

CS 445 The Greedy Paradigm

Credit: Erik Demaine and Carola Wenk

Greedy Paradigm:

One of the main **paradigms** we discuss in this course.

Basic idea.

1. Start with an empty solution
2. Improve the solution by a sequence of steps
3. In each step, we pick the improvements that brings maximum local improvement, without worrying what happen after the next step.

Problem: Might lead to a local optimum which is worse than the global optimum, and which we maybe can find using more global very.

Examples: steepest decent, Tetris.

Advantageous: Usually very easy to implement, and does lead to optimal global solutions in many cases.

Are the rules when “greedy” can work ?

Yes, but we will discuss them after seeing a few examples of greedy algorithms.

Minmal spanning tree: Sometimes greedy is optimal

But first some review of graphs

Graphs (review)

Definition. A *directed graph (digraph)*

$G = (V, E)$ is an ordered pair consisting of

- a set V of *vertices* (singular: *vertex*),
- a set $E \subseteq V \times V$ of *edges*.

In an *undirected graph* $G = (V, E)$, the edge set E consists of *unordered* pairs of vertices.

Assume: No parallel (two edges connecting the same vertices):

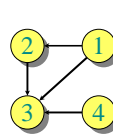
In either case, we have $|E| = O(V^2)$. Moreover, if G is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$.

(Review CLRS, Appendix B.4 and B.5.)

Adjacency-matrix representation

The *adjacency matrix* of a graph $G = (V, E)$, where $V = \{1, 2, \dots, n\}$, is the matrix $A[1 \dots n, 1 \dots n]$ given by

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$

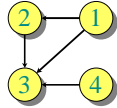


A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

$\Theta(V^2)$ storage
 \Rightarrow *dense*
representation.

Adjacency-list representation

An **adjacency list** of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to v .



$Adj[1] = \{2, 3\}$
 $Adj[2] = \{3\}$
 $Adj[3] = \{1, 4\}$
 $Adj[4] = \{3\}$

For undirected graphs, $|Adj[v]| = degree(v)$.
 For digraphs, $|Adj[v]| = out-degree(v)$.

Handshaking Lemma: $\sum_{v \in V} deg(v) = 2|E|$ for undirected graphs \Rightarrow adjacency lists use $\Theta(V + E)$ storage — a **sparse** representation (for either type of graph).

Cute property: In any party the # people shaking an odd number of hands is even

More definitions

A graph $G(V, E)$ is **connected** if there is a path of edges connecting every pair of vertices of G .

A subgraph $G'(V, E')$ is a **spanning graph** for G , if

1. it has the same set of vertices V ,
2. it is also connected, and
3. $E' \subseteq E$

A **tree** is a cycle-free connected graph.

Definitions about trees

Let G be a graph with m edges and n vertices. All the following properties are equivalent.

1. G is a tree (i.e, G is connected and cycle-free)
2. G is connected, and contains $n-1$ edges.
3. G is cycle-free, but **adding** any edge to G might caused it to stop being cycle-free.
4. G is connected, but **removing** any edge from G would caused it to stop being connected.
5. Between every pair of vertices there is **exactly** one path.
6. Etc etc

Minimum spanning trees (MST) problem

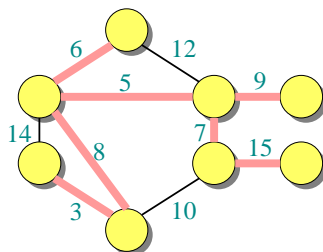
Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}$.

- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)

Output: A **spanning tree** T — a tree that connects all vertices — of minimum total sum of weights of its edges.

$$w(T) = \sum_{(u,v) \in T} w(u,v).$$

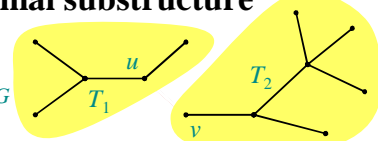
Example of MST



Optimal substructure

MST T :

(Other edges of G are not shown.)



Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2 .

Theorem. The subtree T_1 is an MST of $G_1 = (V_1, E_1)$, the subgraph of G **induced** by the vertices of T_1 :

$$V_1 = \text{vertices of } T_1,$$

$$E_1 = \{ (x, y) \in E : x, y \in V_1 \}.$$

Similarly for T_2 .

Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1 , then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G . \square

MST exhibits another powerful property which leads to an even more efficient algorithm.

