• Homework 5 will be posted tomorrow
• Homework 4
  • Due Saturday at 2am (so, basically Friday night)
  • Problem C is a bonus point!
    • BUT: I will randomly select one of the people to present their solution next Thursday for an additional bonus point
    • If chosen, you **cannot opt-out** of presenting (you’ll have to give it your best shot)
• Midterm: Tuesday, March 10, 2015
  • Practice midterm online
  • Will only cover up to and including dynamic programming
    • i.e., this lecture will not be in the midterm
Exercise: In an undirected graph, suppose you are doing a DFS and have processed one child $c_1$ of the root. After the child has finished, there is another child $c_2$ still in the INIT state that must be processed. What happens to the graph if we remove the root?
• DFS can also be used to find bridges and articulation points
• **Definition**: a bridge is an edge whose removal increases the number of connected components
• **Definition**: an articulation point is a vertex whose removal increases the number of connected components
• Problems involving finding bridges and articulation points usually defined for undirected, connected graphs
  • Harder for directed graphs
  • E.g., book’s example: sabotaging road networks
  • Recall naïve approach for finding bridges and articulation points
    • Runtime $O(V^2 + VE)$
• Consider the root of a DFS tree
  • After a child is processed, if other unprocessed children exist, then the root is an articulation point
• For non-roots:
  • Each vertex in the DFS tree will be visited in some order, keep track of two numbers:
    • dfs_num: counter for when the vertex is visited for the first time
    • dfs_low: lowest dfs_num reachable from the vertex
      o But ignore immediate parents
  • Initially, dfs_num = dfs_low for all vertices
  • dfs_low can only be made smaller if there are cycles (back edges)
• If a vertex $u$ has neighbor $v$ with $\text{dfs\_low}(v) \geq \text{dfs\_num}(u)$, then $u$ is an articulation point
  • No back edge from $v$ to ancestors of $u$
  • In order for $v$ and descendants to reach ancestors of $u$, traversal must pass through $u$
• **Exercise**: how do we adapt this algorithm to find bridges?
• Same algorithm can be used to find bridges, except: $\text{dfs\_low}(v) > \text{dfs\_num}(u)$ implies that $(u,v)$ is a bridge
  • Note that it no longer includes equality
• **Spanning tree**
  • Given: a connected, undirected graph \( G = (V, E) \)
    • \( V \) is the set of vertices, \( E \) is the set of edges
  • A spanning tree is a set of edges that is a tree and “covers” all vertices \( V \)
    • There can be several trees
  • The spanning tree with the minimum cost (sum of edge weights) is called the **Minimum Spanning Tree**
Minimum Spanning Tree

Minimum spanning tree
Cost: 4 + 2 + 6 + 6 = 18
• **Kruskal's algorithm for finding the MST**
  - Repeatedly finds edges with minimum costs that does not form a cycle
  - Greedy algorithm, provably correct

• **Kruskal's algorithm pseudocode**
  - Sort edges by increasing weight
  - While there are unprocessed edges left
    - Pick an edge $e$ with minimum cost
    - If adding $e$ to the MST does not form a cycle, add $e$ to MST
• Kruskal's algorithm pseudocode
  • How to store and sort edges?
    • Using an edge list and Collections.sort
  • How to test for cycles?
    • using disjoint sets and union-find (**Exercise**: how?)
  • Runtime?
    • Sort: $O(|E| \log |E|)$
    • Processing: for each edge, check union-find: $O(|E|) \times O(1)$
    • Total: $O(|E| \log |E|) = O(|E| \log |V|)$

• **Exercise**: If the weights of the edges are integers within a small range (e.g., [0, 100]), can Kruskal’s be made faster?
Kruskal’s Algorithm: Example

*Pick smallest edge*  
Cycle formed, ignore

*Pick smallest edge*  
No cycle

*Pick smallest edge*  
No cycle
Algorithm not done! The edge list hasn't yet been exhausted

Pick smallest edge
Cycle formed, ignore

Pick smallest edge
Cycle formed, ignore
ArrayList<Edge> edgeList = parseEdgeList();
Collections.sort(edgeList);

int mstCost = 0;
UnionFind uf = new UnionFind(nVertices);

for (Edge e : edgeList) { // for each edge
    if (!uf.isSameSet(e.A, e.B)) { // if no cycle
        mstCost += e.w; // add it
        uf.union(e.A, e.B);
    }
}

System.out.println(mstCost);
Quick summary of Prim’s algorithm:
1. Begin with a set of vertices \( V \), initialized with an arbitrary vertex, and empty set of edges \( E \)
2. From all edges, add edge \((u, v)\) with least weight such that \( u \) is in \( V \) and \( v \) is not
   • We’re building a tree, and continuously adding vertices to the tree
3. Repeat step 2 until all vertices have been added to the tree

