23. Minimum Spanning tree

→ Greedy Approach: Prim's, Kruskal
→ Dijkstra's

Problem:

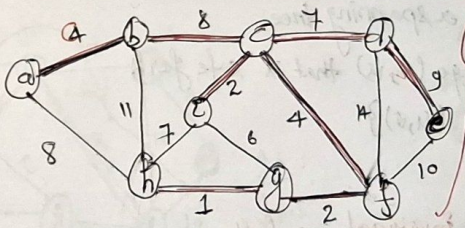
- A town has a set of houses & a set of roads.
- A road connect 2 & only 2 houses.
- A road connecting houses u & v has a repair cost $w(u,v)$.
- **Goal:**
  1. Everyone stays connected: can reach every house from all other houses.
  2. Total repair cost is minimum.

* Modal as a graph:

- Undirected graph $G = (V, E)$
- Weight $w(u,v)$ on each edge $(u,v) \in E$.
- Find $T \subseteq E$ such that
  1. T connects all vertices (T is a spanning tree)
  2. $w(T) = \sum_{u,v \in T} w(u,v)$ is minimized.

A minimum spanning tree (MST) is a spanning tree whose weight is minimum overall spanning trees.

Figure 23.1



There is more than one MST. Replace edge (b,c) by (a,h). Get a diff. spanning tree with weight 37.

4 + 8 + 7 + 9 + 2 + 4 + 2 + 1 = 37.

23.1 : Growing a minimum Spanning tree

Some properties of an MST:

→ It has $|V|-1$ edges
→ It has no cycles
→ It might not be unique.

Building up the solution

→ We will build a set A of edges.
→ Initially, A has no edges
→ As we add edges to A, maintain a loop invariant:

Loop Invariant : A is a subset of some MST.

→ Add only edges that ~~retain~~ maintain the invariant. If A is a subset of some MST, an edge (u,v) is SAFE for A iff $A \cup \{(u,v)\}$ is also a subset of some MST. So, we will add only safe edges.

Generic MST Algorithm

```
GENERIC-MST (G, w)
1  A ← ϕ
2  while A is not a spanning tree
3      find an edge (u,v) that is safe for A
4      A = A ∪ {(u,v)}
5  return A
```

We use the loop invariant as follows:

① **Initialization :** After line 1, the set A trivially satisfies the loop invariant.

② **Maintenance :** The loop in lines 2-4 maintains the invariant by adding only safe edges.

③ **Termination :** All edges added to A are in a minimum spanning tree & so the set A returned in line 5 must be a minimum spanning tree.

How to find a safe edge ?

$w(h,g) \to 1$

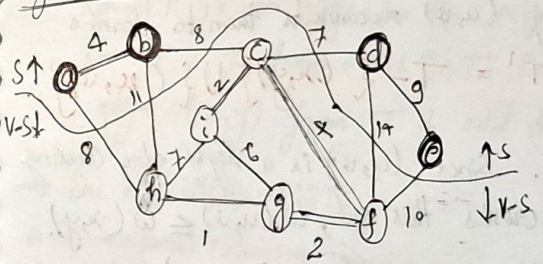
Let $S \subset V$ be any set of vertices that includes h but not g (so that g is in $V-S$). In any MST, there has to be one edge (at least) that ~~contains~~ connects S with $V-S$. Why not choose the edge with min. weight (which would be (h,g) in this case.)

Some definition : Let $S \subset V$ & $A \subseteq E$.

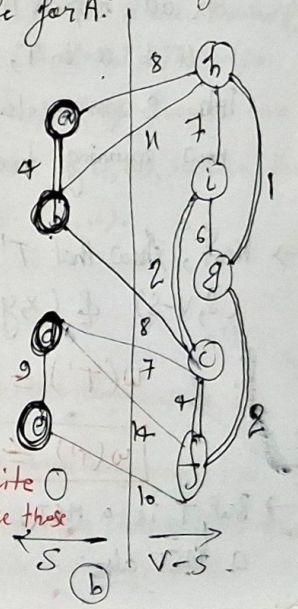
→ A cut $(S, V-S)$ is partition of vertices into disjoint set V & $S-V$.
→ Edge $(u,v) \in E$ crosses cut $(S, V-S)$ if one endpoint is in S & the other is in $V-S$.
→ A cut respects A iff no edges in A crosses the cut
→ An edge is a light edge crossing a cut iff its weight is minimum of any edge crossing the cut.

Theorem 23.1 : 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 & Let (u,v) be a light edge crossing $(S, V-S)$. Then, edge (u,v) is a safe for A.

Figure 23.2



2-ways of viewing a cut $(S, V-S)$ of the graph from fig. 22.1 (a) Black ● vertices are in the set S & white ○ vertices are in $V-S$. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d,c) is unique light edge crossing the cut.



