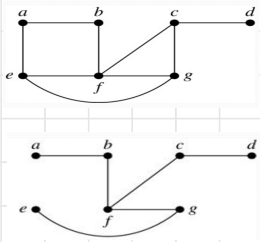
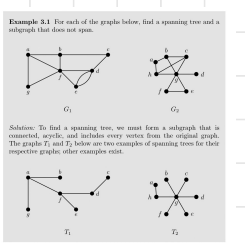
CHP 3

Spanning Tree

- connected (every vertex reachable)
- min edges possible
- no cycle

$e = v - 1$ (edges = e, vertex = v)

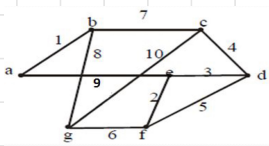




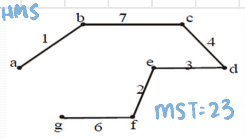
PRIMS ALGO

- Start with any vertex
- draw smaller adjacent node
- move connectedly
- no cycles

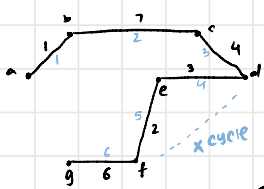
Order of adding the edges:
{a, b}, {b, c}, {c, d}, {d, e}, {e, f}, {f, g}



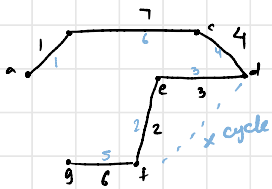
AFTER ALGORITHMS



— adding sequence



— adding sequence



7 - 1 = 6

6 edges used

KRUSKALS ALGO

- start with smallest
- adding order with small to big
- move disconnectedly (jump to small edges)
- no cycles

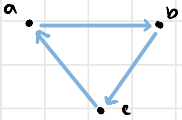
Order of adding the edges:
{a, b}, {e, f}, {e, d}, {c, d}, {g, f}, {b, c}

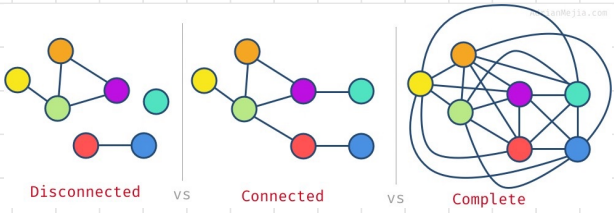
MIN SPANNING TREE

- connected weighted graph
- smallest sum of weights of edges

Simple graph

- no multiple edge
- no loop





3.1

Acyclic

↳ no loops

↳ no circuits → start, end same

Tree

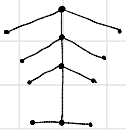
↳ is acyclic

↳ connected

Forest

↳ acyclic

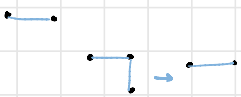
↳ not connected



3.2

TREE PROPERTIES

edges = $n - 1$ (n → vertices)
total degree of a tree = $2n - 2$



↳ every tree with 2 vertices has a leaf

↳ given a tree with leaf v, graph T-v is still a tree

↳ every tree is minimally connected

removal of any edge, disconnects the graph

↳ if any edge is added to a tree then

T+e contains only 1 cycle

3.2

THE ENUMERATION

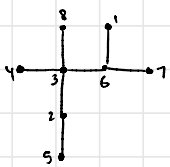
PRUFER SEQUENCE

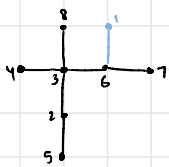
↳ a tree on $n > 2$
↳ length = $n-2$

→ missing vill → Prufer

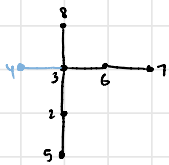
STEPS

1. find leaf with smallest label → l_i
2. make s_i neighbour to l_i
3. Delete that edge
4. Repeat till 2 edges left
5. List all s_i

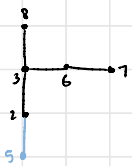




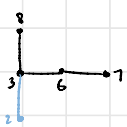
T_0: $l_1 = 1$ $s_2 = 6$



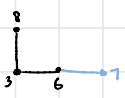
T_1: $l_2 = 4$ $s_2 = 3$



T_2: $l_3 = 5$ $s_2 = 2$



T_3: $l_4 = 2$ $s_4 = 3$



T_4: $l_5 = 7$ $s_5 = 6$

T_5: $l_6 = 6$ $s_6 = 3$

→ 2 edges left so stop

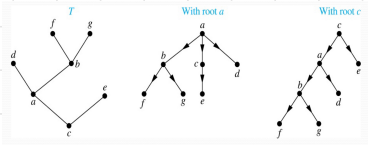
Prufer sequence = (6, 3, 2, 3, 6, 3)

