Warm up:
Show that no cycle crosses a cut exactly once
no cycle crosses a cut exactly once

- Assume the cycle crosses the cut once
- Consider some edge \((u, v)\) in the cycle which crosses the cut
- If we remove \((u, v)\) then there is still a path from \(u\) to \(v\) which must somewhere cross the cut
Today’s Keywords

• Graphs
• Minimum Spanning Tree
• Prim’s Algorithm
• Shortest path
• Dijkstra’s Algorithm
• Breadth-first search
• Chapter 22
• Chapter 23
Homeworks

• HW7 due Thursday, November 14 @ 11pm
  – Written (use latex)
  – Graphs!
• HW10B also due Thursday, November 14 @ 11pm
  – No late submissions allowed
• Reminder: I will not have office hours Monday
  – Tuesday 11-1 instead
Definition: \( G = (V, E) \)

\( w(e) = \text{weight of edge } e \)

\[ V = \{A, B, C, D, E, F, G, H, I\} \]

\[ E = \{(A, B), (A, C), (B, C), \ldots\} \]
Definition: Path

A sequence of nodes \((v_1, v_2, ..., v_k)\)

s.t. \(\forall 1 \leq i \leq k - 1, (v_i, v_{i+1}) \in E\)

Simple Path:
A path in which each node appears at most once

Cycle:
A path of > 2 nodes in which \(v_1 = v_k\)
Definition: Minimum Spanning Tree

A Tree $T = (V_T, E_T)$ which connects ("spans") all the nodes in a graph $G = (V, E)$, that has minimal cost

$$\text{Cost}(T) = \sum_{e \in E_T} w(e)$$

How many edges does $T$ have? $V - 1$
Definition: Cut

A Cut of graph $G = (V, E)$ is a partition of the nodes into two sets, $S$ and $V - S$

Edge $(v_1, v_2) \in E$ crosses a cut if $v_1 \in S$ and $v_2 \in V - S$ (or opposite), e.g. $(A, C)$

A set of edges $R$ Respects a cut if no edges cross the cut e.g. $R = \{(A, B), (E, G), (F, G)\}$
Consider any cut \((S, V - S)\) in a graph \(G = (V, E)\), the minimum weight edge crossing that cut is in some MST of \(G\).
Consider any cycle in a graph $G = (V, E)$, the maximum weight edge on that cycle is not in some MST of $G$

What is our strategy?

Assume we have a MST Already:

2 cases:

1. Tree does not have max weight edge
2. Tree has max weight edge
Consider any cycle $c = (v_1, v_2, \ldots, v_k, v_1)$ in a graph $G = (V, E)$, the maximum weight edge $e$ on that cycle is not in some MST of $G$.

Consider some MST $T$, Case 1: (the easy case) If $e \notin T$ Then claim holds.
Consider any cycle $c = (v_1, v_2, \ldots v_k, v_1)$ in a graph $G = (V, E)$, the maximum weight edge $e$ on that cycle is not in some MST of $G$

Consider some MST $T$, Case 2:

Consider if $e = (v_1, v_2) \in T$

Let $(S, V - S)$ be a cut which $e$ crosses

There is some other edge $e'$ not in $T$ which crosses $(S, V - S)$

Build tree $T'$ by exchanging $e'$ for $e$
Consider any cycle \( c = (v_1, v_2, \ldots v_k, v_1) \) in a graph \( G = (V, E) \), the maximum weight edge \( e \) on that cycle is *not* in *some* MST of \( G \)

Consider some MST \( T \),
Case 2:

If \( e = (v_1, v_2) \in T \)

\( T' = T \) with edge \( e' \) instead of \( e \)

We assumed \( w(e) \geq w(e') \)

\( w(T') = w(T) - w(e) + w(e') \)

\( w(T') \leq w(T) \)

So \( T' \) is also a MST!

