CSE 2331

Foundations II:
Data Structures and Algorithms
Graphs (Review)
Vertex Degree

\[ \text{deg}(v_i) = \text{degree of } v_i = \# \text{ edges incident on } v_i. \]

What is the degree of each vertex?
Sum of $\deg(v_i)$

$m =$ number of graph edges.

Proposition: $\sum_{v_i \in V(G)} \deg(v_i) = 2m$.

**Proof 1:**

$\sum_{v_i \in V} \deg(v_i) =$ sum of $\#$ edges incident on vertices

$\quad =$ sum of $\#$ vertices incident on edges

$\quad = 2 \times (\# \text{ edges}) = 2m.$

**Proof 2:** Let $\delta_{ij} = 1$ if $e_j$ is incident on $v_i$ and 0 otherwise.

$\sum_{v_i \in V} \deg(v_i) = \sum_{v_i \in V} \sum_{e_j \in E} \delta_{ij} = \sum_{e_j \in E} \sum_{v_i \in V} \delta_{ij} = \sum_{e_j \in E} 2 = 2m.$
Graph Representation

Adjacency matrix:

\[
\begin{bmatrix}
  v_1 & v_2 & v_3 & v_4 & v_5 \\
  v_1 : & 0 & 1 & 1 & 0 & 0 \\
  v_2 : & 1 & 0 & 1 & 1 & 1 \\
  v_3 : & 1 & 1 & 0 & 0 & 1 \\
  v_4 : & 0 & 1 & 0 & 0 & 0 \\
  v_5 : & 0 & 1 & 1 & 0 & 0 \\
\end{bmatrix}
\]

\(A[i, j] = 1\) if \((v_i, v_j)\) is an edge.

Adjacency lists:

\[
\begin{align*}
  v_1 : & (v_2, v_3) \\
  v_2 : & (v_1, v_3, v_4, v_5) \\
  v_3 : & (v_1, v_2, v_5) \\
  v_4 : & (v_2) \\
  v_5 : & (v_2, v_3)
\end{align*}
\]

\(V[i].\text{degree} = \text{size of adj list}.

\(V[i].\text{AdjList}[k] = \text{k}^{th} \text{ element of the adjacency list}.\)
Sample Graph Algorithm

Input : Graph $G$ represented by adjacency lists.

Func($G$)
1 $k \leftarrow 0$;
2 foreach vertex $v_i \in V(G)$ do
3     foreach edge $(v_i, v_j)$ incident on $v_i$ do
4         $k \leftarrow k + 1$;
5     end
6 end
7 return $(k)$;
Connected Graphs

Definition. A path in a graph is a sequence of vertices \((w_1, w_2, \ldots, w_k)\) such that there is an edge \((w_i, w_{i+1})\) between every two adjacent vertices in the sequence.

Definition. A graph is connected if for every two vertices \(v, w \in V(G)\), the graph has some path from \(v\) to \(w\).

Is the following graph connected?
Connected Graphs

Is the following graph connected?
Connected Graphs

Is the following graph connected?
Minimum Spanning Tree
Edge Weights: Example

Graph with weighted edges:
- $v_1$ to $v_2$: 7
- $v_1$ to $v_6$: 8
- $v_2$ to $v_3$: 6
- $v_2$ to $v_5$: 11
- $v_3$ to $v_4$: 5
- $v_4$ to $v_8$: 4
- $v_4$ to $v_5$: 3
- $v_5$ to $v_6$: 9
- $v_5$ to $v_7$: 7
- $v_3$ to $v_8$: 5
- $v_1$ to $v_2$: 2
A **spanning tree** of graph $G$ is a tree composed of all the vertices and some of the edges of $G$.

The **weight** of a spanning tree is the sum of the edge weights.
Minimum (Weight) Spanning Tree: Example
Minimum (Weight) Spanning Tree: Example 2

Note: Weight of edge \((v_6, v_7)\) is 6 (previously was 7).
Two Minimum (Weight) Spanning Trees

What is the other minimum (weight) spanning tree of this graph?
Minimum (Weight) Spanning Tree: Example

![Graph](image)
Prim’s Minimum Spanning Tree Algorithm

Input : Weighted graph G
Output : Minimum spanning tree of G

PrimMST(G)

