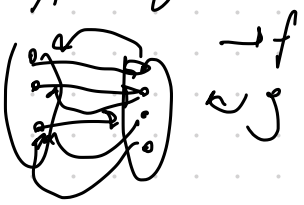
1. a) WTS injective $\Rightarrow$ has left inv $\wedge$ has left inv $\Rightarrow$ injective

(i) WTS injective $\Rightarrow$ has left inverse

Let $f: A \to B$ and assume $\forall x, y \in A,\ x \neq y \Rightarrow f(x) \neq f(y)$

Prove $\exists\, g: B \to A,\ \forall a \in A,\ g(f(a)) = a$.



We know by definition of functions,

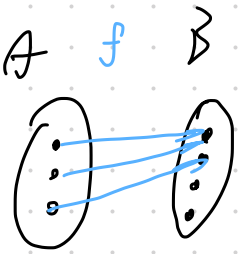
$\forall a \in A, \exists! b \in B, f(a) = b$, that each input has only a single output.

We can define the function $g: B \to A$ to have the property: $\forall b \in B, \exists a \in A, f(a) = b \Rightarrow g(b) = a$,

reversing $f: A \to B$ We can give this property because $f: A \to B$ is injective so each a, b pair is unique.

Thus by the property given to $g: B \to A$, g is a left inverse of f.

(ii) WTS Left inverse $\Rightarrow$ injective



will prove the contrapositive $\neg$ injective $\Rightarrow \neg$ left inverse.

Let $f: A \to B$. Assume f is not injective, that is:

$\exists x, y \in A,\ x \neq y \wedge f(x) = f(y)$, prove f has no left inverse.

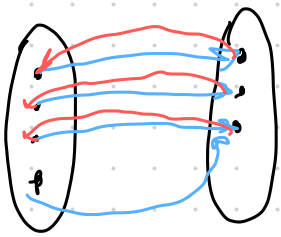
Let $a_1, a_2 \in A$ and assume $f(a_1) = f(a_2)$ and $a_1 \neq a_2$, then

$\exists b \in B,\ b = f(a_1) = f(a_2)$. Let $b = f(a_1) = f(a_2)$ For contradiction assume $f: A \to B$ has a left inverse, $g: B \to A$, where $\forall a \in A, g(f(a)) = a$.

Then we have $g(b) = g(f(a_1)) = a_1$ and $g(b) = g(f(a_2)) = a_2$

So $g(b) = a_1 = a_2$ but $a_1 \neq a_2$ $\Rightarrow\Leftarrow$. We have a contradiction, thus $f: A \to B$ cannot have a left inverse if it is not injective.

1 b) WTS right inv $\Rightarrow$ surjective → Contrapositive?



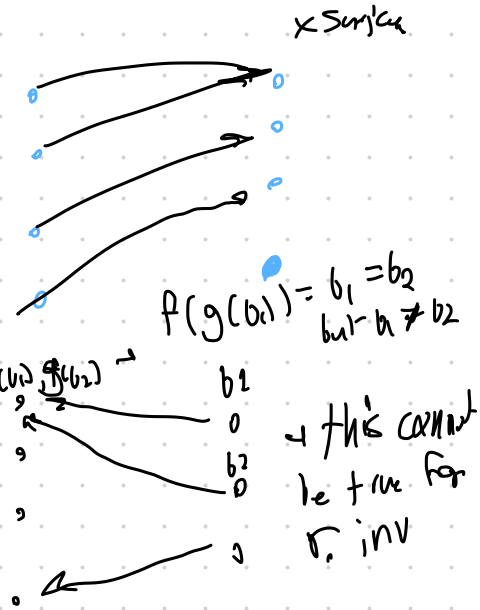
Prove contrapositive ¬ surjective $\Rightarrow$ ¬ right inv

Proof. Let $f: A \to B$, $g: B \to A$. Assume f is not surjective: $\exists b \in B, \forall a \in A, b \neq f(a)$. Prove there is no right inverse of f, i.e. $\exists b \in B, f(g(b)) \neq b$.

Let $b \in B$ be such that $\forall a \in A, b \neq f(a)$.

