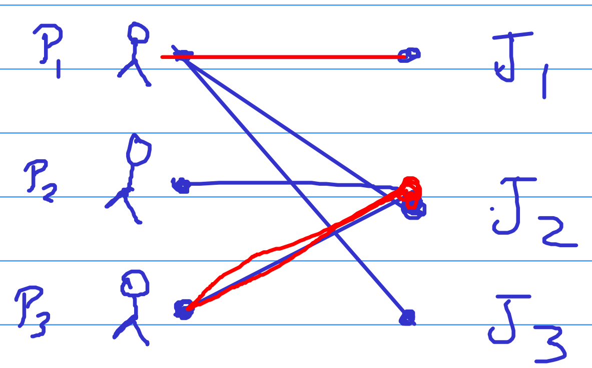
Sep 23

Bipartite Matching / Assignment problem

- Around a hundred years old
- Algorithms 1950's
- Influential topic in combinatorics and CS
- Linear programming primal dual.
- NP and coNP
- Randomized algorithm, parallel algorithms online algorithms, counting algorithms
- Connected to other problems like max flow min cut

Problem Definition: There n people, n jobs.

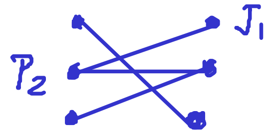
A person is suitable for only a subset of jobs. Assign one job to each person.



Assignment should be one-one. (at most one)

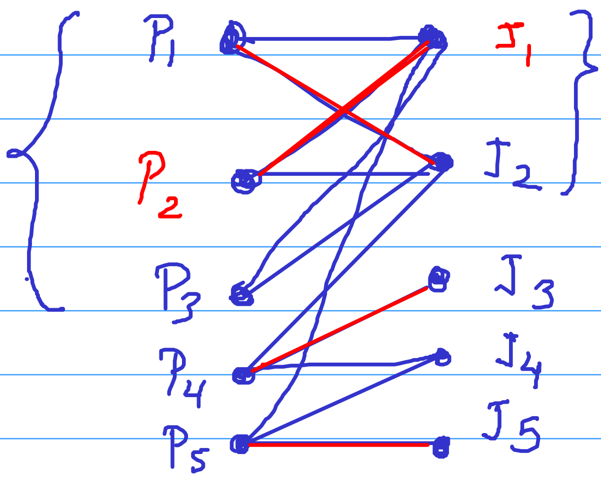
Maximize the number of jobs assigned.

Sep 27



Bipartite Graph: a graph where $V = V_1 \cup V_2$ and each edge connects a vertex in V_1 to a vertex in V_2.

Matching: In a graph $G(V, E)$, a subset of edges $M \subseteq E$ is called a matching if no two edges in M share a common vertex.



Max matching = 4

Problem: Given a bipartite graph, find a maximum size matching.

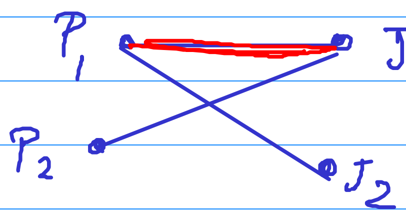
Applications:

- TA allocation
- Course $\underbrace{\text{room + slot}}$ allocation
- Taxi
- Car pooling / bikepooling
- Kidney donations

Indirect applications: ① Airline Scheduling
② Chinese Postman problem

Approaches:

① Match with the first thing you see.



② Programming Assignment 1 was a special case of bipartite matching.

people with intervals —— jobs at given times.

Can we process both the sides in certain orders and keep matching the first available vertex?

Order of their degrees.

completely fine.

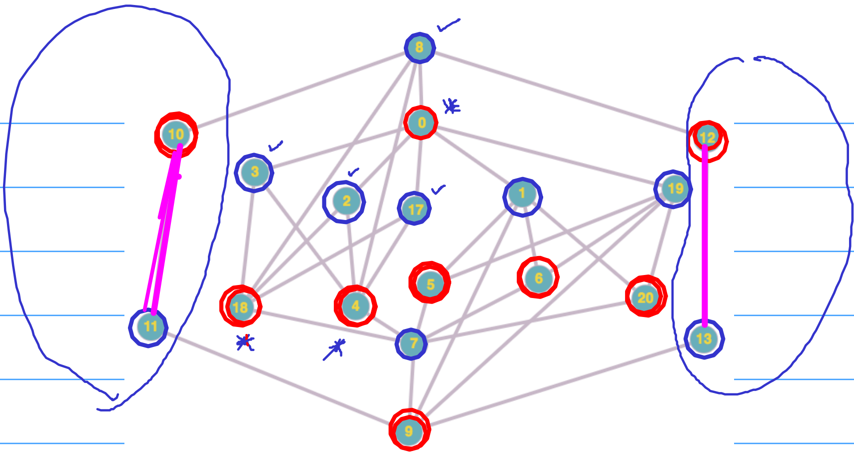
Claim:
- match all degree one vertices with neighbors.
- find a maximum matching in the remaining graph
- Together it's a maximum matching in the original graph.