A subset A of the edges is shaded //; note that the cut $(S, V-S)$ respects A, since no edge of A crosses the cut. ⓑ The same graph with the vertices in the set S on the left & the vertices in the set V−S on the right. An edge crosses the cut if it connects a vertex on the left with a vertex on the right.

Proof: Let T be a MST that includes A & assume that T does not contain the light edge (u,v), since if it does, we are done. We shall construct another MST T' that includes $A \cup \{(u,v)\}$ by using a cut-and-paste technique, thereby showing that (u,v) is a safe edge for A.

→ The edge (u,v) form a cycle with the edges on the simple path p from u to v in T, as figure-23.3.

→ Since u & v are on opposite sides of the cut $(S, V-S)$, at least one edge in T lies on the simple path p and also crosses the cut.

→ Let (x,y) be any such edge. The edge (x,y) is not in A, because the cut respects A. Since (x,y) is on the unique simple path from u to v in T, removing ~~a mst spanning tree~~ (x,y) breaks T into 2 components. Adding (u,v) reconnects them to form a new spanning tree $T' = T - \{(x,y)\} \cup \{(u,v)\}$.

→ Next, show that T' is MST. Since (u,v) is a light edge crossing $(S, V-S)$ & (x,y) also crosses this cut, $w(u,v) \leq w(x,y)$.

$$\therefore \quad w(T') = w(T) - w(x,y) + w(u,v)$$

$$\boxed{w(T') \leq w(T)}$$

→ But T is a MST, so that $w(T) \leq w(T')$; thus T' must be a MST also.

→ It remains to show that (u,v) is actually a safe edge for A. We have $A \subseteq T'$, since $A \subseteq T$ & $(x,y) \notin A$; thus $A \cup \{(u,v)\} \subseteq T'$. Consequently, since T' is a MST, (u,v) is safe for A.

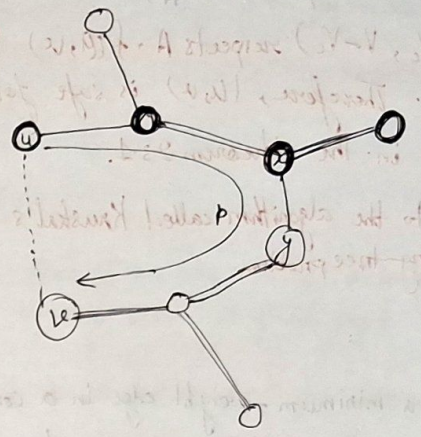


Figure 23.3: The proof of theorem 23.1. Black vertices ● are in S, & white vertices ○ are in V−S. The edges in the MST T are shown, but the edges in the graph G are not. The edges in A are shaded //, & (u,v) is a light edge crossing the cut $(S, V-S)$. The edge (x,y) is an edge on the unique simple path p from u to v in T. To form a MST T' that contains (u,v), remove the edge (x,y) from T & add the edge (u,v).

→ So, in GENERIC-MST:

* A is a forest containing connected components. Initially, each component is a single vertex.
* Any safe edge merges two of these components into one. Each component is a tree.
* Since an MST has exactly $|V|-1$ edges, the for loop iterates $|V|-1$ times. Equivalently, after adding $|V|-1$ safe edges, we're down to just one component.

Corollary 23.2

Let $G = (V, E)$ be 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 $C = (V_C, E_C)$ be connected component (tree) in the forest $G_A = (V, A)$. If (u, v) is a light edge connecting C to some other component in G_A, then (u, v) is a safe for A.

Proof: → The cut $(V_C, V - V_C)$ respects A, & (u, v) is a light edge for this cut. Therefore, (u, v) is safe for A.
Set $\boxed{S = V_C}$ in the theorem 23.1.

→ This naturally leads to the algorithm called Kruskal's algorithm to solve the minimum-spanning-tree problem.

Exercises

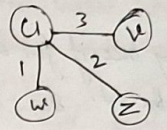
23.1-1: Let (u, v) be a minimum-weight edge in a connected graph G. Show that (u, v) belongs to some minimum spanning tree of G.

Suppose that A is an empty set of edges. Then, make any cut that has (u, v) crossing it. Then, since that edge is of minimal weight, we have that (u, v) is a light edge of that cut, & so it is safe to add. Since we add it, then once we finish constructing the tree, we have that (u, v) is contained in a minimum spanning tree.

23.1-2: Professor Sabatier conjectures the following converse of Theorem 23.1. 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 MST for G, Let $(S, V - S)$ be any cut of G that respects A, & let (u, v) be a safe edge for A crossing $(S, V - S)$. Then, (u, v) is a light edge for the cut. Show that the professor's conjecture is incorrect by giving a counterexample.

