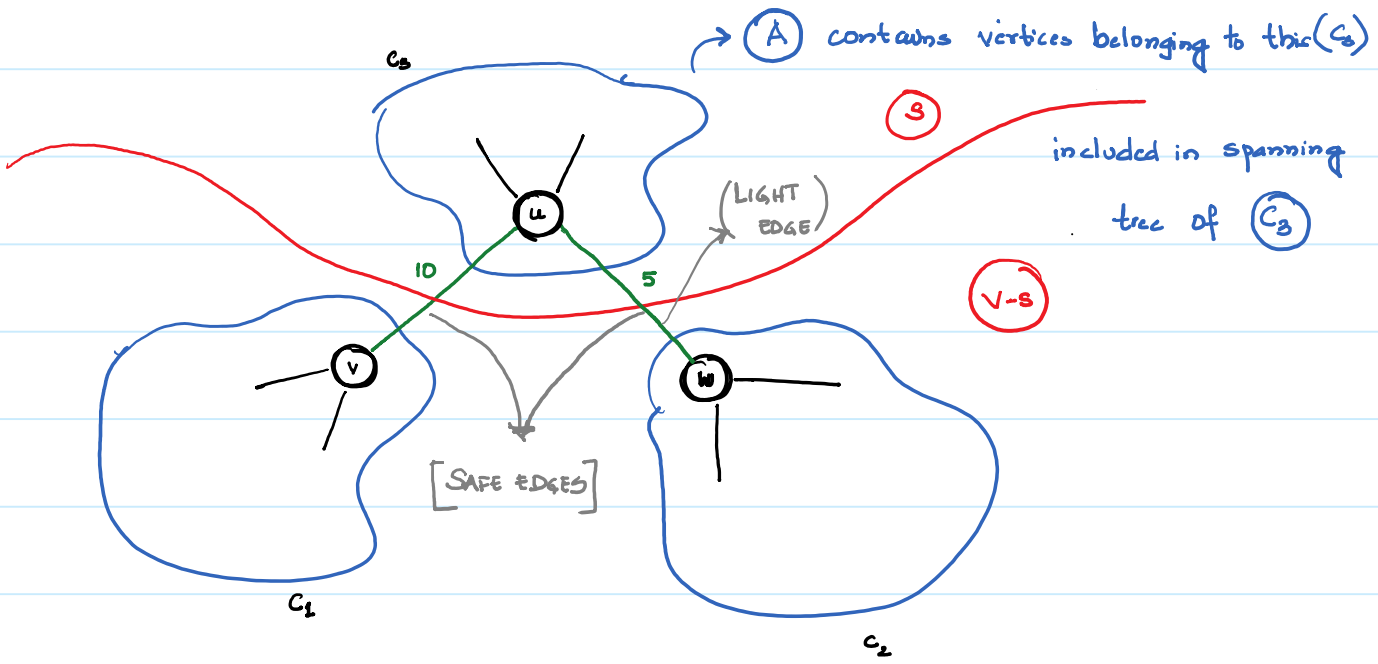
23.1-2) Professor Sabatier conjectures converse of Theorem 23.1. Let $G = (V, E)$ be a connected undirected graph with a real valued weight function w defined on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V-S)$ be any cut of G that respects A, and let (u,v) be a safe edge for A crossing $(S, V-S)$. Then (u,v) is a light edge for the cut. Show that professor's conjecture is incorrect by giving a counter example.

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. Let $(S, V-S)$ be any cut of G that respects A. Let (u,v) be light edge crossing $(S, V-S)$ then edge (u,v) is safe for A.

INTUITION : Connecting different connected components by single edges (E)



C_1, C_2, C_3 represent three connected components (sub) joined to main component C.

Here minimum spanning tree of the whole graph formed by component (C) includes both the edges (u,v) , (u,w)

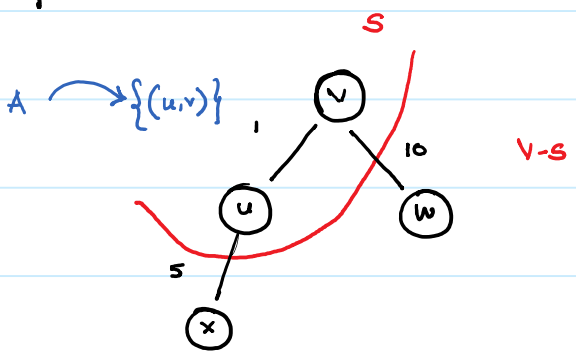
Now define a cut as shown in RED to include

c_1 in S and c_2, c_3 in V-S

It can be observed that both edges (u,v) , (u,w) are safe but only one of it i.e (u,w) is a light edge

Thus proving professors statement false :

Trivial example :



G(V,E)

MST for G is T = G

A is subset of MST - (T)

Edges (u,x) , (v,w) are safe but only (u,x) is LIGHT EDGE

Proving professors converse false

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 true by giving a counter example.