HW

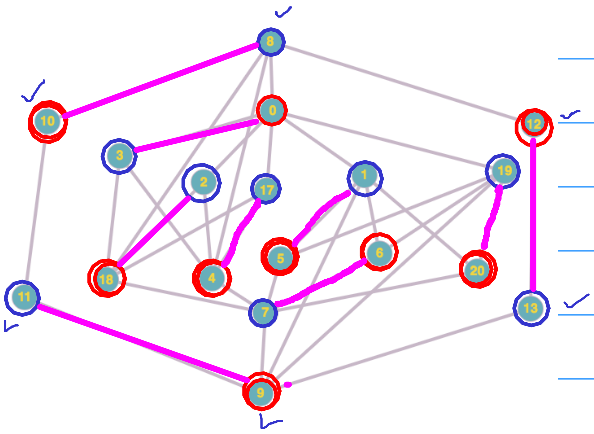
$G \xrightarrow{\text{match deg 1}} \xrightarrow[\text{matched}]{\text{remove}} H$

$\text{Optimal}(G) \longleftarrow \text{Optimal}(H)$

Matching deg 2 vertices first.



After matching (10, 11) (12, 13), note that the blue vertices {8, 3, 2, 17} have in total only three available neighbors. Hence, something must remain unmatched.



Everyone is matched.

Perfect matching.

This example demonstrates that the greedy strategy "matching smaller degree first" doesn't work.

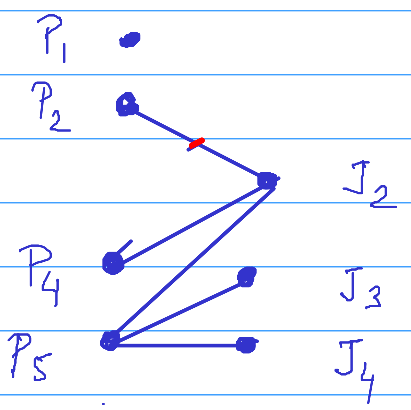
Dynamic Programming:

A matching can be of

- Job 1 $\longrightarrow$ P_1
- Job 1 $\longrightarrow$ P_2

  $\vdots$ $\quad\quad$ $\vdots$
- Job 1 $\longrightarrow$ P_n
- Job 1 $\longrightarrow$ unassigned.

Can we find the optimal solution in each of these cases recursively?



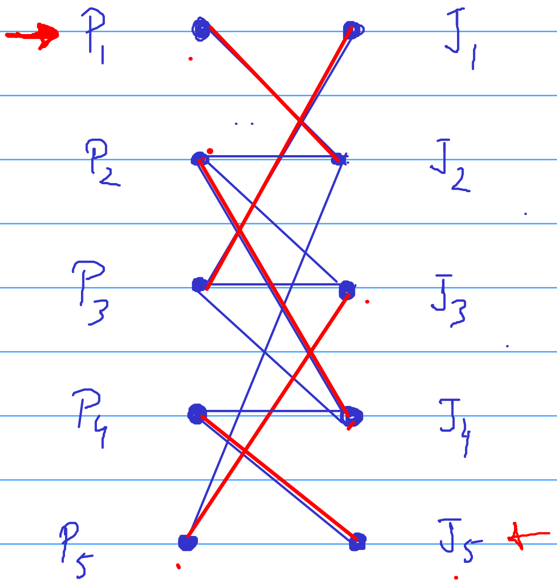
J_1 is matched with P_1 $\longrightarrow$ remove both vertices

Subproblem at k th recursive call

P_{i_1}	J_{k+1}	Exponentially many distinct subproblems.
P_{i_2}	J_{k+2}	
$\vdots$	$\vdots$	
P	J_n	

Augmenting Approach

Try to increase the size of the matching by adding and possibly removing some matching edges.



