Bipartite Graphs

- What is a bipartite graph?
- Properties of bipartite graphs
- Matching and maximum matching
  - alternative paths
  - augmenting paths
- Hopcroft-Karp algorithm

Bipartite Graph

- 1. A graph G is *bipartite* if the node set V can be partitioned into two sets V_1 and V_2 in such a way that no nodes from the same set are adjacent.
- 2. The sets V_1 and V_2 are called the *color classes* of G and (V_1, V_2) is a bipartition of G. In fact, a graph being bipartite means that the nodes of G can be colored with at most two colors, so that no two adjacent nodes have the same color.

Bipartite Graph

3. We will depict bipartite graphs with their nodes colored black and white to show one possible bipartition.
4. We will call a graph *m by n bipartite*, if $|V_1| = m$ and $|V_2| = n$, and a graph a *balanced bipartite graph* when $|V_1| = |V_2|$.

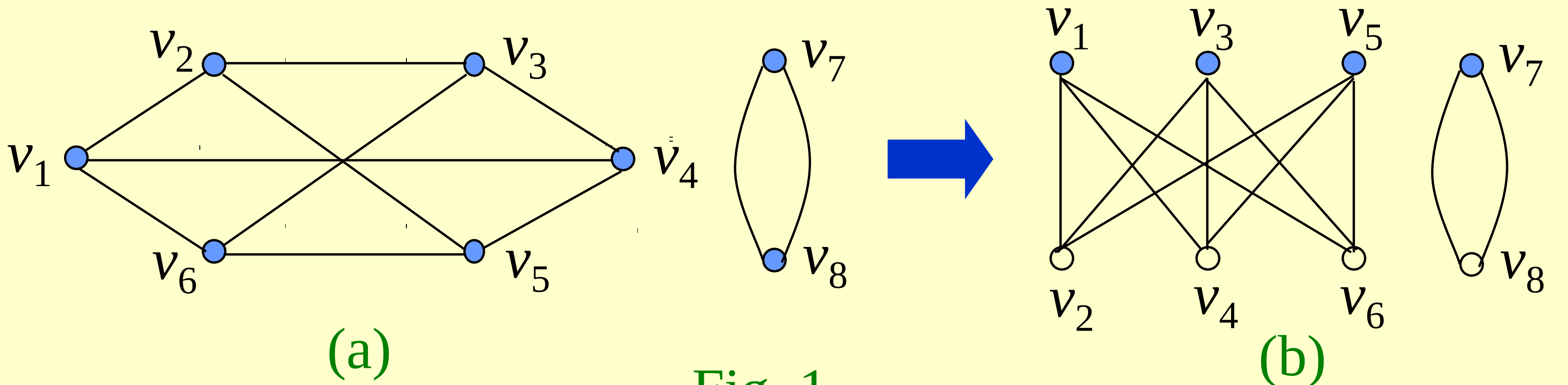


Fig. 1

Properties

Property 3.1 A connected bipartite graph has a unique bipartition.

Property 3.2 A bipartite graph with no isolated nodes and *p* connected components has 2^{p-1} bipartitions.

For example, the bipartite graph in the above figure has two bipartitions. One is shown in the figure and the other has $V_1 = \{v_1, v_3, v_5, v_8\}$ and $V_2 = \{v_2, v_4, v_6, v_7\}$.

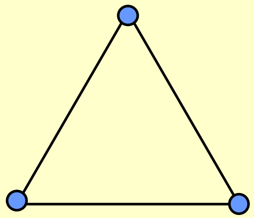
Properties

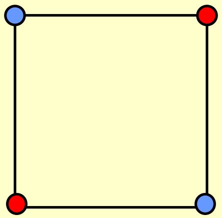
The following theorem belongs to König (1916).

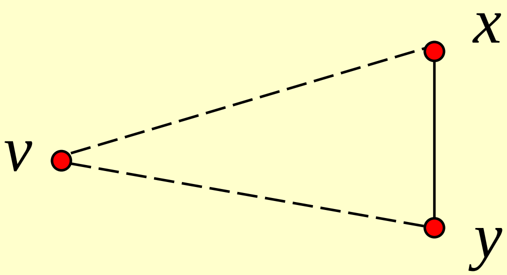
Theorem 3.3 A graph *G* is bipartite if and only if *G* has no cycle of odd length.

Corollary 3.4 A connected graph is bipartite if and only if for every node *v* there is no edge (x, y) such that *distance*(v, x) = *distance*(v, y).

Corollary 3.5 A graph *G* is bipartite if and only if it contains no closed walk of odd length.



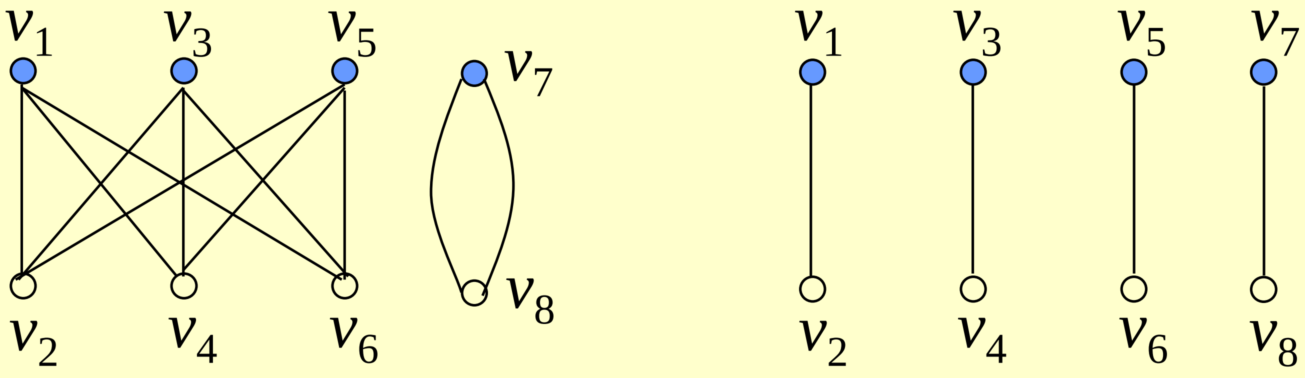




Matching and Maximum Matching

- **Maximum matching**
  - A set of edges in a bipartite graph *G* is called a *matching* if no two edges have a common end node.
  - A matching with the largest possible number of edges is called a *maximum matching*.

Example. A maximum matching for the bipartite graph in Fig. 1(b) is shown below.



Maximum Matching

■ Maximum matching

- Many discrete problems can be formulated as problems about maximum matchings. Consider, for example, probably the most famous:

  A set of boys each know several girls, is it possible for the boys each to marry a girl that he knows?

This situation has a natural representation as a bipartite graph with bipartition (V_1, V_2), where V_1 is the set of boys, V_2 the set of girls, and an edge between a boy and a girl represents that they know one another. The marriage problem is then the problem: does a maximum matching of G have $|V_1|$ edges?

Alternative and Augmenting Path

- **Maximum matching**

  Let M be a matching of a graph G.

  - A node v is said to be *covered*, or saturated by M, if some edge of M is incident with v. We will also call an unsaturated node *free*.
  - A path or cycle is *alternative*, relative to M, if its edges are alternatively in M and $E \backslash M$.
  - A path is an *augmenting path* if it is an alternating path with free origin and terminus.
  - P - a path. C - a cycle.

    $|P|$ - the number of edges in P

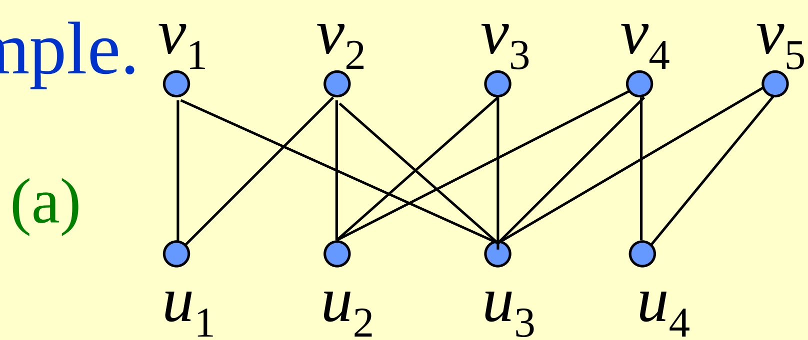
    $|C|$ - the number of edges in C.

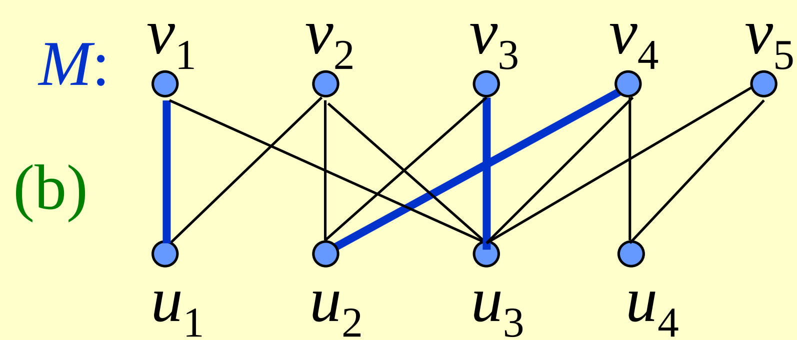
Properties of Matchings

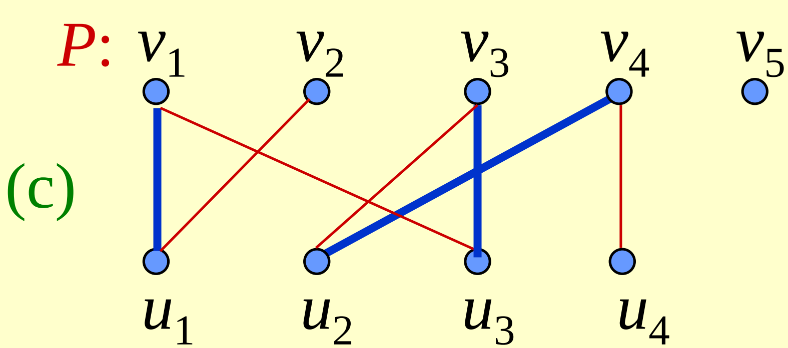
Property 3.3 Let M be a matching and P an augmenting path relative to M. Then the symmetric difference of M and P, denoted $M \Delta P$, is also a matching of G and $|M \Delta P| = |M| + 1$.

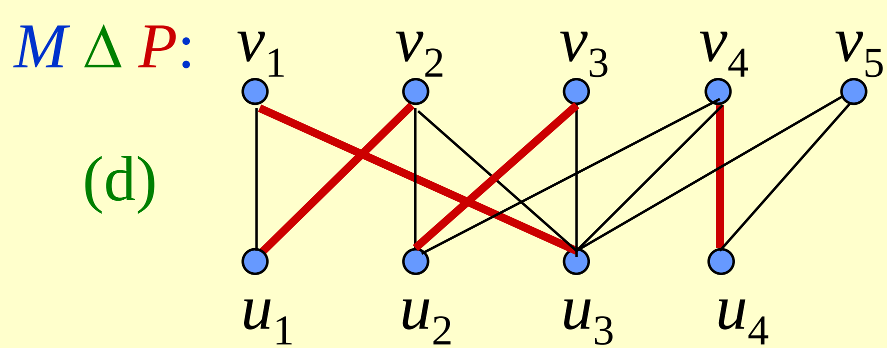
$M \Delta P = (M \backslash P) \cup (P \backslash M)$

Example.







