Minimum Spanning Trees and Prim’s Algorithm

Version of September 23, 2016
Outline

- Spanning trees and minimum spanning trees (MST).
- Tools for solving the MST problem.
- Prim’s algorithm for the MST problem.
  - The idea
  - The algorithm
  - Analysis
Spanning Trees

Definition

A subgraph $T$ of a undirected graph $G = (V, E)$ is a spanning tree of $G$ if it is a tree and contains every vertex of $G$.

Example

Graph

spanning tree 1

spanning tree 2

spanning tree 3
Theorem

*Every connected graph has a spanning tree.*

Question

Why is this true?

Question

Given a connected graph $G$, how can you find a spanning tree of $G$?
Weighted Graphs

Definition

A **weighted graph** is a graph, in which each edge has a **weight** (some real number) Could denote length, time, strength, etc.

Example

![Weighted Graph Diagram]

Definition

**Weight of a graph**: The sum of the weights of all edges
A **Minimum spanning tree (MST)** of an undirected connected weighted graph is a spanning tree of **minimum weight** (among all spanning trees).

**Example**

- **weighted graph**
- **Tree 1. w=74**
- **Tree 2, w=71**
- **Tree 3, w=72**
The minimum spanning tree may not be unique

Note: if the weights of all the edges are distinct, MST is provably unique (proof will follow from later results).
Definition (MST Problem)

Given a connected weighted undirected graph $G$, design an algorithm that outputs a minimum spanning tree (MST) of $G$. 

Version of September 23, 2016
Outline

- **Spanning trees** and minimum spanning trees (MST).
- **Tools for solving the MST problem.**
- **Prim’s algorithm** for the MST problem.
  - The idea
  - The algorithm
  - Analysis
A tree is an **acyclic** graph

1. start with an **empty** graph
2. try to **add** edges one at a time, subject to not creating a cycle
3. if after adding each edge we are sure that the resulting graph is a **subset** of some minimum spanning tree, then, after \( n - 1 \) steps we are done.

Hard part is ensuring (3)!
Generic Algorithm for MST problem

Definition

Let $A$ be a set of edges such that $A \subseteq T$, where $T$ is some MST. Edge $(u, v)$ is safe edge for $A$, if $A \cup \{(u, v)\}$ is also a subset of some MST.

- If at each step, we can find a safe edge $(u, v)$, we can grow a MST.

Generic-MST($G$, $w$)

begin
  $A$ = EMPTY;
  while $A$ does not form a spanning tree do
    find an edge $(u, v)$ that is safe for $A$;
    add $(u, v)$ to $A$;
  end
  return $A$
end
Some Definitions

Definition
Let $G = (V, E)$ be a connected and undirected graph. A cut $(S, V - S)$ of $G$ is a partition of $V$.

Example

Definition
An edge $(u, v) \in E$ crosses the cut $(S, V - S)$ if one of its endpoints is in $S$, and the other is in $V - S$. A cut respects a set $A$ of edges if no edge in $A$ crosses the cut. An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut.
How to Find a Safe Edge?

**Lemma**

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$

Let $A$ be a subset of $E$ that is included in some minimum spanning tree for $G$.

Let

- $(S, V - S)$ be *any* cut of $G$ that respects $A$
- $(u, v)$ be a light edge crossing the cut $(S, V - S)$

Then, edge $(u, v)$ is *safe* for $A$.

This implies we can find a safe edge by:

1. first finding a cut that respects $A$,
2. then finding a light edge crossing that cut.

That light edge is a safe edge.
Let \( A \subseteq T \), where \( T \) is a MST.

