Matchings

Jim Newton

September 22, 2026

BING lecture notes

Please feel free to review these notes. They are intended to supplement the lectures.

- cube2x2x2.pdf
- Floyd-Warshall-lecture.pdf
- Floyd-Warshall-slides.pdf
- graph-isomorphism.pdf
- graph-machine-representation.pdf
- matchings-lecture.pdf
- matchings-slides.pdf
- single-source-shortest-paths.pdf

Download folder | Edit

Forward

In this lecture we will look at:

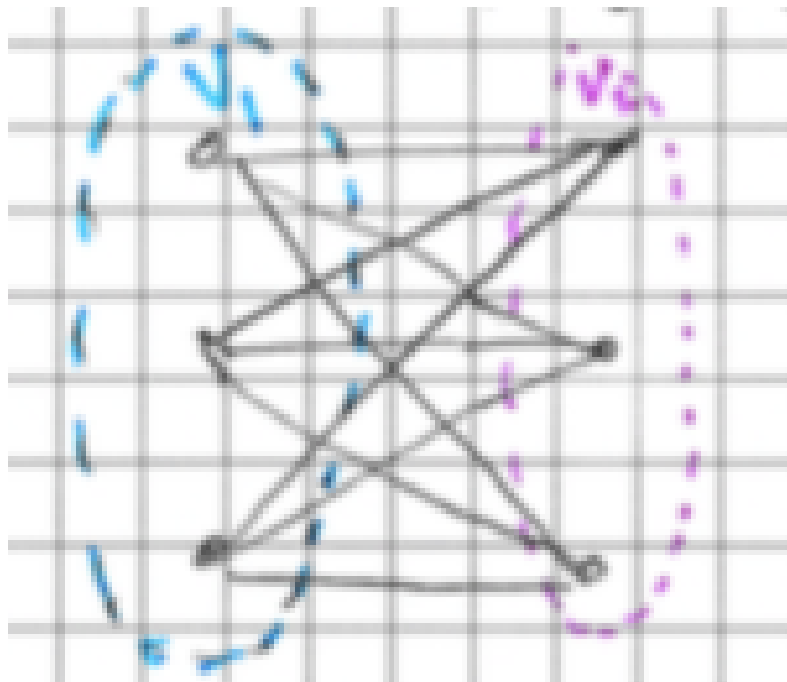
- Homework problems
- Matchings
  - Maximum, Maximal, and Perfect
- Alternating and Augmenting paths
- Finding a maximum matching
- Finding an augmenting path
  - Manual search
  - Naïve search
  - Systematic search

Definition (bipartite graph)

A *bipartite* graph, $G = (V, E)$, is a graph for which V can be partitioned into $V = V_1 \cup V_2$ where $V_1 \cap V_2 = \emptyset$ (denoted $V = V_1 \sqcup V_2$) and every edge $e \in E$ is of the form x, y with $x \in V_1$ and $y \in V_2$.

Definition (bipartite graph)

A *bipartite* graph, $G = (V, E)$, is a graph for which V can be partitioned into $V = V_1 \cup V_2$ where $V_1 \cap V_2 = \emptyset$ (denoted $V = V_1 \sqcup V_2$) and every edge $e \in E$ is of the form x, y with $x \in V_1$ and $y \in V_2$.



The utility graph is bipartite.

What is a matching?

Matching is a weaker concept than bipartite.

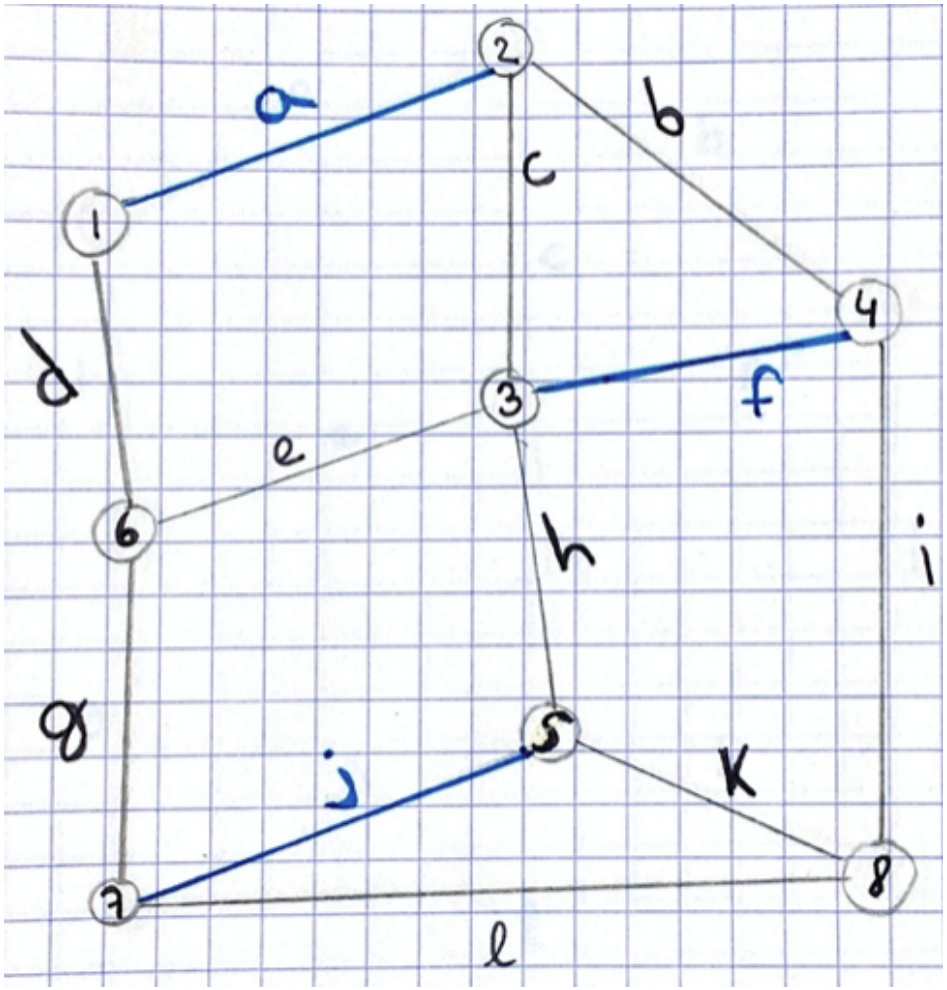
What is a matching?

Matching is a weaker concept than bipartite.

Definition (matching)

If $G = (V, E)$ is a simple undirected, simple graph, a *matching*, $M \subseteq E$, is a set of non-incident edges of G.