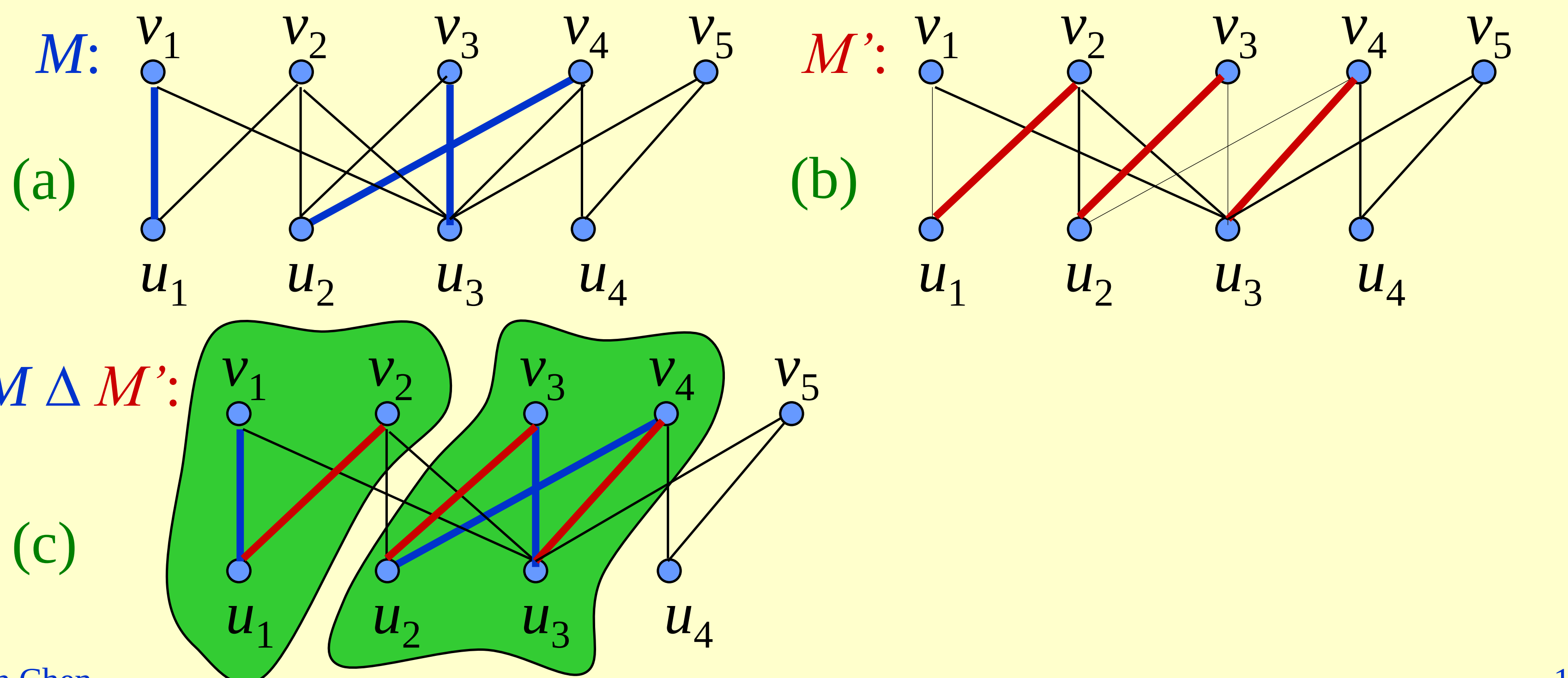


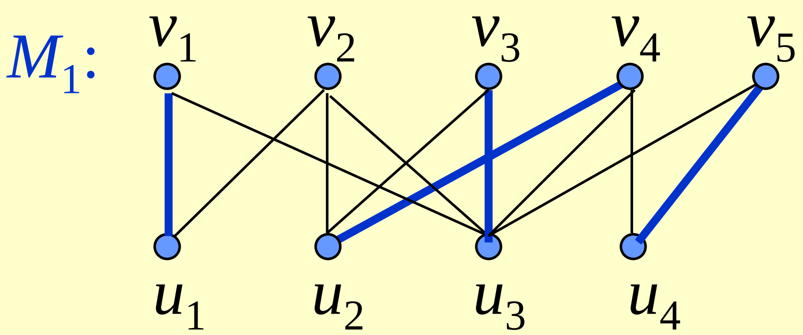
Properties of Matchings

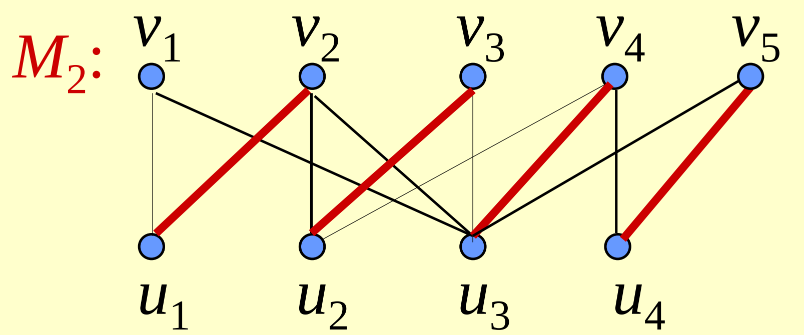
Property 3.4 Let M and M' be matchings in G. Then, each connected component of $M \Delta M'$ is one of the following:

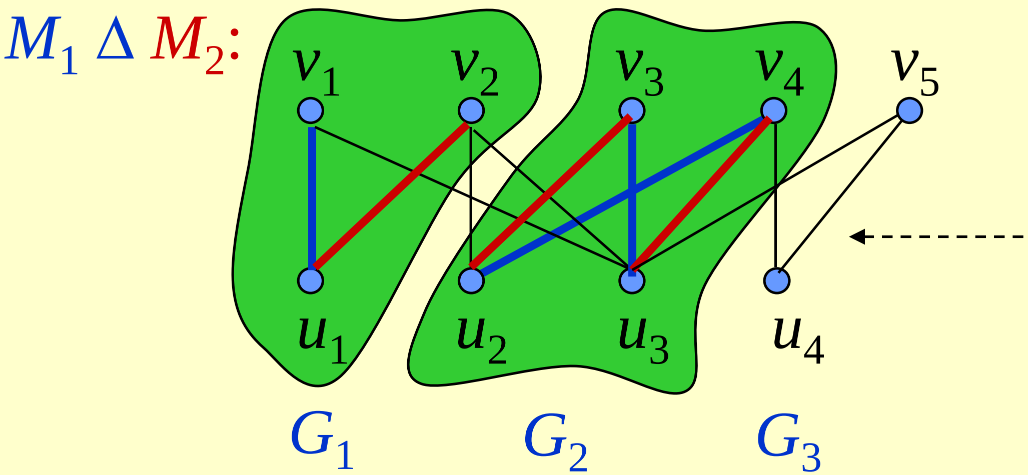
(1) an even cycle with edges alternatively in $M \backslash M'$ and $M' \backslash M$, or

(2) a path whose edges are alternatively in $M \backslash M'$ and $M' \backslash M$.









This edge is considered to be an augmenting path relative to M_1, also to M_2.

Properties of Matchings

Proposition 3.5 Let M and M' be matchings in G. If $|M| = r$, $|M'| = s$ and $s > r$, then $M \Delta M'$ contains at least $s - r$ node-disjoint augmenting paths relative to M.

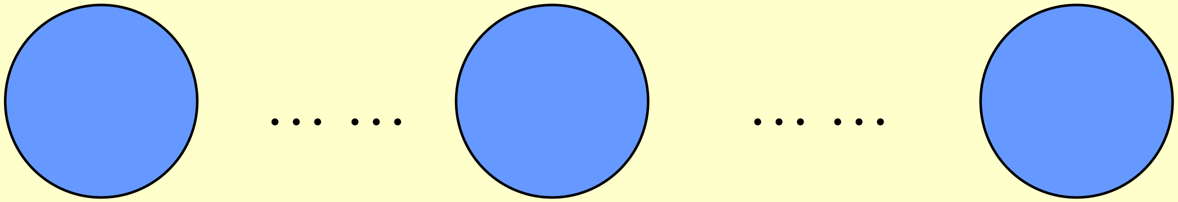
Proof. Let the components of $M \Delta M'$ be $C_1, C_2, ..., C_k$. Let $f(C_i) = |C_i \cap M'| - |C_i \cap M|$ $(1 \leq i \leq k)$. Then it follows from Property 3.4 that $f(C_i) \in \{-1, 0, 1\}$ for each $1 \leq i \leq k$. $f(C_i) = 1$ if and only if C_i is an augmenting path relative to M. To complete the proof, we need only observe that

$$\sum_{j=1}^{k} f(C_i) = |M' \backslash M| - |M \backslash M'| = |M'| - |M| = s - r.$$

Thus, there are at least $s - r$ components with $f(C_i) = 1$, and at least $s - r$ node-disjoint augmenting paths relative to M.

C_i contains one more edge from M' or from M.

$M \, \Delta \, M'$



$C_1 = |M'_1| - |M_1|$ $\quad$ $C_i = |M'_i| - |M_i|$ $\quad$ $C_k = |M'_k| - |M_k|$

If C_i is a cycle or an even path, $f(C_i) = 0$.
If C_i is an augmenting path relative to M', $f(C_i) = -1$.
If C_i is an augmenting path relative to M, $f(C_i) = 1$.

$$\sum_{j=1}^{k} f(C_i) = s - r$$

Properties of Matchings

Combining Proposition 3.5 with Property 3.3, we can deduce the following theorem (Berge, 1957).

Theorem 3.6 *M* is a maximum matching if and only if there is no augmenting path relative to *M*.

Proof. *If-part*. If there is no augmenting path, *M* must be a maximum matching. Otherwise, let *M'* be a maximum matching. According to Proposition 3.5, $M \Delta M'$ contains at least $|M'| - |M|$ node-disjoint augmenting paths relative to *M*. Contradiction.

Only-if-part. If *M* is a maximum matching, there is definitely no augmenting path relative to *M*. Otherwise, let *P* be an augmenting path relative to *M*. Then, $M \Delta P$ is a larger matching than *M*.

Properties of Matchings

The above theorem implies the following property.

Property 3.7 If M is a matching of G, then there exists a maximum matching $\boldsymbol{M}$ of G such that the set of nodes covered by M is also covered by $\boldsymbol{M}$.

Proof. If M is a maximum matching, the property trivially holds. Otherwise, consider an augmenting path P relative to M. Then, according to Property 3.1, $M \Delta P$ is also a matching of G with

$$|M \Delta P| = |M| + 1.$$

Moreover, the nodes covered by M are also covered by $M \Delta P$. Repeating the above process will prove the property.

Properties of Matchings

The following theorem was obtained by Dulmage and Mendelsohn (1958).