3.3

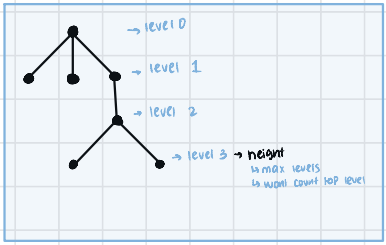
ROOTED TREES

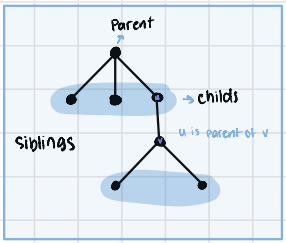
↳ one root only

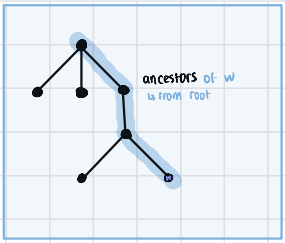
↳ all edges direct away from root

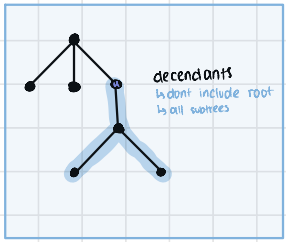


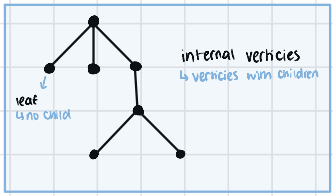
edge = vertices - 1





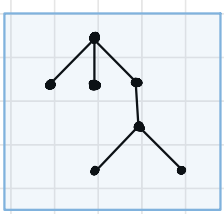






K-ary rooted tree

↳ every internal vertex has <= m children



Binary Tree

↳ ordered 2 m-ary root tree

↳ left children: left subtree

↳ right children: right subtree

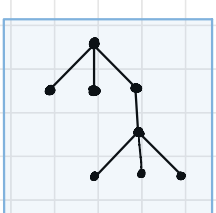
edges
e = n - 1
vertices

full m-ary n = mi + 1
internal vertex

$$e = \frac{n-1}{i}$$

full K-ary tree

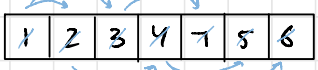
↳ every internal vertex has exactly m children



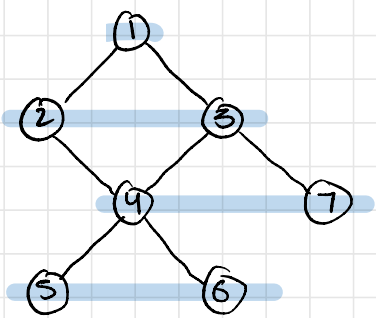
moves →

Breadth-First Search (BFS)

- ↳ start from top, put in Queue
- ↳ put adj vertices in Queue
- ↳ take out vertice when explored (take it out of queue)
- ↳ move to next vertice in Queue
- ↳ repeat till all vertices explored



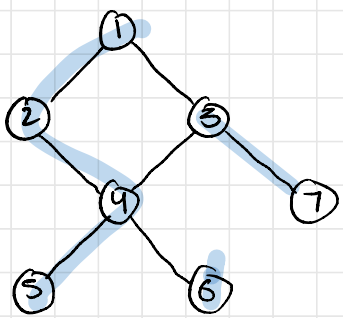
1 2 3 4 7 5 6

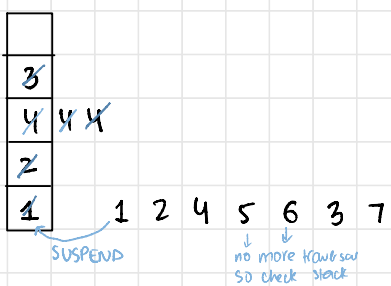


moves ↓

Depth First Search (DFS)

- ↳ start from top, write it
- ↳ move to adj vertex
- ↳ suspend prev vertex to stack
- ↳ repeat till no adj vertex
- ↳ backtrack by checking stack
- ↳ repeat till no vertices left

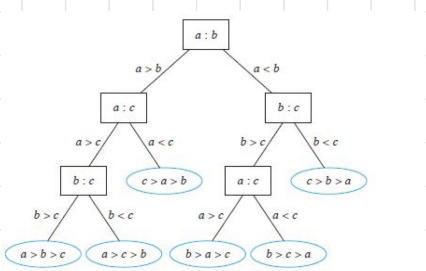




3.4.2

Decision Tree

- ↳ start with one root



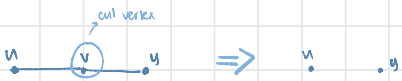
ishma hafeez
notes

4.1 Connectivity Measures

Cut vertex → no cycles

↳ deleting a vertex which disconnects graph

↳ v is on every $u - y$ path (vertices)

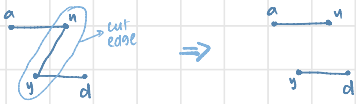


bridge

↳ deleting an edge which disconnects graph

↳ e is on every $u - y$ path (vertices)

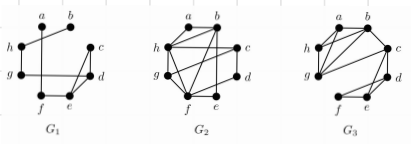
↳ e lies on no cycle of G

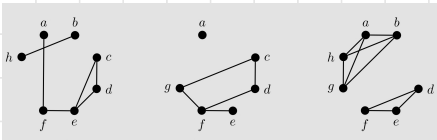


K-connected (K(G))

↳ smallest cut vertex set

a)



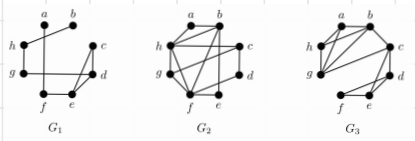


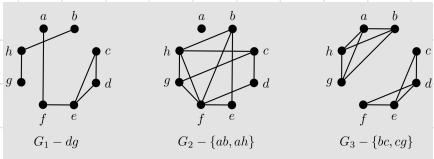
$K(G_1) = 1$ $K(G_2) = 2$ $K(G_3) = 1$

K-edge connected (K'(G))

↳ smallest bridge

a)





$K'(G_1) = 1$ $K'(G_2) = 2$ $K'(G_3) = 2$

The Triple $(\kappa, \kappa', \delta)$

→ Whitneys Theorem ?

