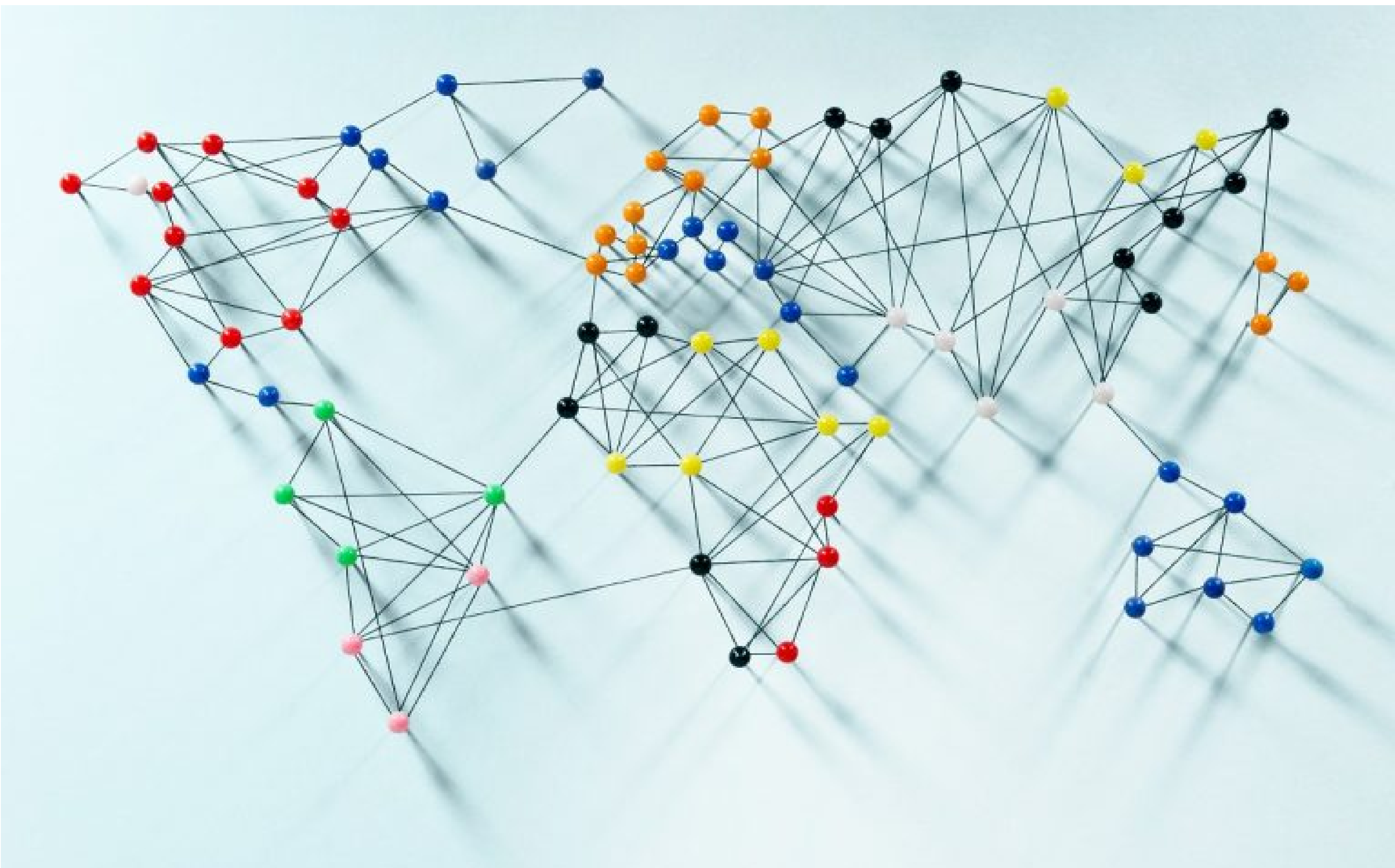
Graph Theory and Algorithms

An Introduction to the world of Graphs



Devansh Jain,
Indian Institute of Technology, Bombay

(Mentor: Tathagat Verma)

Contents

Preface

Preface

About the project

adedsd

About the report

sdsd

References and Acknowledgement

sdfsd

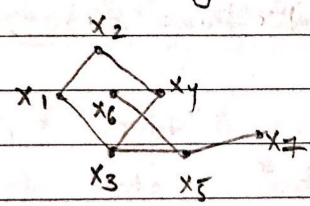
MIT 6.042J

About it

Lecture 6: Graph Theory and Coloring

L-6 Graph are ~~very~~ 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 subset of V called edges. (can be empty set). ⇒

$V=\{x_1,x_2,x_3\}$
$E=\phi$

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

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

equivalent to $x_1 - x_2$

Lecture 6: Graph Theory and Coloring

24

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

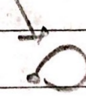
⇒ An edge $e = (x_i, x_j)$ is said to be incident to its end-points x_i & x_j.

⇒ The number of edges incident to a node is called the degree of the node.

~~deg(x)~~.

$\deg(x_5) = 3$

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

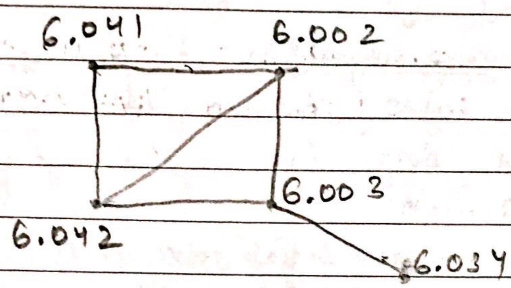


$|V|$ → cardinality notation i.e. no. of elements in set V.

Men-Women partners ratio

Lecture 6: Graph Theory and Coloring

25



Slots
5-7
7-9
9-11
11-1
1-3

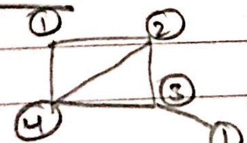
Graph-coloring problem

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

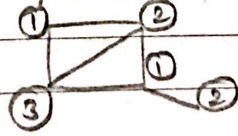
Minimum value of K for which such a coloring exists is the Chromatic Number of G $\chi(G)$

→ Color ⇒ Slots
Nodes ⇒ Courses

Option 1



Option 2



$\chi(G) = 3$

2 colors can't work because of triangle formation

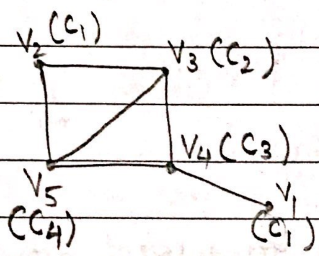
Lecture 6: Graph Theory and Coloring

26

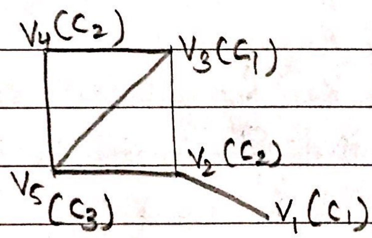
Date ______
Page ______

Basic graph coloring algorithm

1. Order the nodes $V_1, \ldots, V_n$
2. Order the colors $C_1, \ldots$
3. For $i = 1, 2, \ldots, n$
   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
$P(n) =$ "If every node in n-node graph G with max. degree d then this basic algorithm uses atmost $d+1$ colors for G."

Base case:

$n = 1 \Rightarrow |E| = 0 \Rightarrow d = 0 \Rightarrow$ one color

$\therefore P(1)$ 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: $V_1, \ldots, V_n, V_{n+1}$

Let G' be the graph after removing V_{n+1}

max. degree of nodes in $G \leq d$

$\therefore$ Basic algorithm uses atmost $d+1$ colors for G'.

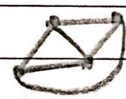
Lecture 6: Graph Theory and Coloring

28

In G, color modes $V_1 \ldots V_n$ same as G'.
V_{n+1} has atmost d neighbours therefore atleast one color is not present in V_{n+1}'s neighbours.
$\therefore$ Give V_{n+1} that color.