1 $U \leftarrow V(G) - \{v_1\}$ ; /* $V(G) =$ set of vertices of graph $G$ */
2 $v_1$.parent $\leftarrow$ NULL;
3 while $(U \neq \emptyset)$ and (∃ edge from $(V(G) - U)$ to $U$) do
4 \hspace{1em} ($v_i, v_j$) $\leftarrow$ minimum weight edge from $V(G) - U$ to $U$;
5 \hspace{1em} $v_j$.parent $\leftarrow v_i$;
6 \hspace{1em} $U \leftarrow U - \{v_j\}$;
7 end
Minimum Spanning Tree: Example 2
Prim’s Minimum Spanning Tree Algorithm

PrimMST(G)

1. \( U \leftarrow V(G) - \{v_1\} \); /* \( V(G) = \text{set of vertices of graph } G \) */
2. \( v_1\.parent \leftarrow \text{NULL}; \)
3. \( \text{while } (U \neq \emptyset) \text{ and } (\exists \text{ edge from } (V(G) - U) \text{ to } U) \text{ do} \)
4. \( (v_i,v_j) \leftarrow \text{minimum weight edge from } V(G) - U \text{ to } U; \)
5. \( v_j\.parent \leftarrow v_i; \)
6. \( U \leftarrow U - \{v_j\}; \)
7. \( \text{end} \)

**MST Theorem:** Let \( T \) be a subtree of a minimum spanning tree. If \( e \) is a minimum weight edge connecting \( T \) to some vertex not in \( T \), then \( T \cup \{e\} \) is a subtree of a minimum spanning tree.
Proof of MST Theorem

**MST Theorem:** Let $T$ be a subtree of a minimum spanning tree. If $e$ is a minimum weight edge connecting $T$ to some vertex not in $T$, then $T \cup \{e\}$ is a subtree of a minimum spanning tree.

**Proof.** By the theorem’s hypothesis, $T$ is a subtree of some minimum spanning tree $A$ of $G$.

If $e$ is not an edge of $A$, then $A \cup \{e\}$ contains a cycle. Some edge $e'$ of this cycle must be an edge from $T$ to a vertex not in $T$.

Since $e$ is a minimum weight edge from $T$ to vertices not in $T$, $\text{weight}(e) \leq \text{weight}(e')$.

Replacing $e'$ by $e$ in $A$ gives a tree $B$ where $\text{weight}(B) \leq \text{weight}(A)$. Therefore, $B$ is a minimum spanning tree of $G$ containing $T \cup \{e\}$. $\square$
Running Time of PrimMST: Naive Implementation

PrimMST(G)

1. $U \leftarrow V(G) - \{v_1\}$; /* $V(G)$ = set of vertices of graph $G$ */
2. $v_1$.parent $\leftarrow$ NULL;
3. while ($U \neq \emptyset$) and (exists edge from ($V(G) - U$) to $U$) do
4.    $(v_i, v_j) \leftarrow$ minimum weight edge from $V(G) - U$ to $U$;
5.    $v_j$.parent $\leftarrow v_i$;
6.    $U \leftarrow U - \{v_j\}$;
7. end

$n = \text{number of vertices of } G.$

$m = \text{number of edges of } G.$

What is the running time of PrimMST in terms of $n$ and $m$?
Minimum Spanning Tree: Storing Costs at Vertices
Prim’s MST: Storing Costs at Vertices

PrimMST(G)

1 \( U \leftarrow V(G) ; \quad /* V(G) = \text{set of vertices of graph } G */ */

2 \textbf{foreach } v_i \in V(G) - \{v_1\} \textbf{ do } v_i\.cost \leftarrow \infty ;

3 \( v_1\.cost \leftarrow 0; \)

4 \( v_1\.parent \leftarrow \text{NULL}; \)

5 \textbf{while } (U \neq \emptyset) \textbf{ and } (v_i\.cost < \infty \text{ for some } v_i \in U) \textbf{ do}

6 \hspace{1em} v_j \leftarrow v_i \in U \text{ with minimum } v_i\.cost;

7 \hspace{1em} U \leftarrow U - \{v_j\} ; \quad /* \text{Remove } v_j \text{ from } U */ */

8 \hspace{1em} /* (v_j, v_j\.parent) \text{ is an MST edge} */

9 \hspace{1em} \textbf{foreach} \text{ edge } (v_j, v_k) \text{ incident on } v_j \textbf{ do}

10 \hspace{2em} \textbf{if} \ (v_k \text{ is in } U \text{ and } \text{weight}(v_j, v_k) < v_k\.cost) \textbf{ then}

11 \hspace{3em} v_k\.parent \leftarrow v_j;