Thus the claim holds
Prim’s Algorithm

Start with an empty tree $A$
Pick a start node
Repeat $V - 1$ times:
   Add the min-weight edge which connects to node in $A$ with a node not in $A$
Prim’s Algorithm

Start with an empty tree $A$
Pick a start node
Repeat $V - 1$ times:
   Add the min-weight edge which connects to node in $A$ with a node not in $A$
Prim’s Algorithm

Start with an empty tree $A$
Pick a start node
Repeat $V - 1$ times:
    Add the min-weight edge which connects to node in $A$ with a node not in $A$
Prim’s Algorithm

Start with an empty tree \( A \)
Pick a start node
Repeat \( V - 1 \) times:
Add the min-weight edge which connects to node in \( A \) with a node not in \( A \)
Prim’s Algorithm

Start with an empty tree $A$

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in $A$ with a node not in $A$

Keep edges in a Heap

$O(E \log V)$
Prim's Algorithm

Initialize $d_v = \infty$ for each node $v$

Keep a priority queue $PQ$ of nodes, using $d_v$ as key

Pick a start node $s$, set $d_s = 0$

While $PQ$ is not empty:

$v = PQ.extractmin()$

for each $u \in V$ s.t. $(v, u) \in E$:

$PQ.decreaseKey(u, \min(d_u, w(v, u)))$

$d_u$ is the cost to add node $u$ to the tree with only one edge.
Prim’s Algorithm

Initialize \( d_v = \infty \) for each node \( v \)
Keep a priority queue \( PQ \) of nodes, using \( d_v \) as key
Pick a start node \( s \), set \( d_s = 0 \)
While \( PQ \) is not empty:

\[ v = PQ.\text{extractmin}() \]
for each \( u \in V \) s.t. \((v,u) \in E\):

\[ PQ.\text{decreaseKey}(u, \min(d_u, w(v,u))) \]
Prim’s Algorithm

Initialize $d_v = \infty$ for each node $v$

Keep a priority queue $PQ$ of nodes, using $d_v$ as key

Pick a start node $s$, set $d_s = 0$

While $PQ$ is not empty:

$v = PQ.extractmin()$

for each $u \in V$ s.t. $(v, u) \in E$:

$PQ.decreaseKey(u, \min(d_u, w(v, u)))$
Prim’s Algorithm

Initialize $d_v = \infty$ for each node $v$
Keep a priority queue $PQ$ of nodes, using $d_v$ as key
Pick a start node $s$, set $d_s = 0$
While $PQ$ is not empty:
  $V$ loops
  $v = PQ.extractMin()$ \( O(\log V) \)
  for each $u \in V$ s.t. $(v, u) \in E$: \( E \) times total
    $PQ.decreaseKey(u, \min(d_u, w(v, u)))$ \( O(\log V) \)

$O(E \log V + V \log V)$
Single-Source Shortest Path

Find the quickest way to get from UVA to each of these other places

Given a graph $G = (V, E)$ and a start node $s \in V$, for each $v \in V$ find the least-weight path from $s \to v$ (call this weight $\delta(s, v)$)

(assumption: all edge weights are positive)
Dijkstra’s Algorithm

Given some start node \( s \)
Start with an empty tree \( A \)
Repeat \( V - 1 \) times:
   Add the “nearest” node not yet in \( A \)
Dijkstra’s Algorithm

Given some start node $s$
Start with an empty tree $A$
Repeat $V - 1$ times:
  Add the “nearest” node not yet in $A$
Dijkstra’s Algorithm

Given some start node $s$
Start with an empty tree $A$
Repeat $V − 1$ times:
    Add the “nearest” node not yet in $A$
Dijkstra’s Algorithm

Given some start node $s$
Start with an empty tree $A$
Repeat $V - 1$ times:
Add the “nearest” node not yet in $A$

VERY similar to Prim’s!
Prim’s Algorithm

Initialize \( d_v = \infty \) for each node \( v \)
Keep a priority queue \( PQ \) of nodes, using \( d_v \) as key
Pick a start node \( s \), set \( d_s = 0 \)
While \( PQ \) is not empty:
   \[ v = PQ\text{-extract}\text{min}() \]
   for each \( u \in V \) s.t. \( (v, u) \in E \):
   \[ PQ\text{-decreaseKey}(u, \min(d_u, w(v, u))) \]
Dijkstra’s Algorithm

Initialize $d_v = \infty$ for each node $v$
Keep a priority queue $PQ$ of nodes, using $d_v$ as key
Pick a start node $s$, set $d_s = 0$
While $PQ$ is not empty: 

- $v = PQ.extractmin()$ \( O(\log V) \)
- for each $u \in V$ s.t. $(v, u) \in E$: \( E \) times total
  - $PQ.decreaseKey(u, \min(d_u, d_v + w(v, u)))$ \( O(\log V) \)