Then $g(b) = a$ for some $a \in A$. From assumption

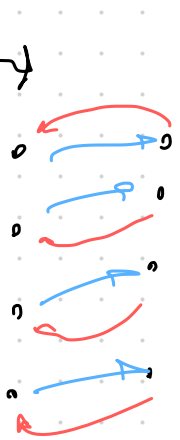
We know $f(a) \neq b$, and so since $a = g(b)$, $f(g(b)) \neq b$.



c) Bijective $\Leftrightarrow$ true inverse

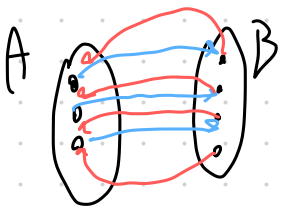
→ ¬left inverse ∨ ¬right inverse $\Rightarrow$ ¬bijective

(i) Proof. Let $f: A \to B$, assume f has no left inverse or no right inverse, then either f is not injective or is not surjective thus either way, f cannot be bijective



true inverse $\Rightarrow$ bijective

(ii) Proof. Let $f: A \to B$ and assume $g: B \to A$ is true inverse. Then g is left inverse and right inverse, so f is injective and surjective, therefore it is bijective.



d) WTS $f: A \to B$ injective $\Rightarrow \exists g: B \to A$, surjective.

Proof. Let $f: A \to B$ and assume it is injective. Then $\exists g: B \to A$ such that g is a left inverse of f. So, $\forall a \in A, g(f(a)) = b$, so every element in A has some corresponding $b \in B$ such that $h(b) = a$ and $f(a) = b$. Then by definition of surjective, g is surjective as $\forall a \in A, \exists b \in B, g(b) = a$.

2.a) split: String × Char → List[String]

str2int : String → Int

↑ only if is a int itself

b) algo: first split by '\n' ('10,3\n4,5' → ['10,3', '4,5'])

then for each item in list (map) split by ',' (['10,3', '4,5']

→ [['10', '3'], ['4', '5']])

then for each sublist for each string, call str2int

split[(split (rawData, '\n')),'

map(map(str2int(map(split ('\n', split (',', a))))) rawdata

Main.hs | Utils.hs

```haskell
import Utils (split, str2int, println, add)

rawData = "0,10,1,3,4,3\n1,3,1,5,5,3\n1,4,3,2,4,7\n0,4,9,3,3,3"

main = do

  -- TODO replace () with the correct function
  let prep = map(map str2int) . map(split ',') . split '\n'
  -- Uncomment below to test your code
  print(prep rawData)

  putStrLn "End of Program"
```

Console | Shell

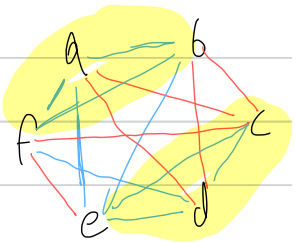
```
runhaskell Main.hs
[[0,10,1,3,4,3],[1,3,1,5,5,3],[1,4,3,2
,4,7],[0,4,9,3,3,3]]
End of Program
```

3. a) WTS any person friends/strangers w/ at least 3 other people

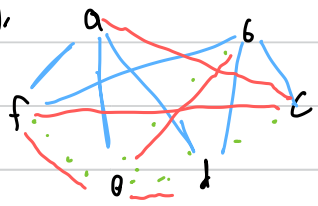
Let p be a person in the party. Besides p, there are 5 other people in the party in which p is either friends or strangers with.
By pigeon hole principle 3 pigeonholes (slots for friends or slots for enemies) and 5 people. Then p can be friends with atleast 3 people and at most strangers with 2 or vice versa.

b) Let a, b, c be people in the party. ⟹

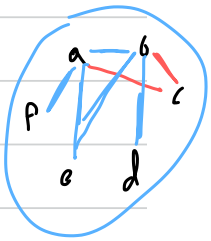
Claim: If there are atleast 3 people that have atleast 3 friends/strangers, then there is a group of 3 friends/strangers



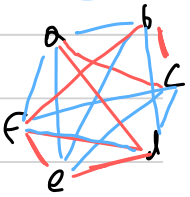
Let a, b, c all have atleast 3 friends in the party.
a is friends with atleast 3/5 people from {b, c, d, e, f},
and b is friends with atleast 3/5 of {a, c, d, e, f}
c is friends with atleast 3/5 of {a, b, d, e, f}

