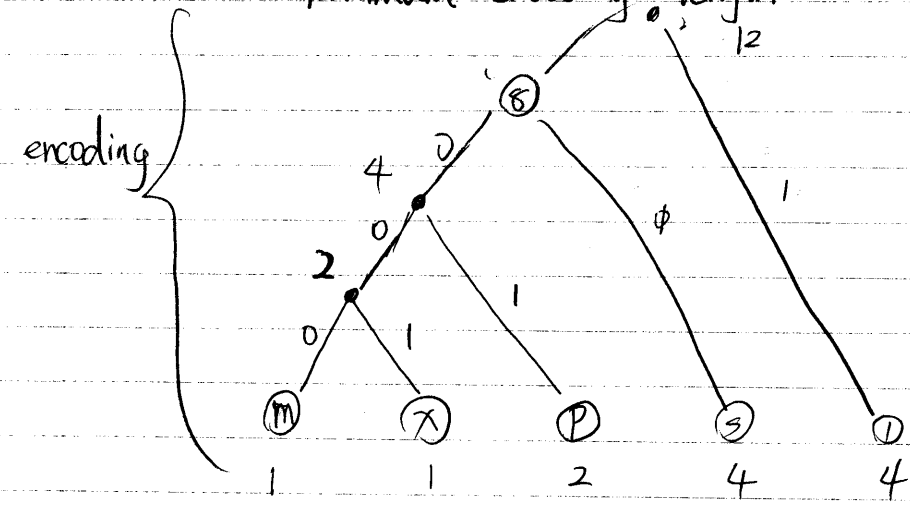
2018.3.27 Lecture 16 5.2 Huffman code

Huffman code:

optimal prefix-free code

↓ minimum encoding length



$\sum_x f_x l_x$ (frequence x length)

encoding length / cost

m = 1x4 =4

x = 1x4 =4

p = 2x3 =6

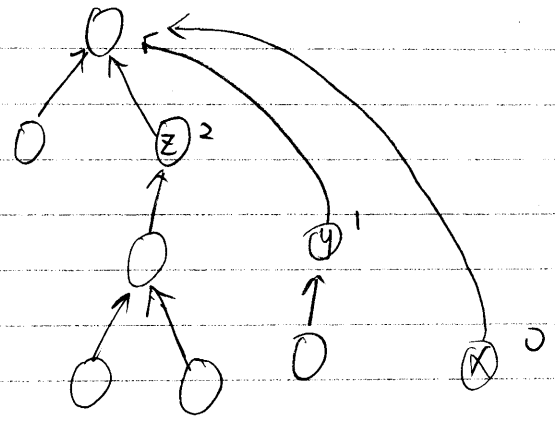
s = 2x4 =8

i = 4x1 =4

fixed code length 36 vs Huffman code 26 bit.

2018.3.27 Lecture 16 5.1 Minimum spanning tree

Trick : path compression



find (x) :

we also change the parrent pointer for all node on the path to point to the root !

amortized analysis

1). find → $O(\log |V|)$ in worst case

subsequent operations are $O(1)$

overall amortized cost

$O(\log^* |V|)$

# of repeates application of $\log$ before we ~~we~~ got the value 1

$|V| = 2^{65536}$

1) $\log_2 |V| = 65536 = 2^{16}$

2) $\log_2 2^{16} = 16 = 2^4$

3) $\log_2 2^4 = 4 = 2^2$

4) $\log 2^2 = 2 = 2^1$

5)

