Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Domiyes

Algorithmen für Programmierwettbewerbe

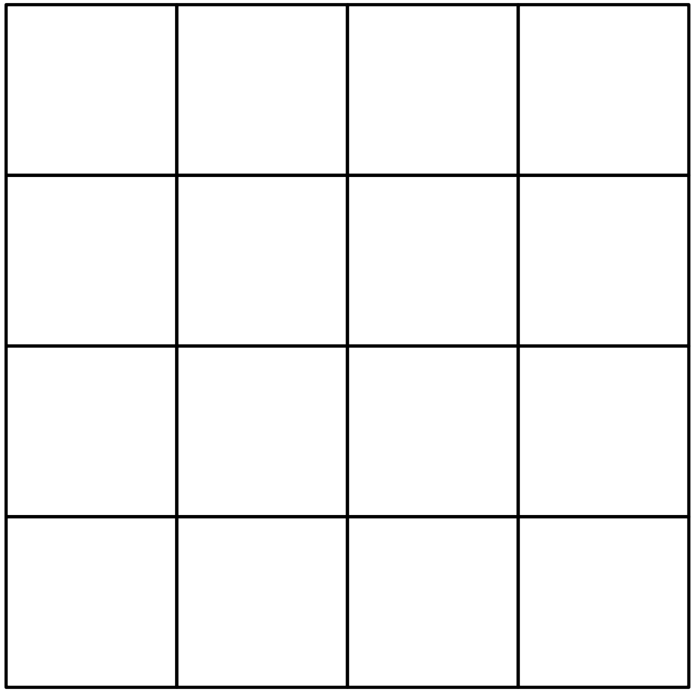
Sommersemester 2021

Sarah Bäurich

Florian Strunz

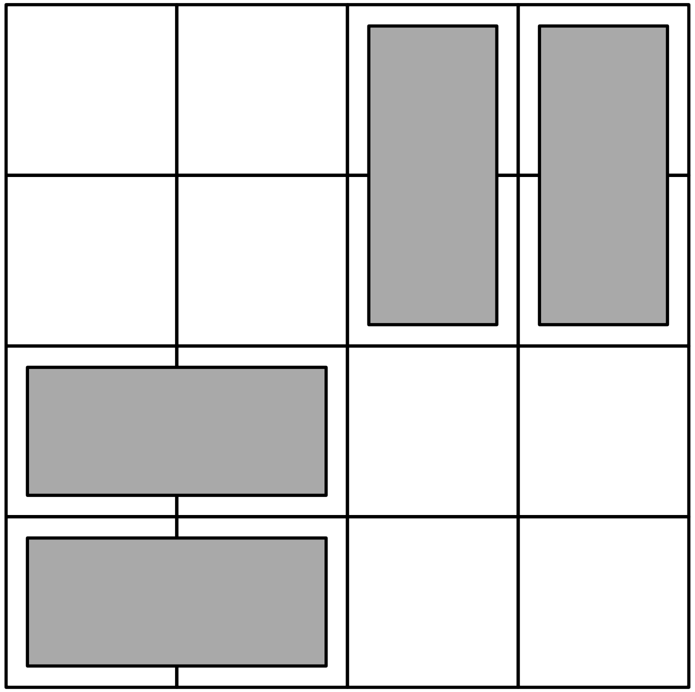
The Problem

Input: A set of dominoes positioned on a board.



The Problem

Input: A set of dominoes positioned on a board.

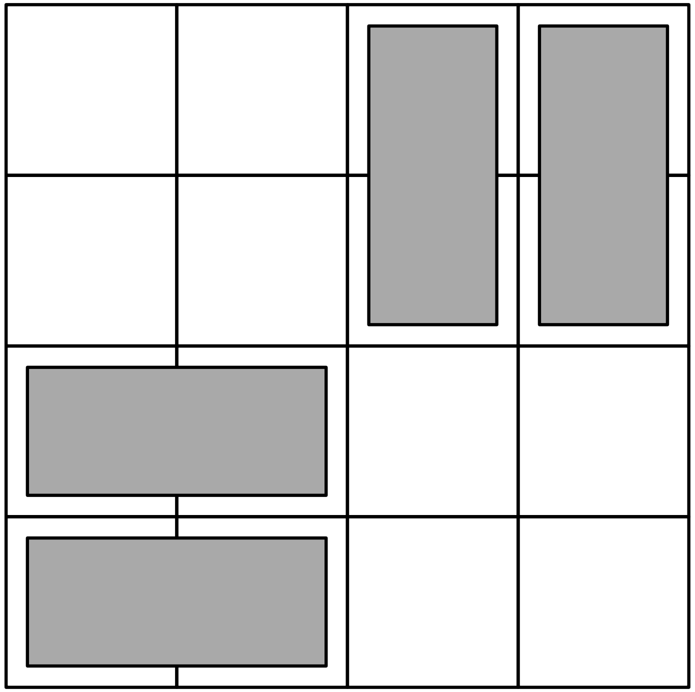


The Problem

Input: A set of dominoes positioned on a board.

Output: A numbering of domino endpoints such that...

- Adjacent endpoints have the same number.