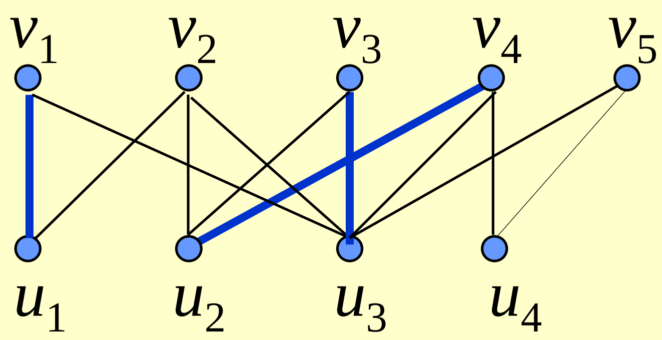
Theorem 3.8 Let G be a bipartite graph with bipartition (V_1, V_2). Let M_1 and M_2 be matchings in G. Then, there is a matching $M \subseteq M_1 \cup M_2$ such that M covers all the nodes of V_1 covered by M_1 and all the nodes of V_2 covered by M_2.

Proof. Let U_i be the nodes of V_1 covered by M_i ($i = 1, 2$). Let W_i be the nodes of V_2 covered by M_i ($i = 1, 2$). Let $G_1, G_2, \ldots, G_k$ be the connected components of $M_1 \Delta M_2$. By Property 3.4, each G_i ($1 \le i \le k$) is an even cycle or a path. Let $M_{1i} = G_i \cap M_1$ and $M_{2i} = G_i \cap M_2$. Define in each G_i a matching π_i:

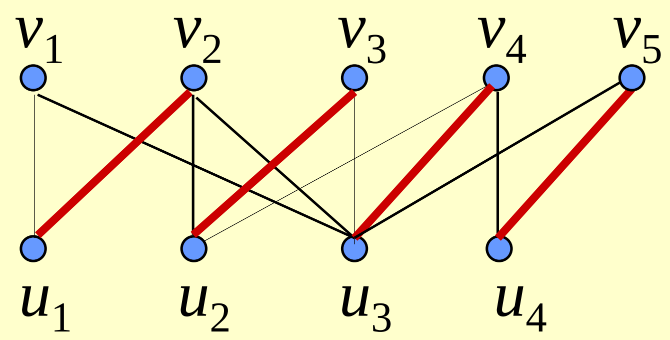
$$\pi_i = \begin{cases} M_{1i} & \text{if } G_i \text{ is a cycle} \\ M_{1i} & \text{if there is a node } v \in V \cap (U_1 \backslash U_2) \text{ in } G_i \\ M_{2i} & \text{if there is a node } v \in V \cap (W_2 \backslash W_1) \text{ in } G_i \end{cases}$$

Then, it is not difficult to check that $M = (M_1 \cap M_2) \cup \pi_1 \cup \pi_2 \ldots \cup \pi_k$ is the required matching.

M_1:



M_2:



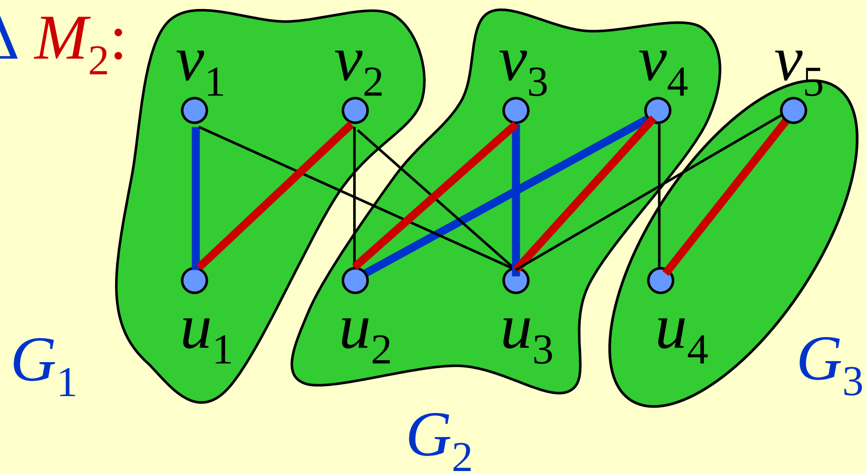
$U_1 = \{v_1, v_3, v_4\}$

$U_2 = \{v_2, v_3, v_4, v_5\}$

$W_1 = \{u_1, u_2, u_3\}$

$W_2 = \{u_1, u_2, u_3, u_4\}$

$M_1 \Delta M_2$:



$\pi_1 = M_{11} = G_1 \cap M_1$

$\pi_2 = M_{12} = G_2 \cap M_1$

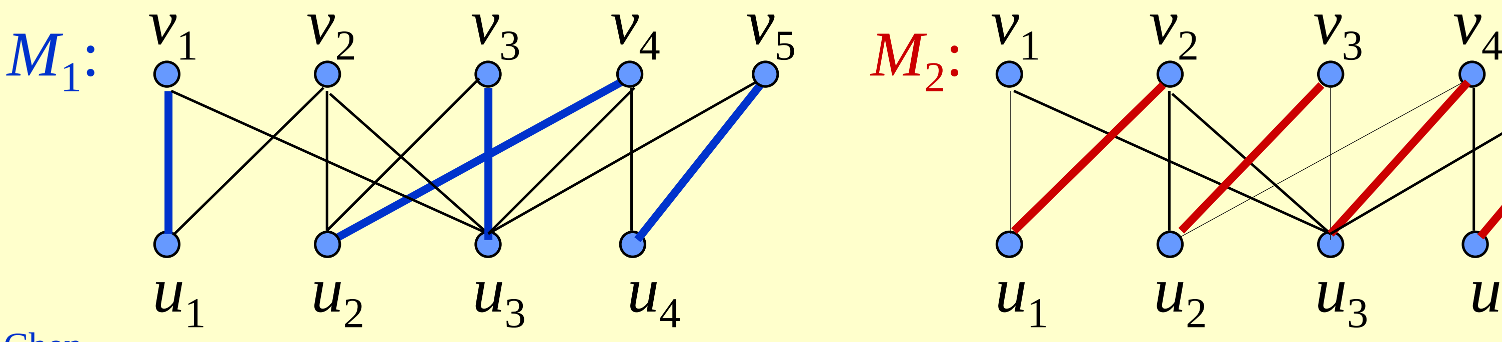
$\pi_3 = M_{23} = G_3 \cap M_2$

$$M = (M_1 \cap M_2) \cup \pi_1 \cup \pi_2 \cup \pi_3$$

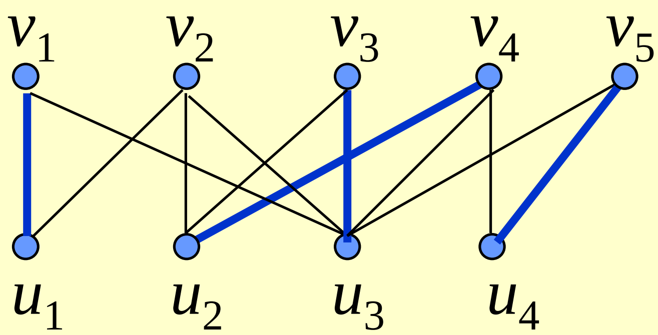
Properties of Matchings

Theorem 3.9 A maximum matching M of a bipartite graph G can be obtained from any other maximum matching M' by a sequence of transfers along alternating cycles and paths of even length.

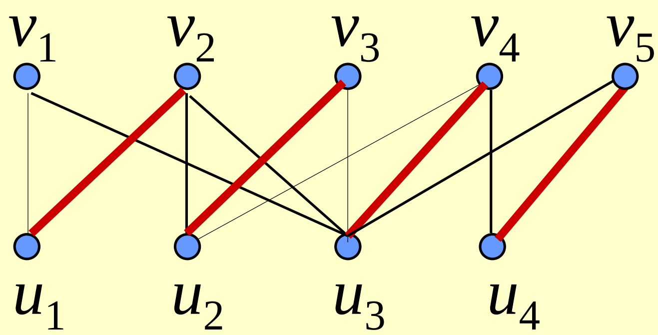
Proof. By Property 3.4, every component of $M \Delta M'$ is an alternating even cycle or an alternating path relative to M'. By Property 3.3 and Theorem 3.6, a component of $M \Delta M'$, if it is a path, must be of even length. (Otherwise, if it is an odd path, it must be an augmenting path relative to M or to M', contradicting the fact that both M and M' are maximum.) Then, changing M' in each component in turn will transform M' into M.



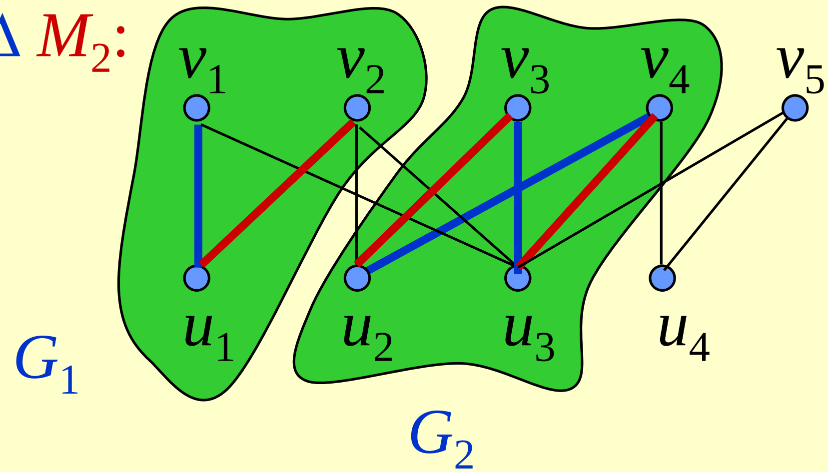
M_1:



M_2:

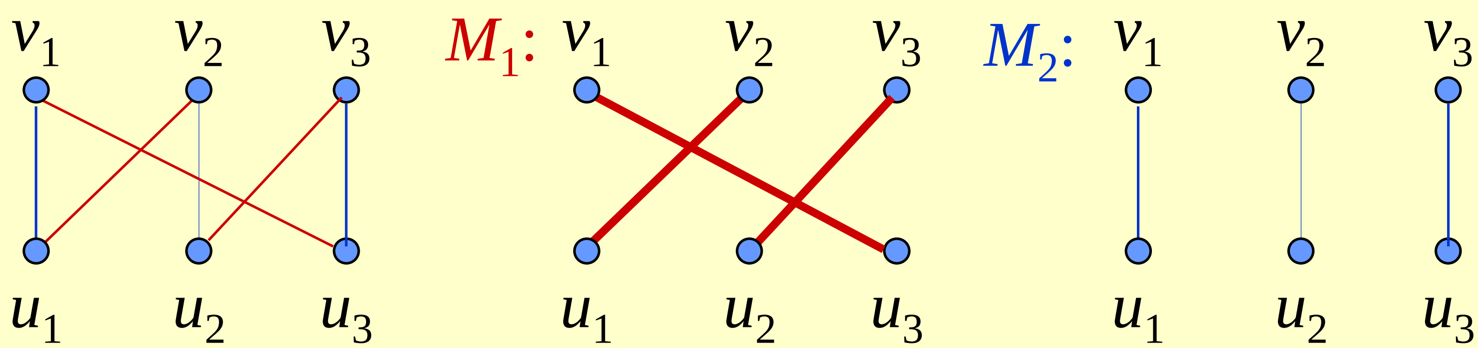


$M_1 \Delta M_2$:



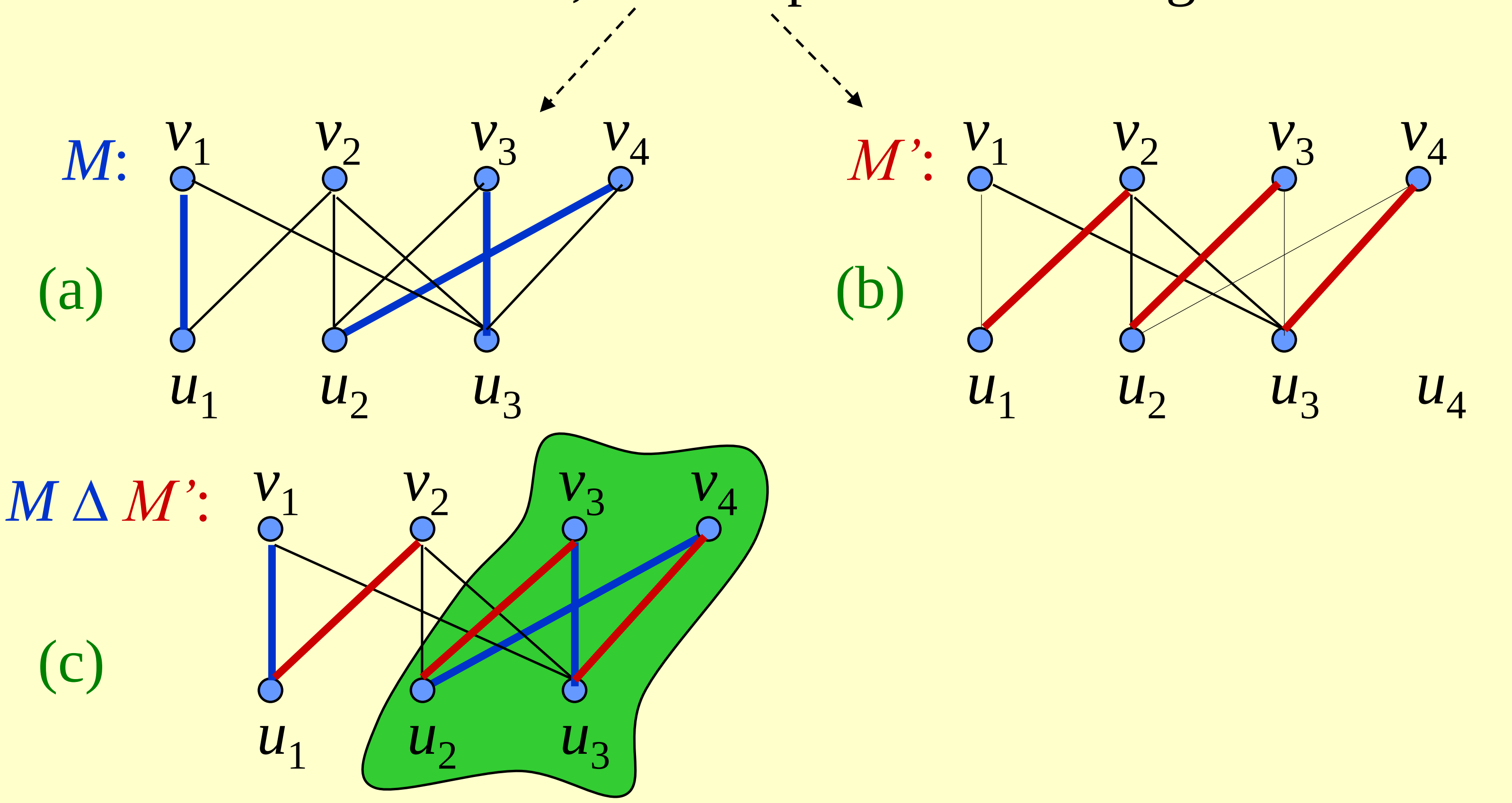
Properties of Matchings

A perfecting matching of a graph G is a matching which covers every node of G. Clearly, if a graph has two perfect matchings M and M', all components of $M \Delta M'$ are even cycles. Therefore, according to Theorem 3.9, we can deduce the following result.



Corollary 3.10 Assume that bipartite graph G has a perfect matching M. Then, any other perfect matching can be obtained from M by a sequence of transfers along alternating cycles relative to M.

Maximum, but not perfect matching



Algorithm

- **Algorithms - finding a maximum matching**

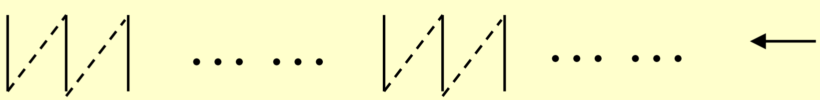
Lemma 3.11 Let M be a matching with $|M| = r$ and suppose that the cardinality of a maximum matching is s. Then, there exists an augmenting path relative to M of length at most $2\lfloor r/(s-r)\rfloor + 1$.

Proof. Let M' be a maximum matching. Then, by Proposition 3.5, $M \Delta M'$ contains $s - r$ node-disjoint augmenting paths relative to M. It is easy to see that these paths contain at most r edges from M. So one of these augmenting paths must contain at most $\lfloor r/(s-r)\rfloor$ edges from M and so at most $2\lfloor r/(s-r)\rfloor + 1$ edges altogether.

Proof. Let M' be a maximum matching. Then, by Proposition 3.5, $M \,\Delta\, M'$ contains s - r node-disjoint augmenting paths relative to M. It is easy to see that these paths contain at most r edges from M. So one of these augmenting paths must contain at most $\lfloor r/(s-r) \rfloor$ edges from M and so at most $2\lfloor r/(s-r) \rfloor + 1$ edges altogether.

Since $s > r$, there is at lease $s - r$ augmenting paths in $M \,\Delta\, M'$. On each of them the number of edges in M' is one larger than the number of edges in M.

Consider these augmenting paths:



P_1 P_i

at least $s - r$ augmenting paths relative to M

If each P_i contains more than $\lfloor r/(s-r) \rfloor$ edges, then M will have more than r edges. Contradiction.

Algorithm

Lemma 3.12 Let M be a matching and P be a shortest augmenting path relative to M. Let Q be an augmenting path relative to $M \Delta P$. Then, $|Q| \geq |P| + 2|P \cap Q|$.

Proof. Consider $M' = M \Delta P \Delta Q$. Then, M' is a matching. By Property 3.3, $|M'| = |M \Delta P| + 1 = (|M| + 1) + 1 = |M| + 2$. According to Proposition 3.5, $M \Delta M'$ contains at least two node-disjoint augmenting paths relative to M. Let $P_1, \ldots, P_k$ $(k \geq 2)$ be such paths. Since P is a shortest augmenting path relative to M, we have

$$|M \Delta M'| = |M \Delta (M \Delta P \Delta Q)| = |\varnothing \Delta P \Delta Q|$$

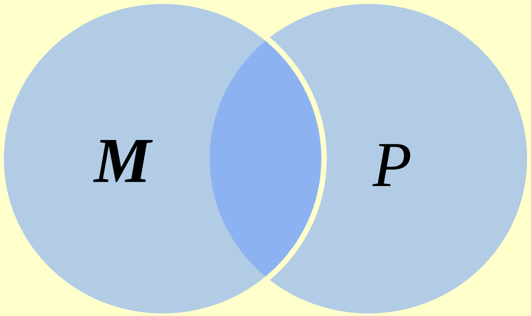
$$= |P \Delta Q| \geq |P_1| + \ldots + |P_k| \geq 2|P|.$$

Note that $|P \Delta Q| = |P| + |Q| - 2|P \cap Q| \geq 2|P|$. Therefore, we have $|Q| \geq |P| + 2|P \cap Q|$.

About symmetric difference

Commutative:

$\boldsymbol{M} \Delta P = P \Delta \boldsymbol{M}$

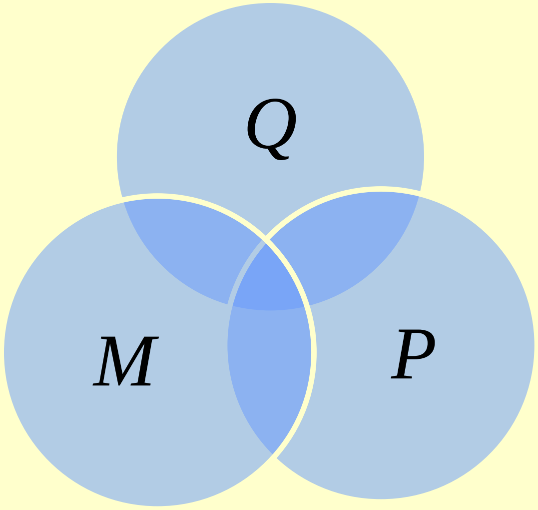


Associative:

$\boldsymbol{M} \Delta P \Delta Q$

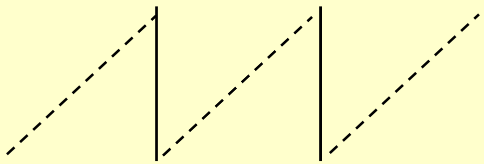
$= \boldsymbol{M} \Delta (P \Delta Q)$

$= (\boldsymbol{M} \Delta P) \Delta Q$

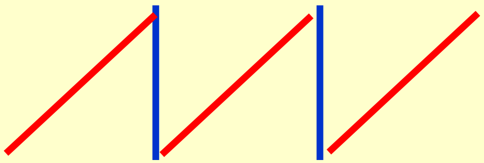


Example.

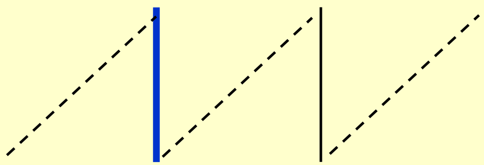
G:



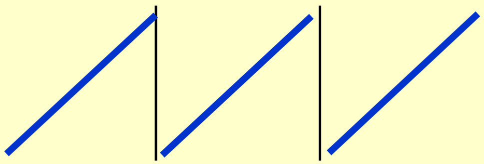
Q:



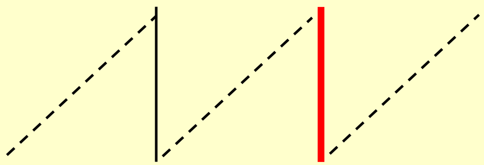
M:



$M \,\Delta\, P \,\Delta\, Q$:



P:

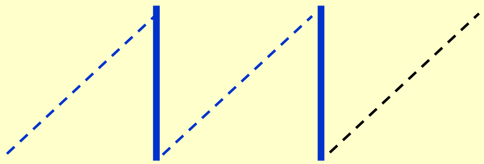


$|P| = 1$

$|Q| = 5$

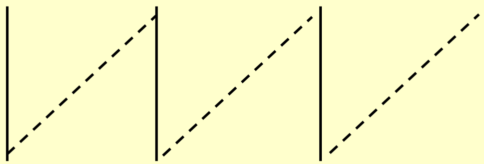
$|P \cap Q| = 1$

$M \,\Delta\, P$:

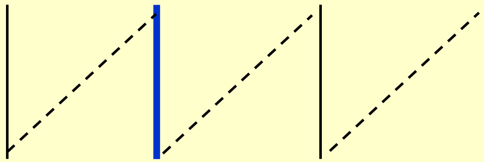


Example.