Case 1: \((u, v) \in T\)

- \( A \cup \{(u, v)\} \subseteq T \).
- Hence \((u, v)\) is safe for \( A \).
Case 2: \((u, v) \notin T\)

- **Idea:** construct another MST \(T'\) s.t. \(A \cup \{(u, v)\} \subseteq T'\).
- Consider the unique path \(P\) in \(T\) from \(u\) to \(v\).
- Since \(u\) and \(v\) are on opposite sides of the cut \((S, V - S)\),
  - There is at least one edge in \(P\) that crosses the cut.
  - Let \((x, y)\) be such an edge.
- Since the cut respects \(A\), \((x, y) \notin A\).
- Since \((u, v)\) is a light edge crossing the cut, we have \(w(u, v) \leq w(x, y)\).
Adding \((u, v)\) to \(T\), creates a cycle with \(P\). Removing any edge from this cycle gives a tree again. In particular, adding \((u, v)\) and removing \((x, y)\) creates a new tree \(T'\).

The weight of \(T'\) is

\[
w(T') = w(T) - w(x, y) + w(u, v) \\ \leq w(T)
\]

Since \(T\) is a MST, \(W(T) \leq W(T')\) so \(W(T') = W(T)\) and \(T\) is also an MST.

But \(A \cup \{(u, v)\} \subseteq T'\), so \((u, v)\), is safe for \(A\).

The Lemma is proved.
Spanning trees and minimum spanning trees (MST).

Tools for solving the MST problem.

Prim's algorithm for the MST problem.

- The idea
- The algorithm
- Analysis
The generic algorithm gives us an idea how to ’grow’ a MST.

- If you read the theorem and proof carefully, you will notice that the choice of a cut (and hence a corresponding light edge) in each iteration is arbitrary.
- We can select any cut (that respects current edge set $A$) and find a light edge crossing that cut to proceed.
- Different ways of choosing cuts correspond to different algorithms.
- The two major ones are Prim’s algorithm and Kruskal’s algorithm,
Prim’s Algorithm

Prim’s algorithm
- grows a tree, adding a new light edge in each iteration, creating a new tree.

Growing a tree
- Start by picking any vertex $r$ to be the root of the tree.
- While the tree does not contain all vertices in the graph: find shortest edge leaving tree and add it to the tree.

We will show that these steps can be implemented in total $O(E \cdot \log V)$. 
Step 0:
- Choose any element $r$; set $S = \{r\}$ and $A = \emptyset$.
- (Take $r$ as the root of our spanning tree.)

Step 1:
- **Find a lightest edge** such that one endpoint is in $S$ and the other is in $V \setminus S$.
- **Add** this edge to $A$ and its (other) endpoint to $S$.

Step 2:
- If $V \setminus S = \emptyset$, then stop and output (minimum) spanning tree $(S, A)$; Otherwise, go to Step 1.
Connected graph

Step 0

\[ S = \{a\} \]

\[ V \setminus S = \{b, c, d, e, f, g\} \]

lightest edge = \{a, b\}
Step 1.1 before
S={a}
V \ S = \{b,c,d,e,f,g\}
A={} 
lighest edge = \{a,b\}

Step 1.1 after
S={a,b}
V \ S = \{c,d,e,f,g\}
A={\{a,b\}} 
lighest edge = \{b,d\}, \{a,c\}
Step 1.2 before
S={a,b}
V \ S = \{c,d,e,f,g\}
A=\{\{a,b\}\}
lighest edge = \{b,d\}, \{a,c\}

Step 1.2 after
S={a,b,d}
V \ S = \{c,e,f,g\}
A=\{\{a,b\},\{b,d\}\}
lighest edge = \{d,c\}
Step 1.3 before
S={a,b,d}
V \ S = \{c,e,f,g\}
A=\{\{a,b\}, \{b,d\}\}
lighest edge = \{d,c\}

Step 1.3 after
S={a,b,c,d}
V \ S = \{e,f,g\}
A=\{\{a,b\}, \{b,d\}, \{c,d\}\}
lighest edge = \{c,f\}
Prim’s Example – Continued

Step 1.4 before
\(S = \{a, b, c, d\}\)
\(V \setminus S = \{e, f, g\}\)
\(A = \{\{a, b\}, \{b, d\}, \{c, d\}\}\)
lightest edge = \{c, f\}