$\log^* 2^{65536} = 5$

$2 \times |E|$ find operation

$= O(|E| \cdot \log^* |V|) \simeq O(5|E|)$

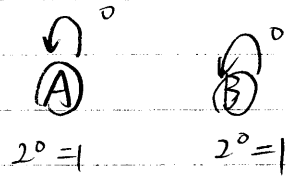
↑ practical size

2^{65535}

2018.3.27 Lecture 16 5.1 Minimum spanning tree

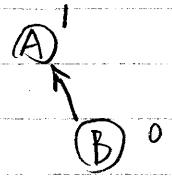
☆ proof by induction

base r=0



$2^0 = 1$ $2^0 = 1$

base r=1



$\geq 2^1 = 2$

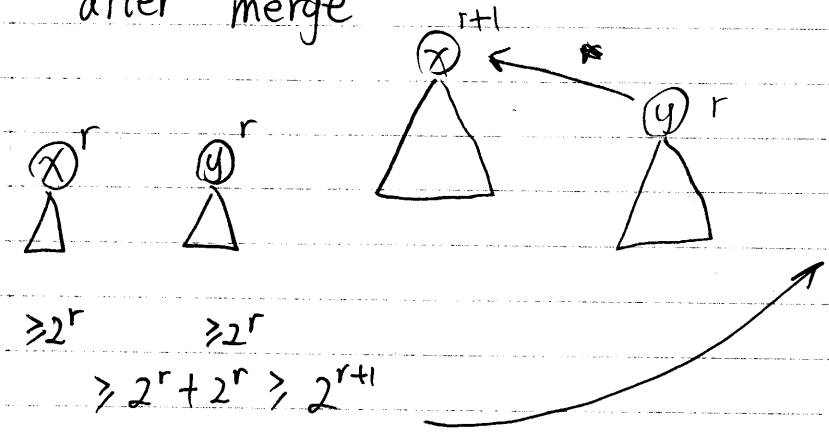
node in that component

Induction: 2 rank r node

x, & y

$|C_x| \geq 2^r$ $|C_y| \geq 2^r$

after merge



$\geq 2^r$ $\geq 2^r$

$\geq 2^r + 2^r \geq 2^{r+1}$

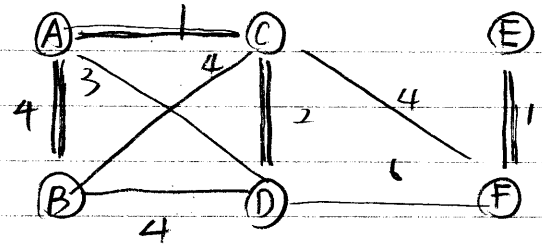
$n \geq 2^r$, rank of final component

$\log(n) \geq r$

⇒ largest rank to $\log n = \log |V|$

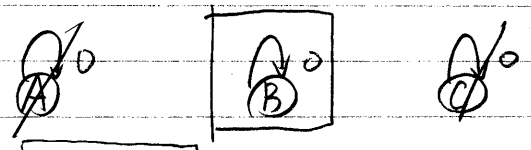
Trick: path Compression

2018. 3. 27 Lecture 16 5.1 Minimum spanning tree

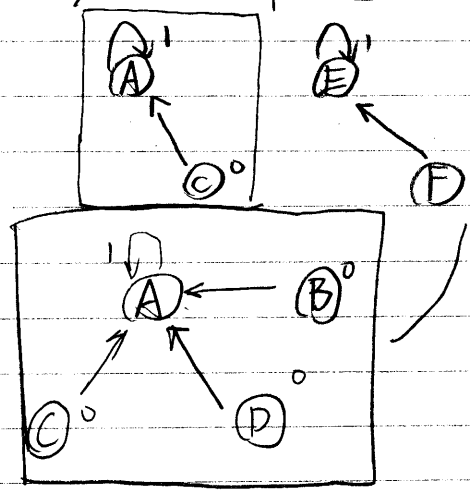


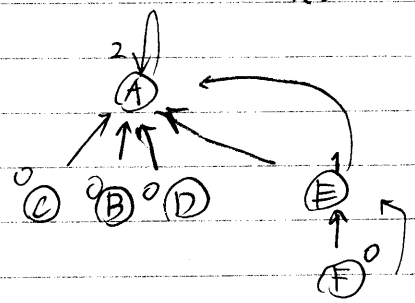
union: only look at roots of the two components

(a). If C_x & C_y's roots have the same rank, increase C_x's rank by 1 — C_y's root point to C_x