$\therefore$ $P(n+1)$ is true

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



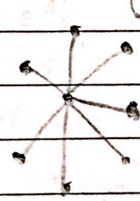
Degree $= n-1$
$\chi(K_n) = n$

n

$d = n/2$
$\chi(G) = 2$

Bipartite graph

n

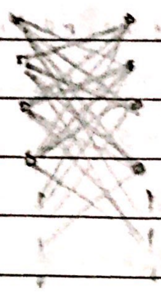


Star graph
$d = n-1$
$\chi(G) = 2$

The Algorithm may be much better than the theorem

Lecture 6: Graph Theory and Coloring

29



Good ordering (2 Colors)

$(C_1)\ V_1$ • • $V_5\ (C_2)$
$(C_1)\ V_2$ • • $V_6\ (C_2)$
$(C_1)\ V_3$ • • $V_7\ (C_2)$
$(C_1)\ V_4$ • • $V_8\ (C_2)$
⋮

Bad ordering ($n/2$ colors)

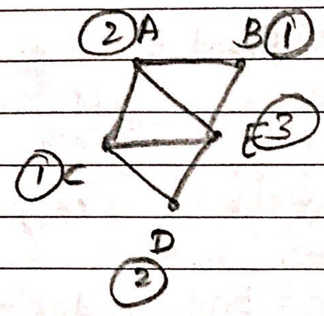
$(C_1)\ V_1$ • • $V_2\ (C_1)$
$(C_2)\ V_3$ • • $V_4\ (C_2)$
$(C_3)\ V_5$ • • $V_6\ (C_3)$
$(C_4)\ V_7$ • • $V_8\ (C_5)$
⋮

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

resource allocation problems (load balancing traffic on internet)

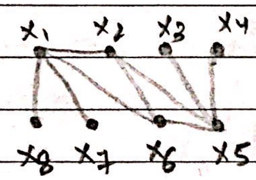
30



$\chi(G)=3$

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

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

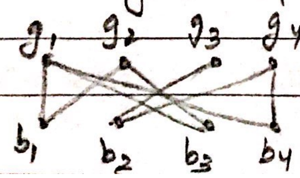


$\{x_1-x_6, x_2-x_5\}$ is a matching of size 2.

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

Size 4 matching is not possible in this graph.

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



$\{g_1-b_1, g_2-b_3, g_3-b_2, g_4-b_4\}$ is a matching

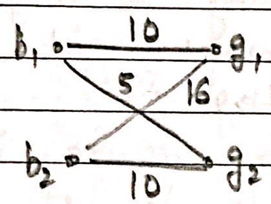
Lecture 7: Matching Problems

31

Weights?

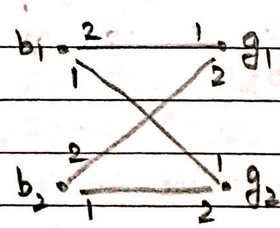
⇒ The weight of a matching (M) is the sum of the weights on the edge of M.

⇒ A minimum weight matching for graph G is a perfect matching for G with the minimum weight.



Min. weight matching
$\{b_1 - g_1, b_2 - g_2\}$
weight = 20

Preference list - similar to weights



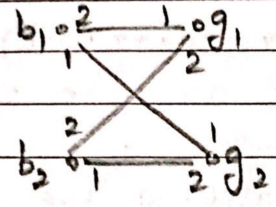
If $b_1 - g_1$, $b_2 - g_2$ then $b_1 - g_2$ will be rogue couple.

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

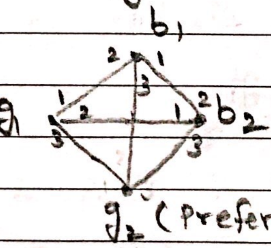
32



$b_1 - g_2$ & $b_2 - g_1$ 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.



(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 Δ is symmetric)

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

Lecture 7: Matching Problems

33

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	Day1	Day2	Day3	Day4	
1 ~~C~~BEAD	A	245	5	5	5	~~A-5~~
~~2 ABECD~~	B		2	21	2	B-2
3 DCBAE	C	1	14	4	4	C-4
4 ~~A~~CDBE	D	3	3	3	3	D-3
5 ABDEC	E				1	~~E-1~~

Lecture 7: Matching Problems

34

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

Theorem1 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.

Lecture 7: Matching Problems

35

Date
Page

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 invarient 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.
Case1 G rejects B on day d+1.
Then there was a better boy
⇒ P holds on Day d+1
Case2 G rejected B before day d+1
P on day d ⇒ G had a better boy on day d.
⇒ 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
⇒ every girl by lemma 1 has a better suitor
⇒ every girl is married ⇒ every boy is married
⇒ B is married.

Lecture 7: Matching Problems

36

classmate
Date
Page

Theorem3 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

$\Rightarrow$ G has a better suitor than B (lemma 1)

$\Rightarrow$ G married someone whom she prefers over B.

$\Rightarrow$ G is not rogue with B.

Case 2 G didn't reject B

$\Rightarrow$ B never serenaded G

$\Rightarrow$ G is lower in order than B's spouse.

$\Rightarrow$ B is not rogue with G

$\therefore$ No rogue couples.

Lecture 7: Matching Problems

37

Let S be the set of all stable matching
$S \neq \phi$ 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 → The Marriage Algorithm

Lecture 8: Graph Theory II: Minimum Spanning Trees

38

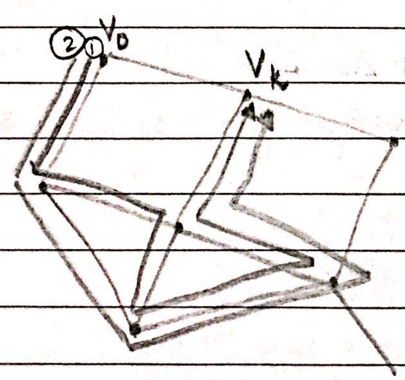
L-8
- Walks & Paths
- Connectivity
- Cycles & Closed walk
- Spanning Tree (ST)
- Min-weight spanning tree (MST)

Walks & Paths

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

Eg:- $v_0 - v_1 - \cdots - v_k$
Start ... end

It has k edges & length = k



① → walk, not path
② → walk, and path

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

Lecture 8: Graph Theory II: Minimum Spanning Trees

39

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

Proof $\exists$ walk u to v

By well-ordering principle: walk of minimal length.

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

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 $\exists_{i \neq j}$ $v_i = v_j$

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

Now this a shorter walk

thus contradicting our assumption. ∎

Lecture 8: Graph Theory II: Minimum Spanning Trees

(40)

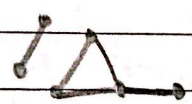
Date ________
Page ________

Connectivity

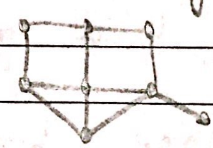
⇒ u and v are connected if there is a path from u to v.

⇒ 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

~~(also known as tours)~~

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

$$v_0 - v_1 - \cdots - v_k = v_0$$

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

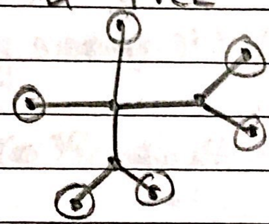
Lecture 8: Graph Theory II: Minimum Spanning Trees

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

Lemma3 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. ⓪

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 (Lemma2)
By $P(n)$: this subgraph has $n-1$ edges.
Re-attach v $\therefore$ 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

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.

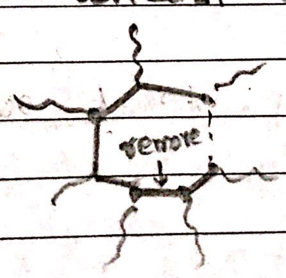
Theorem Every connected graph has a ST.

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.



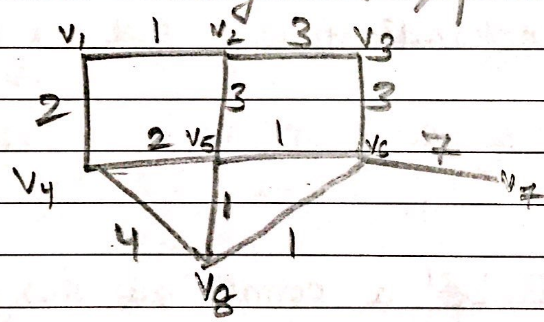
if we remove one edge of the cycle, we still have a connected subgraph.

Lecture 8: Graph Theory II: Minimum Spanning Trees

44

Subgraph we obtain is smaller than T (w.r.t. number of edges) thus contradicting our minimum condition on T.

Minimum-weighted spanning tree



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

Weight of $ST_1 = 2 + 2 + 3 + 1 + 3 + 7 + 1 = 19$

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

Weight of $ST_2 = 1 + 2 + 2 + 1 + 3 + 7 + 1 = 17$

Lecture 8: Graph Theory II: Minimum Spanning Trees

45

classmate

Date

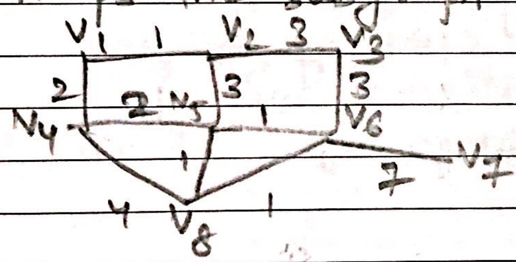
Page

⇒ 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) $V_5 - V_6$ ⇒ $V_5 - V_8$ (X)

+2 (4) $V_1 - V_4$

+2 (5) $V_4 - V_5$ ⇒ $V_4 - V_8$ (X), $V_2 - V_5$ (X)

+3 (6) $V_2 - V_3$ ⇒ $V_3 - V_6$ (X)

+7 (7) $V_6 - V_7$

17

Lecture 8: Graph Theory II: Minimum Spanning Trees

46

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 picked, we know S is an MST.