12 \hspace{3em} v_k\.cost \leftarrow \text{weight}(v_j, v_k);

13 \hspace{2em} \textbf{end}

14 \textbf{end}
Minimum Spanning Tree:
Storing Costs at Vertices
Running Time Analysis

PrimMST(G)

1 \( U \leftarrow V(G) \); /* \( V(G) = \text{set of vertices of graph } G \) */
2 foreach \( v_i \in V(G) - \{v_1\} \) do \( v_i.\text{cost} \leftarrow \infty \);
3 \( v_1.\text{cost} \leftarrow 0; \)
4 \( v_1.\text{parent} \leftarrow \text{NULL}; \)
5 while \((U \neq \emptyset) \) and \((v_i.\text{cost} < \infty \) for some \( v_i \in U) \) do
6 \hspace{1em} \( v_j \leftarrow v_i \in U \) with minimum \( v_i.\text{cost}; \)
7 \hspace{1em} \( U \leftarrow U - \{v_j\}; \) /* Remove \( v_j \) from \( U \) */
8 \hspace{2em} /* (v_j, v_j.\text{parent}) is an MST edge */
9 foreach edge \((v_j, v_k)\) incident on \( v_j \) do
10 \hspace{2em} if \((v_k \text{ is in } U \text{ and weight}(v_j, v_k) < v_k.\text{cost}) \) then
11 \hspace{3em} \( v_k.\text{parent} \leftarrow v_j; \)
12 \hspace{3em} \( v_k.\text{cost} \leftarrow \text{weight}(v_j, v_k); \)
13 \hspace{2em} end
14 end
# Priority Queue

<table>
<thead>
<tr>
<th>Operations</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q.Insert(Object x, Key k);</td>
<td>(Heap Implementation)</td>
</tr>
<tr>
<td>$x \leftarrow Q.DeleteMin();$</td>
<td></td>
</tr>
<tr>
<td>Q.IsEmpty();</td>
<td></td>
</tr>
<tr>
<td>Q.DecreaseKey(Object x, Key k);</td>
<td></td>
</tr>
<tr>
<td>Q.IsNotNull();</td>
<td></td>
</tr>
<tr>
<td>Q.Contains(Object x);</td>
<td></td>
</tr>
<tr>
<td>$k \leftarrow Q.Key(Object x);$</td>
<td></td>
</tr>
<tr>
<td>$k \leftarrow Q.MinKey();$</td>
<td></td>
</tr>
</tbody>
</table>

$s =$ number of objects in the queue (queue size).
Prim’s MST: Storing Costs at Vertices

PrimMST(G)
1 $U \leftarrow V(G)$ ; /* $V(G) = set$ of vertices of graph $G$ */
2 foreach $v_i \in V(G) - \{v_1\}$ do $v_i$.cost $\leftarrow \infty$ ;
3 $v_1$.cost $\leftarrow 0$;
4 $v_1$.parent $\leftarrow$ NULL;
5 while ($U \neq \emptyset$) and ($v_i$.cost $< \infty$ for some $v_i \in U$) do
6  $v_j \leftarrow v_i \in U$ with minimum $v_i$.cost;
7  $U \leftarrow U - \{v_j\}$ ; /* Remove $v_j$ from $U$ */
8  /* $(v_j, v_j$.parent) is an MST edge */
9  foreach edge $(v_j, v_k)$ incident on $v_j$ do
10     if ($v_k$ is in $U$ and weight$(v_j, v_k) < v_k$.cost) then
11        $v_k$.parent $\leftarrow v_j$;
12        $v_k$.cost $\leftarrow$ weight$(v_j, v_k)$;
13    end
14 end
Prim’s MST: Priority Queue of Vertices

PrimMST(G)

1. foreach $v_i \in V(G) - \{v_1\}$ do $Q$.Insert($v_i, \infty$);
2. $Q$.Insert($v_1, 0$); /* $Q$ is a priority queue of vertices */
3. $v_1$.parent ← NULL;

4. while $Q$.IsNotEmpty() and ($Q$.MinKey() ≠ $\infty$) do
5.   $v_j$ ← $Q$.DeleteMin(); /* ($v_j, v_j$.parent) is an MST edge */
6.   foreach edge ($v_j, v_k$) incident on $v_j$ do
7.     if ($Q$.Contains($v_k$) and $Q$.Key($v_k$) > weight($v_j, v_k$))
8.       then
9.         $v_k$.parent ← $v_j$;
10.        $Q$.DecreaseKey($v_k$, weight($v_j, v_k$));
11.     end
12. end
Shortest Paths
Single Source Shortest Path: Example
Single Source Shortest Path: Example
Shortest Path Theorem

**Shortest Path Theorem:** If $(v_1, v_2, \ldots, v_{i-1}, v_i)$ is a shortest path from $v_1$ to $v_i$, then $(v_1, v_2, \ldots, v_{i-1})$ is a shortest path from $v_1$ to $v_{i-1}$.