Let G be the graph with 4 vertices: u, v, w, z. Let the edges of the graph be (u, v), (u, w), (w, z) with weights $(3, 1, \& 2)$ respectively.

Suppose A is the set $\{(u, w)\}$. Let $S = A$. Then S clearly respects A. Since G is a tree, its MST is itself, so A is trivially a subset of a MST. Moreover, every edge is safe, In particular, (u, v) is safe but not a light edge for the cut. Therefore, professor's conjecture is false.



23.1-3: Show that if an edge (u, v) is contained in some MST, then it a light edge crossing some cut of the graph.

Let T_0 & T_1 be the two trees that are obtained by removing edge (u, v) from a MST. Suppose that V_0 & V_1 are the vertices of T_0 & T_1 respectively. Consider, the cut which separates V_0 from V_1. Suppose to a contradiction that there is some edge that has weight less than that of (u, v) in this cut. Then, we could construct a MST of the whole graph by adding that edge to $T_1 \cup T_0$. This would result in a minimum spanning tree that has weight less than the original MST that containing (u, v).

23.1-4: Give an example of a connected graph such that the set of edges $\{(u, v)\}$: there exists a cut $(S, V - S)$ such that (u, v) is a light edge crossing $\{(S, V - S)\}$ does not form a MST.

Let G be a graph on 3 vertices, each connected to the other 2 by an edge, & such that each edge has weight 1. Since every edge has the same weight, every edge is a light edge for a cut which it spans. However, if we take all edges, we get a cycle

23.1-5: Let e be a maximum-weight edge on some cycle of connected graph $G = (V, E)$. Prove that there is a MST of $G' = (V, E - \{e\})$ that is also a MST of G. That is, there is a MST of G that does not include e.

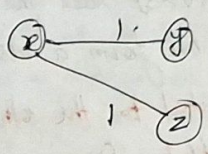
Let A be any cut that causes some vertices in the cycle on one side of the cut, & some vertices in the cycle on the other. For any of these cuts, we know that the edge e is not a light edge for this cut. Since all the other cuts won't have the edge e crossing it, we won't have that the edge is light for any of those cuts either. This means that we have that e is not safe.

23.1-6: Show that a graph has a unique minimum spanning tree, if for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not by giving a counterexample.

Suppose that for every cut of G, there is a unique light edge crossing the cut. Let us consider two MSTs $T \& T'$ of G. We will show that every edge of T is also in T', which means that $T \& T'$ are the same tree & Hence there is a unique MST!

Consider any edge $(u,v) \in T$. If we remove (u,v) from T, then T becomes disconnected, resulting in a cut $(S, V-S)$. The edge (u,v) is a light edge crossing the cut $(S, V-S)$ & by Excercise 23.1-3).

Now consider the edge $(x,y) \in T'$ that crosses $(S, V-S)$. It, too, is a light edge crossing this cut. Since the light edge crossing $(S, V-S)$ is unique, the edges (u,v) & (x,y) are the same edge. Thus, $(u,v) \in T'$. Since we chose (u,v) arbitrarily, every edge in T is also in T'.



Here, the graph is its own MST & so the MST is unique. Consider the cut $(\{x\}, \{y,z\})$. Both of the edges (x,y) & (x,z) are light edges crossing the cut & they are both light edges.

23.1-7: Argue that if all edge weights of a graph are positive, then any subset of edges that connects all vertices & has minimum ~~that~~ total weight must be a tree. Give an example to show that the same conclusion does not follow if we allow some weights to be nonpositive.

First, we show that the subset of edges of minimum total weight that connects all the vertices is a tree. To see this, suppose not, that it had a cycle. This would mean that removing any of the edges in this cycle would mean that the remaining edges would still connect all the vertices, but would have a total weight that's less by the weight of edge that was removed. This would have contradict the minimality of total weight of the subset of vertices. Since the subset of edges forms a tree, & has minimal total weight, it must be a minimum spanning tree.

To see that this conclusion is not true iff we allow negative edge weights, we provide a construction; Consider the graph K_3 with all ~~vertices~~ edge weights equal to -1. The only minimum weight set of edges that connects the graph has total weights equal to -3, & consists of all the edges. This is clearly not a MST because it is not a tree, which can be easily seen because it has one more edge than a tree on the three vertices should have. Any MST of this weighted graph must have weight that is at least -2.

23.1-8: Let T be a MST of a graph G & 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 edge weights of T'.

Suppose that L' is another sorted list of edge weights of a minimum ST. If $L' \neq L$, there must be a first edge (u,v) in T or T' which is of smaller weight than the corresponding edge (x,y) in the other set. Without loss of generality, assume (u,v) is in T.

