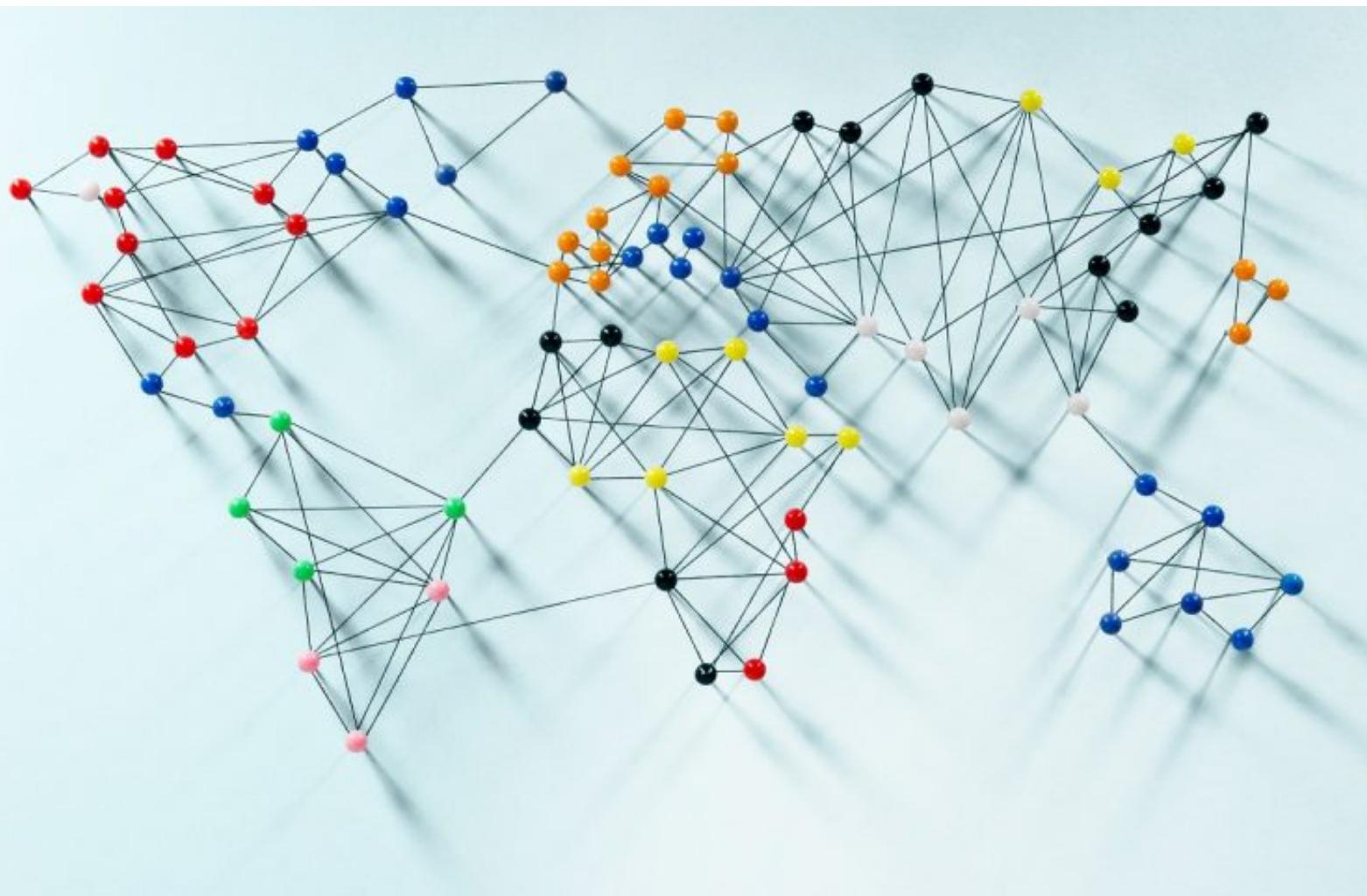


# Graph Theory and Algorithms

An Introduction to the world of Graphs



**Devansh Jain,**  
Indian Institute of Technology, Bombay

## Contents

1 Preface

ii

## 1 Preface

Preface

### About the project

adedsd

### About the report

sdsd

### References and Acknowledgement

sdfsd

(23)

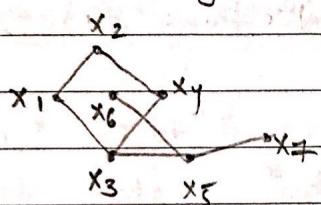
CLASSMATE

Date \_\_\_\_\_

Page \_\_\_\_\_

L-6 Graphs are <sup>incredibly</sup> very useful structures in computer science. They come up in all sorts of applications, scheduling, optimization, communications, the design and analysis of algorithms.

⇒ Informally, a graph is just a bunch of dots and lines connecting the dots



⇒ A graph  $G$  is a pair of sets  $(V, E)$  where

- $V$  is a non-empty set of items called vertices or nodes
- $E$  is a set of 2-item subsets of  $V$  called edges (can be empty set).

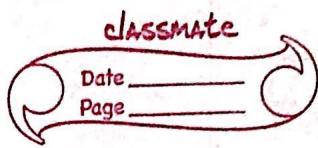
$$V = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$$

$$E = \{ (x_1, x_2), (x_2, x_4), (x_4, x_3), (x_3, x_1), (x_5, x_6), (x_3, x_5), (x_5, x_7) \}$$

equivalent

to  $x_1 - x_2$

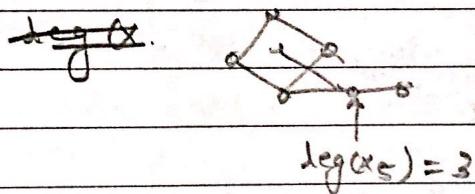
24



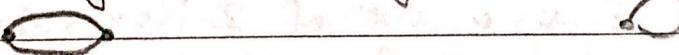
$\Rightarrow$  Two nodes  $x_i$  &  $x_j$  are adjacent if they're connected by an edge i.e.  $x_i - x_j \in E$

$\Rightarrow$  An edge  $e = (x_i, x_j)$  is said to be incident to its end-points  $x_i$  &  $x_j$ .

$\Rightarrow$  The number of edges incident to a node is called the degree of the node.



$\Rightarrow$  Graph is simple if it has no loops or multiples edges (multi-edge).



$|V| \rightarrow$  cardinality notation  
i.e. no. of elements in set V.

Men-Women partners ratio

25

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

6.041

6.002

6.042

6.003

6.034

Slots

5-7

7-9

9-11

11-1

1-3

### Graph-coloring problem

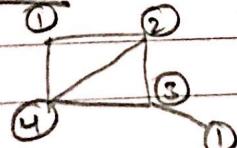
Given a graph  $G$  and  $K$  colors, assign a color to each node so adjacent nodes get different colors.

Minimum value of  $K$  for which such a coloring exists is the Chromatic number of  $G$   
 $\chi(G)$

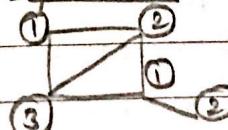
→ Color  $\Rightarrow$  Slots

Nodes  $\Rightarrow$  Courses

Option 1



Option 2



$$\chi(G) = 3$$

2 colors can't work because of triangle formation

(26)

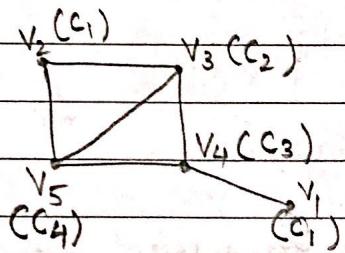
classmate

Date \_\_\_\_\_

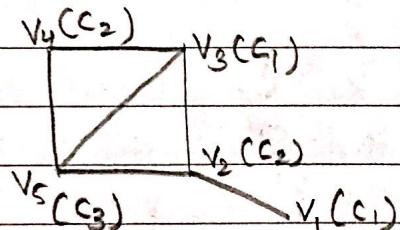
Page \_\_\_\_\_

Basic graph coloring algorithm

1. Order the nodes  $v_1, \dots, v_n$
2. Order the colors  $c_1, \dots$
3. For  $i = 1, 2, \dots, n$
4. Assign the lowest legal color to  $v_i$



We know this is not best case.



So, if you change the ordering, you may get a better answer.

Now this algorithm is an example of what's known as a greedy algorithm.

You just go one step after the next, taking the best you can do at each step. You never go back and try to make things better.

(27)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Thm.

If every node in  $G$  has degree  $d$ , then this basic algorithm uses atmost  $d+1$  colors for  $G$ .

Proof

By induction,

Induction hypothesis:

//Never induction on  $d$ , induction on  $n$  or  $e$   
↑  
node

Predicate

↓ "If every node in  $n$ -node graph  $G$  with max. degree  $d$   
 $P(n) =$  then this basic algorithm uses atmost  $d+1$  colors  
 for  $G$ ."

Base case:

 $n=1 \Rightarrow |E|=0 \Rightarrow d=0 \Rightarrow$  one color∴  $P(1)$  is trueInd. step: Assume  $P(n)$  is trueLet  $G$  be any  $n+1$  node graph.Let  $d$  be the max. degree of nodes in  $G$ Order the nodes:  $v_1, \dots, v_n, v_{n+1}$ Let  $G'$  be the graph after removing  $v_{n+1}$   
max. degree of nodes in  $G' \leq d$ ∴ Basic algorithm uses atmost  $d+1$  colors  
for  $G'$ .

(28)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

In  $G$ , color nodes  $v_1, \dots, v_n$  same as  $G'$ .

$v_{n+1}$  has atmost  $d$  neighbours therefore  
atleast one color is not present in  
 $v_{n+1}$ 's neighbours.

∴ Give  $v_{n+1}$  that color.

∴  $P(n+1)$  is true

$K_n \rightarrow$  n-node complete graph (or clique)



$$\text{Degree} = n-1$$

$$\chi(K_n) = n$$

(n)

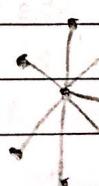


$$d = n/2$$

$$\chi(G) = 2$$

Bipartite graph

(n) star graph



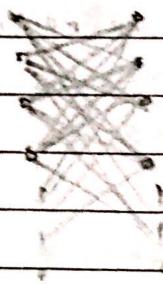
$$d = n-1$$

$$\chi(G) = 2$$

The Algorithm may be much better than the theorem

(29)

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Good ordering - (2 colors)

(C <sub>1</sub> ) V <sub>1</sub>	•	V <sub>5</sub> (C <sub>2</sub> )
(C <sub>1</sub> ) V <sub>2</sub>	•	V <sub>6</sub> (C <sub>2</sub> )
(C <sub>1</sub> ) V <sub>3</sub>	•	V <sub>7</sub> (C <sub>2</sub> )
(C <sub>1</sub> ) V <sub>4</sub>	•	V <sub>8</sub> (C <sub>2</sub> )

Bad ordering ( $\lceil \frac{n}{2} \rceil$  colors)

(C <sub>1</sub> ) V <sub>1</sub>	•	V <sub>3</sub> (C <sub>2</sub> )
(C <sub>2</sub> ) V <sub>3</sub>	•	V <sub>4</sub> (C <sub>2</sub> )
(C <sub>3</sub> ) V <sub>5</sub>	•	V <sub>6</sub> (C <sub>2</sub> )
(C <sub>4</sub> ) V <sub>7</sub>	•	V <sub>8</sub> (C <sub>5</sub> )

A graph  $G = (V, E)$  is said to be bipartite if  $V$  can be split into  $V_L, V_R$  so that all the edges connect  $V_L$  to  $V_R$ .

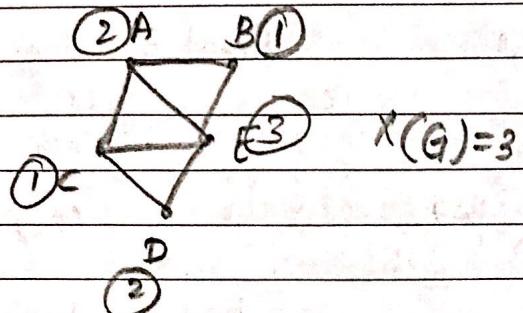
resource allocation problems (load balancing traffic on internet)

CLASSMATE

(30)

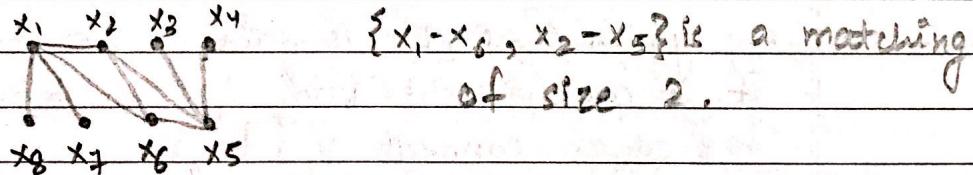
Date \_\_\_\_\_

Page \_\_\_\_\_



Matching algorithm - usage: online dating agencies, assignments problems (matching interns to hospitals on match day)

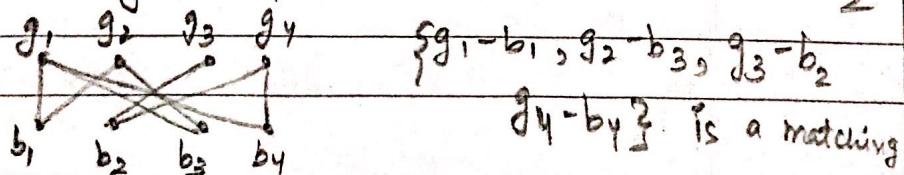
L-7  $\Rightarrow$  Given a graph  $G = (V, E)$ , a matching is a subgraph of  $G$  where every node has degree 1.



$\{x_1-x_7, x_2-x_6, x_3-x_5\}$  is a matching of size 3.

Size 4 matching is not possible in this graph.

$\Rightarrow$  A matching is perfect if it has size  $\frac{|V|}{2}$



(31)

classmate

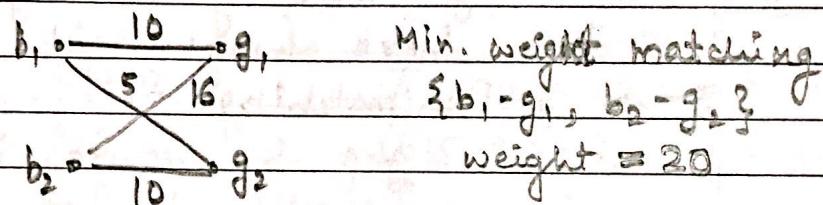
Date \_\_\_\_\_

Page \_\_\_\_\_

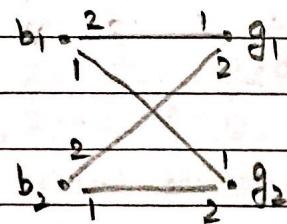
Weights?

$\Rightarrow$  The weight of a matching ( $M$ ) is the sum of the weights on the edge of  $M$ .

$\Rightarrow$  A minimum weight matching for graph  $G$  is a perfect matching for  $G$  with the minimum weight.