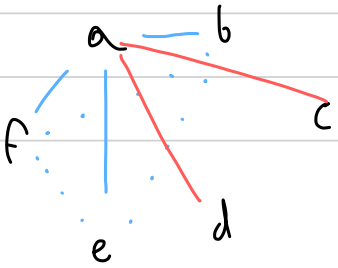


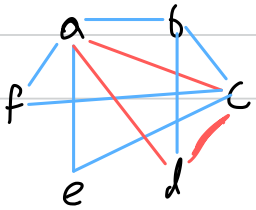
Case1: If all 3 are friends w/ each other, then mutual friends.



Case2: 2 of a, b, c are friends and one is strangers with both.
Suppose a & b friends & strangers with c. Then a & b share at least one friend (not c). ∴ a & b







Garbage

b) I claim: if at least 3 people in the party with at least 3 friends, or 3 people with at least

a, b, c be people. Claim: if 3 people who have 3 friends, must have mutual group.

Assume not mutual friends.

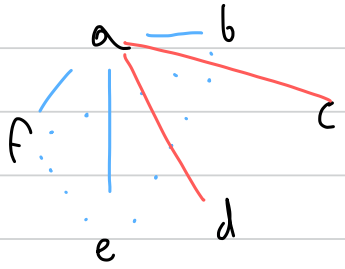
Case 1: a, b, c are not friends, then they must all be strangers, thus they are mutual strangers.

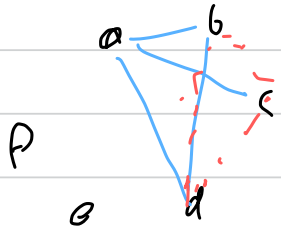
case 2: a & b friends only, strangers with c.
Then a & b share at least 1 friend as there are 3 people left in the party, in which they are friends with 2 of them (by pigeonhole principle, they share a friend). Thus a and b have a shared friend, and so they are a group of 3 mutual friends.

Case 3: a friends with b, b friends with c, c stranger to a.
a can be friends with 2/3 people remaining and b can be friends with 2/3 people remaining. If they share one friend, then mutual group, else, they share no mutual friends and a is strangers to b's friends. Then c can be friends with both of a's friends or one of a's friends and b's friend who is not a or c (mutual friends this way). Then if c is strangers to b's friend, we have mutual strangers with a, c, and b's friend who is not a or c.
(can prove same for strangers, just change wording // vice versa)

Since we have 6 people in the party, we have, at all times a group of at least 3 people that have at least 3 friends or a group of at least 3 people with at least 3 strangers, so the claim is always True.

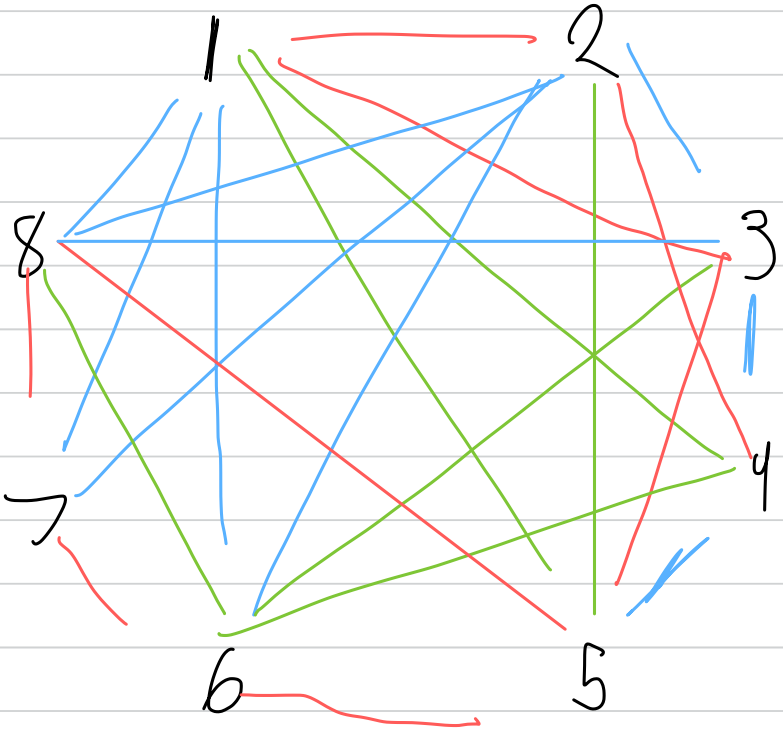
3b)