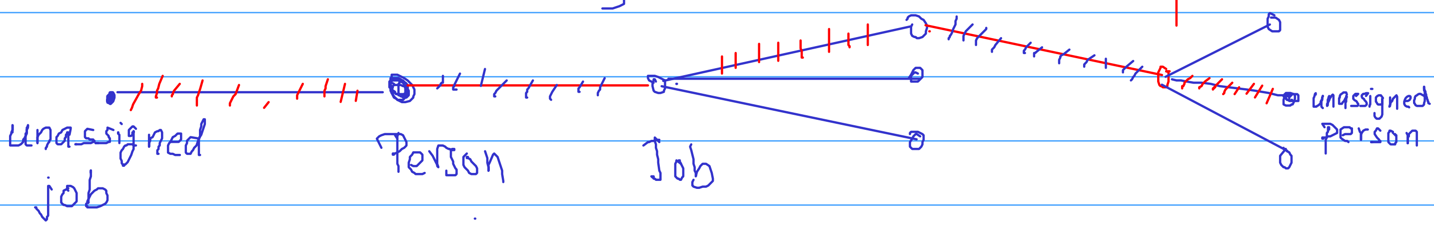
J_1 — P_3,

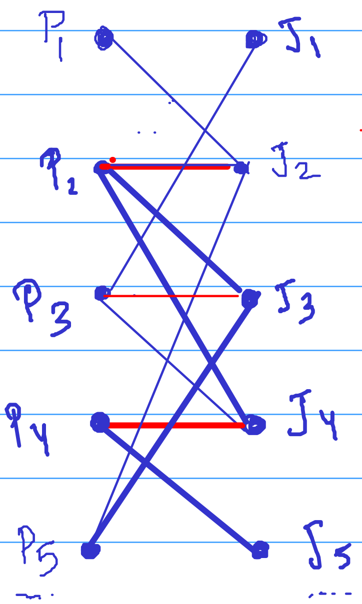
J_2 — P_1, P_2, P_5

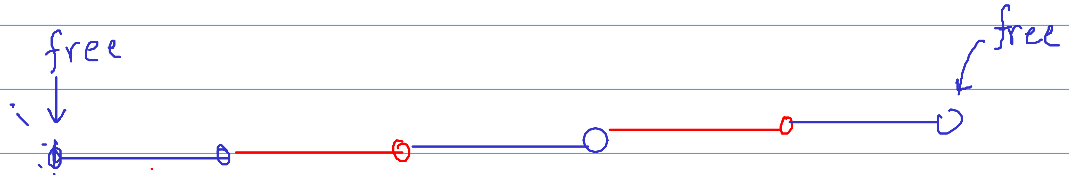
J_3 — P_2, P_3, P_5

J_4 — P_2, P_3, P_4

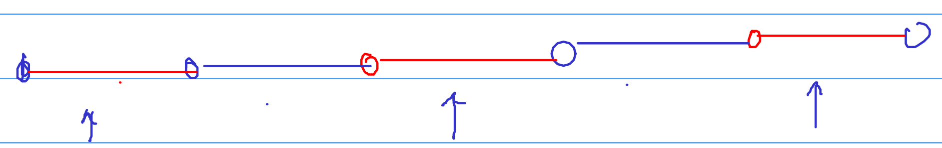
J_5 —





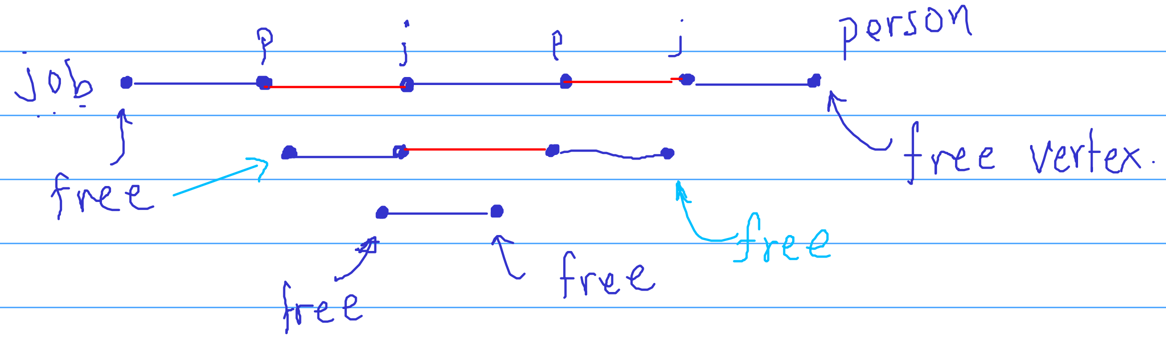


augmenting the matching.



Augmenting Path (with respect to the current matching)

A path that
- starts with a free vertex,
- alternates between matching and non-matching edges
- ends with a free vertex.



If an augmeting path is found then the matching can be augmented by swapping matching and non-matching edges on the path.

Two questions

- Is there always an augmenting path?
  (yes, if current matching is not max)
- If there is one how to efficiently find it.