Preference list - similar to weights



If  $b_1 - g_1, b_2 - g_2$ , then  $b_1 - g_2$  will be rogue couple.

Given a matching  $M$ ,  $x \& y$  form rogue couple if they prefer each other to their mates in  $M$ .

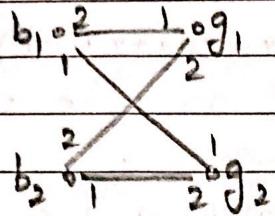
A matching is stable if there aren't any rogue couple.

(32)

classmate

Date \_\_\_\_\_

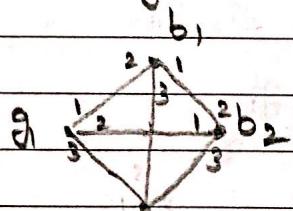
Page \_\_\_\_\_



$b_1-g_2$  &  $b_2-g_1$  is stable matching

If  $b-g$  matching is only allowed then there always exist a stable perfect matching.

If  $b-b$  &  $g-g$  is allowed, it may not always possible to find stable perfect matching.



$g_2$  (Preference doesn't matter)

Theorem: There is no stable perfect matching

Proof: By contradiction,

If there is a stable matching,  $g_2$  would be paired to someone without loss of generality (by symmetry) (the triangle is symmetric)

$\{g_2-b_2, g_1-b_1\} \Rightarrow b_1, b_2$  form rogue couple.

(33)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

1962

Stable Marriage Algorithm (<sup>David Lloyd</sup> Gale-Shapley Algo)

- N boys & N girls

- each boy has his own ranked list of all the girls. Same for each girl

- to find stable perfect matching.

Eg:- 5 boys & 5 girls

1 ♂	CBEAD	A 35214
2 ♂	ABECD	B 52143
3 ♂	DCBAE	C 43512
4 ♂	ACDBE	D 12345
5 ♂	ABDEC	E 23415

Using greedy algorithm,

1-C, 2-A, 3-D, 4-B, 5-E

Rogue Couple : 4-C

Using mating algorithm,  
Serendaders

Girls	Day 1	Day 2	Day 3	Day 4	
1 ABEAD	A 2 4(5)	5	5	5	A-5
2 ABEDC	B	2	21	2	B-2
3 DCBAE	C 1	14	4	4	C-4
4 ACDBE	D 3	3	3	3	D-3
5 ABDEC	E			1	E-1

34

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Need to show:

- ① Algorithm terminates (Matching returned)
- ② Everyone gets matched (Perfect matching)
- ③ No rogue couple (Stable matching)
- ④ It runs quickly
- ⑤ Fairness

Theorem 1 Algo terminates in less than  $N^2+1$  days

Proof By contradiction.

Suppose, algo doesn't terminate in  $N^2+1$  days.

Claim: If we don't terminate on a day,  
then atleast one girl had more than  
one boy and so she rejected atleast  
one boy who crosses her name.

If the algo doesn't terminate in  $N^2+1$  days  
then atleast  $N^2+1$  cross-outs have happened.

As there are  $N^2$  names on  $N$  lists, there  
are atmost  $N^2$  cross-outs. ( $< N^2+1$ )

Thus, not possible.

(35)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Predicate  $P = \text{"If a girl } G \text{ ever rejected a boy } B, \text{ then } G \text{ has a suitor who she prefers to } B."$

Lemma 1 :  $P$  is an invariant for the algo

Proof: Base case : By induction on number of days.

Base case: Day 0 (Vacuously true)

(No one rejected yet)

Ind step:  $P$  holds at the end of Day  $d$ .

Case 1  $G$  rejects  $B$  on day  $d+1$ .

Then there was a better boy

$\Rightarrow P$  holds on Day  $d+1$

Case 2  $G$  rejected  $B$  before day  $d+1$

$P$  on day  $d \Rightarrow G$  had a better boy on day  $d$ .

$\Rightarrow P$  holds on Day  $d+1$ .

Theorem 2 Everyone is married in Algo

Proof By contradiction

Assume that some boy  $B$  is not married,

then he was rejected by every girl which

$\Rightarrow$  every girl by Lemma 1 has a better suitor

$\Rightarrow$  every girl is married  $\Rightarrow$  every boy is married

$\Rightarrow B$  is married.

(36)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Theorem 3 Algo produces a stable matching (No rogue couple)

Proof

let B & G be any pair that are not married.

To prove: B & G aren't rogue.

Case 1 G rejected B

⇒ G has a better suitor than B (Lemma 1)

⇒ G married someone whom she prefers over B.

⇒ G is not rogue with B.

Case 2 G didn't reject B

⇒ B never serenaded G

⇒ G is lower in order than B's spouse.

⇒ B is not rogue with G

∴ No rogue couples.

37

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Let  $S$  be the set of all stable matching.  
 $S \neq \emptyset$  as algo produces one stable matching.

For each person  $P$ , we define the realm of possibility for  $P$  to be  $\{Q \mid \exists M \in S \mid \{P, Q\} \in M\}$

A person's optimal mate is his/her favorite from the realm of possibility.

A person's pessimal mate is his/her least favorite from the realm of possibility.

Theorem 4 Algo marries every boy with his optimal mate.

Proof —

Theorem 5 Algo marries every girl with her pessimal mate.

Proof —

TMA  $\rightarrow$  The Marriage Algorithm

38

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

L-8

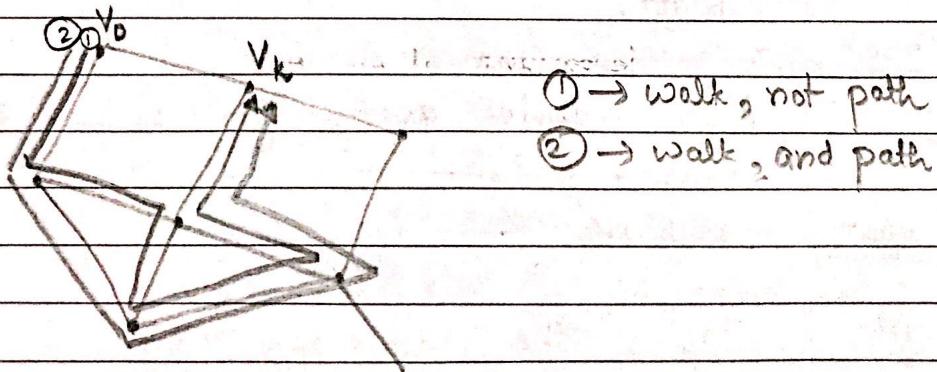
- Walks & Paths
- Connectivity
- Cycles & Closed walk
- Spanning Tree (ST)
- Min-weight spanning tree (MST)

### Walks & Paths

⇒ A walk is a sequence of vertices that are connected by edges

Eg:-  $v_0 - v_1 - \dots - v_k$   
Start                                                          end

It has  $k$ -edges  $\Rightarrow$  length =  $k$



⇒ A path is a walk where all  $v$ 's are distinct

(39)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Lemma 1 If I walk from  $u$  to  $v$ , then I path from  $u$  to  $v$ .

Proof I walk  $u$  to  $v$ .

By well-ordering principle: Walk of minimal length.

$$u = v_0 - v_1 - \dots - v_k = v$$

We prove that this walk (of minimal length) is a path.

Case  $k=0$  No edge length = 0

Case  $k=1$  Just one edge length = 1

Case  $k \geq 2$

Suppose this walk is not a path

then  $i \neq j$   $v_i = v_j$

$$u = v_0 - \dots - v_i = v_j - \dots - v_k$$

Now this a shorter walk

thus contradicting our assumption.

(40)

classmate

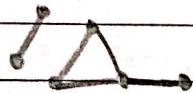
Date \_\_\_\_\_

Page \_\_\_\_\_

Connectivity

- $\Rightarrow$   $u$  and  $v$  are connected if there is a path from  $u$  to  $v$ .
- $\Rightarrow$  A graph  $G = (V, E)$  is connected if every pair of vertices  $(v_1, v_2) \in V \times V$  are connected.

Not connected graph



Connected graph

Cycles & closed walks

(also known as tours)

- $\Rightarrow$  A closed walk is a walk starts and ends at exactly same vertex.

$$v_0 - v_1 - \dots - v_k = v_0$$

a closed walk with and

- $\Rightarrow$  If  $k \geq 3$ , if all  $v_0, \dots, v_{k-1}$  are distinct then it is called a cycle

(41)

classmate

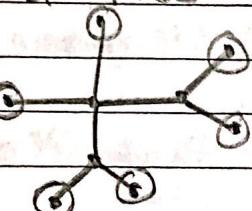
Date \_\_\_\_\_

Page \_\_\_\_\_

Trees

⇒ A connected and acyclic graph is called a tree.

⇒ A leaf is a node with degree = 1 in a tree.



Lemma 2 Any connected subgraph of a tree is a tree.

Proof By contradiction, if the connected subgraph is not a tree then it must have a cycle.

As the subgraph is a part of a graph, the graph must have a cycle.

But the graph is a tree.

We get a contradiction. ■

(42)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Lemma 3 A tree with  $n$  vertices has  $n-1$  edges.

Proof By Induction

P( $n$ ) Ind. Hypo. :  $P(n)$  = "There are  $n-1$  edges in an  $n$ -vertex tree"

Base case:  $P(1)$  is true. ①

Ind. step: Assume that  $P(n)$  is true

Let  $T$  be an  $(n+1)$ -vertex tree,

Let  $v$  be a leaf of the tree.

Delete  $v$ : this creates a connected subgraph  
(which is also a tree (Lemma))

By  $P(n)$ : this subgraph has  $n-1$  edges.

Re-attach  $v$ :  $T$  has  $(n-1)+1=n$  edges  
(degree of  $v=1$ )

$P(n+1)$  is true



43

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

## Spanning Trees

→ A spanning tree (ST) of a connected graph  $G = (V, E)$  is a subgraph that is a tree with same vertices as the graph.

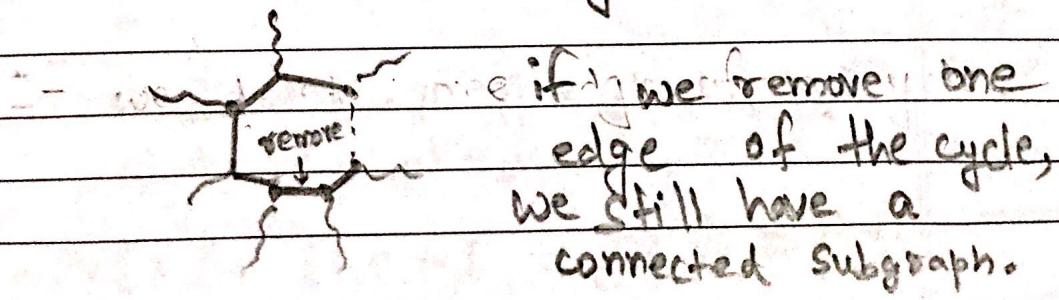
Theorem Every connected graph has a ST.

Proof By contradiction.

Assume a connected graph  $G$  with no ST.

Let  $T$  be a connected subgraph of  $G$  and has same vertices as  $G$  and has minimum number of edges.

As  $T$  is not a spanning tree, it must have atleast one cycle.



(44)

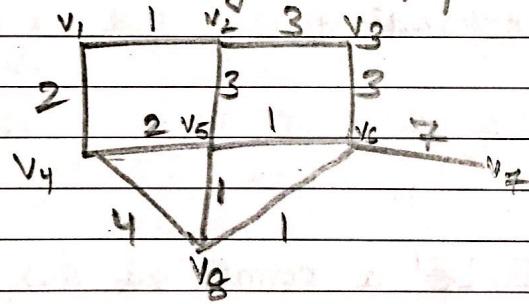
classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Subgraph we obtain is smaller than T  
(w.r.t. number of edges)

thus contradicting our minimum condition on T.

Minimum-weighted spanning tree



$$ST_1 = \{v_1-v_4, v_4-v_5, v_5-v_2, v_5-v_6, v_6-v_3, v_6-v_7, v_5-v_8\}$$

$$\text{Weight of } ST_1 = 2 + 2 + 3 + 1 + 3 + 7 + 1 = 19$$

$$ST_2 = \{v_1-v_2, v_1-v_4, v_4-v_5, v_5-v_6, v_6-v_3, v_6-v_7, v_5-v_8\}$$

$$\text{Weight of } ST_2 = 1 + 2 + 2 + 1 + 3 + 7 + 1 = 17$$

(45)

classmate

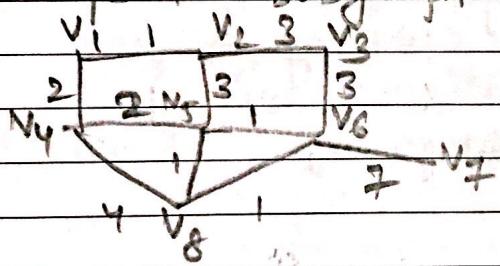
Date \_\_\_\_\_

Page \_\_\_\_\_

⇒ The minimum spanning tree (MST) of an edge-weighted graph is defined as the ST of  $G$  with the smallest possible sum of edge-weights.

Algo

- Grow a subgraph one edge at a time, such that at each step:
- Add the minimum weighted edge keeps the subgraph acyclic.



- + 1 ①  $v_6 - v_8$
- + 1 ②  $v_1 - v_2$
- + 1 ③  $v_5 - v_6 \Rightarrow v_5 - v_8 \text{ } \times$
- + 2 ④  $v_1 - v_4$
- + 2 ⑤  $v_4 - v_5 \Rightarrow v_4 - v_8 \text{ } \times, v_2 - v_5 \text{ } \textcircled{R}$
- + 3 ⑥  $v_2 - v_3 \Rightarrow v_3 - v_8 \text{ } \times$
- + 7 ⑦  $v_6 - v_7$

46

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Theorem For any connected weighted graph  $G$ , the algo produces a MST.

Lemma Let  $S$  consists of first  $k$ -edges, by the algo, then  $\exists$  MST  $T = (V, E)$  for  $G$  such that  $S \subseteq E$

Proof

(Assumed lemma is true)

Graph  $G = (V, E)$  $|V| = n$ 

1) Suppose  $< n-1$  edges are picked, then  $\exists$  edge in  $E - S$  that can be added without creating a cycle.

2) Once  $n-1$  edges are <sup>picked</sup>, we know  $S$  is an MST.

