2. Matching

- Maximum bipartite matchings
- Stable marriage problem

Matching in graphs

given a graph $G = (V, E)$,

a **matching** $M = \{e_1, \ldots\} \subseteq E$ is a subset of edges such that no two edges share a node

equivalently, a matching is a subgraph of G where the degrees of nodes are at most one

a node is **matched** or **used** if it has degree one in M
otherwise we say the node is **free** of **unused**

similarly, an edge is either matched or free (used or unused)

a matching is **maximum** if it has maximum cardinality among all matchings

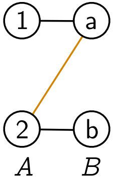
a matching is **maximal** if no edge can be added to the matching while preserving the matching property

a **perfect matching** is a matching that matches all nodes

a graph is **bipartite** if there is a partition $V = A \cup B$ such that all edges in E are between A and B

Maximum Bipartite Matching

consider a scenario where each person give a subset of hospitals that he is willing to work at, and hospitals a subset of candidates



$A = \{1, 2\}$: candidates

$B = \{a, b\}$: hospitals

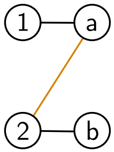
$E = \{(1, a), (2, a), (2, b)\}$

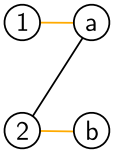
$M = \{(2, a)\}$

consider a bipartite graph $G = (A, B, E)$

- a **matching** is a subset of edges $M \subseteq E$, such that no two edges share a node
- **size** of a matching is the number of edges in the matching

problem. find a maximum (cardinality) bipartite matching unlike the MST problem, a greedy algorithm fails





$A = \{1, 2\}$: candidates

$B = \{a, b\}$: hospitals

$E = \{(1, a), (2, a), (2, b)\}$

$M_1 = \{(2, a)\}$

$M_3 = \{(1, a), (2, b)\}$

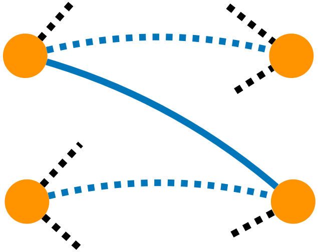
we focus on the bipartite matching example

- ★ an **alternating path** with respect to M is a simple path in G whose edges alternate between used and unused
- ★ an **augmenting path** with respect to M is an alternating path that starts and ends at unmatched nodes (with odd lengths)

algorithm for finding a maximum bipartite matching

- ★ start with $M = \{\}$
- ★ while there is an augmenting path P, replace M by $M \oplus P$
- ★ when there is no augmenting path, return M

$$M \oplus P = \{(i,j) \,|\, (i,j) \in M \setminus P \text{ or } (i,j) \in P \setminus M\}$$



length (2k+1) augmenting path has
k edges in M and
k+1 edges not in M

fact 1. each time we take the symmetric difference $M \oplus P$, the size of M increases by one

fact 2. augmenting path operation preserves matchings

Q. does this ensure correctness of the algorithm?

fact 1. each time we take the symmetric difference $M \oplus P$, the size of M increases by one
proof. an augmenting path P of size $2k+1$ has k used edges and $k+1$ unused edges, i.e. $|P \cap M| = k$, and $|P \setminus M| = k+1$. taking the symmetric difference changes used edges to unused and vice versa.

fact 2. augmenting path operation preserves matchings
proof. matching is a graph where all nodes degrees are at most one. after taking the symmetric difference, the interior nodes of the path remain degree one. The two end points' degree increase by one. But by definition an augmenting path starts and ends at unmatched nodes (with degree zero in the matching), and hence consequently have degree one after symmetric difference. taking the symmetric difference

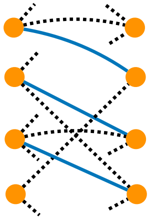
correctness

theorem.[Berge 1957] if the current matching M does not have the maximum cardinality, then there exists at least one augmenting path (notice this is also true for non-bipartite graphs)

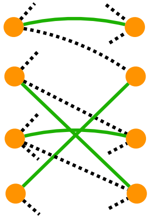
proof of correctness. we prove by constructing an augmenting path. Since M is not maximum matching, let M^* be the maximum matching

$$|M| < |M^*|$$

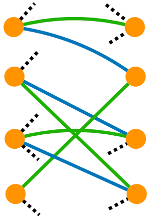
Consider the symmetric difference $M_\Delta \triangleq M \oplus M^*$, which contains the edges appear either one of M or M' but not both.