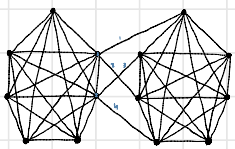
↳ $\kappa(G) \leq \kappa'(G) \leq \delta(G)$

↳ κ = no of vertices to join A and B → cut vertex

↳ κ' = no of edges joining A and B → cut bridge

↳ $\delta = n-1$ → make A and B complete graph with n vertices → min degree

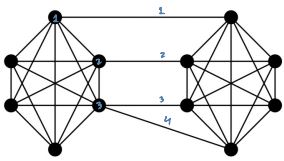
Q) $(\kappa, \kappa', \delta) = (2, 4, 6)$



$6 = n-1 \Rightarrow n = 7$

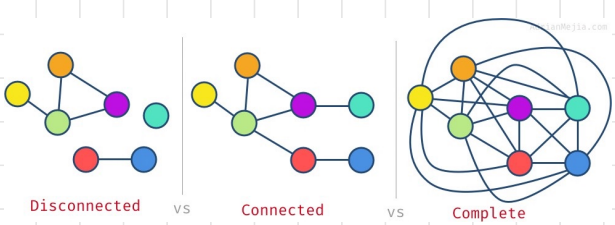
2 vertices joins A and B with 4 edges

Q) $(\kappa, \kappa', \delta) = (3, 4, 5)$



$5 = n-1 \Rightarrow n = 6$

3 vertices joins A and B with 4 edges



4.2 Connectivity and Pams

2 connected

- ↳ atleast 3 vertices
- ↳ connected
- ↳ no cut vertices

cycle

one edge

one vertice

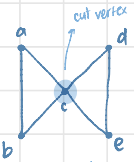
BLOCK

- ↳ 2 connected graph subgraph
- ↳ no cut vertex → non seperable

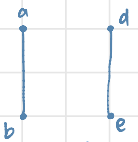
cycle

one edge

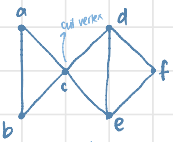
one vertice



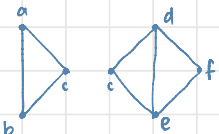
NO BLOCK



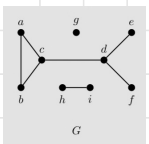
BLOCKS



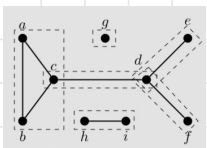
NO BLOCK



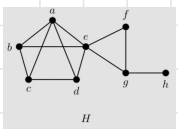
BLOCKS



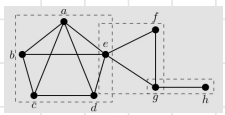
NO BLOCK



BLOCKS



NO BLOCK



BLOCKS

Subdivision

- ↳ add a vertex v between edge (u,y)
- ↳ proves effect for 2 connected graphs

G' = 2 connected ⟶ G = 2 connected

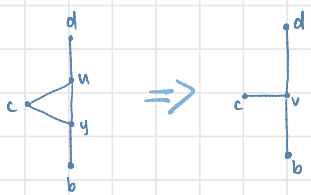
↓

obtained by subdividing an edge in G

Contradiction (G/e)

→ creates a smaller graph

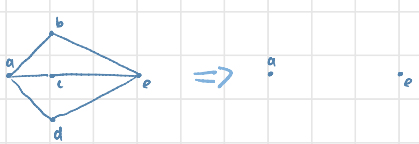
- ↳ replace edge (u,y) with v



Seperating set

- ↳ atleast 1 cut vertex
- ↳ make a set

Seperating set S: (b,c,d)



disjoint path

↳ no common vertices

(internally disjoint)

↳ no common edges

eg A: (a,b,c,d)

B: (e,f,g)

internally disjoint path

↳ only start and end vertices in common

↳ deleting (internal vertex) it will still leave a connected graph

eg P: (b,c,d,e)

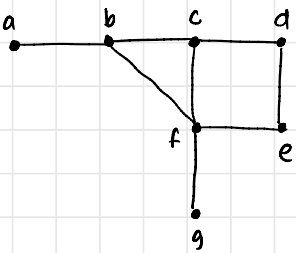
Q: (b,f,e)

edge disjoint path

↳ no common edges

e.g K: (a,b,c)

R: (d,e,f)



max internal disjoint path = 2

P: (b,c,d,e)

Q: (b,f,e)

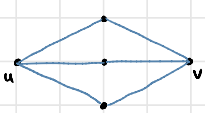
Q_2: (b,c,f,e)

Mengers Theorem

↳ let u any y be non adjacent vertices

↳ Then the

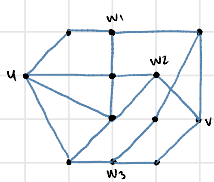
min no. of v seperating u any y = max no. of internally disjoint u-y paths



max no. of internally joint vertices = 3

HENCE

min no. of v seperating u-v = 3



min seperating set w: (w_1, w_2, w_3) = 3 (for u-v)