Algorithm at a High level

- Always maintain a matching
- Try to find an augmenting path.
  If found, augment the matching.
- Keep repeating till a point when no augmenting path can be found.

equivalently, if there is no augmeting path, the M is max.

Lemma 1 (existence) Let M be a matching in a graph G. If M is not maximum then there exists an augmenting path with respect to M.

Proof: Let M^* be a maximum matching.

M (red)

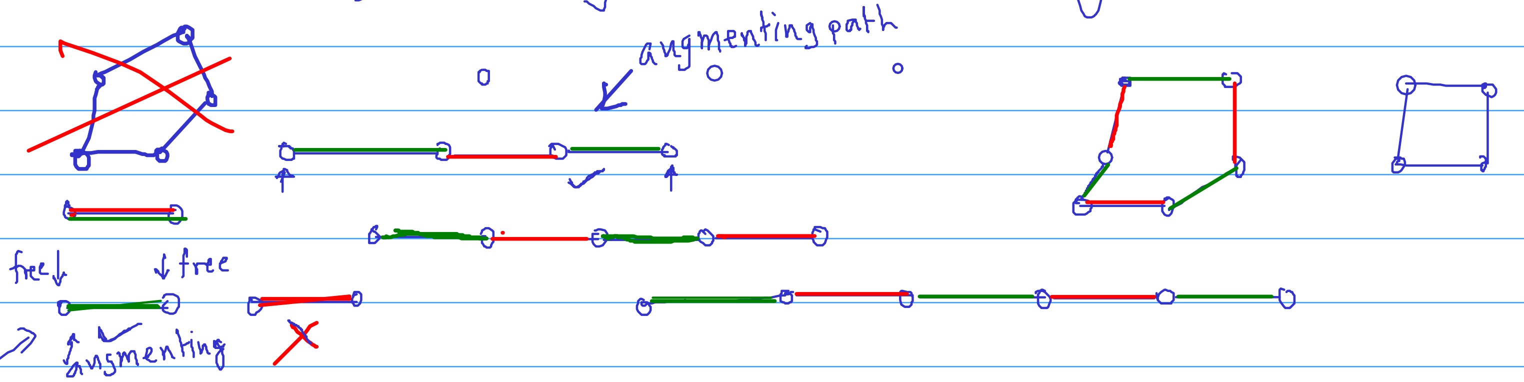
M^* (green)

Take union of M and M^*

[a vertex can't have two green or two red edges.]

Claim: It will be a collection of alternating paths and cycles.

Obs: In $M \cup M^*$, every vertex has degree 2 or less.



A vertex can have at most one edge from M and at most one edge from M^*.

Thus, the edges must alternate between M and M^*.

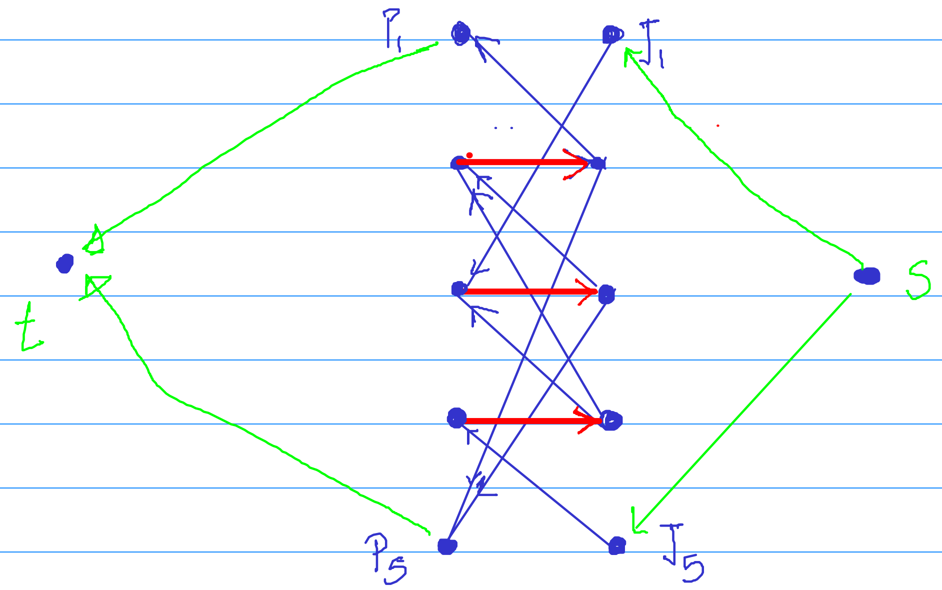
Since $|M^*| > |M|$, there must be a path having more edges from M^* than M.

That's an augmenting path w.r.t. M.