M

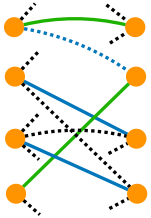


M*



M ⊕ M*

There exists at least one alternating path



M'

|M'| > |M|

proof of correctness. we prove by constructing an augmenting path.

Since M is not maximum matching, let M^* be the maximum matching

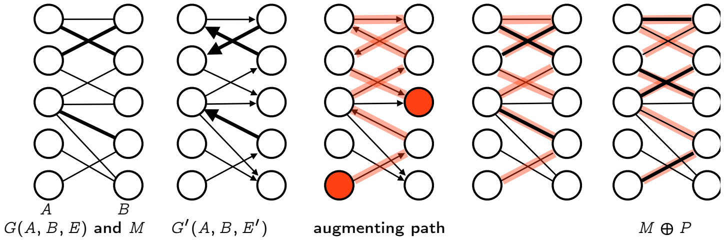
$$|M| \; < \; |M^*|$$

Consider the symmetric difference $M_\Delta \triangleq M \oplus M^*$, which contains the edges appear either one of M or M' but not both.
Each node in M_Δ has degree at most two.
So M_Δ consists of simple paths and cycles.
A *cycle* has even number of edges and those edges alternate between being in M and in M^* (since M and M' have degree at most one).
Since $|M^*| > |M|$, there must be at least one *path* P with more edges from M^* than M.

P is an augmenting path w.r.t. M, since it has odd cardinality (more number of edges from M^*), alternating edges (symmetric difference), and starts and ends at unused nodes (if it was used by M then the symmetric difference would be an isolated node with degree zero or degree two).

how do we find an augmenting path?

concretely, find an augmenting path by a modified breadth first search



first, given a bipartite graph $G = (A, B, E)$ and a matching M, define a directed graph $G' = (A \cup B, E')$ such that

$$E' = \{(i,j) \in E \setminus M \mid i \in A, j \in B\} \cup \{(i,j) \in M \mid i \in B, j \in A\}$$

modified BFS for finding augmenting paths

- ⋆ for each (unused) node a in A, repeat
- ⋆ use BFS on the directed graph G' to find a path P starting from a and ends at an unused node in B
- ⋆ if P found, then output P

complexity

- ⋆ if a node is used, it stays used. Each augmenting path increases the number of used nodes by two. So we have to do $\min(|A|, |B|)$ augmentations
- ⋆ finding an augmenting path requires a BFS, and this takes $O(|E|)$ and we need to repeat BFS starting from each of the $\min(|A|, |B|)$ nodes
- ⋆ total complexity is $O(|E| \min(|A|, |B|)^2)$

unweighted bipartite matching is the simplest problem

weighted bipartite graphs

- ⋆ weighted augmenting paths can solve the problem
- ⋆ also, Hungarian algorithm solves the problem in time $O(|V|^2 \log |V| + |V||E|)$

general (non-bipartite) graphs

- ⋆ Edmonds' algorithm solves the problem in time $O(\sqrt{|V|}\,|E|)$

Maximum bipartite matching example

flight scheduling

- You are planning the flight schedule for a airline for one day. You have two pieces of information at your disposal:
- The time needed for a direct flight between any two airports. For example, you might be told that the time needed for a flight from New York to San Francisco is 7 hours.
- A list of all desired flights that must run throughout the year. For each flight you know its origin, destination and departure details (times are given in GMT). For example, a flight may be characterized by the 3-tuple "New York / San Francisco / GMT 11:30". Note that the arrival time can be found from the first piece of information: this flight arrives at San Francisco at GMT 18:10 on the same day.
- For simplicity, assume that it is possible for a plane to depart an airport on another flight immediately after it has arrived on a previous flight (i.e., there is no delay/service time between flights). Planes may also make flights that are not in the flight list.

We want to determine the minimum number of planes the company needs to purchase, in order to fulfill all its flight requirements. Formulate this as a maximum bipartite matching problem

- JFK-SFO : 10:00-17:00
- CMI-ORD : 8:00-9:00
- CMI-ORD : 19:00-20:00
- ORD-CMI : 18:00-19:00
- ORD-JFK : 9:00-13:00
- ORD-SFO : 10:00-16:00
- SFO-ORD : 17:00-23:00
- JFK-CMI : 14:00-18:00

Maximum bipartite matching and minimum vertex cover

- Maximum independent set / minimum vertex cover problem

  An **independent set** of a graph G is a set of nodes S such that no pair of nodes in S are joined by an edge in G.