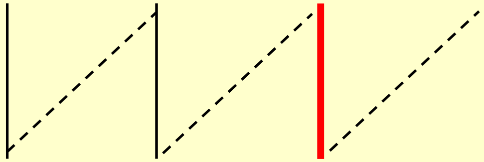
G:



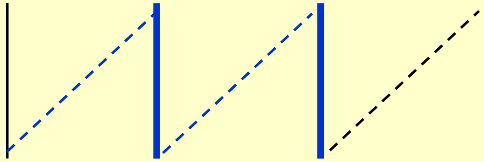
M:



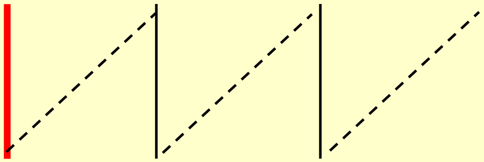
P:



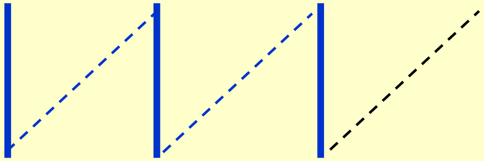
$M \Delta P$:



Q:



$M \Delta P \Delta Q$:



$|P| = 1$

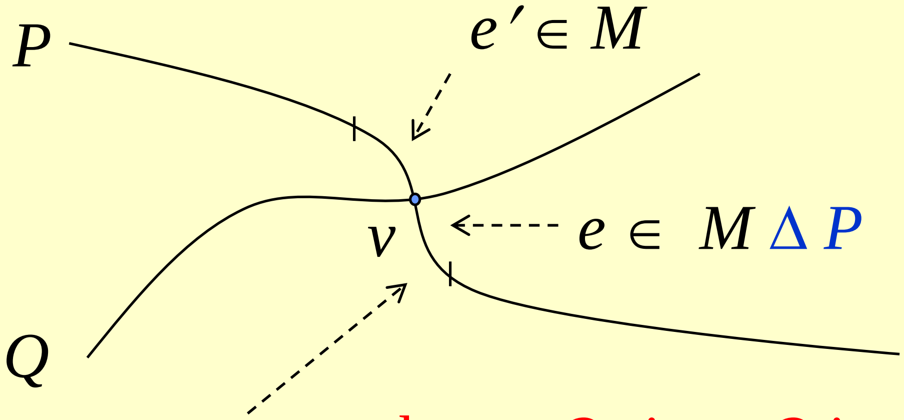
$|Q| = 1$

$|P \cap Q| = 0$

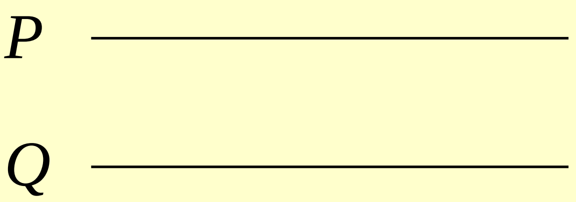
Algorithm

Corollary 3.13 Let P be a shortest augmenting path relative to a matching M, and Q be a shortest augmenting path relative to $M \Delta P$. Then, if $|P| = |Q|$, the paths P and Q must be node-disjoint. Moreover, Q is also a shortest augmenting path relative to M.

Proof. According to Lemma 3.12, we have $|P| = |Q| \geq |P| + 2|P \cap Q|$. So $P \cap Q = \Phi$. Thus, P and Q are edge-disjoint. Assume that P and Q share a common node v. Consider the edge e incident with v in $M \Delta P$. Then, P and Q must share e, contradicting $P \cap Q = \Phi$. Therefore, P and Q are also node-disjoint.



P and Q are node-disjoint.



e must be on Q since Q is an augmenting path of $M \Delta P$.

Algorithm

Hopcroft-Karp algorithm (1973)

The whole computation process is divided into a number of stages, at which some partial matching has been constructed and some way is sought to increase it. At stage i, we have the matching M_i and we search for $\{Q_1, Q_2, ..., Q_k\}$, a maximal set of node-disjoint, shortest augmenting paths, relative to M_i. Then, according to Corollary 3.13, Q_2 is a shortest augmenting path relative to $M \Delta Q_1$, Q_3 is a shortest augmenting path relative to $(M \Delta Q_1) \Delta Q_2$, ..., and Q_k is a shortest augmenting path relative to $(M \Delta Q_1 \Delta Q_2 ... \Delta Q_{k-2}) \Delta Q_{k-1}$. Therefore, the new matching for the next stage is formed as

$$M_{i+1} = M_i \Delta Q_1 \Delta Q_2 ... \Delta Q_k.$$

Algorithm

Proposition 3.14 Let s be the cardinality of a maximum matching in a bipartite graph G. Then, to construct a maximum matching by the above process requires at most $2\lfloor\sqrt{s}\rfloor + 2$ stages.

Proof. It can be derived from Property 3.3 and Lemma 3.11.

Then, the running time should be bounded by O($|E|\sqrt{s}$) since at each stage O($|E|$) edges will be visited.

Proof of Proposition 3.14

Let M_0, M_1, …, M_s be a sequence of matchings in G, where $M_0 = \Phi$ and $M_{i+1} = M_i \Delta P_i$ and P_i is shortest augmenting path relative to M_i for each $i = 1, …, s - 1$. Then, an upper bound for the number of stages we require can be given by bounding the number of distinct integers in the sequence $|P_0|$, $|P_1|$, …, $|P_s|$. Let $r = \lfloor s - \sqrt{s} \rfloor$. Then, since $|M_r| = r$, we have

$$|P_r| \leq 2 \lfloor r/(s - r) \rfloor \leq 2|\sqrt{s}| + 1.$$

Thus, for each $i \leq r$, $|P_i|$ is one of the $\lfloor \sqrt{s} \rfloor + 1$ odd number less than or equal to $2\lfloor \sqrt{s} \rfloor + 1$. Certainly, $s - r = \lceil \sqrt{s} \rceil$ paths contribute at most $\lceil \sqrt{s} \rceil$ distinct integers to the collection, and so the total number is at most

$$\lfloor \sqrt{s} \rfloor + 1 + \lceil \sqrt{s} \rceil \leq 2 \lfloor \sqrt{s} \rfloor + 2.$$

Algorithm

Main process:

$G \longrightarrow M_1 \longrightarrow G_1 \longrightarrow G_1' \longrightarrow$ *Node-disjoint augmenting paths* $\boldsymbol{P}_1$

Randomly choose some edges as M_1.

$M_1 \,\Delta\, \boldsymbol{P}_1$

$M_2 \longrightarrow G_2 \longrightarrow G_2' \longrightarrow$ *Node-disjoint augmenting paths* $\boldsymbol{P}_2$

$M_2 \,\Delta\, \boldsymbol{P}_2$

$M_3 \longrightarrow G_3 \longrightarrow G_3' \longrightarrow$ *Node-disjoint augmenting paths* $\boldsymbol{P}_3$

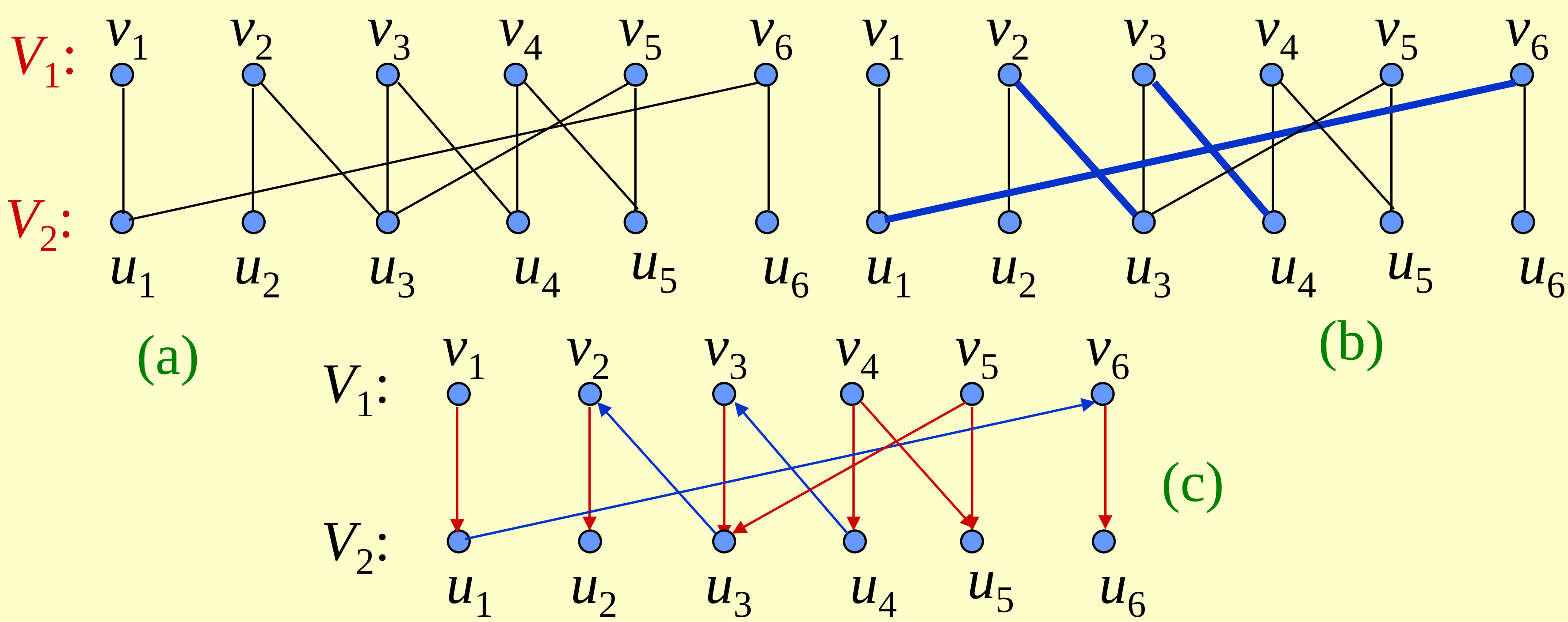
$M_3 \,\Delta\, \boldsymbol{P}_3$

$\vdots$

Algorithm

Let M_i be the matching of G produced at stage i. We define a directed graph G_i (called an *alternating graph*) with the same node set as G, but with edge set

$$E(G_i) = \{u \rightarrow v \mid u \in V_1, v \in V_2, \text{ and } (u, v) \in E \backslash M_i\}$$
$$\cup \{v \rightarrow u \mid u \in V_1, v \in V_2, \text{ and } (u, v) \in M_i\}.$$



Algorithm

First step:

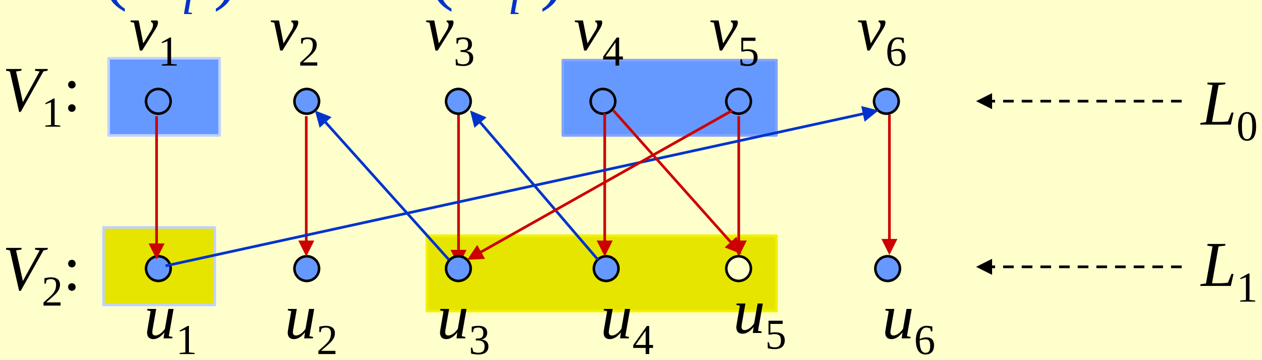
From G_i, construct a subgraph G_i' (called a *layered graph*) described below.