Proof By Induction:

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

<u>Base Case:</u> $m=0 \Rightarrow S=\phi \subseteq E$ for any MST $T=(V,E)$

<u>Ind. Step:</u> Assume $P(m)$ 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

47

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

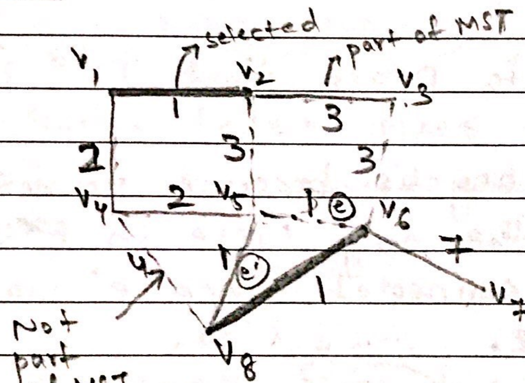
Case 1: $e \in E^* \Rightarrow S \cup \{e\} \subseteq E^* \Longrightarrow P(m+1)$ holds

Case 2 $e \notin E^*$:

$S = \{v_1 - v_2, v_6 - v_8\}$

$E^* = \{v_1 - v_2, v_2 - v_3, v_1 - v_4, v_4 - v_5, v_5 - v_8, v_6 - v_8, v_6 - v_7\}$

$e = v_5 - v_6$



~~From the algo,~~

Algo $\Rightarrow S \cup \{e\}$ has no cycle
(This is how we choose $(m+1)^{th}$ edge)

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

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

Lecture 8: Graph Theory II: Minimum Spanning Trees

48

Algo could have selected e or e'

$\Rightarrow$ weight of $e \leq$ weight of e'.

Swap e and e' in T^*:

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.

weight of T^{**} = weighted sum of E^{**}

= Weighted sum of E^* − weight of e' + weight of e

= weight ~~sum of~~ of T^* − weight of e' + weight of e

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. $\square$

Lecture 9: Communication Networks

49

L-9 → L-10

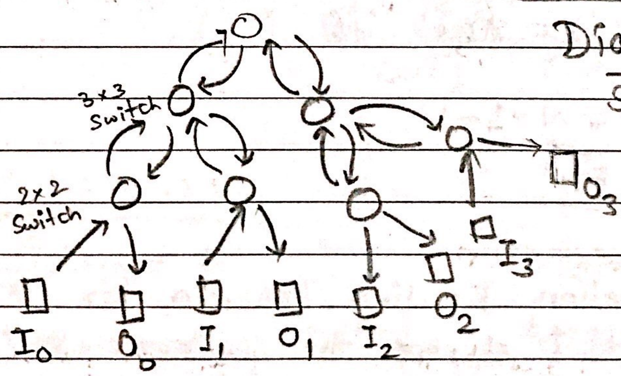
Communication Networks

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

N x N Networks ($N = 2^k$).

Complete Binary Tree

○ = switch (direct packets through a network)
□ = terminal (source and destination of data)



4 x 4 Network

Diameter = 6

Switch count = 7

$\pi(i) = i$
↳ Congestion = 1

$\pi(i) = N-1-i$
↳ Congestion = 4

Max Congestion = 4

N x N Network

Diameter = $2(1 + \log_2 N)$

Max. Switch size = 3 x 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 b/w Input and Output that are farthest apart
of a network
(Maximum length of shortest path

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

Eg:- $\pi(i) = N-1-i$
$\pi(i) = i$

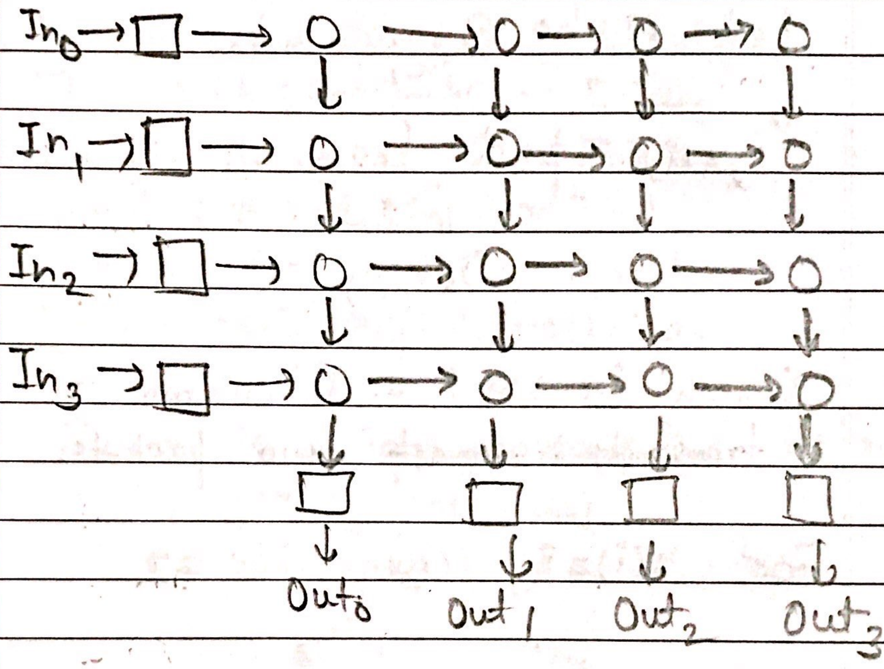
Permutation Routing Problem for π
- For each i direct the packet at I_i to $O_{\pi(i)}$
path taken is denoted by $P_{i,(\pi(i))}$

The congestion of paths $P_{0,\pi(0)}, \ldots, P_{N-1,\pi(N-1)}$ 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

2D Arrays (Grid structure)



4x4 Network

Diameter = 8

Max. switch size = 2x2

No. of Switches = 16

Max. congestion = 2

NXN Network

Diameter = 2N

Max. switch size = 2x2

No. of switches = N^2

Max. congestion = 2

Lecture 9: Communication Networks

52

classmate
Date
Page

Theorem ~~Max~~ Congestion of an N-input array is 2.

Proof Let π 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 $\pi(i)$ ~~← 2 packets~~ transmits atmost two packets.

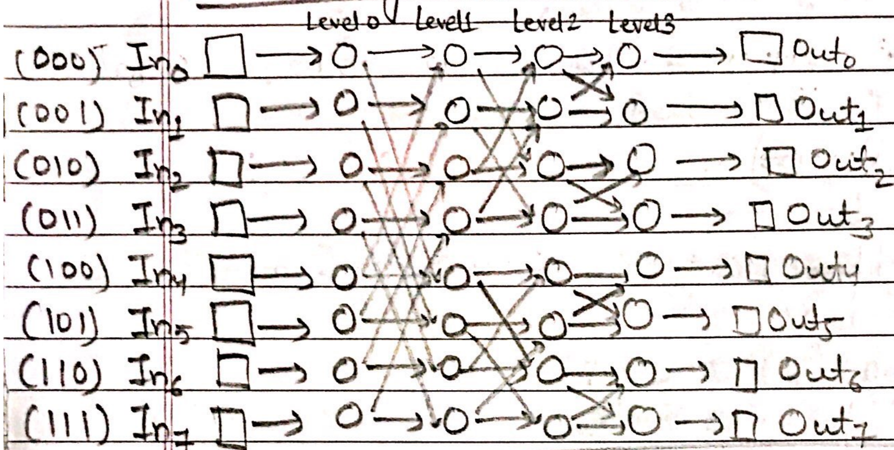
For $\pi(i) = i$, congestion $= 2$

∎

Lecture 9: Communication Networks

53

Butterfly ~~Structure~~ Network



Switch is uniquely identified its row & col

$$(\overbrace{(b_1 \cdots b_{\log N})}^{\text{binary rep. of input}}, \underbrace{l}_{\text{level}})$$

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

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

Routing

$$((x_1 \cdots x_{\log N}), 0) \longrightarrow ((y_1 x_2 \cdots x_{\log N}), 1)$$

$$((y_1 y_2 x_3 \cdots x_{\log N}), 2)$$

$$((y_1 \cdots y_l x_{l+1} \cdots x_{\log N}), l)$$

$$((y_1 \cdots y_{\log N}), \log N) \longleftarrow ((y_1 \cdots y_l x_{l+1} \cdots x_{\log N}), l)$$

Lecture 9: Communication Networks

54

In3 $\longrightarrow$ Out5

Eg:- $011 \longrightarrow 101$

$$
\begin{array}{l}
\text{In3} \rightarrow (011, 0) \\
\qquad \downarrow \\
\qquad (111, 1) \\
\qquad \downarrow \\
\qquad (101, 2) \\
\qquad \downarrow \\
\text{Out5} \leftarrow (101, 3)
\end{array}
$$