HENCE

max no. of internally joint vertices = 3

By deleting 3 internally joint vertices

the graph disconnects

4.4 NETWORK FLOW

Network
- a weighted graph
- capacity
- starting vertex s (source) → has only outgoing edges
- ending vertex t (sink) → has only incomming edges

flow f(e)
- a function that assigns a value f(e) to each arc
- a flow is feasible if
  - $c(e) \geq f(e)$ for all edges e
  - $f(e) \geq 0$ for all edges e
  - $f^+(v) = f^-(v)$ for all vertices (except s,t)
  - $f^-(s) = f^+(t) = 0$

value of flow (f)
$|f| = f^+(s) = f^-(t)$

capacity c(e)
- each edge (arc e) has a weight

slack k
- difference b/w capacity and flow
- $k(e) = c(e) - f(e)$

chain k
- a path where direction of arc is ignored

Augmented flow Algo

↳ gives max flow f

STEPS

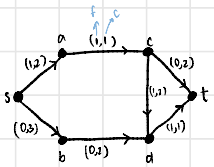
1. label $s \to (-\infty)$
2. go to all unlabled vertices with slack and if
   - ↳ outgoing vertice $(v^+, \_\_\_)$
   - ↳ incoming vertice $(v^-, \_\_\_)$

   if no slack then dont label

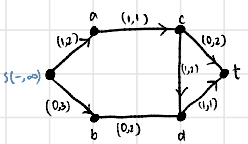
   s: (∞, ∞)
   labled v: $\min(\varepsilon(v), k(vu))$
   unlabled v: $\min(\varepsilon(v), f(uv))$
3. backtrack slacks to get s-t chain
4. remove all vertice labels except s and t and if
   - ↳ outgoing: increase flow by $t(\delta)$
   - ↳ incoming: decrease flow by $t(\delta)$

   Repeat until no slack to t

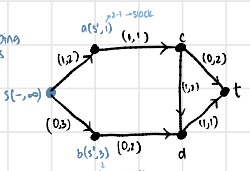
0)



step 1

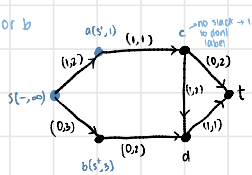


step 2 outgoing from s



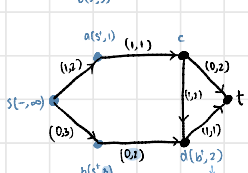
scan a or b

step 2 outgoing from a



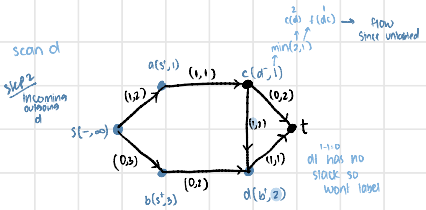
scan b

step 2 outgoing from b



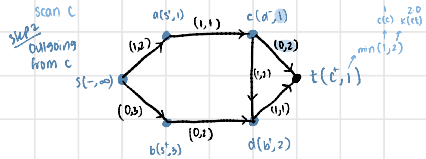
scan d

step 2 incoming outgoing d

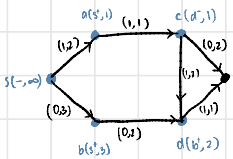


scan c

step 2 outgoing from c



step 3 as t is now labeled, find chain s-t by backtracking slacks from t

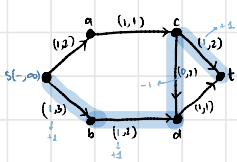


$t \leftarrow c \leftarrow d \leftarrow b \leftarrow s$ = sbdct

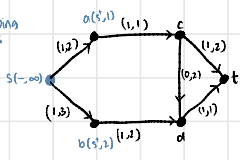
step 4 $t(\delta) = 1$

increase all outgoing by 1 (sb, bd, ct)

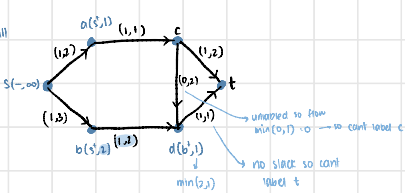
decrease all incoming by 1 (ca)



step 2 outgoing from s



step 3 scan all labels as before



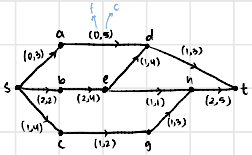
Since no more vertices to label

max flow: incoming flow to t

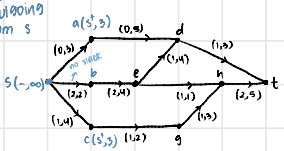
$= 1 + 1$

$= 2$

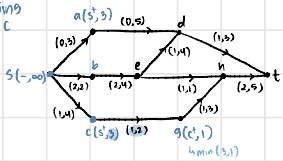
Q)