*Proof.* If there is a shorter path $P$ from $v_1$ to $v_{i-1}$, then $P \cup (v_{i-1}, v_i)$ would be shorter than $(v_1, v_2, \ldots, v_{i-1}, v_i)$. 

Shortest Path Tree Versus Min Spanning Tree

Shortest Path Tree:

Minimum Spanning Tree:
Single Source Shortest Path: Example 2

Note: Weight of edge \((v_2, v_3)\) is 5 (previously was 6).
Two Shortest Path Trees

What is the other shortest path tree of this graph?
Single Source Shortest Path: Example
Dijkstra’s Shortest Path Algorithm

**Input**: Weighted graph $G$.

Start vertex $v_s$.

**Output**: Shortest path tree from $v_s$ to all vertices of $G$.

\[
\text{DijkstraShortestPath}(G, v_s) \]

1. $U \leftarrow V(G) - \{v_s\}$ ; /* $V(G) =$ set of vertices of graph $G$ */
2. $v_s$.parent $\leftarrow$ NULL;
3. $v_s$.distance $\leftarrow$ 0;
4. while $(U \neq \emptyset)$ and (exists edge from $(V(G) - U)$ to $U$) do
5. \hspace{1em} $(v_i, v_j) \leftarrow$ edge from $V(G) - U$ to $U$ which minimizes $v_i$.distance + weight$(v_i, v_j)$;
6. \hspace{1em} $v_j$.parent $\leftarrow v_i$;
7. \hspace{1em} $v_j$.distance $\leftarrow v_i$.distance + weight$(v_i, v_j)$;
8. \hspace{1em} $U \leftarrow U - \{v_j\}$;
9. end
Single Source Shortest Path: Example 2
Dijkstra’s Shortest Path Algorithm

DijkstraShortestPath(G, vs)

1. \( U \leftarrow V(G) - \{v_s\} \); /* \( V(G) \) = set of vertices of graph \( G \) */
2. \( v_s\.parent \leftarrow \text{NULL} \);
3. \( v_s\.distance \leftarrow 0 \);
4. while \( (U \neq \emptyset) \) and (\( \exists \) edge from \( (V(G) - U) \) to \( U \)) do
   5. \((v_i, v_j) \leftarrow \text{edge from } V(G) - U \text{ to } U \text{ which minimizes} \)
      \( v_i\.distance + \text{weight}(v_i, v_j) \);
   6. \( v_j\.parent \leftarrow v_i \);
   7. \( v_j\.distance \leftarrow v_i\.distance + \text{weight}(v_i, v_j) \);
   8. \( U \leftarrow U - \{v_j\} \);
9. end

\( d(v_s, v_i) = \text{shortest distance from } v_s \text{ to } v_i \).

**Shortest Path Theorem 2:** Let \( T \) be a subtree of a shortest path tree where \( v_s \in V(T) \). If \( e \in E(G) \) is an edge connecting \( v_i \in V(T) \) to \( v_j \notin V(T) \) which minimizes \( d(v_s, v_i) + \text{weight}(v_i, v_j) \), then \( T \cup \{e\} \) is a subtree of a shortest path tree.
Proof of Shortest Path Theorem 2

\[ d(v_s, v_i) = \text{shortest distance from } v_s \text{ to } v_i. \]

**Shortest Path Theorem 2:** Let \( T \) be a subtree of a shortest path tree where \( v_s \in V(T) \). If \( e \in E(G) \) is an edge connecting \( v_i \in V(T) \) to \( v_j \not\in V(T) \) which minimizes \( d(v_s, v_i) + \text{weight}(v_i, v_j) \), then \( T \cup \{e\} \) is a subtree of a shortest path tree.

*Proof.* Let \( e = (v_a, v_b) \) where \( v_a \in V(T) \) and \( v_b \not\in V(T) \). Let \( P \) be the path from \( v_s \) to \( v_a \) in \( T \). We claim that \( P \cup \{e\} \) is a shortest path from \( v_s \) to \( v_b \).