<u>N×N Network</u>

Diameter = $2 + \log N$

Max. Switch size = 2×2

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

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

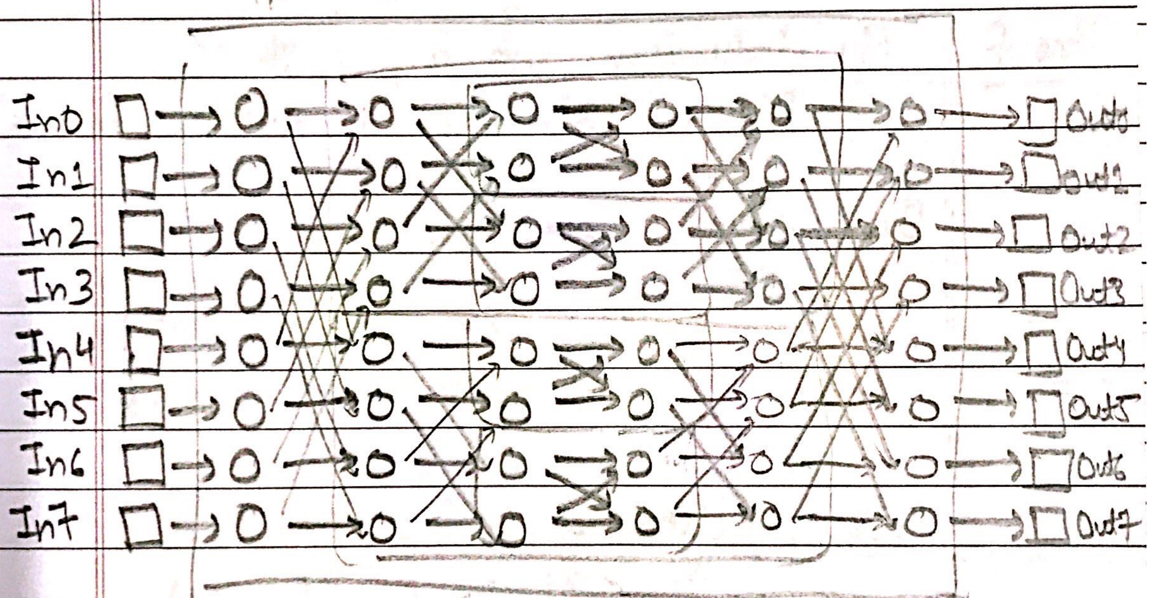
$= \sqrt{N/2}$ ($N \in 2^{2n+1}$)

Lecture 9: Communication Networks

55

Benes

1960s : Benes, a Bell Labs researcher



N x N Network

Diameter = $1 + 2\log N$

Max switch size = 2×2

No of switches = $2N \log N$

Congestion = 1

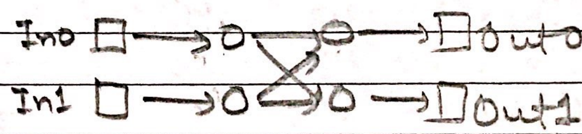
Lecture 9: Communication Networks

56

Theorem The Congestion of N-input Benes network is 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: $a = 1$ $N = 2$



$\pi(0) = 0$	$\pi(0) = 1$
$\pi(1) = 1$	$\pi(1) = 0$
Congestion = 1	Congestion = 1
(Just straight)	(Just cross-over)

Ind. Step: Assume P(a) is true

Example: $\pi(0) = 1$ $\pi(4) = 3$
$\pi(1) = 5$ $\pi(5) = 6$
$\pi(2) = 4$ $\pi(6) = 0$
$\pi(3) = 7$ $\pi(7) = 2$

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

Lecture 10: Graph Theory III

57

L-10

L-9

Euler tours
Directed graphs
- Definitions
- # walks No. of
- strong connectivity
- DAGs

Tournament Graphs

Euler tour

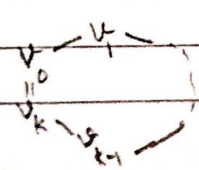
- he lived in Konigsberg
- seven bridges (birth of graph theory)

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: ($\Rightarrow$)

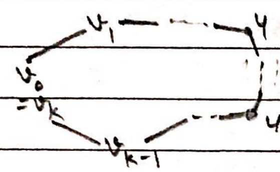
Assume $G = (V, E)$ has an Euler tour



Lecture 10: Graph Theory III

58

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



($\Leftarrow$)

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

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

① $v_k - u$ not in $W \Rightarrow v_0 - v_1 - \cdots - v_k - u$ is longer than $W \Rightarrow$ Contradicts

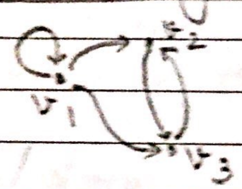
$\therefore$ All the edges in E incident of v_k are traversed in W.

② $v_k \neq v_0 \Rightarrow v_k$ has odd degree in walk W. As we have showed that all edges in E 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)



$v_2 \longrightarrow v_3$

(Tail) (Head)

indegree $(v_2) = 2$

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

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

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

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

$$A = \begin{array}{c|c} & \begin{matrix} v_1 & v_2 & v_3 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \end{matrix} & \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \qquad A^2 = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Lecture 10: Graph Theory III

60

classmate
Date
Page

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

By induction:

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

Base case $k=1$
Edge $v_i \to v_j$: $p_{ij}^{(1)} = 1 = a_{ij}^{(1)}$

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

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

Ind. step

Assume $P(k)$ holds.

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

$v_i \overset{k}{\rightsquigarrow} v_h \xrightarrow{1} v_j$

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

Ind. step

$$= a_{ij}^{(k+1)} \implies P(k+1) \text{ holds}$$

Matrix multiplication

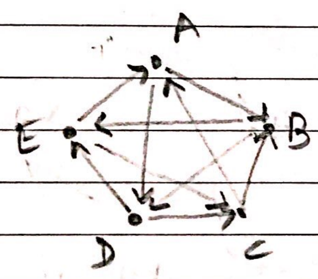
Lecture 10: Graph Theory III

61

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

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

<u>Tournament Graph</u>



either u beats v: $u \rightarrow v$

or v beats u: $v \rightarrow u$

⟶ E beats A, C and loses to B, D.

⇒ 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

62

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 $P(n)$, $v_1 \to v_2 \to \cdots \to v_n$

Approach ①
If $v \to v_1$ is an edge, then $P(n+1)$ holds.
(if $v_1 \to v$ and $v \to v_n$) then
Claim: $\exists\, i$ such that $v_i \to v$ & $v \to v_{i+1}$
Proof by contradiction
If no such i exists and we know $v_1 \to v$ then $v_i \to v$ $\forall i$ (by induction)
But $v \to v_n$ contradicts this.

~~There is also proof by strong induction.~~
See it.

Lecture 10: Graph Theory III

63

Approach ②

Case 1 $v \rightarrow v_1$

Case 2 $v_1 \rightarrow v$

Smallest i such that $v \rightarrow v_i$

$\Rightarrow v_k \rightarrow v \quad \forall \; k < i$

$\Rightarrow v_{i-1} \rightarrow v$

$v_1 \rightarrow \cdots \rightarrow v_{i-1} \rightarrow v_i \rightarrow \cdots \rightarrow v_n$

$v_{i-1} \rightarrow v \rightarrow v_i$

A new directed Hamiltonian path is form

$\therefore$ $P(n+1)$ holds

Lecture 10: Graph Theory III

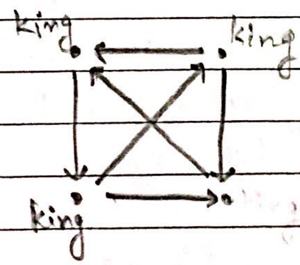
64

King Chicken Problem

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

u "virtually" pecks v if
- $u \to v$, or
- $\exists_w$ $u \to w \to v$

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

classmate
Date
Page

$\Rightarrow \exists_v : v \rightarrow u$, 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 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 :- $R = \{ (a,b) :$ student a is taking class $b \}$

$(a,b) \in R$: aRb, $a \sim_R b$

Relational symbol

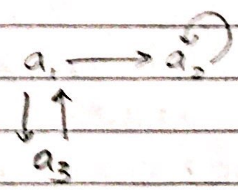
Lecture 11: Relations, Partial Orders, and Scheduling

66

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

Eg:- $A = \mathbb{Z}$: xRy iff $x \equiv y \pmod{5}$
$A = \mathbb{N}$: xRy iff $x \mid y$
$A = \mathbb{N}$: xRy 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
* relfexive if $xRx \;\; \forall\, x \in A$
* symmetric if $xRy \Rightarrow yRx \;\; \forall\, x, y \in R$
* anti-symmetric if $xRy \wedge yRx \Rightarrow x = y$
  (&)
* transitive if $xRy \wedge yRz \Rightarrow xRz$

Lecture 11: Relations, Partial Orders, and Scheduling

67

Ex	Reflexive	Symmetric	Anti-Symm	Transitive	
$x \equiv y \pmod 5$	✓	✓	✗ (2,7)	✓	→ Equivalence
$x \mid y$	✓	✗ (1,2)	✓	✓	→ Partial order
$x \leq y$	✓	✗ (1,2)	✓	✓	

⇒ An equivalence relation is reflexive, symmetric, transitive.