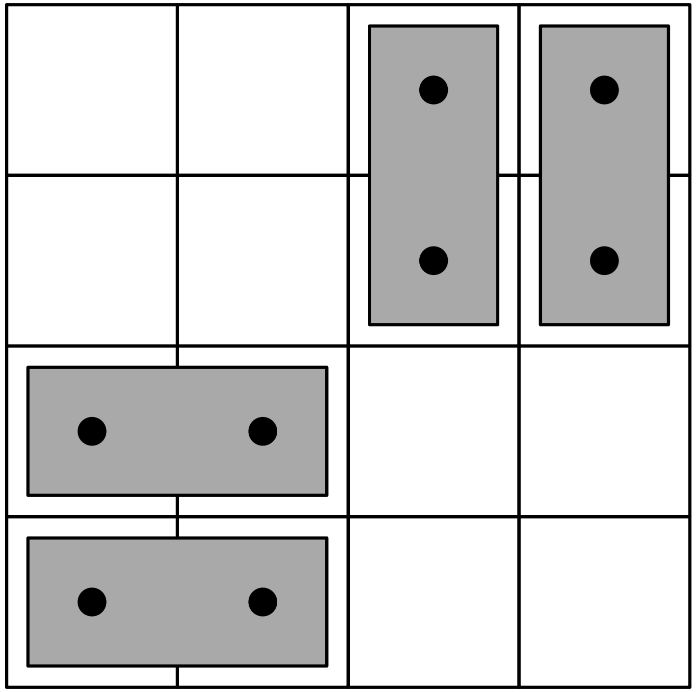


The Problem

Input: A set of dominoes positioned on a board.

Output: A numbering of domino endpoints such that...

- Adjacent endpoints have the same number.

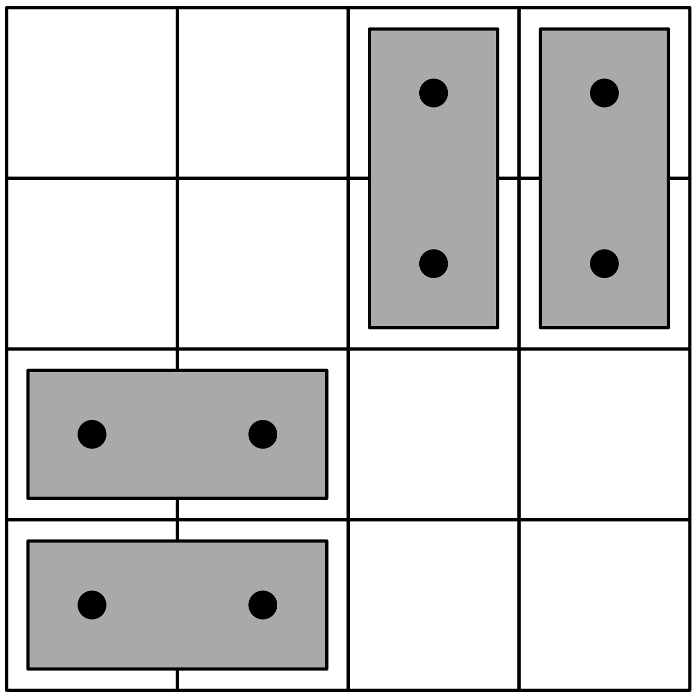


The Problem

Input: A set of dominoes positioned on a board.

Output: A numbering of domino endpoints such that...

- Adjacent endpoints have the same number.
- Every number is used at most twice.
- Endpoints belonging to the same domino have differing numbers.

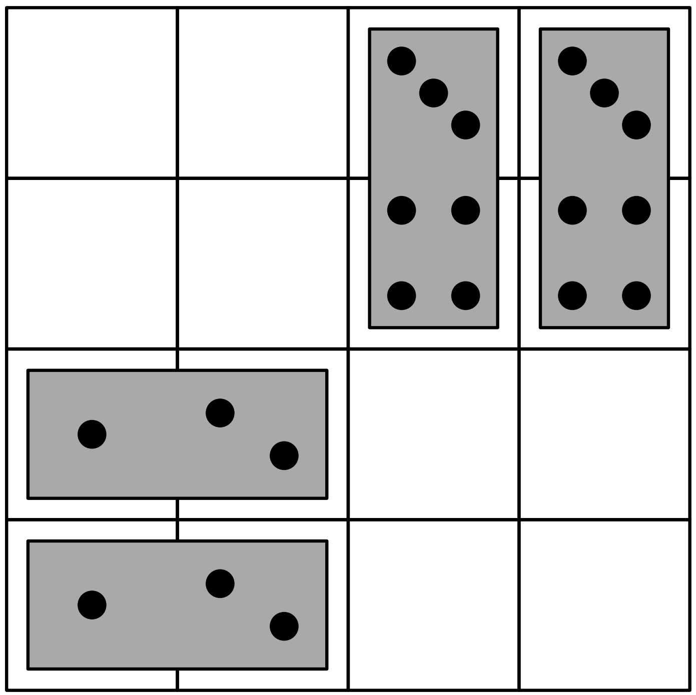


The Problem

Input: A set of dominoes positioned on a board.

Output: A numbering of domino endpoints such that...

- Adjacent endpoints have the same number.
- Every number is used at most twice.
- Endpoints belonging to the same domino have differing numbers.

