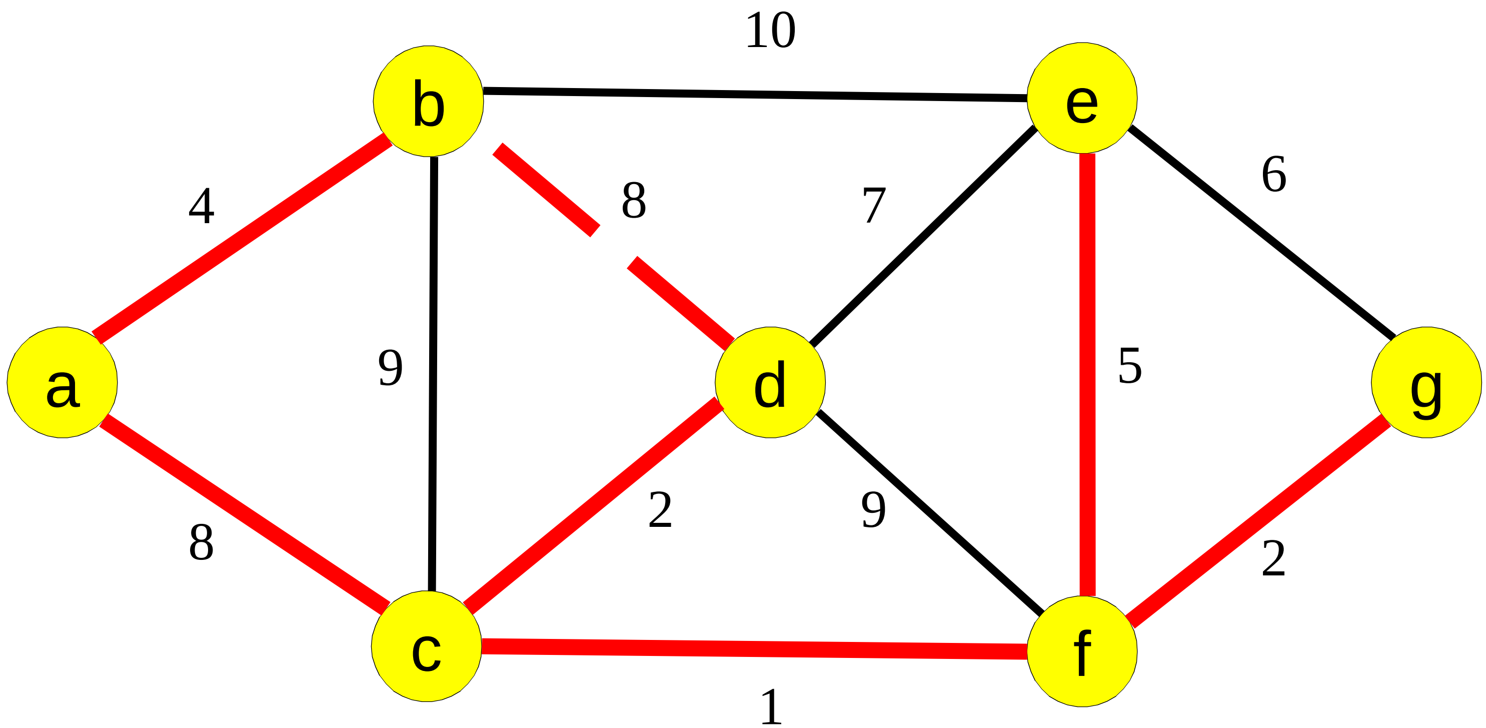
- Traveling salesman
- Graph coloring
- Knapsack
- N-Queens

MSTs

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



MSTs



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 <u>forest</u> (union of disjoint trees)

