CS213/293 Data Structure and Algorithms 2024

Lecture 17: Graphs - minimum spanning tree

Instructor: Ashutosh Gupta

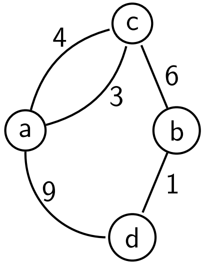
IITB India

Compile date: 2024-11-02

Topic 17.1

Labeled graph

Labeled graph



Definition 17.1

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/weight).*

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 17.2

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

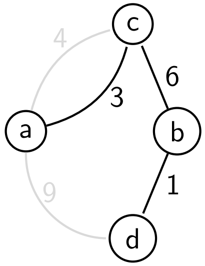
Definition 17.3

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

Definition 17.4

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

Example 17.1

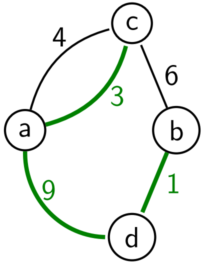


The above is an MST of the graph in the previous slide.

Example: MST

Example 17.2

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



We can achieve an MST by replacing 9 by 6.

Observation:

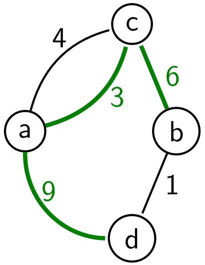
- consider an edge e that is not part of the spanning tree.
- add e to the spanning tree, which must form exactly one cycle.
- if e is not the maximum edge in the cycle, the spanning tree is not the minimum.

Observation: minimum edge will always be part of MST.

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

Example 17.3

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



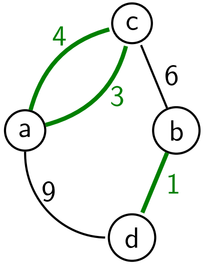
Can we keep applying this observation on greater and greater edges?

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

No. The method will not always work.

Example 17.4

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



Topic 17.2

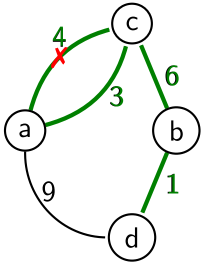
Kruskal's algorithm

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

Maybe. Let us try.

Example 17.5

In the following graph, let us construct MST.



It is an MST.

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

Kruskal's algorithm

Algorithm 17.1: $\mathrm{MST}($ Graph $G = (V, E))$

```
1 $Elist :=$ sorted list of edges in $E$ according to their labels;
2 $E' = \emptyset$;
3 for $e = (\{v, v'\}, \_) \in Elist$ do
4     if $v$ and $v'$ are not connected in $(V, E')$ then
5         $E' := E' \cup \{e\}$
6 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 17.6

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

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

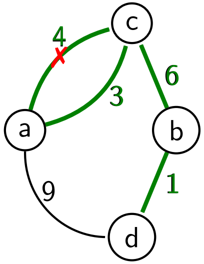
After adding edge 1: $\{\{a\},\{c\},\{b,d\}\}$.

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 of the edge are in the same set or not.



Correctness of Kruskal's algorithm

Theorem 17.1

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

Proof.

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

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

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: $L(e_i) > L(e'_i)$**:**

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: $L(e_i) < L(e'_i)$**:**

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

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

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

Therefore, $e_1, ..., e_i$ has a cycle. Contradiction. $\square$

Exercise 17.1

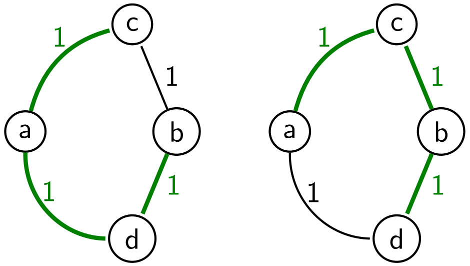
a. Prove the above theorem after relaxing the condition of unique edge lengths.

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

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

Example 17.7

The following graph has multiple MSTs.



Reverse-delete algorithm: upside down Kruskal

Here is an interesting variation of Kruskal's idea, i.e., large edges are not in MST.

Algorithm 17.2: MSTREVERSE(Graph $G = (V, E)$)