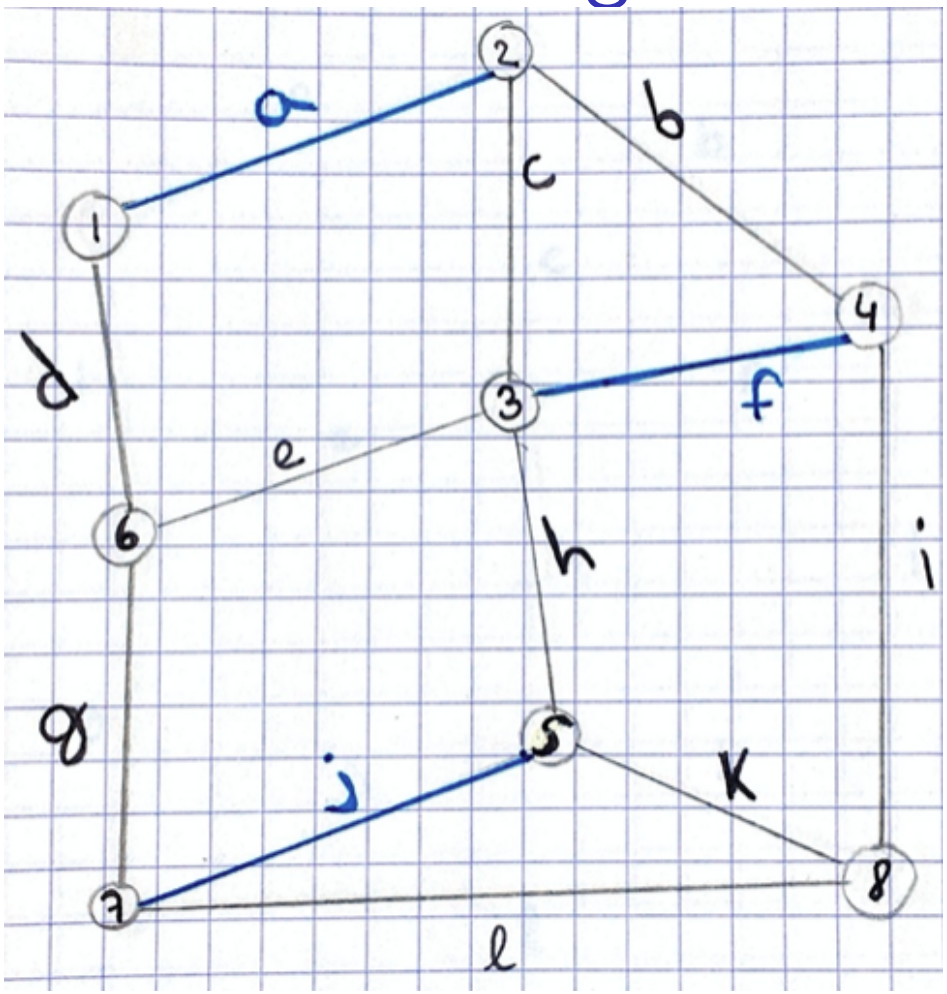
What is a matching?



M_1 is a matching in this graph.

$$M_1 = \{a, f, j\} = \{\{1, 2\}, \{3, 4\}, \{5, 7\}\}$$

What is a matching?



M_2 is not matching in this graph.

$$M_1 = \{a, f, j\} = \{\{1,2\},\{3,4\},\{5,7\}\}$$

$$M_2 = \{a, b, j\} = \{\{1,2\},\{2,4\},\{5,7\}\}$$

Maximal vs Maximum Matching

Definition (maximal matching)

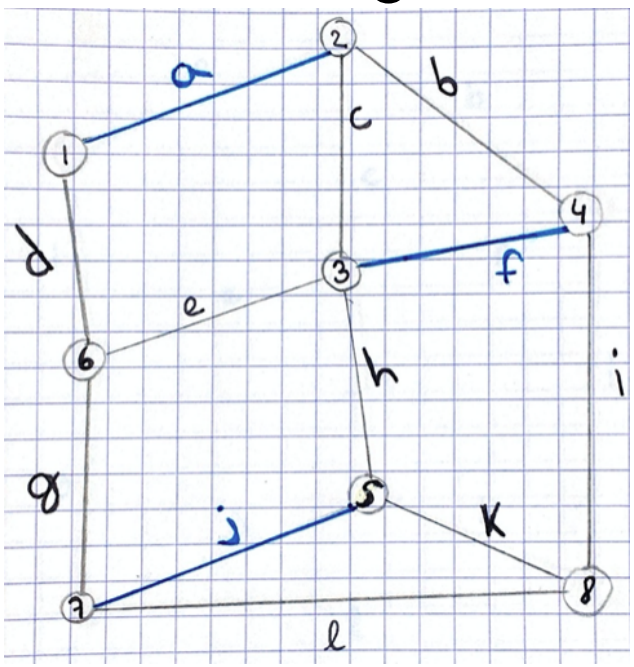
A matching, M, is said to be *maximal*, if there exists no matching $M' \neq M$ for which $M \subset M'$

Definition (maximum matching)

A matching, M, is said to be *maximum* (or *matching of maximum cardinality*), if there exists no matching M' such that $|M'| > |M|$.

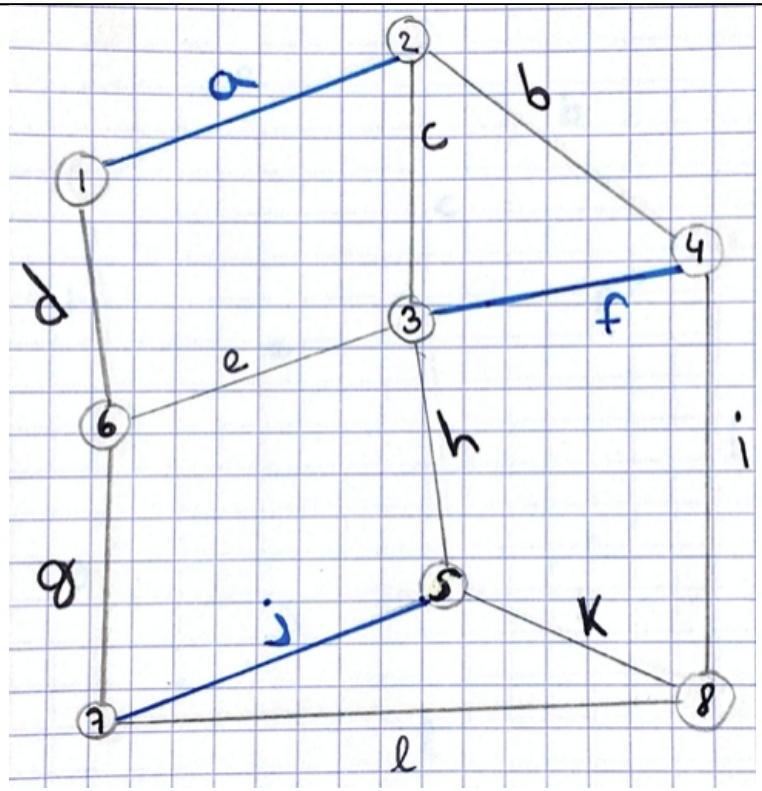
Maximal vs Maximum Matching

One might think that *maximal* and *maximum* matchings are the same thing, but counter-intuitively, they are not.



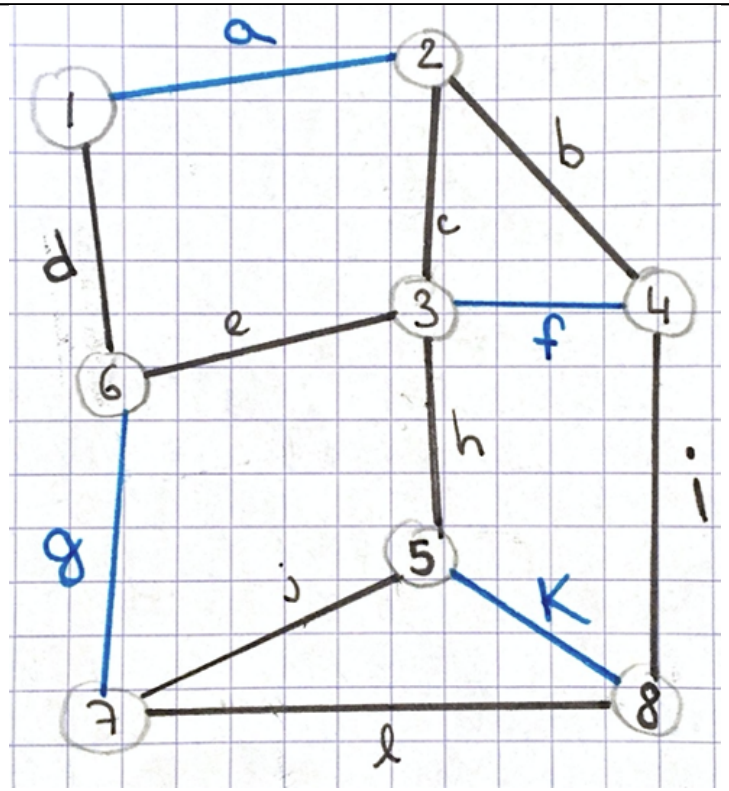
$M_1 = \{a, f, j\}$ is a *maximal* matching. Why? Because no other edge can be added and remain a mapping. For example in $\{a, f, j, e\}$ edge e and f are incident. Likewise in $\{a, f, j, \ell\}$ edge j and ℓ are incident, etc.

