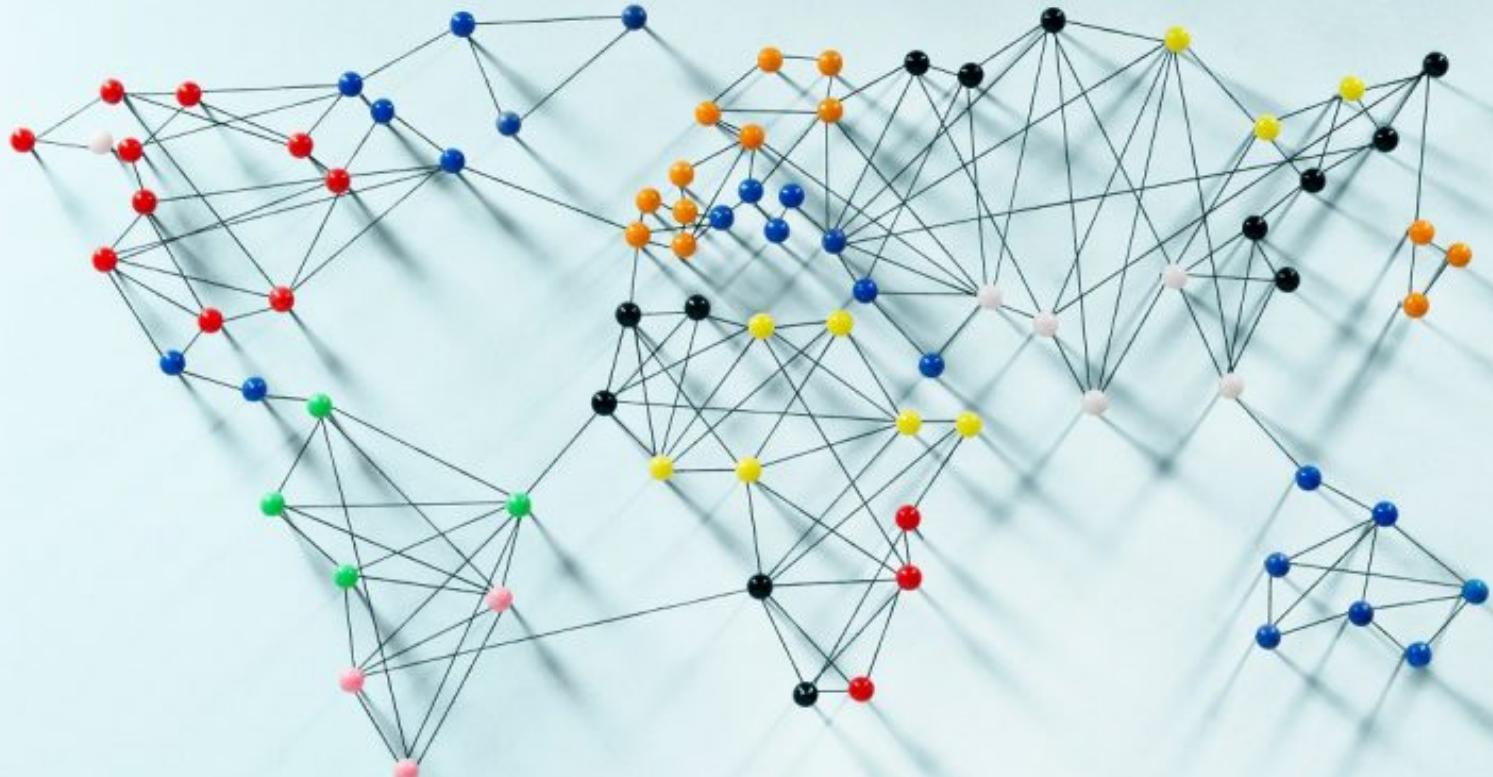


# Graph Theory and Algorithms

An Introduction to the world of Graphs



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

(Mentor: Tathagat Verma)

## Contents

<b>Preface</b>	<b>ii</b>
About the report . . . . .	ii
About the project and its future . . . . .	ii
 <b>MIT 6.042J/18.062J (Fall 2010) : Mathematics for Computer Science</b>	<b>I</b>
Lecture 6: Graph Theory and Coloring . . . . .	23
Lecture 7: Matching Problems . . . . .	30
Lecture 8: Graph Theory II: Minimum Spanning Trees . . . . .	38
Lecture 9: Communication Networks . . . . .	49
Lecture 10: Graph Theory III . . . . .	57
Lecture 11: Relations, Partial Orders, and Scheduling . . . . .	65
Part of Lecture 13: Asymptotics . . . . .	73
 <b>MIT 6.006 (Fall 2011) : Introduction to Algorithms</b>	<b>II</b>
Lecture 13: Breadth-First Search (BFS) . . . . .	95
Lecture 14: Depth-First Search (DFS), Topological Sort . . . . .	101
Lecture 15: Single-Source Shortest Paths Problem . . . . .	107
(Tutorial) Recitation 15: Shortest Paths . . . . .	111
Lecture 16: Dijkstra . . . . .	117
(Tutorial) Recitation 16: Rubik's Cube, StarCraft Zero . . . . .	123
Lecture 17: Bellman-Ford . . . . .	124
Lecture 18: Speeding up Dijkstra . . . . .	129
 <b>References</b>	<b>III</b>

P.S. The page numbers are depicted on the top-left of the image of the hand-written notes

## Preface

Before there were computers, there were algorithms. But now that there are computers, there are even more algorithms, and algorithms lie at the heart of computing

In the domain of mathematics and computer science, graph theory is the study of graphs that concerns the relationship among edges and vertices.

In recent years, graph theory has established itself as an important mathematical tool in a wide variety of subjects, ranging from operational research and chemistry to genetics and linguistics, and from electrical engineering and geography to sociology and architecture. At the same time it has also emerged as a worthwhile mathematical discipline in its own right.

### About the report

This work is written as a part of the Summer Of Science, 2020 project organized by the MnP Club, IIT Bombay. (Mentor: Tathagat Verma)

I have followed the course MIT 6.042J/18.062J and MIT 6.006 offered by MIT. The report consists mainly of my notes for these lectures.

All content here is highly inspired by the courses and at several points might simply be a paraphrasing of the course content. Nevertheless, I have tried to compress the contents of the course, skipping a few of the details, while still preserving sufficient depth.

Prerequisites for understanding the content:

- You should have some programming experience. In particular, you should understand recursive procedures and simple data structures such as arrays and linked lists.
- You should have some facility with mathematical proofs, and especially proofs by mathematical induction.

Though it may be hard to believe for a report of this size, space constraints prevented me from including many interesting algorithms and problems.

### About the project and its future

Even after completion of SoS 2020, I intend to learn further in this field.

I would be moving on to MIT 6.046J/18.410J (Design and Analysis of Algorithms) and MIT 6.854J/18.415J (Advanced Algorithms - a Graduate level course).

More courses in this context are MIT 6.045J/18.400J, MIT 18.404J/6.840J, MIT 6.079/6.975, MIT 18.315.

## **MIT 6.042J/18.062J (Fall 2010) : Mathematics for Computer Science**

This course is available on OpenCourseWare MIT ([ocw.mit.edu](http://ocw.mit.edu)) as well as on Youtube channel MIT OpenCourseWare.

### **Course Instructor(s)**

Prof. Tom Leighton  
Dr. Marten van Dijk

### **Course Description**

This course covers elementary discrete mathematics for computer science and engineering. It emphasizes mathematical definitions and proofs as well as applicable methods. Topics include formal logic notation, proof methods; induction, well-ordering; sets, relations; elementary graph theory; integer congruences; asymptotic notation and growth of functions; permutations and combinations, counting principles; discrete probability. Further selected topics may also be covered, such as recursive definition and structural induction; state machines and invariants; recurrences; generating functions.

In this report, I have included notes of Lectures 6 thru 11 which cover 'Mathematical basics of Graph Theory'.

More detailed notes of the course are present [here](#)

## Lecture 6: Graph Theory and Coloring

(23)

classmate

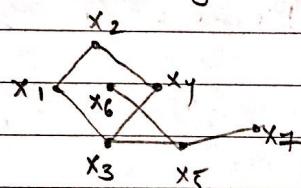
Date \_\_\_\_\_

Page \_\_\_\_\_

L-6

Graphs are incredibly 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  $x_1 - x_2$

$$\begin{matrix} x_1 & \text{---} & x_2 \\ | & & | \\ x_3 & & x_4 \\ | & & | \\ x_5 & \text{---} & x_6 \\ | & & | \\ x_7 & & x_8 \end{matrix}$$

$$E = \emptyset$$

## Lecture 6: Graph Theory and Coloring

24

classmate

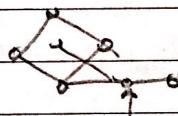
Date \_\_\_\_\_

Page \_\_\_\_\_

$\Rightarrow$  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.



$$\deg(x_5) = 3$$

$\Rightarrow$  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

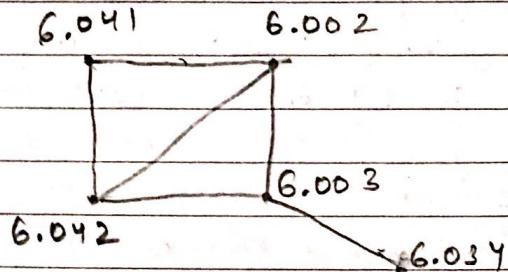
## Lecture 6: Graph Theory and Coloring

(25)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Slots

5-7

7-9

9-11

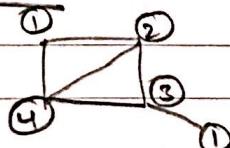
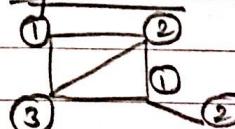
11-1

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  $\Rightarrow$  SlotsNodes  $\Rightarrow$  CoursesOption 1Option 2

$$\chi(G) = 3$$

2 colors can't work because of triangle formation

## Lecture 6: Graph Theory and Coloring

(26)

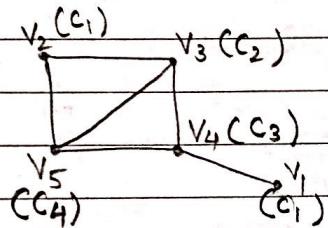
classmate

Date \_\_\_\_\_

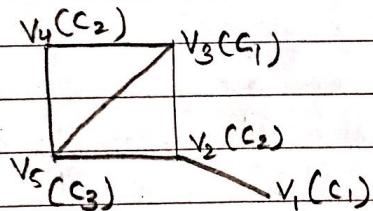
Page \_\_\_\_\_

Basic graph coloring algorithm

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



We know this is not best case.



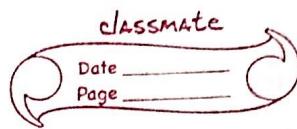
So, if you change the ordering, you may get a better answer.

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

## Lecture 6: Graph Theory and Coloring

(27)



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

Proof By induction,

Induction Hypothesis:

//Never 'induct on  $d$ ', induct 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 true

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

Let  $G$  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  $G'$ .

## Lecture 6: Graph Theory and Coloring

(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)



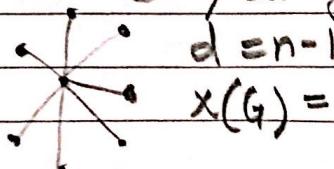
$$d = n/2$$

$$\chi(G) = 2$$

∴

Bipartite graph

(n) star graph



$$d = n-1$$

$$\chi(G) = 2$$

The Algorithm may be much better than the theorem

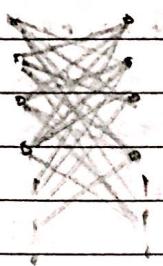
## Lecture 6: Graph Theory and Coloring

(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 ( $\lceil n/2 \rceil$  colors)

(C <sub>1</sub> ) v <sub>1</sub>	• v <sub>2</sub> (C <sub>1</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>3</sub> )
(C <sub>4</sub> ) v <sub>7</sub>	• v <sub>8</sub> (C <sub>4</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  $V_R$ .