$O(E \log V + V \log V)$
Dijkstra’s Algorithm Proof Strategy

• Proof by induction
• Idea: show that when node $u$ is removed from the priority queue, $d_u = \delta(s, u)$
  – Claim 1: when $u$ is removed from the queue, $d_u \geq \delta(s, u)$
    • i.e. $d_u$ is at least the length of the shortest path
  – Claim 2: if we consider any path $(s, \ldots, u)$, $w(s, \ldots, u) \geq d_u$
    • i.e. $d_u$ is no longer than any other path from $s$ to $u$, including the shortest one
Proof of Dijkstra’s

• Base case:
  \[-i = 0, u = v_1 = s, \delta(s, v_1) = 0\]

• Assume that nodes \(v_1 = s, \ldots, v_i\) have been removed from \(PQ\) already, and for each of them \(d_{v_i} = \delta(s, v_i)\)

• Let node \(u\) be the \((i + 1)^{th}\) node extracted
Proof of Dijkstra’s: Claim 1

• Let node $u$ be the $(i + 1)^{th}$ node extracted

• Claim 1: $d_u \geq \delta(s, u)$
  
  • Proof: node $u$ has a path of weight $d_u$ from $s$
    – Discovering a path was how we updated the key!
  
  • Since $d_u$ is the weight of SOME path, its weight is at least that of the **SHORTEST** path
Proof of Dijkstra’s: Claim 2

- Let node $u$ be the $(i + 1)^{th}$ node extracted
- for any path $(s, ..., u), w(s, ..., u) \geq d_u$
- Extracted nodes define a cut of the graph
- Let edge $(x, y)$ be the last edge in this path which crosses the cut
Proof of Dijkstra’s: Claim 2

• Let node $u$ be the $(i + 1)^{th}$ node extracted
• for any path $(s, ..., u)$, $w(s, ..., u) \geq d_u$
• Extracted nodes define a cut of the graph
• Let edge $(x, y)$ be the last edge in this path which crosses the cut
Proof of Dijkstra’s: Claim 2

• Let node $u$ be the $(i + 1)^{th}$ node extracted
• for any path $(s, ..., u), w(s, ..., u) \geq d_u$
• Extracted nodes define a cut of the graph
• Let edge $(x, y)$ be the last edge in this path which crosses the cut

By definition

\[ w(s, ..., u) \geq \delta(s, x) + w(x, y) + w(y, ..., u) \]

\[ \geq d_y + w(y, ..., u) \]

\[ \geq d_u + w(y, ..., u) \]

\[ \geq d_u \]

No negative edge weights

Because otherwise, $u$ would not be next extracted

We updated $y$’s key $d_y$
when we extracted $x$ if
\[ d_x + w(x, y) < d_y \]
Proof of Dijkstra’s: Finale

• Claim 1: \( d_u \geq \delta(s, u) \)

• Claim 2: \( d_u \leq w(s, ..., u) \) for any path from \( s \) to \( u \) (including the shortest one)

• 1&2 Together: \( w(s, ..., u) \geq d_u \geq \delta(s, u) \)
  - therefore \( \delta(s, u) \geq d_u \geq \delta(s, u) \)
  - \( d_u = \delta(s, u) \)
Breadth-First Search

- Input: a node $s$
- Behavior: Start with node $s$, visit all neighbors of $s$, then all neighbors of neighbors of $s$, ...
- Output: lots of choices!
  - Is the graph connected?
  - Is there a path from $s$ to $u$?
  - Shortest number of “hops” from $s$ to $u$

Sounds like Dijkstra’s!
Dijkstra’s Algorithm

Initialize $d_v = \infty$ for each node $v$
Keep a priority queue $PQ$ of nodes, using $d_v$ as key
Pick a start node $s$, set $d_s = 0$
While $PQ$ is not empty:

$v = PQ\.extract\text{min}()$
for each $u \in V$ s.t. $(v, u) \in E$: 

$PQ\.decrease\text{Key}(u, \min(d_u, d_v + w(v, u)))$
BFS

Keep a queue $Q$ of nodes

Pick a start node $s$

$Q.enqueue(s)$

While $Q$ is not empty:

$v = Q.dequeue()$

for each “unvisited” $u \in V$ s.t. $(v, u) \in E$:

$Q.enqueue(u)$
BFS: Shortest “Hops” Path

Keep a queue $Q$ of nodes
Pick a start node $s$
$Q\.enqueue(s)$
hops = 0
While $Q$ is not empty:
    $v = Q\.dequeue()$
    hops += 1
for each “unvisited” $u \in V$ s.t. $(v, u) \in E$:
    $u\.hops = hops$
    $Q\.enqueue(u)$