Maximal Matching



$M_1 = \{a, f, j\}$
$|M_1| = 3$

Maximum Matching

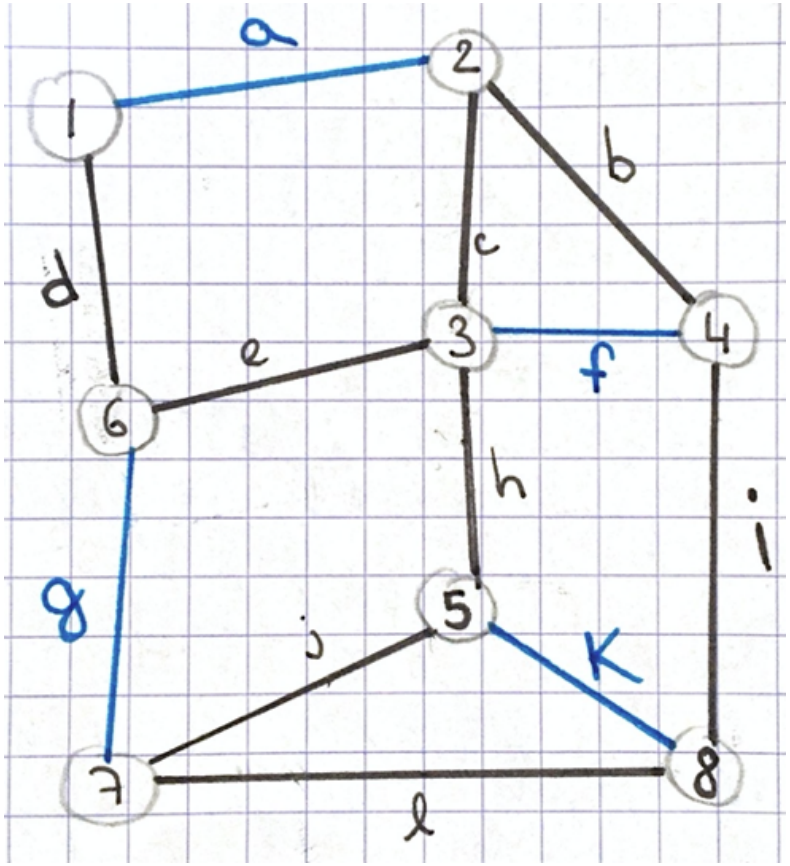


$M_2 = \{a, f, g, k\}$
$|M_2| = 4 > |M_1|$

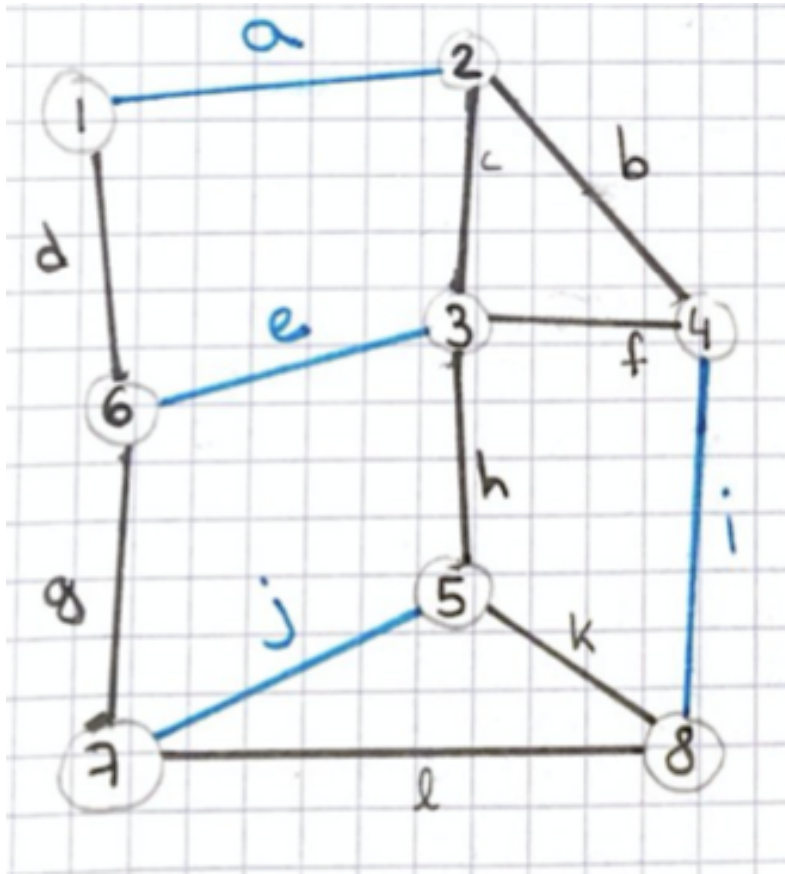
We know M_2 is maximum because

$$\bigcup M_2 = a \cup f \cup g \cup k$$
$$= \{1, 2\} \cup \{3, 4\} \cup \{6, 7\} \cup \{5, 8\}$$
$$= \{1, 2, 3, 4, 5, 6, 7, 8\} = E$$

A maximum matching is not always unique



$M_2 = \{a, f, g, k\}$
$|M_2| = 4$



$M_3 = \{a, e, i, j\}$
$|M_3| = 4$

Perfect matching

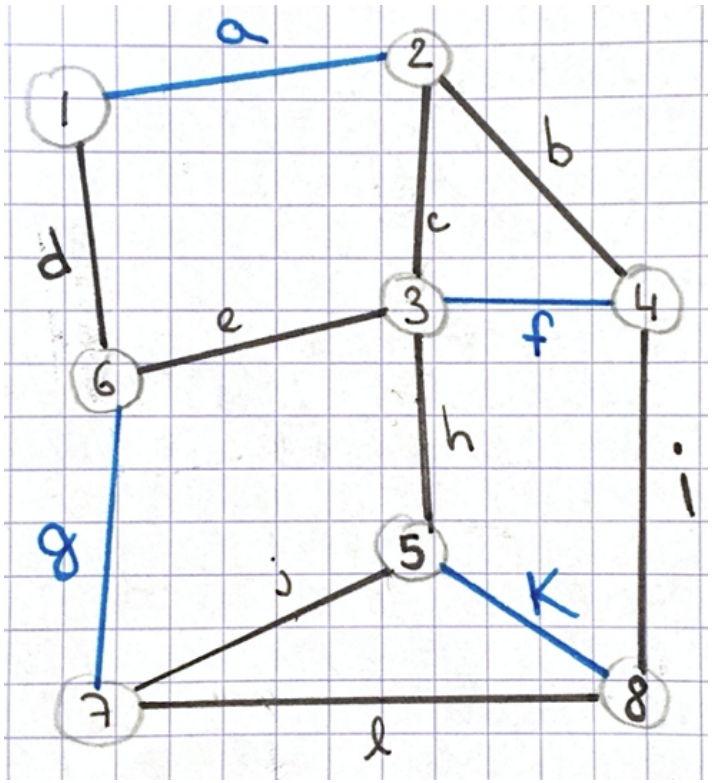
Definition (perfect matching)

A matching M of graph $G = (V, E)$, is said to be *perfect*, if
I.e., every vertex, $v \in V$ is an endpoint of some edge in M.