## Lecture 7: Matching Problems

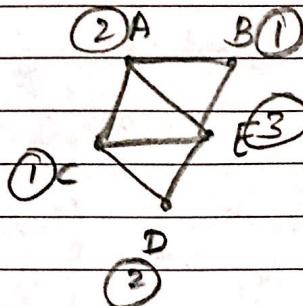
(30)

resource allocation problems (load balancing traffic on internet)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

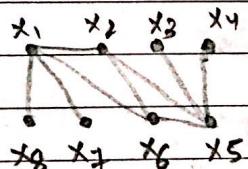


$$\chi(G) = 3$$

L-7

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

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

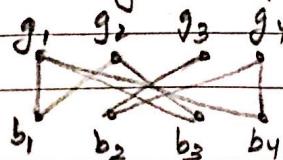


$\{x_1-x_6, x_2-x_5\}$  is a matching of size 2.

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

Size 4 matching is not possible in this graph.

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



$\{g_1-b_1, g_2-b_3, g_3-b_2$

$\} \text{ is a matching}$

## Lecture 7: Matching Problems

(31)

classmate

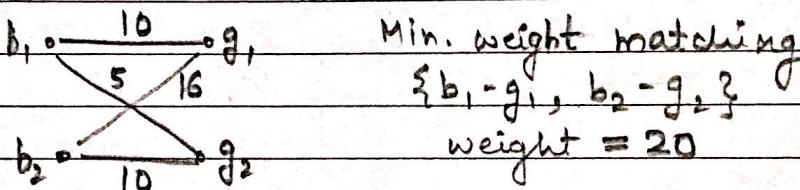
Date \_\_\_\_\_

Page \_\_\_\_\_

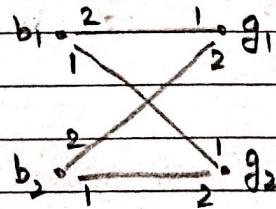
Weights?

$\Rightarrow$  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.

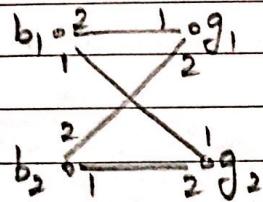
## Lecture 7: Matching Problems

(32)

classmate

Date \_\_\_\_\_

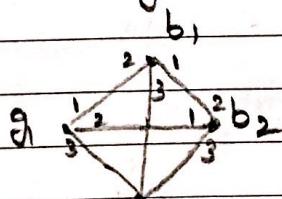
Page \_\_\_\_\_



$b_1-g_2 \& b_2-g_1$   
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  $\Delta$  is symmetric)

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

## Lecture 7: Matching Problems

(33)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

1962  
 David Lloyd  
Stable Marriage Algorithm (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,  
serenaders

Girls	Day 1	Day 2	Day 3	Day 4	
① CBEAD	A	2 4 5	5	5	A-5
② ABECD	B		2 1	2	B-2
③ DCBAE	C	1	4	4	C-4
④ ACDBE	D	3	3	3	D-3
⑤ ABDEC	E			1	E-1

## Lecture 7: Matching Problems

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  $N^2+1$  daysProof

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.

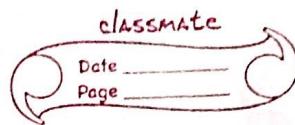
If the algo doesn't terminate in  $N^2+1$  days  
 then atleast  $N^2+1$  cross-outs have happened.

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

Thus, not possible.

## Lecture 7: Matching Problems

(35)



Predicate  $P =$  "If a girl  $G$  ever rejected a boy  $B$ , then  $G$  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

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

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

$\Rightarrow B$  is married.

## Lecture 7: Matching Problems

(36)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

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

Let B &amp; G be any pair that are not married.

To prove: B &amp; 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

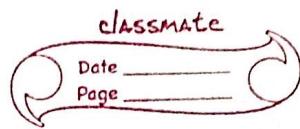
⇒ G is lower in order than B's spouse.

⇒ B is not rogue with G

∴ No rogue couples.

## Lecture 7: Matching Problems

37



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  $\{Q \mid \exists M \in S \{P, Q\} \in M\}$

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

## Lecture 8: Graph Theory II: Minimum Spanning Trees

38

## classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

- L-8

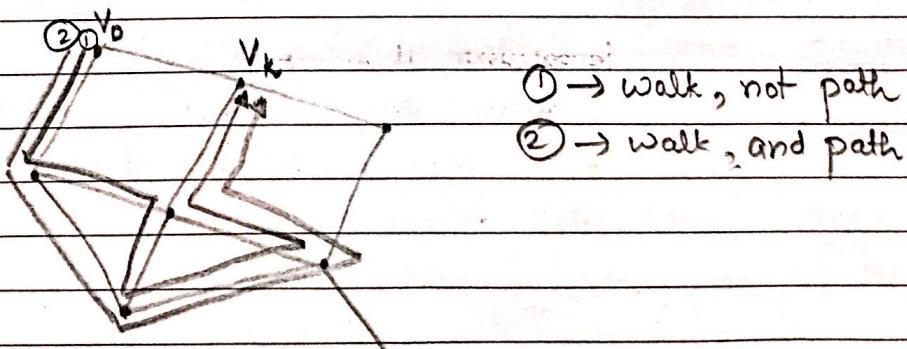
  - Walks & Paths
  - Connectivity
  - Cycles & Closed walk
  - Spanning Tree (ST)
  - Min-weight spanning tree (MST)

## 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 of length =  $k$



$\Rightarrow$  A path is a walk where all  $v_i$ 's are distinct

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(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.

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(40)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Connectivity

- ⇒  $u$  and  $v$  are connected if there is a path from  $u$  to  $v$ .
- ⇒ 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

- ⇒ A closed walk is a walk starts and ends at exactly same vertex.

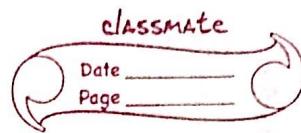
$$v_0 - v_1 - \dots - v_k = v_0$$

a closed walk with  
and

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

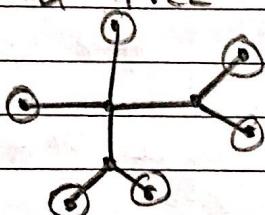
## Lecture 8: Graph Theory II: Minimum Spanning Trees

(41)

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. ■

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(42)

classmate \_\_\_\_\_

Date \_\_\_\_\_

Page \_\_\_\_\_

Lemma 3 A tree with  $n$  vertices has  $n-1$  edges.Proof By InductionP(n) Ind. Hypo. :  $P_n$  "There are  $n-1$  edges in an  $n$ -vertex tree"Base case:  $P(1)$  is true.  $\textcircled{1}$ Ind. step: Assume that  $P(n)$  is trueLet  $T$  be an  $(n+1)$ -vertex tree,Let  $v$  be a leaf of the treeDelete  $v$ ; 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

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(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.

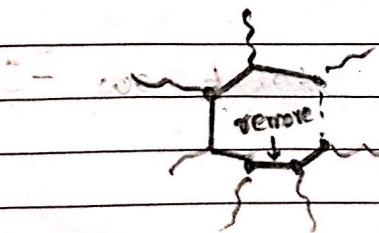
Theorem Every connected graph has a ST.

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.



if we remove one edge of the cycle, we still have a connected subgraph.

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(u4)

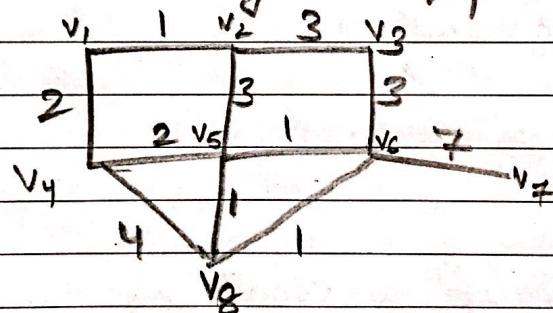
classmate

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



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

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

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

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

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(45)

classmate

Date \_\_\_\_\_

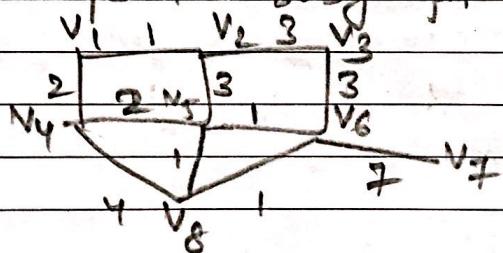
Page \_\_\_\_\_

$\Rightarrow$  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.



$$+1 \quad ① \quad V_6 - V_8$$

$$+1 \quad ② \quad V_1 - V_2$$

$$+1 \quad ③ \quad V_5 - V_6 \Rightarrow V_5 - V_8 \times$$

$$+2 \quad ④ \quad V_1 - V_4$$

$$+2 \quad ⑤ \quad V_4 - V_5 \Rightarrow V_4 - V_8 \times, V_2 - V_5 \times$$

$$+3 \quad ⑥ \quad V_2 - V_3 \Rightarrow V_3 - V_6 \times$$

$$+7 \quad ⑦ \quad V_6 - V_7$$

---



---

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(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 as 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:  $\mid P(m)$  holds

Let  $e$  denote the  $(m+1)$ th selected edge.

Let  $S$  denote the first  $m$  selected edges.

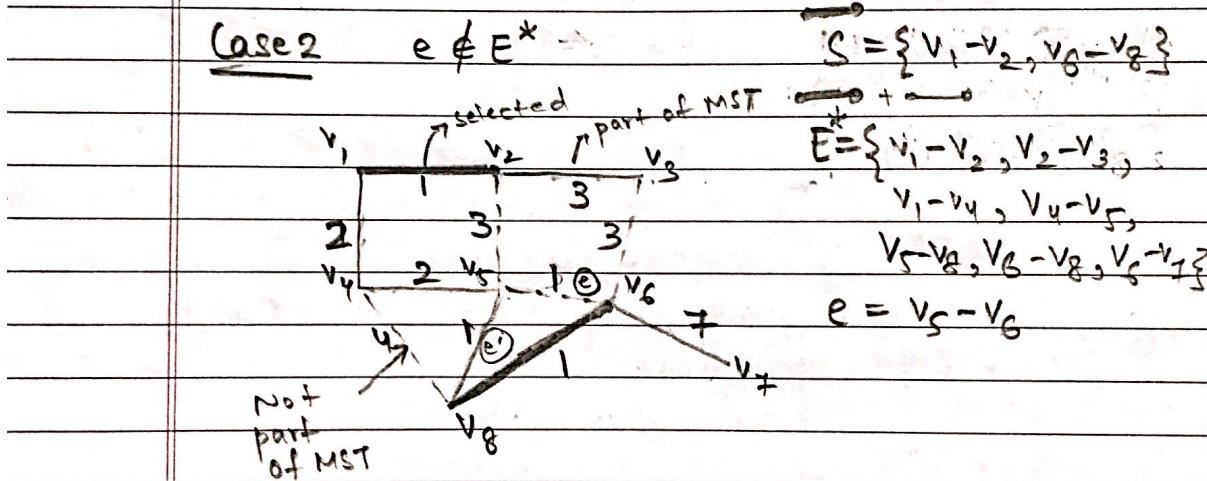
## Lecture 8: Graph Theory II: Minimum Spanning Trees



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

Case 2  $e \notin E^*$



$\hookrightarrow$  Aigo  $\Rightarrow S \cup \{e\}$  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)

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

## Lecture 8: Graph Theory II: Minimum Spanning Trees

(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  $G$ .

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

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.