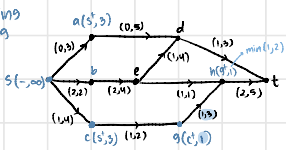
step 1,2 outgoing from s



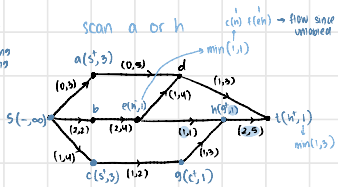
step 2 outgoing from c — scan a or c



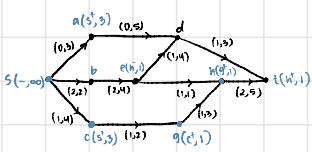
step 2 outgoing from g — scan a or g



step 2 incoming outgoing h — scan a or h

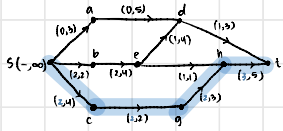


step 3 as t is now labeled, find chain s-t by backtracking slacks from t

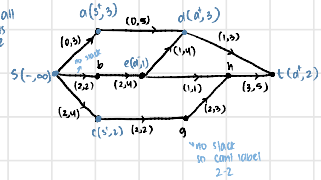


t ← h ← g ← c ← s = scant

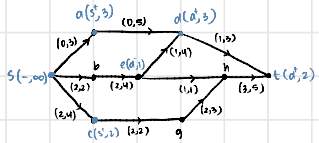
step 4 — t(δ) = 1
increase all outgoing by 1
decrease all incoming by 1



step v scan all labels as before

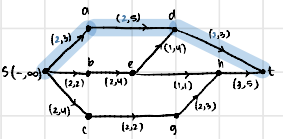


step 3 as t is now labeled, find chain s-t by backtracking slacks from t

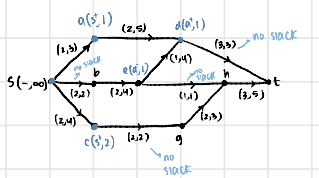


t ← d ← a ← s = sadt

step 4 — t(δ) = 2
increase all outgoing by 2
decrease all incoming by 2



step v scan all labels as before



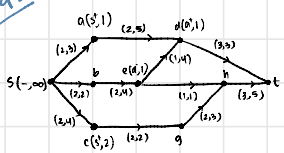
Since no more vertices to label
max flow: incoming flow to t
= 3 + 3
= 6

MAX FLOW MIN CUT

STEPS

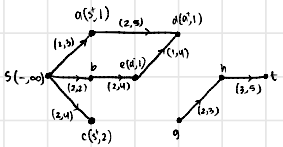
1. Apply Augmenting flow algo
2. Get max flow
3. find min cut such that max flow = min cut

Step 1 solved above



Step 2 max flow = 6

Step 3 min cut: (d,t) + (e,h) + (c,g)
= 3 + 1 + 2
= 6



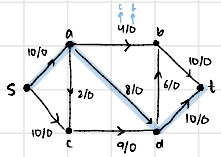
max flow = min cut
13 = 13

min cut (P, $\bar{P}$)

mas s, mas t

- deleting edges which will seperate the graph
- sum the flow of those cut edges

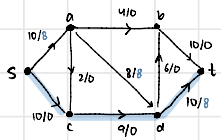
8)



bottle neck

sadt = 8

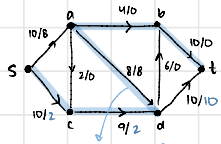
augment 8



k(dt): 2

scdt = 2

augment 2

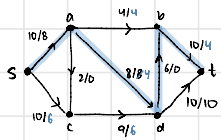


backward edge so subtract

k(ab): 4

scdabt = 4

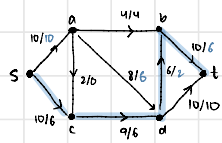
augment 4



k(sa)

sadbt = 2

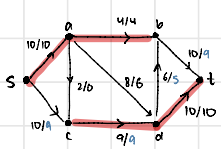
augment 2



k(cd)

scdbt = 3

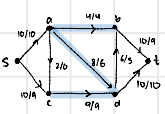
augment 3

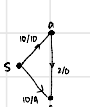


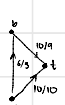
Since no other path possible

max flow: 8+2+4+2+3 = 19

min cut = $_{sa}^{4} + _{ad}^{6} + _{cd}^{9}$ = 19







Another way for Augmenting flow

↳ find path

↳ get bottleneck capacity → lowest capacity

↳ augment path → add bottleneck to flow of selected path

Bipartite Graph Matching

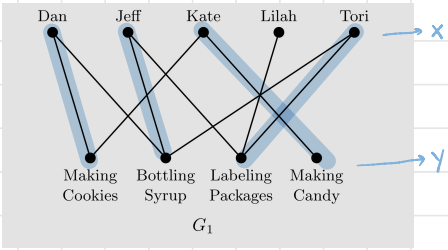
Matching (M)

↳ no edges share a node

↳ |M|: no of edges in matching

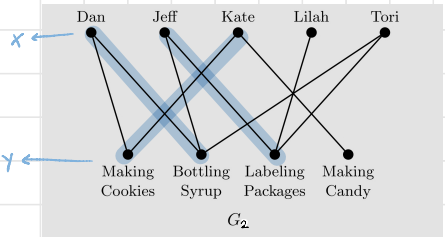
saturated vertice: has a matching

unsaturated vertice: has a no matching



saturated vertices: Dan, Jeff, Kate, Tori, Cookies, Syrup, Label, Candy

unsaturated vertices: Lilah



saturated vertices: Dan, Jeff, Kate, Cookies, Syrup, Label

unsaturated vertices: Lilah, Tori, Candy

maximal: M can't be enlarged by adding an edge e.g. G1, G2

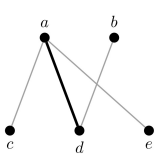
maximum: M is the largest size amongst all possible matching e.g. G1