PROOF BY CONTRADICTION :

Let T, T' be two minimum spanning trees corresponding to graph $G(V,E)$ with property, for every cut of the graph there is unique light edge

For T, T' to be distinct they should differ by atleast two edges

Let the differed edges be (u,v), (a,b) such that

$$\{(u,v) \notin T',\ (u,v) \in T\} \text{ & } \{(a,b) \in T',\ (a,b) \notin T\}$$

Let us define our cut $S = \{u, a\}$. & $V-S$ contains $\{v, b\}$.

As there exists a UNIQUE LIGHT EDGE spanning any cut.

Now consider T, T' only one of (u,v), (a,b) (or) something else can be a light edge spanning this cut.

if (u,v) is light edge of this cut then

$(a,b) \in T'$ can be replaced with (u,v) forming a more minimum SPT

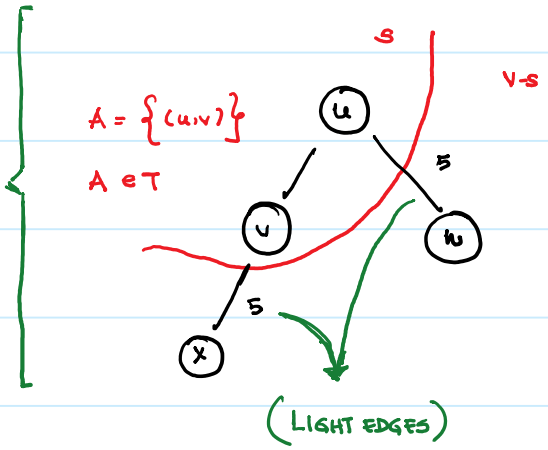
Wlg if (a,b) is light edge (u,v) in T can be replaced.

So, T, T' are no longer minimum spanning trees → [CONTRADICTION !!]

COUNTER EXAMPLE FOR CONVERSE :

Consider this trivial MST for graph G (V,E) defined as follows :

UNIQUE MST



Here the graph G is itself MST

A is subset of MST (Graph itself)

Cut S with vertices {u,v}

V-S with vertices {w,x}

have edges (u,w), (v,x) both are

(LIGHT EDGES) with same weight

but the graph has (UNIQUE MST)

28.1-10) Given a graph G and a minimum spanning tree T, suppose we decrease the weight of one of the edges in T. Show that T is still a minimum spanning tree for G. More formally, let T be a minimum spanning tree for G with edge weights given by weight function w. Choose one edge $(x,y) \in T$ and a positive number k and 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 minimum spanning tree for G with edge weights given by w'

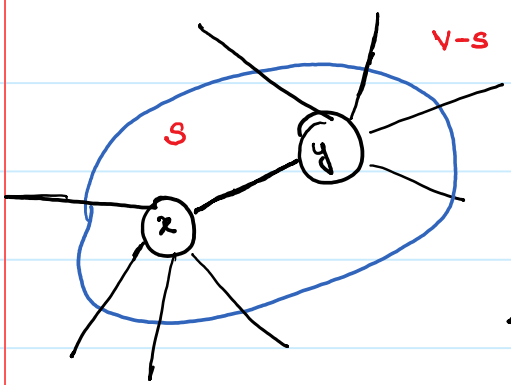
For graph G, $(x,y) \in$ minimum spanning tree (MST) $- T$

Weight of edge (x,y) is decreased

T.B.P : T is still the minimum spanning tree (MST)

Consider the following in cut in G where

$S = \{x, y\}$ $\quad V - S = V \setminus \{x, y\}$



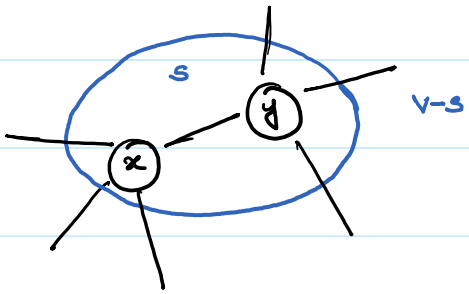
Given edge (x,y) is part of MST

To include (x,y) in MST

$[(w(x,y)) + (\text{light edge spanning this cut})]$

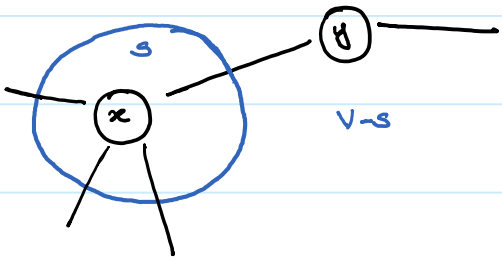
$\leq [(\text{light edge spanning this cut}) + (\text{light edge spanning the cut formed after including one of } (x,y) \text{ in } V-S)] \rightarrow (1)$