Let C be the graph obtained by adding (u,v) to L'. Then we must have introduced a cycle. If there exists an edge on that cycle which is of larger weight than (u,v), we remove it to obtain a tree C' of weight strictly smaller than the weight of T', contradicting the fact that T' is a MST.

Thus, every edge on the cycle must be of lesser or equal weight than (u,v). Suppose that every edge is of strictly smaller weight. Remove (u,v) from T to disconnect it into 2 components. There must exist some edge besides (u,v) on the cycle which would connect these, & since it has smaller weight we can use that edge instead of create a spanning tree with less weight than T, a contradiction. Thus, some edge on the cycle has same weight as (u,v). Replace that edge by (u,v). The corresponding lists $L \& L'$ remain unchanged since we have swapped out an edge of equal weight but the number of edges which $T \& T'$ have in common has increased by 1.

If we continue in this way, eventually they must have every edge in common, contradicting the fact that their edge weights differ somewhere. Therefore, all MSTs have the same sorted list of edge weights.

23.1-9 : Let T be a MST of a graph $G = (V, E)$, and Let V' be a subset of V. Let T' be the subgraph of T included by V', & Let G' be the subgraph of G included by V'. Show that if T' is connected, then T' is MST of G'.

Suppose that there was some cheaper spanning tree than T'. That is, we have that there is some T'' so that $w(T'') < w(T')$. Then, Let S be the edges in T but not in T'. We can then construct a MST of G by considering $S \cup T''$.

This is a sappanning tree since $S \cup T'$ is, & T'' makes all the vertices in V' connected just like T' does.

However, we have that

$$w(S \cup T'') = w(S) + w(T'') < w(S) + w(T') = w(S \cup T') = w(T)$$

This means the we just found a spanning tree that has a lower total weight than a minimum spanning tree. This is a contradiction & so our assumption that there was a spanning tree of V' cheaper than T' must be false.

23.1-10 : Given a graph G & MST T, suppose that we decrease the weight of one of the edges in T. Show that T is still a MST for G. More formally, Let T be a MST for G with edge weights given by weight function w. Choose one edge $(x,y) \in T$ & positive number k, & define the weight function w' by

$$w'(u,v) = \begin{cases} w(u,v) & \text{; if } (u,v) \neq (x,y), \\ w(x,y) - k & \text{; if } (u,v) = (x,y). \end{cases}$$

Show that T is a MST for G with edge weights given by w'.

* We prove by cut. Originally, (x,y) is the light edge in a certain cut (V_1, V_2). Decreasing the weights of (x,y), (x,y) is still a light edge. So T is a MST for G with edge weights given by w'.

* Let $w(T) = \sum_{(x,y) \in T} w(x,y)$. We have $w'(T) = w(T) - k$. Consider any other spanning tree T', so that $w(T) \leq w(T')$. If $(x,y) \notin T'$, then $w'(T') = w(T') \geq w(T) > w'(T)$. If $(x,y) \in T'$, then $w'(T') = w(T') - k \geq w(T) - k = w'(T)$. Either way, $w'(T) \leq w'(T')$ & so T is a MST for weight function w'.

23.1-11: Given a graph G & MST T, suppose that we decrease the weight of one of the edge not in T. Give an algorithm for finding the MST in modified graph.

* If (u,v) is not in MST, decrease the weight of (u,v), we may form a new MST T'.

Condition 1: If the weightest edge e in path from $u \to v$ is greater than edge (u,v). Then we can replace e with (u,v).

Condition 2: If the weightest edge e in path from $u \to v$ is less or equal edge (u,v). Then we need not change.

* If we were to add in this newly decreased edge to the given tree, we would be creating a cycle. Then, if we were to remove any one of the edges along this cycle, we could still have a spanning tree. This means that we look at all the weights along this cycle formed by adding in the decreased edge & remove the edge in the cycle of maximum weight. This does exactly what we want since we could only possibly want to add in the single decreased edge, & then, form there we change the graph back to be a tree in the way that makes its total weight minimized.

14:52
15/10/2024

Kruskal's Algorithm

$G = (V, E)$ is a connected, undirected, weighted graph. $w : E \to R$

```
MST-KRUSKAL (G, w)
① A = ∅
② for each vertex v ∈ G.V                    } O(|V|)
③     MAKE-SET (v)  ⟶ O((V+E)·α(V)) times
④ sort the edges of G.E into nondecreasing order by weight w.  O(E log E)
⑤ for each edge (u,v) ∈ G.E, taken in nondecreasing order by weight
⑥     if FIND-SET(u) ≠ FIND-SET(v)           } O(E)
⑦         A = A ∪ {(u,v)}
⑧         UNION (u,v)
⑨ return A                                   ⟶ O(1)
```