Hallmark for “greedy” algorithms

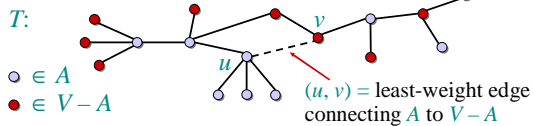
Greedy-choice property
A locally optimal choice is globally optimal.

Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$ be any subset of vertices of V .

Suppose that $(u, v) \in E$ is the cheapest edge connecting A to $V - A$. Then, $(u, v) \in T$.

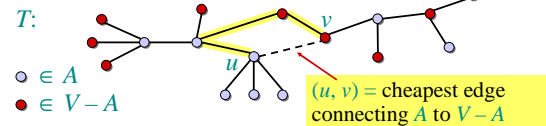
Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.



Proof of theorem

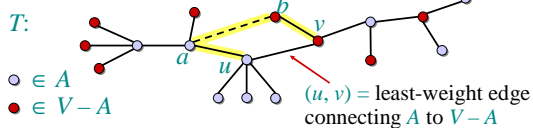
Proof. Suppose $(u, v) \notin T$. Cut and paste.



Consider the unique simple path from u to v in T .

Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.

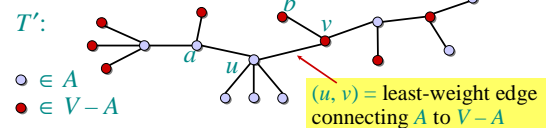


Consider the unique simple path from u to v in T .

Swap (u, v) with the first edge, say (a, b) on this path that connects a vertex in A to a vertex in $V - A$.

Proof of theorem

Proof. Suppose $(u, v) \notin T$. Cut and paste.



Consider the unique simple path from u to v in T .

Swap (u, v) with the first edge, say (a, b) on this path that connects a vertex in A to a vertex in $V - A$.

A lighter-weight spanning tree than T results, since $w(a, b) > w(u, v)$. \square

Prim's algorithm

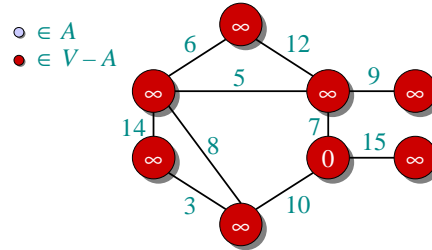
IDEA: Maintain $V - A$ as a priority queue Q . Key each vertex in Q with the weight of the cheapest edge connecting it to a vertex in A (can be ∞)

```

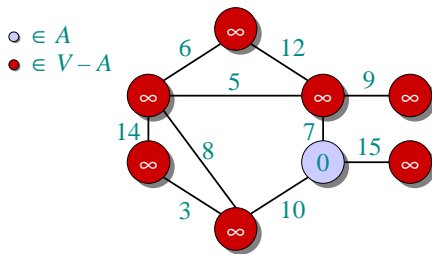
 $Q \leftarrow V$ 
 $key[v] \leftarrow \infty$  for all  $v \in V$ 
 $key[s] \leftarrow 0$  for some arbitrary  $s \in V$ 
while  $Q \neq \emptyset$ 
do  $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
  for each  $v \in \text{Adj}[u]$ 
  do if  $v \in Q$  and  $w(u, v) < key[v]$ 
    then  $\{key[v] \leftarrow w(u, v); \triangleright \text{DECREASE-KEY}$ 
       $\pi[v] \leftarrow u\}$ 
  
```

At the end, the set of edges, $\{(v, \pi[v])\}$ forms the MST.