Ex: $A = \mathbb{Z} : xRy$ iff $x = y$

$A = \mathbb{Z} : xRy$ iff $x \equiv y \pmod n$, $n \in \mathbb{N}$

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

$$[x] = \{ y : xRy \}$$

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

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

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

Lecture 11: Relations, Partial Orders, and Scheduling

68

classmate
Date
Page

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

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

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

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

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

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

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

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).

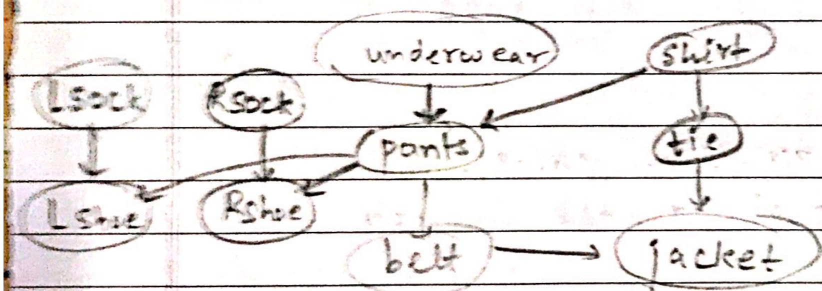
Lecture 11: Relations, Partial Orders, and Scheduling

69

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

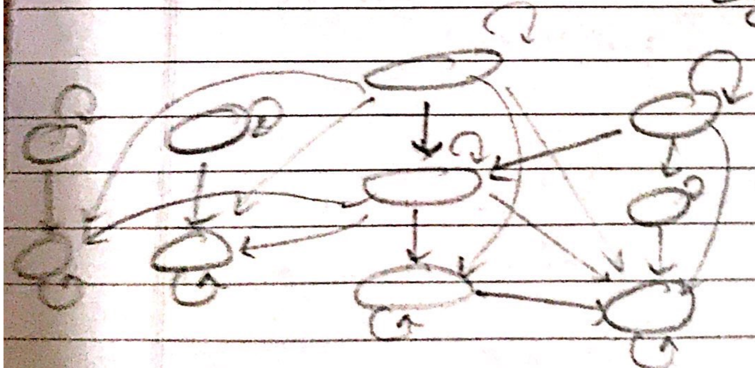
$(A, \preceq)$ is actually called partial ordered set. or poset

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



(Hesse diagram)

Transitive $\Rightarrow$ No directed cycles
Antisymmetric $\Rightarrow$ Only one directed edges b/w two vertices



(Poset)

Lecture 11: Relations, Partial Orders, and Scheduling

70

A Hasse diagram for a poset $(A, \preceq)$ is a directed graph with vertex set A and edge set $\preceq$ 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 $a_1, \ldots, a_n$ such that

$a_1 \preceq a_2 \preceq a_3 \ldots \preceq a_n \preceq a_1 \Rightarrow a_n \preceq a_1$ — II

(induction)

By transitivity, $a_1 \preceq a_2 \preceq \ldots \preceq a_n$

$\Rightarrow a_1 \preceq a_n$ — I

anti-symmetry

$a_1 = a_n$

Contradicts distinct element assumption. $\blacksquare$

Lecture 11: Relations, Partial Orders, and Scheduling

71

Conclusion of the theorem

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

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

⇒ a and b are comparable if $a \preceq b$ or $b \preceq a$.

⇒ 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 $(A, \preceq)$ is a total order $(A, \preceq_T)$ such that

$$\preceq \subseteq \preceq_T$$

$$(x \preceq y \Rightarrow x \preceq_T y)$$

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 $y \preceq x$.

$\Rightarrow$ $x \in A$ is maximal if $\neg \exists$ $y \in A, y \neq x$ s.t. $x \preceq y$.

not, there exists

Lemma Every finite poset has a minimum element.

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

Proof Let $C = (a_1 \preceq a_2 \dots \preceq a_n)$ be the max length chain.

Case 1: $a \notin \{a_1, \dots, a_n\}$

if $a \preceq a_1$ then C is not the longest chain.

So, $\neg a \preceq a_1$

Part of Lecture 13: Asymptotics

73

classmate
Date
Page 61

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

if $a \preceq a_1$ then we have a directed cycle which contradicts Theorem ___.

So, $\neg\, a \preceq a_1$

$\therefore \forall\, a \in A \quad \neg\, a \preceq a_1$

So, by definition, a_1 is the minimum element.

Add Parallel Task Scheduling & Dilworth's Lemma.

L-13
(45:30)

Asymptotic Notation

tilde $f(x) \sim g(x)$ if $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$

oh, big-oh $f(x) = O(g(x))$ if $\lim_{x\to\infty} \left|\frac{f(x)}{g(x)}\right| < \infty$ (finite)

Multiple usage:

$f(x) \leq O(g(x))$; $f(x)$ is $O(g(x))$; $f(x) \in O(g(x))$ — Formal math way

$O(g(x))$ is a set of all func. that grow slowly than $g(x)$

Part of Lecture 13: Asymptotics

74

classmate
Page 62

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

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

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

Proof $\lim_{x \to \infty} \frac{x^2}{x} = \lim_{x \to \infty} x$ 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 $\lim_{x \to \infty} \frac{x^{10}}{e^x} = 0$ 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 $\lim_{x \to \infty} \frac{4^x}{2^x} = \lim_{x \to \infty} 2^x \longrightarrow \infty$

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

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

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

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

$f(x) \leq O(g(x))$ 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

classmate
Date
Page 64

theta $f(x) = \Theta(g(x))$ if $0 < \lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| < \infty$

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

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

$\frac{x}{\ln(x)} \notin \Theta(x)$ | $\frac{x}{\ln(x)} \in O(x)$

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

$O \Rightarrow \leq$
$\Omega \Rightarrow \geq$
$\Theta \Rightarrow =$
$o \Rightarrow <$ ($\leq$ not $=$)
$\omega \Rightarrow >$ ($\geq$ not $=$)

Part of Lecture 13: Asymptotics

77

classmate
Date
Page 65

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

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

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

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

$x^2 \in \omega(0)$

<u>What not do</u>
<u>example.</u>

MIT 6.006

About it

95

L-13 **Graph I : BFS**

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

Graph search

"explore" a graph

Eg:- (s) ⟿ (t) find path

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

V → set of vertices

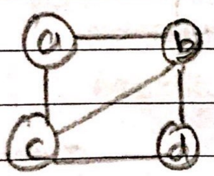
E → set of edges → $e = \{v, w\}$ ← unordered pairs (Undirected Graph)

OR

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

Every undirected graph is a directed graph

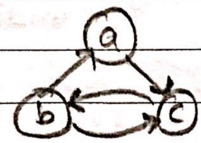
$\{v, w\} \to (v, w), (w, v)$



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

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

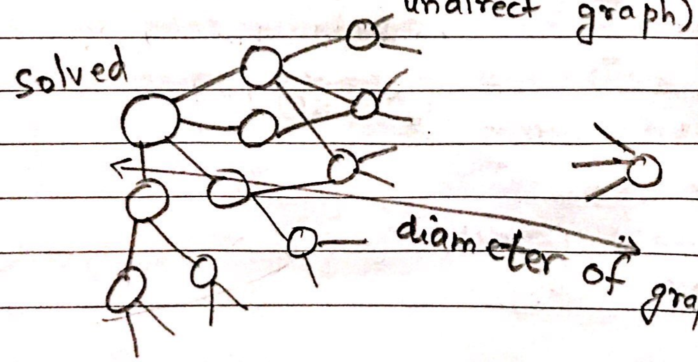
Pocket Cube : 2x2x2
- configuration graph
  - vertex for each possible state of cube

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

    24 symmetries

    1/3rd of states can only be reached without breaking it apart
  - edge for each possible move

    (as every move is undoable, undirect graph)



2x2x2 : 11

3x3x3 : 20

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

97

Graph representation:

Adjacency Lists

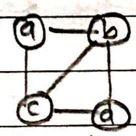
array Adj of size $|V|$

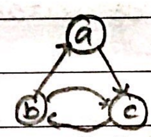
- each element is a pointer to a linked list

for each vertex $u \in V$,

Adj[u] stores u's neighbours

↳ $\{u \in V \mid (u, v) \in E\}$





Adj[a] = {b, c}
Adj[b] = {a, c, d}
Adj[c] = {a, b, d}
Adj[d] = {b, c}

Adj[a] = {c}
Adj[b] = {a, c}
Adj[c] = {b}

Object-oriented pattern

Vertex $V \longrightarrow$ V.neighbours = 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

Implicit representation

- Adj(u) is a function
- v.neighbours() is a method
  Uses less space

This would be helpful for puzzles like Rubik's cube.

Fun fact: 7x7x7 has configurations more than number of atoms in the known universe.