Let a be a person in the party. a must be friends with at least 3 people or strangers with at least 3 people by PHP. WLOG, assume a has three friends b, c, d (people in party). If any of them b, c, d are friends, then they form a group of 3 mutual friend with a. Otherwise they're all strangers to one another forming a group of 3 mutual strangers

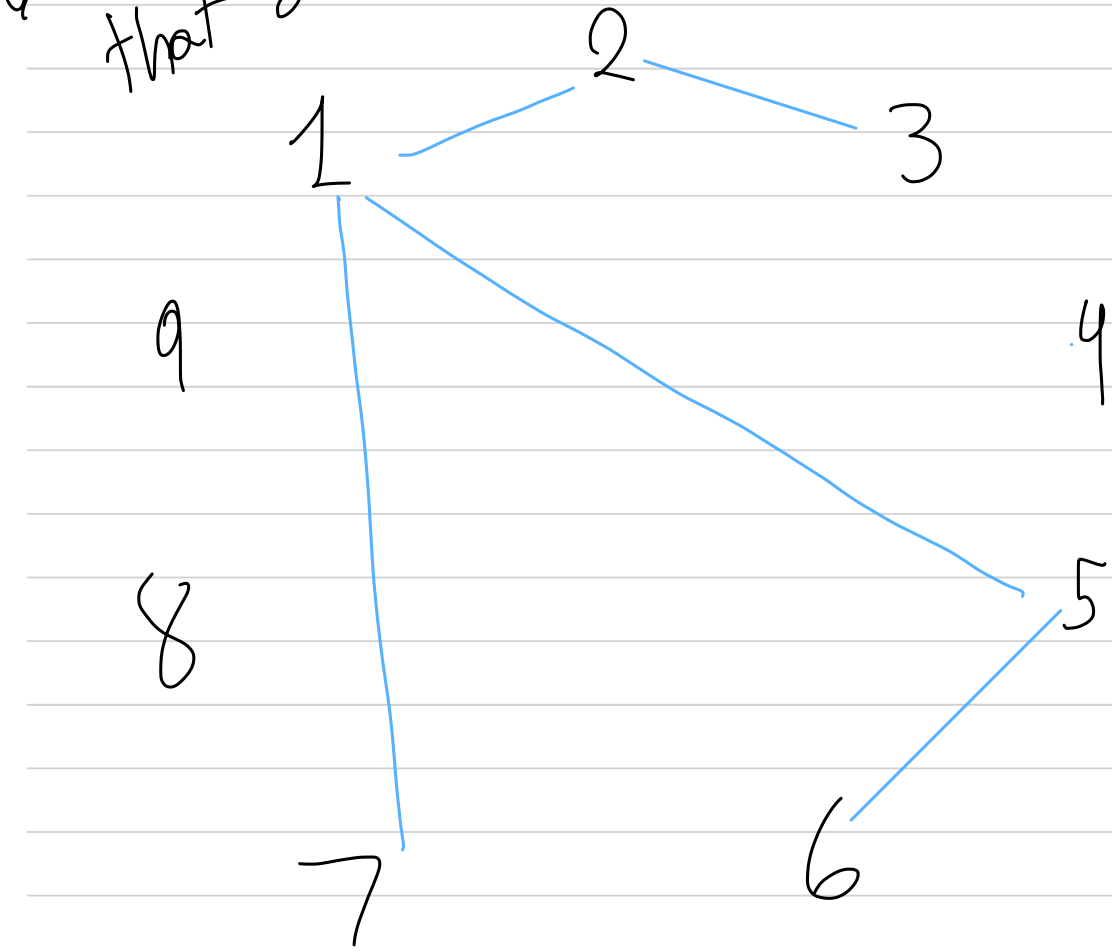
3.c)