Greedy algorithm
- Also \( O(|E| \log |V|) \) running time
- More detail in textbook
Exercise: Given a connected weighted graph length that stores the road length between E pairs of cities \(i\) and \(j\) (\(1 \leq V \leq 1000\), \(0 \leq E \leq 10000\)), the price \(p[i]\) of fuel at each city \(i\), and the fuel tank capacity \(c\) of a car (\(1 \leq c \leq 100\)), determine the cheapest trip cost from starting city \(s\) to ending city \(e\) using a car with fuel capacity \(c\). All cars use one unit of fuel per unit of distance and start with an empty fuel tank.
• For directed, weighted graphs:
  • Without negative weights
• Dijkstra’s algorithm:
  • Recall BFS: instead of enqueuing just neighbors, enqueue (total weight to source, neighbor), with the priority queue sorting by weight
  • Minimum distance from source to other vertices stored in array
    • Array updated as smaller-weight paths are found
    • Initialized with inf
  • \(O((|V| + |E|) \log |V|)\)
Dijkstra’s Algorithm: Example

Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)
{(0, 2)}

Distance table

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>d[i]</td>
<td>INF</td>
<td>INF</td>
<td>0</td>
<td>INF</td>
<td>INF</td>
</tr>
</tbody>
</table>

Start from node 2
Dijkstra’s Algorithm: Example

Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)
{(0, 2)}
{(2, 1), (6, 0), (7, 3)}

Distance table

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>d[i]</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>INF</td>
</tr>
</tbody>
</table>

Add all unvisited nodes from node 2 to the priority queue.
The PQ sorts the distances so the “next closest” node floats to the top.
Right now the closest node is 1, followed by 0, then 3.
Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)
{(0, 2)}
{(2, 1), (6, 0), (7, 3)}
{(5, 3), (6, 0), (7, 3), (8, 4)}

Poll from the PQ to get node 1.
Add all neighboring nodes to node 1 that haven't been polled yet.
BUT be sure to add all nodes that may already be in the queue with longer distances – there may be a shorter way to reach them.
Dijkstra's Algorithm: Example

Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

\{ (0, 2) \}
\{ (2, 1), (6, 0), (7, 3) \}
\{ (5, 3), (6, 0), (7, 3), (8, 4) \}
\{ (6, 0), (7, 3), (8, 4) \}

Poll from the PQ to get node 3.
Since we know there is a faster way to get node 4, don't bother adding node 4 to PQ
Dijkstra’s Algorithm: Example

Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)
{ (0, 2) }
{ (2, 1), (6, 0), (7, 3) }
{ (5, 3), (6, 0), (7, 3), (8, 4) }
{ (6, 0), (7, 3), (8, 4) }
{ (7, 3), (7, 4), (8, 4) }

Distance table

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>d[i]</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Poll from the PQ to get node 0.
Since we know there is a faster way to get node 4, don't bother adding node 4 to PQ
Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)
{(0, 2)}
{(2, 1), (6, 0), (7, 3)}
{(5, 3), (6, 0), (7, 3), (8, 4)}
{(6, 0), (7, 3), (8, 4)}
{(7, 3), (7, 4), (8, 4)}
{(7, 4), (8, 4)}

Now the (7, 3) state is ignored because it's been determined that 7 is a longer path than another existing path to node 3
Dijkstra’s Algorithm: Example

Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)

\{(0, 2)\}
\{(2, 1), (6, 0), (7, 3)\}
\{(5, 3), (6, 0), (7, 3), (8, 4)\}
\{(6, 0), (7, 3), (8, 4)\}
\{(7, 3), (7, 4), (8, 4)\}
\{(7, 4), (8, 4)\}
\{(8, 4)\}

Distance table

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>d[i]</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

Nowhere to go, so nothing is added to the PQ
Dijkstra's Algorithm: Example

Question: Shortest paths from 2 to all other nodes?

Priority Queue (distance, node index)
{(0, 2)}
{(2, 1), (6, 0), (7, 3)}
{(5, 3), (6, 0), (7, 3), (8, 4)}
{(6, 0), (7, 3), (8, 4)}
{(7, 3), (7, 4), (8, 4)}
{(7, 4), (8, 4)}
{(8, 4)}

Distance table

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>d[i]</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

State (8, 4) is ignored because 8 > 7
Exercise: Will Dijkstra’s work for undirected, unweighted graphs?

Exercise: How do you find the single-source longest path?
• Dijsktra’s will not work for negative weights
  • Greedy nature (i.e., the priority queue)
  • Infinite loop if there’s a cycle
• What to do in case of negative weights?
  • Bellman-Ford: relax all E edges V-1 times
    • Arbitrary order
  • If you can continue to relax edges after V-1 times, then a negative cycle exists
  • Running time: \(O(|E| \times |V|)\)
Problem: UVA 104: Arbitrage

Summary: Take advantage of currency fluctuations to make money.

Input: Table of exchange rates

Output: Shortest sequence of currencies to buy that will yield a profit of more than 1%
Competitive Programming 4.1-4.4