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

Breadth-first Search

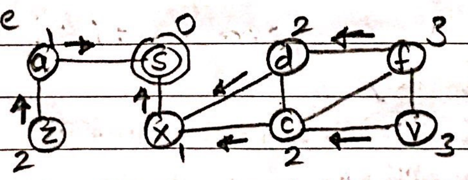
- Visit all the nodes reachable from given $s \in V$
- $O(V+E)$ time
- look at nodes reachable in
  0 moves, 1 move, 2 move, ...
  $\{s\}$ Adj[s]
- careful to avoid duplicates

99

```
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



(←) parent direction

i=1 next = [a,x]
i=2 next = [z, d, c]
i=3 next = [f, v]
i=4 next = []

100

Shortest paths

$v \leftarrow parent[v] \leftarrow parent[parent[v]]$

$\uparrow$

$\vdots$

$\uparrow$

s

is the shortest path from s to v.
of length $level[v]$.

Each vertex is checked once

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

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

R-13 (Halloween Day)

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

101

L-14

Graph II: DFS
- depth-first search
- edge classification
- cycle detection
- topological sort

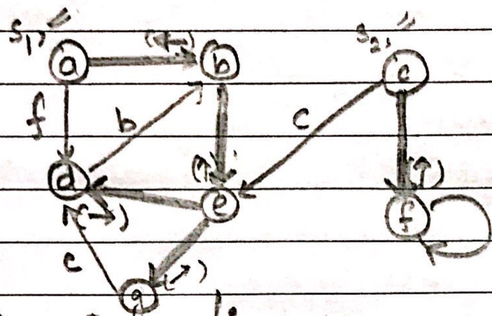
Depth-first search (DFS)
- recurisevly explore graph, backtracking as necessary
- careful not to repeat