F	S	t
9	0	0
8	1	0
8	0	1
7	0	2
7	1	1
7	2	1
6	3	0
6	0	3
6	2	1
6	1	2
5	4	0
5	0	4
5	2	2
5	3	1
5	1	3
4	5	0
4	0	5
4	4	1
4	1	4
4	2	3
4	3	2
3	3	3
3	6	0
3	0	6
3	5	1

3 2 4
3 4 2
2 6 0
2 6 1

& always can that somehow 3 tops

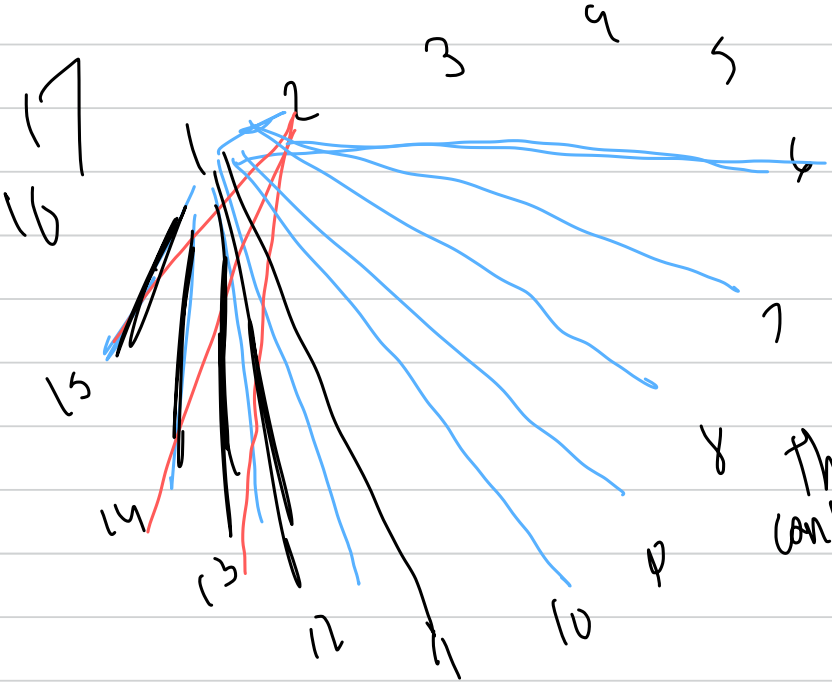


3. c) Let $k \in \mathbb{N}$.

Have: Complete graph w/ 3 "types" of edges, want k such that for any graph of size k, have mutual group

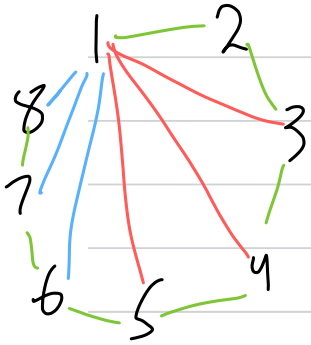
Need: value of k.

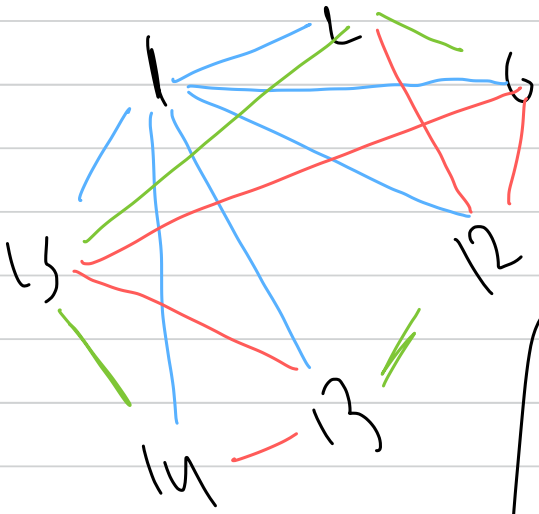
(17) 16 ppl, always have a group of 6
(is can have 5, 5, 5)