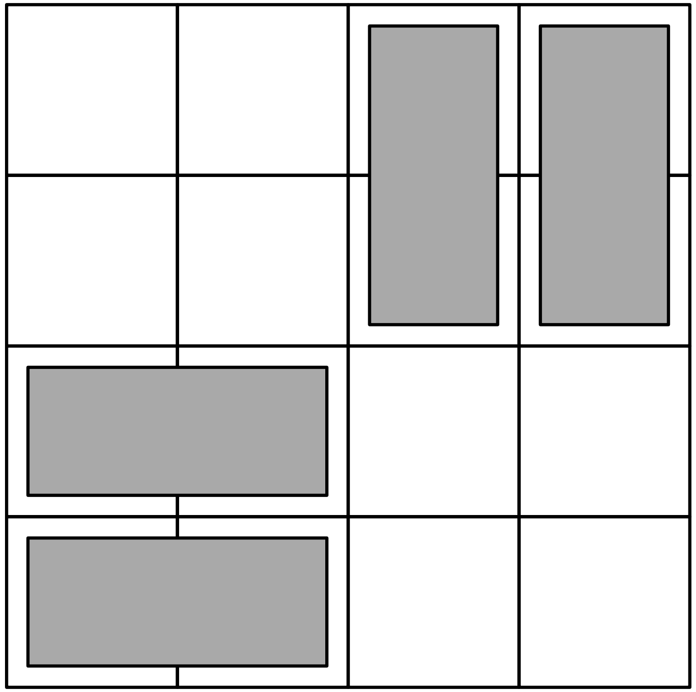


Modeling the Problem

- Can we solve this problem graph theoretically?

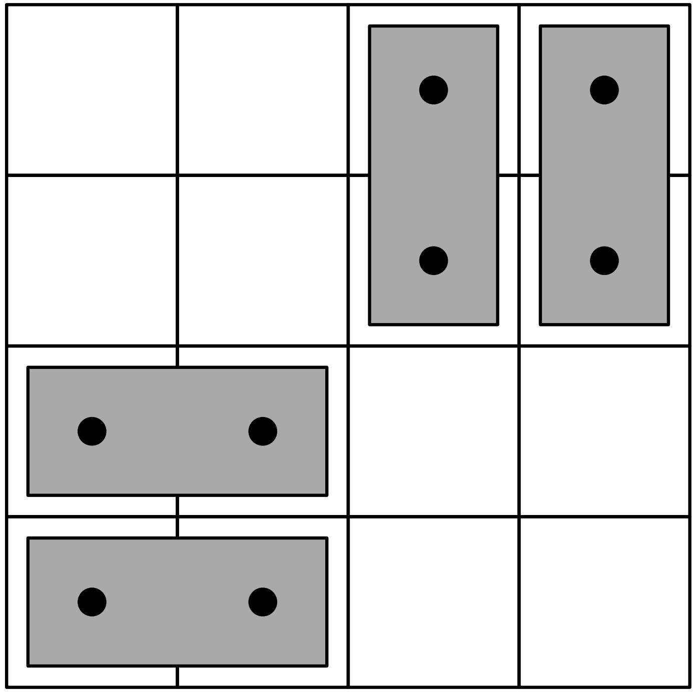
Modeling the Problem

- Can we solve this problem graph theoretically?
- In the *domino graph* $D = (V, E)$...



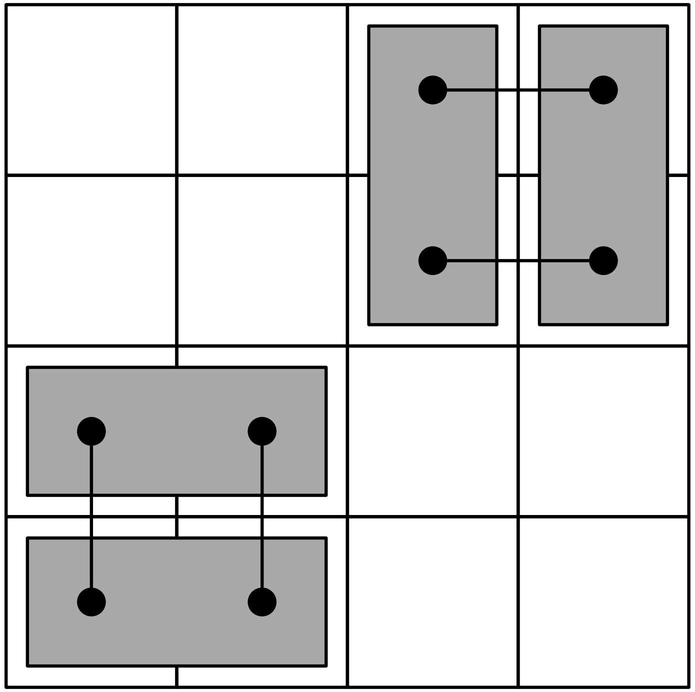
Modeling the Problem

- Can we solve this problem graph theoretically?
- In the *domino graph* $D = (V, E)$...
  - there is a node in V for each domino endpoint.



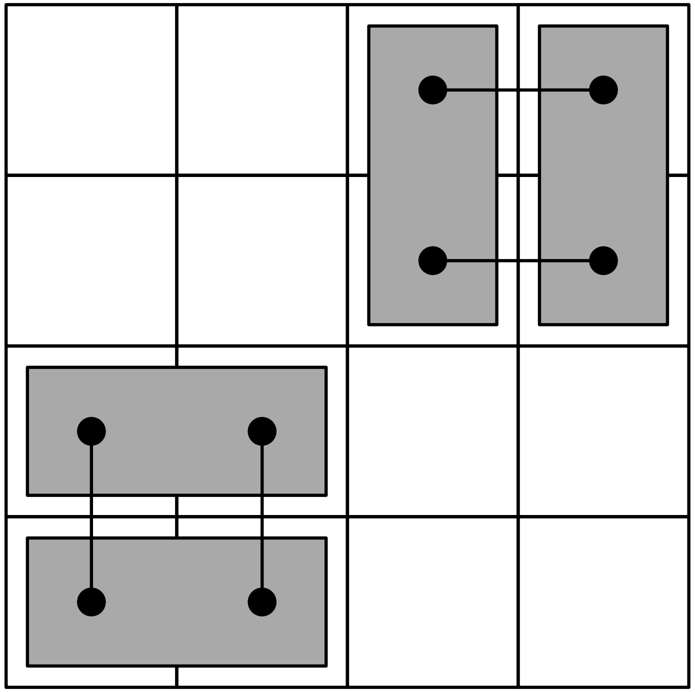
Modeling the Problem

- Can we solve this problem graph theoretically?
- In the *domino graph* $D = (V, E)$...
  - there is a node in V for each domino endpoint.
  - $uv \in E$ iff u is adjacent to v and uv is not on the domino