Analysis of ~~Time complexity~~

→ Assuming the implementation of disjoint-set data structure, already seen in chapter 21, that uses union by rank & path compression:

$$O((V+E)\alpha(V)) + O(E \log E)$$

→ Since G is connected, $|E| \geq |V| - 1 \Rightarrow O(E\alpha(V)) + O(E \log E)$

→ $\alpha(|V|) = O(\log V) = O(\log E)$

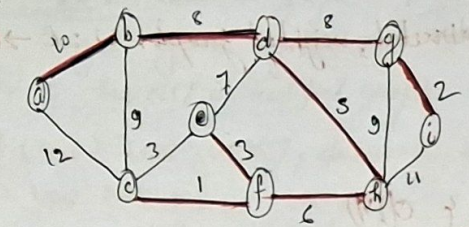
→ Therefore, total time is $O(E \log E)$

→ $|E| \leq |V|^2 \Rightarrow \log|E| = O(2 \log V) = O(\log V)$.

→ $\therefore$ $\boxed{T(G) = O(E \log V)}$ Ans.

→ If edges are already sorted, $O(E\alpha(V))$, which is almost linear.

Ex:-

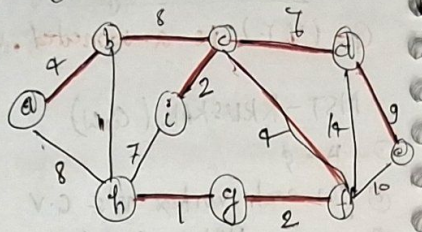


Kruskal's algorithm

1(c,f) : safe
2(g,i) : safe
3(e,f) : safe
3(c,e) : reject [create cycle]
5(d,h) : safe
6(f,h) : safe
7(c,d) : reject
8(a,g) : safe
8(b,d) : safe
9(b,c) : reject
9(g,h) : reject
10(a,b) : safe

Examined (c,e) before (e,f) then would have (c,e) safe & (e,f) rejected

Fig 23.4



1(h,g) : safe
2(g,f) : safe
2(c,i) : safe
4(a,b) : safe
4(c,f) : safe
7(c,d) : safe
7(h,i) : reject
8(a,h) : reject
8(b,c) : ~~safe~~ Reject
9(d,e) : safe
10(e,f), 11(b,h), 14(d,f) Reject

* →

PRIM'S ALGORITHM

PRIM(V, E, w, r) → start node (root); w → weight; E → edge; V → vertex

```
1  Q = φ
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 ≠ φ
8      do u = EXTRACT-MIN(Q)
9         for each v ∈ Adj[u]
10            do if v ∈ Q & w(u,v) < key[v]
11                then π[v] = u
12                     DECREASE-KEY(Q, v, w(u,v))
```

Analysis of Time Complexity

Depends on how the priority queue is implemented:

→ Suppose Q is binary heap.

- Initialize Q & first for loop : $O(V \log V)$
- Decrease key of r : $O(\log V)$
- While loop :
  - $|V|$ Extract-Min calls $\Rightarrow O(V \log V)$
  - $\leq |E|$ Decrease-key calls $\Rightarrow O(E \log V)$
- Total : $O(E \log V)$

