CS213/293 Data Structure and Algorithms 2023

Lecture 18: Graphs - minimum spanning tree

Instructor: Ashutosh Gupta

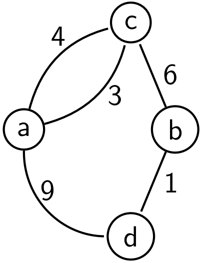
IITB India

Compile date: 2023-11-06

Topic 18.1

Labeled graph

Labeled graph



Definition 18.1

A graph $G = (V, E)$ is consists of

- *set V of vertices and*
- *set E of pairs of*
  - *unordered pairs from V and*
  - *labels from $\mathbb{Z}^+$ (known as length).*

For $e \in E$, we will write $L(e)$ to denote the length.

The above is a labeled graph $G = (V, E)$, where

$V = \{a, b, c, d\}$ and

$E = \{(\{a, c\}, 3), (\{a, c\}, 4), (\{a, d\}, 9), (\{b, c\}, 6), (\{b, d\}, 1)\}$.

$L((\{a, c\}, 3)) = 3$.

Minimum spanning tree (MST)

Consider a labeled graph $G = (V, E)$.

Definition 18.2

A spanning tree of G is a labeled tree (V, E'), where $E' \subseteq E$.

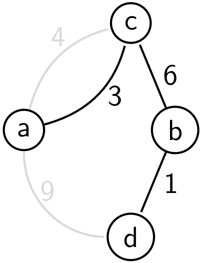
Definition 18.3

A length of G is $\sum_{e \in E} L(e)$.

Definition 18.4

A minimum spanning tree of G is a spanning tree G' such that the length of G is minimum.

Example 18.1

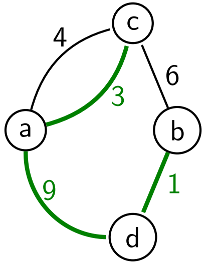


The above is an MST of the graph in the previous slide.

Example: MST

Example 18.2

Consider the following spanning tree (green edges). Is this an MST?



We can achieve an MST by replacing 9 by 6.

Observation:

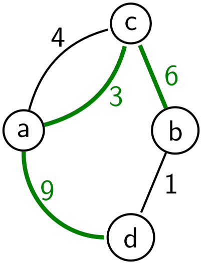
- consider an edge e that is not part of the spanning tree.
- add e to the spanning tree, which must form exactly one cycle.
- if e is not the maximum edge in the cycle, the spanning tree is not the minimum.

Observation: minimum edge will always be part of MST.

Apply the previous observation, the edge will definitely replace another edge.

Example 18.3

In the following spanning tree, if we add edge 1, we can easily remove another edge and obtain a spanning tree with lower length.



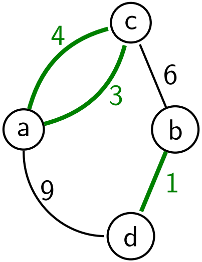
Can we keep applying this observation on greater and greater edges?

Idea: Should we create MST using minimum $|V| - 1$ edges?

No. The method will not always work.

Example 18.4

In the following graph, the minimum tree edges form a cycle.



Topic 18.2

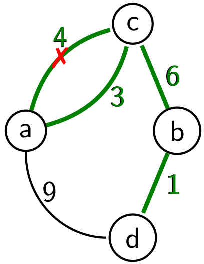
Kruskal's algorithm

Idea: Keep collecting minimum edges, while avoiding cycle-causing edges.

Maybe. Let us try.

Example 18.5

In the following graph, let us construct MST.



This is an MST.

Are we lucky? Or, do we have a real procedure?

Kruskal's algorithm

Algorithm 18.1: $\mathrm{MST}($ Graph $G = (V, E)$, vertex v $)$

$Elist :=$ sorted list of edges in E according to their labels;
$E' = \emptyset$;
for $e = (\{v, v'\}, \_) \in Elist$ **do**
    if *v and v' are not connected in (V, E')* **then**
        $E' := E' \cup \{e\}$
return (V, E')

In $(\{v, v'\}, \_)$, $\_$ indicates that the value is don't care.

Running time complexity $O(\underbrace{|E| log(|E|)}_{sorting} + |E| \times IsConnected)$

How do we check connectedness?

We maintain sets of connected vertices.

The sets merge as the algorithm proceeds.

We will show that checking connectedness can be implemented in $O(\log |V|)$ using union-find data structure.

