CS 253: Algorithms

Chapter 23

Minimum Spanning Tree
Minimum Spanning Trees

- **Spanning Tree**
  - A tree (i.e., connected, acyclic graph) which contains all the vertices of the graph

- **Minimum Spanning Tree**
  - Spanning tree with the **minimum sum of weights**

- **Spanning forest**
  - If a graph is not connected, then there is a spanning tree for each connected component of the graph
Applications of MST

- Find the least expensive way to connect a set of cities, terminals, computers, etc.
Example

Problem
- A town has a set of houses and a set of roads
- A road connects 2 and only 2 houses
- A road connecting houses \( u \) and \( v \) has a repair cost \( w(u, v) \)

Goal: Repair enough (and no more) roads such that:

1. Everyone stays connected
   i.e., can reach every house from all other houses
2. Total repair cost is minimum
Minimum Spanning Trees

- A connected, undirected graph:
  
  Vertices = houses,       Edges = roads

- A **weight** \( w(u, v) \) on each edge \((u, v) \in E\)

Find \( T \subseteq E \) such that:

1. \( T \) connects all vertices
2. \( w(T) = \sum_{(u,v) \in T} w(u, v) \) is minimized
Properties of Minimum Spanning Trees

- Minimum spanning tree is **not** unique

- MST has no cycles (by definition): 

- # of edges in a MST: $|V| - 1$
Prim’s Algorithm

- Starts from an arbitrary “root”: $V_A = \{a\}$
- At each step:
  - Find a **light edge** crossing $(V_A, V - V_A)$
  - Add this edge to set $A$ (The edges in set $A$ always form a single tree)
  - Repeat until the tree spans all vertices
How to Find Light Edges Quickly?

Use a priority queue $Q$:

- Contains vertices not yet included in the tree, i.e., $(V - V_A)$
  - $V_A = \{a\}$, $Q = \{b, c, d, e, f, g, h, i\}$
- We associate a key with each vertex $v$:
  \[
  \text{key}[v] = \text{minimum weight of any edge } (u, v) \text{ connecting } v \text{ to } V_A
  \]

**Key**[$a$] = min($w_1, w_2$)
After adding a new node to $V_A$ we update the weights of all the nodes adjacent to it.

e.g., after adding $a$ to the tree, $Key[b]=4$ and $Key[h]=8$

$Key[v] = \infty$ if $v$ is not adjacent to any vertices in $V_A$
Example

Q = \{a, b, c, d, e, f, g, h, i\}  
V_A = \emptyset  
Extract-MIN(Q) \Rightarrow a

key [b] = 4 \quad \pi [b] = a  
key [h] = 8 \quad \pi [h] = a

Q = \{b, c, d, e, f, g, h, i\}  
V_A = \{a\}  
Extract-MIN(Q) \Rightarrow b
Example

$Q = \{c, d, e, f, g, h, i\}$ $V_A = \{a, b\}$
key $[c] = 8$ $\pi [c] = b$
key $[h] = 8$ $\pi [h] = a$ - unchanged

Extract-MIN($Q$) $\Rightarrow$ c

$Q = \{d, e, f, g, h, i\}$ $V_A = \{a, b, c\}$
key $[d] = 7$ $\pi [d] = c$
key $[f] = 4$ $\pi [f] = c$
key $[i] = 2$ $\pi [i] = c$

7 $\infty$ 4 $\infty$ 8 2
Extract-MIN($Q$) $\Rightarrow$ i
Example

\[ Q = \{d, e, f, g, h\} \quad V_A = \{a, b, c, i\} \]

key \[ [h] = 7 \quad \pi [h] = i \]

key \[ [g] = 6 \quad \pi [g] = i \]

\[ 7 \infty 4 6 8 \]