Sep 30

Efficient algorithm to find an augmenting path?

can some standard path finding algorithm be used?



- Direct all the matching edges from left to right.
- Direct all the non-matching edges from right to left.

This ensures that any directed path alternates between matching and non-matching edges.

Create a source s and destination t.

→ edges from s to free vertices on the right

→ edges from free vertices on the left to t.

Find a directed path from s to t.

Claim 1 (HW): any directed path from s to t gives us an augmenting path.

Claim 2 (HW): Any augmenting path in the original graph gives us a directed path from s to t.

Running time:

$$|\text{maximum matching}| \times O(|E|) = O(|V| \cdot |E|).$$

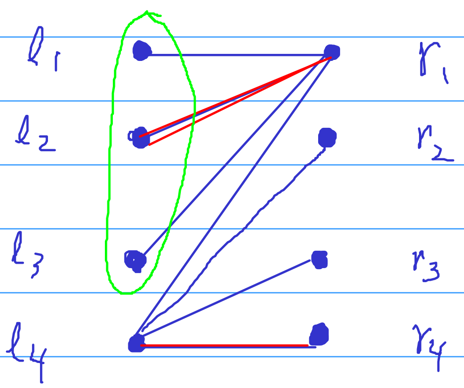
Better $O(|E| \cdot \sqrt{|V|})$

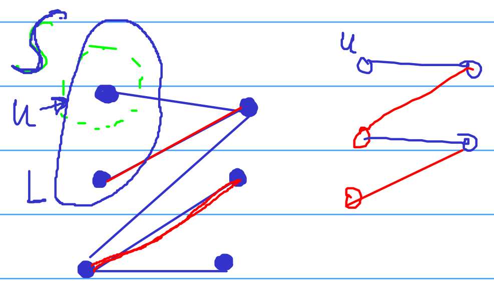
Hall's theorem: Let $G(L \cup R, E)$ be a bipartite graph. If there is no matching that covers all the vertices in L then

$$\exists\ S \subseteq L, \quad \text{s.t.} \quad |N(S)| < |S|$$

$N(S)$ ← neighbors of S

"easily verifiable proof for the non-existence of a matching covering all vertices in L"





Proof strategy: Take a maximum matching. Some vertex u in L must be free.

If we try to find an augmenting path starting from u, we should fail.

That means all alternating paths starting from u must end in L.

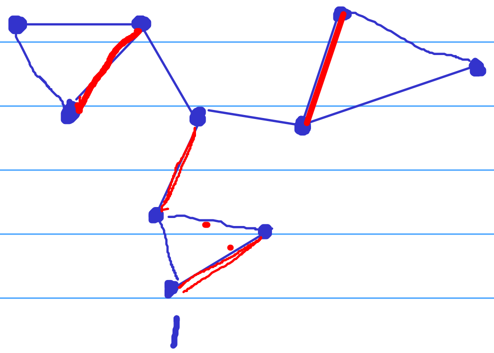
HW Use this fact to construct the desired set S.

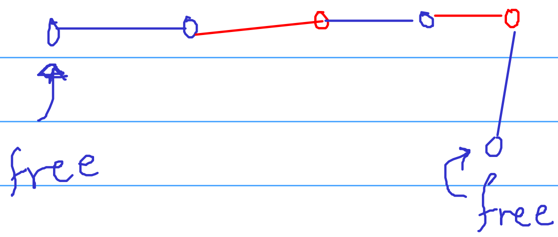
Generalized Version: Let $G(L \cup R, E)$ be a bipartite graph. Let maximum matching size in G be $|L|-k$. Then

$$\exists\ S \subseteq L, \quad \text{s.t.} \quad |N(S)| = |S| - k$$

Matching in General Graphs.

Roommate allocation: n students, given a set of pairs of students who agree to share a room, find maximum number of disjoint pairs.





Does the same augmenting path algorithm work?

What works: If M is not a maximum matching then there exists an augmenting path.

What is not clear: how to find an augmenting path.

Standard path finding algorithms will not work.

Edmonds [1965]

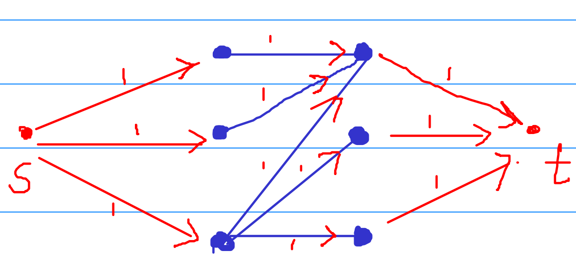
→ first to define polynomial time as efficient

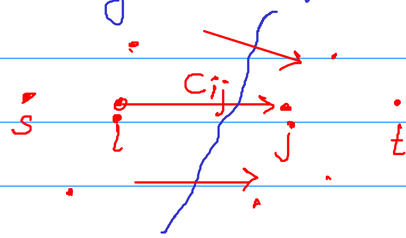
Oct 4

Problems Related to bipartite matching

- Max flow min cut
- max no. of edge disjoint paths
- Taxi scheduling.