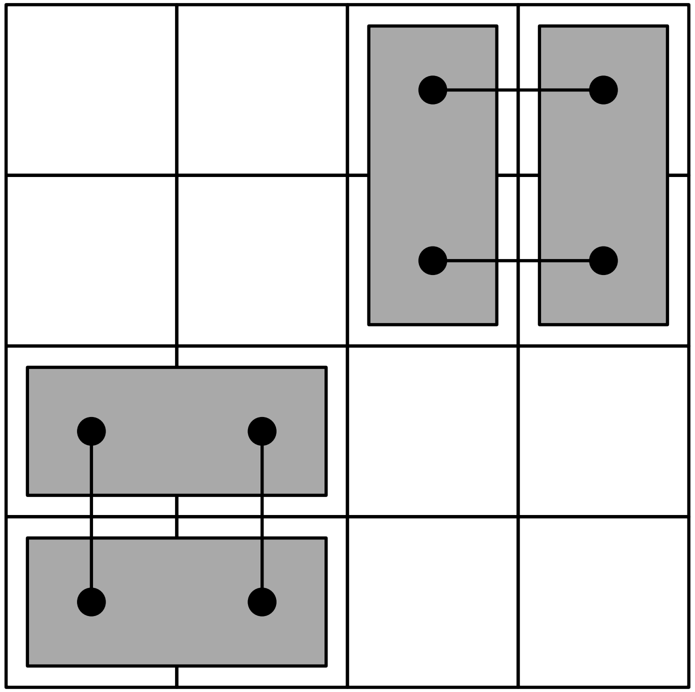
Modeling the Problem — II

- What does a solution to our numbering problem look like in D?



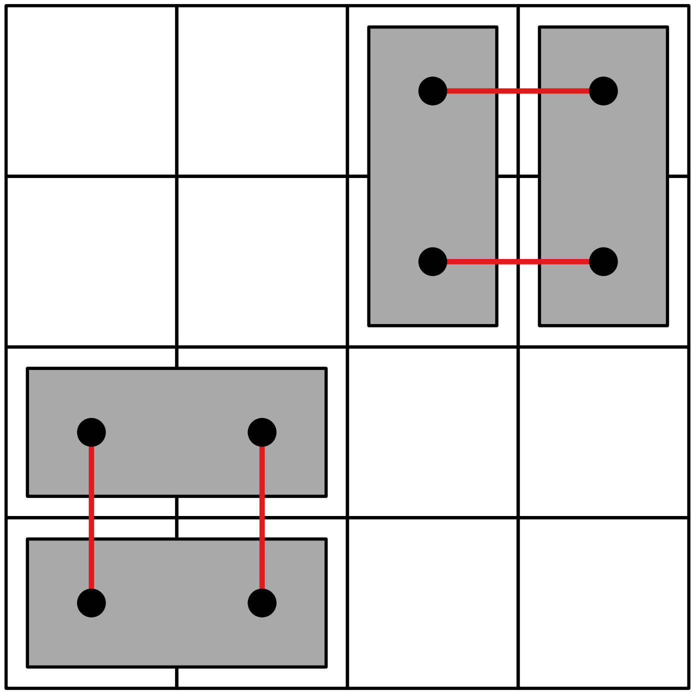
Modeling the Problem — II

- What does a solution to our numbering problem look like in D?
- If a solution exists, then there is a *perfect matching* in the domino graph.



Modeling the Problem — II

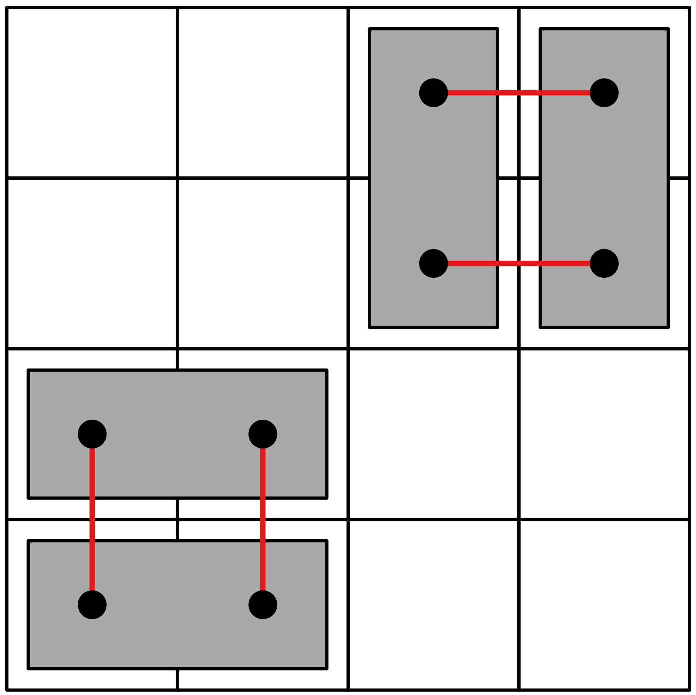
- What does a solution to our numbering problem look like in D?
- If a solution exists, then there is a *perfect matching* in the domino graph.