Let L_0 be the set of free nodes (relative to M_i) in V_1 and define L_j ($j > 0$) as follows:

$E_{j\text{-}1} = \{u \to v \in E(G_i) \mid u \in L_{j\text{-}1}, v \notin L_0 \cup L_1 \cup \ldots \cup L_{j\text{-}1}\}$,

$L_j = \{v \in V(G_i) \mid \text{for some } u, u \to v \in E_{j\text{-}1}\}$.

Define $j^* = \min\{j \mid L_j \cap \{\text{free nodes in } V_2\} \neq \Phi\}$. G_i' is formed with $V(G_i')$ and $E(G_i')$ defined below.



Algorithm

First step:

If $j^* = 1$, then

$V(G_1') = L_0 \cup (L_1 \cap \{\text{free nodes in } V_2\})$,

$E(G_1') = \{u \rightarrow v \mid u \in L_0 \text{ and } v \in \{\text{free nodes in } V_2\}\}$.

If $j^* > 1$, then

$V(G_i') = L_0 \cup L_1 \cup \ldots \cup L_{j^*-1} \cup (L_{j^*} \cap \{\text{free nodes in } V_2\})$,

$E(G_i') = E_0 \cup E_1 \cup \ldots \cup E_{j^*-2} \cup \{u \rightarrow v \mid u \in L_{j^*-1} \text{ and } v \in \{\text{free nodes in } V_2\}\}$.

With this definition of the graph G_i', directed paths from L_0 to L_{j^*} are precisely in one-to-one correspondence with shortest augmenting paths relative to M_i in G.

Algorithm

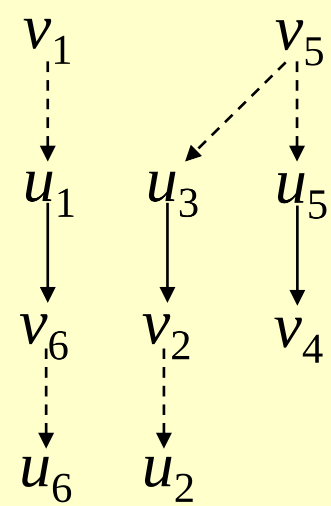
Second step:

In this step, we will traverse G_i' in a depth-first searching fashion to find a maximal set of node-disjoint paths from L_0 to L_{j*}.

- For this, a stack structure *stack* is used to control the graph exploring.
- In addition, we use *c*-*list*(v) to represent the set of v's child nodes.

Algorithm

G_2':



Algorithm *finding-augmenting-paths*(G_i')

begin

let x be the first element in L_0;

push(x, *stack*); mark v;

while *stack* is not empty **do** {

  v := top(*stack*);

  while c-*list*(v) $\neq \Phi$ **do** {

    let u be the first element in c-*list*(v);

    if u is marked **then** remove u from c-*list*(v)

    else {*push*(u, *stack*); mark u; $v := u$;}

  }

  if v is neither in L_{j^*} nor in L_0 **then** *pop*(*stack*)

  else {**if** v is in L_{j^*} **then** output all the elements in *stack*;

      (*all the elements in *stack* make up an augmenting path.*)

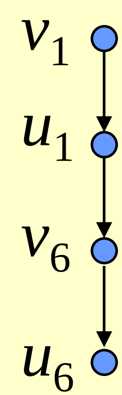
      remove all elements in *stack*;

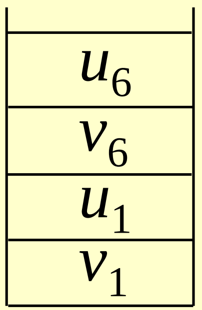
      let v be the next element in L_0;

      push(v, *stack*); mark v;

}

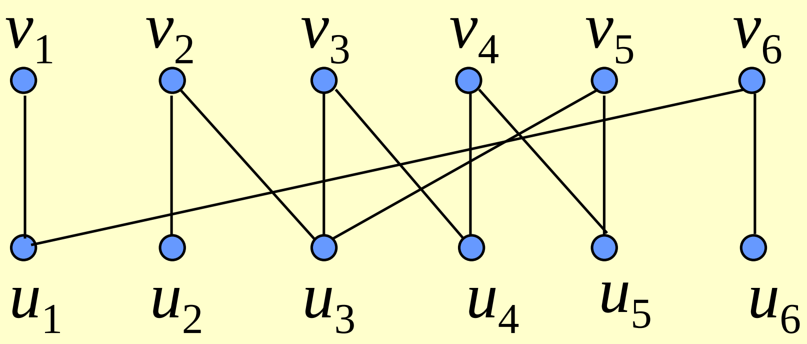
end





Example Trace

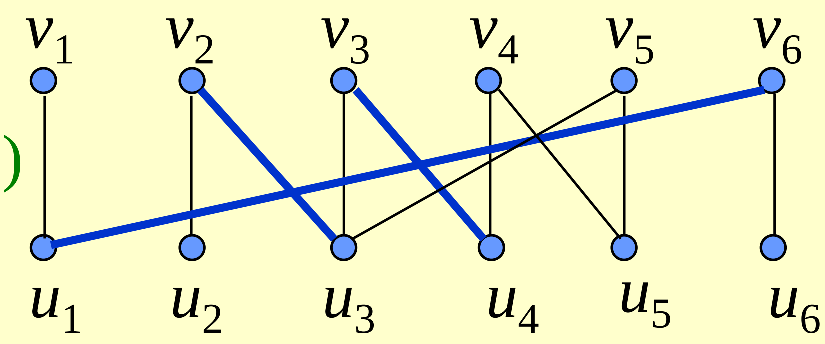
Example.



(a)

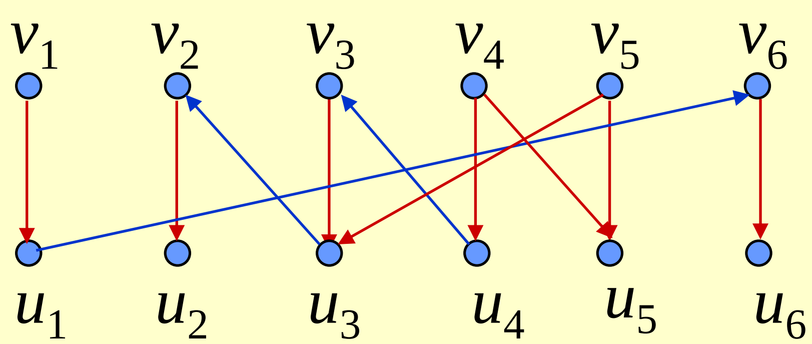
M_1:

(b)



G_1:

(c)

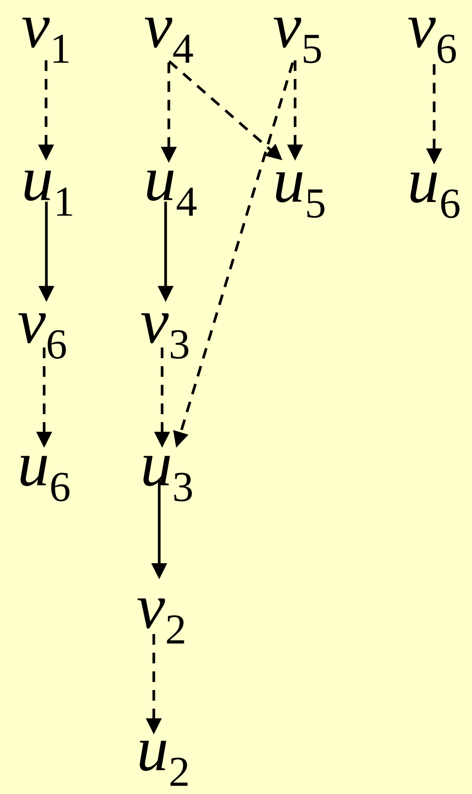


G_1' will be constructed as follows:

$L_0 = \{v_1, v_4, v_5\}$
$E_0 = \{(v_1, u_1), (v_4, u_4), (v_4, u_5), (v_5, u_3), (v_5, u_5)\}$
$L_1 = \{u_1, u_3, u_4, u_5\}$ /*$j^* = 1$ since u_5 is free.*/

(If u_5 is not free, the following layers will be constructed.)
$E_1 = \{(u_1, v_6), (u_3, v_2), (u_4, v_3)\}$
$L_2 = \{v_6, v_2, v_3\}$
$E_2 = \{(v_6, u_6), \cancel{(v_3, u_3)}, (v_2, u_2)\}$
$L_3 = \{u_6, u_2\}$
/*u_3 is not in L_3 since it already appears in L_1 .

G_1:

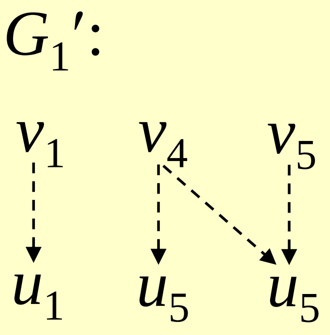


Example Trace

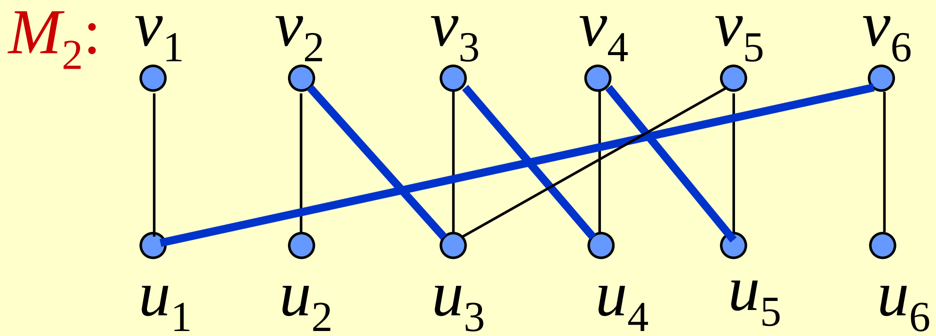
Since L_1 contains free node u_5 in V_2, $j^* = 1$. Therefore, we have

$$V(G_1') = \{v_1, v_4, v_5\} \cup \{u_5\}, \text{ and}$$

$$E(G_1') = \{(v_4, u_5), (v_5, u_5)\}$$



Note that $v_4 \rightarrow u_5$ is an augmenting path relative to M_1, and $v_5 \rightarrow u_5$ is another. By applying the second step of Hopcroft-Karp algorithm to G_1', $v_4 \rightarrow u_5$ will be chosen, yielding a new matching $M_2 = M_1 \Delta \{v_4 \rightarrow u_5\}$ as shown in the following figure.