Proof

By Induction:

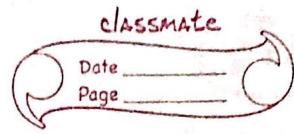
$P(m) = \# G \# S$  consisting of first  $m$  selected edges  $\exists$  MST of  $G$  such that  $S \subseteq E$ .  
 $T = (V, E)$

Base Case:  $m = 0 \Rightarrow S = \emptyset \subseteq E$  for any MST.  
 $T = (V, E)$

Assume

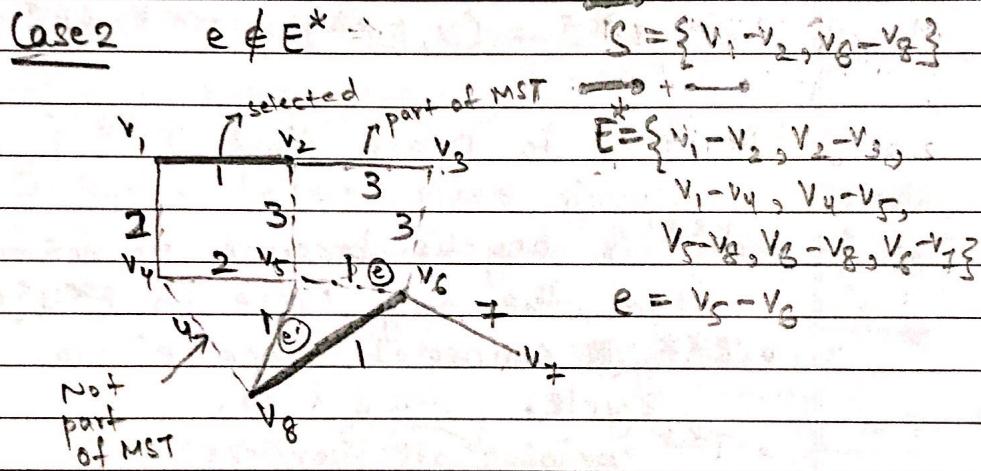
Ind. Step:  $P(m)$  holdslet  $e$  denote the  $(m+1)$ th selected edge.Let  $S$  denote the first  $m$  selected edges.

47



Let  $T^* = (V, E^*)$  be MST of  $G$  such that  
 $S \subseteq E^*$

Case 1:  $e \in E^* \Rightarrow S \cup \{e\} \subseteq E^* \Rightarrow P(m+1)$  holds



From the algo,

$\{S \cup \{e\}\}$  has no cycle

(This is how we choose  $(m+1)^{\text{th}}$  edge)

$T^*$  is a tree  $\Rightarrow (V, E^* \cup \{e\})$  must contain  
 $= (V, E^*)$  a cycle. (Proof in book)

→ this cycle has edge  $e' \in E^* - S$

(48)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Algo could have selected  $e$  or  $e'$   
 $\Rightarrow$  weight of  $e \leq$  weight of  $e'$ .

Swap  $e$  and  $e'$  in  $T^*$ :

$$\text{Let } T^{**} = (V, E^{**}) : E^{**} = (E^* - \{e'\}) \cup \{e\}$$

We need to prove that  $T^{**}$  is an MST.

- $T^{**}$  is acyclic because we removed  $e'$  from the only cycle in  $E^* \cup \{e\}$ .
- $T^{**}$  is connected since  $e'$  was on a cycle.
- $T^{**}$  contains all vertices of  $G$

$\Rightarrow T^{**}$  is ST of  $G$ .

$$\begin{aligned} \text{weight of } T^{**} &= \text{weighted sum of } E^{**} \\ &= \text{Weighted sum of } E^* - \text{weight of } e' \\ &\quad + \text{weight of } e \\ &= \text{Weight } \cancel{\text{sum}} \text{ of } T^* - \text{weight of } e' \\ &\quad + \text{weight of } e \end{aligned}$$

As weight of  $e \leq$  weight of  $e'$

Weight of  $T^{**} \leq$  weight of  $T^*$

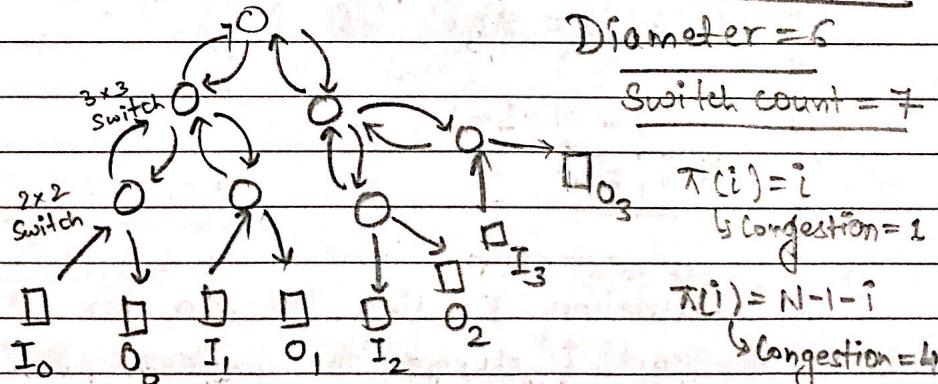
And as  $T^*$  is MST, equality holds &  $T^{**}$  is also MST.  $\square$

49

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_L-9L-10Communication Networks

- great application of graph theory
- how do you route packets through networks

 $N \times N$  Networks ( $N = 2^k$ )Complete Binary Tree $\circ$  = switch (direct packets through a network) $\square$  = terminal (source and destination of data)4x4 Network $N \times N$  NetworkDiameter =  $2(1 + \log_2 N)$ Max. Switch size =  $3 \times 3$ Switch count =  $2N - 1$ Max Congestion =  $N$

(50)

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

**Latency:** Time that is required for a packet to travel from an input to an output.

**Diameter:** Length of the shortest path of a network b/w Input and Output that are farthest apart  
(Maximum length of shortest path)

A permutation is a function  $\pi: \{0, \dots, n-1\} \rightarrow \{0, \dots, n-1\}$  such that no two numbers are mapped to the same value  
( $\pi(i) = \pi(j) \text{ iff } i=j$ )

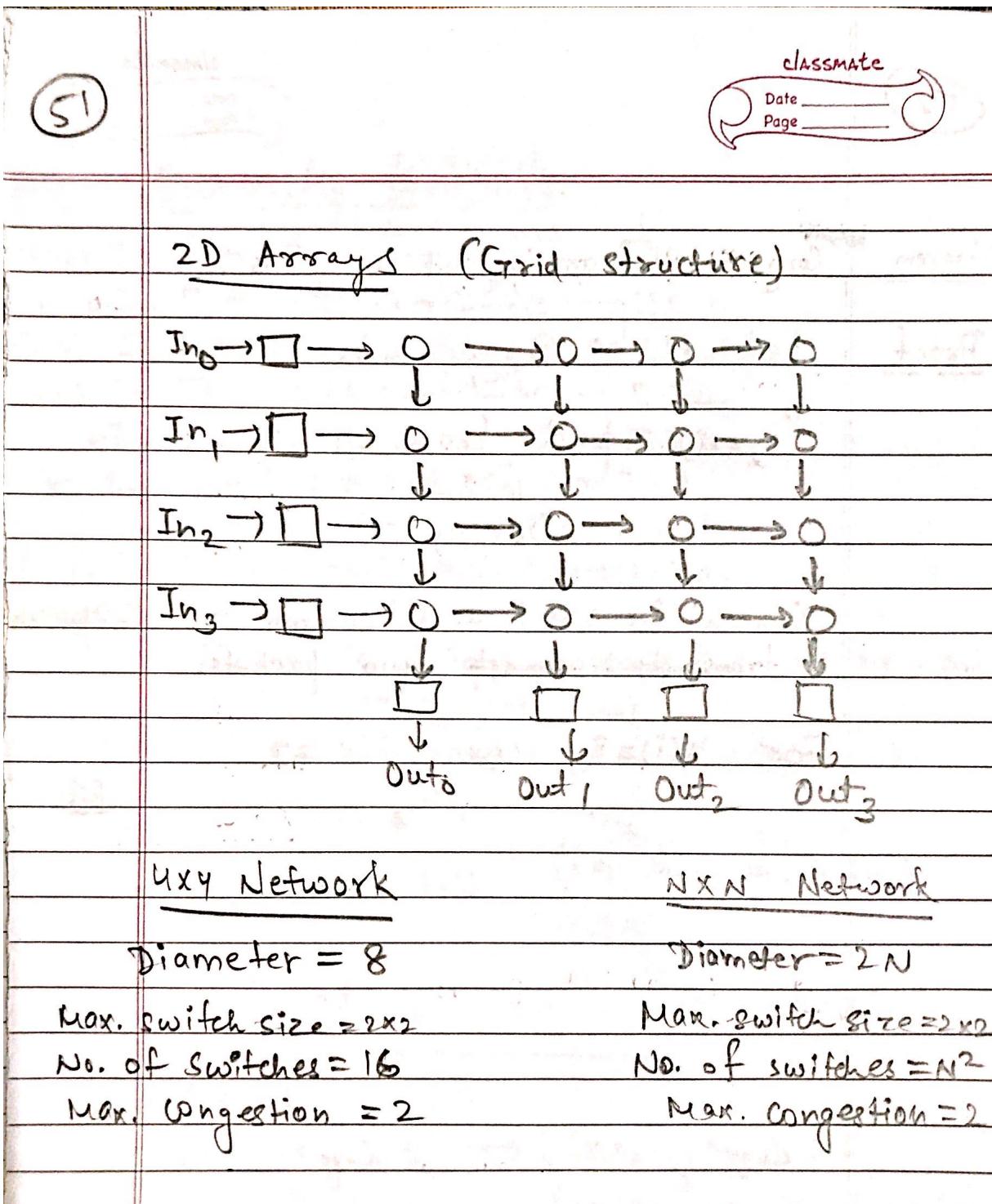
$$\text{Eg: } \begin{aligned} \pi(i) &= n-1-i \\ \pi(i) &= i \end{aligned}$$

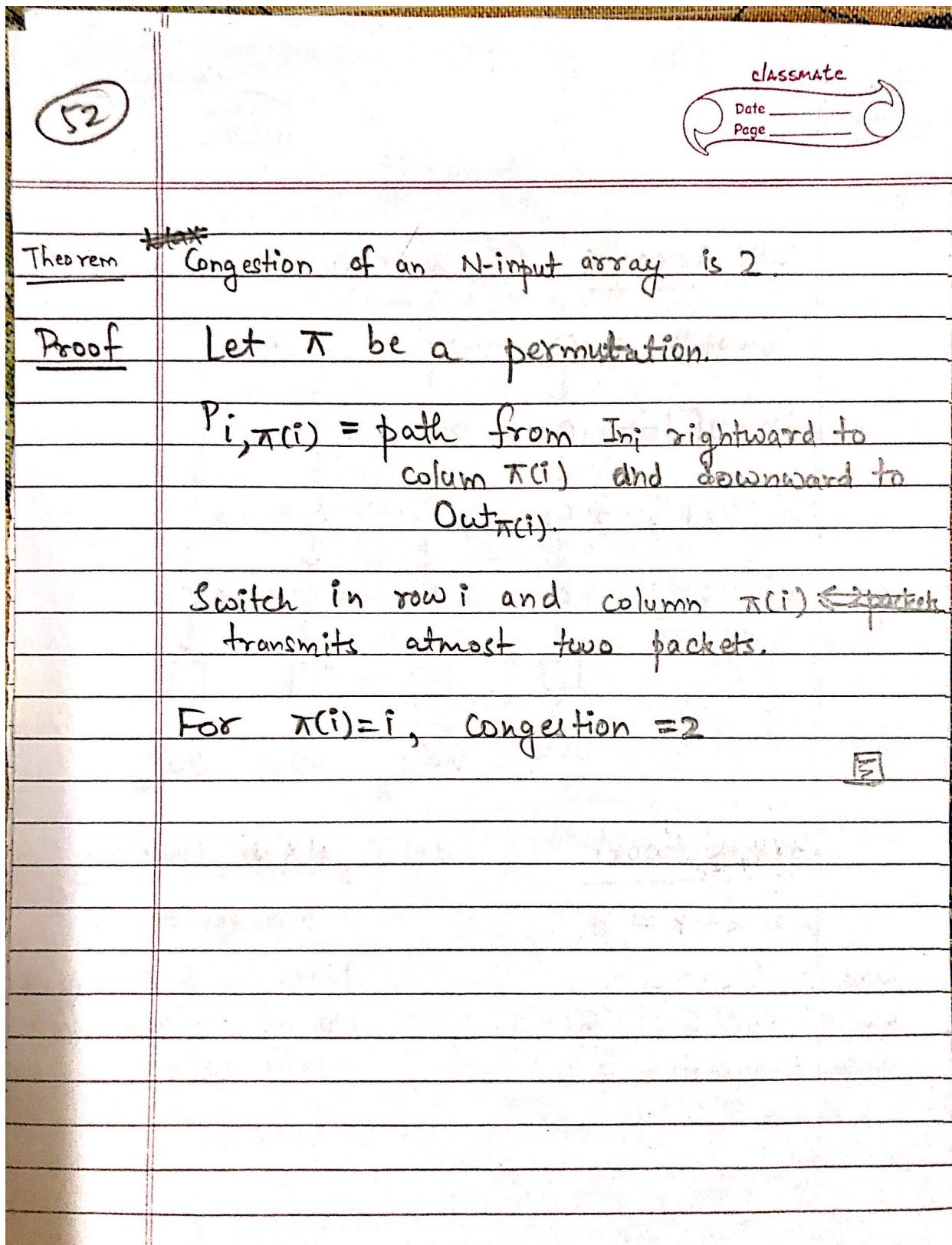
Permutation Routing Problem for  $\pi$

- For each  $i$  direct the packet at  $I_i$  to  $O_{\pi(i)}$   
path taken is denoted by  $P_{i, \pi(i)}$

The congestion of paths  $P_{0, \pi(0)}, \dots, P_{n-1, \pi(n-1)}$  is equal to the largest number of paths that pass through a single switch.

Max congestion = max possible value of congestion. (over permutations with best path)





53

CLASSMATE  
 Date \_\_\_\_\_  
 Page \_\_\_\_\_

### Butterfly Network Structure

Level 0      Level 1      Level 2      Level 3

(000) In<sub>0</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>0</sub>

(001) In<sub>1</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>1</sub>

(010) In<sub>2</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>2</sub>

(011) In<sub>3</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>3</sub>

(100) In<sub>4</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>4</sub>

(101) In<sub>5</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>5</sub>

(110) In<sub>6</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>6</sub>

(111) In<sub>7</sub>  $\square \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow \square$  Out<sub>7</sub>

---

Switch is uniquely identified by its row & col  
 binary rep. of input

$((b_1, \dots, b_{\log N}), l)$

$((b_1, \dots, \overline{b_{l+1}}, \dots, b_{\log N}), l+1)$        $((b_1, \dots, \overline{b_{l+1}}, \dots, b_{\log N}), l+1)$

↑  
 Complement  
 of  $(l+1)^{\text{th}}$  bit.