Perfect matching

Definition (perfect matching)

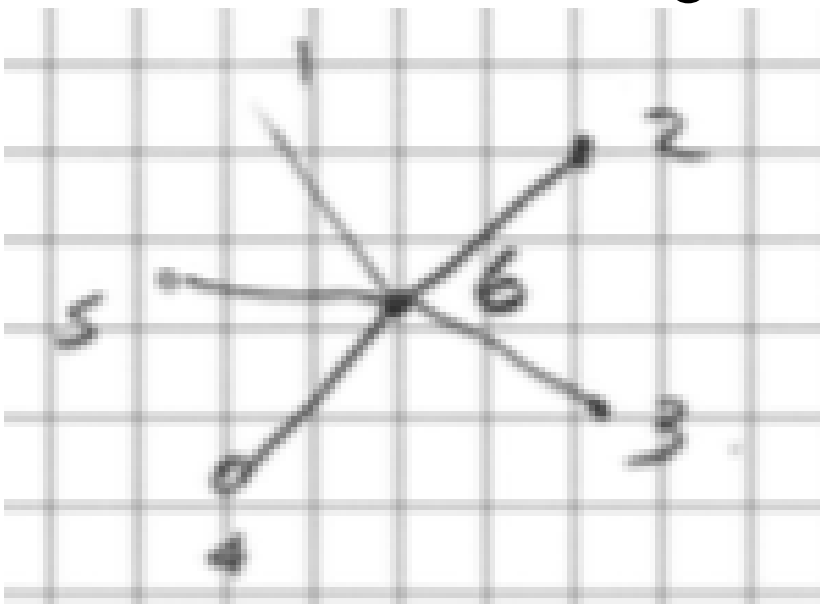
A matching M of graph $G = (V, E)$, is said to be *perfect*, if
I.e., every vertex, $v \in V$ is an endpoint of some edge in M.



This is a perfect matching, because $\bigcup M = V$.

Perfect matching

Some maximum matchings are not perfect:



$M_7 = \{\{1, 6\}\}$.
M_7 is maximum but no edge contains $v = 2$ (for example).

Some claims about matchings

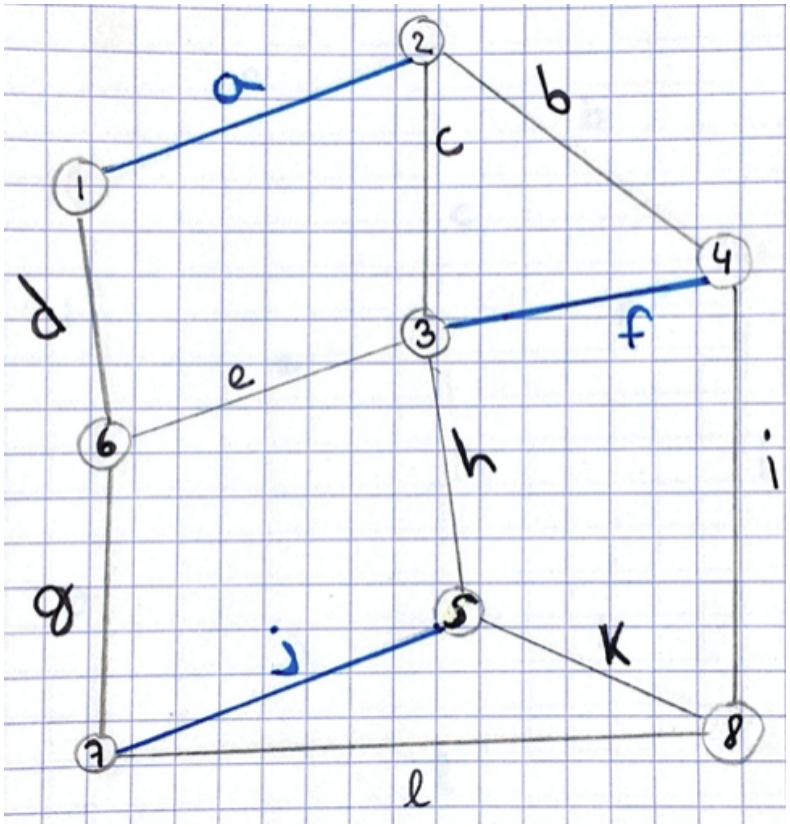
1. Every perfect matching is maximum.
2. Every maximum matching is maximal.
3. Some maximal matchings are not maximum.
4. No matching, M of graph $G = (V, E)$ contains more than $\lfloor \frac{|V|}{2} \rfloor$ edges. *I.e.*, $|M| \leq \lfloor \frac{|V|}{2} \rfloor$.
5. Every perfect matching contains exactly $\frac{|V|}{2}$ edges.
6. Any matching with $\lfloor \frac{|V|}{2} \rfloor$ edges is maximum.
7. Some maximum matchings have fewer than $\lfloor \frac{|V|}{2} \rfloor$ edges.

End of video 1

Start of video 2

Augmenting paths

Finding a maximum matching by hand



$$M = \{\{1,2\},\{3,4\},\{5,7\}\}$$
$$= \{a, f, j\}$$

Some alternating paths are:

$\{d, a, c\}$

$\{a, c, f, i\}$

$\{d, a, b, f, h, j, \ell\}$

$\{a\}$

$\{a, c\}$

Finding a maximum matching by hand

Definition (alternating path)

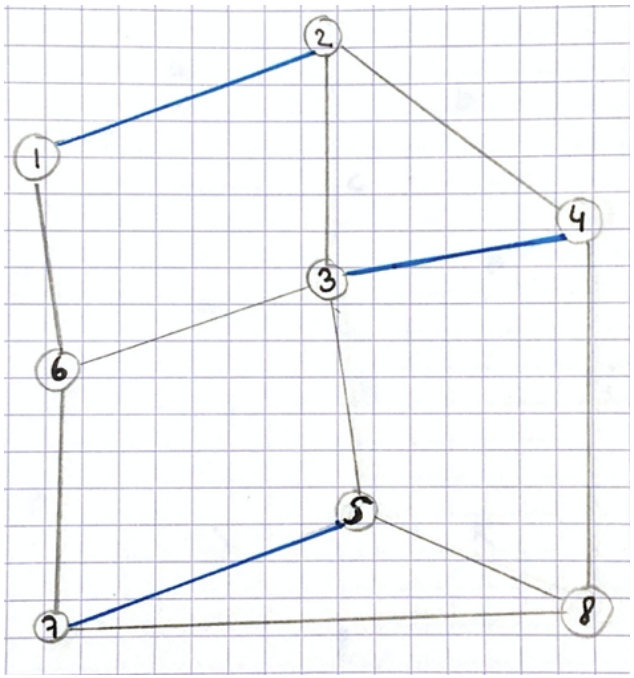
Given graph $G = (V, E)$, an *alternating path* with respect to matching, M, is a sequence of edges $(e_1, e_2, \ldots, e_k)$ such no vertex is repeated, and e_i is incident to e_{i+1} for $1 \leq i < k$, and

- either $e_i \in M$ if i is even and $e_j \in M \setminus E$ if j is odd,
- or $e_i \in M$ if i is odd and $e_j \in M \setminus E$ if j is even.

Finding a maximum matching by hand

Definition (free vertex)

Given a graph $G = (V, E)$, and a matching $M \subseteq E$, a vertex is called *free* (or *exposed*, or *unmatched*), if it is not an endpoint of any edge in M. Otherwise it is called a *matched* vertex.