Extract-MIN(Q) \[ \Rightarrow f \]

\[ Q = \{d, e, g, h\} \quad V_A = \{a, b, c, i, f\} \]

key \[ [g] = 2 \quad \pi [g] = f \]

key \[ [d] = 7 \quad \pi [d] = c \text{ unchanged} \]

key \[ [e] = 10 \quad \pi [e] = f \]

\[ 7 10 2 8 \]

Extract-MIN(Q) \[ \Rightarrow g \]
Example

$$Q = \{d, e, h\} \quad V_A = \{a, b, c, i, f, g\}$$

key $[h] = 1 \quad \pi [h] = g$

$7 \ 10 \ 1$

Extract-MIN(Q) $\Rightarrow h$

$$Q = \{d, e\} \quad V_A = \{a, b, c, i, f, g, h\}$$

$7 \ 10$

Extract-MIN(Q) $\Rightarrow d$
Example

$Q = \{e\} \quad V_A = \{a, b, c, i, f, g, h, d\}$

$\text{key} \ [e] = 9 \quad \pi \ [e] = d$

$9$

Extract-MIN$(Q) \Rightarrow e$

$Q = \emptyset \quad V_A = \{a, b, c, i, f, g, h, d, e\}$
\textbf{PRIM}(V, E, w, r)

1. \( Q \leftarrow \emptyset \)
2. \textbf{for} each \( u \in V \)
3. \hspace{1cm} \textbf{do} \( \text{key}[u] \leftarrow \infty \)
4. \hspace{1cm} \pi[u] \leftarrow \text{NIL}
5. \hspace{1cm} \text{INSERT}(Q, u)
6. \hspace{1cm} \text{DECREASE-KEY}(Q, r, 0) \quad \% \text{key}[r] \leftarrow 0
7. \textbf{while} \( Q \neq \emptyset \)
8. \hspace{1cm} \textbf{do} \( u \leftarrow \text{EXTRACT-MIN}(Q) \)
9. \hspace{2cm} \textbf{for} each \( v \in \text{Adj}[u] \)
10. \hspace{3cm} \textbf{do} if \( v \in Q \) and \( w(u, v) < \text{key}[v] \)
11. \hspace{4cm} \textbf{then} \( \pi[v] \leftarrow u \)
12. \hspace{1cm} \text{DECREASE-KEY}(Q, v, w(u, v))

\textbf{Total time:} \( O(V\lg V + E\lg V) = O(E\lg V) \)

\( O(V) \) if \( Q \) is implemented as a \textit{min-heap}

\( O(\lg V) \)

\( \text{O}(V) \) \text{min-heap operations:} \( \text{O}(V\lg V) \)

\( \text{O}(E) \text{ times total} \)

\( \text{Constant} \)

\( \text{O}(\lg V) \)

\( \text{O}(E\lg V) \)
Prim’s Algorithm

- Total time: \( O(E \lg V) \)

- Prim’s algorithm is a “greedy” algorithm
  - Greedy algorithms find solutions based on a sequence of choices which are “locally” optimal at each step.

- Nevertheless, Prim’s greedy strategy produces a globally optimum solution!
Kruskal’s Algorithm

- How is it different from Prim’s algorithm?
  - **Prim’s algorithm** grows one tree all the time
  - **Kruskal’s algorithm** grows multiple trees (i.e., a forest) at the same time.
  - Trees are merged together using **safe** edges
  - Since an MST has exactly $|V| - 1$ edges, after $|V| - 1$ merges, we would have only one component
Kruskal’s Algorithm

- Start with each vertex being its own component
- Repeatedly merge two components into one by choosing the **light** edge that connects them
- Which components to consider at each iteration?
  - Scan the set of edges in monotonically increasing order by weight