Dijkstra’s Algorithm

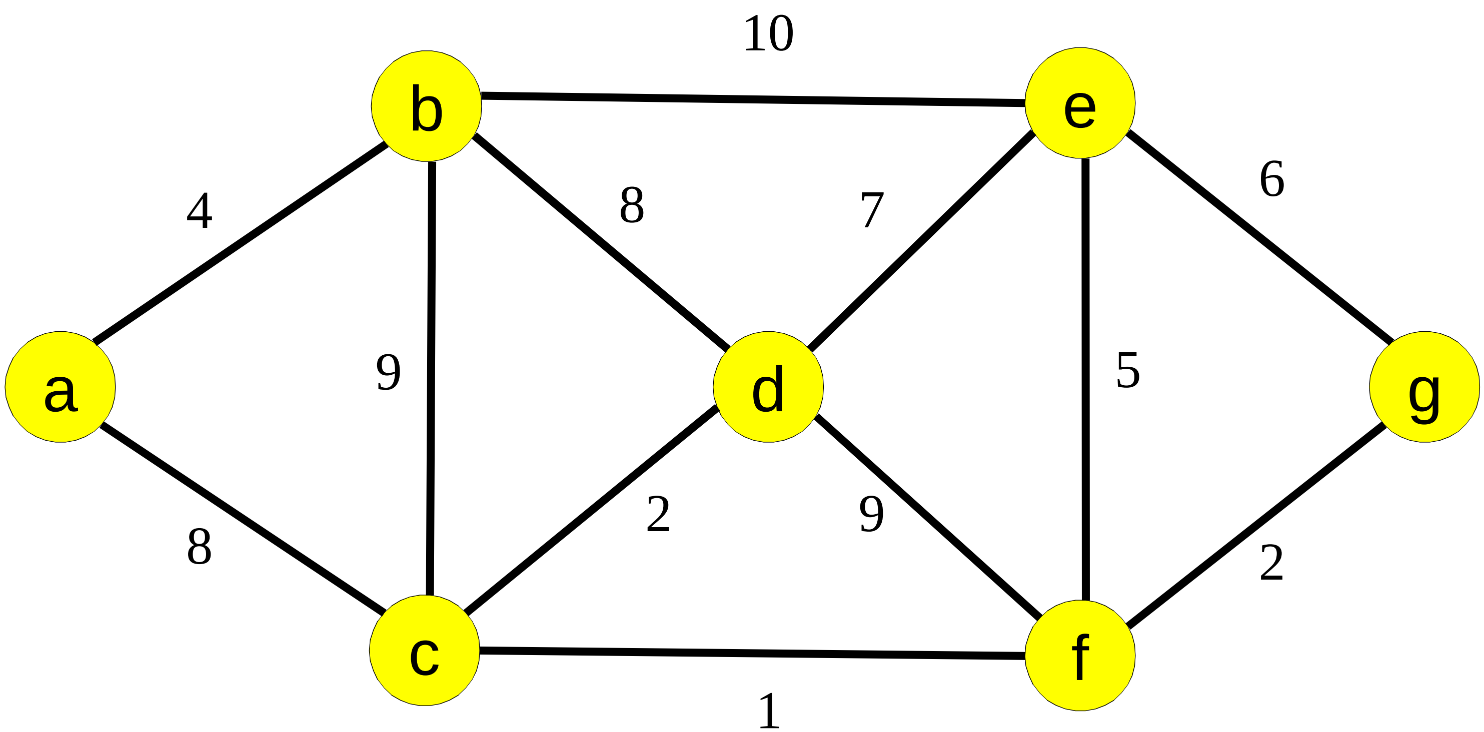
- Greedy algorithm for solving shortest path problem
- Assume non-negative weights
- Find shortest path from v_s to each other vertex