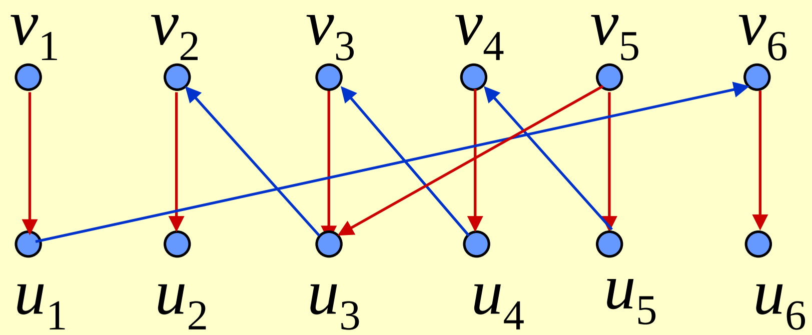
Example Trace

At a next stage, we will construct G_2 as shown in Figure 7. G_2' will then be constructed as follows:

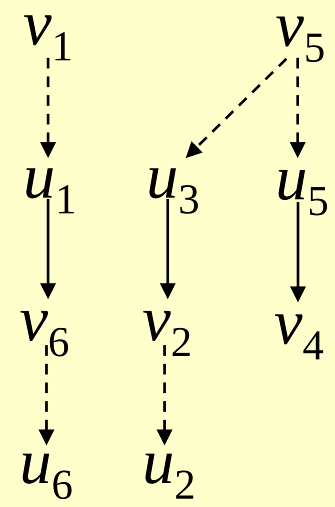
$L_0 = \{v_1, v_5\}$,
$E_0 = \{(v_1, u_1), (v_5, u_3), (v_5, u_5)\}$,
$L_1 = \{u_1, u_3, u_5\}$,
$E_1 = \{(u_1, v_6), (u_3, v_2), (u_5, v_4)\}$,
$L_2 = \{v_2, v_4, v_6\}$,
$E_2 = \{(v_2, u_2), (v_4, u_4), (v_6, u_6)\}$,
$L_3 = \{u_2, u_4, u_6\}$,
/*j^* = 3 since u_2 and u_6 are free.*/
$E_3 = \{(u_4, v_3)\}$,
$L_4 = \{v_3\}$,
$E_4 = \{(v_3, u_3)\}$.

This part will not be created.

G_2:



G_2':



Since L_3 contains two free nodes u_2 and u_6 in V_2, $j^* = 3$. So we have

$$V(G_2') = L_0 \cup L_1 \cup L_2 \cup \{u_2, u_6\}, \text{ and}$$
$$E(G_2') = E_0 \cup E_1 \cup \{(v_2, u_2), (v_6, u_6)\}.$$

$L_0 = \{v_1, v_5\}$,

$E_0 = \{(v_1, u_1), (v_5, u_3), (v_5, u_5)\}$,

$L_1 = \{u_1, u_3, u_5\}$,

$E_1 = \{(u_1, v_6), (u_3, v_2), (u_5, v_4)\}$,

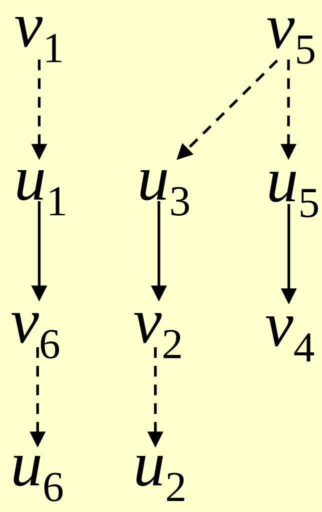
$L_2 = \{v_2, v_4, v_6\}$,

$E_2 = \{(v_2, u_2), (v_6, u_6), \cancel{(v_4, u_4)}\}$,

$L_3 = \{u_2, u_6, \cancel{u_4}\}$.

/*j^* = 3 since u_2 and u_6 are free.*/

G_2':



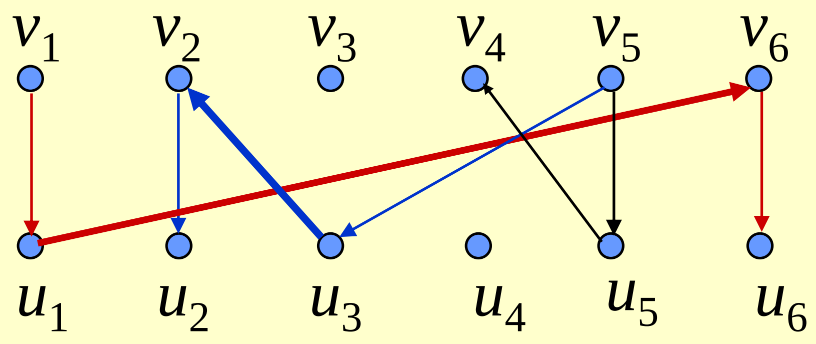
Example Trace

Since L_3 contains two free nodes u_2 and u_6 in V_2, $j^* = 3$. So we have

$$V(G_2') = L_0 \cup L_1 \cup L_2 \cup \{u_2, u_6\}, \text{ and}$$

$$E(G_2') = E_0 \cup E_1 \cup \{(v_2, u_2), (v_6, u_6)\}.$$

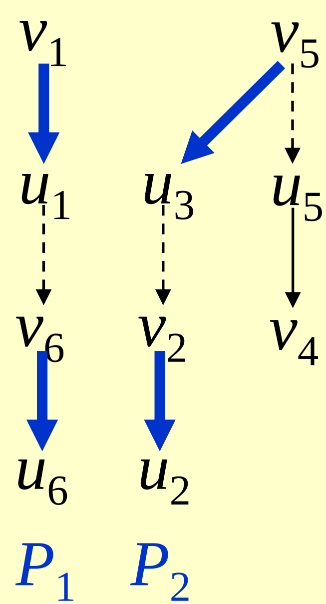
In Fig. 8, we show G_2', which contains two augmenting paths P_1 and P_2, where $P_1 = v_1 \rightarrow u_1 \rightarrow v_6 \rightarrow u_6$ (represented by red edges in Fig. 8) and $P_2 = v_5 \rightarrow u_3 \rightarrow v_2 \rightarrow u_2$ (represented by blue edges in Fig. 8) .



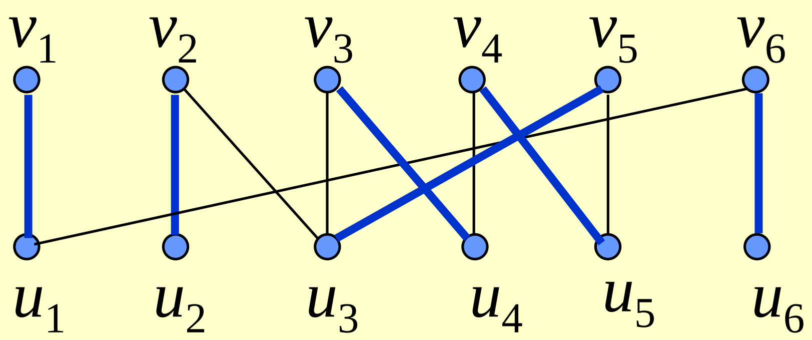
Example Trace

In Fig. 8, we show G_2', which contains two augmenting paths P_1 and P_2, where $P_1 = v_1 \rightarrow u_1 \rightarrow v_6 \rightarrow u_6$ and $P_2 = v_5 \rightarrow u_3 \rightarrow v_2 \rightarrow u_2$. By applying the second step of Hopcroft-Karp algorithm, these two augmenting paths will be found. The maximum matching $M_3 = M_2 \Delta P_1 \Delta P_2$ is shown in Fig. 9.

G_2':

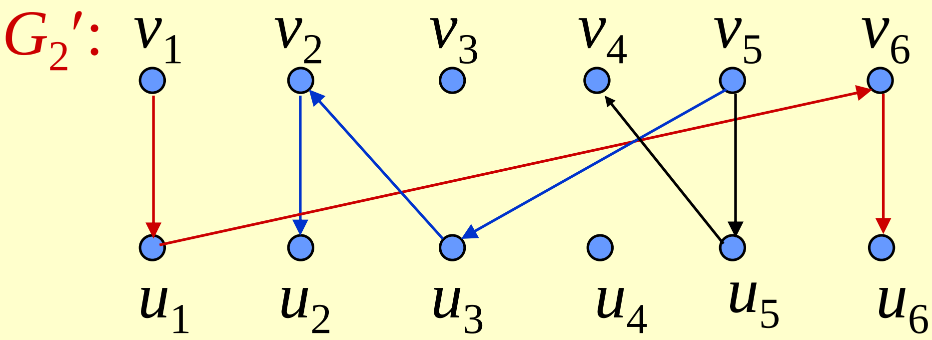


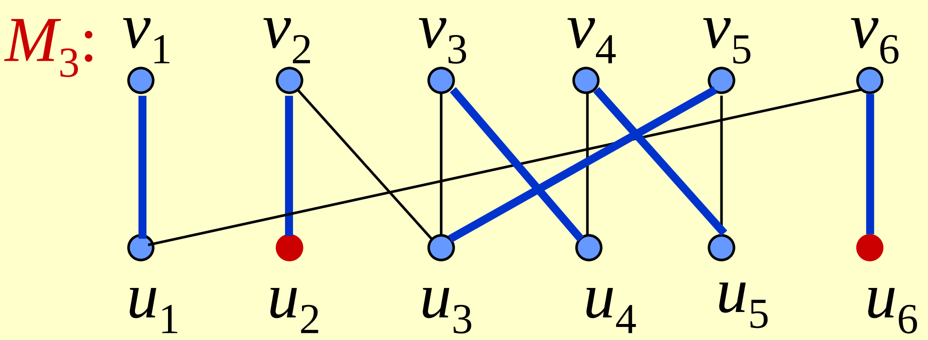
M_3:



Example Trace

In order to have a better understanding of the second step of Hopcroft-Karp algorithm, we trace the execution steps when applying it to the graph shown in Figure 8.