Modeling the Problem — II

- What does a solution to our numbering problem look like in D?
- If a solution exists, then there is a *perfect matching* in the domino graph.

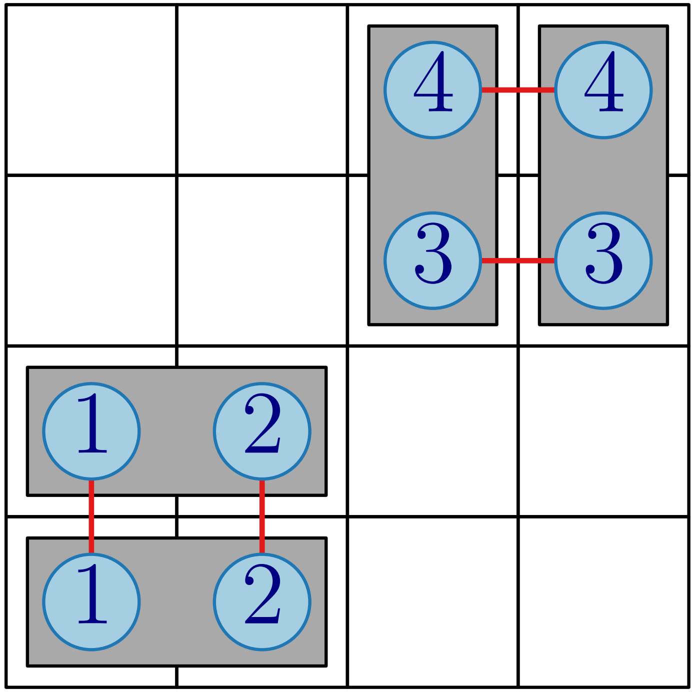
Algorithm:
Calculate a perfect matching and give nodes in the same matching edge the same number.



Modeling the Problem — II

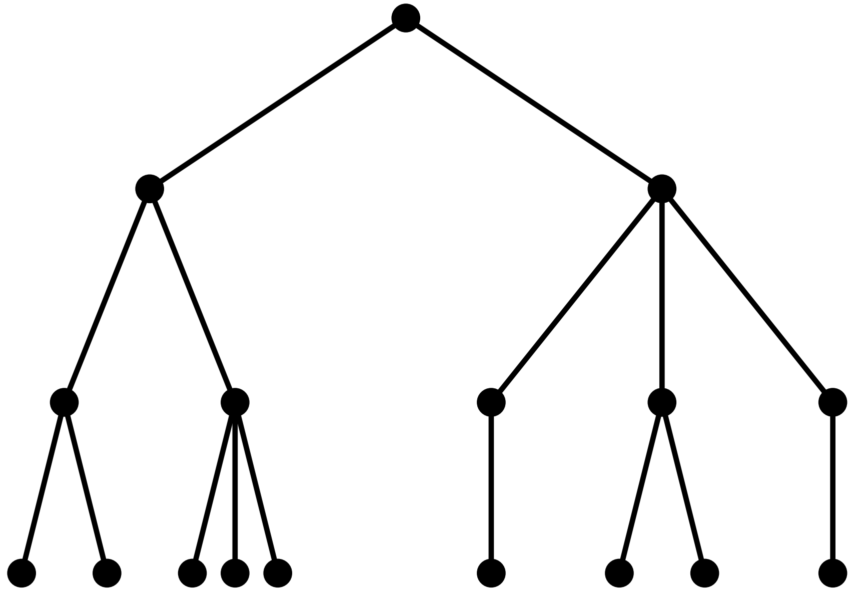
- What does a solution to our numbering problem look like in D?
- If a solution exists, then there is a *perfect matching* in the domino graph.

Algorithm:
Calculate a perfect matching and give nodes in the same matching edge the same number.



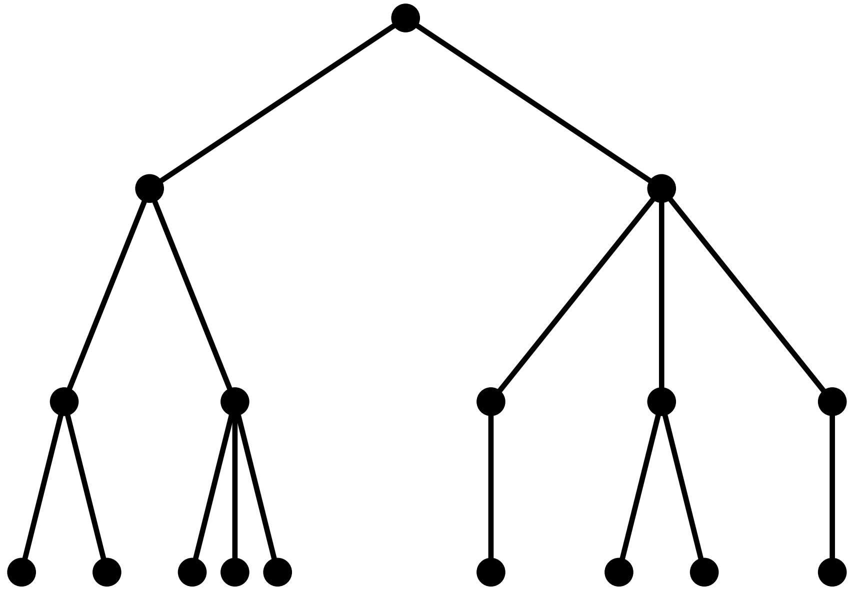
Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...