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

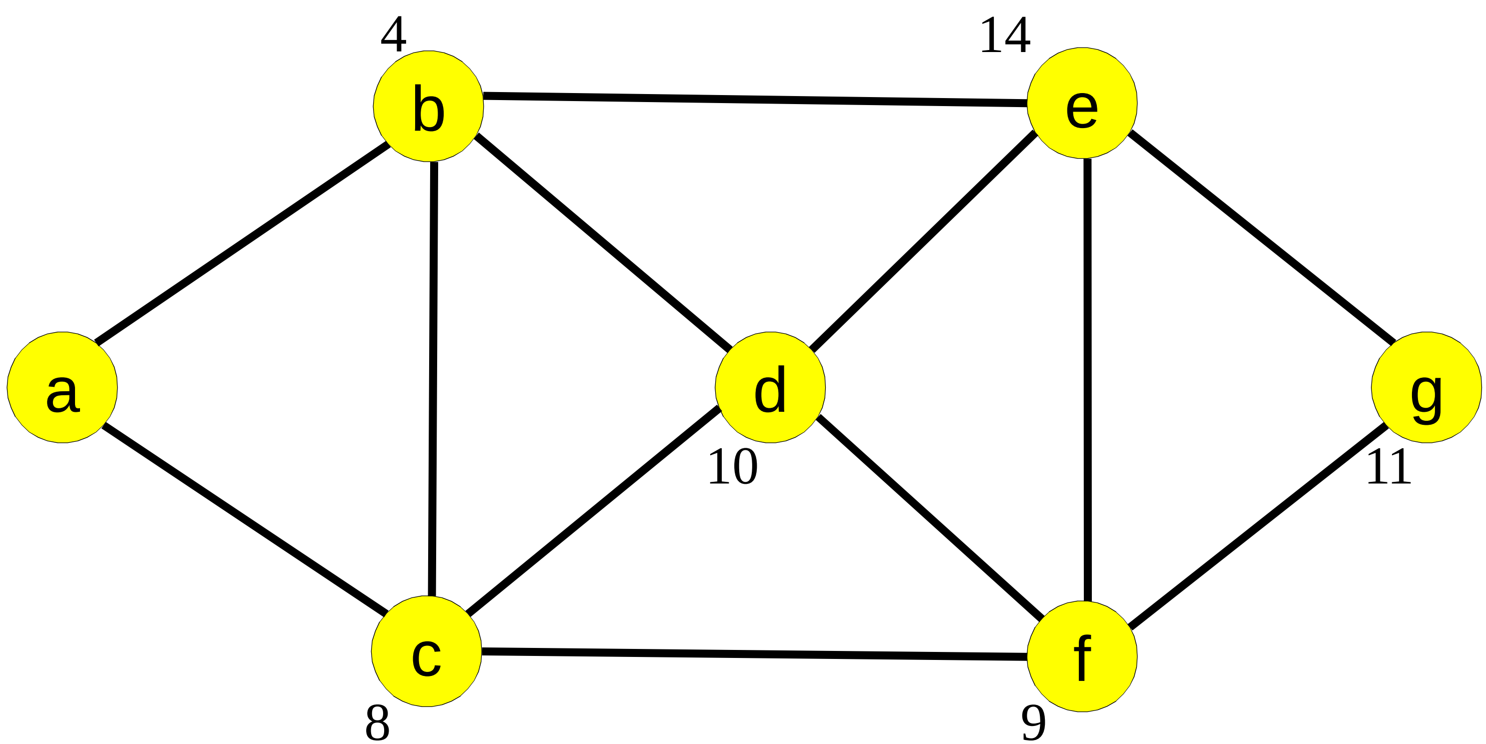
Shortest Path

- Do Djikstra's algorithm starting at node a



Shortest Path

- V_s = 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

- Brute force
  - “Simple” straight-forward way
  - Usually inefficient
- Greedy
  - Decisions never reconsidered
  - Are never optimal? -- FALSE, e.g. fractional knapsack

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?

```
// returns the nth Fibonacci number
long long fib(int n) {
    if (n <= 0) return 0;
    if (n <= 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Dynamic Programming

- What family best describes the implementation below?
  - Divide and conquer

```
// returns the nth Fibonacci number
long long fib(int n) {
    if (n <= 0) return 0;
    if (n <= 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Top-down Solution

```
long long fib(int n) {
    static unordered_map<int,long long> computed;

    // base cases
    if (n <= 0) return 0;
    if (n == 1) return 1;

    // look up
    auto found = computed.find(n);
    if (found != computed.end()) return found->second;

    // compute
    computed[n] = fib(n - 1) + fib(n - 2);
    return computed[n];
}
```

Bottom-up solution

```
long long fib(int n) {
    long long fib_prev = 0;
    long long fib_cur = 1;
    for (int i = 0; i < n - 1; ++i) {
        long long tmp = fib_cur;
        fib_cur += fib_prev;
        fib_prev = tmp;
    }

    return fib_cur;
}
```

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 courses!
  - 388 Introduction to Computer Security, 442 Computer Vision, 445 Introduction to Machine Learning, 467 Autonomous Robotics, 475 Introduction to Cryptography, 477 Introduction to Algorithms, 478 Logic Circuit Synthesis and Optimization, 480 Logic and Formal Verification , 481 Software Engineering , 482 Introduction to Operating Systems, 483 Compiler Construction, 484 Database Management Systems, 485 Web Database and Information Systems, 487 Interactive Computer Graphics , 489 Computer Networks, 490 Programming Languages, 492 Introduction to Artificial Intelligence, 493 User Interface Development, 494 Computer Game Design and Development
- EECS 381
  - Advanced C++
  - Object-oriented programming
- Learning on your own
  - Evaluate libraries written by others (e.g., Boost) and decide when to borrow, and when to “roll your own”

Thank you!