  Q. Given a bipartite graph G, find the maximum independent set.

  A **vertex cover** of a graph G is a set of nodes $K \subseteq V$ such that each edge of G has at least one end point in K.

  Q. Given a bipartite graph G, find the minimum vertex cover.

  theorem. S is an independent set iff $V \setminus S$ is a vertex cover

  proof.

  S is an independent set

  $\iff$ for all pairs of nodes i and j in S, $(i, j) \notin E$

  $\iff$ for all $(i, j) \in E$, either $i \notin S$ or $j \notin S$

  $\iff$ $V \setminus S$ is a vertex cover

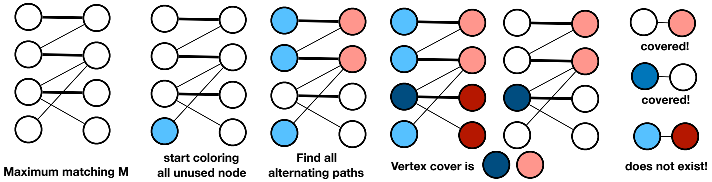
theorem. (König's theorem) the size of a maximum bipartite matching equals the size of the minimum vertex cover

$$|M^*| = |K^*|$$

- **proof.**

fact 1. Clearly, $|M| \leq |K|$ for any matching M and any vertex cover K. Therefore, it follows that the inequality is true also for the maximum matching and the minimum vertex cover: $|M^*| \leq |K^*|$.

claim 1. What is surprising is that the other direction is also true. To prove the other direction, we want to find K such that $|K| \leq |M^*|$. Then, $|K^*| \leq |K| \leq |M^*|$ and we are done proving $|M^*| = |K^*|$. A naive guess will be to take one node from an edge in the matching, but how?



fact 1. all **light red** and **dark blue** are USED

fact 2. no matching connects both **light red** and **dark blue**

hence, number of matching $|M^*|$ is at least as large as number of $|K|$.

theorem. (König's theorem) the size of a maximum bipartite matching equals the size of the minimum vertex cover

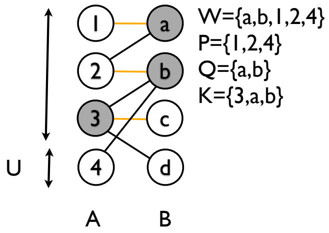
$$|M^*| = |K^*|$$

- **proof (formal).**

fact 1. Clearly, $|M| \leq |K|$ for any matching M and any vertex cover K. This is because for K to be a vertex cover, it needs to cover the edges in $M \subseteq E$ at least. And since M is a matching, no edgesin M share the same node. So the number of nodes necessary to cover M is at least $|M|$. And to cover all the edges in E requires more nodes than that. Therefore, any vertex cover K has more nodes than any matching M.

Therefore, it follows that the inequality is true also for the maximum matching and the minimum vertex cover: $|M^*| \leq |K^*|$.

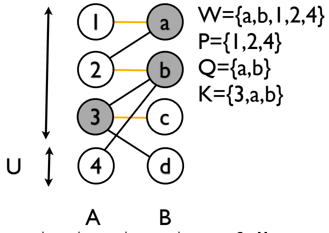
What is surprising is that the other direction is also true. To prove the other direction, we want to find K such that $|K| \leq |M^*|$. Then, $|K^*| \leq |K| \leq |M^*|$ and we are done proving $|M^*| = |K^*|$. A naive guess will be to take one node from each edge in the matching. Let's make this formal.



- ★ Let U be the set of unused nodes in A
- ★ Let W be the set of nodes reachable from U via alternating paths (including U)
- ★ Let $P = W \cap A$
- ★ Let $Q = W \cap B$

fact there cannot be an edge from P to $B \setminus Q$, since the endpoint in $B \setminus Q$ would then be in W and thus in Q

- ▸ Then, $K = Q \cup (A \setminus P)$ is a vertex cover, since
  - ★ edges between P and Q are covered by Q
  - ★ no edges between P and $B \setminus Q$
  - ★ edges between $A \setminus P$ and B are covered by $A \setminus P$



We are left to show that $|K| \leq |M^*|$. It follows directly from the following claims.