Maximum Matchings in Forests

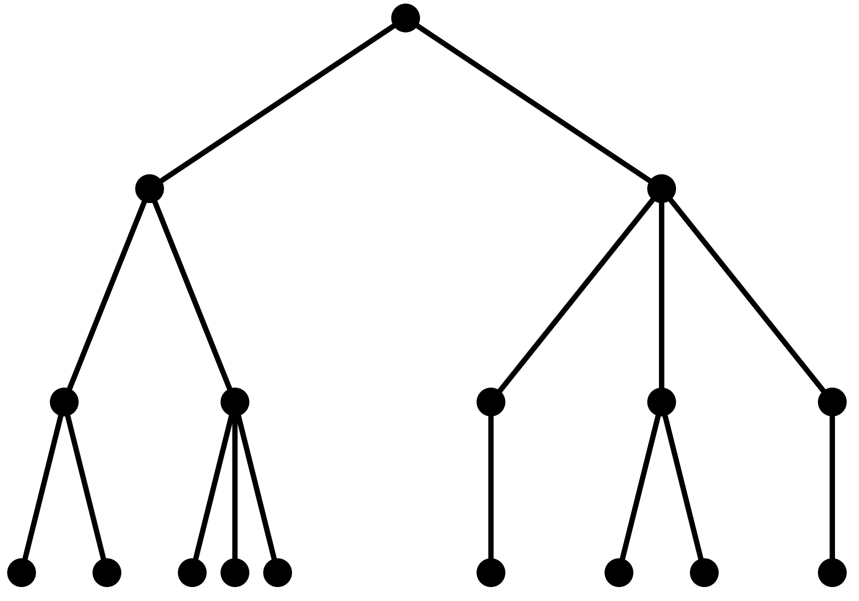
- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.



Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

  1. Find all leaves L.



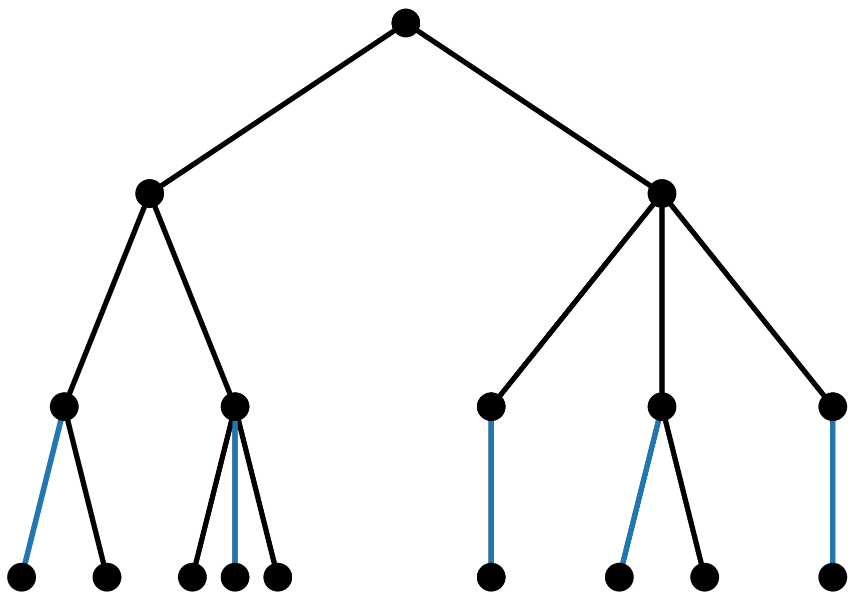
Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

1. Find all leaves L.
2. For each edge uv with $v \in L$...

   $M = M \cup \{uv\}$.

   Delete u and v.



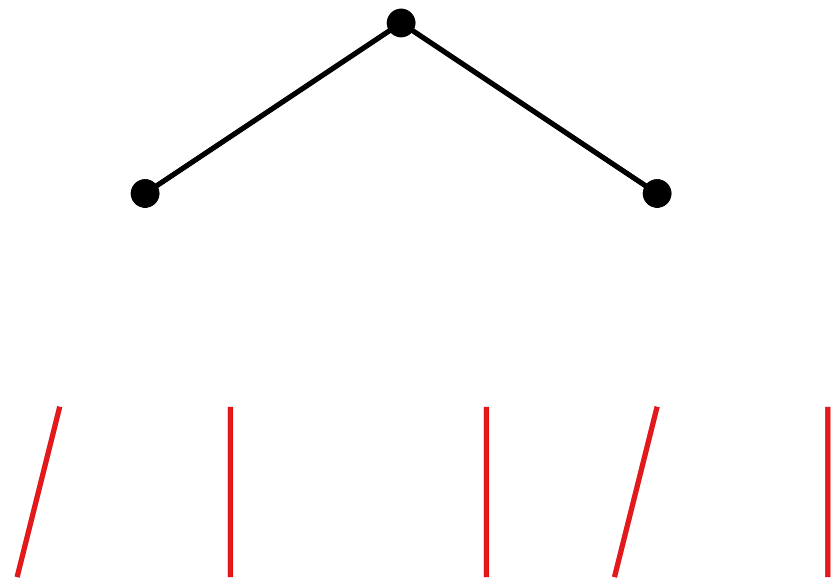
Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

1. Find all leaves L.
2. For each edge uv with $v \in L$...

   $M = M \cup \{uv\}$.