(b). If C_y's rank is lower, just change the root pointer to C_x's






```
find(x):
    while (x ≠ π(x))
        x = π(x)
    return x
```

cost of find(x)? $O(\log |v|)$

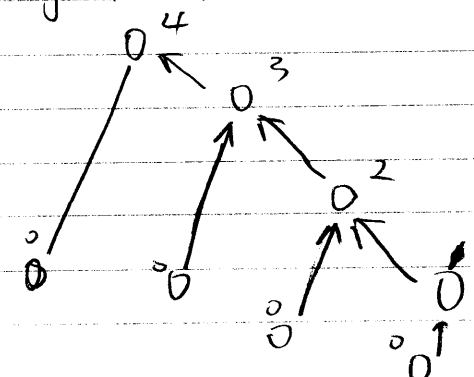
rank of a component determine the running time

find(x) = $O(r)$ time

where r is the rank of C_x

(at most r steps for finding (x)

goal: show that $r = O(\log |v|)$



Observation: If rank of a component in r then, there are at least 2^r nodes that component.

2018.3.27 Lecture 16 5.1 Minimum Spinning tree.

MST: Minimum Spanning tree

Kruskal's Algorithm

1). sort the edge in increasing order

2). add edge as long as there are no cycle in the partial MST

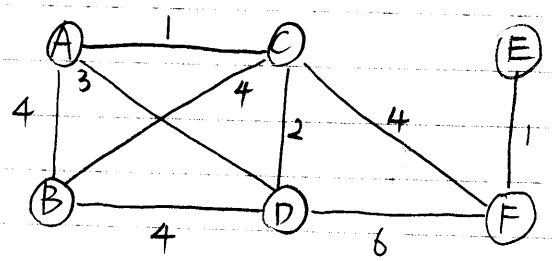
$O(|E| \cdot |V|)$

union-find data structure

Kruskal's maintain a set of connected components

partial tree

forest ← MST



$O(E \log |E|)$ 1) Sort the edge in increase order

2). Create the union-find data structure on V

$O(|V|)$ { A 0, B 0, C 0, D 0, E 0, F rank of the node

$\pi(x)$: parent pointer for x

$r(x)$: rank of x — use rank to calculate the tree

x is a root iff $x = \pi(x)$

3). For each edge $e = (x, y)$: in sorted order

$O(1)$ if find(x) ≠ find(y): } ⇒ looking at roots, x, y belong to different compon

C_{xy} = union(C_x, C_y)

remove C_x, C_y

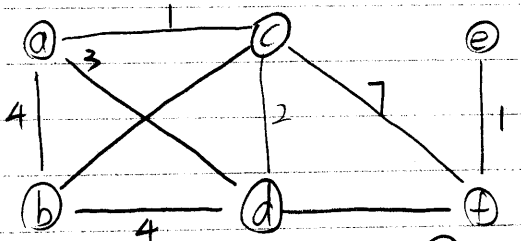
- proportional to the height of tree
- make short tree poin to high tree

total: $O(|E| \log |V|)$

→ $O(|E| \log |V|)$

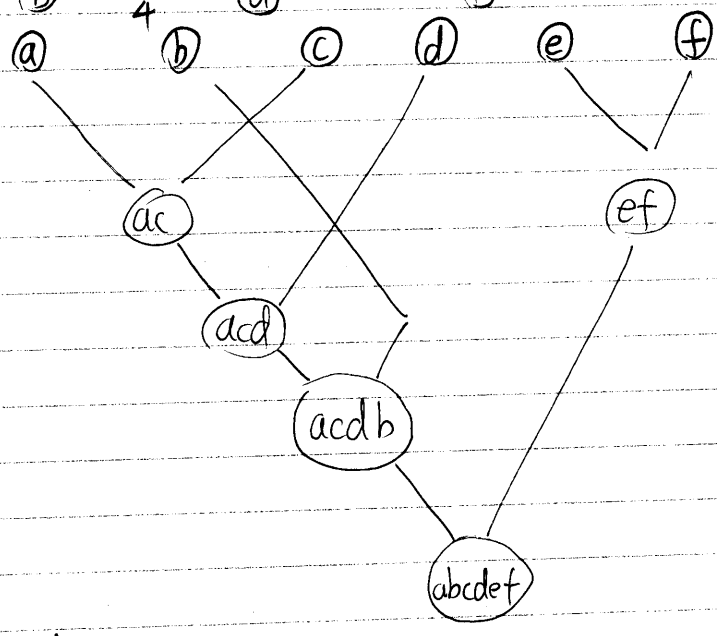
→ $O(5|E|)$

2017.3.23 Lecture 15 5.1 Minimum spanning tree.



Union-find data structure

1).



cycle can only be with a component

a - d

find operation : are they part of the same component ?