---

**Routing**

$$\begin{array}{c}
 ((x_1, \dots, x_{\log N}), 0) \rightarrow ((y_1, x_2, \dots, x_{\log N}), 1) \\
 \quad \downarrow \\
 \quad ((y_1, y_2, x_3, \dots, x_{\log N}), 2) \\
 \quad \downarrow \\
 ((y_1, \dots, y_{\log N}), \log N) \leftarrow ((y_1, \dots, \overline{y_l}, x_{l+1}, \dots, x_{\log N}), l)
 \end{array}$$

(54)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

In<sub>3</sub> → Out<sub>5</sub>

Eg:- 011 → 101

In<sub>3</sub> → (011, 0)

(111, 1)



(101, 2)

Out<sub>5</sub> ← (101, 3)N × N Network

$$\text{Diameter} = 2 + \log N$$

$$\text{Max. Switch size} = 2 \times 2$$

$$\text{No. of Switches} = N(1 + \log N)$$

$$\text{Congestion} = \sqrt{N} (N e^{2^n})$$

$$= \sqrt{N/2} (N e^{2^{n+1}})$$

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

Benes

1960s - Benes, a Bell Labs researcher

$N \times N$  Network

Diameter =  $1 + 2 \log N$

Max switch size =  $2 \times 2$

No of switches =  $2N \log N$

Congestion = 1

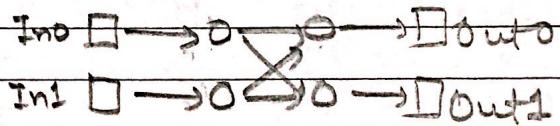
(56)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Theorem | Congestion of  $N$ -input Benes network  
is 1, when  $N = 2^a$  for some  $a \geq 1$

ProofBy induction on  $a$ , $P(a) =$  "This theorem is true for  $a$ ".Base case:  $a=1 N=2$ 

$$\pi(0) = 0$$

$$\pi(1) = 1$$

Congestion = 1  
(Just straight)

$$\pi(0) = 1$$

$$\pi(1) = 0$$

Congestion = 1  
(Just cross-over)

Ind. Step: Assume  $P(a)$  is true

$$\text{Example: } \pi(0) = 1 \quad \pi(4) = 3$$

$$\pi(1) = 5 \quad \pi(5) = 6$$

$$\pi(2) = 4 \quad \pi(6) = 0$$

$$\pi(3) = 7 \quad \pi(7) = 2$$

Constraint graph

If two packets must pass through different subnetworks then there is an edge b/w them.

(57)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-10

## Euler tours



## Directed graphs

- Definitions
- No. of walks
- Strong connectivity
- DAGs

## Tournament Graphs

Euler tour

- He lived in Konigsberg
- Seven bridges (birth of graph theory)

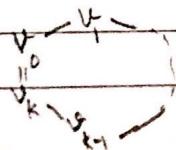
An Euler tour is a walk that traverses every edge exactly once, and starts and finishes at same vertex.

Theorem

A connected graph has an Euler tour iff every vertex has even degree.

Proof $(\Rightarrow)$ 

Assume  $G = (V, E)$  has an Euler tour



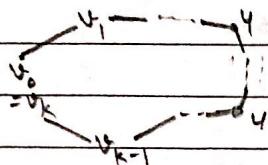
(58)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Since every edge in  $E$  is traversed once;  
 clearly  $\deg(u) = (\# \text{ times } u \text{ appears in this tour } v_0 \dots v_k) \times 2$



(≤)

For  $G = (V, E)$ , assume  $\deg(v)$  is even  $\forall v \in V$

Let  $N: v_0 - v_1 - \dots - v_k$  be the longest walk that traverses no edge more than once.

①  $v_k - u$  not in  $W \Rightarrow v_0 - v_1 - \dots - v_k - u$  is longer than  $W \Rightarrow$  Contradicts  
 $\therefore$  All the edges <sup>int</sup> incident of  $v_k$  are traversed in  $W$ .

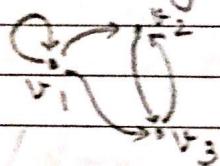
②  $v_k \neq v_0 \Rightarrow v_k$  has odd degree in walk  $W$   
 As we have showed that all edges <sup>int</sup> of  $v_k$  are in  $W$ , so  $\deg(v_k)$  is odd in  $G$   
 $\Rightarrow$  Contradicts as  $\deg(v)$  is even  $\forall v \in V$ .

(59)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Directed graph (digraphs)

$$\text{indegree}(v_2) = 2$$

$$\text{outdegree}(v_2) = 1$$

Theorem Let  $G = (V, E)$  be an  $n$ -node graph with  $V = \{v_1, \dots, v_n\}$ . Let  $A = \{a_{ij}\}$  denote the adjacency matrix for  $G$  that is

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \rightarrow v_j \text{ is an edge.} \\ 0 & \text{otherwise} \end{cases}$$

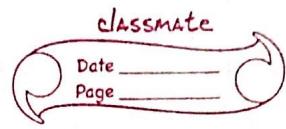
Let  $p_{ij}^{(k)} = \#$  directed walks of length  $k$  from  $v_i$  to  $v_j$ .

$$\text{then } A^k = \{p_{ij}^{(k)}\}$$

$$A = \begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & 1 & 1 & 1 \\ v_2 & 0 & 0 & 1 \\ v_3 & 0 & 1 & 0 \end{bmatrix} \quad A^k = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 1 & 3 & 3 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

(60)

Proof

$a_{ij}^{(k)}$  denote the  $(i, j)^{\text{th}}$  entry in  $A^k$

By induction:

Predicate  $\rightarrow P(k) = \text{"Theorem is true for } k\text{"}$   
 $= \text{"}\forall i, j \quad a_{ij}^{(k)} = p_{ij}^{(k)}\text{"}$

Base case  $k=1$

Edge  $v_i \rightarrow v_j : p_{ij}^{(1)} = 1 = a_{ij}^{(1)}$

No edge  $v_i \rightarrow v_j : p_{ij}^{(1)} = 0 = a_{ij}^{(1)}$

$\therefore \forall i, j \quad a_{ij}^{(1)} = p_{ij}^{(1)}$

Ind. step

Assume  $P(k)$  holds.

$$p_{ij}^{(k+1)} = \sum_{h: v_h \rightarrow v_j \text{ is an edge in } E} p_{ih}^{(k)} = \sum_{h=1}^n p_{ih}^{(k)} \cdot a_{hj}^{(k)}$$

$v_i \xrightarrow{k} v_n \xrightarrow{k} v_j$

Ind. step  $\rightarrow$

$$= \sum_{h=1}^n a_{ih}^{(k)} \cdot a_{hj}^{(k)}$$

$$= a_{ij}^{(k+1)} \Rightarrow P(k+1)$$

Matrix multiplication holds

(6)

classmate

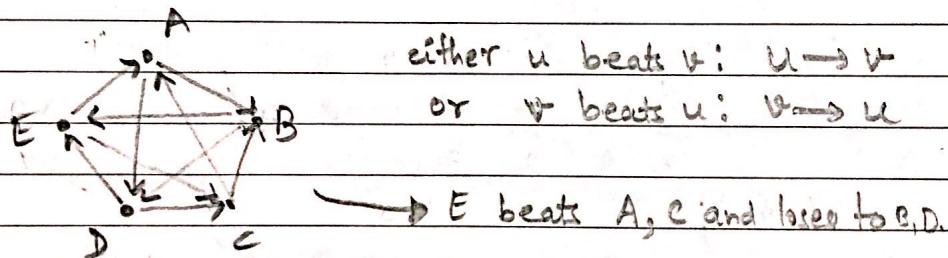
Date \_\_\_\_\_

Page \_\_\_\_\_

$\Rightarrow$  A digraph  $G = (V, E)$  is called strongly connected if  $\forall u, v \in V \exists$  directed path from  $u$  to  $v$  in  $G$ .

$\Rightarrow$  A digraph is called directed acyclic graph (DAG) if it does not contain any directed cycles.

### Tournament Graph



$\Rightarrow$  A directed Hamiltonian path is a directed walk that visits every vertex exactly once.

Theorem Every tournament graph contains a directed Hamiltonian path.

(62)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Proof By induction (on number of nodes)

$P(n)$  = "Every tournament graph on  $n$ -nodes contains a directed Ham-pair."

Base case :  $P(1)$  holds

Ind. step : Assume  $P(n)$  holds

Consider a tournament graph on  $(n+1)$ -nodes

Take out a node  $v$ . We are left with a tournament graph on  $n$ -nodes.

By  $P(n)$ ,  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$

Applying  $\text{Q} \rightarrow$  If  $v \rightarrow v_i$  is an edge, then  $P(n+1)$  holds.

(if  $v_i \rightarrow v$  and  $v \rightarrow v_n$ ) then

Claim:  $\exists i$  such that  $v_i \rightarrow v$  &  $v \rightarrow v_{i+1}$

Proof by contradiction

If no such  $i$  exists and we know

$v \rightarrow v \not\in$  then  $v_i \rightarrow v \nexists i \in \{1, 2, \dots, n\}$  (by induction)

But  $v \rightarrow v_n$  contradicts this.

There is also proof by strong induction.  
See it.

63

CLASSMATE  
Date \_\_\_\_\_  
Page \_\_\_\_\_

Approach

Case 1  $v \rightarrow v_i$

Case 2  $v_i \rightarrow v$

Smallest  $i$  such that  $v \rightarrow v_i$   
 $\Rightarrow v_k \rightarrow v$  &  $k < i$   
 $\Rightarrow v_{i-1} \rightarrow v$

$(v_1 \rightarrow \dots \rightarrow v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \dots \rightarrow v_n)$

directed  
A new Hamiltonian path  
is formed

$\therefore P(n+1)$  holds

(64)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

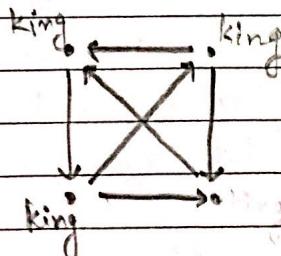
King Chicken Problem

either chicken  $u$  pecks chicken  $v$ :  $u \rightarrow v$   
 or chicken  $v$  pecks chicken  $u$ :  $v \rightarrow u$

$u$  "virtually" pecks  $v$  if

- $u \rightarrow v$ , or
- $\exists w \ u \rightarrow w \rightarrow v$

A chicken that virtually pecks every other chicken is called king chicken.

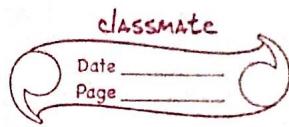


Theorem The chicken with highest outdegree is definitely a king.

Proof By contradiction,

Let  $u$  have highest outdegree.  
 Suppose  $u$  is not king

(65)



$\Rightarrow \exists v : v \rightarrow u$ , and  $\nexists w : u \rightarrow w \Rightarrow v \rightarrow w$

(Outdegree of  $v$ )  $\geq$  (Outdegree of  $u$ ) + 1

So, our assumption is false. ■

L-11

- Relations
  - Properties
  - Equivalent Relations
  - Partial Orders
    - Hasse diagram
    - Total Order
    - Topological Sort
  - Parallel Task Scheduling
    - Dilworth's Lemma

### Relations

$\rightarrow$  A relation from a set  $A$  to a set  $B$   
is a subset  $R \subseteq A \times B$

Ex :-  $R = \{(a, b) : \text{student } a \text{ is taking class } b\}$

$(a, b) \in R : a R b$ ,  $a \not R b$

$\nwarrow$  Relational symbol

(66)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

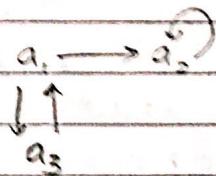
→ Relation on A is a subset  $R \subseteq A \times A$

Eg:-  $A = \mathbb{Z} : xRy \text{ iff } x \equiv y \pmod{5}$

$A = \mathbb{N} : xRy \text{ iff } x|y$

$A = \mathbb{N} : xRy \text{ iff } x \leq y$

Set A together with R is a directed graph  $G = (V, E)$  with  $V = A, E = R$ .



### Properties

A relation R on A is

\* reflexive if  $xRx \quad \forall x \in A$

\* symmetric if  $xRy \Rightarrow yRx \quad \forall x, y \in R$

\* anti-symmetric if  $xRy \wedge yRx \Rightarrow x=y$

\* transitive if  $xRy \wedge yRz \Rightarrow xRz$

(67)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Ex

Reflexive Symmetric Anti-Symmetric Transitive

 $x \equiv y \pmod{5}$  ✓ ✓ ✗ (2, 7) → Equivalence

 $x \mid y$  ✓ ✗ (1, 2) → Divisibility

 $x \leq y$  ✓ ✗ (1, 2) → Order

⇒ An equivalence relation is reflexive, symmetric, transitive.

Ex!  $A = \mathbb{Z} : x R y \text{ iff } x = y \in \mathbb{Z}$   
 $A = \mathbb{Z} : x R y \text{ iff } x \equiv y \pmod{5}$

⇒ The equivalence class of  $x \in A$  is the set of all elements in A related to  $x$  by  $R$ : denoted by  $[x]$   
 $[x] = \{y : x R y\}$

Ex:  $A = \mathbb{Z} : x R y \text{ iff } x \equiv y \pmod{5}$

$[7] = \{ \dots, -3, 2, 7, 12, 17, \dots \}$

$[2] = [7] = [12] = [17] = \dots$

(68)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

A partition of  $A$  is a collection of disjoint nonempty sets  $A_1, \dots, A_n \subseteq A$  whose union is  $A$ .

Ex:  $A = \mathbb{Z} : xRy \text{ iff } x \equiv y \pmod{5}$

$$A_0 = \{ \dots -5, 0, 5, \dots \}$$

$$A_1 = \{ \dots -4, 1, 6, \dots \}$$

$$A_2 = \{ \dots -3, 2, 7, \dots \}$$

$$A_3 = \{ \dots -2, 3, 8, \dots \}$$

$$A_4 = \{ \dots -1, 4, 9, \dots \}$$

Theorem The equivalence classes of equivalence relation on a set  $A$  form a partition of  $A$ .

### Partial Orders

A relation is a (weak) partial order if it is reflexive, antisymmetric and transitive.

A relation is a (strong) partial order if it is irreflexive, antisymmetric and transitive  
 (See book).

(69)

classmate

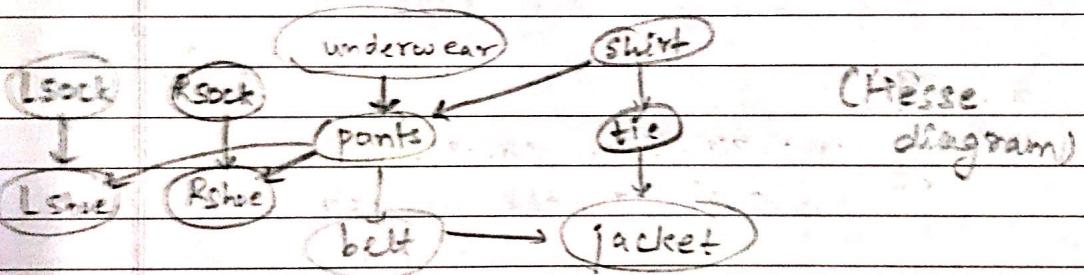
Date \_\_\_\_\_

Page \_\_\_\_\_

A partial order relation is denoted by  $\leq$  instead of  $R$ .