```
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(V, Adj, s)
```

102

classmate

Date
Page



(Not strongly connected, can get from 'b' to 'c')

(←) parent direction

Running time

$O(V+E)$ (Linear time)

- visit each vertex once in DFS alone $O(V)$
- DFS-Visit(..., .., v) called atmost once per vertex v.
  ↳ pay $|Adj[v]|$

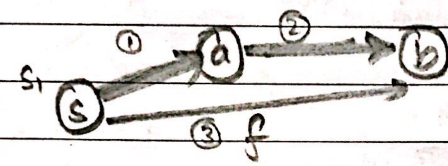
$$O\left(\sum_{v\in V}|Adj[V]|\right) = O(E) \longrightarrow O(V+E)$$

Edge classification

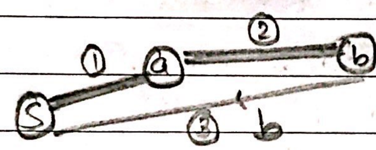
- tree edges (parent pointers) (Bold)
  visit new vertex via that edge.
- forward edges (f)
  goes from a node to a descendant in the tree
- backward edges (b)
  goes from a node to an ancestor in the tree
- cross edges (or others) (c)
  between two non-ancestor-related subtrees

103

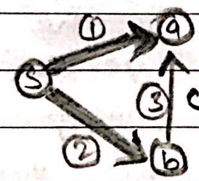
For undirected graphs,
forward and cross edges don't exist.



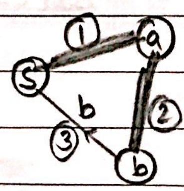
DIR.



UNDIR.



DIR.



UNDIR.

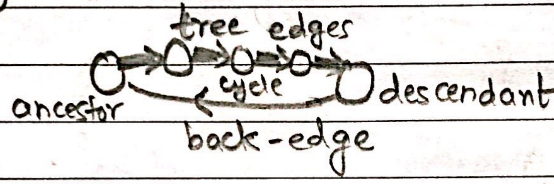
Useful for cycle detection and topological sort.

104

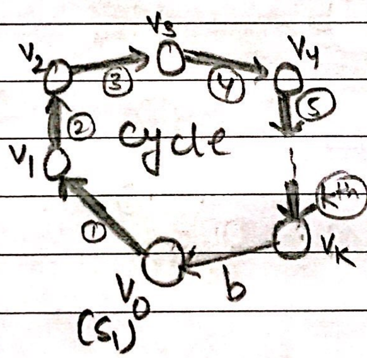
Cycle detection

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

Proof. $(\Leftarrow)$



$(\Rightarrow)$

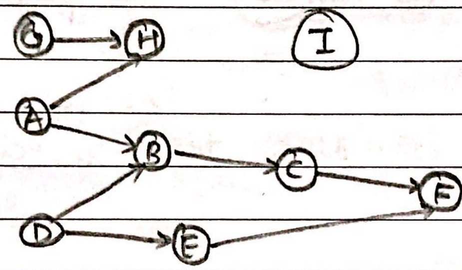


Assuming v_0 is the first vertex in cycle visited by DFS. then $v_k \to v_0$ is back-edge

1.05

Job Scheduling

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

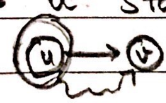


Topological sort (Sorting vertices):
run DFS
output reverse of finishing times of vertices.

Correctness

for any edge $e = (u, v)$
v finishes before u finishes. (To show)

Case 1: u starts before v



$\Rightarrow$ visit v before u finishes

visit u → start u → visit v → start v → finish v → finish u

106

classmate
Date
Page

Case 2: v starts before u

$u \rightarrow v$ # contradiction
cycle

else

$u \rightarrow v$ No path from v to u
then v finishes first
(Actually even before
u starts)

R-14 Procedurally analysed
example of appling DFS
on a direct graph and an undirected
graph.

107

L-15

Shortest Path I : Intro

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

G(V, E, W)

V → vertices, E → edges, W → weight $w: E \to \mathbb{R}$

Two algos

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

Path $p = \langle v_0, v_1, \ldots, v_k \rangle$ (sequence)

$(v_i, v_{i+1}) \in E \quad \forall \quad 0 \leq i < k$

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

find p with minimum weight.

A → B 1 paths
A → C 2 paths
A → E 4 paths
A → G 8 paths

Paths increase exponentially

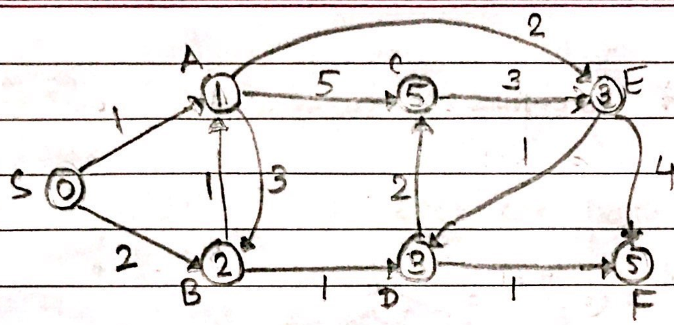
Weighted graphs

$V_0 \xrightarrow{p} V_k$ (V_0) is a path from V_0 to V_0 of weight 0.

Shortest path weight from u to v as

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

109



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

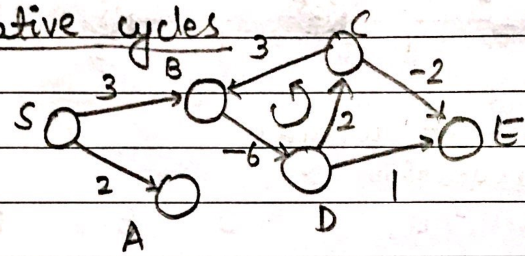
	S	A	B	C	D	E	F
	0	∞	∞	∞	∞	∞	∞
From S	0	1	2	∞	∞	∞	∞
From A	0	1	2(<4)	6	∞	3	∞
From C	0	1	2	6	∞	3(<9)	∞
From E	0	1	2	6	4	3	7
From D	0	1	2	6	4	3	5(<7)
From F	0	1	2	6	4	3	5
From B	0	1(<3)	2	6	3(<4)	3	5
From D	0	1	2	5(<6)	4	3	5
From C	0	1	2	5	4	3	5

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

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

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

Negative cycles



$\delta(s,s)=0 \qquad \delta(s,A)=2$

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

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

General structure (no negative cycles)

Initialize for $u \in V$ $\quad d[u] \leftarrow \infty \quad \pi[u] \leftarrow NIL$

$d[s] \leftarrow 0$

Repeat select edge (u,v) [somehow]

"Relax" edge(u,v) : if $d[v] > d[u] + w(u,v)$

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

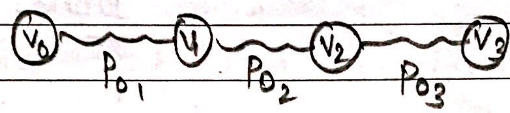
$\pi[v] \leftarrow u$

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

(111)

Optimal substructure

- Subpaths of a shortest path are shortest paths



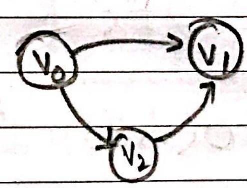
$\{P_{0_1}, P_{0_2}, P_{0_3}\} = SP$

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

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

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

Triangle inequality (used in L-16)

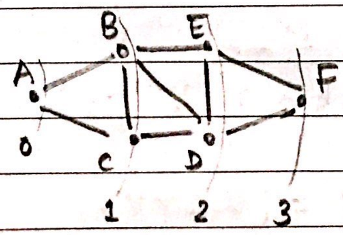


$\delta(V_0, V_1) \leq \delta(V_0, V_2) + \delta(V_2, V_1)$

R-15

Shortest path

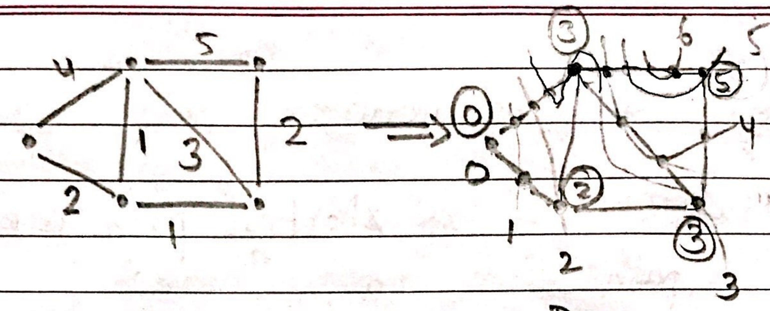
Thinking of something simple.



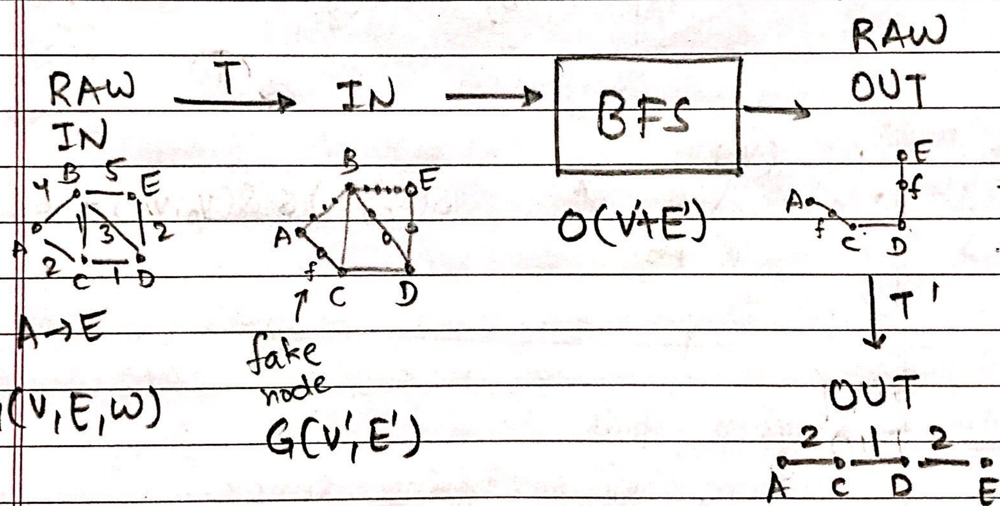
BFS

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

112



Done.

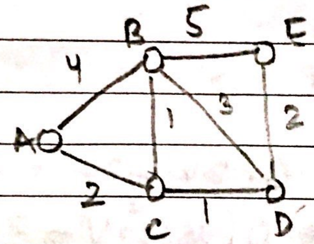


To find out running time, we need to know V', E'.

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

For small paths, this is a nice algo.

113



Task

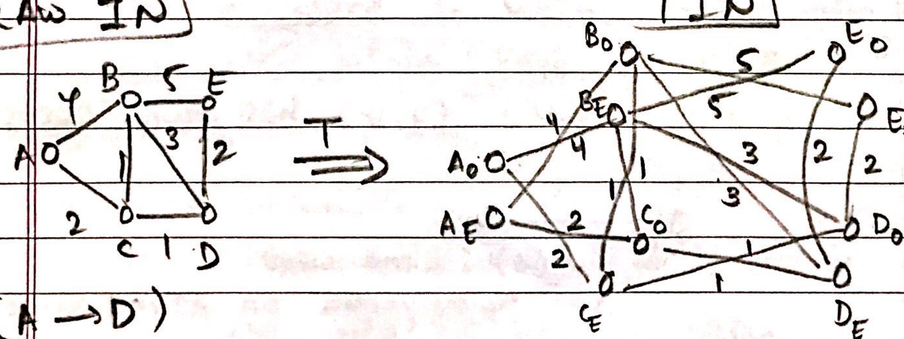
Find the shortest path with odd no. of edges.

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 X —— Y you can reach y in odd #edges, this way.

RAW IN

IN



(A → D)

(Messed-up graph!)

↓

Dijkstra

$(A \to C \to B \to D) \xleftarrow{T'} (A_E \to C_0 \to B_E \to D_0) \leftarrow$ Dijkstra

2 1 3

OUT

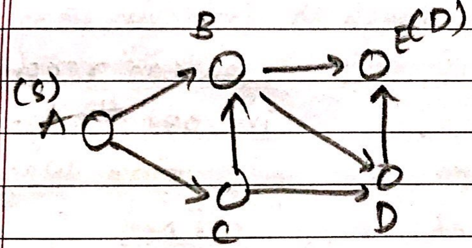
RAW OUT

114

$E' = 2E \qquad V' = 2V$

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

A hard one (took me 20 min to understand)



Given DIR. Graph

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

$$u \xrightarrow[t_c(s)\,:\,\text{time cost}]{f_c\,:\,\text{fuel cost}} v$$

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.

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

115

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

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

Transform(T)

$v \xrightarrow{(\times M)} (v, t)$

$(u,v) \xrightarrow{(\times M)} \left((u,t),\ (v, t+t_c((u,v),t))\right)$

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_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

$i=0$ ∞
$i=1$ ∞ No such path
$i=2$ ∞
$i=3$ ∞
⋮
$i=k-1$ ∞
$i=k$ $p \longrightarrow$ eco-friendly fastest path.
$i=k+1$
⋮

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)

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

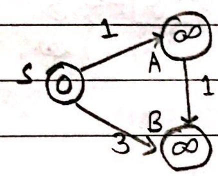
Complexity of algo
$= O(V' \lg V' + E')$
$= O(V \cdot M \lg(V \cdot M) + E \cdot M + V \cdot M)$
$= O(V \lg V + E)$
(though, constant is much worse than Dijkstra)

117

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.

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

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

$\downarrow$ $\pi[A]=S$, $\pi[B]=S$

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

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

S.P. from S to B : $B \leftarrow \underbrace{\pi[B]}_{=A} \longleftarrow \underbrace{\pi[\pi[B]]}_{=S} \longleftarrow \cdots \longleftarrow S$

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

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

118

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

satisfies

Inductive step:

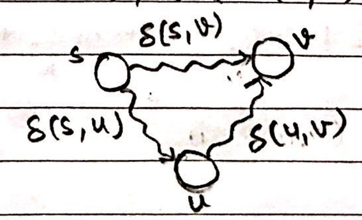
For $n=k$ steps, the lemma satisfies.

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

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

By triangle-inequality

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



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

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

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

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

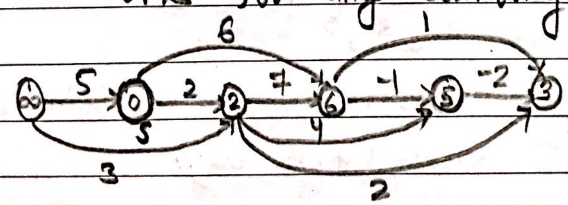
(119)

<u>DAGs</u> (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. <u>One</u> pass over vertices in topologically sorted order relaxing each edge that leaves that vertex.

$O(V+E)$ time

• This works for any starting vertex.



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

classmate
Date
Page

(120)

Demo of Dijkstra

Mechanically validating Dijkstra
(28:28)

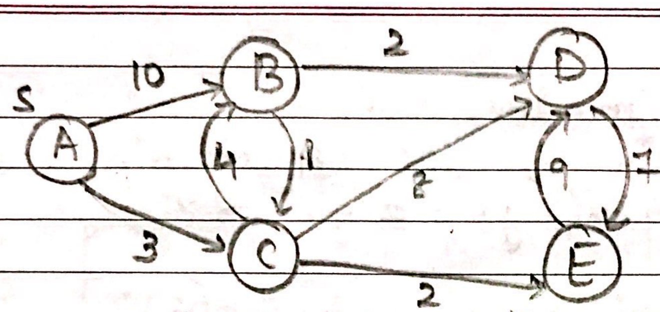
Dijkstra is a greedy algo.
It is easy to understand but not so easy to prove its correctness.

Dijkstra (G, ω, s):
  Initialize (G, s) $S \leftarrow \phi$ $Q = V[G]$
  $d[s] = 0$
  while $Q \neq \phi$
    $u \leftarrow$ extract_min(Q)
    $S \leftarrow S \cup \{u\}$
    for each vertex $v \in Adj[u]$
      Relax (u, v, ω)

Q is a priority-queue of $d[v]\ \forall\ v \in V$.
extract_min(Q) returns vertex not yet evaluted and has minimum d[ ].

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

121



d[A]	0	0	0	0	0	0
d[B]	∞	10	7	7	7	7
d[C]	∞	3	3	3	3	3
d[D]	∞	∞	11	11	7	7
d[E]	∞	∞	5	5	5	5
S	{ }	{A}	{A,C}	{A,C,E}	{A,C,E,B}	{A,C,E,B,D}
Q	{A,B,C,D,E}	{B,C,D,E}	{B,D,E}	{B,D}	{D}	{ }

A C E B D

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

classmate

Date
Page

122

Binary min heap:

$\Theta(\lg V)$ for extract-min, update

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

Fibonacci heap (not 6.006):

$\Theta(\lg V)$ for extract-min

$\Theta(1)$ ammortized for update

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

Theoretically,

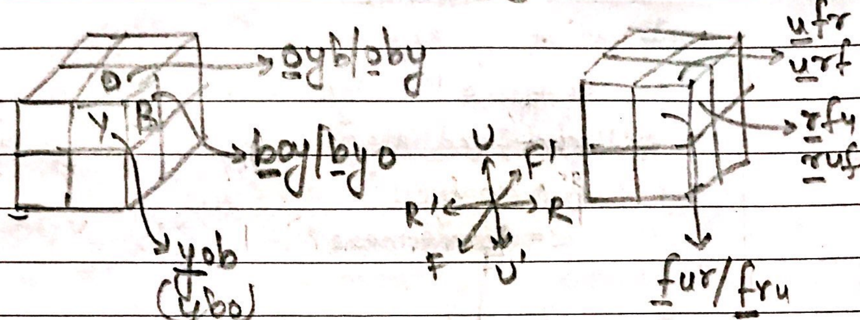
complexity is $\Theta(E \lg V)$ (worst-case)

but practically, $\Theta(V \lg V + E)$ (ammortized).

classmate
Date
Page

123

R-16 2×2×2 Rubik's cube



Plastic faces Wireframe

8 cublets
24 faces

Configuration is an array of 24

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

Total configurations: 24!

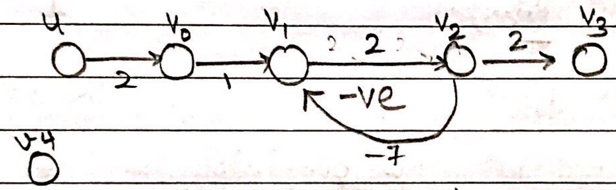
Six types of moves: $f_c, f_{cc}, r_c, r_{cc}, u_c, u_{cc}$

Finding Starcraft game strategy
(Using Dijkstra)

124

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 $v \in V$ $d[v] \leftarrow \infty$ $\pi[v] \leftarrow$ None

$d[s]=0$

Main loop : Repeat → select edge [somehow]

→ Relax (u,v,W)

until you can't relax anymore.

125

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 lg V + E) time algo (Dijkstra)