$O(\log |V|)$ time

union operation : $O(1)$ merge 2 component into 1

$E \cdot \log|V| + O(|E| \cdot \log|V|)$

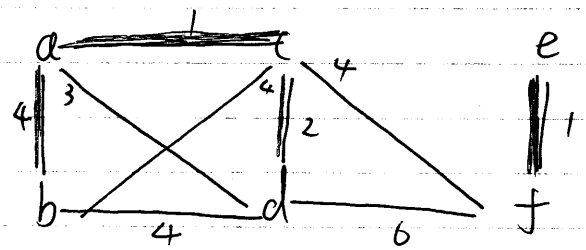
sort — for each edge — find operation

$O(|E| \cdot \log|V|)$ ← Kruskal's time

2017.3.23 Lecture 15 5.1 Minimum Spinning tree.

kruskal's algorithm

maintain a "forest": Pick set of edge in increasing order of weights
current "tree" can be disconnected, but there are no cycles



a). Sort:

ac	ef	cd	ad	ab	bc	bd	cf	df
1	1	2	3	4	4	4	4	6
✓	✓	✓	✗	✓	✗	✗	✓	

b). ~~keeping~~ an adding ~~order~~ edges in sorted order

1). check for cycle each time in the forest] DFS on the forest

2). stop where we have a single tree & all vertices have been checked

step a: sorting $O(|E| \log |E|) = O(|E| \log |V|)$ $|E| = O(|V|^2)$ $\log |E| = \log |V|^2 = 2 \log |V|$

step b(1): $O(|E|)$ ← outer for loop

$O(|E| \cdot |V|)$ { → detect cycle for each edge in the forest

DFS: $O(|V| + |E'|) = O(|V| + |V|) = O(|V|)$

final MST has exactly $|V|-1$ edge & $|V|$ ve

Cost: $|E| \log |V| + |E| \cdot |V|$

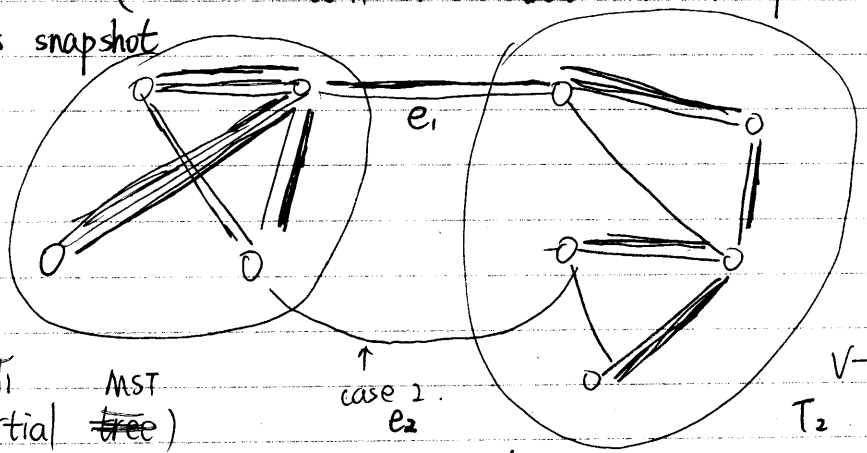
correctness: cut property (given a cut that respects the forest, adding the least edge that cross the cut, always lead to som MST

2018. 3. 23. Lecture 15 5.1 Minimum spanning tree.

proof: Given G, and given a partial MST, T

cut-property:
- select a cut that ~~respects~~ T
- pick the smallest weighted e and add to T
  $T' = T \cup \{e\}$
- we ~~sh~~ will show that T' is part of some MST

Prim's snapshot



Assume $w(e_1) \leq w(e_2)$

Case 1: there is only one edge cross the ~~tree~~ cut
e_1 is the only one, the smallest as well, e_1 has to be select