- $A \setminus P$ are used (by definition)
- all nodes in Q are used (otherwise we have an augmenting path, and the matching is not maximum)
- there is no matched edge between Q and $A \setminus P$ (otherwise the left endpoint of that used edge must be included in P)

Therefore, the number of edges in M^* is at least the number of nodes in K

$$|K| \leq |M^*|$$

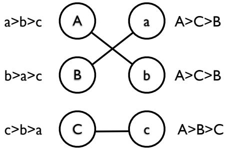
The stable marriage problem

stable marriage problem

- there are n hospitals $\{a, b, c, \ldots\}$
- there are n residents $\{A, B, C, \ldots\}$
- each hospital submits its list of preferences, e.g. $H > B > R > \ldots$
- each resident submits its list of preferences, e.g. $d > f > b > g > \ldots$
- all hospitals (or residents) have to be ranked in each preference list

problem. find a matching of residents to hospitals such that there is no **instability**

- ⋆ a **matching** is a set of pairs $(a, F), (c, B), \ldots$ such that each hospital or resident appears exactly once

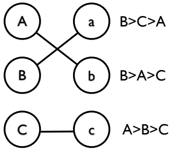


the pair (A, a) is unstable

- ⋆ a matching is **unstable** if there exists a pair of a resident and a hospital, who are not assigned under current matching, and prefer each other over their assigned partners

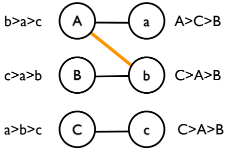
Approach 1. Take each resident and assign his favorite hospital that has not been taken already

- this might give a matching that is unstable, e.g. B:c>a



Approach 2. Take any matching and swap partners of unstable pairs if there exists any instability.

- this might oscillate over matchings that are unstable



$A : abcca \cdots$

$B : baabb \cdots$

$C : ccbac \cdots$

theorem. for any preference lists, there always exists a stable matching

proof. we prove by contradiction.
Gale and Shapley in 1962, came up with an algorithm that finds a stable matching for any preference list. (Typically there might be many stable matchings)

algorithm

- ⋆ WHILE there exists unmatched residents
- ⋆ Each 'unengaged' resident 'propose' to a hospital he has not proposed to yet
- ⋆ Each hospital chooses the most favorable resident out of those who are proposing and his current resident he is engaged to, and gets engaged possibly to a better resident
- ⋆ RETURN the matching

lemma 1. the algorithm produces a perfect matching

lemma 2. the matching is stable

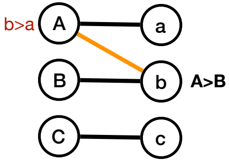
lemma 1. the algorithm produces a perfect matching
proof. we prove by construction.

> Once a hospital is engaged, it will stay engaged in the future (the partner might change). Suppose there is a pair (a, A) that is not engaged by the end, this can only happen if the hospital never got engaged. This can only happen if no one ever proposed to that hospital. However, this is a contradiction, since resident a must have proposed to every hospital, before he realizes that he has no one to propose anymore.

will this terminate? yes.
there are at most n^2 proposals and no proposal is repeated. So the algorithm terminates in less than n^2 iterations.

lemma 2. the matching is stable
proof. (formal) we prove by contradiction.

- ★ Suppose A is matched to a but prefers b over a. Also, b is matched to B but prefers A over B. Then, the pair (A, b) is unstable.



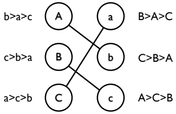
- ★ From the outcome of the algorithm we know

- ★ A-a implies A has proposed to a
- ★ B-b implies B has proposed to b
- ★ b:A>B but B-b implies A has never proposed to b (otherwise A and b will be matched)

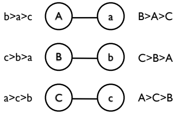
- ★ This implies that A proposed to a before b, which only happened if A preferes $a > b$.
- ★ This contradicts the assumption that A prefers $b > a$.

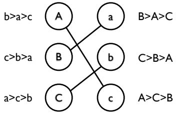
fact this algorithm always produces the same matching, regardless of the order in which we choose the proposals

is this matching optimal/fair?