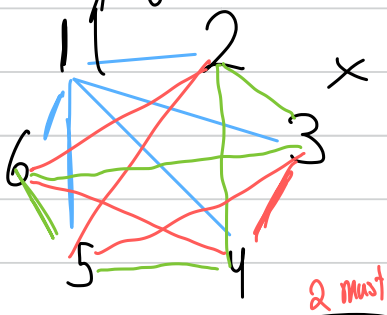
have a group of 6 - 1 is friends w all, then one of them can be friends w/ elv else

& Make problem smaller

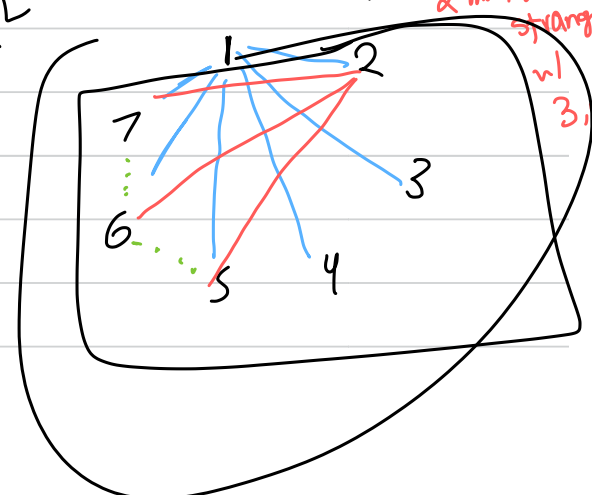




only has 5 friends



2 must be enemies/strangers w/ 3, 4, 5, & 6?



3c) $k=17$, when $k=17$ there must be at all times someone who is friends, strangers, or enemies with at least 6 people (at 16 we have the possibility of 5 friends, 5 enemies, 5 strangers). Let a be a person in the party, w.l.o.g. assume a has 6 friends. Let $S_1 \subseteq S$ be the friends of a. Then if any of them are friends, have a group of mutual friends. Else, let $b \in S_1$, by pigeonhole principle and w.l.o.g. b must be strangers or enemies with 3 people in S_1, suppose strangers (w.LOG). Let these be c, d, e. If any of c, d, e are friends, form mutual friends with a, if any of them strangers with another, then form mutual strangers with b, otherwise they all are enemies, thus since 3 of them, mutual enemies.

for any graph with nxn vertices where $n \geq 4$, there exists a set of edges for that graph such that the graph is connected, has a Hamiltonian cycle and an independent set of size n

4.a) every square in chessboard is a vertex

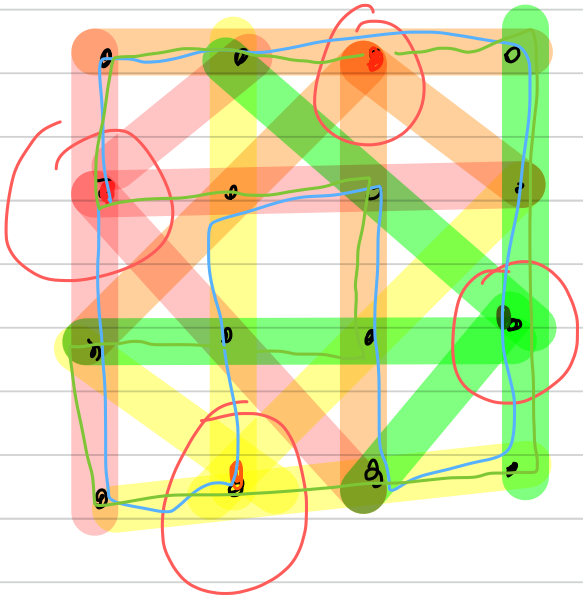
(model chessboard as graph)

independent sets??

for a graph with nxn vertices possible to have n independent sets???