## Lecture 9: Communication Networks

49

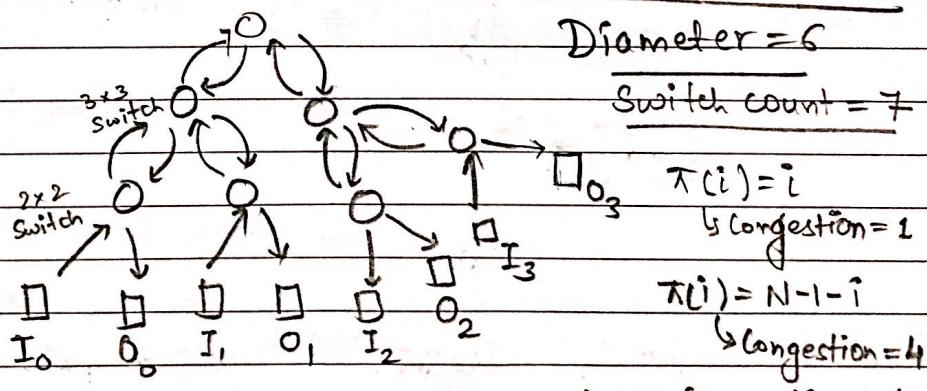
classmate \_\_\_\_\_

Date \_\_\_\_\_

Page \_\_\_\_\_

L-9Communication 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 =  $N$

## Lecture 9: Communication Networks

(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  $\pi$

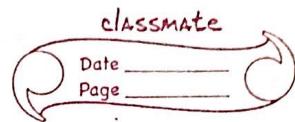
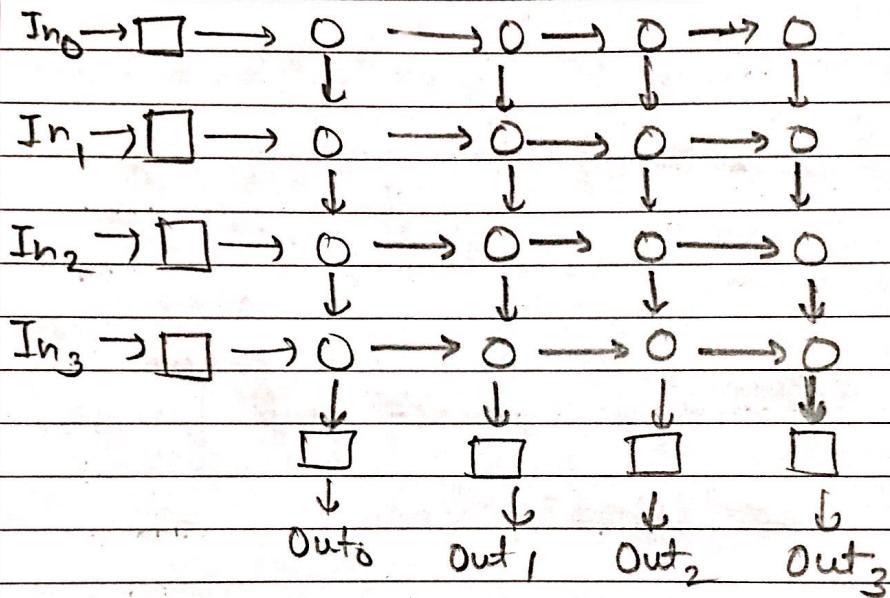
- For each  $i$  direct the packet at  $I_i$  to  $O_{\pi(i)}$   
path taken is denoted by  $P_{i, \pi(i)}$

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)

## Lecture 9: Communication Networks

(51)

2D Arrays (Grid structure)4x4 Network

$$\text{Diameter} = 8$$

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

$$\text{No. of switches} = 16$$

$$\text{Max. Congestion} = 2$$

N x N Network

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

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

$$\text{No. of switches} = N^2$$

$$\text{Max. Congestion} = 2$$

## Lecture 9: Communication Networks

(52)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

TheoremCongestion of an  $N$ -input array is 2.ProofLet  $\pi$  be a permutation. $P_{i,\pi(i)}$  = path from  $In_i$  rightward to column  $\pi(i)$  and downward to  $Out_{\pi(i)}$ .Switch in row  $i$  and column  $\pi(i)$  transmits almost two packets.For  $\pi(i)=i$ , Congestion = 2

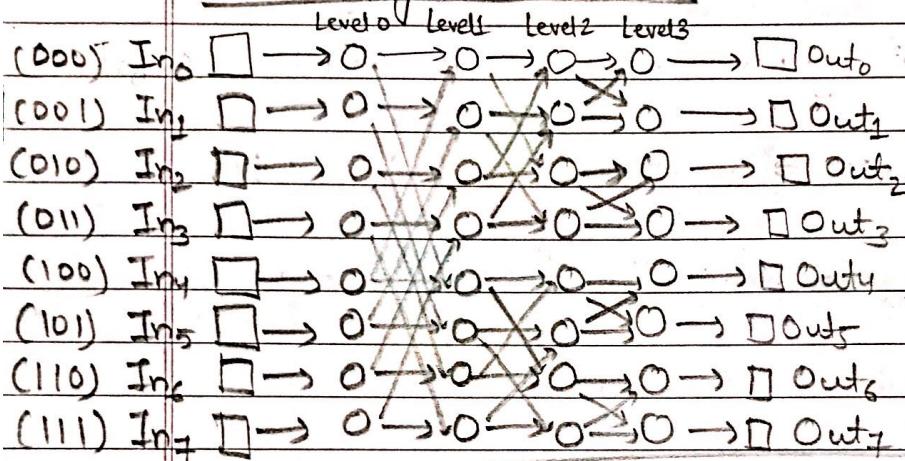
## Lecture 9: Communication Networks

53

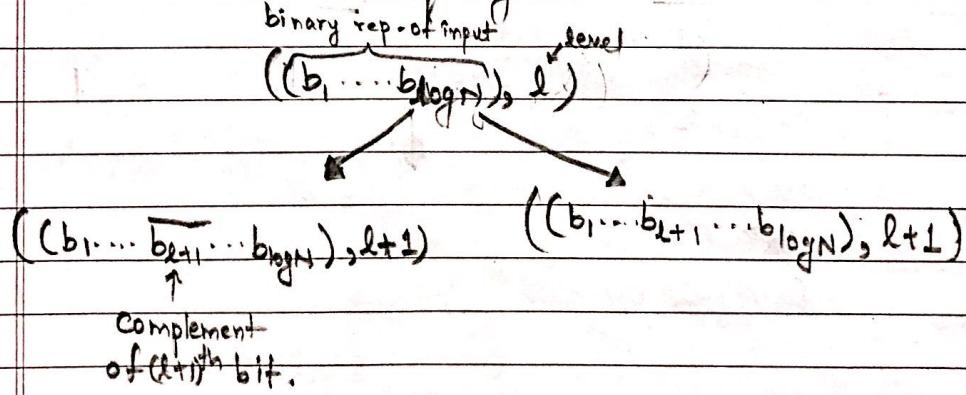
classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Butterfly Network

Switch is uniquely identified by its row & col



## Routing

$$\left( (x_1 \dots x_{\log N}), 0 \right) \rightarrow \left( (y_1 x_2 \dots x_{\log N}), 1 \right)$$

$$\left( (y_1 y_2 x_3 \dots x_{\log N}), 2 \right)$$

$$\left( (y_1 \dots y_{\log N}), \log N \right) \leftarrow \left( (y_1 \dots y_e x_{e+1} \dots x_{\log N}), l \right)$$

## Lecture 9: Communication Networks

(54)

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)NNN Network

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

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

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

$$\begin{aligned} \text{Congestion} &= \sqrt{N} (N \in 2^{2n}) \\ &= \sqrt{N/2} (N \in 2^{2n+1}) \end{aligned}$$

## Lecture 9: Communication Networks

(55)

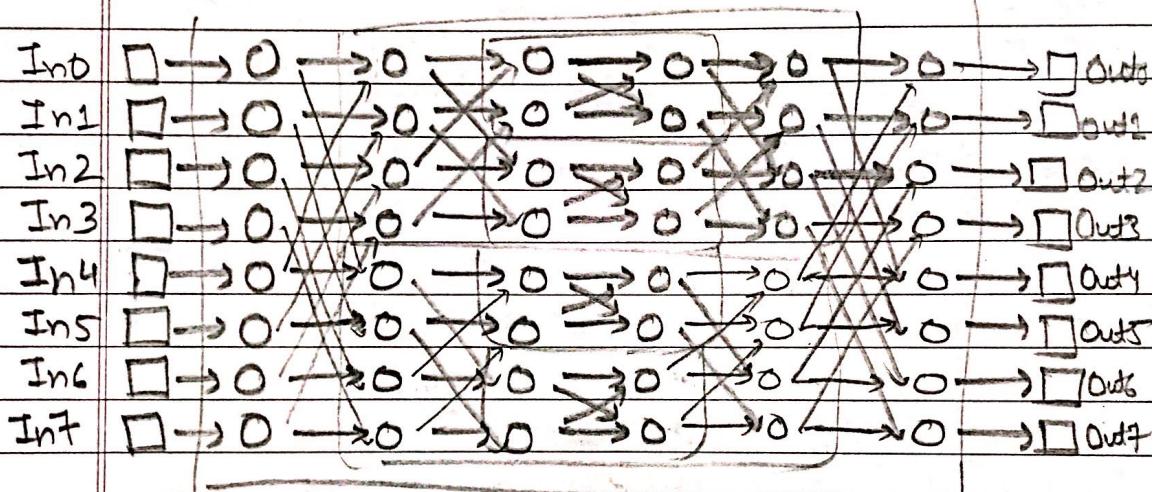
classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Benes

1960s - Benes, a Bell Labs researcher

NxN Network

$$\text{Diameter} = 1 + 2 \log N$$

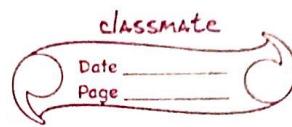
$$\text{Max switch size} = 2 \times 2$$

$$\text{No of switches} = 2N \log N$$

$$\text{Congestion} = 1$$

## Lecture 9: Communication Networks

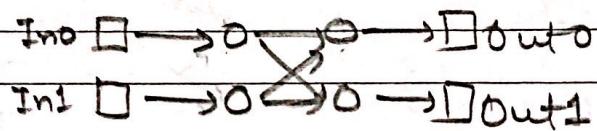
(56)



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

Proof By induction on  $a$ ,  
 $P(a) =$  "This theorem is true for  $a$ ".

Base case :  $a=1 N=2$



$$\pi(0) = 0$$

$$\pi(1) = 1$$

$$\text{Congestion} = 1$$

(Just straight)

$$\pi(0) = 1$$

$$\pi(1) = 0$$

$$\text{Congestion} = 1$$

(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.

## Lecture 10: Graph Theory III

(57)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-10

Euler tours

Directed graphs

- Definitions  $\rightarrow$  No. of walks
- Strong connectivity
- DAGs

Tournament Graphs

Euler tour

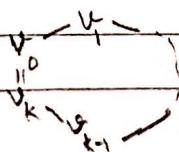
- he lived in Konigsberg
- Seven bridges (birth of graph theory)

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.

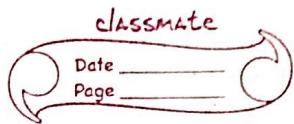
Proof ( $\Rightarrow$ )

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

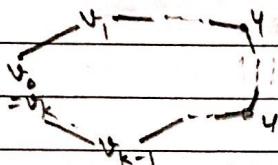


## Lecture 10: Graph Theory III

(58)



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

 $(\Leftarrow)$ 

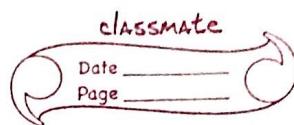
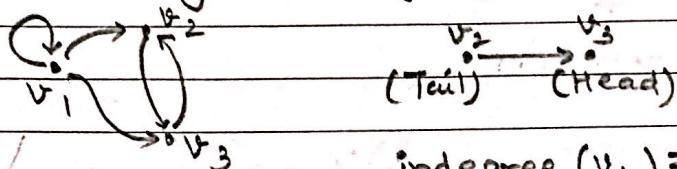
For  $G = (V, E)$ , assume  $\deg(v)$  is even  $\forall v \in V$   
Let  $W: v_0 - v_1 - \dots - v_k$  be the longest walk that traverses no edge more than once.