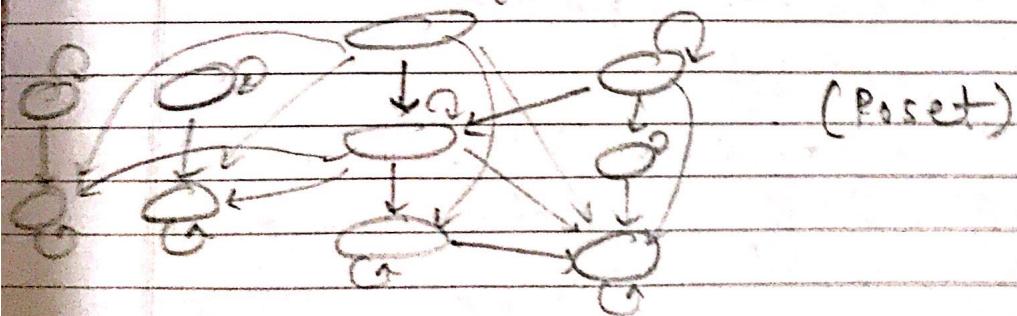
$(A, \leq)$  is actually called partial ordered set. or poset

A poset is a directed graph with vertex set  $A$  and edge set  $\leq$ .



Transitive  $\Rightarrow$  No directed cycles

Antisymmetric  $\Rightarrow$  Only one directed edges b/w two vertices



(70)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

A Hasse diagram for a poset  $(A, \leq)$  is a directed graph with vertex set  $A$  and edge set  $\leq$  minus:

- \* all self-loops, and
- \* all edges implied by transitivity

Theorem A poset has no directed cycles other than self loops.

Proof By contradiction

Suppose  $\exists n (\geq 2)$  distinct elements

$a_1, \dots, a_n$  such that

$a_1 \leq a_2 \leq a_3 \leq \dots \leq a_n \leq a_1 \Rightarrow a_1 \leq a_1$

(induction)

By transitivity  $\rightarrow a_1 \leq a_2 \leq \dots \leq a_n \rightarrow a_1 \leq a_n$

( $\neg$ )

$a_1 = a_n$

Contradicts distinct element assumption.  $\blacksquare$

(IV)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Conclusion of the theorem

So, deleting self-loops from a poset, makes a directed acyclic graph (DAG)

⇒ a and b are incomparable if neither  $a \leq b$  nor  $b \leq a$

⇒ a and b are comparable if ~~neither~~  $a \leq b$  or  $b \leq a$ .

⇒ A total order is a partial order in which every pair of elements is comparable.

Hesse diagram of a total order is a straight line.

A total order consistent with a partial order is called a Topological sort.

A topological of a poset  $(A, \leq)$  is a total order  $(A, \leq_T)$  such that

$$\leq \subseteq \leq_T$$

$$(x \leq y \Rightarrow x \leq_T y)$$

(72)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Theorem Every finite poset has a topological sort.

Proof  $\Rightarrow x \in A$  is minimal if there doesn't exist  $y \in A : y \neq x$  such that  $y \leq x$ .

$\Rightarrow x \in A$  is maximal if  $\forall y \in A, y \neq x$  s.t.  $x \leq y$ .  
 ↓  
 not there exists

Lemma Every finite poset has a minimum element.

$\Rightarrow$  A chain is a sequence of elements such that  $a_1 \leq a_2 \leq \dots \leq a_t$  (length of chain is  $t$ ).  
 distinct

Proof Let  $c: (a_1, \leq a_2, \dots, a_n)$  be the max length chain.

Case 1:  $a \notin \{a_1, \dots, a_n\}$   
 if  $a \leq a_1$ , then  $c$  is not the longest chain.

So,  $a \nleq a_1$

(73)

classmate

Date \_\_\_\_\_  
Page 61Case 2:  $a \in \{a_1, \dots, a_n\}$ 

if  $a \leq a_i$ , then we have  
 a directed cycle which  
 contradicts Theorem 1.

So,  $\nexists a \leq a_i$ , $\therefore \nexists a \in A : a \leq a_i$ ,So, by definition,  $a_i$  is the  
 minimum element.

Add Parallel Task Scheduling

&amp; Dilworth's Lemma.

L-13

(45:30)

Asymptotic Notationtilde  $f(x) \sim g(x)$  if  $\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = 1$ oh, big-oh  $f(x) = O(g(x))$  if  $\lim_{x \rightarrow \infty} \left| \frac{f(x)}{g(x)} \right| < \infty$ 

(finite)

Multiple usage:

Formal math way  
 $f(x) \leq O(g(x))$ ;  $f(x)$  is  $O(g(x))$ ;  $f(x) \in O(g(x))$ 
 $O(g(x))$  is a  
 set of all func.  
 that grow slowly than  $g(x)$

(74)

classmate

Date \_\_\_\_\_  
Page 62Theorem

Let  $f(x) = x$ ,  $g(x) = x^2$   
 Then  $f(x) \in O(g(x))$

Proof

$\lim_{x \rightarrow \infty} \frac{x}{x^2} = 0$  which is finite

Theorem

$x^2 \notin O(x)$

Proof

$\lim_{x \rightarrow \infty} \frac{x^2}{x} = \lim_{x \rightarrow \infty} x$  is infinite.

Q. Is  $x^2 \in O(10^6 x)$  true?

A. No.

Q. Is  $10^6 x^2 \in O(x^2)$  true?

A. Yes

Q. Is  $x^2 + 100x + 10^7 = O(x^2)$  true?

A. Yes

Theorem

$x^{10} \in O(e^x)$

Proof

$\lim_{x \rightarrow \infty} \frac{x^{10}}{e^x} = 0$  which is finite.  
 (L'Hopital's rule)

75

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

63

Theorem

$$4^x \notin O(2^x)$$

Proof

$$\lim_{x \rightarrow \infty} \frac{4^x}{2^x} = \lim_{x \rightarrow \infty} 2^x \rightarrow \infty$$

Q.  
A:Is  $10 \in O(1)$  true?

Yes

 $f(x) \geq O(g(x))$  is meaningless.

omega

$$f(x) = \Omega(g(x)) \text{ if } \lim_{x \rightarrow \infty} \left| \frac{f(x)}{g(x)} \right| > 0$$

Theorem

$$f(x) = O(g(x)) \text{ iff } g(x) = \Omega(f(x))$$

$$f(x) \leq O(g(x)) \text{ iff } g(x) \geq \Omega(f(x))$$

$$x^2 = \Omega(x)$$

$$2^x = \Omega(x^2)$$

$$\frac{x}{100} = \Omega(100x + 25)$$

(36)

classmate  
Date \_\_\_\_\_  
Page 64

theta  $f(x) = \Theta(g(x))$  if  $\lim_{x \rightarrow \infty} \left| \frac{f(x)}{g(x)} \right| < \infty$

Theorem  $f(x) = \Theta(g(x))$  iff  $f(x) = O(g(x))$  &  
 $f(x) = \Omega(g(x))$

$$10x^3 - 20x + 1 \in \Theta(x^3)$$

$$\frac{x}{\ln(x)} \notin \Theta(x) \quad \left| \frac{x}{\ln(x)} \in O(x) \right.$$

$T(n) = \Theta(n^2)$  means  $T$  grows quadratically in  $n$ .

$$O \Rightarrow \leq$$

$$L \Rightarrow \geq$$

$$\Theta \Rightarrow =$$

$$o \Rightarrow < \quad (\leq \text{ not } =)$$

$$w \Rightarrow > \quad (\geq \text{ not } =)$$

(77)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

65

little oh  $f(x) = o(g(x))$  iff  $\lim_{x \rightarrow \infty} \left| \frac{f(x)}{g(x)} \right| = 0$

little omega  $f(x) = \omega(g(x))$  iff  $\lim_{x \rightarrow \infty} \left| \frac{f(x)}{g(x)} \right| = \infty$

$$\frac{x}{\ln(x)} \in o(x)$$

$$\frac{x}{100} \notin o(x) \quad \frac{x}{100} \in \Theta(x)$$

$$x^2 \in \omega(o)$$

what not do  
example.

(95)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-13Graph I : BFS

- applications of graph search
- graph representation
- BFS → Breadth-first search

Graph search

"explore" a graph

Eg:- find path

Recall: graph  $G = (V, E)$

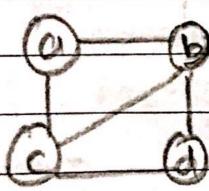
Every undirected graph is a directed graph  $\{u, v\} \rightarrow \{v, u\}$

$\downarrow$  set of vertices

$\downarrow$  set of edges

Undirected,  $e = \{v, w\} \leftarrow$  unordered pairs  
Graph OR

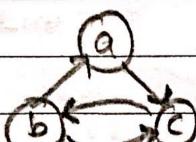
Directed  $\rightarrow e = (v, w) \leftarrow$  ordered pairs  
Graph



UNDIRECTED

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

$$E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{a, c\}, \{b, d\}\}$$



DIRECTED

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

$$E = \{(a, c), (b, c), (c, b), (b, a)\}$$

(96)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Applications:

- web crawling
- social networking
- network broadcast
- garbage collection
- model checking
- check mathematical conjecture
- solving puzzles & games

Pocket Cube:  $2 \times 2 \times 2$ 

- configuration graph
- vertex for each possible state of cube

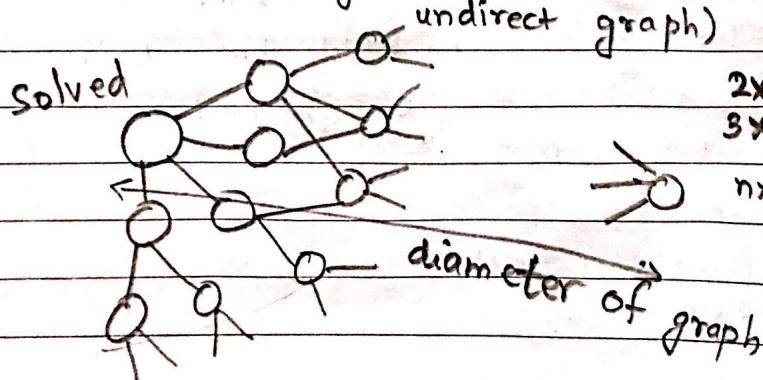
$$\# \text{ vertices} = 8! \cdot 3^8 = 264,539,520$$

24 symmetries

$\frac{1}{3}$ rd of states can only be reached  
without breaking it apart

- edge for each possible move

(as every move is undoable,  
undirected graph)



$$2 \times 2 \times 2 : 11$$

$$3 \times 3 \times 3 : 20$$

$$n \times n \times n : \Theta\left(\frac{n^2}{\lg n}\right)$$

(97)

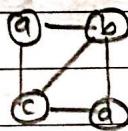
classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Graph representation:Adjacency Listsarray  $\text{Adj}$  of size  $|V|$ 

- each element is a pointer to a linked list

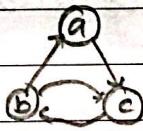
for each vertex  $u \in V$ , $\text{Adj}[u]$  stores  $u$ 's neighbours $\hookrightarrow \{v \in V \mid (u, v) \in E\}$ 

$$\text{Adj}[a] = \{b, c\}$$

$$\text{Adj}[b] = \{a, c, d\}$$

$$\text{Adj}[c] = \{a, b, d\}$$

$$\text{Adj}[d] = \{b\}$$



$$\text{Adj}[a] = \{c\}$$

$$\text{Adj}[b] = \{a, c\}$$

$$\text{Adj}[c] = \{b\}$$

Object-oriented patternVertex  $v \mapsto v.\text{neighbours} = \text{Adj}[v]$ 

The reason why object-oriented is not used is because you can simply have adjacency list of sub-graph in the direct method.

If you are dealing with only one graph,  
object-oriented method is more cleaner.

(98)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Implicit representation

- $\text{Adj}(u)$  is a function
- $v.\text{neighbours}()$  is a method  
uses less space

This would be helpful for puzzles like

Rubik's cube.

Fun fact:  $7 \times 7 \times 7$  has configurations more than number of atoms in the known universe.

Space used by the basic rep. is  $\Theta(|V|+|E|)$

Breadth-first Search

- visit all the nodes reachable from given  $s \in V$
- $O(V+E)$  time
- look at nodes reachable in 0 moves, 1 move, 2 moves, ...  
 $\{s\}$        $\text{Adj}[s]$
- careful to avoid duplicates

(99)

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_BFS(s, Adj)

level = {s: 0}

parent = {s: None}

i = 1

frontier = [s] ← level i-1

while frontier:

next = [] ← level i

for u in frontier:

for v in Adj[u]:

if v not in level:

level[v] = i

parent[v] = u

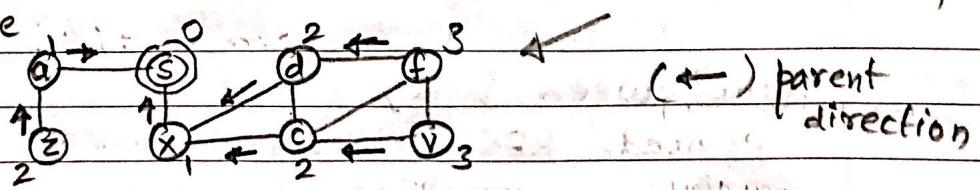
next.append(v)

frontier = next

i += 1

UNDIRECTED Graph

Example



i=1      next = [a, x]

i=2      next = [z, d, c]

i=3      next = [f, v]

i=4      next = []

(100)

CLASSMATE

Date \_\_\_\_\_

Page \_\_\_\_\_

Shortest paths

$$v \leftarrow \text{parent}[v] \leftarrow \text{parent}[\text{parent}[v]]$$

↑  
↓  
↑  
↓  
S

is the shortest path from  $s$  to  $v$ .  
of length  $\text{level}[v]$ .

Each vertex is checked once

Each edge is considered once (or twice)  
(UBDG)

$$\sum_{v \in V} |\text{Adj}[v]| = \begin{cases} 2|E| & (\text{UDG}) \\ |E| & (\text{DG}) \end{cases}$$

R-13 (Halloween Day)

Revised BFS

and operations on Byte Vs Word  
(8 bit) (16 bit)

101

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

---

L-14 Graph II: DFS

- depth-first search
- edge classification
- cycle detection
- topological sort

---

Depth-first search (DFS)

- recursively explore graph, backtracking as necessary
- careful not to repeat

parent =  $\{ s : \text{None} \}$

DFS-Visit( $V, \text{Adj}, s$ ):

```

for  $v$  in  $\text{Adj}[s]$ :
    if  $v$  not in parent:
        parent[ $v$ ] =  $s$ 
        DFS-Visit( $v, \text{Adj}, v$ )
    
```

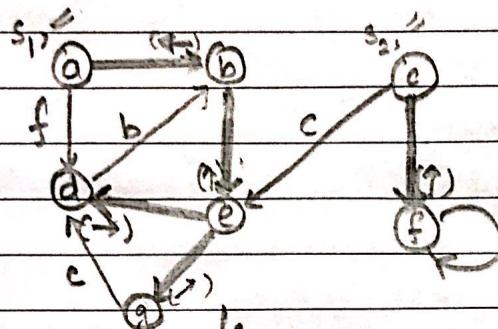
DFS( $V, \text{Adj}$ ):

```

parent = {}
for  $s$  in  $V$ :
    if  $s$  not in parent:
        parent[ $s$ ] = None
        DFS-Visit( $s, \text{Adj}, s$ )
    
```

102

CLASSMATE

Date \_\_\_\_\_  
Page \_\_\_\_\_(Not strongly connected,  
can get from 'b' to 'c')( $\leftarrow$ ) parent directionRunning time $O(V+E)$  (Linear time)- visit each vertex once in DFS alone  $O(V)$ - DFS-Visit(...,  $v$ ,  $v$ ) called atmost  
once per vertex  $v$ .↳ pay  $|Adj[v]|$  $O(V+E)$ 

$$O\left(\sum_{v \in V} |Adj[v]|\right) = O(E)$$

Edge classification- free edges (parent pointers) (Bold)

visit new vertex via that edge.