→ Suppose we could do DECREASE-KEY in $O(1)$ amortized time

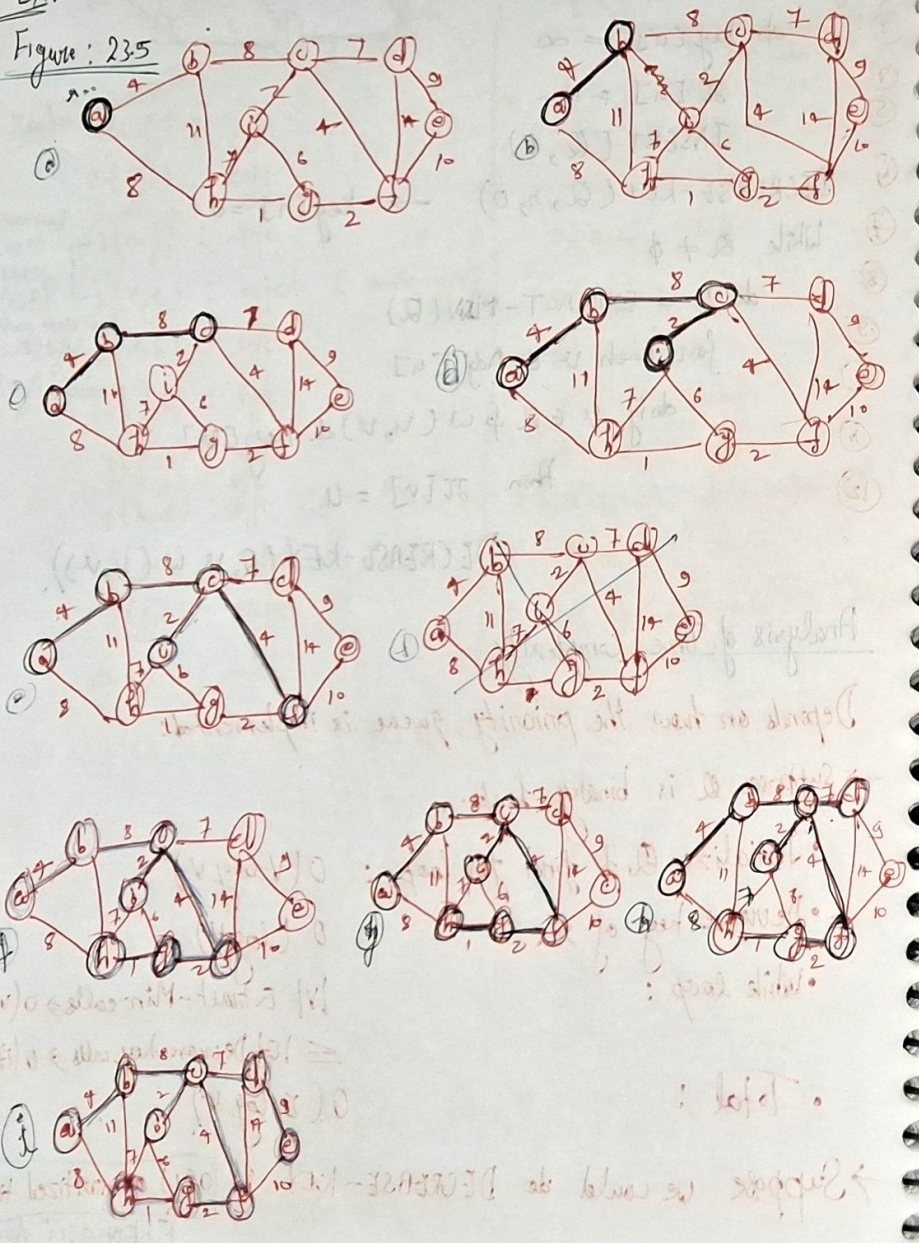
Fibonacci heap

Then $\leq |E|$ DECREASE-KEY calls take $O(E)$ time altogether

Total time = $O(V \log V + E)$

Ex:-

Figure: 23.5



Excersises:

23.2-1 : Kruskal's algo. can return diff. spanning trees for the same I/p 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 Algo. so that the algo. returns T.

* The reason why there may be several diff. MST is that we have several choice on same weighted edge.

Given MST T, we wish to sort the edges in Kruskal's algo. such that it produces T. For each edge e in T simply make sure that it preceeds any other edge not in T with weight $w(e)$.

{OR}

* Suppose that we wanted to pick T as our MST. Then, to obtain this tree with Kruskal's Algo., we will order the edges first by their weights, but then will resolve ties in edge weights by picking an edge first if it is contained in the MST & treating all the edges that aren't in T as being slightly larger, even though they have the same actual weights.

With this ordering, we will still be finding a tree of the same weight as all the MST, $w(T)$. However, since we prioritize the edges in T, we have that we will pick them over any other edges that may be in other MSTs.

23.2-2 : Suppose that we represent the Graph $G=(V,E)$ as an adjacency matrix. Give an example implementation of prim's algo. for this case that runs in $O(V^2)$ time.

```
PRIM-ADJ ( G, w, r)
  initialize A with every entry = (NIL, ∞)
  T = {r}
  for i = 1 to V
      if Adj[r,i] != 0
          A[i] = (r, w(r,i))
  for each u in V-T
      k = min {A[i].2}  // Access to w.
      T = T U {k}
      k.π = A[k].1  // Access to k
      for i = 1 to V
          if Adj[k,i] != 0 && Adj[k,i] < A[i].2
              A[i] = (k, Adj[k,i])
```

$T(n) = O(V^2)$

23.2-3: For a sparse graph $G = (V, E)$, where $|E| = \Theta(V)$, is the implementation of prim's algorithm with a Fibonacci heap asymptotically faster than the binary-heap implementation? what about for a dense graph, where $|E| = \Theta(V^2)$? How must the sizes $|E|$ & $|V|$ be related for the Fibonacci-heap implementation to be asymptotically faster than the binary-heap implementation?

Prim's Algo. implemented with a Binary heap has runtime $O((V+E)\log V)$, which in the sparse case, is just $O(V \log V)$. The implementation with Fibonacci heap is

$$O(E + V\log V) = O(V + V\log V) = O(V \log V).$$

→ In the sparse case, the two algorithms have the same asymptotic runtimes.