Example: connected sets

Example 18.6

Let us see connected sets in the progress of Kruskal's algorithm.

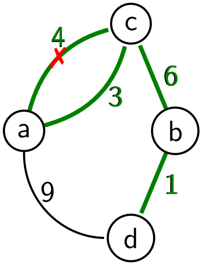
Initial connected sets: $\{\{a\},\{b\},\{c\},\{d\}\}$.

After adding edge 1: $\{\{a\},\{c\},\{b,d\}\}$.

After adding edge 3: $\{\{a,c\},\{b,d\}\}$.

When we consider edge 4: a and c are already connected.

After adding edge 6: $\{\{a,b,c,d\}\}$.

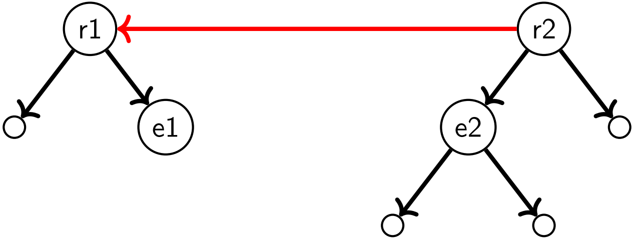


Each time we consider an edge we need to check if both ends are in the same set or not.

Union find

Equivalent classes are maintained as forest. The data structure has two operations.

- IsConnect(e1,e2): traverse to their root and see if the roots are the same
- merge(r1,r2): make the root of the taller tree parent of the root of the other tree



Exercise 18.1

Prove that merge is $O(1)$ and IsCoonect is $O(\log n)$.

Correctness of Kruskal's algorithm

Theorem 18.1

For a graph $G = (V, E)$, $MST(G)$ returns an MST of G.

Proof.

Let us assume edge lengths are unique. (can be relaxed!)

Let us suppose $MST(G)$ returns edges $e_1, ..., e_n$, which are written in increasing order.

Let us suppose an MST consists of edges $e'_1, ..., e'_n$, which are written in increasing order.

Let i be the smallest index such that $e_i \neq e'_i$. ...

Correctness of Kruskal's algorithm(2)

Proof(Contd.)

case: $L(e_1) > L(e'_i)$**:**

Kruskal must have considered e'_i before e_i.

$e_1, ..., e_{i-1}, e'_i$ has a cycle because Kruskal skipped e'_i.

Therefore, $e'_1, ..., e'_n$ has a cycle. Contradiction. ...

Correctness of Kruskal's algorithm(3)

Proof(Contd.)

case: $L(e_1) < L(e'_i)$**:**

Consider graph $e'_1, ..., e'_n, e_i$, which has exactly one cycle. Let C be the cycle.

For all $e \in C - \{e_i\}$, $L(e_i) > e$ because $e'_1, ..., e'_n$ is MST. (Why?)

Therefore, $C \subseteq \{e_1, ..., e_i\}$.

Therefore, $e_1, ..., e_i$ has a cycle. Contradiction. □

Exercise 18.2

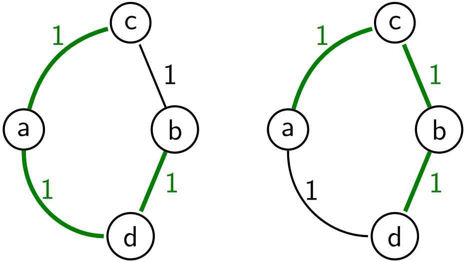
a. Prove the above theorem after relaxing the condition of unique edge lengths.

b. Prove that MST is unique if edge lengths are unique.

Example: MST is not unique if edge lengths are not unique.

Example 18.7

The following graph has multiple MSTs.



Topic 18.3

Prim's algorithm

Cut of a graph

Consider a labeled graph $G = (V, E)$.

Definition 18.5

For a set $S \subseteq V$, a cut C *of G is* $\{(\{v, v'\}, \_) \in E | v \in S \wedge v' \notin S\}$.

The minimum edge of a cut will always be part of MST.

Theorem 18.2

For a labeled graph $G = (V, E)$, the minimum edge of a non-empty cut C will be part of MST.

Proof.

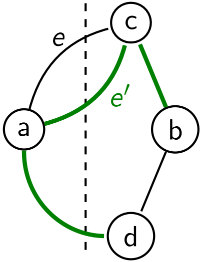
Let C be a cut of G for some set $S \subseteq E$ and $e \in C$ be the minimum edge.

Let us assume MST G'(green) does not contain e.