The length of \( P \) is \( d(v_s, v_a) \).
The length of \( P \cup \{e\} \) is \( d(v_s, v_a) + \text{weight}(v_a, v_b) \).

Let \( P' \) be any path from \( v_s \) to \( v_b \). Let \( (v_x, v_y) \) be the first edge in \( P' \) where \( v_x \in V(T) \) and \( v_y \not\in V(T) \). Since \( (v_x, v_y) \) is the first such edge in \( P' \), path \( P' \) passes through \( v_x \) and then through edge \( (v_x, v_y) \). The length of \( P' \) is at least \( d(v_s, v_x) + \text{weight}(v_x, v_y) \).
Since \( e = (v_a, v_b) \) minimizes \( d(v_s, v_i) + \text{weight}(v_i, v_j) \) over all \( (v_i, v_j) \) where \( v_i \in T \) and \( v_j \not\in V(T) \),

\[
d(v_s, v_a) + \text{weight}(v_a, v_b) \leq d(v_s, v_x) + \text{weight}(v_x, v_y).
\]

Thus, the length of \( P' \) is greater than or equal to \( P \cup \{e\} \). Since this is true for any path \( P' \), path \( P \cup \{e\} \) is a shortest path to \( v_b \) and \( T \cup \{e\} \) is a subtree of a shortest path tree. \( \square \)
Running Time of Dijkstra’s Algorithm

DijkstraShortestPath(G, v_s)

1 $U \leftarrow V(G) - \{v_s\};$ /* $V(G)$ = set of vertices of graph $G$ */
2 $v_s$.parent $\leftarrow$ NULL;
3 $v_s$.distance $\leftarrow$ 0;
4 while $(U \neq \emptyset)$ and ($\exists$ edge from $(V(G) - U)$ to $U$) do
5 \hspace{1em} $(v_i, v_j) \leftarrow$ edge from $V(G) - U$ to $U$ which minimizes $v_i$.distance + weight($v_i, v_j$);
6 \hspace{1em} $v_j$.parent $\leftarrow v_i$;
7 \hspace{1em} $v_j$.distance $\leftarrow v_i$.distance + weight($v_i, v_j$);
8 \hspace{1em} $U \leftarrow U - \{v_j\}$;
9 end
Shortest Path Tree: Storing Cost at Vertices
Dijkstra’s Algorithm: Storing Costs at Vertices

DijkstraShortestPath(G, vs)

1. $U \leftarrow V(G)$; /* $V(G)$ = set of vertices of graph $G$ */
2. foreach $v_i \in V(G) - \{v_s\}$ do $v_i$.distance $\leftarrow \infty$;
3. $v_s$.distance $\leftarrow 0$;
4. $v_s$.parent $\leftarrow$ NULL;
5. while ($U \neq \emptyset$) and ($v_i$.distance $< \infty$ for some $v_i \in U$) do
6.      $v_j \leftarrow v_i \in U$ with minimum $v_i$.distance;
7.      $U \leftarrow U - \{v_j\}$; /* Remove $v_j$ from $U$ */
8.      /* $(v_j, v_j$.parent) is a shortest path edge */
9.     foreach edge $(v_j, v_k)$ incident on $v_j$ do
10.    newDist $\leftarrow v_j$.distance + weight($v_j, v_k$);
11.       if ($v_k \in U$ and newDist $< v_k$.distance) then
12.          $v_k$.parent $\leftarrow v_j$;
13.         $v_k$.distance $\leftarrow$ newDist;
14.     end
15. end
Dijkstra's Algorithm: Priority Queue of Vertices

DijkstraShortestPath(G, vs)

1. foreach $v_i \in V(G) - \{vs\}$ do $Q$.Insert($v_i, \infty$);
2. $Q$.Insert($vs$, 0); /* $Q$ is a priority queue of vertices */
3. $vs$.parent $\leftarrow$ NULL;
4. $vs$.distance $\leftarrow$ 0;
5. while $Q$.IsNotEmpty() and ($Q$.MinKey() $\neq \infty$) do
6.   $v_j \leftarrow Q$.DeleteMin(); /* ($v_j$, $v_j$.parent) is a shortest path edge */
7.   foreach edge ($v_j$, $v_k$) incident on $v_j$ do
8.     newDist $\leftarrow v_j$.distance + weight($v_j$, $v_k$);
9.     if ($Q$.Contains($v_k$) and newDist $< Q$.Key($v_k$)) then
10.        $v_k$.parent $\leftarrow v_j$;
11.        $Q$.DecreaseKey($v_k$, newDist);
12.        $v_k$.distance $\leftarrow$ newDist;
13.    end
14.  end
15. end
Maximum Flow
$c(u, v) = \text{capacity of edge } (u, v)$. 
Flow

\[ c(u, v) = \text{capacity of edge } (u, v). \]
\[ f(u, v) = \text{flow along edge } (u, v). \]
Capacity constraint: \( 0 \leq f(u, v) \leq c(u, v) \).