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  $\overset{\text{int}}{\wedge}$  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  $\overset{\text{int}}{\wedge}$  of  $v_k$  are in  $W$ , so  $\deg(v_k)$  is odd in  $G$   
 $\Rightarrow$  Contradicts as  $\deg(v)$  is even  $\forall v \in V$ .

## Lecture 10: Graph Theory III

(59)

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  $v_j$ .

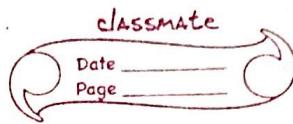
$$\text{then } A^k = \{p_{ij}^{(k)}\}$$

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

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

## Lecture 10: Graph Theory III

(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.

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

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

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

Ind. step  $\rightarrow$

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

Matrix multiplication holds

## Lecture 10: Graph Theory III

(61)

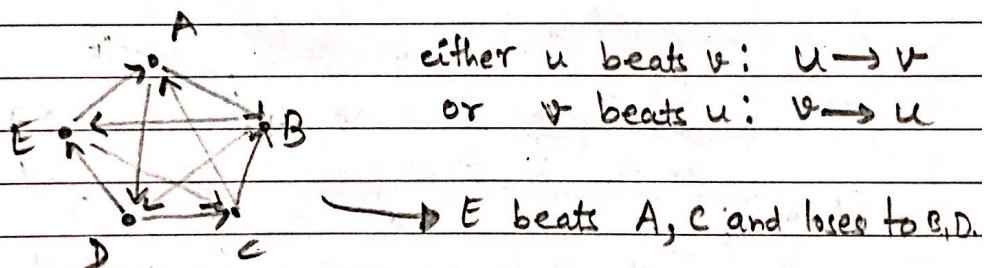
classmate

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.

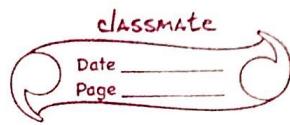
Tournament Graph

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

Theorem Every tournament graph contains a directed Hamiltonian path.

## Lecture 10: Graph Theory III

(G2)



Proof By induction (on number of nodes)

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

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

Approach 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

$v \rightarrow v_n$  then  $v_i \rightarrow v \neq i$  (by induction)

But  $v \rightarrow v_n$  contradicts this.

There is also proof by strong induction.  
See if.

## Lecture 10: Graph Theory III

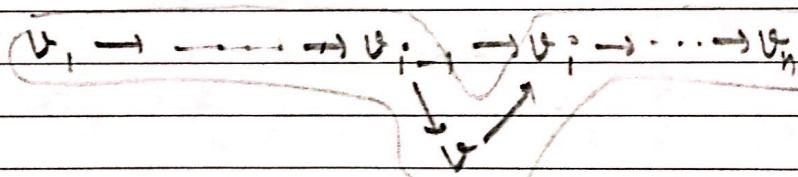
(63)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

APPROX ⑦

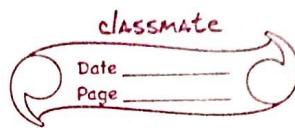
Case 1  $v \rightarrow v_i$ Case 2  $v_i \rightarrow v$ Smallest  $i$  such that  $v \rightarrow v_i$  $\Rightarrow v_k \rightarrow v$  &  $k < i$  $\Rightarrow v_{i-1} \rightarrow v$ 

directed  
A new Hamiltonian path  
is formed

$\therefore P(n+1)$  holds

## Lecture 10: Graph Theory III

(64)

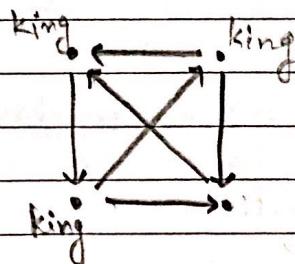
King Chicken Problem

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.



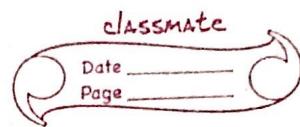
Theorem The chicken with highest outdegree is definitely a king.

Proof By contradiction,

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

## Lecture 11: Relations, Partial Orders, and Scheduling

(65)



$$\Rightarrow \exists_v : v \rightarrow u, \text{ and } \forall_w : u \rightarrow w \Rightarrow v \rightarrow w$$

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

So, our assumption is false.

L-11

- Relations
- Properties
- Equivalent Relations
- Partial Orders
  - Hasse diagrams
  - 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  $R \subseteq A \times B$

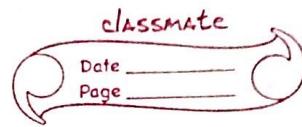
Ex :-  $R = \{(a, b) : \text{student } a \text{ is taking class } b\}$

$(a, b) \in R : a R b, a \sim R b$

↑  
Relational  
Symbol

## Lecture 11: Relations, Partial Orders, and Scheduling

(66)



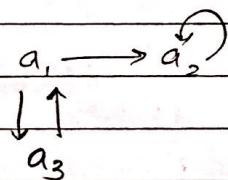
$\Rightarrow$  Relation on A is a subset  $R \subseteq A \times A$

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

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

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

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  $xRy \wedge yRz \Rightarrow xRz$

## Lecture 11: Relations, Partial Orders, and Scheduling

(67)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

<u>Ex</u>	Reflexive	Symmetric	Anti-Symm	Transitive	
$x \equiv y \pmod{5}$	✓	✓	X (2, 7)	✓	→ Equivalence
$x \mid y$	✓	X (1, 2)	✓	✓	→ Partial orders
$x \leq y$	✓	X (1, 2)	✓	✓	

⇒ An equivalence relation is reflexive, symmetric, transitive.

Ex:  $A = \mathbb{Z} : x R y \text{ iff } x = y \quad n \in \mathbb{N}$   
 $A = \mathbb{Z} : x R y \text{ iff } x \equiv y \pmod{n}$

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

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

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

## Lecture 11: Relations, Partial Orders, and Scheduling

(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.

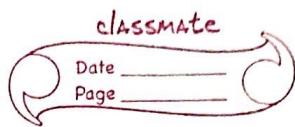
Partial Orders

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).

## Lecture 11: Relations, Partial Orders, and Scheduling

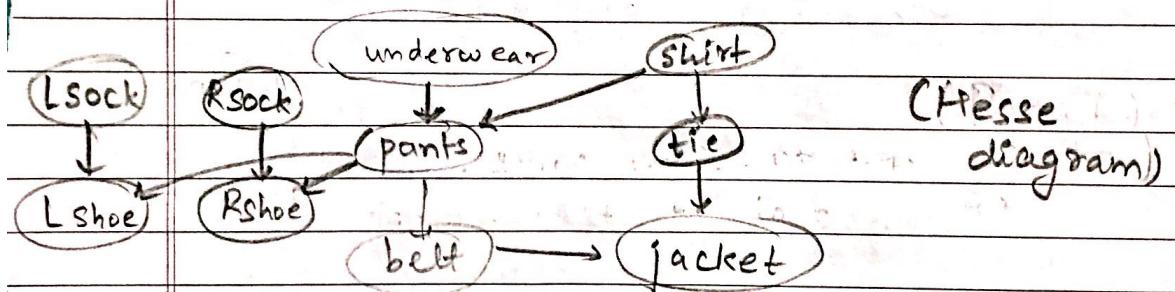
(69)



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

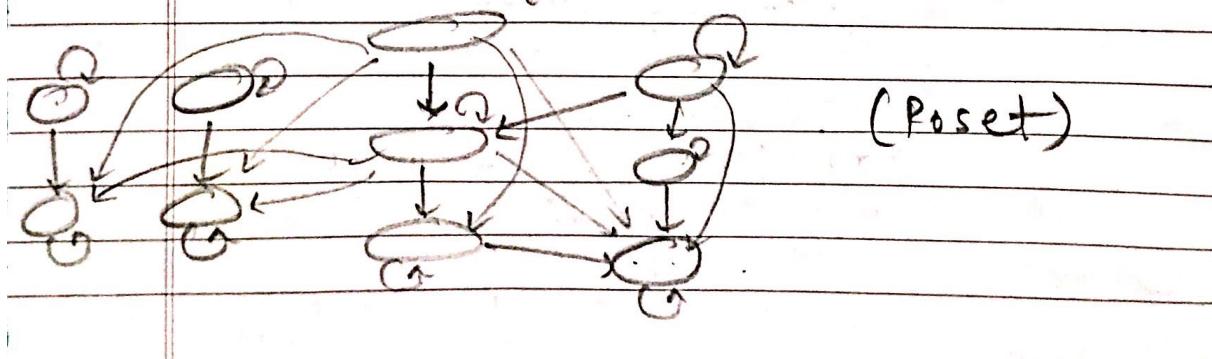
$(A, \leq)$  is actually called partial ordered set, or poset

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



## Lecture 11: Relations, Partial Orders, and Scheduling

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

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

(induction)

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

(1)

By anti-symmetry

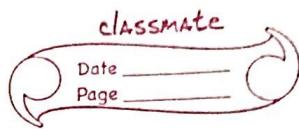
$\downarrow$   
anti-symmetry

$$a_1 = a_n$$

Contradicts distinct element assumption. ■

## Lecture 11: Relations, Partial Orders, and Scheduling

(II)

Conclusion of the theorem

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

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

$\Rightarrow$  a and b are comparable if  $a \leq b$  or  $b \leq a$ .

$\Rightarrow$  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.

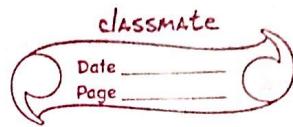
A topological of a poset  $(A, \leq)$  is a total order  $(A, \leq_T)$  such that

$$\leq \subseteq \leq_T$$

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

## Lecture 11: Relations, Partial Orders, and Scheduling

(72)



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  $C = (a_1, \leq a_2, \dots, \leq a_n)$  be the max length chain.

case1:  $a \notin \{a_1, \dots, a_n\}$   
 if  $a \leq a_1$ , then  $C$  is not the longest chain.

So,  $\exists a \not\leq a_1$

## Part of Lecture 13: Asymptotics

73

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Case 2:  $a \in \{a_1, \dots, a_n\}$ 

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

So,  $\forall a \neq a_i$ , $\therefore \nexists a \in A \setminus \{a_i\}$ ,So, by definition,  $a_i$  is the  
 minimum element.

Add Parallel Task Scheduling

&amp; Dilworth's Lemma.

L-13

Asymptotic Notation

(45:30)

tilde  $f(x) \sim g(x)$  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  $g(x)$ .