The above equation shown pictorially. →



I

without loss of generality assume light edge corresponds to one of edges of vertex (y)



II

Now, for this picture edge (x,y) should be one of the light edges corresponding to the defined cut $S = \{x\}$, $V-S = V \setminus \{x\}$ for it to be part of MST.

OPERATION : $[w'(x,y) = w(x,y) - k]$ and keeping all the other edges same.

For the graph G consider (II) cut which establishes for (x,y) to be part of the MST (x,y) should be one amongst <u>light edges</u>

So, all other edges spanning the cut $\geq w(x,y)$

Now modified graph has $w'(x,y)$ instead of $w(x,y)$.

$w'(x,y) < w(x,y) \leq$ All other edges spanning the cut

So, $w'(x,y)$ shall be the only light edge spanning the cut now

G T which includes (x,y) shall remain the <u>minimum spanning</u> tree.

Athmaja

Consider MST → T for original Graph G (V, E)

In this MST for edge (x, y) weight has been decreased by k

Let T' be the new MST for graph G' with reduced weight w'

For T' to be different than T

[NEW MST T'] [ORIGINAL MST]

$w'(T') < w(T) - k$ [REDUCED WEIGHT] → (1)

Now take this (T') → for original graph (G)

here (x, y) may or may not be in T'

$\Rightarrow w(T') \leq w'(T') + k$ [REDUCTION] → (2)

($w(T')$: WEIGHT IN ORIGINAL GRAPH FOR NEW MST T')

($w'(T')$: WEIGHT IN MODIFIED GRAPH FOR NEW MST)

If (x, y) is not in T' → $w(T') = w'(T')$

if present → $w(T') = w'(T') + k$

From (1) & (2) $w'(T') < w(T) - k$, $w(T') \leq w'(T') + k$

$\Rightarrow \boxed{w(T') < w(T)}$

CONTRADICTION !!! that T is not (MST) under w

25-3) 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 bottleneck spanning tree is the weight of the maximum-weight edge in T.

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

b) Give a linear time algorithm that given a graph G and an integer b, determines whether the value of the bottleneck spanning tree is atmost (b)

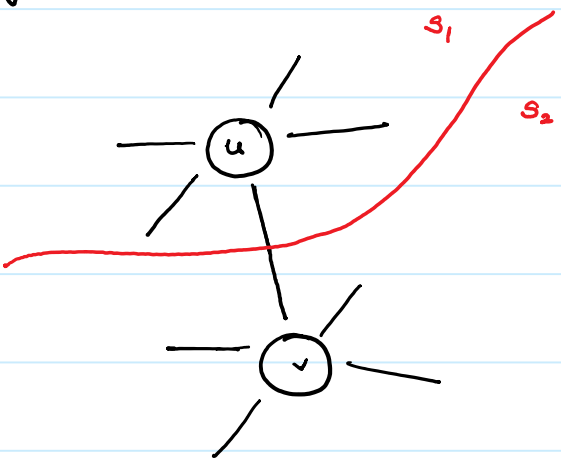
a) Bottleneck spanning tree : SPANNING TREE of G whose largest weight is MINIMUM over all spanning trees of G.

Value of BOTTLENECK SPANNING TREE maximum weight of edge in (T)

MINIMUM SPANNING TREE ⟹ BOTTLENECK SPANNING TREE

Let T be MST G (u,v) be some edge of MST whose weight is larger than the value of bottleneck spanning tree.

Consider graph G(V,E) as follows



S_1 contains all vertices reachable from u

S_2 contains all vertices reachable from v

since (u,v) is part of MST of graph then

for the cut defined above formed by (S_1, S_2), (u,v) should be one of the light edge.

As there exists some edge in bottleneck spanning tree with lesser weight than (u,v) crossing the cut (S_1, S_2).