\( c(u, v) \) = capacity of edge \((u, v)\).

\( f(u, v) \) = flow along edge \((u, v)\).
Flow Conservation

Flow conservation: For all $u \in V(G) - \{s, t\}$,

flow in to $u = \text{flow out from } u$, or

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v).$$

$f(u, v) = \text{flow along edge } (u, v)$. 
A flow is a function $f : E(G) \rightarrow \mathbb{R}$ where

1. Capacity constraint: $0 \leq f(u, v) \leq c(u, v)$;
2. Flow conservation: $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$. 
The value of the flow, $|f|$, is the amount flowing out of node $s$:

$$|f| = \sum_{v \in V(G)} f(s, v).$$
Lemma:

flow out of node $s$ = flow in to node $t$, or

$$|f| = \sum_{v \in V(G)} f(s, v) = \sum_{v \in V(G)} f(v, t).$$
Max Flow Problem

![Graph Diagram]

The value of the flow, $|f|$, is the amount flowing out of node $s$:

$$|f| = \sum_{v \in V(G)} f(s, v).$$

Max flow problem: Given a flow network $G$, find the flow with the maximum value.
Flow

\[ s \xrightarrow{0/4} v_2 \xrightarrow{0/13} s, \quad s \xrightarrow{0/16} v_1 \xrightarrow{0/12} v_3 \xrightarrow{0/7} t, \quad v_1 \xrightarrow{0/9} v_2 \xrightarrow{0/14} v_4 \xrightarrow{0/4} t \]
Flow: Example 2
Flow: Example 2

\[ s \rightarrow v_2 \rightarrow v_1 \rightarrow v_3 \rightarrow t \]
\[ s \rightarrow v_4 \rightarrow v_3 \rightarrow t \]
\[ s \rightarrow v_4 \rightarrow v_2 \rightarrow v_1 \rightarrow t \]
Residual Capacity

Flow network (part):

Residual capacity:

\[ c_f(u, v) = \begin{cases} 
  c(u, v) - f(u, v) & \text{if } (u, v) \in E(G), \\
  f(v, u) & \text{if } (v, u) \in E(G), \\
  0 & \text{otherwise.}
\end{cases} \]
Residual Network

Flow network:

Residual network:
An **augmenting path** is a path from $s$ to $t$ in the residual network.
Residual Network

Flow network:

![Diagram of a flow network with nodes labeled s, v1, v2, v3, v4, t and edges with capacities labeled 11/16, 12/12, 0/9, 3/7, 7/14, 8/13, 4/4, and 15/20.]

Residual capacity:

\[
c_f(u, v) = \begin{cases} 
    c(u, v) - f(u, v) & \text{if } (u, v) \in E(G), \\
    f(v, u) & \text{if } (v, u) \in E(G), \\
    0 & \text{otherwise.}
\end{cases}
\]

Assume that \( E(G) \) never contains both \((u, v)\) and \((v, u)\).
What is the residual network?
What is the residual network?
Residual Network

$G$ is a directed graph where $E(G)$ never contains both $(u, v)$ and $(v, u)$.

$f$ is a flow in $G$.

Residual capacity:

$$c_f(u, v) = \begin{cases} 
  c(u, v) - f(u, v) & \text{if } (u, v) \in E(G), \\
  f(v, u) & \text{if } (v, u) \in E(G), \\
  0 & \text{otherwise}.
\end{cases}$$

$G_f$ is the residual network whose edges have capacities $c_f(u, v) > 0$. 