$M_1 = \{\{1, 2\}, \{3, 4\}, \{5, 7\}\}$

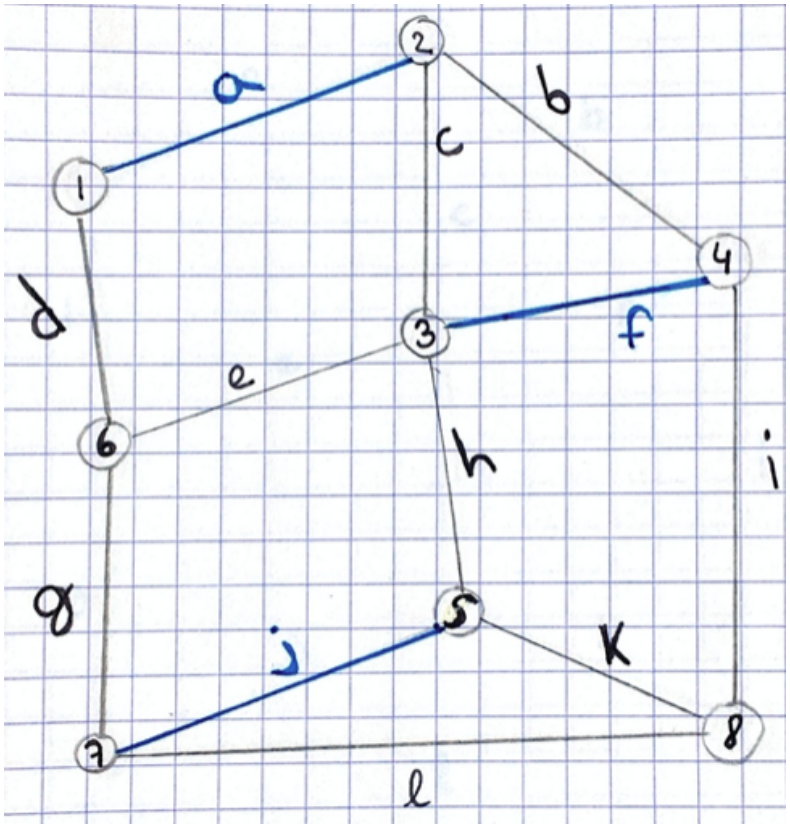
The free vertices are 6 and 8.

Finding a maximum matching by hand

Definition (augmenting path)

An *augmenting path* is an alternating path between two free vertices.

Finding a maximum matching by hand



$$M = \{\{1,2\},\{3,4\},\{5,7\}\}$$
$$= \{a, f, j\}$$

The only free vertices are 6 and 8, so alternating paths with 6 and 8 as endpoints are augmenting.

Some augmenting paths are:

$$P_1 = d\ a\ c\ f\ i$$
$$P_2 = g\ j\ k$$
$$P_3 = e\ f\ i$$
$$P_4 = d\ a\ b\ f\ h\ j\ \ell$$

Some claims about augmenting paths

1. An augmenting path has k matched edges and $k + 1$ unmatched edges, for some integer, k.
2. Every augmenting path has an odd number of edges because $k + k + 1$ is odd.

Berge's Lemma

Berge's Lemma (1957) states that if M is a matching for a graph, then M is maximum if and only if there is no augmenting path.

Alternatively, there is an augmenting path if and only if M is not maximum.

Maximum matching algorithm

Definition (set exclusive or (XOR))

Given two sets M and P, the exclusive-or of the two sets, denoted $M \oplus P$, is defined as follows:

$$M \oplus P = (M \setminus P) \cup (P \setminus M)$$

I.e., the set of elements in either M or P but not in both.

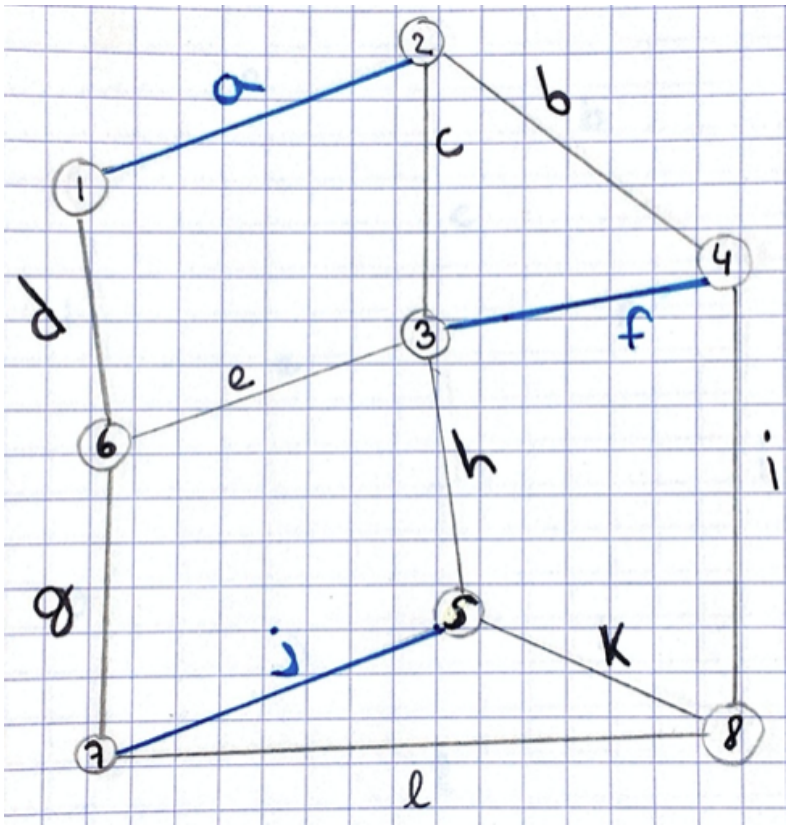
Remember that $M \oplus P = P \oplus M$,
so it does not matter which order you use to compute it.

Maximum matching algorithm

If M is a matching, and P is the set of edges in an augmenting path, then $M \oplus P$ is a matching exactly one larger than M:

$$|M \oplus P| = |M| + 1$$

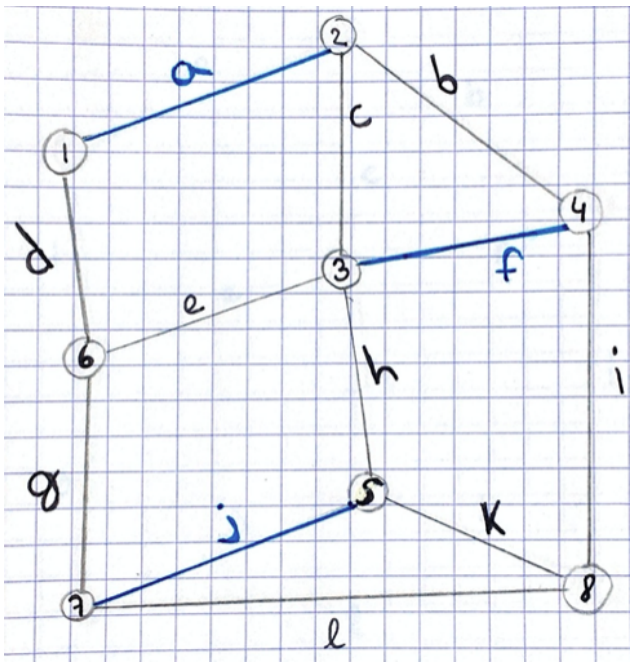
Maximum matching algorithm



$$M = \{a, f, j\}$$

$$P_1 = d\ a\ c\ f\ i$$
$$P_2 = g\ j\ k$$
$$P_3 = e\ f\ i$$
$$P_4 = d\ a\ b\ f\ h\ j\ \ell$$