$V = \{$ chess squares $\}$

$E = \{$ valid queen moves $\}$ each queen has Hamiltonian cycle



→ Connected

→ Queens not attacking each other ⇒ Queen squares are not adjacent

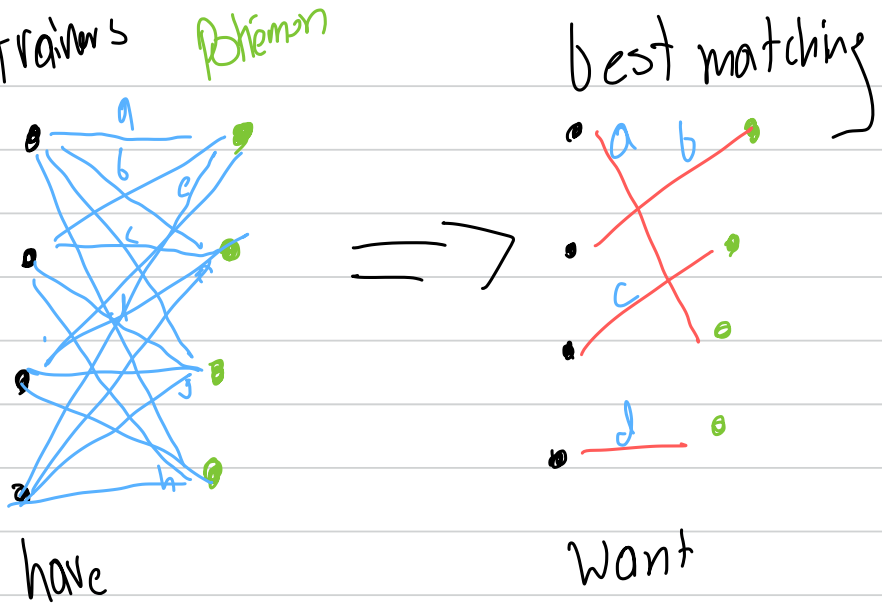
→ can form Hamiltonian cycle

(some #)

/ \

if a graph has more than 4 rows and columns and that graph is connected, then there exists an independent set such that $|k| = n$ (#rows/cols) and for each vertex is in, then is a Hamiltonian cycle

Goal



* Complete Bipartite Graph, $G = (V, E)$

V = {starter pokémons and trainers}

E = {compatability between trainers & pokémons}

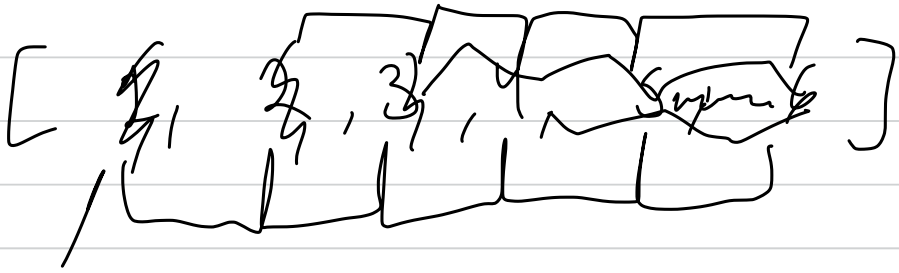
↳ weights are compatability score b/w trainer & pokémon

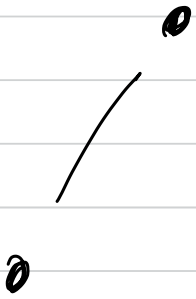
Problem: Maximize compatability, find a matching in the graph such that for all other matchings, this matching has the heighest weight.

i.e. some matching $M \subseteq E$ such that $|M|$ is largest for all matchings.

5.6) locations = { city : (x, y) }

cities = [city1, city2 ...]





* on .ipynb

6.

6.

Suppose want to train a computer vision model recognizing Human limbs / body parts (personal project I aim to create). We give data to to model for body parts

6. min spanning tree independent set
shortest path
matching
Traveling Salesman

Summer is coming, which means a visit to Canada's Wonderland! You have your list of rides you want do ride, but unfortunately it's really busy. However your graph skills are coming in handy. Find the best possible route between them assuming approximate wait times constant

V = {rides that you want to go on and entrance}

E = {paths between attractions / rides}

↳ weights: time it takes to get between rides.

→ Find minimum spanning tree
↳ Want to optimize waiting time
Low rides: lower weight, and visit all nodes ≈ MST
↳ MST will show us in what order to visit each ride which optimizes waiting time.