→ Dense case:

* The binary heap implementation has a runtime of

$$O((V+E)\log V) = O((V+V^2)\log V) = O(V^2 \log V).$$

* The Fibonacci heap implementation has a runtime of

$$O(E + V\log V) = O(V^2 + V\log V) = O(V^2)$$

So, in the dense case, we have that the fibonacci heap implementation is asymptotically faster.

→ The Fibonacci heap implementation will be asymptotically faster so long as $E = \omega(V)$.

Suppose that we have some function that grows more quickly than linear, say if $E = f(V)$.

→ The binary heap implementation will have runtime of

$$O((V+E)\log V) = O((V + f(V))\log V)$$
$$= O(f(V)\log V)$$

However, we have that the runtime of the Fibonacci heap implementation will have runtime of

$$O(E + V\log V) = O(f(V) + V\log V)$$

This runtime is either $O(f(V))$ or $O(V \log V)$ depending on if $f(V)$ grows more orders quickly than $V\log V$ respectively.

In either case, we have that the runtime is faster than $O(f(V)\log V)$

23.2-4: Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

If W is a constant we can use counting sort

→ Sorting the edges: $O(E \log E)$ time.

→ $O(E)$ operations on a disjoint-set forest taking $O(E\alpha(V))$.

The sort dotsv dominates & hence the total time is $O(E \log E)$.

Sorting using counting sort when the edges fall in the range $1, \ldots |V|$ yields $O(V+E) = O(E)$ time sorting. The total time is then $O(E\alpha(V))$.

If the edges fall in the range $1, \ldots W$ for any constant W we still need to use $\Omega(E)$ time for sorting & the total running time cannot be improved further.

23.2-5: Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Prim's algorithm run? what if the edge weights are integers in the range from 1 to W for some constant W?

The running time of prim's algo. is composed:

→ $O(V)$ initialization.

→ $O(V)$ time for Extract-MIN.

→ $O(E)$ time for DECREASE-KEY.

If the edges are in the range $1, \ldots, |V|$ the Van Emde Boas priority queue can speed up EXTRACT-MIN & DECREASE-KEY to $O(\lg\lg V)$.
Thus yielding a total running time of $O(V \lg\lg V + E \lg\lg V) = O(E \lg\lg V)$

If the edges are in the range from 1 to W, we can implement the queue as an array $A[1 \ldots W+1]$ where the ith slot holds a doubly linked list of the edges with weight i.
The $(W+1)$st slot contains ∞. EXTRACT-MIN now runs in $O(W) = O(1)$ times since we can simply scan for the first nonempty slot & return the first element of that list.
DECREASE-KEY runs in $O(1)$ times as well since it can be implemented by moving an element from one slot to another.

***23.2-6:** Suppose that the edge weights in a graph are uniformly distributed over the half-open interval $[0,1)$. Which algo., Kruskal's or Prim's, can you make run faster?

* Kruskal, using bucket sort.

* For input drawn from a uniform distribution (OR?) I would use bucket sort with Kruskal's algo. for expected linear time sorting of edges by weight. This would achieve expected runtime $O(E\alpha(V))$

***23.2-7:** Suppose that a graph G has a MST already computed. How quickly can we update the MST if we add a new vertex & incident edges to G?

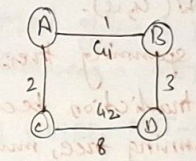
→ If there is only one edge, just add this edge.

→ If there are $k (k > 1)$ edges, then we need to remove $k-1$ edges. We can find cycles by Union-Find & remove the weightest edge in an Union. This algorithm need $(k-1)$ passes.

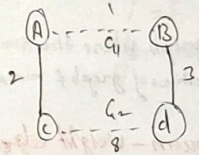
23.2-8: Professor Toole proposes a new divide-&-conquer algo. for computing MSTs, which goes a follows, Given a graph $G = (V, E)$, partition the set V of vertices into 2 sets V_1 & V_2 such that $|V_1|$ & $|V_2|$ differ by at most 1. Let E_1, E_2 be the set of edges that are incident only on vertices V_1 & V_2 respectively. Recursively solve a MST problem on each of the two subgraphs $G_1 = (V_1, E_1)$ & $G_2 = (V_2, E_2)$. Finally, select the MST minimum-weight edge in E that crosses the cut (V_1, V_2) & use this edge to unit the resulting two MSTs into a single spanning tree.

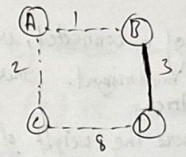
Either argue that the algo. correctly compute a MST of G, or provide an example for which the algo. fails.