We would add edge (c, f)
Example

1. Add (h, g)  \{g, h\}, \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{i\}
2. Add (c, i)  \{g, h\}, \{c, i\}, \{a\}, \{b\}, \{d\}, \{e\}, \{f\}
3. Add (g, f)  \{g, h, f\}, \{c, i\}, \{a\}, \{b\}, \{d\}, \{e\}
4. Add (a, b)  \{g, h, f\}, \{c, i\}, \{a, b\}, \{d\}, \{e\}
5. Add (c, f)  \{g, h, f, c, i\}, \{a, b\}, \{d\}, \{e\}
6. Ignore (i, g)  \{g, h, f, c, i\}, \{a, b\}, \{d\}, \{e\}
7. Add (c, d)  \{g, h, f, c, i, d\}, \{a, b\}, \{e\}
8. Ignore (i, h)  \{g, h, f, c, i, d\}, \{a, b\}, \{e\}
9. Add (a, h)  \{g, h, f, c, i, d, a, b\}, \{e\}
10. Ignore (b, c)  \{g, h, f, c, i, d, a, b\}, \{e\}
11. Add (d, e)  \{g, h, f, c, i, d, a, b, e\}
12. Ignore (e, f)  \{g, h, f, c, i, d, a, b, e\}
13. Ignore (b, h)  \{g, h, f, c, i, d, a, b, e\}
14. Ignore (d, f)  \{g, h, f, c, i, d, a, b, e\}

{a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}
Implementation of Kruskal’s Algorithm

• Uses a disjoint-set data structure (see Chapter 21) to determine whether an edge connects vertices in different components
Operations on Disjoint Data Sets

- **MAKE-SET(u)** – creates a new set whose only member is \( u \)

- **FIND-SET(u)** – returns a representative element from the set that contains \( u \)
  - Any of the elements of the set that has a particular property
  - *E.g.:* \( S_u = \{r, s, t, u\} \), the property is that the element be the first one alphabetically
    - \( \text{FIND-SET}(u) = r \quad \text{FIND-SET}(s) = r \)
  - **FIND-SET** has to return the same value for a given set
Operations on Disjoint Data Sets

- UNION(u, v) – unites the dynamic sets that contain u and v, say $S_u$ and $S_v$
  - E.g.: $S_u = \{r, s, t, u\}$, $S_v = \{v, x, y\}$
    
    $\text{UNION}(u, v) = \{r, s, t, u, v, x, y\}$

- Running time for FIND-SET and UNION depends on implementation.

- We had seen earlier that FIND-SET can be done in $O(\log n)$ time
  and UNION operation can be done in $O(1)$  (see Chapter 21)
KRUSKAL($V$, $E$, $w$)

1. $A \leftarrow \emptyset$
2. for each vertex $v \in V$
   3. do $\text{MAKE-SET}(v)$ \quad $O(V)$
4. sort $E$ into non-decreasing order by $w$
5. for each $(u, v)$ taken from the sorted list
   6. do if $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$
      7. then $A \leftarrow A \cup \{(u, v)\}$ \quad $O(lgV)$
8. UNION($u$, $v$)
9. return $A$

Running time: $O(V + ElgE + ElgV) \Rightarrow O(ElgE)$

- dependent on the implementation of the disjoint-set data structure
- Since $E=O(V^2)$, we have $lgE=O(2lgV)=O(lgV)$
Kruskal’s Algorithm

- \( O(E \ lgE) \) \ or \ \( O(E \ lgV) \)
- Kruskal’s algorithm is “greedy”
- It produces a globally optimum solution
Problem 1

- Compare Prim’s algorithm with and Kruskal’s algorithm assuming:

  (a) sparse graphs: \( E = O(V) \)

\[ \text{Kruskal:} \]
\[ \text{UNION-FIND:} \quad O(\text{ElgE}) = O(\text{VlgV}) \]

\[ \text{Prim:} \]
\[ \text{Binary heap:} \quad O(\text{ElgV}) = O(\text{VlgV}) \]
(b) dense graphs: \[ E = O(V^2) \]

**Kruskal:**

\[ O(\text{ElgE}) = O(V^2 \text{lg} V^2) = O(2V^2 \text{lg} V) = O(V^2 \text{lg} V) \]

**Prim:**

Binary heap: \[ O(\text{Elg V}) = O(V^2 \text{lg} V) \]
Problem 2

Exercise 23.2-4
Analyze the running time of Kruskal’s algorithm when weights are in the range $[1 \ldots V]$

**ANSWER:**

- Sorting can be done in $O(E)$ time (e.g., using counting sort)
- However, overall running time will not change, i.e., $O(ElgV)$
Problem 3

Suppose that some of the weights in a connected graph G are negative. Will Prim’s algorithm still work? What about Kruskal’s algorithm? Justify your answers.