perfect: M saturates very vertex → automatically maximal and maximum

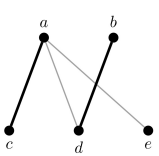
x-matching: M saturates every X vertex

y-matching: M saturates every y vertex e.g. G1

* every K-bipartite graph has perfect matching for all K>0



Maximal Matching of G_2



Maximum Matching of G_2

AUGMENTING PATHS

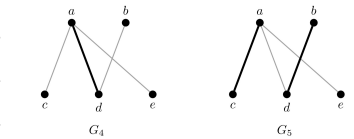
M-Alternating Path

↳ M edges in path alternate with non M edges

M-Augmenting Path

↳ M-alternating path

↳ start and end points of path aren't M → unsaturated



G_4 path: c a d b

↳ alternating

↳ augmenting → as c and b are unsaturated

G_5 path: c a d b

↳ alternating

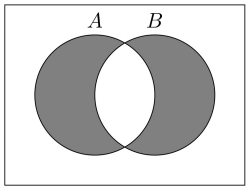
Symmetric difference

↳ $A \Delta B = (A-B) \cup (B-A)$

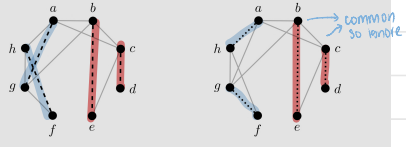
↳ non common

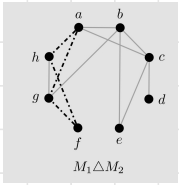
↳ has a matching

↳ every $A \Delta B$ is either a path or even cycle



Example 5.4 Below are two different matchings of a graph G. Find $M_1 \triangle M_2$.



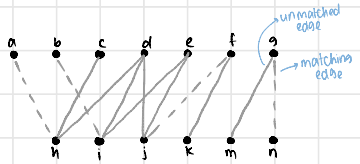


Augmenting Path Algo

↳ finds maximum M → no augmenting path

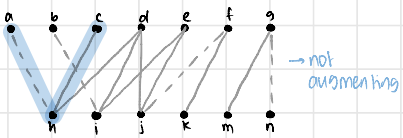
1. U = unsaturated vertices x
2. If U empty, M is maximum matching else select a vertice x from U
3. N(u) = y
4. if y is unsaturated, add xy to M
5. if y is saturated, find maximal M path xy being the starting edge

Q)



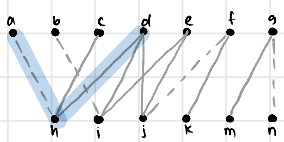
Step 1: U = c, d, e → not empty

Step 2: N(c) = h → unsaturated

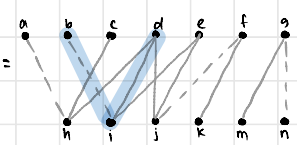


Step 2: N(d) = h, i, j

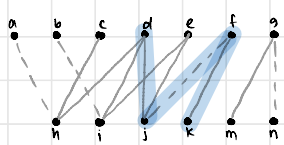
h =



i =



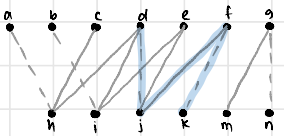
j =



Berges Theorem

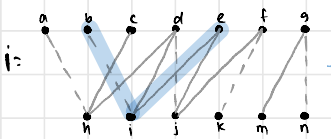
↳ M is maximum if G has no M-augmenting paths

use to show max matching

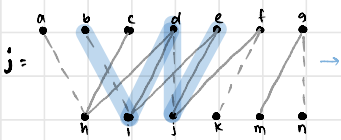


Step 2: N(e) = i, j

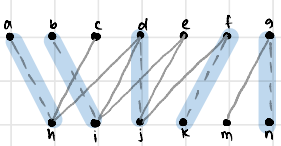
i =



j =



Maximum matching M' = (ah, bi, dj, fk, gn)



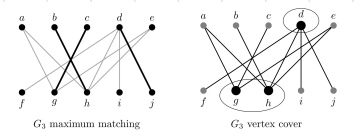
vertex cover Q

↳ every edge of G has atleast one endpoint in Q

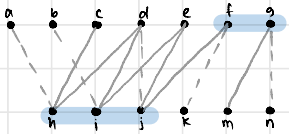
1. Apply Augmenting path algo
2. Define Q = unmarked X vertices and marked Y vertices
3. Q is min vertex cover

↳ more than 1 min cover can exist

- for a bipartite graph G,
  max |M| = min |Q|



1) after applying augmenting path algo

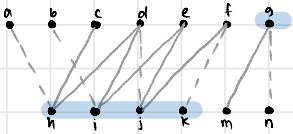


unmarked X = f, g

marked Y = h, i, j

Q = {f, g, h, i, j}, size = 5

2) after applying augmenting path algo



unmarked X = g

marked Y = h, i, j, k

Q = {g, h, i, j, k}, size = 5

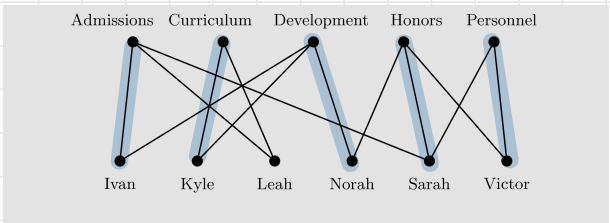
system of distinct representatives (SDR)

↳ r from each set S is distinct

Q) $S_1 = \{1,2,3,4\}$ $S_2 = \{2,4,8\}$, $S_3 = \{2,6\}$, $S_4 = \{4,8\}$

$r_1 = 1$, $r_2 = 2$, $r_3 = 6$, $r_4 = 8$ OR $r_1 = 2$, $r_2 = 4$ $r_3 = 6$, $r_4 = 8$

$r_1 = 1$, $r_2 = 2$, $r_3 = 6$, $r_4 = 2$ ✗



on page 217.