- forward edges ( $f$ )goes from a node to a descendant in  
the tree- backward edges ( $b$ )goes from a node to an ancestor in  
the tree- cross edges (or others) ( $c$ )

between two non-ancestor-related subtrees

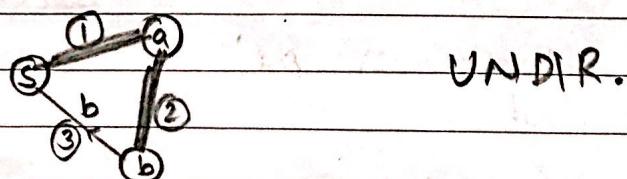
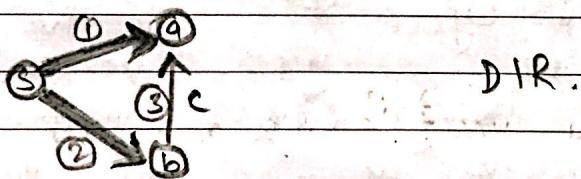
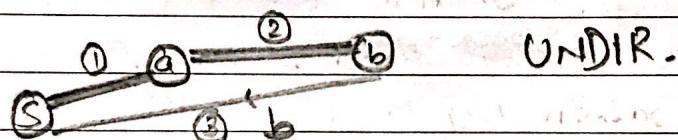
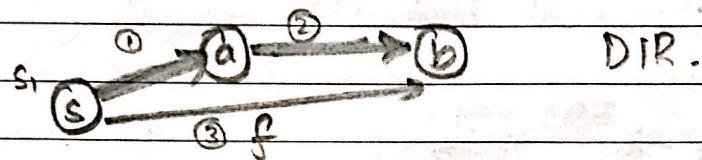
(03)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

For undirected graphs,  
forward and cross edges don't  
exist.



Useful for cycle detection  
and topological sort.

(104)

classmate

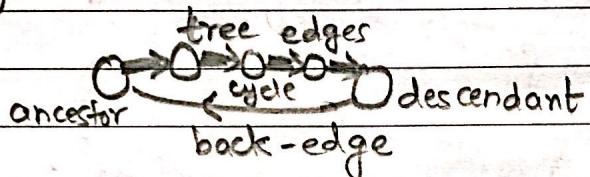
Date \_\_\_\_\_

Page \_\_\_\_\_

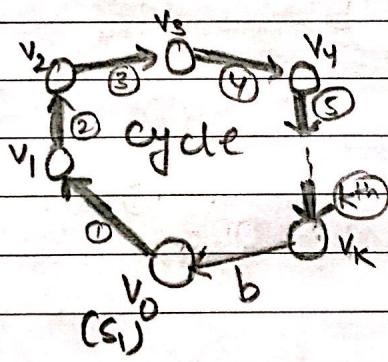
Cycle detection

Thm.  $G$  has a cycle  $\Leftrightarrow$  DFS has a back edge

Proof. ( $\Leftarrow$ )



( $\Rightarrow$ )



Assuming  $v_0$  is the first vertex in cycle visited by DFS.  
 then  $v_k \rightarrow v_0$  is back-edge

105

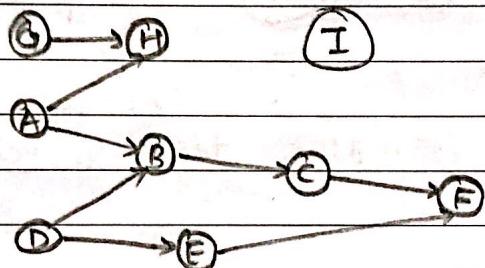
classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

## Job Scheduling

given directed acyclic graph (DAG)  
 order vertices such that all edges point  
 from lower vertices to higher order



Topological sort. (Sorting vertices);  
 run DFS

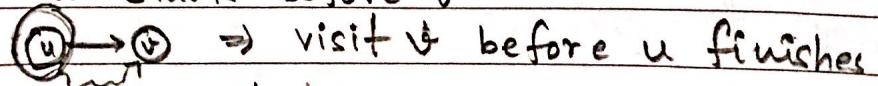
Output reverse of finishing times of  
 vertices.

### Correctness

for any edge  $e = (u, v)$

$v$  finishes before  $u$  finishes. (To show)

Case 1:  $u$  starts before  $v$



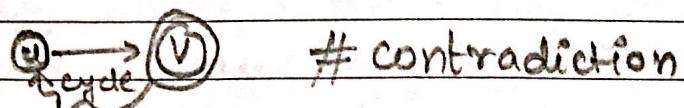
visit  $u \rightarrow$  start  $u \rightarrow$  visit  $v \rightarrow$  start  $v \rightarrow$  finish  $v \rightarrow$  finish  $u$

106

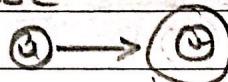
classmate \_\_\_\_\_

Date \_\_\_\_\_  
Page \_\_\_\_\_

Case 2:  $v$  starts before  $u$



else



then  $v$  finishes first  
(Actually even before  
 $u$  starts)

R-14

Procedurally analysed  
example of applying DFS  
on a direct graph and an undirected  
graph.

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

L-15      Shortest Path I : Intro

- Weighted graphs
- General approach
- Negative edges
- Optimal substructure

$G(V, E, w)$

vertices      edges      weight  $w: E \rightarrow \mathbb{R}$

Two algos

- Dijkstra (after Edsger Dijkstra)
  - non-negative weight edges.
  - $O(V \lg V + E) \approx O(E)$        $\because E = O(V^2)$   
(almost)
- Bellman - Ford
  - +/- weight edges
  - $O(VE)$

108

classmate

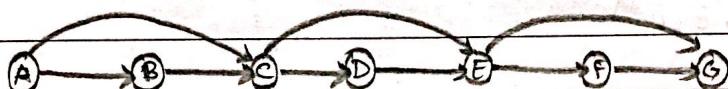
Date \_\_\_\_\_

Page \_\_\_\_\_

Path  $p = \langle v_0, v_1, \dots, v_k \rangle$  (sequence)  
 $(v_i, v_{i+1}) \in E \quad \forall 0 \leq i \leq k$

$$w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

find  $p$  with minimum weight.



$A \rightarrow B$  1 paths

$A \rightarrow C$  2 paths

$A \rightarrow E$  4 paths

$A \rightarrow G$  8 paths

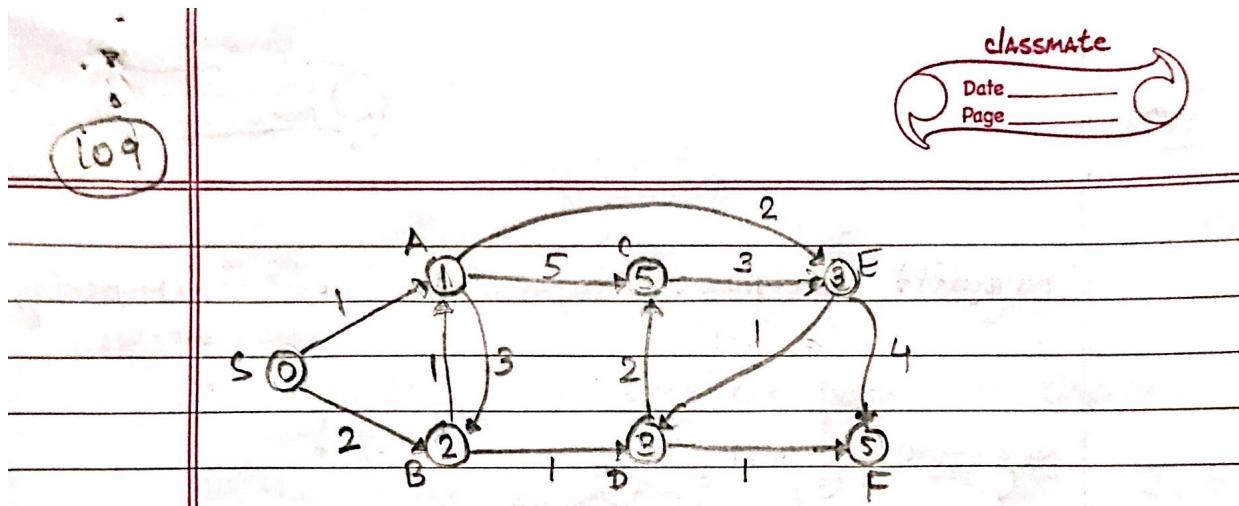
Paths increase exponentially

Weighted graphs

$v_0 \xrightarrow{P} v_k$  ( $v_0$ ) is a path from  $v_0$  to  $v_k$  of weight 0.

Shortest path weight from  $u$  to  $v$  as

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \xrightarrow{P} v\} & \exists \text{ any such path} \\ \infty & \text{otherwise} \end{cases}$$



$d(u)$ : current weight

$\pi(u)$ : predecessor

	S	A	B	C	D	E	F
	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
From S	0	1	2	$\infty$	$\infty$	$\infty$	$\infty$
From A	0	1	2( $\infty$ )	6	$\infty$	3	$\infty$
From C	0	1	2	6	$\infty$	3( $\infty$ )	$\infty$
From E	0	1	2	6	4	3	7
From D	0	1	2	6	4	3	5( $\infty$ )
From F	0	1	2	6	4	3	5
From B	0	1( $\infty$ )	2	6	3( $\infty$ )	3	5
From D	0	1	2	5( $\infty$ )	4	3	5
From C	0	1	2	5	4	3	5

$$\delta(S, A) = 1 \quad \delta(S, B) = 2 \quad \delta(S, C) = 5$$

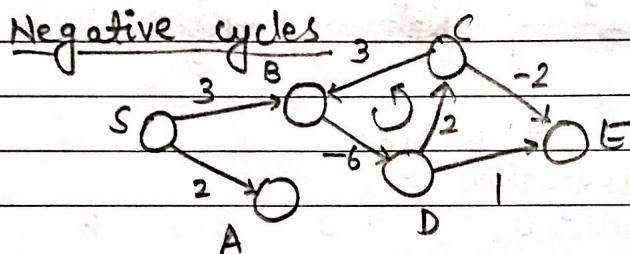
$$\delta(S, D) = 4 \quad \delta(S, E) = 3 \quad \delta(S, F) = 5$$

(110)

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_

Negative weights : reverse tolle, social networking  
(likes & dislikes)



$$s(s, s) = 0 \quad s(s, A) = 2$$

$$s(s, B) = s(s, C) = s(s, D) = s(s, E) = -\infty$$

Bellman-Ford detects negative cycles and thus, terminates in finite time.

General structure (no negative cycles)

Initialize for  $u \in V$      $d[u] \leftarrow \infty$      $\pi[u] \leftarrow \text{NIL}$   
 $d[s] \leftarrow 0$

Repeat    select edge  $(u, v)$     [somehow]  
 "Relax" edge  $(u, v)$  : if     $d[v] > d[u] + w(u, v)$   
 $d[v] = d[u] + w(u, v)$   
 $\pi[v] \leftarrow u$

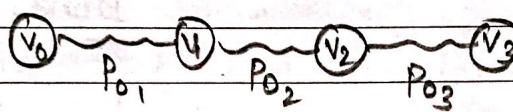
Until all edges have  $d[v] \leq d[u] + w(u, v)$

(11)

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_Optimal substructure

- Subpaths of a shortest path are shortest paths



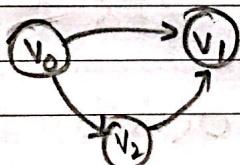
$$\{P_{0,1}, P_{0,2}, P_{0,3}\} = SP$$

$$\text{then } \{P\} = SP$$

$$\{P_{0,2}\} = SP$$

$$\{P_{0,3}\} = SP$$

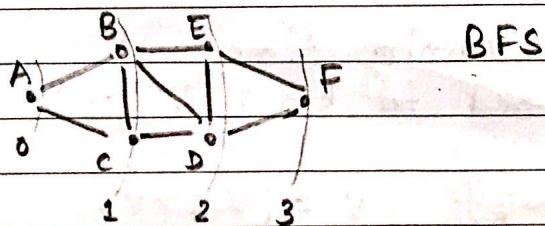
Triangle  
inequality  
(used in i-16)