3.55
Ford-Fulkerson Max Flow Algorithm

\textbf{FFMaxFlow}(G)

1. \textbf{foreach} edge \((u, v) \in E(G)\) \textbf{do} \(f(u, v) \leftarrow 0\);
2. Compute residual network \(G_f\);
3. Search for path \(P\) in residual network \(G_f\);
4. \textbf{while} there exists a path \(P\) from \(s\) to \(t\) in \(G_f\) \textbf{do}
5. \hspace{1em} \(x \leftarrow \min\{c_f(u, v)|(u, v) \in P\}\);
6. \hspace{1em} Increase flow in \(G\) by \(x\) along path \(P\);
7. \hspace{1em} Compute residual network \(G_f\);
8. \hspace{1em} Search for path \(P\) in residual network \(G_f\);
9. \textbf{end}
Ford-Fulkerson Max Flow Algorithm (Detailed)

\[ \text{FFMaxFlow}(G) \]

1. \textbf{foreach} edge \((u, v) \in E(G)\) \textbf{do} \(f(u, v) \leftarrow 0;\)
2. Compute residual network \(G_f;\)
3. Search for path \(P\) in residual network \(G_f;\)
4. \textbf{while} there exists a path \(P\) from \(s\) to \(t\) in \(G_f\) \textbf{do}
5. \hspace{1em} \(x \leftarrow \min\{c_f(u, v)|(u, v) \in P\};\)
6. \hspace{2em} /* Increase flow in \(G\) by \(x\) along path \(P\) */
7. \hspace{1em} \textbf{foreach} edge \((u, v) \in P\) \textbf{do}
8. \hspace{2em} \hspace{1em} \textbf{if} \((u, v) \in E(G)\) \textbf{then} \(f(u, v) \leftarrow f(u, v) + x;\)
9. \hspace{2em} \hspace{1em} \textbf{else} \(f(v, u) \leftarrow f(v, u) - x;\)
10. \hspace{2em} /* \((v, u) \in E(G)\) */
11. \hspace{1em} \textbf{end}
12. Compute residual network \(G_f;\)
13. Search for path \(P\) in residual network \(G_f;\)
14. \textbf{end}
Max Flow Example

\[\text{3.58}\]
Cuts of Flow Networks
A cut \((S, T)\) of a flow network \(G\) is a partition of \(V(G)\) into \(S\) and \(T = V(G) - S\) such that \(s \in S\) and \(t \in T\).
A cut \((S, T)\) of a flow network \(G\) is a partition of \(V(G)\) into \(S\) and \(T = V(G) - S\) such that \(s \in S\) and \(t \in T\).

The capacity of the cut \((S, T)\) is:

\[
c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v).\]
Cut
The capacity of the cut \((S, T)\) is 
\[
c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v).
\]

A minimum cut of \(G\) is a cut whose capacity is minimum over all cuts of \(G\).
Minimum Cut

\[
\begin{array}{c}
s \\
\downarrow \ 5 \ \\
\downarrow \ 7 \\
v_1 \\
\uparrow \ 6 \\
\uparrow \ 4 \\
v_2 \\
\rightarrow \ 9 \\
\rightarrow \ 2 \\
v_3 \\
\rightarrow \ 8 \\
\rightarrow \ t \\
v \end{array}
\]

\[3.64\]
Minimum Cut

Graph:

- Source: $s$
- Sinks: $t$
- Vertices: $v_1, v_2, v_3, v_4$

Weights:
- $s$ to $v_2$: 4
- $v_2$ to $v_1$: 5
- $v_1$ to $v_3$: 7
- $v_3$ to $t$: 4
- $s$ to $v_4$: 3
- $v_2$ to $v_4$: 2
- $v_3$ to $v_4$: 8

Minimum Cut Weight: 3.65
Flows and Cuts

$|f| = \sum_{v \in V(G)} f(s, v)$ is the value of flow $f$.

$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$ is the capacity of cut $(S, T)$.

**Lemma** (Cut Lemma). For any flow $f$ and any cut $(S, T)$,

$$|f| \leq c(S, T).$$
Theorem. For any flow network $G$, 
max flow of $G = \text{min cut of } G$!
Max-flow min-cut theorem

Proof: The following three conditions are equivalent:

1. $f$ is a maximum flow of $G$.
2. There are no augmenting paths in the residual network $G_f$.
3. $|f| = c(S, T)$ for some cut $(S, T)$ of $G$. 
Max-flow min-cut theorem: (1) $\Rightarrow$ (2)

The following three conditions are equivalent:

1. $f$ is a maximum flow of $G$.
2. There are no augmenting paths in the residual network $G_f$.
3. $|f| = c(S, T)$ for some cut $(S, T)$ of $G$.