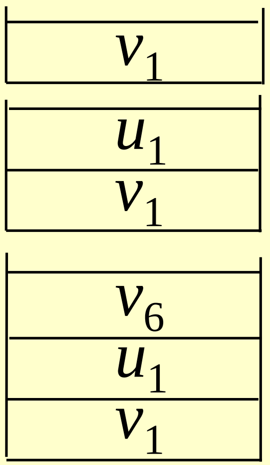


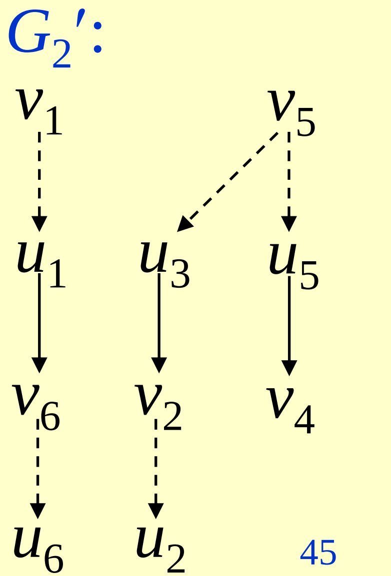


Step 1: $L_0 = \{v_1, v_5\}$
$push(v_1, stack)$; mark v_1;

Step 2: $c\text{-}List(v_1) = \{u_1\}$
$push(u_1, stack)$; mark u_1;

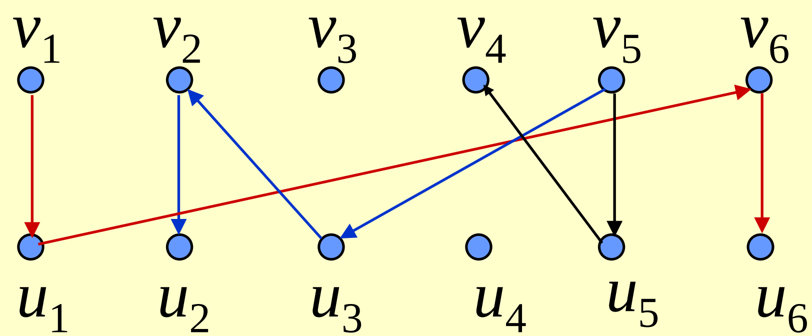
Step 3: $c\text{-}List(u_1) = \{v_6\}$
$push(v_6, stack)$; mark v_6;





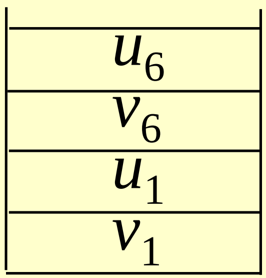
$L_3 = \{u_2, u_4, u_6\}$

$L_0 = \{v_1, v_5\}$



Step 4: $c\text{-}List(v_6) = \{u_6\}$;

$push(u_6, stack)$; mark u_6;



Step 5: $c\text{-}List(u_6) = \phi$;

u_6 is in L_3; /* $j^* = 3$.*/;

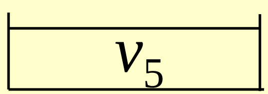
Output all the nodes in *stack*, which make up an augmenting path:

$v_1 \rightarrow u_1 \rightarrow v_6 \rightarrow u_6$;

empty stack;

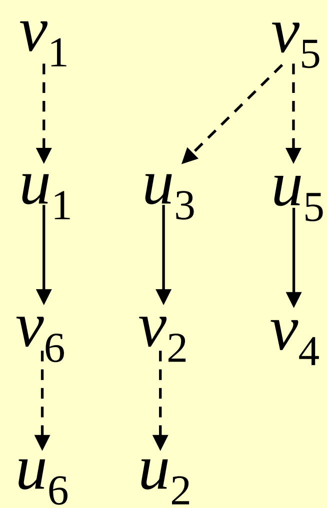
$push(v_5, stack)$; mark v_5;

/* v_5 is the next element in L_0.*/



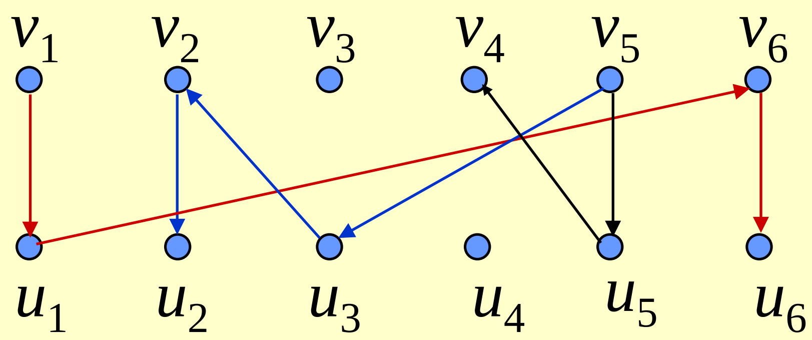
u_6 is a free node.

G_2':



$L_3 = \{u_2, u_4, u_6\}$

$L_0 = \{v_1, v_5\}$



Step 6: $c\text{-}List(v_5) = \{u_5, u_3\}$;

push(u_5, *stack*); mark u_5;

Step 7: $c\text{-}List(u_5) = \{v_4\}$;

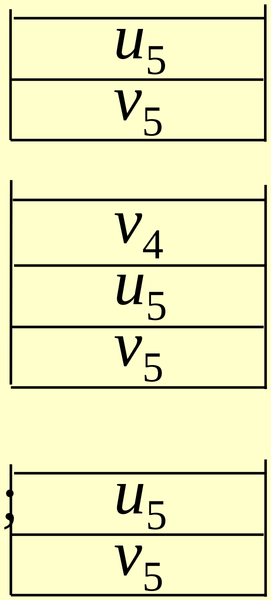
push(v_4, *stack*); mark v_4;

Step 8: $c\text{-}List(v_4) = \phi$;

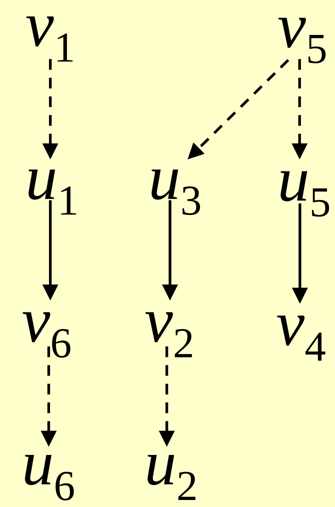
v_4 is neither in L_3 nor in L_0;

/*j* = 3. */

pop(*stack*);

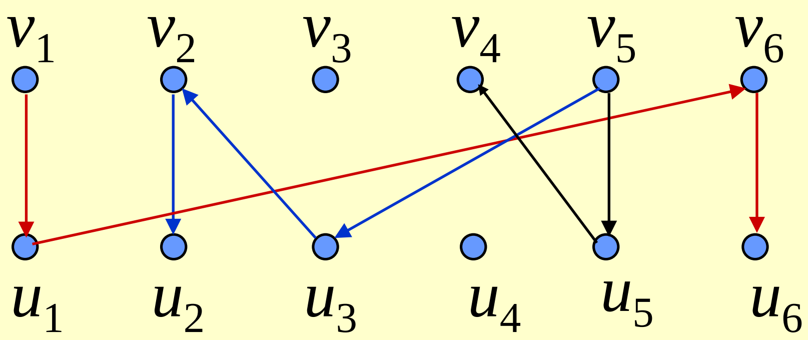


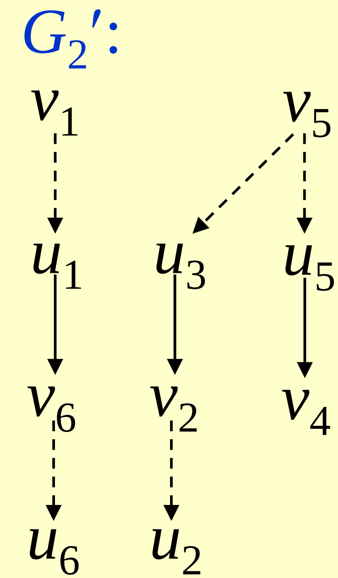
G_2':



$L_3 = \{u_2, u_4, u_6\}$

$L_0 = \{v_1, v_5\}$

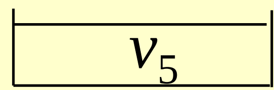




Step 9: $c\text{-}List(u_5) = \{v_4\}$;

v_4 is marked; remove v_4 from the list;

$c\text{-}List(u_5) = \phi$;



Step 10: u_5 is neither in L_3 nor in L_0;

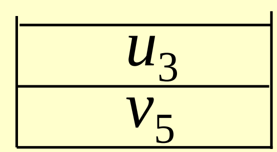
pop(*stack*);

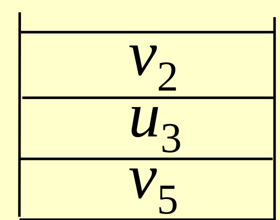
$c\text{-}List(v_5) = \{u_3\}$; /* u_5 is removed from $c\text{-}List(v_5)$.*/

Step 11: *push*(u_3, *stack*); mark u_3;

$c\text{-}List(u_3) = \{v_2\}$;

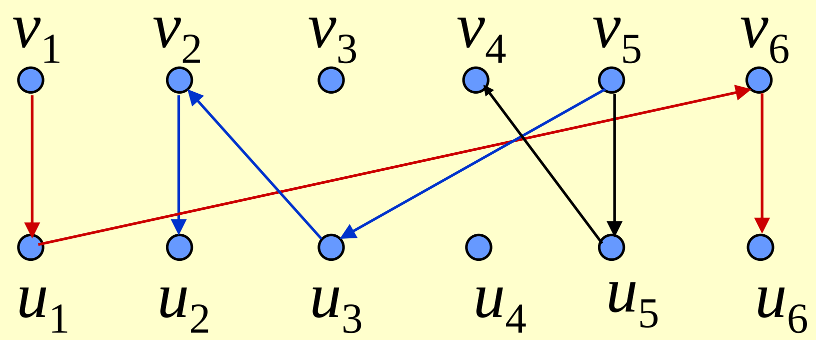
push(v_2, *stack*); mark v_2;

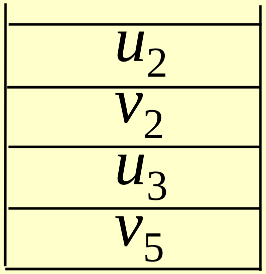




$L_3 = \{u_2, u_4, u_6\}$

$L_0 = \{v_1, v_5\}$





Step 12: $c\text{-}List(v_2) = \{u_2\}$;
$push(u_2, stack)$; mark u_2;

Step 13: $c\text{-}List(u_2) = \phi$;
u_2 is in L_3; /* j^* = 3. */;
Output all the nodes in *stack*, which make up an augmenting path:
$v_5 \rightarrow u_3 \rightarrow v_2 \rightarrow u_2$;
empty stack;
Now *stack* is empty and no element in L_0 will be pushed into *stack*.
Stop.

Bipartite Graph

Hopcroft-Karp algorithm (1973)

In the above example, we choose the matching shown in Figure 5(b) as an initial matching for ease of explanation. In fact, we can choose any edge in the bipartite graph as an initial matching and then apply Hopcroft-Karp algorithm. Of course, the final matching found may be different from that shown in Figure 9.

Project Requirement

1. Implementation of the algorithm in C++.
2. Documentation.