## Part of Lecture 13: Asymptotics

(74)

classmate

Date

Page

62

Theorem

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)

## Part of Lecture 13: Asymptotics

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

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

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

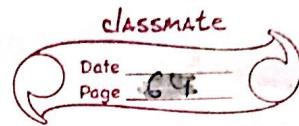
$$x^2 = \Omega(x)$$

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

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

## Part of Lecture 13: Asymptotics

(76)



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  $n$ .

$$O \Rightarrow \leq$$

$$O \Rightarrow \geq$$

$$\Theta \Rightarrow =$$

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

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

## Part of Lecture 13: Asymptotics

77

classmate

Date

Page

65

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

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

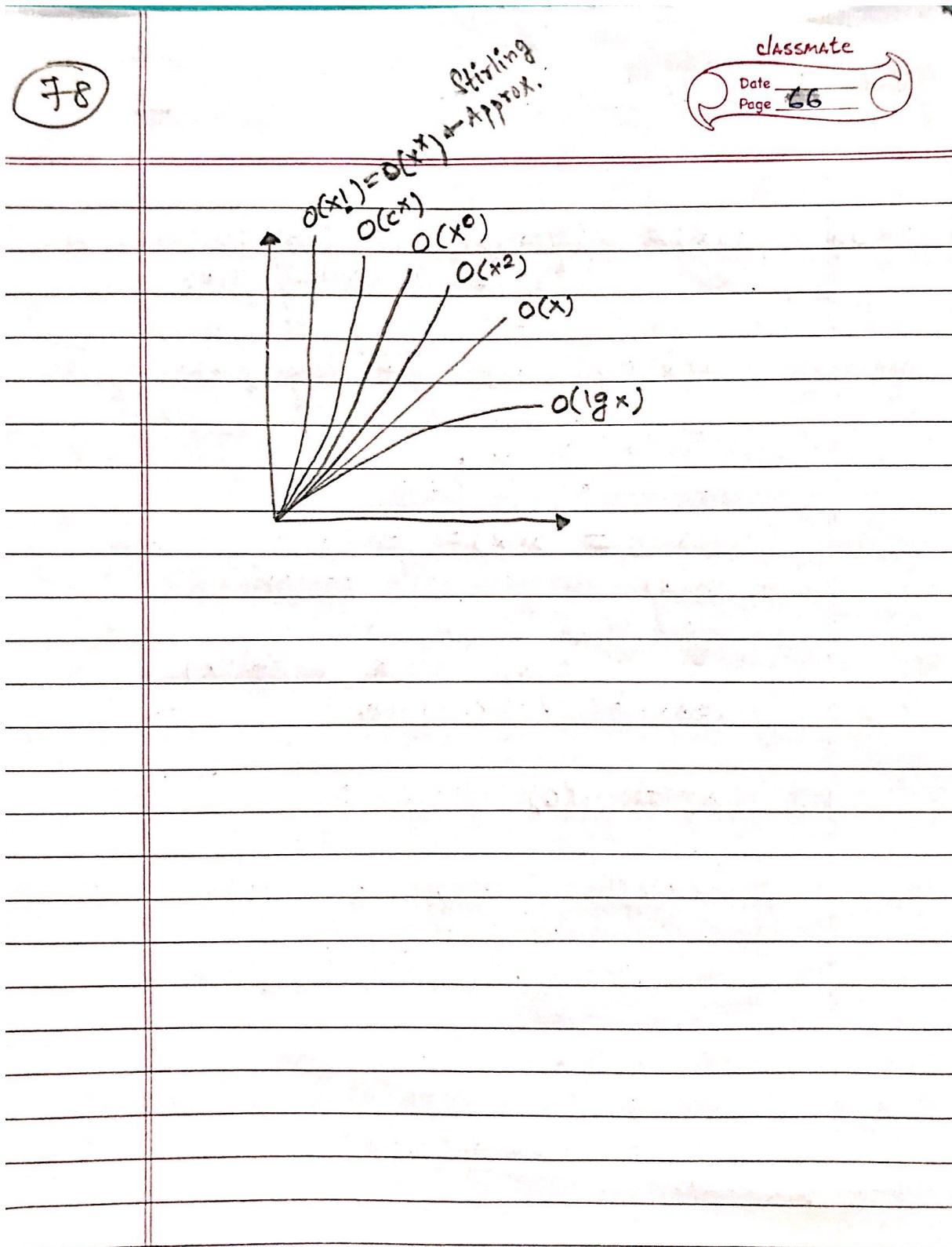
$$\frac{x}{\ln(x)} \in o(x)$$

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

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

Followed by : " what not to do example." clip.

## Part of Lecture 13: Asymptotics



## MIT 6.006 (Fall 2011) : Introduction to Algorithms

This course is available on OpenCourseWare MIT ([ocw.mit.edu](http://ocw.mit.edu)) as well as on Youtube channel MIT OpenCourseWare.

### Course Instructor(s)

Prof. Erik Demaine  
Prof. Srini Devadas

### Course Description

This course provides an introduction to mathematical modeling of computational problems. It covers the common algorithms, algorithmic paradigms, and data structures used to solve these problems. The course emphasizes the relationship between algorithms and programming, and introduces basic performance measures and analysis techniques for these problems.

In this report, I have included my notes of Lectures 13 thru 18 which cover 'Basic algorithms of Graph Theory'.

More detailed notes of the course are present [here](#)

## Lecture 13: Breadth-First Search (BFS)

95

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-13

### Graph 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  $\{u, v\} \rightarrow (u, v), (v, u)$

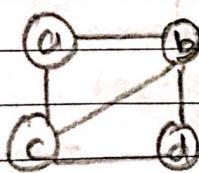
↓  
set of vertices

↓  
set of edges

undirected,  $e = \{u, v\}$  ← unordered pairs

graph OR

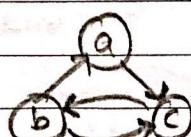
Directed →  $e = (u, v)$  ← ordered pairs



UNDIRECTED

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

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



DIRECTED

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

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

## Lecture 13: Breadth-First Search (BFS)

(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

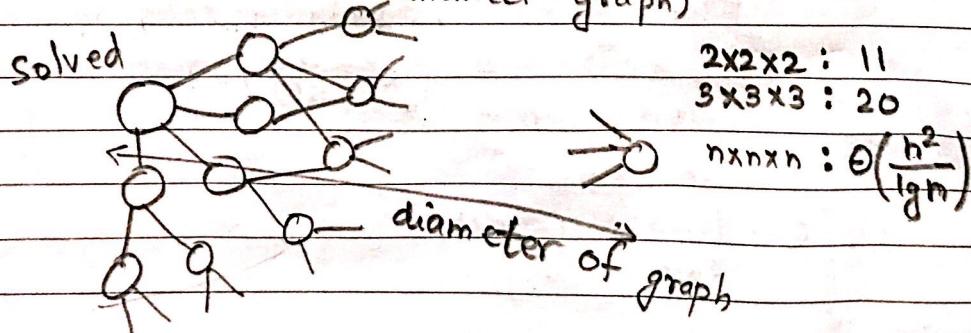
- vertex for each possible state of cube

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

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)

## Lecture 13: Breadth-First Search (BFS)

(97)

classmate

Date \_\_\_\_\_

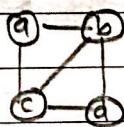
Page \_\_\_\_\_

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

$$\hookrightarrow \{v \in V \mid (u, v) \in E\}$$

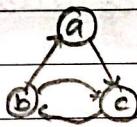


$$\text{Adj}[a] = \{b, c\}$$

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

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

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



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

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

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

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

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.

## Lecture 13: Breadth-First Search (BFS)

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

## Lecture 13: Breadth-First Search (BFS)

(99)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

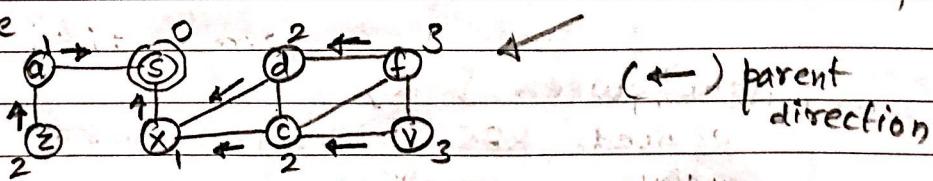
BFS( $s$ , Adj) $\text{level} = \{s: 0\}$  $\text{parent} = \{s: \text{None}\}$  $i=1$  $\text{frontier} = [s] \leftarrow \text{level } i-1$ 

while frontier:

 $\text{next} = [] \leftarrow \text{level } i$ for  $u$  in frontier:for  $v$  in  $\text{Adj}[u]$ :if  $v$  not in level: $\text{level}[v] = i$  $\text{parent}[v] = u$  $\text{next.append}(v)$  $\text{frontier} = \text{next}$  $i += 1$ 

UNDIRECTED Graph

Example

 $i=1 \quad \text{next} = [a, x]$  $i=2 \quad \text{next} = [z, d, c]$  $i=3 \quad \text{next} = [f, v]$  $i=4 \quad \text{next} = [e]$

## Lecture 13: Breadth-First Search (BFS)

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)  
(UDG)

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

R-13

(Halloween Day)

Revised BFS

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

## Lecture 14: Depth-First Search (DFS), Topological Sort

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: None}

DFS-Visit(V, Adj, s):

```

for v in Adj[s]:
    if v not in parent:
        parent[v] = s
        DFS-Visit(v, Adj, v)
    
```