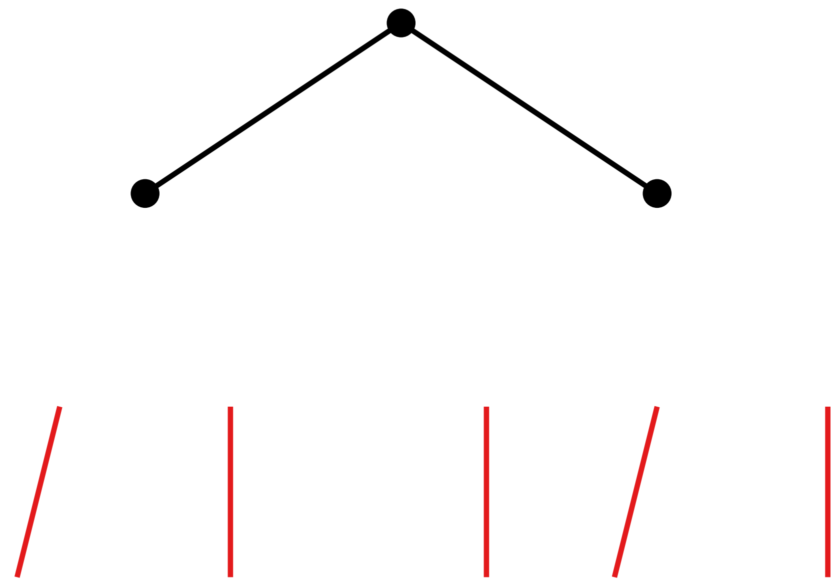
   Delete u and v.



Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

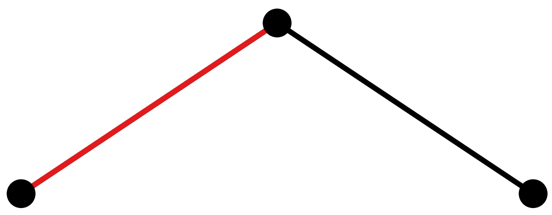
1. Find all leaves L.
2. For each edge uv with $v \in L$...
   $M = M \cup \{uv\}$.
   Delete u and v.
3. If there are remaining leaves, go to step 1.

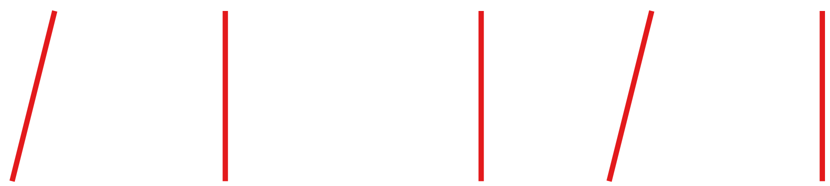


Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

1. Find all leaves L.
2. For each edge uv with $v \in L$...
   $M = M \cup \{uv\}$.
   Delete u and v.
3. If there are remaining leaves, go to step 1.

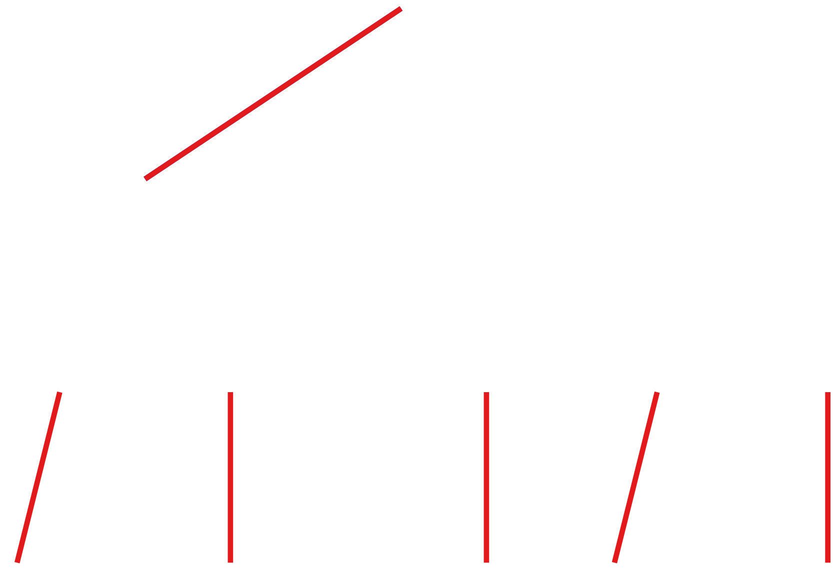




Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

1. Find all leaves L.
2. For each edge uv with $v \in L$...
   $M = M \cup \{uv\}$.
   Delete u and v.
3. If there are remaining leaves, go to step 1.