↳ This is a contradiction to our assumption that (u,v) is part of MST as the lighter edge of bottleneck spanning tree can be included here to form a much lighter minimum spanning tree

b) Linear time algorithm to check for graph G and almost (b)

$O(E)$

```
for each e in G.E do
    if w(e) > b then
        G.remove(e)
```

First let us remove all edges with weight greater than [b]

$O(V+E)$ ← `if (IS_CONNECTED(G)):` → BFS (or) DFS to check if the resulted graph after discarding edges is connected (or) not

```
    return TRUE
return FALSE
```

Any path traced by DFS (or) BFS shall be bottleneck spanning trees of weight almost (b)

RUNTIME : $O(V+E)$ (linear)

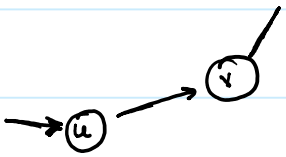
24.3-6) We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \le r(u, v) \le 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Reliable paths consist of paths with maximum probability of reaching the other vertex.

Let $u_1, u_2, \ldots u_k$ be vertices in reliable path from u to v.

RELIABILITY : $r(u, u_1)\, r(u_1, u_2) \cdots r(u_k, v)$

OPTIMAL SUBSTRUCTURE — Reliable path consists of reliable sub paths.



INTUITION — Similar to Dijkstra's algorithm

we need to modify RELAXATION procedure such that reliability of reaching vertex (v) when passing through vertex (u) is increased

RELIABILITY OF CHANNEL (u,v)

if $\longrightarrow$ $R(u) \left(r(u, v) \right) > R(v)$ $\longrightarrow$ include (u, v) in the path.

RELIABILITY OF REACHING (u) — RELIABILITY OF REACHING (v)

Runtime also remains the same as DIJKSTRA'S ALGORITHM

$O(V \log V + E)$ if FIBONOCCI HEAPS are used.

MAX-RELIABILITY-ALGORITHM (G, r, s) :

INITIALIZE-SINGLE-SOURCE (G, s)

$S = \phi$

$Q = G.V$

O(V) ← while $Q \neq \phi$:

O(lgV) ← u = EXTRACT-MAX-RELIABLE-CHANNEL (Q)

(FIBONACCI HEAPS)

$S = S \cup \{u\}$

O(E) AGGREGATE ANALYSIS

for each vertex $v \in G.adj[u]$

RELAX (u, v, r)

[Can be stopped once destination is reached]

O (VlgV + E)

INITIALIZE-SINGLE-SOURCE (G, s) :

for each vertex v in G.V :

(// Initial reliability of all v) $R[v] = 0$

$R[s] = 1$ // Reliability of reaching the source

RELAX (u, v, r) :

if $R[u] \cdot r(u,v) > R[v]$

$R[v] = R[u] \cdot r(u,v)$

$v.\pi = u$.

EXTRACT-MAX-RELIABLE-CHANNEL (Q) :

↳ It maintains a priority queue and extracts the maximum reliability channel (edge) of all the edges spanning the cut (S, V-S)

RUNTIME

This modified DIJKSTRA's algorithm indeed computes the MOST RELIABLE PATH between vertices (u, v) in O(VlgV + E) (FIBONACCI HEAPS)

24-3-8) Let $G = (V, E)$ be a weighted, directed graph with non-negative weight function $w : E \to \{0, 1, \ldots W\}$ for some non-negative integer W. Modify Dijkstra's algorithm to compute shortest paths from a given source vertex s in $O(WV + E)$

Given, $G \to$ Weighted, directed graph with non-negative weights

$w : E \to \{0, 1, 2 \ldots W\}$

ORIGINAL DIJKSTRA'S ALGORITHM:

PRIORITY QUEUE → INSERT, EXTRACT MIN, DECREASE-KEY

INITIALISE-SINGLE-SOURCE (G, s):

$S = \emptyset$

$Q = G.V$

$O(V)$ ← while $Q \neq \emptyset$ } $O(V \lg V)$

$O(\lg V)$ ← u = EXTRACT-MIN (Q) } $O(V \lg V)$

→ [IMPROVEMENT POSSIBILITY]

AMORTIZED FOR FIBONACCI HEAPS

$S = S \cup \{u\}$

$O(E)$ AGGREGATE ANALYSIS { for each vertex $v \in G.Adj[u]$ } $O(E)$

RELAX (u, v, w)

RELAX (u, v, w)

if $v.d > u.d + w(u, v)$

$v.d = u.d + w(u, v)$

$v.\pi = u$

Running time depends on implementation of PRIORITY-QUEUE

FIBONACCI HEAPS: $O(V \lg V + E)$.

Idea of improvement: As we now know the range of weights for edges.