(1) $\Rightarrow$ (2): If $G_f$ had an augmenting path $P$, then we could increase $|f|$ by adding flow along $P$ to $f$. 
Max-flow min-cut theorem: (2) \Rightarrow (3)

The following three conditions are equivalent:

1. $f$ is a maximum flow of $G$.
2. There are no augmenting paths in the residual network $G_f$.
3. $|f| = c(S, T)$ for some cut $(S, T)$ of $G$.

(2) \Rightarrow (3): Assume $G_f$ has no augmenting path.
Let $S = \{v \in V(G) : \text{there is a path from } s \text{ to } v \text{ in } G_f\}$.
Let $T = V(G) - S$.

Since there is no edge in $G_f$ from any $u \in S$ to any $v \in T$:
- the flow from $S$ to $T$ is $\sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T)$;
- there is no flow from $T$ to $S$.

Thus $|f| = c(S, T)$.
The following three conditions are equivalent:

1. \( f \) is a maximum flow of \( G \).
2. There are no augmenting paths in the residual network \( G_f \).
3. \( |f| = c(S, T) \) for some cut \((S, T)\) of \( G \).

(3) \( \Rightarrow \) (1): Assume \( |f| = c(S, T) \).

By the cut lemma (slide 3.66), \( |f'| \leq c(S, T) \) for any flow \( f' \) in \( G \).

Thus, \( |f'| \leq c(S, T) = |f| \) so \( |f| \) is a maximum flow.
Ford-Fulkerson Max Flow Algorithm
Running Time Analysis
Ford-Fulkerson Max Flow Algorithm

\[ \text{FFMaxFlow}(G) \]

1. \textbf{foreach} edge \((u, v) \in E(G)\) \textbf{do} \(f(u, v) \leftarrow 0;\)
2. Compute residual network \(G_f;\)
3. Search for path \(P\) in residual network \(G_f;\)
4. \textbf{while} there exists a path \(P\) from \(s\) to \(t\) in \(G_f\) \textbf{do}
5. \hspace{2em} \(x \leftarrow \min\{c_f(u, v) | (u, v) \in P\};\)
6. \hspace{2em} Increase flow in \(G\) by \(x\) along path \(P;\)
7. \hspace{2em} Compute residual network \(G_f;\)
8. \hspace{2em} Search for path \(P\) in residual network \(G_f;\)
9. \textbf{end}
Ford-Fulkerson Max Flow: Time Analysis

**Lemma.** If all capacities are integers, then FFMaxFlow increases the flow value by a positive integer at each iteration.

\[ m = \# \text{ graph edges}. \]

**Proposition.** If all capacities are integers, then the Ford-Fulkerson Algorithm runs in \( O(m|f^*|) \) time where \( f^* \) is the max flow.
Multi-Source/Sink Max-Flow
Multiple Sources and Sinks

Sources: $s_1$ and $s_2$.
Sinks: $t_1$ and $t_2$.

Flow value $|f| = \sum_{s_i} \sum_{v_j} f(s_i, v_j)$. 

3.76
Reduction

Multi-Source/Sink Max-Flow Problem: Given a flow network $G$ with multiple sources and sinks, find max $|f|$ over all flows $f$ in $G$.

Single Source/Sink Max-Flow Problem: Given a flow network $G$ with one source and sink, find max $|f|$ over all flows $f$ in $G$.

Reduce the Multi-Source/Sink Max-Flow Problem to the Single Source Max Flow Problem.

Reduce $P$ to $Q$: Turn problem $P$ into $Q$ such that the solution to $Q$ gives the solution to $P$. 
Multi-Source/Sink Max-Flow Problem

Reduce Multi-Source/Sink Max-Flow Problem to Single Source/Sink Max-Flow Problem:
Multi-Source/Sink Max-Flow Problem

Reduce the Multi-Source/Sink Max-Flow Problem to the Single Source Max Flow Problem:

Let $G$ be a flow network with multiple sources $s_i$ and sinks $t_i$.

Create flow network $G'$ from $G$ with a single source and sink as follows:

- Add new source $s^*$ and new sink $t^*$;
- Add directed edges from $s^*$ to each $s_i$. Set capacity of each new edge to $\infty$.
- Add directed edges from each $t_i$ to $t^*$. Set capacity of each new edge to $\infty$.

$G'$ has flow with value $F$ from $s^*$ to $t^*$ if and only if $G$ has flow with value $F$ from the $s_i$ to the $t_i$. 