Step 1.4 after
\(S = \{a, b, c, d, f\}\)
\(V \setminus S = \{e, g\}\)
\(A = \{\{a, b\}, \{b, d\}, \{c, d\}, \{c, f\}\}\)
lightest edge = \{f, g\}
Prim’s Example – Continued

Step 1.5 before
S = \{a, b, c, d, f\}
V \setminus S = \{e, g\}
A = \{\{a, b\}, \{b, d\}, \{c, d\}, \{c, f\}\}
lighest edge = \{f, g\}

Step 1.5 after
S = \{a, b, c, d, f, g\}
V \setminus S = \{e\}
A = \{\{a, b\}, \{b, d\}, \{c, d\}, \{c, f\}, \{f, g\}\}
lighest edge = \{f, e\}
Step 1.6 before
S={a,b,c,d,f,g}
V \ S = \{e\}
A=\{\{a,b\},\{b,d\},\{c,d\},\{c,f\}, \{f,g\}\}
lighest edge = \{f,e\}

Step 1.6 after
S={a,b,c,d,e,f,g}
V \ S = \{\}
A=\{\{a,b\},\{b,d\},\{c,d\},\{c,f\}, \{f,g\},\{f,e\}\}
MST completed
Outline

- **Spanning trees** and minimum spanning trees (MST).
- Strategy for solving the MST problem.
- **Prim’s algorithm** for the MST problem.
  - The idea
  - The algorithm
  - Analysis
Recall Idea of Prim’s Algorithm

Step 0: Choose any element \( r \) and set \( S = \{ r \} \) and \( A = \emptyset \). (Take \( r \) as the root of our spanning tree.)

Step 1: Find a lightest edge such that one endpoint is in \( S \) and the other is in \( V \setminus S \). Add this edge to \( A \) and its (other) endpoint to \( S \).

Step 2: If \( V \setminus S = \emptyset \), then stop and output the minimum spanning tree \((S, A)\); Otherwise go to Step 1.

Questions

1. Why does this produce a minimum spanning tree?
2. How does the algorithm find the lightest edge and update \( A \) efficiently?
3. How does the algorithm update \( S \) efficiently?
How does the algorithm update $S$ efficiently?

**Answer:** Color the vertices.
- Initially all are white.
- Change the color to black when the vertex is moved to $S$.
- Use $\text{color}[v]$ to store color.

How does the algorithm find a lightest edge and update $A$ efficiently?

**Answer:**
1. Use a priority queue to find the lightest edge.
2. Use $\text{pred}[v]$ to update $A$. 
Reviewing Priority Queues

Priority Queue is a data structure
  - can be implemented as a heap

Supports the following operations:

*Insert*(\(u,\text{key}\)): Insert \(u\) with the key value \(\text{key}\) in \(Q\).
\(u = \text{Extract-Min}()\): Extract the item with minimum key value.
*Decrease-Key*(\(u,\text{new-key}\)): Decrease \(u\)'s key value to \(\text{new-key}\).

**Remark**: We already saw how to implement Insert and Extract-Min (and Delete) in \(O(\log |Q|)\) time.
Same ideas can also be used to implement Decrease-Key in \(O(\log |Q|)\) time.
Alternatively, can implement Decrease-Key using Delete followed by Insert.
Using a Priority Queue to Find the Lightest Edge

Each item of the queue is a pair \((u, \text{key}[u])\), where

- \(u\) is a vertex in \(V \setminus S\),
- \(\text{key}[u]\) is the weight of the lightest edge from \(u\) to any vertex in \(S\). (The endpoint of this edge in \(S\) is stored in \(\text{pred}[u]\), which is used to build the MST tree.)

\[
\begin{align*}
\text{key}[f] &= 8, \quad \text{pred}[f] = e \\
\text{key}[i] &= \infty, \quad \text{pred}[i] = \text{nil} \\
\text{key}[g] &= 16, \quad \text{pred}[g] = c \\
\text{key}[h] &= 24, \quad \text{pred}[h] = b \\
\end{align*}
\]