Case 2:
Two trees connected by e_2, final MST
If there are ~~two~~ more edges crossing the ~~tree~~ cut
show that selecting the least cost
cost of ~~final~~ final T: $cost(T_1) + cost(T_2) + w(e_2)$

$\exists$ T' that includes e_1
$T' = T_1 \cup \{e_1\} \cup T_2$
$cost(T') = cost(T_1) + cost(T_2) + w(e_1)$
$cost(T') \leq cost(T)$ because $w(e_1) \leq w(e_2)$
But T is a MST, So it least cost
$cost(T') = cost(T)$

2018.3.23 Lecture 15 5.1 Minimum spanning tree

```
while PQ is not empty
    u = deletemin(PQ)
    for all neighbors x of u
        if cost(x) > ~~cost+w~~ cost(u,x)
            cost(x) = w(u,x)
            decreasekey(x, cost(x))
```

O(|V|) for deletemin | binary tree PQ
O(|E|) decrease key

O(log|V|) cost

∴ Total cost: O((|V|+|E|) log |V|)

PQ:
A - 0
~~*~~ - ∞
↓
B - 4
~~C~~
D - ~~3~~ 2
~~*~~ - ∞
↓
B-4
D-2
F-4
*-∞

Why does greedy work for MST

~~cut - property~~:

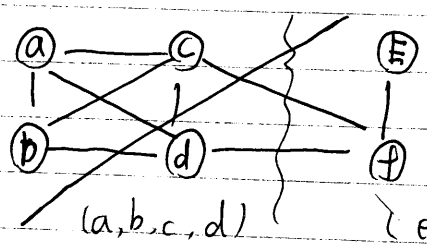
If we have a cut, then we can always choose the smallest edge that cross the cut.

G: graph

$S \subseteq V$: subset of vertics

(S, V-S) is a ~~set~~ cut (or 2-way partition) of G

S ↑ subset $\bar{S}$ ↑ complement V-S

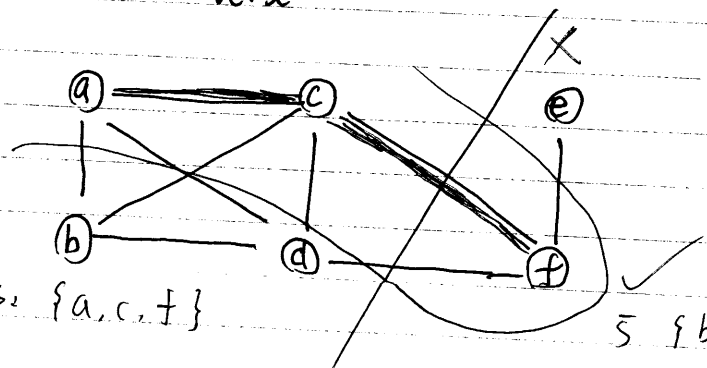


cut (S, V-S)

~~repeats~~ T = (V, E')

respect / iff no edge in E' spans or cross from S to V-S or vice-verse

不要把 MST 切断.



2018.3.23 Lecture 15 5.1 Minimum Spanning tree

MST: Minimum spanning tree

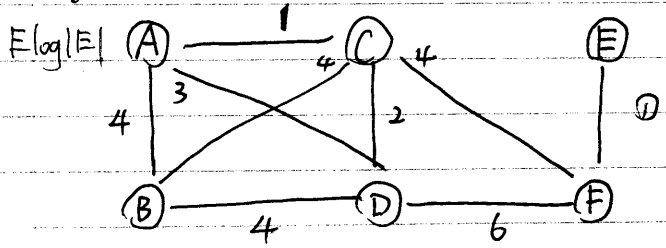
Given $G = (V, E)$ with weights

select a min cost tree, $T = (V, E')$ $E' \leq E$.