Theorem 5.4 (Hall's Marriage Theorem) Given a bipartite graph $G = (X \cup Y, E)$, there exists an X-matching if and only if $|S| \leq |N(S)|$ for any $S \subseteq X$.

→ missing

Question 10.[4 marks]

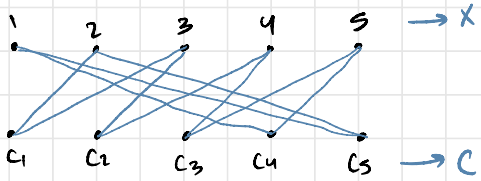
Consider the 3×5 Latin rectangle $L = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \\ 4 & 5 & 1 & 2 & 3 \end{pmatrix}$

Define a bipartite graph G with bipartition (X, Y) associated with L as follows:

(i) $X = \{1, 2, 3, 4, 5\}$,
(ii) $Y = \{C_1, C_2, C_3, C_4, C_5\}$, where C_i is the ith column of L,
(iii) 'i' in X is adjacent to 'C_j' in Y if and only if 'i' does not appear in 'C_j'.

(a) Draw the diagram of G.
(b) What is the degree of each vertex in X?
(c) What is the degree of each vertex in Y?
(d) Is G 2-regular?
(e) Does G contain a perfect matching?
(f) Display a perfect matching in G if your answer to (e) is 'yes'.

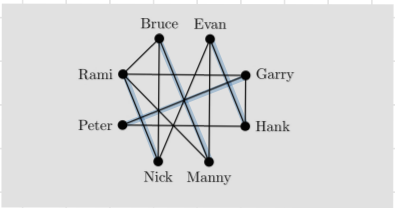
a)



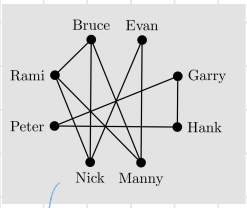
C1 = 1,5,4 = 2 3
C2 = 2 1 5 = 3 4
C3 = 3 2 1 = 4 5
C4 = 4 3 2 = 1 5
C5 = 5 4 3 = 1 2

5.2 matching in general graphs

	Bruce	Evan	Garry	Hank	Manny	Nick	Peter	Rami
Bruce	·	·	·	·	Y	Y	·	Y
Evan	·	·	·	Y	Y	Y	·	·
Garry	·	·	·	Y	·	·	Y	Y
Hank	·	Y	Y	·	·	·	Y	·
Manny	Y	Y	·	·	·	·	·	Y
Nick	Y	Y	·	·	·	·	·	Y
Peter	·	·	Y	Y	·	·	·	·
Rami	Y	·	Y	·	Y	Y	·	·



Example 5.10 Halfway through the canoe trip from Example 5.9, Rami will no longer share a canoe with Garry, and Hank angered Evan so they cannot share a canoe. Update the graph model and determine if it is now possible to pair the eight men into four canoes.



no possible perfect matching

graph is disconnected but can still have perfect matching

flower

↳ union of 2 M-alternating paths

↳ from an unsaturated vertex u to vertex v

↳ where 1 path has odd length and other even

stem

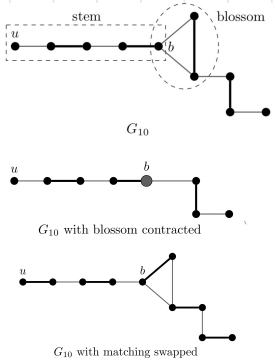
↳ maximal common initial path from u

↳ ends at a vertex b → base

blossom

↳ an odd cycle

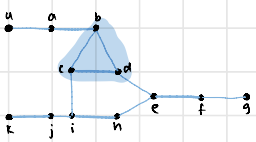
↳ obtained by removing stem from flower

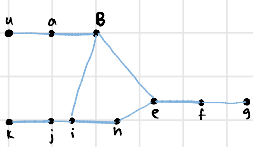


Edmond's Blossom Algo

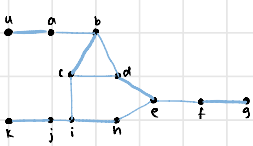
↳ finds max matching

1. Start traversing from u
2. If odd cycle, make a blossom
3. If augmenting path exists → unsaturated end points
   swap all edges





↳ u a B e f g is an augmenting path
↳ so swap all edges



maximum matching → as all vertices saturated

5.2

Stable matching

↳ no unmatched pair is unstable

unstable matching

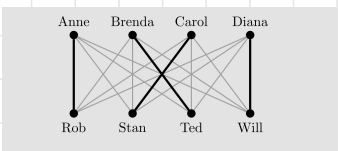
↳ if x any y are matched

but both rank the other higher than their current partner

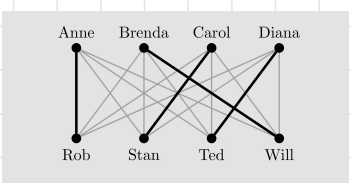
the x and y are an unstable pair

Example 5.13 Four men and four women are being paired into marriages. Each person has ranked the members of the opposite sex as shown below. Draw a bipartite graph and highlight the matching Anne–Rob, Brenda–Ted, Carol–Stan, and Diana–Will. Determine if this matching is stable. If not, find a stable matching and explain why no unstable pair exists.