- Bipartite matching can be solved using Max flow



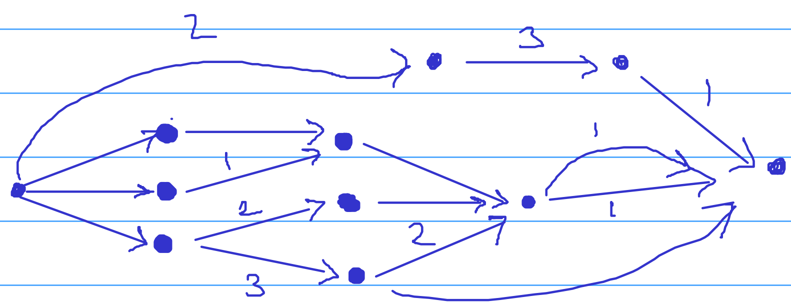


- min cut is related to Hall's block.

$$\text{Max flow} = \text{Min cut}$$

$$\text{Max matching} = |L| + \min_{S \subseteq L}\left(\underbrace{|N(S)| - |S|}\right)$$

- Max flow can be solved using bipartite matching



HW

Taxi Scheduling

A taxi company gets a list of bookings for the next day.

Want to minimize the number of taxis required.

Bookings

B_1 : 9:00 Chembur → Airport 10:00

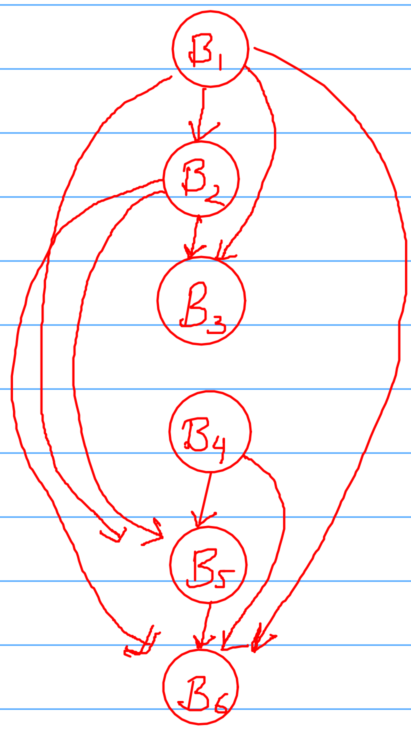
B_2: 10:30 Andheri → IITB 11:15

B_3 : 11:30 IITB → TIFR 1:30

B_4 : 10:15 Dadar → Airport 11:15

B_5 : 12:00 Powai → LTT 12:45

B_6 : 13:00 Chembur → Sion 13:30

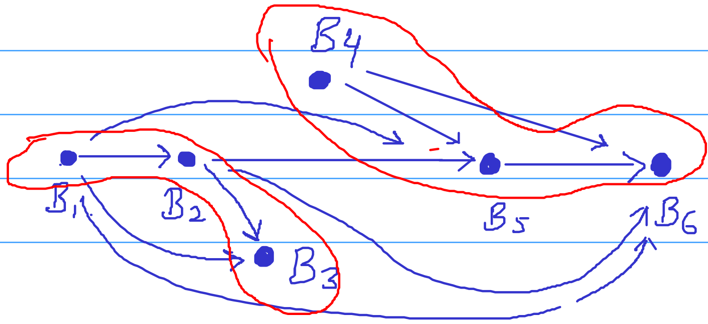


Input: Directed graph — acyclic
— transitively closed

(Partial Order on a set of elements)

$i \to j \to k$

Output: Partition of vertices into minimum number of paths.



Taxi 1 B_1 B_2 B_5

Taxi 2 B_4 B_6

Taxi 3 B_3

Taxi 1 — B_1 B_2 B_3

Taxi 2 — B_4 B_5 B_6

Bottleneck: something that forces the number of taxis to be at least k.

set of k Bookings with no edges among them.

min number of taxis = maximum size of a bottleneck (non-trivial)

Idea 1: Find the longest path and assign a taxi. Recurse.