$cost(T) = \sum_{e \in E'} w(e)$

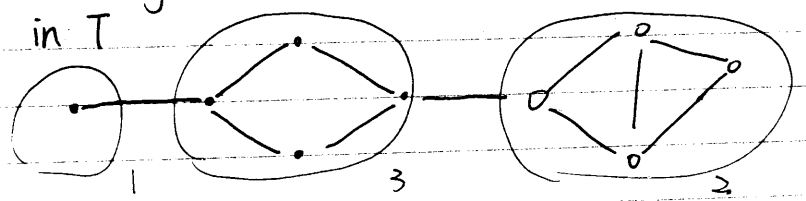
prim's Algorithm

1). Sort edges of G in increasing order of weight



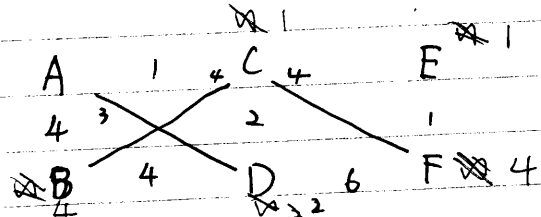
AC	EF	CD	AD	AB	BD	CF	DF
1	1	2	3	4	4	4	6
			↑				
			create cycle				

2). Add edges in sorted order if exactly one end point is in T



Prim's: invariant:

The correct partial MST is always connected



pick a start vertex u_0

$cost(u_0) = 0$, cost of all other vertex = ∞

Insert all vertices into a PQ

4.

a). Sort topologically $v_1, v_2 \cdots v_n$

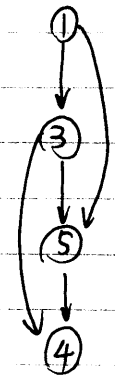
for i is n down to 1

$r(v_i) = w(v_i)$

for $(v_i, u) \in E$:

if $r(u) > r(v_i)$

$r(v_i) = r(u)$



b). Turn G into DAG of SCCs

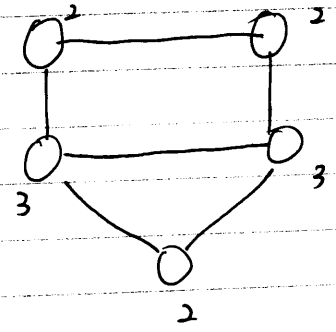
hw 5 Review

2/

(b). matrix → $O(|V|^2)$

For all vertices

3. i) DFS / BFS on G to find all degree → $O(|E|+|V|)$



ii) For $v \in V$

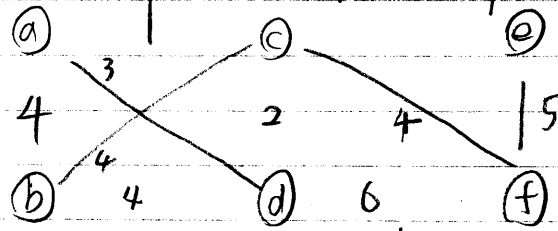
for u adjacent to v:

add 1 to v's node

$\Rightarrow O(|V|+|E|)$

Lecture 14

2018.3.20 Chapter 5.1 Minimum spanning trees

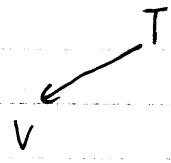


G is undirected

1). ac, cd, ad, ab, bc, bd, cf, ef, df

1 2 3 4 4 4 4 5 6

2).



{a, e}

{a, c, d}

{a, c, d, b}

{a, c, d, b, f}

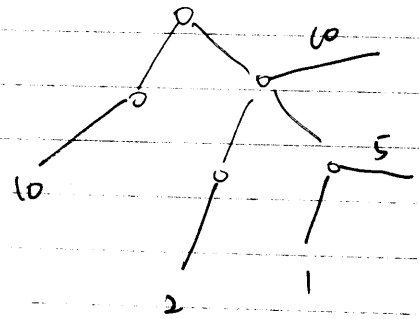
{a, c, d, b, f, e}

cost (T) = 16

1). sort the edge

2). T = { smallest edge }

Iteratively add the least weight edge wi one end point not in tree



Greedy: add the least cost edge

Chapter 5 Greedy Algorithm

2018.3.20

5.1 Minimum spanning tree.

1 Greedy Alogrithm

↘ "local" information

→ pick the best available option

Build up solution piece by piece, always choose next piece that offers most obvious and immedi benefit