Maximum matching algorithm



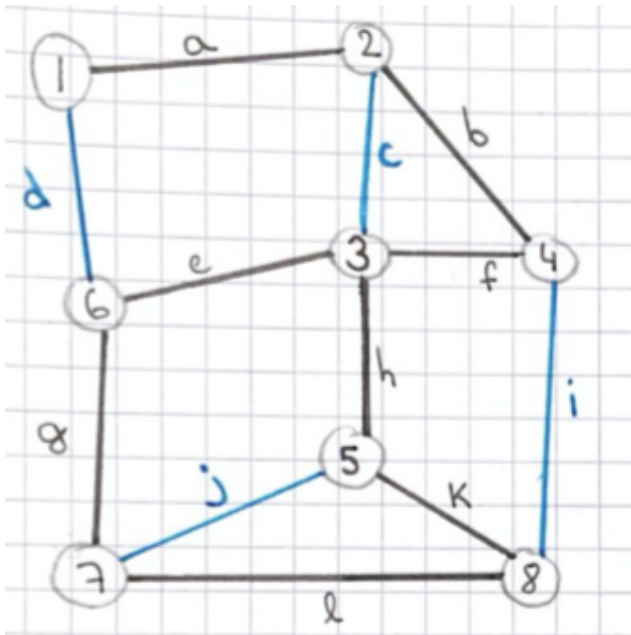
$$M = \{a, f, j\}$$

$$P_1 = d\ a\ c\ f\ i$$

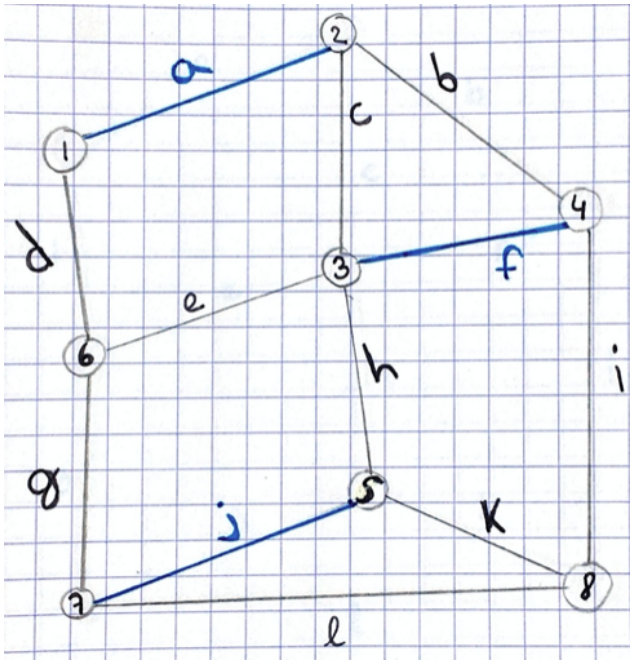
$$M \oplus P_1$$

$$= \{a, f, j\} \oplus \{d, a, c, f, i\}$$

$$= \{d, c, i, j\}$$



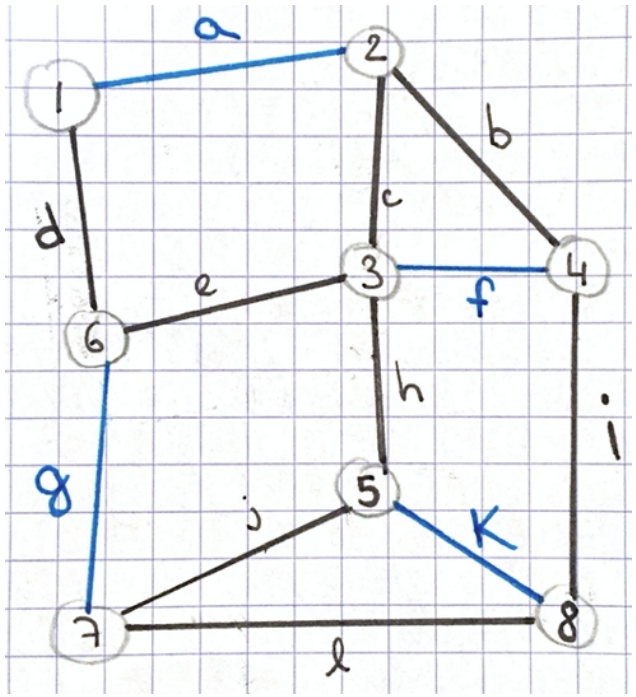
Maximum matching algorithm



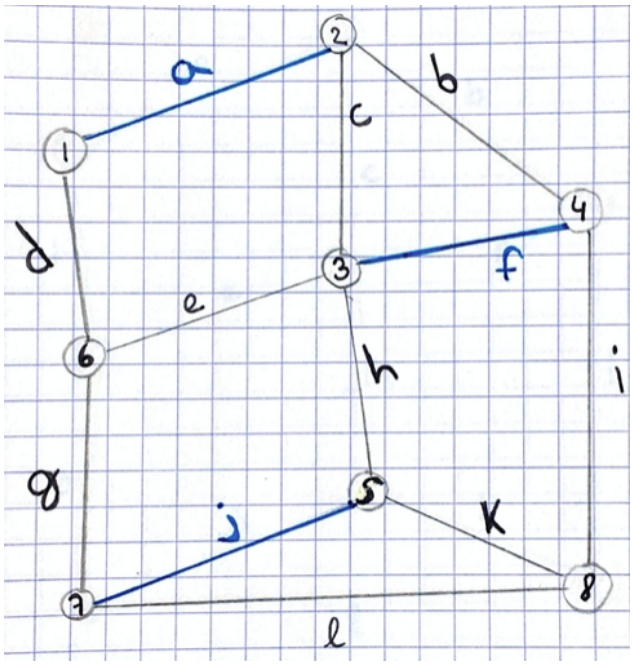
$M = \{a, f, j\}$

$$P_2 = g\ j\ k$$

$$M \oplus P_2 = \{a, f, j\} \oplus \{g, j, k\} = \{a, f, g, k\}$$



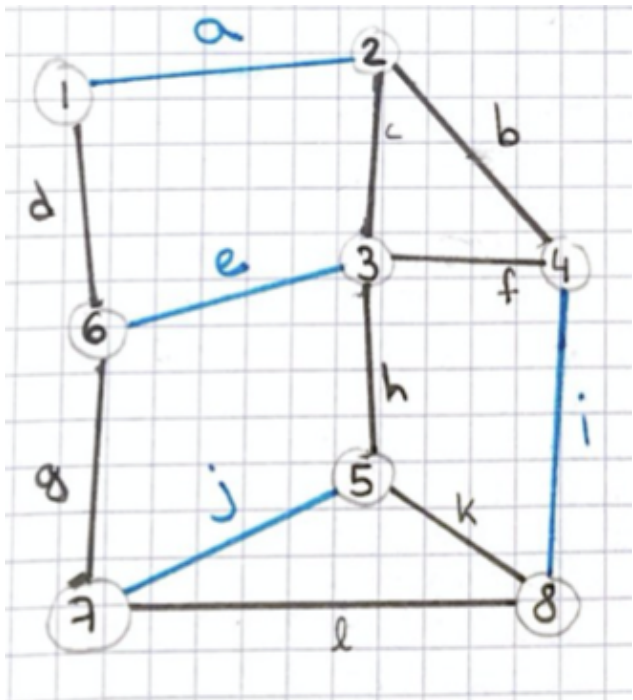
Maximum matching algorithm



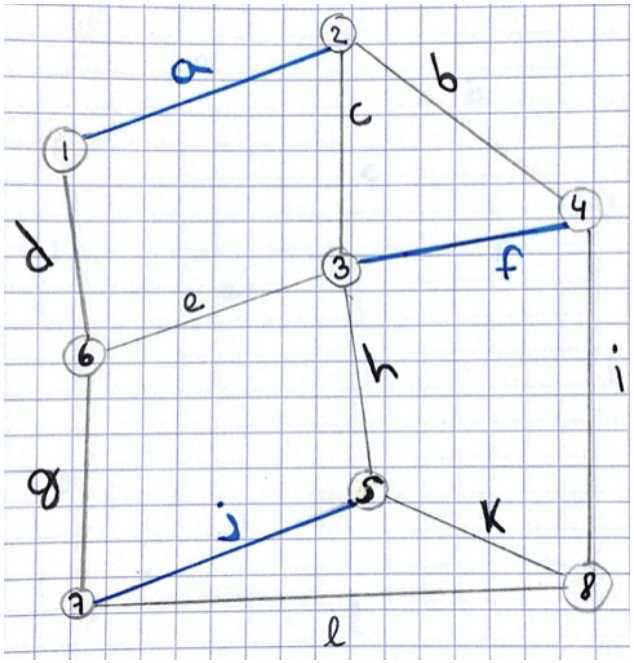
$M = \{a, f, j\}$

$$P_3 = e\ f\ i$$

$$M \oplus P_3 = \{a, f, j\} \oplus \{e, f, i\} = \{a, j, e, i\}$$



Maximum matching algorithm



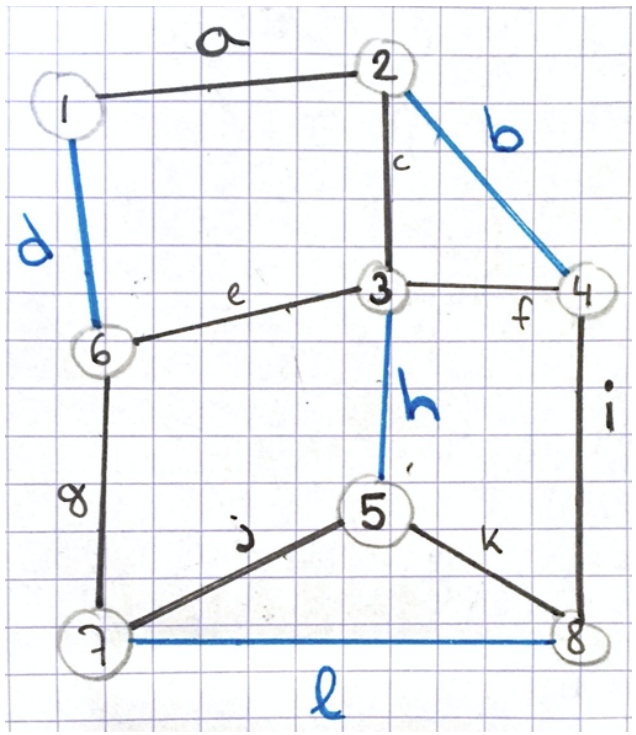
$M = \{a, f, j\}$

$$P_4 = d\ a\ b\ f\ h\ j\ \ell$$

$$M \oplus P_4$$

$$= \{a, f, j\} \oplus \{d, a, b, f, h, j, \ell\}$$

$$= \{d, b, h, \ell\}$$



Maximum matching algorithm

Note that all the derived from P_1, P_2, P_3, and P_4 all have length $|M| + 1 = 4$.

$$\begin{aligned} P_1 = \{d, a, c, f, i\} &\implies M \oplus P_1 = \{d, c, i, j\} \\ P_2 = \{g, j, k\} &\implies M \oplus P_2 = \{a, f, g, k\} \\ P_3 = \{e, f, i\} &\implies M \oplus P_3 = \{a, j, e, i\} \\ P_4 = \{d, a, b, f, h, j, \ell\} &\implies M \oplus P_4 = \{d, b, h, \ell\} \end{aligned}$$

There's no need to try to find the longest possible augmenting path. The first one your find, is sufficient, even if it is the smallest one.

Augmenting path algorithm

```
1.1 Function find-maximum-matching(G, M)
1.2 |   p ← find-augmenting-path-or-none(G,M)
1.3 |   if p is an augmenting path then
    |   |           // recur with M to find |M| + 1
1.4 |   |   return find-maximum-matching(G, M ⊕ p)
1.5 |   else
1.6 |   |   return M
```

Algorithm 1: Implementation of function to find a maximum matching.

Augmenting path algorithm

The algorithm to find a maximum matching is easy if you have the algorithm for `find-augmenting-path-or-none`.
That's the hard part.
Edmond's Blossom Algorithm (1961) provides a solution which is $O(|E||V|^2)$.