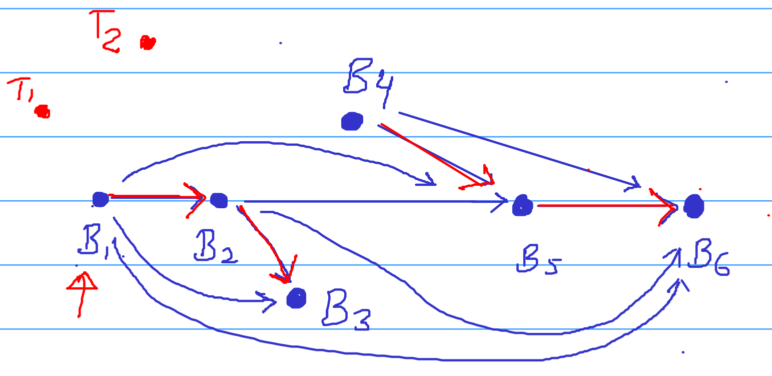
doesn't work on the above example.

Idea 2: Process the bookings in some order. Assign the first available taxi.

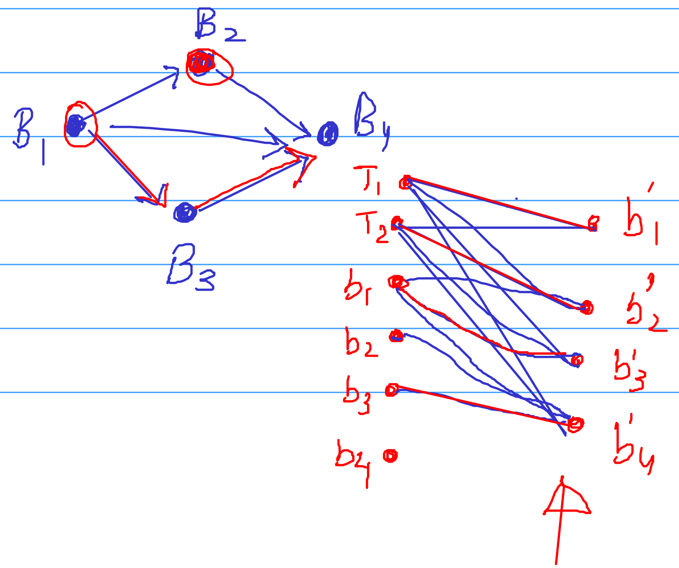
doesn't work HW

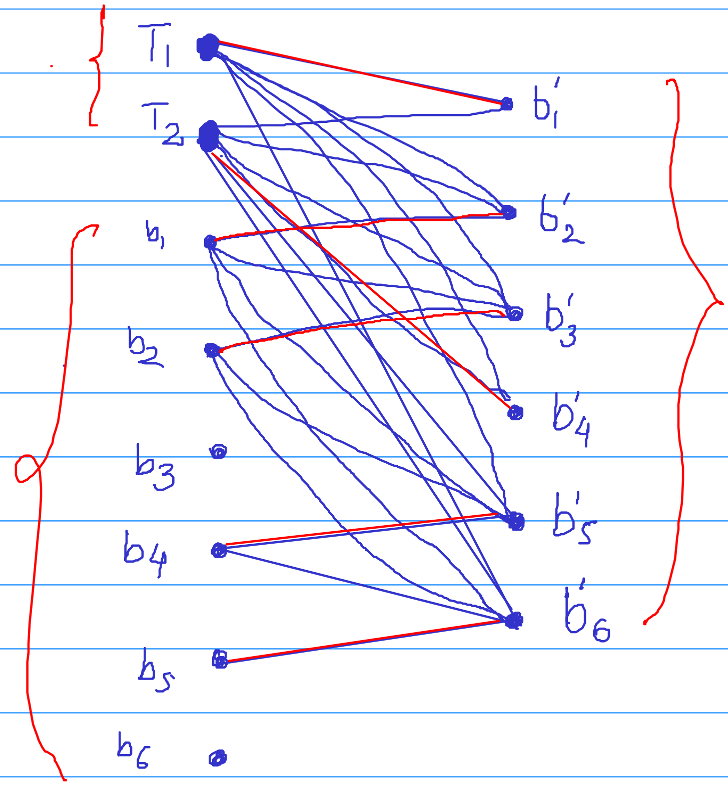
Convert the problem into bipartite matching.

$k=2$



D





$n+k$ G_k n

Oct 5

For the given directed graph D, construct a bipartite graph G_K

leftside vertices = bookings $\{b_i\}$ + k taxis $\{T_i\}$

right side vertices = bookings $\{b_j'\}$

Add edges (T_i, b_j') in G for all k taxis and bookings

Add edge (b_i, b_j') in G if (B_i, B_j) is an edge in D.

Claim 1: If an assignment with k taxis is possible then there is a matching in G_K that matches all right side vertices.