Minimum ~~tree~~ spanning tree

Iput $G = (V, E)$, with weights

Output: 1) Tree

2). spanning tree

span all the vertex

$T = (V, E')$

$T \subseteq G$

↑

subgraph

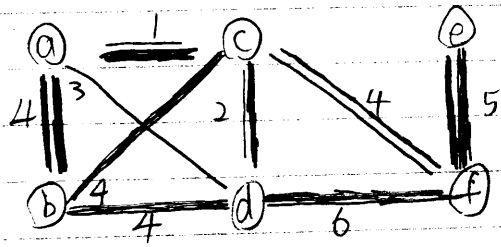
spanning tree: set of vertices in T includes all verties in G

$E' \subseteq E$

3). cost of $T = (V, E')$

$cost(T) = \sum_{e \in E'} w(e)$

find a tree T with smallest positive cost (T)



T_1 is a spanning tree

$cost(T) = 23$

T_2 is a spanning tree

$cost(T_2) = 16$

2. Prim's Algorithm (similar to Dijkstra's)

1) sort the edge in increasing order $|E| \log(|E|)$

2). $T = \{$ 1st edge in tree $\}$

for each edge in sorted order ← $|E|$

add if and only if one end point is in T

add edge to T if one end point is in T & we do no create a cycle $O(1)$ time

$O(E \log(|E|) + |E|)$

2018.3.20 Lecture 14 4.6 Shortest Paths in the presence of negative edges

between (x) any each (y)

source

choose min cost path exponential time method

how to detect cycle (negative)

run bellmon Ford one more iteration, if we still find smaller path, so we can proof it negative cycle.

$O(|V| \cdot |E|)$

Lecture 14

2018.3.20

4.6 Shortest paths in presence of negative edges

3. Bellman - Ford methods.

update the weights of all vertices in ends iteration

$\forall$ vertice $u \in V$

for all neighbours y of u:

update weight:

$$cost(u) + w(u, y) \leq cost(y)$$

$$cost(y) = cost(u) + w(u, y)$$

$X \rightarrow O \rightarrow O \rightarrow \cdots \rightarrow O$

max # of edges in any path from $X \leq |V| - 1$

exactly $|V| - 1$ iteration

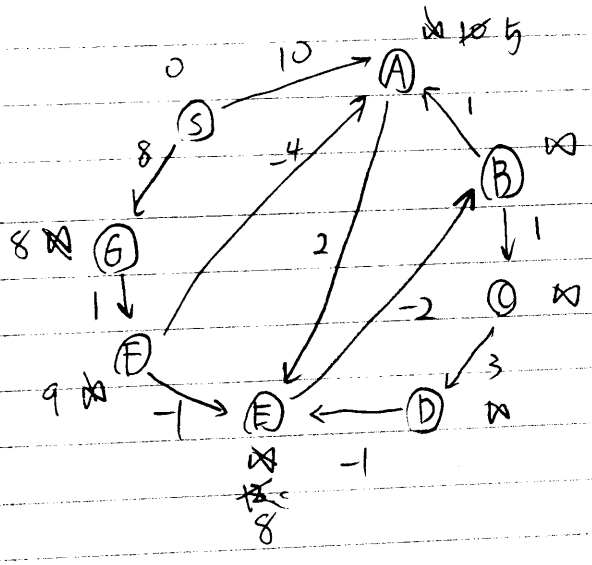
In each iteration, we update all of cost

total cost: $(|V| - 1) \times (|V| + |E|)$

$= O(|V|^2 + |V| \cdot |E|)$

$O(|V| \cdot |E|) \rightarrow$ sparse $O(|V|^2)$

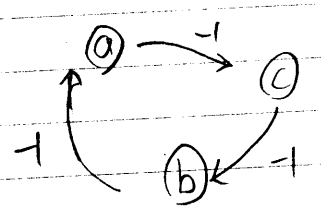
$\searrow$ dense $O(|V|^3)$



Iteration:

1

2



1) disallowed negative cycles

2) shortest cost is ill-defined

how to detect this

If we restrict to "paths" Bellman Ford will not work

1) find all possible paths