ANSWER:
Yes, both algorithms will work with negative weights. There is no assumption in the proof about the weights being positive.
Problem 4

Exercise 23.2-2
Analyze Prim’s algorithm assuming:

(a) an adjacency-list representation of $G$
$O(\text{Elg}V)$

(b) an adjacency-matrix representation of $G$
$O(\text{Elg}V+V^2)$  (see next slide)
PRIM(V, E, w, r)

1. \( Q \leftarrow \emptyset \)
2. for each \( u \in V \)
3. do \( \text{key}[u] \leftarrow \infty \)
4. \( \pi[u] \leftarrow \text{NIL} \)
5. \( \text{INSERT}(Q, u) \)
6. \( \text{DECREASE-KEY}(Q, r, 0) \) \% \( \text{key}[r] \leftarrow 0 \)
7. while \( Q \neq \emptyset \)
8. do \( u \leftarrow \text{EXTRACT-MIN}(Q) \)
9. for each \( v \in \text{Adj}[u] \)
10. do if \( v \in Q \) and \( w(u, v) < \text{key}[v] \)
11. then \( \pi[v] \leftarrow u \)
12. \( \text{DECREASE-KEY}(Q, v, w(u, v)) \)

Total time: \( O(V \lg V + E \lg V) = O(E \lg V) \)

- \( O(V) \) if \( Q \) is implemented as a min-heap
- \( O(\lg V) \) if \( Q \) is implemented as a min-heap
- \( O(V \lg V) \) for \( \text{EXTRACT-MIN}(Q) \)
- \( O(E \lg V) \) for \( \text{DECREASE-KEY}(Q, v, w(u, v)) \)
- \( O(V \lg V) \) for \( \text{INSERT}(Q, u) \)
- \( O(\lg V) \) for \( \text{DECREASE-KEY}(Q, r, 0) \)
PRIM(V, E, w, r)

1. Q ← ∅
2. for each u ∈ V
3. do key[u] ← ∞
4. π[u] ← NIL
5. INSERT(Q, u)
6. DECREASE-KEY(Q, r, 0) % key[r] ← 0
7. while Q ≠ ∅
8. do u ← EXTRACT-MIN(Q)
9. for (j=0; j<|V|; j++)
   do if (A[u][j]=1) and (v∈Q) and (w(u, v)<key[v])
   then π[v] ← u
10. DECREASE-KEY(Q, v, w(u, v))

Total time: \(O(E\lg V + V^2)\)
Problem 5

- Find an algorithm for the “maximum” spanning tree. That is, given an undirected weighted graph G, find a spanning tree of G of maximum cost. Prove the correctness of your algorithm.

  - Consider choosing the “heaviest” edge (i.e., the edge associated with the largest weight) in a cut. The generic proof can be modified easily to show that this approach will work.

  - Alternatively, multiply the weights by -1 and apply either Prim’s or Kruskal’s algorithms without any modification at all!
Problem 6

Exercise 23.1-8

Let T be a MST of a graph G, and let L be the sorted list of the edge weights of T. Show that for any other MST T’ of G, the list L is also the sorted list of the edge weights of T’

Proof: Kruskal’s algorithm will find T in the order specified by L. Similarly, if T’ is also an MST, Kruskal’s algorithm should be able to find it in the sorted order, L’. If L’ ≠ L, then there is a contradiction! Because at the point where they differ, Kruskal’s should have picked the smaller of the two and L’ is impossible to obtain.