We can use a data structure to store vertices as values of indexed by current shortest distance from the source

So, we maintain an array of size VW (worst case scenario) where weights of all edges is W and lie on straight line

Always traverse from 0 to VW and fetch the vertex first reached. i.e

START

INITIAL :

A

0	s
1	
2	
3	
4	
⋮	
16	
17	

2nd iteration after relaxation

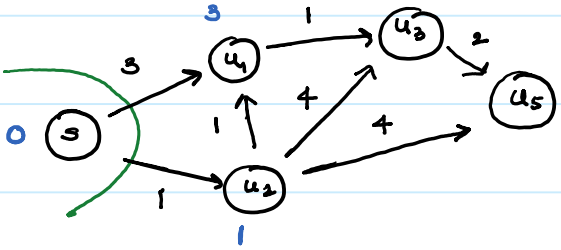
START

[TRAVERSE]

0	
1	u_2
2	
3	u_1
4	
⋮	
16	
17	

Picks up u_2

& relaxes all adjacent vertices

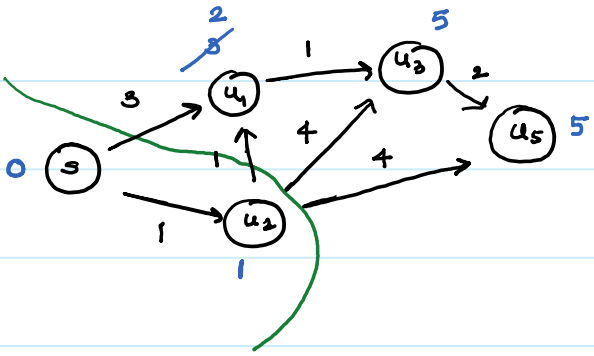


START

0	
1	
2	u_1
3	
4	
5	u_2, u_5
⋮	
16	
17	

[TRAVERSE]

Picks u_1 & relaxes

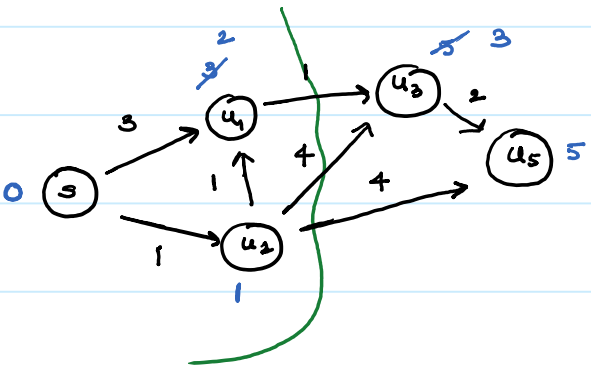


START

0	
1	
2	
3	u_3
4	
5	u_5
⋮	
16	
17	

[TRAVERSE]

Pick u_3 and relax



MINIMUM DISTANCE FROM SOURCE KEEPS INCREASING

MODIFIED-ALGORITHM :

```
INITIALISE-SINGLE-SOURCE (G,s)
A[0].insert (s)        // Initialize array
S = ϕ                  of size (VW+2)
Q = G.V
k = 0
while Q ≠ ϕ :
    // Maintain index
    of minimum (k)     while A[k] ≠ NULL do
                           k = k+1
                       u = A[k].head()
                       A[k].delete(u).
    // RELAXATION
    and updation of    for each vertex v ∈ G.adj[u] do
    data structure values.     if v.d > u.d + w(u,v) then
                                   A[v.d].delete(v)
                                   v.d = u.d + w(u,v)
                                   A[v.d].insert(v)
                                   v.π = u
```

O(VW) [AGGREGATE ANALYSIS]

as k can go from 0 to VW+1

O(E) [AGGREGATE ANALYSIS]

All of the edges are looked only once.

```
INITIALIZE-SINGLE-SOURCE (G,s) :
for each v in G.V
    v.d = (VW+1)  → in place of ∞
    v.π = NULL
s.d = 0.
```

Thus RUNTIME : O(VW+E).

25.3 -8) Suppose $w(u,v) \geq 0$ for all edges $(u,v) \in E$. What is the relationship between the weight functions w and $\hat{w}$?

All weights are non-negative :

$$w(u,v) = \hat{w}(u,v)$$

↳ Values computed as shortest distances on running BELLMAN-FORD ALGORITHM will be ZERO

⇓

Computing G' of JOHNSON'S ALGORITHM

↳ involves placing edge weight of ZERO from s to all vertices

↳ As there are no negative edges. Every path within the graph has no negative edges, its cost cannot be negative and does not beat the trivial path from s to any other vertex

⇓

As we have $h(u) = h(v)$, re weighting does not change weights of the graph. i.e adds & subtracts [ZERO]