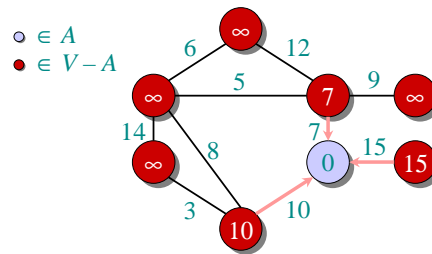
Example of Prim's algorithm



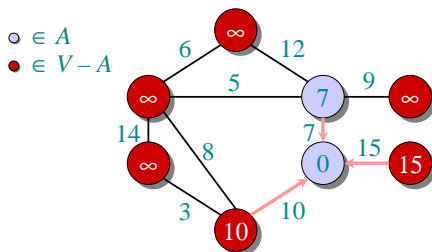
Example of Prim's algorithm



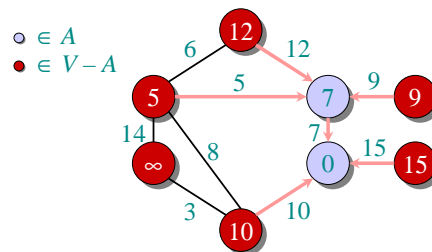
Example of Prim's algorithm



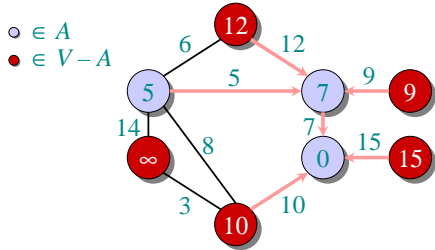
Example of Prim's algorithm



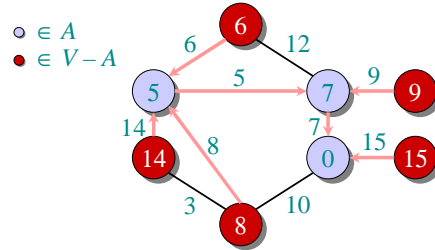
Example of Prim's algorithm



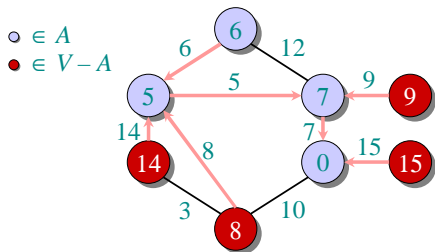
Example of Prim's algorithm



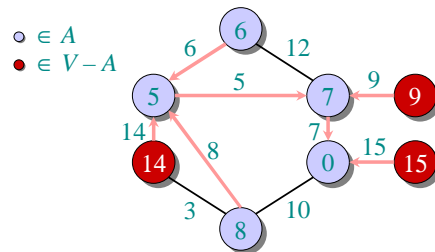
Example of Prim's algorithm



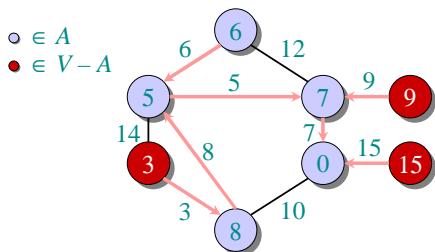
Example of Prim's algorithm



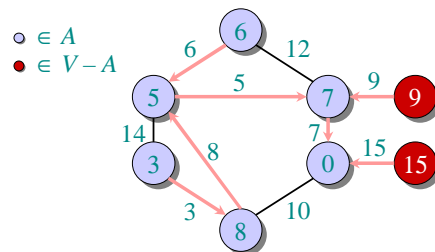
Example of Prim's algorithm



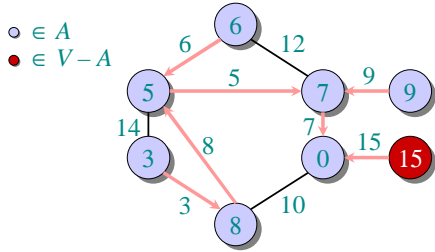
Example of Prim's algorithm



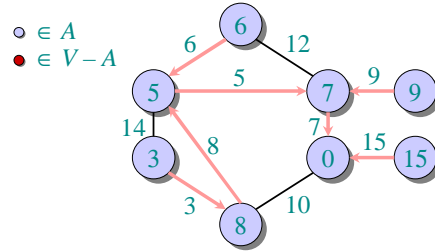
Example of Prim's algorithm



Example of Prim's algorithm



Example of Prim's algorithm



Analysis of Prim

$\Theta(V)$ total $\left\{ \begin{array}{l} Q \leftarrow V \\ key[v] \leftarrow \infty \text{ for all } v \in V \\ key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \end{array} \right.$
 $|V|$ times $\left\{ \begin{array}{l} \text{while } Q \neq \emptyset \\ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\ \text{for each } v \in Adj[u] \\ \text{do if } v \in Q \text{ and } w(u, v) < key[v] \\ \text{then } key[v] \leftarrow w(u, v) \\ \pi[v] \leftarrow u \end{array} \right.$
 Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.
 Time = $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$

Analysis of Prim (continued)

Time = $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$	$O(E + V \lg V)$ worst case

MST algorithms

- Best to date:
- Karger, Klein, and Tarjan [1993].
 - Randomized algorithm.
 - $O(V + E)$ expected time.