\(\rightarrow\) \(f\) has the minimum key

After adding the new edge and vertex \(f\), update the \(\text{key}[v]\) and \(\text{pred}[v]\) for each vertex \(v\) adjacent to \(f\).
begin
    \textbf{foreach} \( u \in V \) \textbf{do}
    \begin{align*}
    \text{color}[u] &= \text{WHITE}; \text{key}[u] = +\infty; \quad // \text{initialize} \\
    \end{align*}
end

\text{key}[r] = 0; \text{pred}[r] = \text{NIL}; \quad // \text{start at root}

Q = \text{new PriQueue}(V); \quad // \text{put vertices in } Q

\textbf{while} Q \text{ is nonempty} \textbf{do}
    \begin{align*}
    u &= \text{Q.Extract-Min}(); \quad // \text{lightest edge} \\
    \textbf{foreach} \ v \in \text{adj}[u] \textbf{do}
    \begin{align*}
    \textbf{if} & \ (\text{color}[v] = \text{WHITE}) \land (w[u, v] < \text{key}[v]) \textbf{then} \\
    \text{key}[v] &= w[u, v]; \quad // \text{new lightest edge} \\
    \text{Q.Decrease-Key}(v, \text{key}[v]); \quad \\
    \text{pred}[v] &= u; \\
    \end{align*}
    \end{align*}
end

\text{color}[u] = \text{BLACK};
end
end
When the algorithm terminates, $Q = \emptyset$ and the MST is

$$T = \{ \{v, \text{pred}[v]\} : v \in V \setminus \{r\} \}.$$ 

- The pred pointers define the MST as an inverted tree rooted at $r$. 
Example for Running Prim’s Algorithm

**Diagram:**

```
  a-----3-----e
 /     |     /
1      |      10
 /     |     /
  b-----3-----d
    |     |    |
    2-----4    1
    |     |    |
    3-----5    |
    |     |    |
    c-----4    f
```

**Table:**

<table>
<thead>
<tr>
<th>u</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>key[u]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pred[u]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**

- Version of September 23, 2016
- Minimum Spanning Trees and Prim’s Algorithm
**Outline**

- **Spanning trees** and minimum spanning trees (MST).
- Strategy for solving the MST problem.
- **Prim’s algorithm** for the MST problem.
  - The idea
  - The algorithm
  - Analysis
begin
    foreach $u \in V$ do
        $key[u] = +\infty; color[u] = \text{WHITE};$ // $O(V)$
    end

    $key[r] = 0; pred[r] = \text{NIL};$
    $Q = \text{new PriQueue}(V);$ // $O(V)$

while $Q$ is nonempty do
    $u = Q.\text{Extract-Min}();$ // Do this for each vertex
    foreach $v \in \text{adj}[u]$ do
        // Do the following for each edge twice
        if ($color[v] = \text{WHITE}) && (w[u, v] < key[v])$ then
            $key[v] = w[u, v]; pred[v] = u;$
            $Q.\text{Decrease-Key}(v, key[v]);$ // This is bottleneck
        end
    end

    color[u] = \text{BLACK};
end
end
The data structure \textit{PriQueue} (heap) supports the following two operations:

- \( O(|V|) \) for creating new Priority Queue
- \( O(\log V) \) for \textbf{Extract-Min} on a PriQueue of size at most \( V \).
  
  Total cost: \( O(V \log V) \)

- \( O(\log V) \) time for \textbf{Decrease-Key} on a PriQueue of size at most \( V \).
  
  Total cost: \( O(E \log V) \).

Total cost is then \( O((V + E) \log V) = O(E \log V) \)
A more advanced Priority Queue implementation called *Fibonacci Heaps* allow

- $O(1)$ for inserting each item
- $O(\log |V|)$ for Extract-Min
- $O(1)$ (amortized) for each Decrease-Key

Since algorithm performs $|V|$ Inserts, $|V|$ Extract-Mins and at most $E$ Decrease-Keys this leads to a $O(|E| + |V| \log |V|)$ algorithm, improving upon the $O(E \log V)$ more naive implementation.