DFS(V, Adj):

```

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

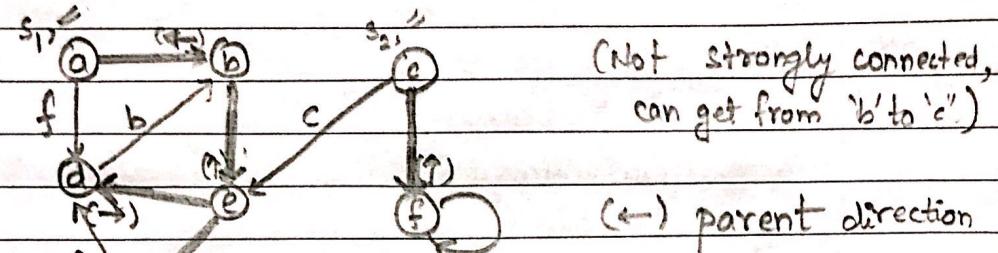
## Lecture 14: Depth-First Search (DFS), Topological Sort

102

CLASSMATE

Date \_\_\_\_\_

Page \_\_\_\_\_

Running time

$$O(V+E) \quad (\text{Linear time})$$

- visit each vertex once in DFS alone  $O(V)$
- DFS-Visit( $\dots, \dots, v$ ) called atmost once per vertex  $v$ .

$\hookrightarrow$  pay  $|Adj[v]|$

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

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

## Lecture 14: Depth-First Search (DFS), Topological Sort

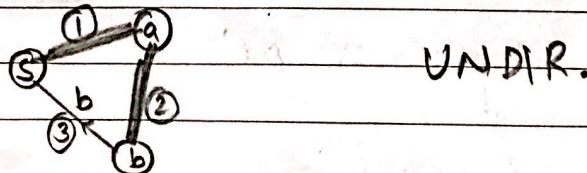
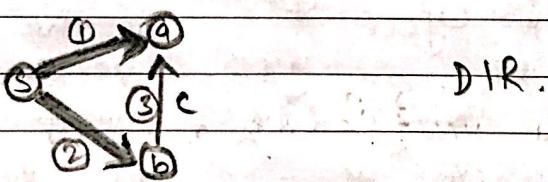
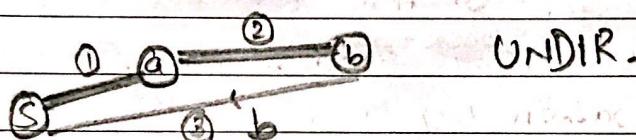
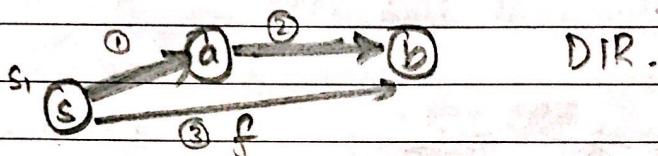
(03)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

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



Useful for cycle detection  
and topological sort.

## Lecture 14: Depth-First Search (DFS), Topological Sort

104.

classmate

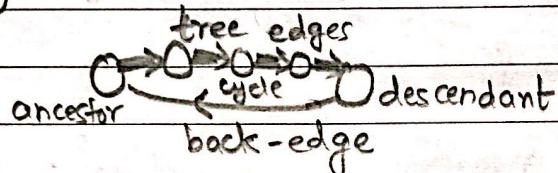
Date \_\_\_\_\_

Page \_\_\_\_\_

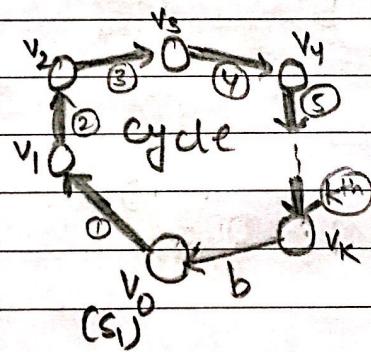
Cycle detection

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

## Lecture 14: Depth-First Search (DFS), Topological Sort

105

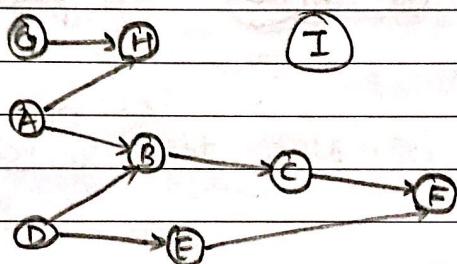
classmate \_\_\_\_\_

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  $v$

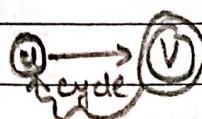
$\text{visit } u \rightarrow \text{visit } v \Rightarrow \text{visit } v \text{ before } u \text{ finishes}$

$\text{visit } u \rightarrow \text{start } u \rightarrow \text{visit } v \rightarrow \text{start } v \rightarrow \text{finish } v \rightarrow \text{finish } u$

## Lecture 14: Depth-First Search (DFS), Topological Sort

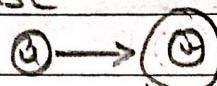
106

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_Case 2:  $v$  starts before  $u$ 

# contradiction

else



No path from  $v$  to  $u$   
 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.

## Lecture 15: Single-Source Shortest Paths Problem

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
  - $O(VE)$

## Lecture 15: Single-Source Shortest Paths Problem

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

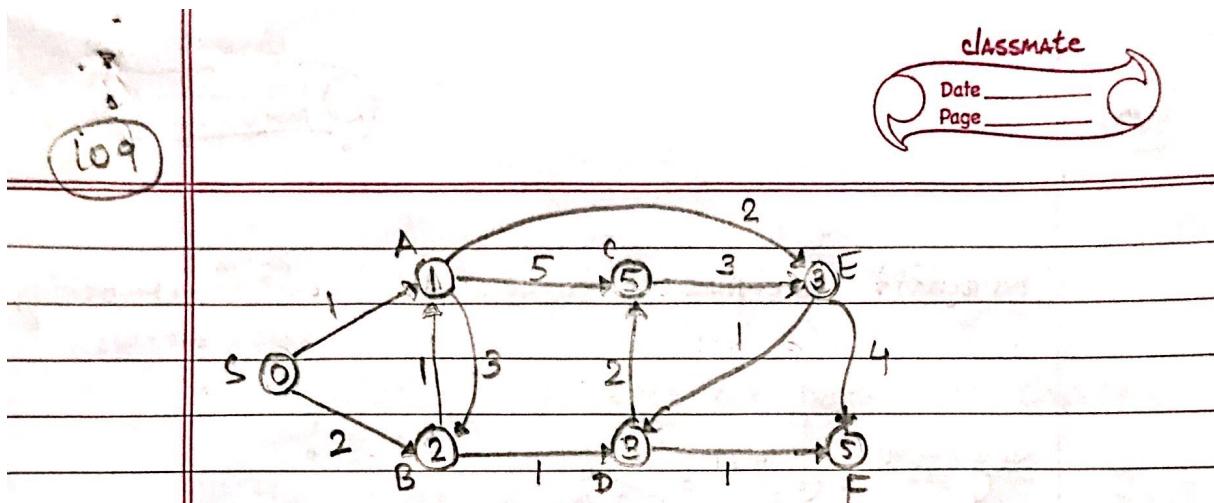
Weighted graphs

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

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

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

## Lecture 15: Single-Source Shortest Paths Problem

 $d(u)$ : current weight $\pi(u)$ : predecessor

	S	A	B	C	D	E	F
From S	0	<u>1</u>	<u>2</u>	<u><math>\infty</math></u>	<u><math>\infty</math></u>	<u><math>\infty</math></u>	<u><math>\infty</math></u>
From A	0	1	2( <u>4</u> )	<u>6</u>	<u><math>\infty</math></u>	<u>3</u>	<u><math>\infty</math></u>
From C	0	1	2	6	<u><math>\infty</math></u>	3( <u>9</u> )	<u><math>\infty</math></u>
From E	0	1	2	6	4	3	<u>7</u>
From D	0	1	2	6	4	3	<u>5</u> ( <u>7</u> )
From F	0	1	2	6	4	3	5
From B	0	1( <u>3</u> )	2	6	3( <u>4</u> )	3	5
From D	0	1	2	<u>5</u> ( <u>6</u> )	4	3	5
From C	0	1	2	5	4	3	5

$\delta(S, A) = 1$

$\delta(S, B) = 2$

$\delta(S, C) = 5$

$\delta(S, D) = 4$

$\delta(S, E) = 3$

$\delta(S, F) = 5$

## Lecture 15: Single-Source Shortest Paths Problem

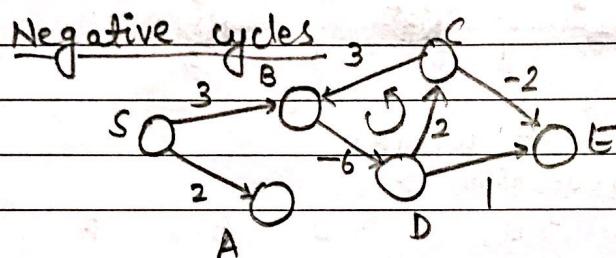
110

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

Negative weights : reverse tolls, 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  $d[v] > d[u] + w(u,v)$   
 $d[v] = d[u] + w(u,v)$   
 $\pi[v] \leftarrow u$

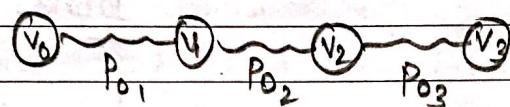
Until all edges have  $d[v] \leq d[u] + w(u,v)$

## (Tutorial) Recitation 15: Shortest Paths

Optimal substructure

- Subpaths of a shortest path are shortest paths

$$\{P_{0_1}, P_{0_2}, P_{0_3}\} = SP$$

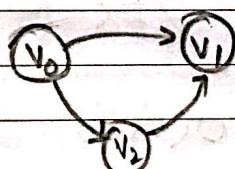


$$\text{then } \{P_{0_1}\} = SP$$

$$\{P_{0_2}\} = SP$$

$$\{P_{0_3}\} = SP$$

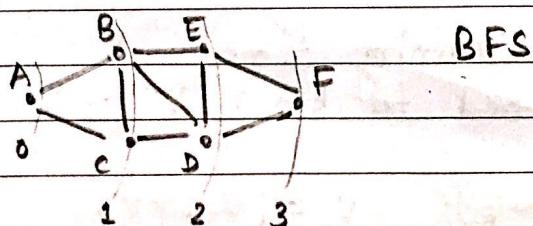
Triangle  
inequality  
used in i-16)



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

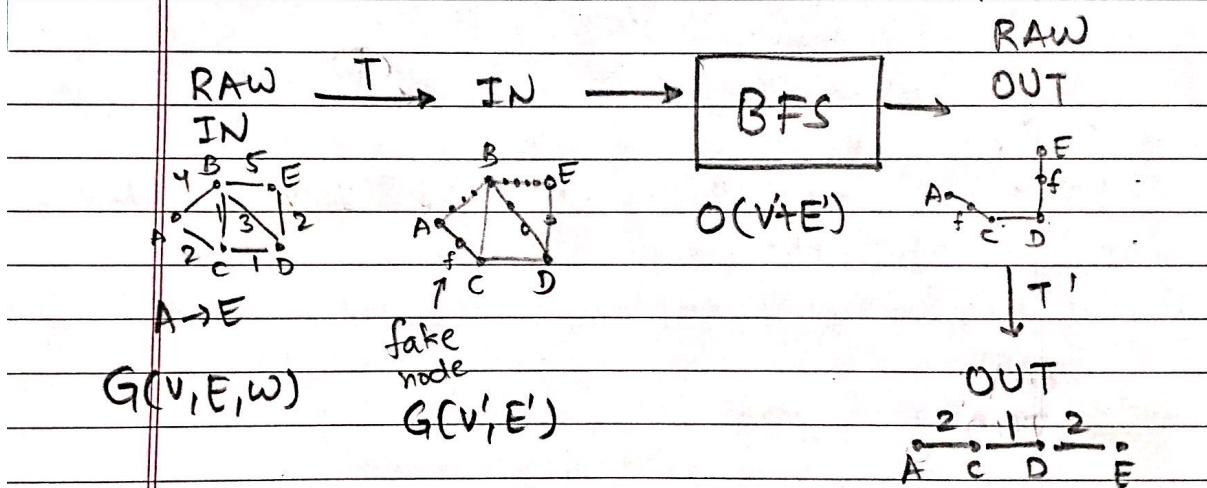
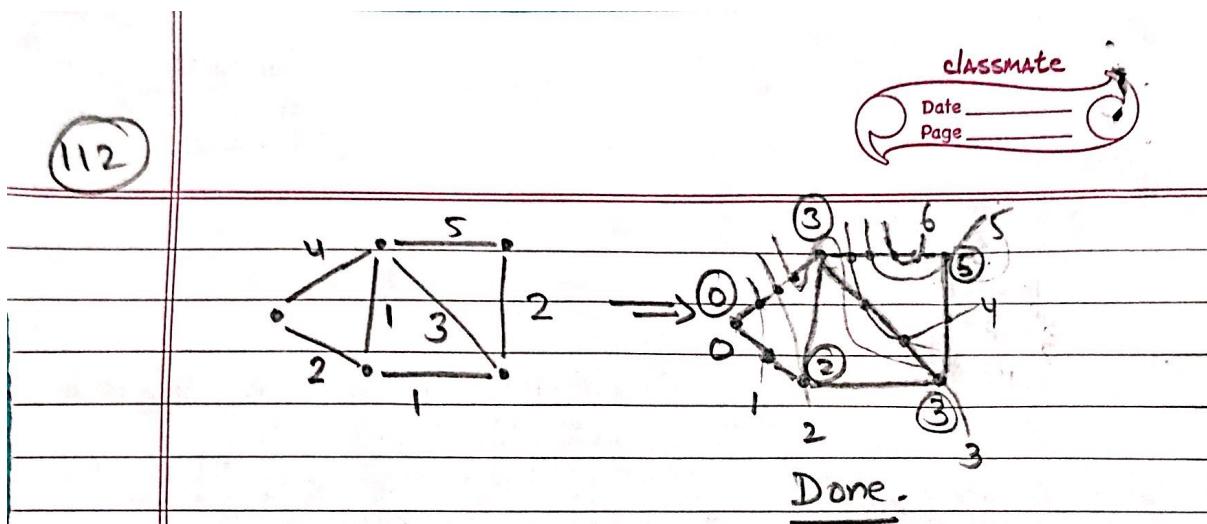
R-15Shortest path

Thinking of something simple.



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

## (Tutorial) Recitation 15: Shortest Paths



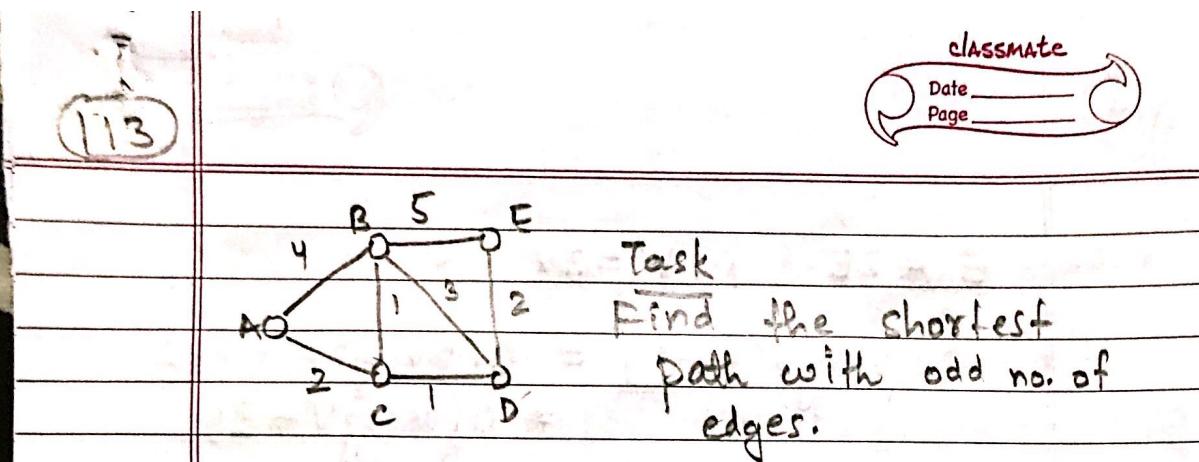
To find out running time,  
we need to know  $V', E'$ .

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

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

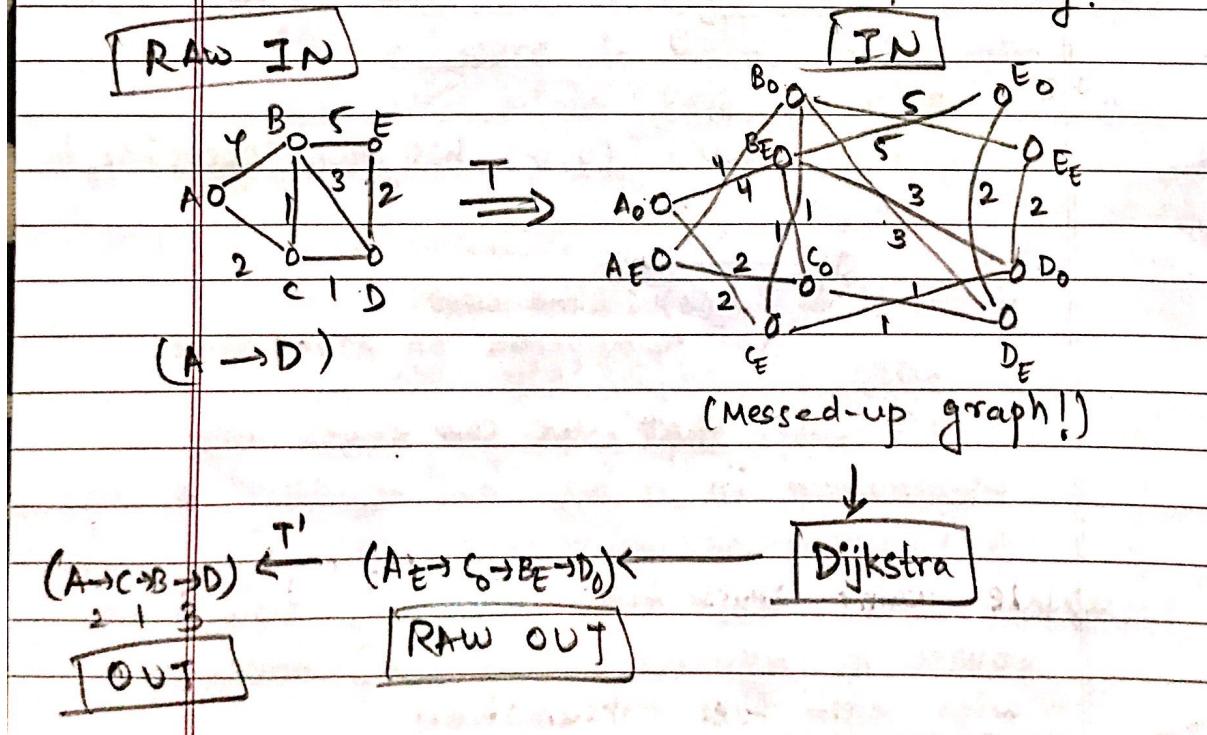
For small paths, this is a nice algo.

## (Tutorial) Recitation 15: Shortest Paths



Thought 1: Need a transform which stores the data of (# edges travelled % 2) (even or odd)

Thought 2: If you reach X in even #edges then  $\otimes \rightarrow Y$  you can reach Y in odd #edges, this way.



## (Tutorial) Recitation 15: Shortest Paths

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

114

$E' = 2E$        $V' = 2V$

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

A hard one (took me 20 min to understand)

Given DIR. Graph