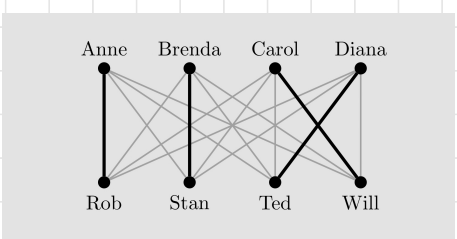
Anne:	t	>	r	>	s	>	w		Rob:	a	>	b	>	c	>	d
Brenda:	s	>	w	>	r	>	t		Stan:	a	>	c	>	b	>	d
Carol:	w	>	r	>	s	>	t		Ted:	c	>	d	>	a	>	b
Diana:	r	>	s	>	t	>	w		Will:	c	>	b	>	a	>	d



unstable as will and Brenda prefer each other over their current mate



unstable as will and carol prefer each other over their current mate



Will's first choice Carol, and vice versa

Anne only prefers Ted over Rob, but Ted prefers Diana over Anne

Brenda first choice is Stan

Diana either Rob or Stan over Ted but neither prefer Diana to their current mate

Gale Shapely Algo

↳ Stable matching of n women and n men

1. Write highest ranking of women for each man
2. If each women receives 1 proposal
   matching is stable
3. If women receives more than 1 proposal
   ↳ accept it if that she prefers that man over other available ✓
   and reject the rest ✗
   ↳ delay to highest rank ?
   and reject the rest ✗

Anne:	t > r > s > w		Rob:	a > b > c > d	
Brenda:	s > w > r > t		Stan:	a > c > b > d	
Carol:	w > r > s > t		Ted:	c > d > a > b	
Diana:	r > s > t > w		Will:	c > b > a > d	

a) Men proposing

Men	Women		
R	A ?		A ✓
S	A ✗ → as it over S		B ✓
T	C ✗		D ✓
W	C ✓		

a) Women proposing

Women	Men
A	T ✓
B	S ✓
C	W ✓
D	R ✓

Stable roomates

Emma:	l > m > z
Leena:	m > e > z
Maggie:	e > l > z
Zara:	e > l > m

E	L ?	L ?	L ?	
L	M ?	M ?	M ?	
M	E ?	E ?	E ?	
Z	E ✗	L ✗	M ✗	

not stable

Gale Shapely Algo (with unacceptable partners)

Anne:	t > r > s > w		Rob:	a > b > c > d	
Brenda:	w > r > t		Stan:	a > b	
Carol:	w > r > s > t		Ted:	c > d > a > b	
Diana:	s > r > t		Will:	c > b > a	

Men	Women			
R	A ?		A ?	A ✓
S	A ✗ → as it over S		B ✗ ← unacceptable pow	
T	C ✗		D ?	D ✓
W	C ✓			

Stable matching

S and B remain unmatched

k Factor

↳ spanning k-regular subgraph

↳ all degrees = k

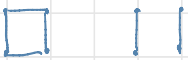
* 1 factor = perfect matching
* If G is 2k-regular, then G has a 2-factor

Spanning subgraph

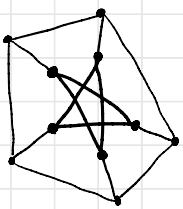
↳ same vertices

↳ doesn't matter connected / disconnected

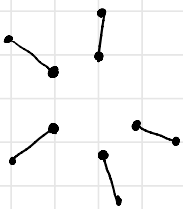
↳ subset edges



Q)



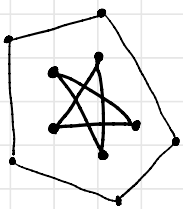
K_3



1 Factor

↳ spanning subgraph → same vertices

↳ 1-regular subgraph → all vertices have 1 degree



2 Factor

↳ spanning subgraph

↳ 2-regular subgraph

k-Factorization

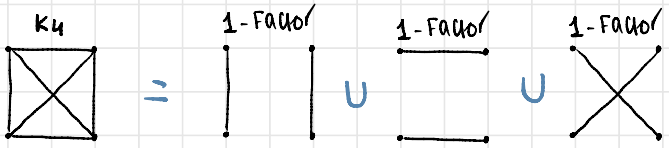
↳ partion the edge set into disjoint k-factors

↳ no same edges

↳ same vertices

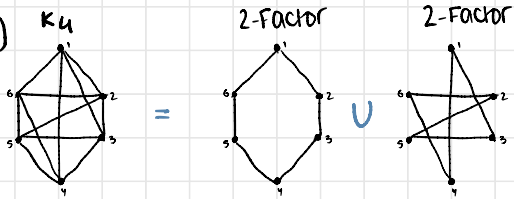
* A graph has k-factorization only if G is 2k-regular
* every k-bipartite graph has 1 factorization

Q)



→ 1 factorization

Q)



→ 2 factorization