Optimal for A,B,C





Optimal for a,b,c

theorem. under this algorithm, **the residents get the best possible hospitals**, in the sense that they cannot get a better hospital under any other stable matching.

proof. we prove by induction

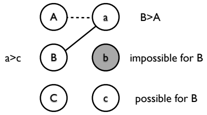
- ⋆ For a resident A, consider a set of 'possible' hospitals, which are the hospitals he can be matched to under a stable matching. We claim that A will never be rejected by a possible hospital. Then, since each resident goes from the best in his list to the worst, as soon as he has proposed to the best possible hospital, he will be matched and never be rejected in the following rounds. So this proves the claim.

we are left to prove that

Each resident will never be rejected by its possible hospitals.

we prove this by **induction** in the time step of the algorithm k

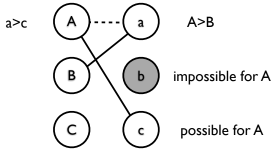
- When $k = 1$, no one has been rejected, so the statement is true.
- Suppose the statement is true up to time step $k - 1$, and A has just been rejected by a in favor of B at time k. Then,
  - ★ B prefers a to all other hospitals except for these who already rejected him.
  - ★ by induction hypothesis, those who rejected B are impossible for B.
- We claim that (A, a) cannot be stable. We prove this by contradiction.
- Suppose there is a stable matching containing the pair (A, a).
- Then, this is a contradiction because
  - ★ B prefers a to any other possible hospitals.
  - ★ a prefers B to A .
  - ★ So the pair (B, a) is unstable.
- Therefore, a is not possible for A.



theorem. under this algorithm where residents make proposals, **the hospitals get the worst possible residents**

proof. we prove by contradiction

- Suppose a hospital a prefers A to B and both A and B are possible residents.
- Suppose the algorithm chooses the pair (A, a) eventually.
- We show there is a contradiction
  - ★ From the previous theorem, we know that a is the best option for A.
  - ★ There exists a stable matching with (B, a) and (A, d), for example.
  - ★ Then, this is a contradiction since the pair (A, a) is unstable. We know that A prefers $a > d$ and a prefers $A > B$.



what is the complexity?

naive implementation has worst-case complexity $O(n^3)$

- ⋆ Consider a simpler version of the algorithm. Start from one resident A and if he is not engaged yet, propose to the next best hospital in his list. The corresponding hospital will choose the better one among his current engaged partner and the new proposal. This also runs at most n^2 proposals, since there are n^2 proposals and we try out one proposal at each iteration and never repeat any proposal. However, at each step, we might have to search for a 'free' resident who is not engaged currently. This takes n time in the worst case. Hence, this algorithm has worst-case complexity (or running time) of n^3.

can we do better?

- ⋆ Now consider a different approach. Start from one resident A and proceed similarly. If he is matched to a hospital who does not have any partner currently, then move to resident B. But, if there was already a partner, then you do the matching as before (match the hospital to the better one of the current parter or A who just proposed). And, next step you move to the one that has been cast out of this match. Because you know for sure that this person is not currently matched. There is no searching for this algorithm, and the running time is at worst-case n^2

here is a list of related questions

- Q. is there a stable matching that is fair for both parties?
- Q. in the worst case, how many stable matchings are there? (open problem)
- Q. organize n men, n women, and n dogs into families of three, each containing one man, one woman, and a dog such that no man, woman, dog have an incentive to desert the current family for a new family. (NP-complete)
- Q. stable roommates problem: $2n$ people have ordered preferences
  - ★ there might not be a solution, for example,
    A: B>C>D
    B: C>A>D
    C: A>B>D
    D: A>B>C
  - ★ there is a $O(n^2)$ time algorithm to determine if there is a stable matching, and to find one if there is

Puzzle: Dynamic Programming

There are four animals on one side of a river and only one boat. A mouse can cross the river in 1 minute, a cat can cross it in 2, a horse in 5 and elephant in 10 minutes. At most two animals can cross the river together at any time. When two people cross the river they will cross it at the rate of the slowest animal. So, for example, if you cross together with the elephant, the journey will take 10 minutes. Someone will have to bring back the boat each time so that another pair can cross the bridge.

(a) Show that, if a mouse escorts each of the animals in turn across the river and then cross back on the boat, the total crossing time is 19 minutes.

(b) Find a sequence of crossings that takes 17 minutes. Is this optimal?

(c) Now, assume that N animals must cross the river, and you know their individual crossing times. You wish to determine the optimal total crossing time. Explain how one could solve this problem optimally (using dynamic programming).