```

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

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

$f_c$ : fuel cost

$t_c(s)$ : time cost  
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

## (Tutorial) Recitation 15: Shortest Paths

115

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

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

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

$$\begin{aligned}
 \text{Transform}(T) \\
 V &\xrightarrow{(x M)} N, +1 \\
 (u, v) &\xrightarrow{(x M)} ((u, t), (v, t+1), (u, v), t))
 \end{aligned}$$

fuel cost =  $f_c$

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

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

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

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

## (Tutorial) Recitation 15: Shortest Paths

116		classmate Date _____ Page _____
	$i=0 \quad \infty$	
	$i=1 \quad \infty$	
	$i=2 \quad \infty$	
	$i=3 \quad \infty$	
	$i=k-1 \quad \infty$	$\left. \begin{matrix} \text{No such path} \\ \rightarrow \text{eco-friendly fastest path.} \end{matrix} \right\}$
	$i=k \quad p$	
	$i=k+1$	
	$\vdots$	

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-16)

$$\text{Here, } V' = V \cdot M$$

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

Complexity of algo

$$= O(V' \lg V' + E')$$

$$= O(V \cdot M \lg(N \cdot M) + E \cdot M + V \cdot M)$$

$$= O(V \lg V + E)$$

(though, constant is much worse than Dijkstra)

## Lecture 16: Dijkstra

117

classmate

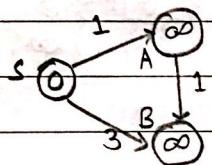
Date \_\_\_\_\_

Page \_\_\_\_\_

L-16

Shortest Paths II : Dijkstra

- Review
- Shortest paths - DAGs
- SPs - graphs w/o  $\rightarrow$  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  $v$ .

$$d[s] = 0, d[A] = 1, d[B] = 3$$

$\downarrow$

$\pi[A] = s, \pi[B] = s$

$\pi(v)$ : predecessor of  $v$  in the S.P. from  $s$  to  $v$ .

$$d[s] = 0, d[A] = 1, d[B] = 2$$

$\pi[A] = s, \pi[B] = A$

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

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

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

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

## Lecture 16: Dijkstra

(118)

classmate

Date \_\_\_\_\_  
Page \_\_\_\_\_ $\text{Relax}(u, v, w)$ :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.

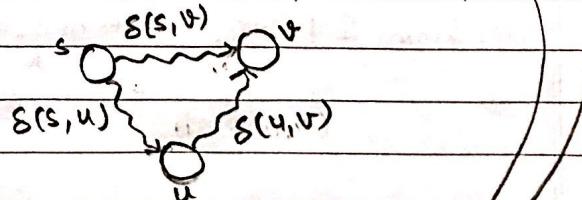
Base case:  $n=0$   $d[v] = \begin{cases} 0 & v=s \\ \infty & \text{otherwise} \end{cases}$  satisfies

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

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



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

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

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

 $\therefore \delta(s, v) \leq d[v]$  Proved.

## Lecture 16: Dijkstra

(119)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

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

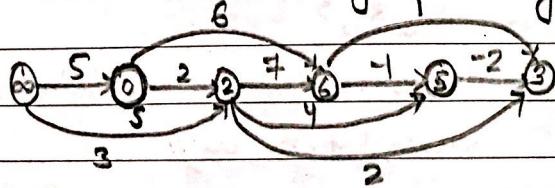
1. Topologically sort the DAG.

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       $\infty$     0     $\infty$      $\infty$      $\infty$      $\infty$ Step 1       $\infty$     0    2    6     $\infty$      $\infty$ Step 2       $\infty$     0    2    6( $\infty$ )    6    4Step 3       $\infty$     0    2    6    5( $\infty$ )    4( $\infty$ )Step 4       $\infty$     0    2    6    5    3( $\infty$ )Step 5       $\infty$     0    2    6    5    3

## Lecture 16: Dijkstra

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

$d[s] = 0$       Initialize ( $G, s$ )       $S' \leftarrow \emptyset$        $Q = V[G]$

while  $Q \neq \emptyset$

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

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

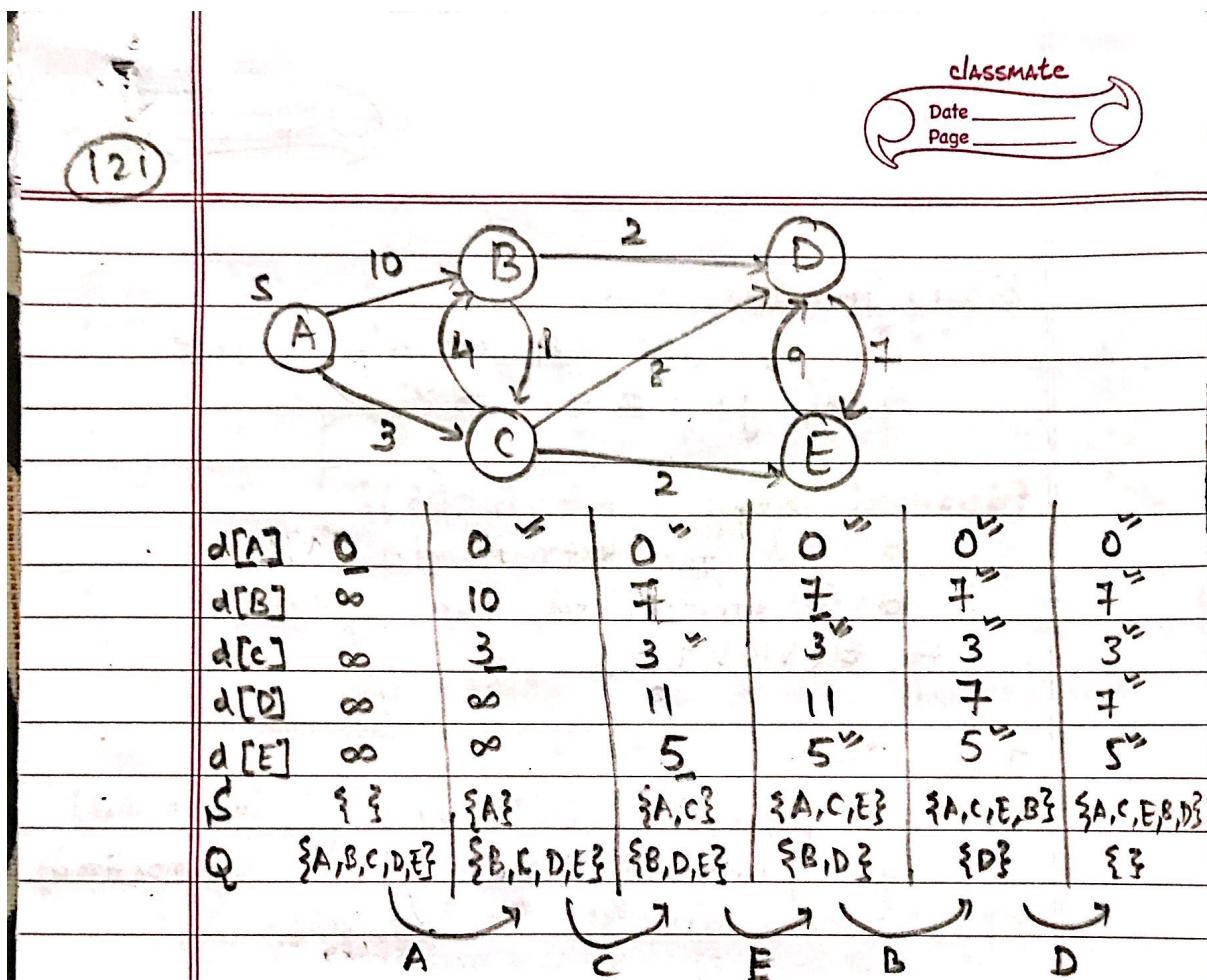
for each vertex  $v \in \text{Adj}[u]$

Relax ( $u, v, \omega$ )

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

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

## Lecture 16: Dijkstra

Complexity $\Theta(V)$  inserts into the priority-queue Q. $\Theta(V)$  extract-min ops. $\Theta(E)$  Decrease-key / update ops.Arrays:  $\Theta(1)$  for insert, update $\Theta(V)$  for extract-min

$$\rightarrow \Theta(V + V^2 + E) = \Theta(V^2) \quad (E = O(V^2))$$

## Lecture 16: Dijkstra

(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):

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

$$\Theta(1) \text{ amortized for update}$$

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

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

## (Tutorial) Recitation 16: Rubik's Cube, StarCraft Zero

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

(123)

R-16      2x2x2 Rubik's cube

The diagram shows two configurations of a 2x2x2 Rubik's cube. On the left, labeled "Plastic faces", the cube is shown with faces labeled yob, boy, oyb, and ybo. Arrows point from each face to its corresponding label. On the right, labeled "Wireframe", the cube is shown as a wireframe with faces labeled ufr, urf, rfy, and ruf. Arrows point from each face to its corresponding label. Between the two configurations, there are arrows pointing from one state to the other, indicating transitions.