```
ASSUME($G$ is connected);
$Elist :=$ the list of edges in $E$ in decreasing order;
for $e \in Elist$ do
    if $(V, toSet(EList) - \{e\})$ is a connected graph then
        $EList.remove(e)$;
return $(V, toSet(EList))$
```

Exercise 17.2

a. Show that the algorithm returns a spanning tree.

b. How can we check the if condition efficiently?

Commentary: Answer: a. Since the graph remains connected at the end, it will return a spanning graph. Let us suppose the algorithm returns a graph with a cycle. Consider the edge e with the maximum weight on the cycle. The algorithm must have considered e at an earlier iteration and removed it. Therefore, we have contradiction. We are yet to show that the spanning tree is minimum. b. `https://dl.acm.org/doi/10.1145/335305.335345`

Correctness of MSTReverse

Let us assume all edge labels are distinct.

Theorem 17.2
$(V, toSet(EList))$ *always contains the MST.*

Proof.
Base case:
Initially, G contains the MST.

Induction step:
Let us suppose at some step $(V, toSet(EList))$ contains the MST T and the algorithm deletes an edge $e \in EList$.

If $e \notin T$, then $(V, toSet(EList) - \{e\})$ still contains T. ...

Correctness of MSTReverse

Proof(Contd.)

If $e \in T$, the deletion of e splits T into two trees T_1 and T_2.

e is deleted because $(V, toSet(EList))$ has a cycle.

Therefore, there is an edge $e' \in toSet(EList) - T$, such that $T_1 \cup T_2 \cup \{e'\}$ is a tree. (Why?)

If $L(e) > L(e')$, then the weight of $T_1 \cup T_2 \cup \{e'\}$ is smaller than the weight of T. Contradiction.

If $L(e) < L(e')$, e' must have been considered in an earlier iteration, where the same cycle was present. Therefore, e' was already deleted. Contradiction. □

Topic 17.3

Prim's algorithm

Cut of a graph

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

Definition 17.5

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 17.3

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

Proof.

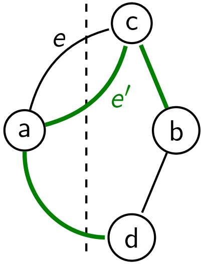
Let C be a cut of G for some set $S \subseteq V$ and $e \in C$ be the minimum edge.

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

Since both S and $V - S$ 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 tree 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 17.8

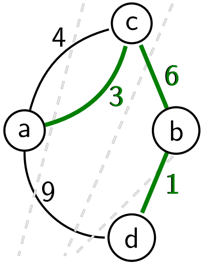
Let us see MST construction via cuts.

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

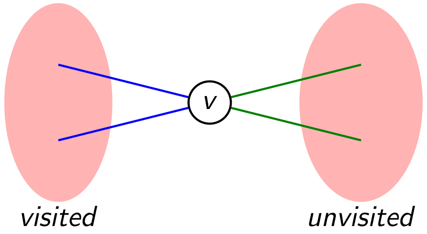
Since the minimum edge on the cut is 3, we add the edge to MST and *visited* $= \{a, c\}$. Now cut has edges 6 and 9.

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

```
1  for $v \in V$ do  $v.visited := False$ ;
2  $r.visited := True$;
3  for $e = (\{v, r\}, \_) \in E$ do  $cut := cut \cup \{e\}$ ;
4  while $cut \neq \emptyset$ do
5      $(\{v, v'\}, \_) := cut.min()$;
6      Assume($\neg v.visited \wedge v'.visited$);              // This condition is always true
7      for $e = (\{v, w\}, \_) \in E$ do
8          if $w.visited$ then
9              $cut.delete(e)$                                    // Cost: $O(\log |E|)$
10         else
11             $cut.insert(e)$                                    // Cost: $O(\log |E|)$
12     $v.visited := True$
```

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.

Can we avoid storing the set of edges? Instead, store the set of next available nodes.

Prim's algorithm with an optimization

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. Since we are using decreasePriority for each edge, we can improve the complexity by using Fibonacci heap, where the cost of decreasePriority is amortized O(1).

Algorithm 17.4: MST(Graph $G = (V, E)$, vertex r)

```
1  Heap unvisited;
2  for $v \in V$ do
3      $v.visited := False$;
4      $unvisited.insert(v, \infty)$                          // Will heapify help?
5  $unvisited.decreasePriority(r, 0)$;
6  while $unvisited \neq \emptyset$ do
7      $v := unvisited.deleteMin()$;                       // Cost: $O(\log |V|)$
8      for $e = (\{v, w\}, k) \in E$ do
9          if $\neg w.visited$ then
10             $unvisited.decreasePriority(w, k)$            // Cost: $O(\log |V|)$
11     $v.visited := True$
```

Running time: $O((|V| + |E|) \log |V|)$ because every node and edge is visited for a heap operation.

Exercise 17.3 *Modify the above algorithm to make it return the MST.*

Example: prim with neighbors

Example 17.9

Let us see MST construction via unvisited neighbors.

We start with vertex *a*. The unvisited neighbors are c and d.

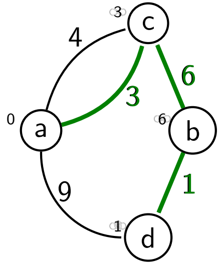
Since the minimum edge to c is 3, we update unvisited(c) = 3. Similarly, we update unvisited(d) = 9.

We add c in visited set. Now unvisited neighbors are b and d.

Since the minimum edge to b is 6, we update unvisited(b) = 6.

We add b in visited set. Now unvisited neighbor is d.

Since the minimum edge to d is 1, we update unvisited(d) = 1. We add d in visited set.



Topic 17.4

Tutorial problems

Exercise: proving with non-unique lengths

Exercise 17.4

Modify proof of theorems 17.1 and 17.3 to support non-unique edges.

Exercise: non-unique lengths

Exercise 17.5

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 tree for directed graphs

Definition 17.6

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

On directed graphs, the problem of finding MST changes to the problem of finding an arborescence.

Exercise 17.6

a. What is anti-arborescence?

b. Show that Kruskal's and Prim's algorithm will not find a minimum spanning arborescence for a directed graph.

c. Give an algorithm that works on a directed graph.

Exercise: Borůvka's algorithm: Prim on steroids

Originally proposed in 1928 (before computers) to optimally layout electrical lines.

Algorithm 17.5: $\mathrm{MST}($ Graph $G = (V, E)$)

```
Assume $G$ is connected.
mst $:= \emptyset$
components $:= \{\{v\} | v \in V\}$
while $components.size > 1$ do
    for $c \in components$ do
        e := c.outgoingEdges.min()
        mst $:= mst \cup \{e\}$
    components := ConnectedComponents( (V,mst) )
return (V,mst)
```

Exercise 17.7

Prove that Borůvka's algorithm returns an MST.

End of Lecture 17