Disproof:
We argue that the algorithm fails. Consider the graph G below. We partition G into V_1 & V_2 as follows: $V_1 = \{A, B\}$, $V_2 = \{C, D\}$, $E_1 = \{(A,B)\}$, $E_2 = \{(C,D)\}$. The set of edges that cross the cut is $E_C = \{(A,C), (B,D)\}$



Now, we must recursively find the MSTs of G_1 & G_2. We can see that in this case, $MST(G_1) = G_1$ & $MST(G_2) = G_2$. The MSTs of G_1 & G_2 are shown below on the left.





The minimum weighted edge of the two edges across the cut is edge (A,C). So (A,C) is used to connect G_1 & G_2.

This is the MST returned by professor Toole algo. It is shown above to the right.

→ We can see that the MST returned by Toole's algo. is not the MST of G, therefore, the algorithm fails.

Problem: 23-3 Bottleneck Spanning tree

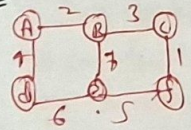
A bottleneck spanning tree T of an undirected graph G is a spanning tree of G whose largest edge weight is minimum over all spanning trees of G. We say that the value of the bottleneck spanning tree is the weight of the maximum-weight edge in T.

(a) Argue that minimum spanning tree is a bottleneck spanning tree.

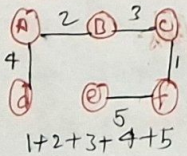
To see that every MST is also a bottleneck spanning tree. Suppose that T is a MST. Suppose there is some edge in it (u,v) that has a weight that is greater than the weight of the bottleneck spanning tree. Then, Let V_1 be the subset of vertices of V that are reachable from u in T, without going through v. Define V_2 symmetrically. Then, consider the cut that separates V_1 from V_2. The only edge that we could've added across this cut is the one of the minimum weight, so we know that there are no edge across this cut of weight 'less' than $w(u,v)$. However, we have that there is a bottleneck spanning tree with less than that weight. This is a contradiction because a bottleneck spanning tree, since it is a spanning tree, must have an edge across this cut.

* MST : A spanning tree of a connected, undirected graph where the sum of the edge weights is minimized. Includes total vertices of graph & minimize the weight of the tree.
* BST : A spanning tree where the weight of the maximum-weight edge in the tree (also called the "bottleneck edge") is minimized. To minimize the most expansive edge in the tree, regardless of the total sum of all edges.

Ex:

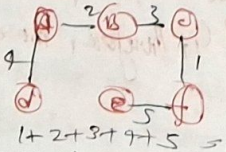


MST



1+2+3+4+5

BST



1+2+3+4+5 = 15

(b) Give a linear-time algo that given a graph G & an integer b, determines whether the value of the bottleneck spanning tree is at most b.

To do this, first process the entire graph & remove any edges that have weight greater than b. If the remaining graph is connected, we can just arbitrarily select any tree in it. It will be a bc a BST of weight at most b. Testing connectivity of a graph can be done in linear time by running a BFS & then making sure that no vertices remain white at the end.

23-4 Alternative MST-Algorithm

In this problem, we give pseudocode for three different algorithms. Each one takes a connected graph & a weight function as I/P & returns a set of edges T. For each algo. either prove that T is a MST or prove that T is not a MST. Also describe the most efficient implementation of each algo., whether or not it computes a MST.

(a) MAYBE-MST-A (G, w)

```
sort the edge into non-increasing order of edge weights w
T = E
for each edge e, taken in non-increasing order by weights
    if T - {e} is a connected graph
        T = T - {e}
return T.
```

This does not return an MST. To see this, we'll show that we never remove an edge which must be part of a MST. If we remove e, then e cannot be a bridge, which means that e lies on a simple cycle of the graph. Since we remove edges in non-increasing

order, the weight of the every edge on the cycle must be less or equal to that of e.

24: Single-Source Shortest Paths

Shortest paths

How to find the shortest route b/w 2 points on a map.

Input:

- Directed graph $G = (V, E)$
- Weight function $w : E \to R$

Weight of path $p = \langle v_0, v_1, \ldots v_k \rangle$

$$= \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

= Sum of edges weights on path p

Shortest-path weight u to v:

$$\delta(u,v) = \begin{cases} \min \{ w(p) : u \overset{p}{\to} v \}; & \text{if there exists a path } u \to v, \\ \infty & ; \text{ otherwise.} \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u,v)$.

Variants

→ Single-source: Find shortest path from a given source vertex $s \in V$ to every vertex $v \in V$.

→ Single-destination: Find shortest paths to a given destination vertex

→ Single-pair: Find shortest path from u to v. No way known that's better in worst case than solving single-source

→ All-pairs: Find shortest path from u to v for all $u, v \in V$. We'll see algorithms for all-pairs in the next chapter