8 cublets  
24 faces

Configuration is an array of 24

yob	boy	oyb	-	-	.	-	-
0	1	2	3	-	.	-	23
(fur)	(ruf)	(urf)					

Total configurations:  $24!$

Six types of moves:  $f_c, f_{cc}, r_c, r_{cc}, u_c, u_{cc}$

Finding Starcraft game strategy  
(Using Dijkstra)

## Lecture 17: Bellman-Ford

124

classmate

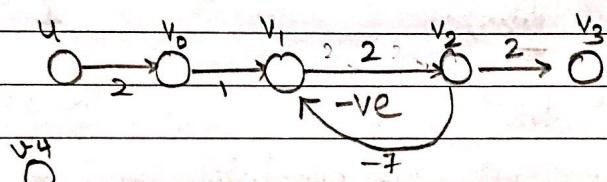
Date \_\_\_\_\_

Page \_\_\_\_\_

L-17

Shortest 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.

## Lecture 17: Bellman-Ford

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 cycle 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  $O(V^2V+E)$  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  $(u, v, w)$ 

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

 $O(E)$  Check

 $\left\{ \begin{array}{l} \text{for each edge } (u, v) \in E \\ \text{if } d[v] > d[u] + w(u, v) \end{array} \right.$ 

then report -ve cycle exists.

Complexity :  $O(VE)$

## Lecture 17: Bellman-Ford

126

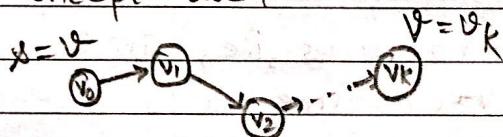
classmate

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.ProofLet  $v \in V$ 

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

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  $d[v_1] = \delta(s, v_1)$ because we will relax edge  $(v_0, v_1)$ (If not, then  $P$  is not a shortest path.)

## Lecture 17: Bellman-Ford

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

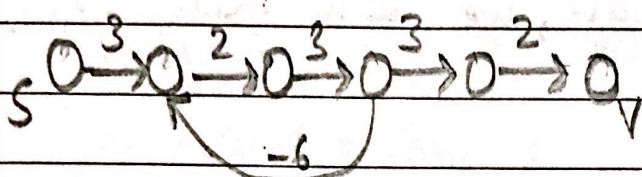
## Lecture 17: Bellman-Ford

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.

## Lecture 18: Speeding up Dijkstra

(129)

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

L-18 Shortest Paths IV : Speeding up Dijkstra

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

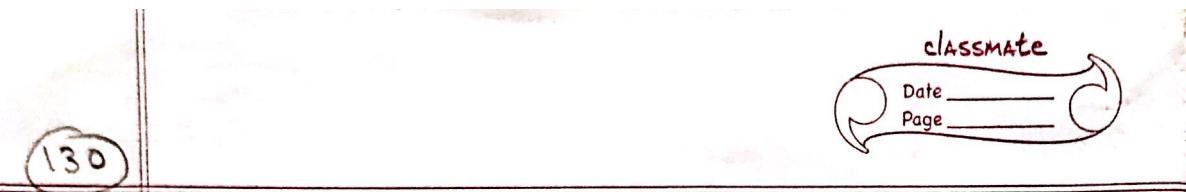
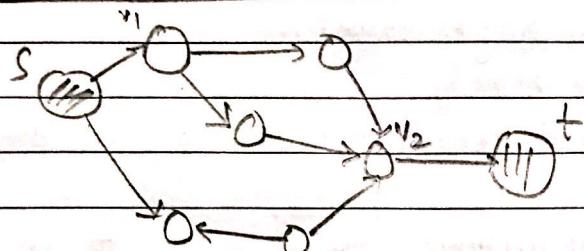
Unlike previous lectures, we're going to be talking about optimizations that don't change the worst-case, or asymptotic complexity, but improve empirical, real-life performance. (Performance in average case)

1.  $\text{Dijkstra}(G, W, s)$ :2. Initialize()  $\leftarrow d[s] = 0, d[u \neq s] = \infty$ 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 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.

## Lecture 18: Speeding up Dijkstra

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  $Q_f$  &  $Q_b$ .

## Lecture 18: Speeding up Dijkstra

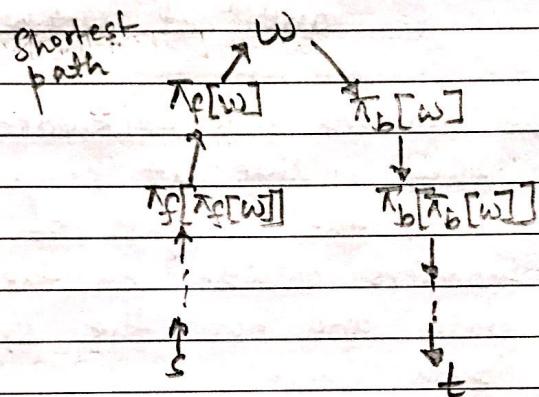
(131)

classmate

Date \_\_\_\_\_

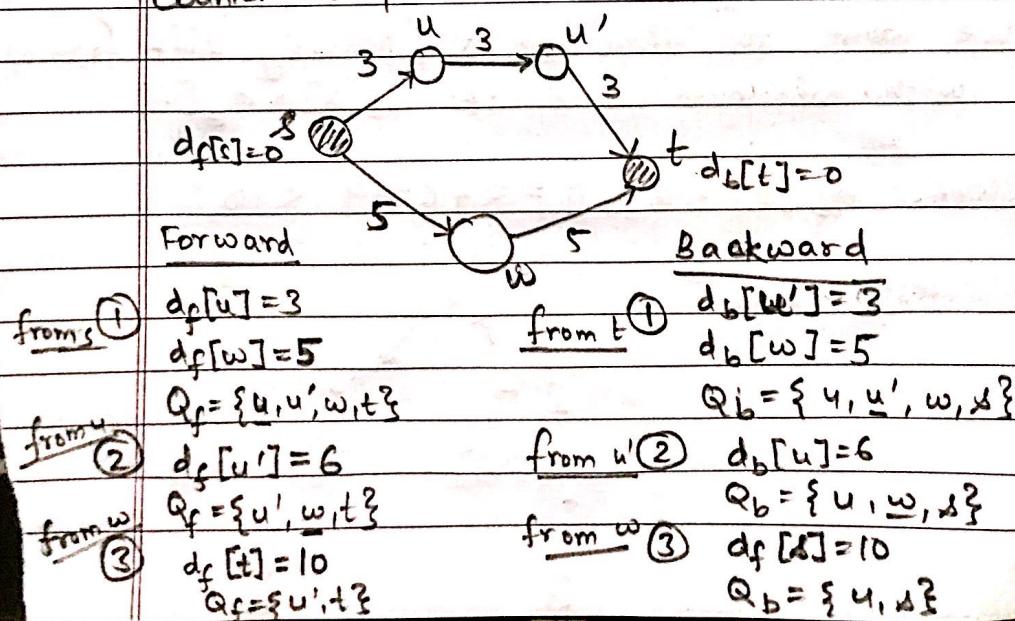
Page \_\_\_\_\_

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



This claim is actually false.

Counter-example:



## Lecture 18: Speeding up Dijkstra

(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  $s \rightarrow w \rightarrow t$ .

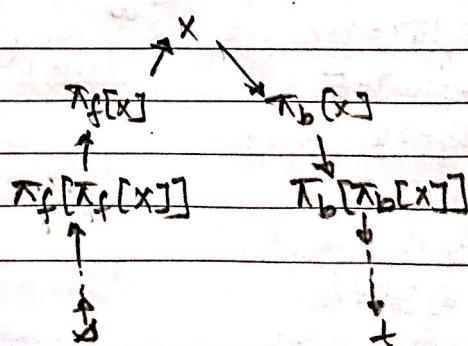
When the forward and backward frontiers collide, they collide at some vertex, regardless of the weights of the edges.

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:



## Lecture 18: Speeding up Dijkstra

133

classmate

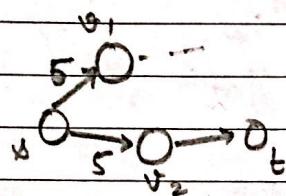
Date \_\_\_\_\_

Page \_\_\_\_\_

Goal-directed search

- modify edge weights with potential functions.

$$\bar{\omega}(v, v) = \omega(v, v) - \lambda(v) + \lambda(v')$$



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

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

Landmark  $l \in V$ 

precompute  $\delta(u, l) \forall u \in V$   
 $\lambda_t(u) = \delta(u, l) - \delta(t, l)$

(Part of 6.046J)

## References

### Site References:

- <https://www.geeksforgeeks.org/fundamentals-of-algorithms/>  
(For Basics of algorithms)
- <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>  
(For concepts of graph algorithms)
- <http://cp-algorithms.com/>  
(Problems on graphs algorithms)
- [https://www.youtube.com/watch?v=09\\_LlHjoEiY&t=7s](https://www.youtube.com/watch?v=09_LlHjoEiY&t=7s)  
(Summary of graph algorithms)

### Book References:

- Introduction to Algorithms - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein - Third Edition (Textbook for MIT 6.006)
- Introduction to Graph Theory (2nd Edition) - Douglas B. West (For Mathematical concepts of Graph Theory)
- GRAPH THEORY - Keijo Ruohonen (For Mathematical concepts of Graph Theory)
- Algorithm Design: Parallel and Sequential - Xiuquan Lv (Algorithm Design)
- Competitive Programmer's Handbook - Antti Laaksonen (For competitive programming)
- NUS CS3233 - Competitive Programming (For competitive programming)

### MIT Courses (which are related):

- MIT 6.042J/18.062J - Mathematics for Computer science
- MIT 6.006 - Introduction to Algorithms
- MIT 6.046J/18.410J - Design and Analysis of Algorithms
- MIT 6.854J/18.415J - Advanced Algorithms
- MIT 6.045J/18.400J - Automata, Computability, and Complexity
- MIT 18.404J/6.840J - Theory of Computation
- MIT 6.079/6.975 - Introduction to Convex Optimization
- MIT 18.315 - Combinatorial Theory: Introduction to Graph Theory, Extremal and Enumerative Combinatorics

I would be studying these courses as well and my notes I make would be available on Github (ID: devansh-dvj)