```
Bellman-Ford (G, w, s)
    Initialize()
    for i=1 to i=|V|-1
        for each edge (u,v) ∈ E          O(VE)
            Relax (u,v,w)  →  Relax(u,v,w):
                                 if d[v] > d[u] + w(u,v):
                                     d[v] = d[u] + w(u,v)
                                     π[v] = u
    for each edge (u,v) ∈ E              O(E) Check
        if d[v] > d[u] + w(u,v)
            then report -ve cycle exists.
```

Complexity : O(VE)

classmate

Date

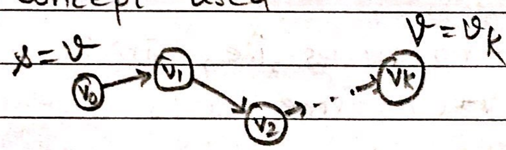
Page

126

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

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

Concept used



$v_0, v_1, \ldots, v_k$ 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 $v \in V$

$p = \langle v_0, v_1, \ldots, v_k \rangle$ $v_0 = s$ to $v_k = v$

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.)

classmate

Date ______
Page ______

127

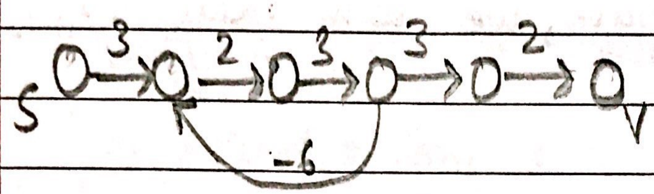
After 2 passes, $d[v_2] = \delta(s, v_2)$ because in 2^{nd} pass, we will relax (v_1, v_2) as well.

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.

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)

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)

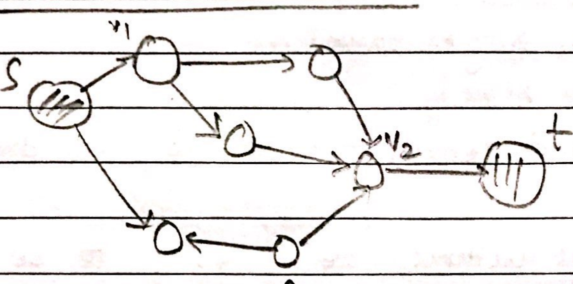
```
1. Dijkstra(G, W, s):
2.    Initialize()  ← d[s]=0 , d[u≠s] = ∞
3.    Q ← V[G]
4.    while Q ≠ ϕ
5.        do u ← extract_min(Q)
6.        for each vertex v ∈ Adj[u]
7.            do Relax(u,v,w)
```

If we are asked to find shortest path from s to t. then we can replace 4. by while($Q = \phi$ && $u \neq t$)

This optimizes the run-time. Worst-case run-time still stays same but average improves.

130

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 $\quad$ Distances for backward search

π_f: normal parent $\quad$ π_b: reverse parent

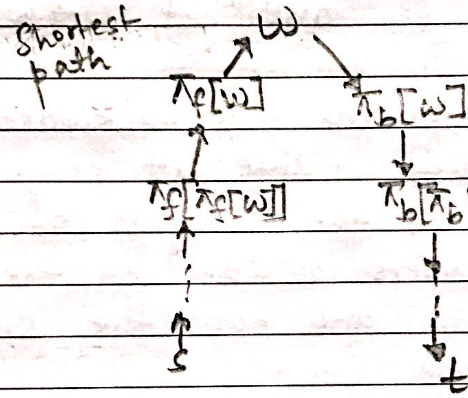
Priority Queues: Q_f (forward)
Q_b (backward).

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

(131)

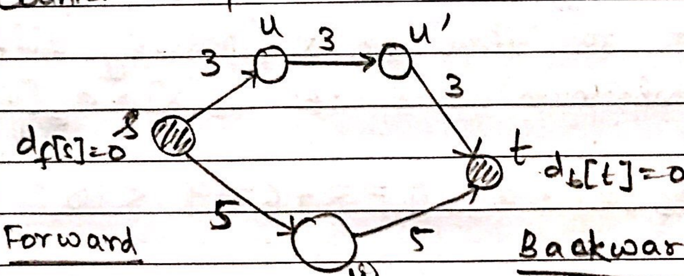
Finding the shortest path

Claim: If w was processed first from both Q_f & Q_b.



This claim is actually false.

Counter-example:



Forward

from s (1) $d_f[u]=3$

$d_f[w]=5$

$Q_f=\{u, u', w, t\}$

from u (2) $d_f[u']=6$

$Q_f=\{u', w, t\}$

from w (3) $d_f[t]=10$

$Q_f=\{u', t\}$

Backward

from t (1) $d_b[u']=3$

$d_b[w]=5$

$Q_b=\{u, u', w, s\}$

from u' (2) $d_b[u]=6$

$Q_b=\{u, w, s\}$

from w (3) $d_f[s]=10$

$Q_b=\{u, s\}$

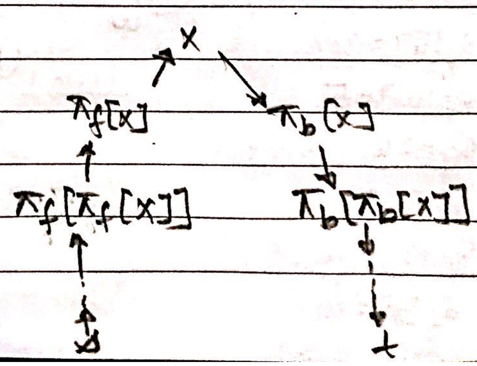
w has been processed from Q_b & Q_f
$\pi_b[w] = t \quad \pi_f[w] = s$
But, the shortest path is $s \to u \to u' \to t$
and not $s \to w \to t$.

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:

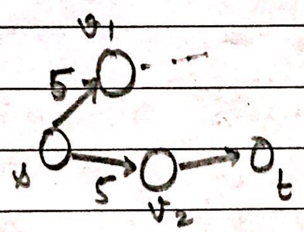


133

Goal-directed search

- modify edge weights with potential functions.

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



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

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

---

landmark $l \in V$

precompute $\delta(u, l)$ $\forall\, u \in V$

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