Puzzle: Dynamic Programming

- Time $t \in \{1, 2, 3, 4, 5\}$
- State S_t
- Action a_t
- cost c_t
- Transition (S_t, a_t, S_{t+1}, c_t)
- Optimal action $a_t^*(S_t)$

Puzzle

- You want to test when a laptop breaks when dropped from a great height. You are provided with n identical laptops. There is a high-rise apartment nearby with k floors labelled $1, \dots, k$. Assume that k is very large. You conduct your tests as follows:

  You choose a floor of the apartment, take the lift to that floor, drop a laptop from the balcony and see if it breaks. If it does not break, you may use that laptop again in future tests. You stop testing when you find a floor i such that the laptop does not break when tossed from floors $j \leq i$, and either $i = k$ or the laptop breaks when tossed from floor $i + 1$. Note that if the laptop breaks when tossed from floor i then it will always break when tossed from floors $j > i$. Similarly, if the laptop does not break when tossed from floor i then it will never break when tossed from floors $j \leq i$.

  You want to find the maximum floor number for which the laptop does not break.

Puzzle continued

(a) Suppose $n = 1$. In the worst case, what is the minimum number of tests you must conduct to obtain a conclusive answer?

(b) Suppose n > k. In the worst case, what is the minimum number of tests you must conduct? What is the minimum value of n for which your algorithm still works?

(c) For general n, find a dynamic programming algorithm to determine the minimum number of tests you must conduct in the worst case.

(d) Now consider a continuous version of the problem: you are now supplied with a moving platform attached to a high-rise apartment with height H. A test consists of picking any real number in the interval $[0, H]$ and dropping the laptop from that height. Suppose you only have time to make m tests, and that you still have n laptops at your disposal. Let $A = \min(H, h0)$, where $h0$ is the height such that the laptop breaks when dropped from heights $> h0$ and does not break when dropped from heights $< h0$. Find an algorithm that determines the minimal error you can calculate A to. In other words, what is the smallest ϵ such that, after you perform the tests, you can guarantee that A lies in an interval of width at most ϵ (regardless of the actual value of $h0$)?

Example

Problem 1.

A university comprises 30 different colleges. College i can accommodate up to c_i students. In the application process, prospective students each choose a first-choice and a second-choice college. (All other colleges are automatically tied for 'third choice'.) All applicants then sit an admissions test, and only the top $\sum_i c_i$ students are granted admission to the University. Based on application forms and the admissions test scores, each college then compiles a preference ranking of all the students. The Graduate Studies Board (GSB) receives all this information and is required to find an allocation of students to colleges such that, for any student j, swapping j's college with any other student's college i will not make both j and i happier. Is this always possible? If so, show the GSB how to do this.

Example

Problem 2.

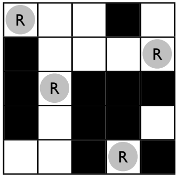
A police station receives notice of m crimes happening simultaneously all over town. There are n police cars at its disposal, where $n > m$. Car i can get to crime scene j in time t_{ij} and one car can only go to one crime scene. Assuming that the t_{ij} are known, devise an algorithm that allocates cars to crime scenes in such a way that the maximum time for each crime scene to be reached by a police car is minimized. Precisely, let $T_j(M)$ be the time required for a police car to reach the crime scene j under matching M. Then, devise an algorithm that finds a matching of police cars to crime scenes such that it minimizes the maximum time to reach a crime scene:

$$\min_M \max_j T_j(M)$$

Example

Problem 3.

You are given an $N \times N$ chessboard. Some cells in the chessboard have been destroyed (call them 'holes') and you are also given a list of these cells. We want to determine the maximum number of rooks you can place on the chessboard such that no two rooks attack each other. (A rook attacks another rook if they both lie in the same row or column - regardless of whether there are any holes between them.) Note that you can't place a rook in a hole.



Formulate this problem as the maximum matching problem.

Example

Problem 4. (Hall's matching theorem)

Define a perfect matching as a matching that matches all the nodes in A of a bipartite graph $G = (A, B, E)$. Hall proved the following celebrated result in 1935.

theorem. For a bipartite graph $G = (A, B, E)$, there exists a *perfect matching* M if and only if for every subset $X \subseteq A$,

$$|X| \le |N(X)| \, , \tag{1}$$

where the neighborhood of a set is defined as

$$N(X) = \{b \in B \,|\, \exists a \in X \text{ with } (a, b) \in E\}$$

- (a) Show that if there is a perfect matching then (1) holds. (this should be easy)
- (b) Show that if (1) holds, then there exists a perfect matching. (there are many ways to prove this claim, using either induction, contradiction, or construction)
- (c) (bonus) Let $G = (A, B, E)$ be a bipartite graph, and let d be the least non-negative integer such that $|N(X)| \ge |X| - d$ for every subset $X \subseteq A$. Show that the size of the maximum matching in this graph is $|A| - d$.