Week 9.2, Wed, Oct 16

Homework 5 Released Soon
Strongly Connected Components (22.5)

Let G be a directed graph.

- G is **strongly connected** if there exists a path between any pair of vertices.
- If G is not strongly connected, decompose G into strongly connected components:
  - sets of vertices in which any two vertices are *mutually reachable*
  - each vertex set cannot be enlarged by adding more vertices without destroying this property.
Strongly connected components of $G$
Determine the strongly connected components (stcc) in $O(n+m)$ time

Perform 2 DFS’s
On what graphs?

• $G^T$ is the transpose of $G$ generated by reversing the direction of every edge
• $G^T$ and $G$ have the same strongly connected components

Record discovery and finish times
Directed Graph G

DFS restarts

d(C) = 17
f(C) = 22

d(B) = 13
f(B) = 14

d(H) = 19
f(H) = 20

d(G) = 3
f(G) = 6

d(J) = 2
f(J) = 7

d(K) = 4
f(K) = 5

d(A) = 1
f(A) = 16

d(F) = 8
f(F) = 15

d(I) = 9
f(I) = 12

d(E) = 10
f(E) = 11
Directed Graph $G^T$

- Run DFS at C (node with max finish time)
  - $d(C)=17$, $f(C)=22$
  - $d(B)=13$, $f(B)=14$
  - $d(D)=18$, $f(D)=21$
  - $d(H)=19$, $f(H)=20$

- Run DFS at J (remaining node with max finish time)
  - $d(J)=2$, $f(J)=7$
  - $d(K)=4$, $f(K)=5$

- Run DFS at A (remaining node with max finish time)
  - $d(A)=1$, $f(A)=16$
  - $d(F)=8$, $f(F)=15$
  - $d(E)=10$, $f(E)=11$
  - $d(I)=9$, $f(I)=12$

Claim: Same strongly connected components as $G$! Why?

Run DFS at C (node with max finish time)
Sketch of algorithm finding the stcc

1. call DFS on G to compute f[u] for each vertex u
   A. Sort nodes in decreasing order of f[u]
   B. (Only requires time $O(n)$ since $1 \leq f[u] \leq 2n$)
2. find $G^T$, the transpose of G
3. call DFS on $G^T$
   • consider the vertices in order of decreasing $f[u]$
4. the second DFS generates one or more tree
   • the vertices in each tree form one strongly connected component
Why does the algorithm find the stcc?
Not obvious.

Create the following “reduced” graph $R=(V_R,E_R)$

- Shrink every stcc into a single vertex.
- Put edges not in a stcc into graph $R$ and remove duplicate edges.

**Graph R is a dag**
There must exist at least one “vertex” that has no incoming edges and at least one vertex with no outgoing edges.
DFS on G starts at A and restarts at vertex C

DFS on $G^T$ starts at vertex C and finds the first stcc

DFS on $G^T$ re-starts at vertex A and finds the second stcc
Let $U$ be a set of vertices of directed graph $G$
- $d(U)$ is the smallest discovery time of any vertex in $U$
- $f(U)$ is the largest finishing time of any vertex in $U$

Assume $C$ and $C'$ are two strongly connected components of $G$.

**Claim 1**: If there is an edge $(u, v)$ in $G$ with $u$ in $C$ and $v$ in $C'$, then $f(C) > f(C')$.

**Claim 2**: If there is an edge $(v, u)$ in the transpose of $G$ with $v$ in $C'$ and $u$ in $C$, then $f(C') < f(C)$. 
Main Idea - Summary

Second DFS on $G^T$

• we start with the component $C$ whose $f(C)$ is the biggest
  (actually we start with $x$ in $C$ where $f(x)$ is the biggest).
• No edges go from inside $C$ to any other component.
• The tree rooted at $x$ contains exactly the vertices in $C$ and we
  generated one strongly connected component.

Repeat the argument for the next sink in graph $R$ until all strongly
connected components have been generated.

Hence, the strongly connected components can be found in $O(n+m)$
time by doing two DFS’s.
4.4 Shortest Paths in a Graph
Shortest Path Problem

Shortest path network.

- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $\ell_e = \text{length of edge } e$.

**Shortest path problem:** find shortest directed path from $s$ to $t$.

Cost of path $s$-$2$-$3$-$5$-$t$  
\[= 9 + 23 + 2 + 16 \]
\[= 50. \]
Dijkstra's algorithm (Greedy).

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + l_e,$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

shortest path to some $u$ in explored part, followed by a single edge $(u, v)$
Dijkstra's Algorithm

Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$
\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,
$$

add $v$ to $S$, and set $d(v) = \pi(v)$. shortest path to some $u$ in explored part, followed by a single edge $(u, v)$

---

![Diagram of Dijkstra's algorithm](image-url)
**Invariant.** For each node \( u \in S \), \( d(u) \) is the length of the shortest \( s-u \) path.

**Pf.** (by induction on \( |S| \))

**Base case:** \( |S| = 1 \) is trivial.

**Inductive hypothesis:** Assume true for \( |S| = k \geq 1 \).

- Let \( v \) be next node added to \( S \), and let \( u-v \) be the chosen edge.
- The shortest \( s-u \) path plus \( (u, v) \) is an \( s-v \) path of length \( \pi(v) \).
- Consider any \( s-v \) path \( P \). We'll see that it's no shorter than \( \pi(v) \).
- Let \( x-y \) be the first edge in \( P \) that leaves \( S \), and let \( P' \) be the subpath to \( x \).
- \( P \) is already too long as soon as it leaves \( S \).

\[
\ell(P) \geq \ell(P') + \ell(x,y) \geq d(x) + \ell(x,y) \geq \pi(y) \geq \pi(v)
\]

- \( \ell(P) \) nonnegative weights
- \( \ell(P') \) inductive hypothesis
- \( \ell(x,y) \) defn of \( \pi(y) \)
- \( d(x) \) Dijkstra chose \( v \) instead of \( y \)
Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain \( \pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e \).

- Next node to explore = node with minimum \( \pi(v) \).
- When exploring \( v \), for each incident edge \( e = (v, w) \), update (decrease key)
  \[ \pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \} . \]

**Efficient implementation.** Maintain a priority queue of unexplored nodes, prioritized by \( \pi(v) \).
Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain \( \pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e \).

- Next node to explore = node with minimum \( \pi(v) \).
- When exploring \( v \), for each incident edge \( e = (v, w) \), update (decrease key)
  \[ \pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \} . \]

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by \( \pi(v) \).

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap ( \dagger )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log_d n )</td>
<td>1</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log_d n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>( m )</td>
<td>1</td>
<td>( \log n )</td>
<td>( \log_d n )</td>
<td>1</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>( n )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>( n^2 )</td>
<td>( m \log n )</td>
<td>( m \log_{m/n} n )</td>
<td>( m + n \log n )</td>
<td></td>
</tr>
</tbody>
</table>

\( \dagger \) Individual ops are amortized bounds
Maximum Capacity Path Problem

Each edge $e$ has capacity $c_e$ (e.g., maximum height)

Capacity of a path is Minimum capacity of any Edge in path

Goal: Find path from $s$ to $t$ with maximum capacity

Solution: Use Dijkstra!
With Small Modification

$$
\pi(v) = \max_{e=(u,v): u \in S} \min \{ \pi(u), c_e \}
$$