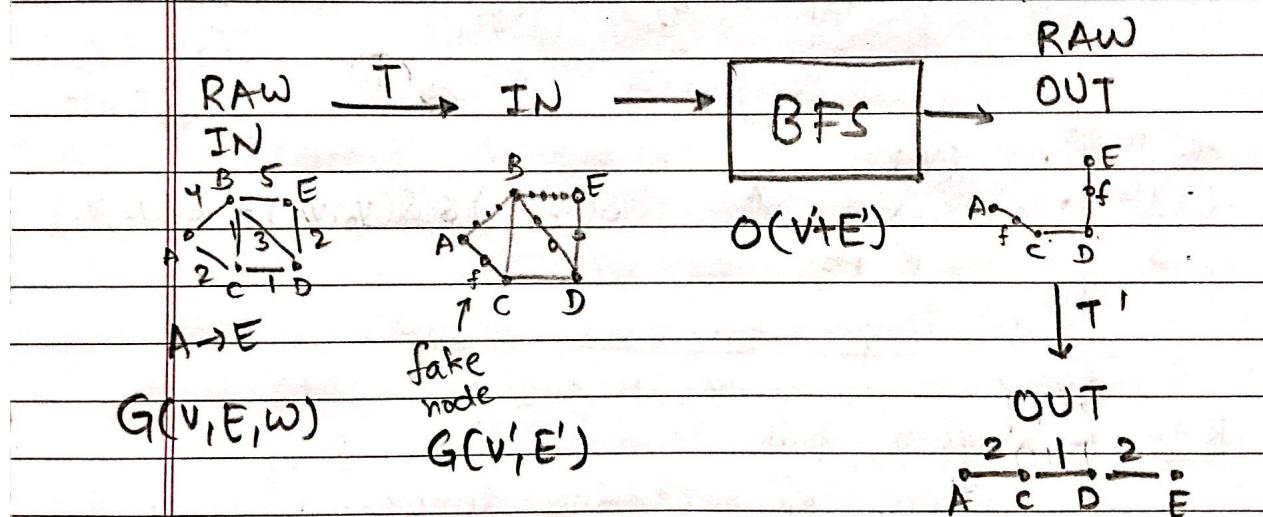
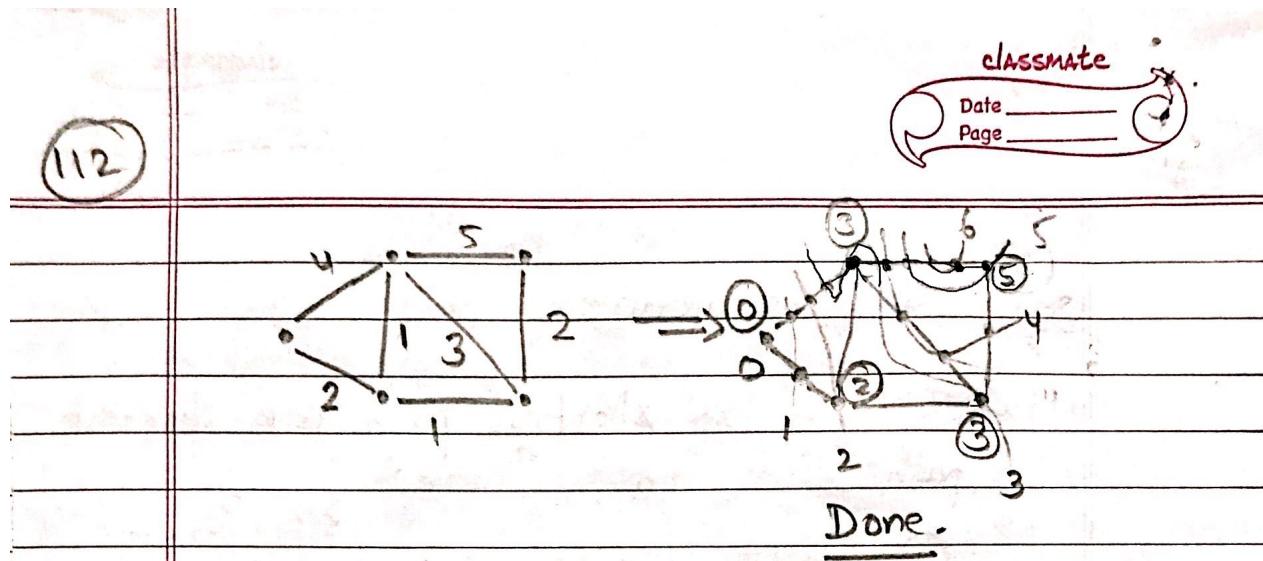
$$\delta(v_0, v_2) \leq \delta(v_0, v_1) + \delta(v_1, v_2)$$

R-15Shortest path

Thinking of something simple.



So, now converting a weighted graph to a non-weighted graph

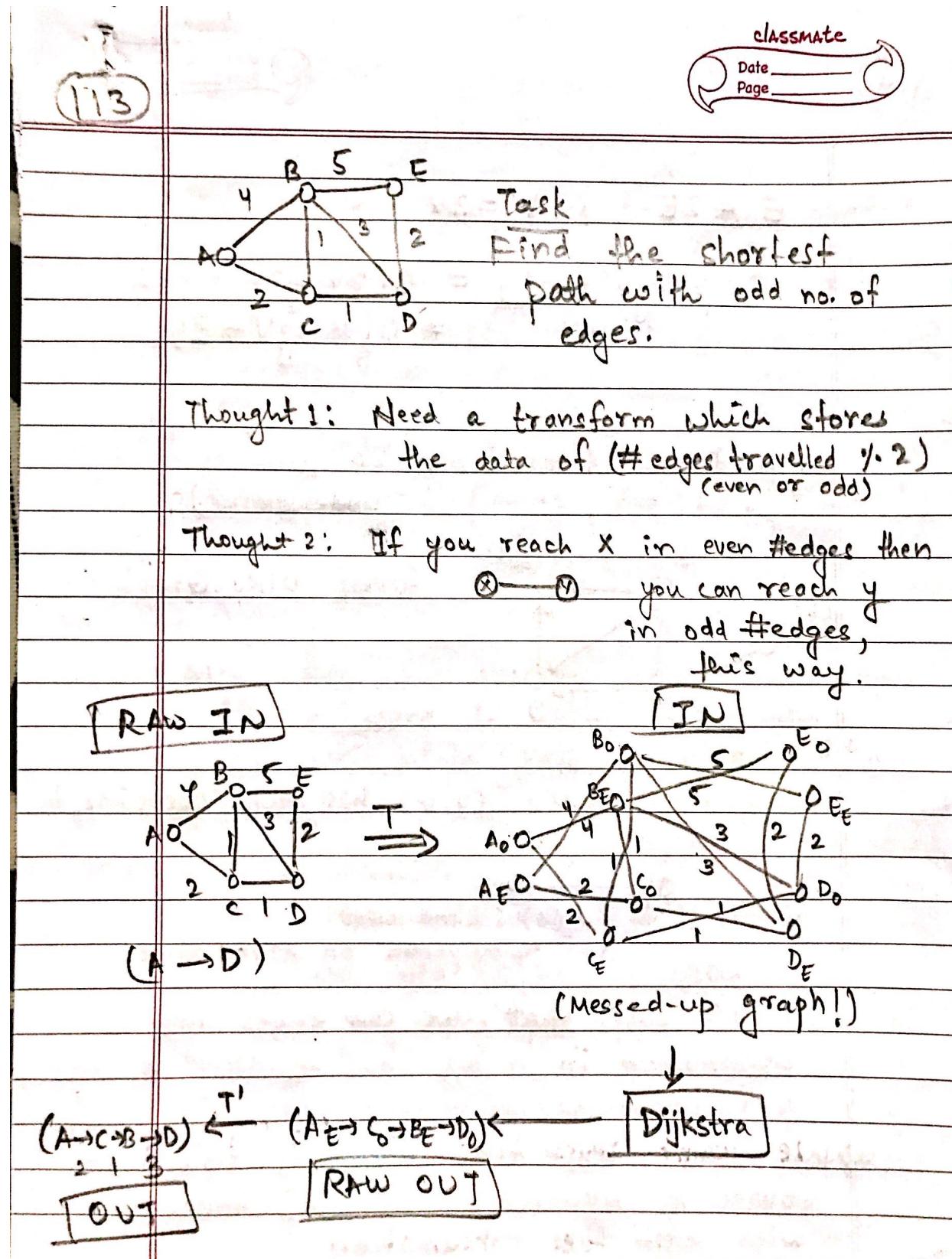


To find out running time,  
we need to know  $V'$ ,  $E'$ .

$$E' = O(WE) \quad V' = (WE + V)$$

$$O(V' + E') = O(WE + V)$$

For small paths, this is a nice algo.



114

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

$$E' = 2E \quad V' = 2V$$

$$\begin{aligned} O(V' \log V' + E') &= O(2V \log 2V + 2E) \\ &= O(V \log V + E) \end{aligned}$$

A hard one (took me 20 min to understand)

Given DIR. Graph

```

graph LR
    A((A)) --> B((B))
    A((A)) --> C((C))
    B((B)) --> C((C))
    C((C)) --> D((D))
    C((C)) --> E((E))
  
```

Now each edge  $(u, v)$  has two costs:  $f_c, t_c$ .

$u \xrightarrow{f_c: \text{fuel cost}} v$

$t_c(s) : \text{time cost}$   
 $\leftarrow \text{depends on start time.}$

We are given that we can reach any destination in a day and  $t_c((u, v), s)$  has resolution of minutes.

$\Rightarrow$  We want least time consuming path from source to destination, if many then one with least fuel consumption

115

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

In previous ques, we had two states:  
even & odd # edges.

Here, every vertex will have 1440 [60x24]  
states ( $M=1440$ )

$$\begin{aligned} \text{Transform}(T) : V &\xrightarrow{(\times M)} V, t \\ (u, v) &\xrightarrow{(\times M)} (u, t), (v, t+t_c(u, v), t)) \end{aligned}$$

fuel cost =  $f_c$

Also, say we are at vertex  $u$  at time  $t$ .  
if we were to wait for one minute  
then we might have a smaller  $t_c$ .

So, new edges  $(u, t), (u, t+1)$  are added.  
fuel cost = 0.

(You can think these as extension of  $(u, u) \xrightarrow{(\times M)} (u, t), (u, t+1)$ )

Now, we use Dijkstra's algo to  
find required path from (source, 0)  
to (dest., i), incrementing i till  
we get a finite soln.

(116)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

 $i=0 \quad \infty$   
 $i=1 \quad \infty$   
 $i=2 \quad \infty$   
 $i=3 \quad \infty$ 
 $i=k+1 \quad \infty$   
 $i \leq k \quad p$   
 $i=k+1$ 

No such path.

eco-friendly fastest path.

Using this we can get the shortest path taking a certain amount of time.  
 (Feature)

Complexity of Dijkstra  
 $= O(V \lg V + E)$  (To be proved  
 in L-1G)

Here,  $V' = V \cdot M$

$E' = E \cdot M + V \cdot M$

Complexity of algo  
 $= O(V' \lg V' + E')$   
 $= O(V \cdot M \lg (V \cdot M) + E \cdot M + V \cdot M)$   
 $= O(V \lg V + E)$

(though, constant is much worse than  
 Dijkstra)

117

classmate

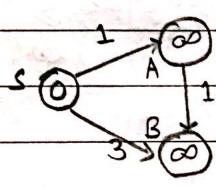
Date \_\_\_\_\_

Page \_\_\_\_\_

L-16

Shortest Paths II : Dijkstra

- Review
- Shortest paths - DAGs
- SPs - graphs w/o -ve edges
- Dijkstra's algo

Review

$d[v]$ : length of current s.p. from source ( $s$ ) to  $v$ .

$$d[s] = 0, d[A] = \infty, d[B] = \infty$$

$\delta(s, v)$ : length of a shortest path from  $s$  to  $v$ .

$d[s] = 0, d[A] = 1, d[B] = 3$        $\pi(v)$ : predecessor of  $v$  in the s.p. from  $s$  to  $v$ .

$$\begin{aligned} d[s] &= 0, d[A] = 1, d[B] = 2 \\ \pi[A] &= s, \pi[B] = A \end{aligned}$$

S.p. from  $s$  to  $B$ :  $B \leftarrow \pi[B] \leftarrow \pi[\pi[B]] \leftarrow \dots \leftarrow s$

$$= A \qquad \qquad = s$$

$$\Rightarrow B \xleftarrow{1} A \xleftarrow{1} s$$

$$d[B] = 2 \quad w(p) = 2$$

(118)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

$\text{Relax}(u, v, w)$ :  
 weight

if  $d[v] > d[u] + w(u, v)$

$$d[v] = d[u] + w(u, v)$$

$$\pi[v] = u$$

Lemma: The relaxation operation maintains the invariant that  $d[v] \geq \delta(s, v) \forall v \in V$ .

Proof: By Induction on # steps.

Base case:  $n=0$   $d[v] = \begin{cases} 0 & v=s \\ \infty & \text{otherwise} \end{cases}$

satisfies

Inductive step:

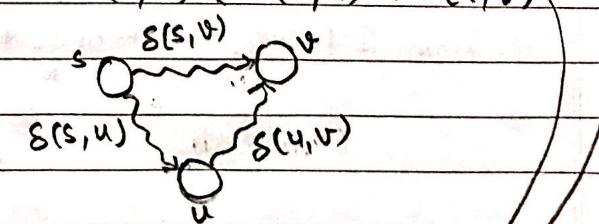
For  $n=k$  steps, the lemma satisfies.

In  $(k+1)^{\text{th}}$  step, we Relax  $(u, v, w)$ .

We know that  $d[u] \geq \delta(s, u)$

By triangle-inequality

$$\delta(s, v) \leq \delta(s, u) + \delta(u, v)$$



$$\delta(s, v) \leq d[u] + \delta(u, v)$$

$$\delta(u, v) \leq w(u, v)$$

$$\therefore \delta(s, v) \leq d[u] + w(u, v)$$

Proved.

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

119

## DAGs (can't have (-ve) cycles)

1. Topologically sort the DAG.  
Path from  $u$  to  $v$  implies that  $u$  is before  $v$  in the ordering.
2. One pass over vertices in topologically sorted order relaxing each edge that leaves that vertex.

$O(V+E)$  time

- This works for any starting vertex.

Step	0	1	2	3	4	5	6	7
Step 0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
Step 1	$\infty$	0	2	6	$\infty$	$\infty$	$\infty$	$\infty$
Step 2	$\infty$	0	<u>2</u>	6( $<9$ )	6	4	$\infty$	$\infty$
Step 3	$\infty$	0	2	<u>6</u>	5( $<6$ )	4( $<7$ )	$\infty$	$\infty$
Step 4	$\infty$	0	2	6	<u>5</u>	3( $<4$ )	$\infty$	$\infty$
Step 5	$\infty$	0	2	6	5	3	$\infty$	$\infty$

(120)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Demo of DijkstraMechanically validating Dijkstra  
(28:28)

Dijkstra is a greedy algo.

It is easy to understand but not so easy to prove its correctness.