Since both S and $E - S$ are not empty, $G' \cap C \neq \emptyset$ and for each $e' \in G' \cap C$ and $L(e') > L(e)$.

$G' \cup \{e\}$ has a cycle containing e and some $e' \in G' \cap C$.

Therefore, $G' \cup \{e\} - \{e'\}$ is a spanning with a smaller length. Contradiction. $\square$



Prim's idea

Start with a single vertex in the visited set.

Keep expanding MST over visited vertices by adding the minimum edge connecting to the rest.

Example: cut progress

Example 18.8

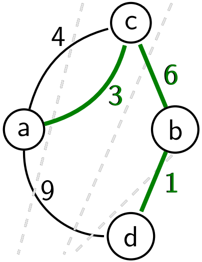
Let us see MST construction via cuts.

We start with vertex a. The cut has edges 4, 3, and 9.

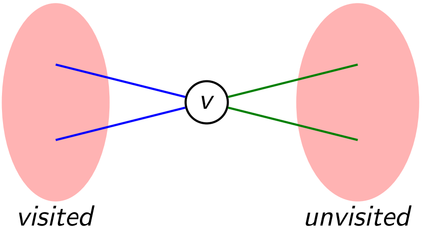
Since the minimum edge on the cut is 3, we add the edge to MST and $visited = \{a, c\}$. Now cut has edges 6 and 9.

Since the minimum edge on the cut is 6, we add the edge to MST and $visited = \{a, c, b\}$. Now cut has edges 1 and 9.

Since the minimum edge on the cut is 1, we add the edge to MST.



Operations during entering the visited set



When a vertex v moves from the unvisited set to the visited set, we need to delete blue edges from the cut and add green edges to the cut.

Prim's algorithm

Algorithm 18.2: $\mathrm{MST}($ Graph $G = (V, E)$, vertex r $)$

for $v \in V$ **do** $v.visited := False$;
$r.visited := True$;
for $e = (\{v, r\}, \_) \in E$ **do** $cut := cut \cup \{e\}$;
while $cut \neq \emptyset$ **do**
    $(\{v, v'\}, \_) := cut.min()$;
    Assume$(\neg v.visited \wedge v'.visited)$; // This condition is always true
    for $e = (\{v, w\}, \_) \in E$ **do**
        if $w.visited$ **then**
            $cut.delete(e)$ // Cost: $O(\log |E|)$
        else
            $cut.insert(e)$ // Cost: $O(\log |E|)$
    $v.visited := True$

Running time: $O(|E| \log |E|)$ because every edge will be inserted and deleted.

Data structure for cut

We may use a heap to store the cut since we need a minimum element.

We need to be careful while deleting an edge from the heap.

Since searching in the heap is expensive, we need to keep the pointer from the edge to the node of the heap.

Prim's algorithm: with an optimization

Algorithm 18.3: MST(Graph $G = (V, E)$, vertex r)

Heap unvisited;
for $v \in V$ **do**
  $v.visited := False$;
  $unvisited.insert(v, \infty)$ // Will heapify help?
$unvisited.decreasePriority(r, 0)$;
while $unvisited \neq \emptyset$ **do**
  $v := unvisited.deleteMin()$;
  for $e = (\{v, w\}, k) \in E$ **do**
    if $\neg w.visited$ **then**
      $unvisited.decreasePriority(w, k)$ // Cost: $O(\log |V|)$
  $v.visited := True$

Running time: $O(|V| + |E| \log |V|)$ because every edge is visited to change the priority of a vertex.

Commentary: decreasePriority reduces the priority of an element in the heap. deleteMin returns the minimum element in the heap and deletes the element in the heap.

Topic 18.4

Tutorial problems

Exercise: proving with non-unique lengths

Example 18.9

Modify proof of theorems 18.1 and 18.2 to support non-unique edges.

Exercise: non-unique lengths

Example 18.10

Kruskal's algorithm can return different spanning trees for the same input graph G, depending on how it breaks ties when the edges are sorted into order. Show that for each minimum spanning tree T of G, there is a way to sort the edges of G in Kruskal's algorithm so that the algorithm returns T.

Exercise: minimum spanning directed rooted tree(arborescence)

Definition 18.6

A graph $G = (V, E)$ is a directed rooted tree if for each $v, v' \in V$ there is exactly one path between v and v'.

Example 18.11

Show that Kruskal's and Prim's algorithm will not find a minimum spanning directed rooted tree.

End of Lecture 18