We will look at a BFS-type brute force algorithm.

End of video 2

Start of video 3

Live coding immutable graph search with single source

End of video 3

Start of video 4

Live coding immutable graph search with multiple sources

End of video 4

Start of video 5

Naïve graph search for augmenting path

One to many mapping

Convert one k-length path to many $k+1$ length paths.

2.1 **Function** *extend-path-by-1*(adj, k, p)
2.2 $\quad$ **return** $\{u :: p \mid u \in adj[p_0] \wedge u \notin p\}$

Algorithm 2: Set comprehension generating possibly empty set of paths, length $1 >$ the given non-looping path.

extend-path-by-1 in Python

To grow paths at beginning:

```
def extend_path_by_1(adj,k,p):
  return [[u]+p for u in adj[p[0]]
              if u not in p]
```

To grow paths at end:

```
def extend_path_by_1_at_end(adj,k,p):
  return [p+[u] for u in adj[p[k]]
              if u not in p]
```

Many to many mapping

Convert many k-length paths to many $k+1$ length paths.

```
3.1 Function extend-paths-by-1(adj, k, paths_old)
3.2 |   for p ∈ paths_old do
3.3 |   |   for q ∈ extend-path-by-1(adj, k, p) do
3.4 |   |   |   collect(q)
3.5 |   return collected paths
```

Algorithm 3: BFS graph descent with immutable data structure

extend-paths-by-1 in Python

```
def extend_paths_by_1(adj,k,paths):
  return [q for p in paths
          for q in extend_path_by_1(adj,k,p)]
```

Naïvely finding an augmenting path

```
4.1  Function
       naive-find-augmenting-path-or-none(adj, E, M)
4.2      paths ← {[v, u] | u ∈ free, {u, v} ∈ E}
4.3      k ← 1
4.4      while paths ≠ ∅ do
4.5          for p ∈ paths do
4.6              if p is an augmenting path then
4.7                  return p
4.8          paths ← extend-paths-by-1(adj, k, paths)
4.9          k ← k + 1
4.10     return None
```

Algorithm 4: Naïve function to find augmenting path

Implementing line 4.2 in Python

$$paths \leftarrow \{[v, u] \mid u \in free, \{u, v\} \in E\}$$

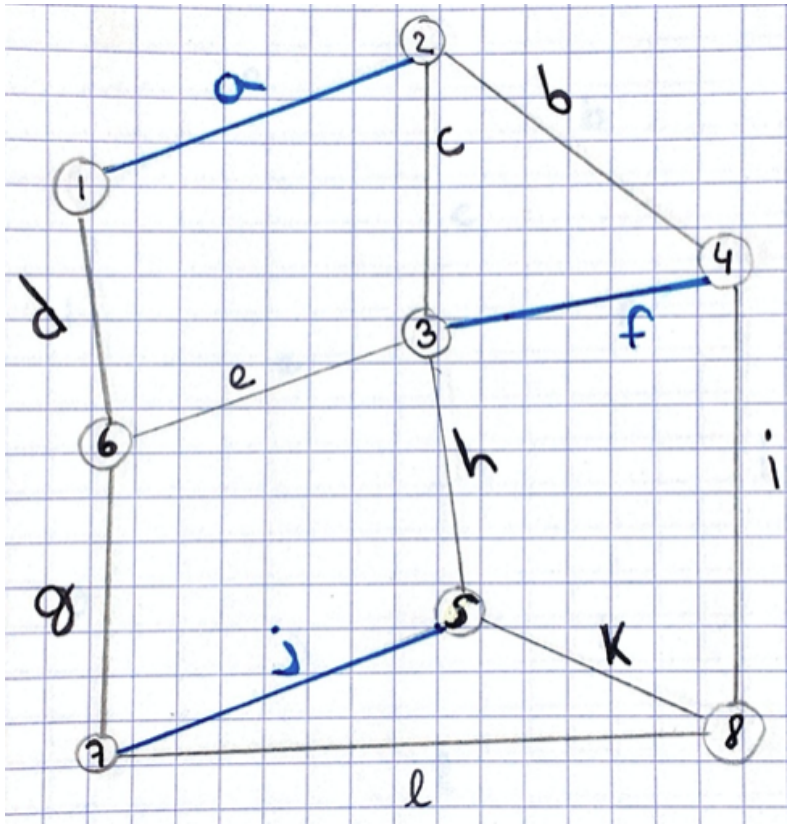
```
paths = [[v,u] for u in free
                for v in adj[u]]
```