Dijkstra ( $G, \omega, s$ ) :

Initialize ( $G, s$ )       $S' \leftarrow \emptyset$        $Q = V[G]$

$d[s] = 0$

while  $Q \neq \emptyset$

$u \leftarrow \text{extract-min}(Q)$

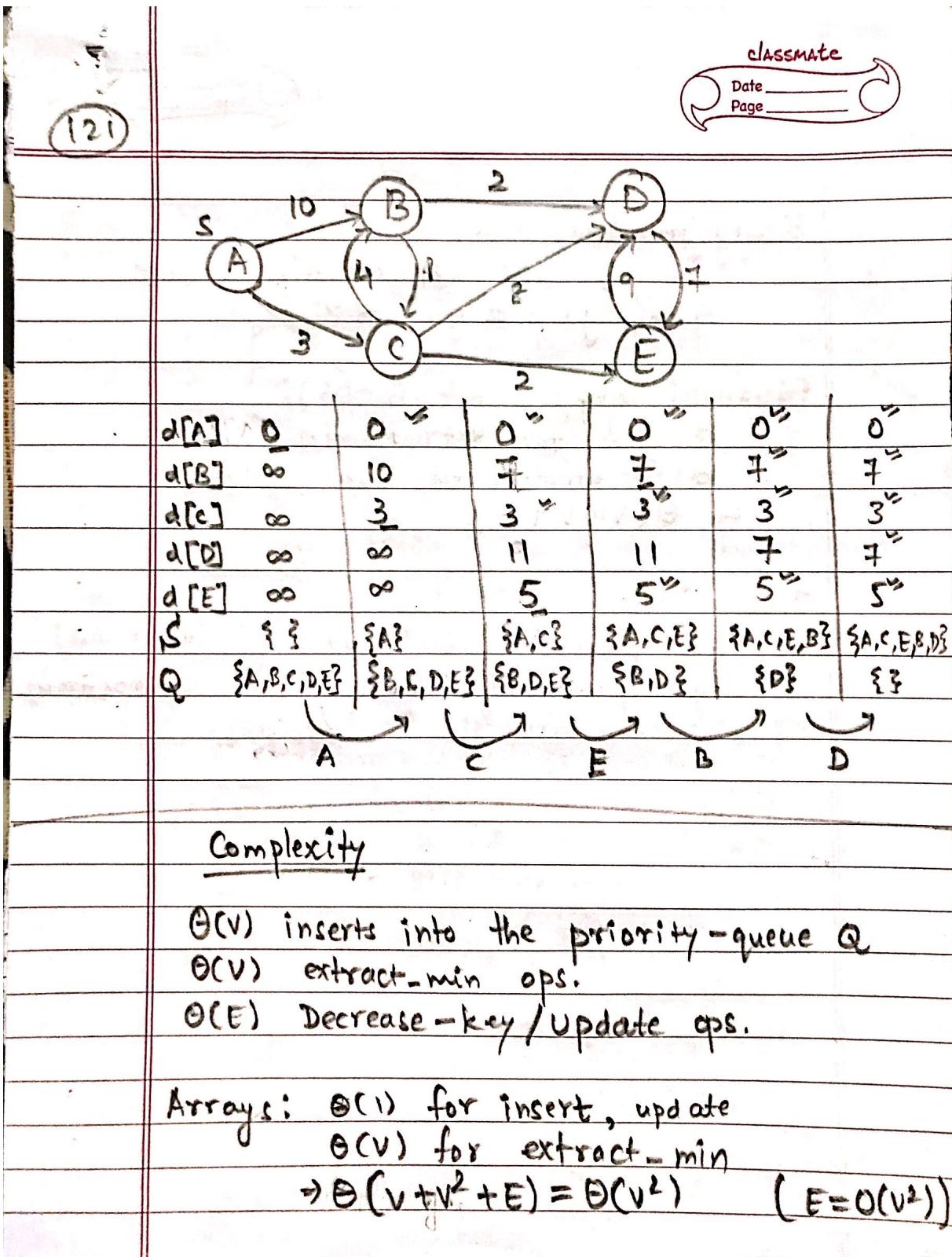
$S' \leftarrow S' \cup \{u\}$

    for each vertex  $v \in \text{Adj}[u]$

        Relax ( $u, v, \omega$ )

$Q$  is a priority-queue of  $d[v] \neq v \in V$ .  
 $\text{extract-min}(Q)$  returns vertex not yet evaluated and has minimum  $d[\cdot]$ .

Proof in CLRS Textbook (Introduction to Algorithms, 3rd ed.)  
(Reference Book 1)



(122)

classmate \_\_\_\_\_

Date \_\_\_\_\_

Page \_\_\_\_\_

Binary min heap:

$$\Theta(\lg V) \text{ for extract-min, update}$$
$$\Rightarrow \Theta(V \lg V + E \lg V)$$

Fibonacci heap (not 6.006):

$$\Theta(\lg V) \text{ for extract-min}$$

 $\Theta(1)$  amortized for update

$$\Rightarrow \Theta(V \lg V + E)$$

Theoretically,

complexity is  $\Theta(E \lg V)$  (worst-case)but practically,  $\Theta(V \lg V + E)$  (amortized).

classmate  
Date \_\_\_\_\_  
Page \_\_\_\_\_

(23)

R-16      2x2x2 Rubik's cube

Plastic faces                          Wireframe

8 cublets                                  24 faces

Configuration is an array of 24

yob	boy	boy					
0	1	2	3				23
(fur)	(urf)	(urf)					

Total configurations: 24!

Six types of moves: f<sub>c</sub>, f<sub>cc</sub>, r<sub>c</sub>, r<sub>cc</sub>, u<sub>c</sub>, u<sub>cc</sub>

Finding Starcraft game strategy  
(Using Dijkstra)

124

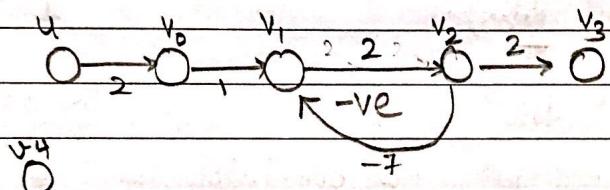
classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-17Shortest Paths III : Bellman-Ford

- Negative cycles
- Generic S.P. Algorithm
- Bellman-Ford Algo
  - Analysis
  - Correctness



$\delta(u, v_1), \delta(u, v_2), \delta(u, v_3)$  not defined ( $-\infty$ )

$$\delta(u, v_0) = 2$$

$$\delta(u, v_4) = \infty$$

This is what is required.

Generic S.P. Algo

Initialize for  $v \in V$      $d[v] \leftarrow \infty$      $\pi[v] \leftarrow \text{None}$

$$d[s] = 0$$

Main loop : Repeat  $\rightarrow$  select edge [somehow]  
 $\rightarrow$  Relax  $(u, v, w)$

until you can't relax anymore.

125

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Problems

① Complexity could be exponential time

(even for +ve edge weights)

② Will not terminate if there is a negative weight cycles reachable from the source

If we assume no cycles, i.e., DAGs, we have a  $O(V+E)$  time algo (using topo. sort)

If we assume no -ve weights, we have a  $O(V|V|V+E)$  time algo (Dijkstra)

Bellman-Ford ( $G, w, s$ )

Initialize()

for  $i=1$  to  $i=|V|-1$  $O(VE)$ for each edge  $(u, v) \in E$ Relax  $(u, v, w)$ 

$\boxed{\text{Relax}(u, v, w):}$   
 if  $d[v] > d[u] + w(u, v)$ :  
 $d[v] = d[u] + w(u, v)$   
 $\pi[v] = u$

 $O(E)$  Check{ for each edge  $(u, v) \in E$ if  $d[v] > d[u] + w(u, v)$ 

then report -ve cycle exists.

Complexity :  $O(VE)$

126

classmate

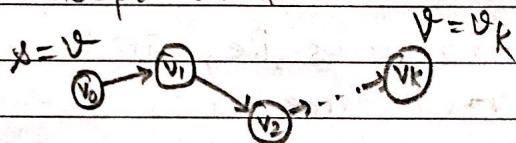
Date \_\_\_\_\_

Page \_\_\_\_\_

Thm. If  $G = (V, E)$  contains no -ve weight cycle then after B-F executes,  $d[v] = \delta(s, v) \forall v \in V$

Corollary If a value  $d[v]$  fails to converge after  $|V|-1$  passes, there exists a -ve weight cycle reachable from  $s$ .

Concept used



$v_0, v_1, \dots, v_k$  is a path from  $s$  to  $v$   
 $\Rightarrow k \leq |V|-1$

If  $k > |V|-1$  then it is a walk with atleast a cycle.

Proof

let  $v \in V$

$$p = \langle v_0, v_1, \dots, v_k \rangle \quad v_0 = s \quad \text{to} \quad v_k = v$$

This path  $p$  is a shortest path with min # edges.  
 No -ve weight cycle  $\Rightarrow p$  has no cycles  $\Rightarrow k \leq |V|-1$

After one pass thru  $E$ , we have  $d[v_i] = \delta(s, v_i)$

because we will relax edge  $(v_0, v_1)$

(If not, then  $p$  is not a shortest path.)

(127)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

After 2 passes,  $d[v_2] = \delta(s, v_2)$  because in 2<sup>nd</sup> pass, we will relax  $(v_1, v_2)$  as well.

After  $k$  passes,  $d[v_k] = \delta(s, v_k)$   
 $\because k \leq |V|-1$

$|V|-1$  passes  $\Rightarrow$  all reachable vertices have  $d[v] = \delta(s, v)$

Proof (Corollary) After  $|V|-1$  passes, we find an edge that can be relaxed.

$\Rightarrow$  current shortest path from  $s$  to some vertex  $v$  is not simple.

$\Rightarrow$  Have a repeated vertex

$\Rightarrow$  A repeated cycle

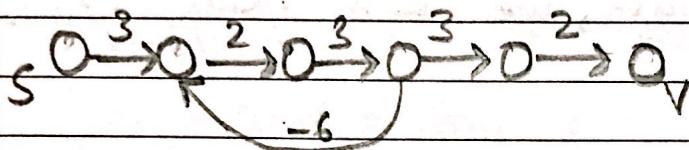
$\Rightarrow$  As the path weight decreases this cycle has -ve weight.

128

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Shortest Simple Path

shortest simple path is 13.

Bellman-Ford can't solve this.

Turns out this is a NP-hard problem.  
i.e. we don't know an algo that is  
better than exponential time to solve  
this problem.

(129)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-18 Shortest Paths IV : Speeding up Dijkstra

- Single source, single target
- Bi-directional search
- Goal-directed search - potentials, landmarks

Unlike previous lectures, we're going to be talking about optimizations that don't change the worst-case, or asymptotic complexity, but improve empirical, real-life performance. (Performance in average case)

1.  $\text{Dijkstra}(G, W, s)$ :

2. Initialize( $s$ )  $\leftarrow d[s] = 0, d[u \neq s] = \infty$
3.  $Q \leftarrow V[G]$
4. while  $Q \neq \emptyset$ 
  5. do  $u \leftarrow \text{extract-min}(Q)$
  6. for each vertex  $v \in \text{Adj}[u]$ 
    7. do  $\text{Relax}(u, v, w)$

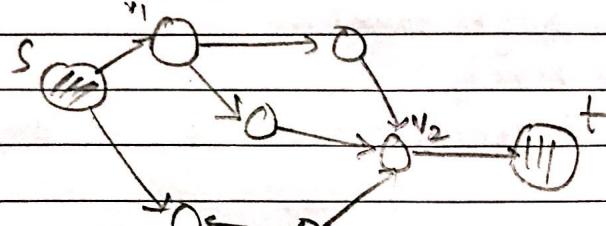
If we are asked to find shortest path from  $s$  to  $t$ . then we can replace 4. by while( $Q \neq \emptyset \& u \neq t$ ) This optimizes the run-time, worst-case run-time still stays same but average improves.

130

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Bi-directional search

$$\pi_f[v_1] = s$$

$$\pi_b[v_2] = t$$

- Alternate forward search from  $s$  backward search from  $t$   
(following edges backward)

$$d_f[s] = 0 \quad d_b[t] = 0$$

$$d_f[u \neq s] = \infty \quad d_b[u \neq t] = \infty$$

distances for  
forward search

$\pi_f$ : normal parent

Priority Queues:

distances for  
backward search

$\pi_b$ : reverse parent

$Q_f$  (forward)  
 $Q_b$  (backward).

Termination condition

- some vertex  $u$  has been processed both in the forward search & backward search i.e., deleted from  $Q_f$  &  $Q_b$ .

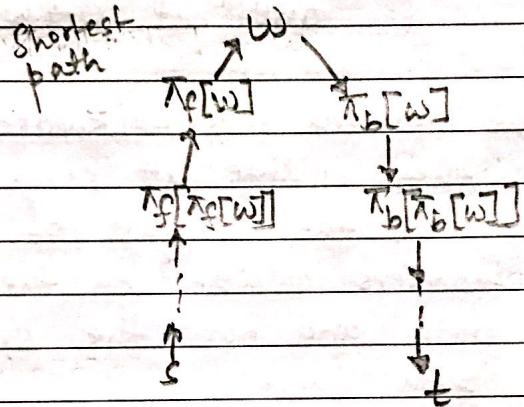
(131)

classmate

Date \_\_\_\_\_

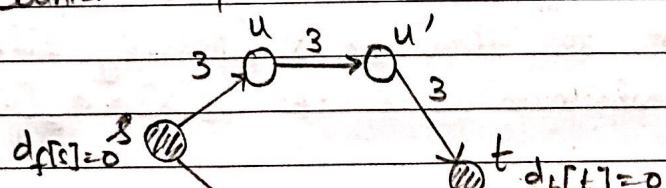
Page \_\_\_\_\_

Finding the shortest path  
 Claim: If  $w$  was processed first from both  $Q_f$  &  $Q_b$ .



This claim is actually false.

Counter-example:



Forward

$$\begin{aligned} \text{from } s \quad ① \quad df[u] &= 3 \\ & df[w] = 5 \\ Q_f &= \{u, u', w, t\} \end{aligned}$$

from u

$$\begin{aligned} ② \quad df[u'] &= 6 \\ Q_f &= \{u', w, t\} \\ \text{from } w \quad ③ \quad df[t] &= 10 \\ Q_f &= \{u', t\} \end{aligned}$$

Backward

$$\begin{aligned} \text{from } t \quad ① \quad db[w'] &= 3 \\ & db[w] = 5 \\ Q_b &= \{u, u', w, t\} \end{aligned}$$

from u'

$$\begin{aligned} ② \quad db[u] &= 6 \\ Q_b &= \{u, w, t\} \\ \text{from } w \quad ③ \quad df[t] &= 10 \\ Q_b &= \{u, t\} \end{aligned}$$

(132)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

w has been processed from  $Q_b$  &  $Q_f$

$$\pi_b[w] = t \quad \pi_f[w] = u$$

But, the shortest path is  $s \rightarrow u \rightarrow u' \rightarrow t$   
and not  $s \rightarrow w \rightarrow t$ .

When the forward and backward frontiers collide, they collide at some vertex, regardless of the weights of the edges.

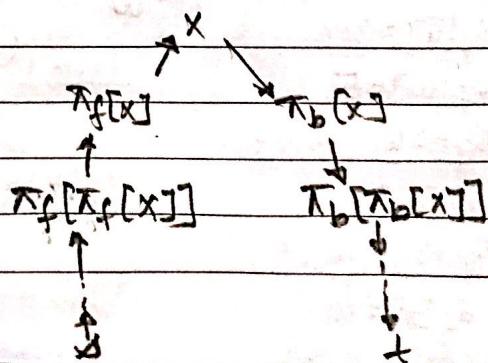
So, the frontiers collide on the shortest length path and not the shortest weight path (here).

So, how do we find the shortest weight path.

We want to find an x (possibly diff from w) with minimum value of  $d_f[x] + d_b[x]$

(Here,  $d_f[u] + d_b[u] = 3 + 6 = 9 < 10$ )

Shortest path:



133

classmate

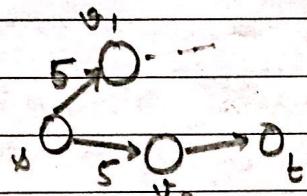
Date \_\_\_\_\_

Page \_\_\_\_\_

Goal-directed search

- modify edge weights with potential functions.

$$\bar{w}(u, v) = w(u, v) - \lambda(u) + \lambda(v)$$



we wish to increase  
the potential value of  $(s, v_1)$   
(like going upwards)  
so that Dijkstra's algo  
chooses  $v_1$  over  $v_2$ .

$$\bar{w}(p) = w(p) - \lambda_t(s) + \lambda_t(t)$$

Landmark  $l \in V$ precompute  $s(u, l) \neq u \in V$ 

$$\lambda_t(u) = s(u, l) - s(t, l)$$