Claim 2: For any matching in G_K that matches all right side vertices, we can convert it into an assignment of $\leq k$ taxis.

Algorithm: input directed graph D.

For each value of $1 \leq k \leq n$

→ construct the bipartite graph G_K from D

→ Find max matching in G_K. (any known algorithm)

→ If all right side vertices are matched then convert it into a taxi allocation (claim 2) with k taxis

Break.

Proof of Claim 1:

Let the allocation with k taxis be

Taxi 1 — $B_{i_1} \to B_{i_2} \to \cdots$
Taxi 2 — $B_{j_1} \to B_{j_2} \to \cdots$
$\vdots$
Taxi k —

We will construct the following matching in G_k

→ For each $1 \leq j \leq k$

- if the first booking served by taxi T_j is B_ℓ

  then match (T_j, b'_ℓ) in G_k

$T \cdots B_q \to B_p \cdots \to B_r$

→ For any booking B_p, if it is served by a taxi immediately after B_q

then match (b_q, b'_p) in G_k $\qquad$ b_r unmatched

Argue that it's a valid matching

- any vertex is matched with at most one

Argue that right side vertices are all matched.

Proof Claim 2:

Let M be a matching in G_k that matches all right side vertices

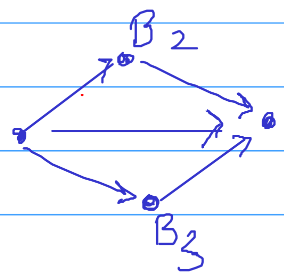
If $(T_i, b_j') \in M$

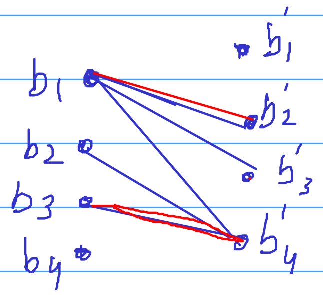
then allocate B_j as the first booking for taxi i

If $(b_p, b_q') \in M$ $\quad T \cdots B_p \to B_q$

then allocate booking B_q to be served immediately after B_p by the same taxi.

Argue that every booking got scheduled.





Back to the algorithm

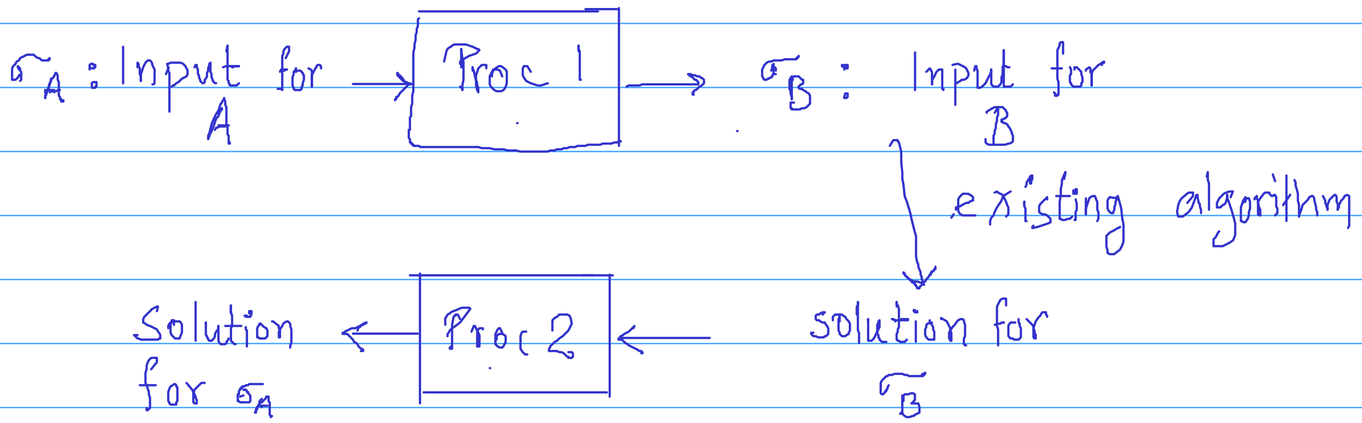
Do we need to try all possible values of k?

Construct Bipartite graph G without the taxi vertices

max matching $\to$ no. of unmatched vertices on right side
$=$
HW min number of taxis required.

Reduction:

- Problem A reduces to Problem B
- $\underline{\underline{A \leq B}}$ (Taxi allocation $\leq$ bipartite matching)
- A can be solved using B.



Proof of correctness involves two things:

1. If there is a solution for σ_A then there exists a solution for σ_B

2. Any solution for σ_B can be converted to a solution for σ_A.