End of video 5

Start of video 6

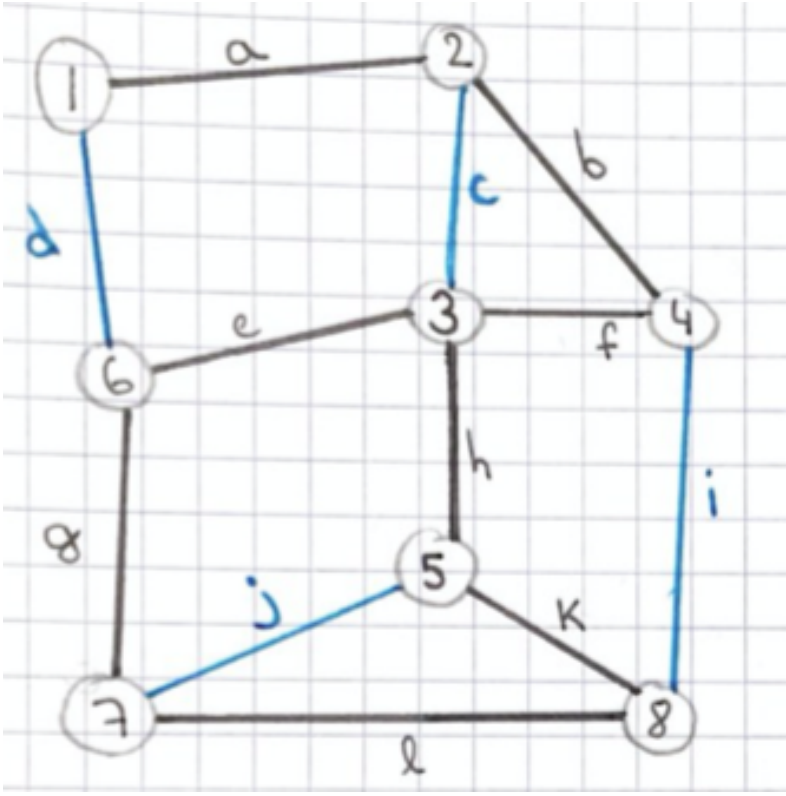
Systematic Augmenting Path Search

First optimization



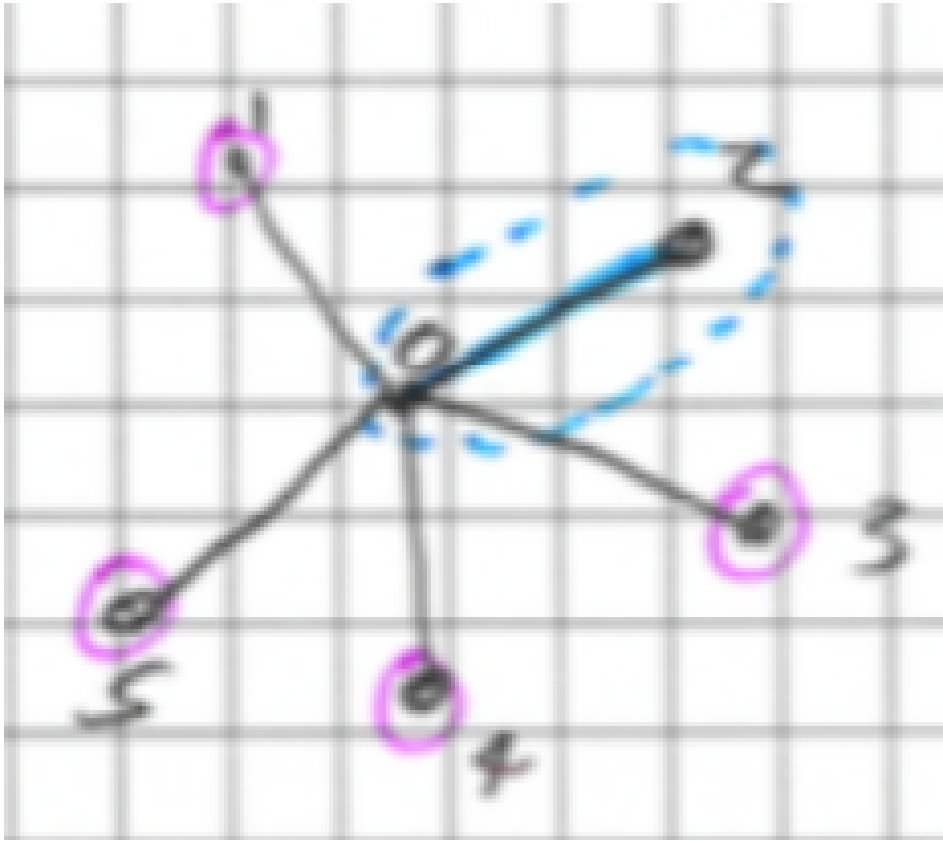
- There must be at least two free non-isolated vertices,

First optimization



- There must be at least two free non-isolated vertices,
- otherwise there is no augmenting path.

First optimization



- There must be at least two free non-isolated vertices,
- otherwise there is no augmenting path.
- Having multiple free vertices does not guarantee an augmenting.

Second optimization

Use `extend-alternating` rather than `extend-paths-by-1` so that alternating paths are assured by construction. No need to check.

Third optimization

Only need to check for augmenting path when number of edges is odd.
Then checking augmenting is simply checking $p_0 \in \mathit{free}$.

```
5.1  Function find-augmenting-path-or-none(adj, E, M)
5.2      free ← generate-free-vertices()
5.3      if |free| < 2 then
5.4          return None
5.5      Ψ ← {[v, u] | u ∈ free, {u, v} ∈ E}
5.6      k ← 1
5.7      while Ψ ≠ ∅ do
5.8          if odd(k) then
5.9              for p ∈ Ψ do
5.10                 if p_0 ∈ free then
5.11                     return p
5.12         Ψ ← extend-alternating(adj, M, k, free, Ψ)
5.13         k ← k + 1
5.14     return None
```

Algorithm 5: Find an augmenting path if possible

How to constructing alternating paths

- If k is odd, collect a matching edge.
- If k is even, collect a non-matching edge.
- If found non-matching edge, check for free vertex.
  - If free vertex, then short-circuit!!
    We found an augmenting path.

```
6.1  Function extend-alternating(adj, M, k, free, Ψ)
6.2      for p ∈ Ψ do
6.3          for u ∈ adj[p_0] do
6.4              if u ∉ p then
6.5                  if odd(k) ∧ {u, p_0} ∈ M then
6.6                      collect [u, p_0, p_1, ...p_k]
6.7                  else if even(k) ∧ {u, p_0} ∉ M then
6.8                      p' ← [u, p_0, p_1, ...p_k]
6.9                      if u ∈ free then
6.10                         return {p'}
6.11                     else
6.12                         collect p'
6.13     return collected paths
```

Algorithm 6: Extend alternating paths if possible

End of video 6