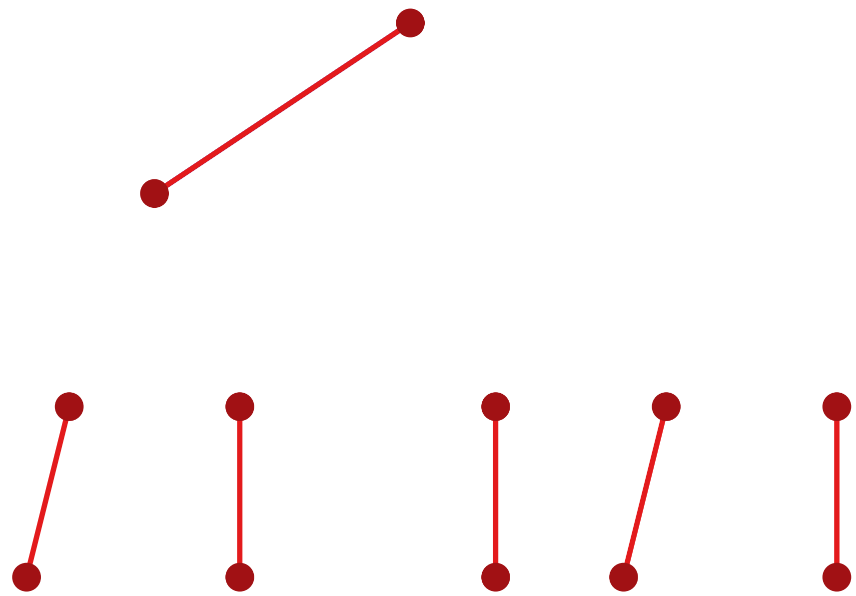
Maximum Matchings in Forests

- Suppose a domino graph is a *tree* (or forest)...
- Then we could calculate a maximum matching M in $\mathcal{O}(V)$ time.

1. Find all leaves L.
2. For each edge uv with $v \in L$...

   $M = M \cup \{uv\}$.

   Delete u and v.
3. If there are remaining leaves, go to step 1.

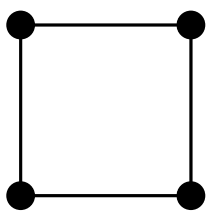
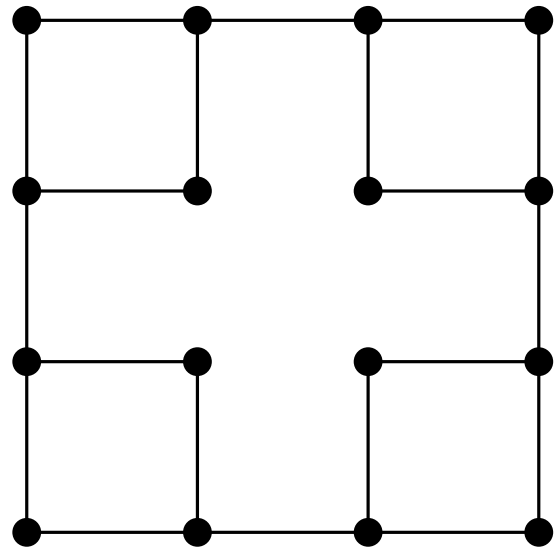


What do domino graphs look like?

But are all domino graphs trees (or forests)?

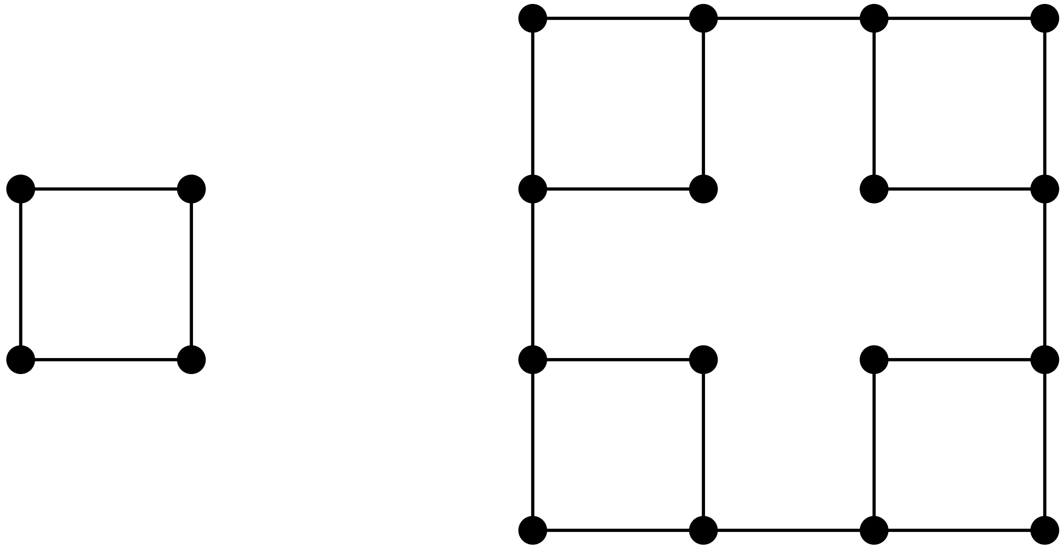
What do domino graphs look like?

But are all domino graphs trees (or forests)?

What do domino graphs look like?

But are all domino graphs trees (or forests)?



Domino graphs can have cycles! $\Rightarrow$ They are not trees. $\Rightarrow$ *Our* $\mathcal{O}(V)$ *algorithm will not work here*.

What do domino graphs look like? — II

Question: What is the maximum degree Δ in the domino graph?

What do domino graphs look like? — II

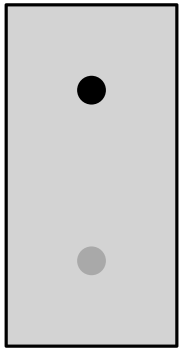
Question: What is the maximum degree Δ in the domino graph?

Let us find out using an example...

What do domino graphs look like? — II

Question: What is the maximum degree Δ in the domino graph?

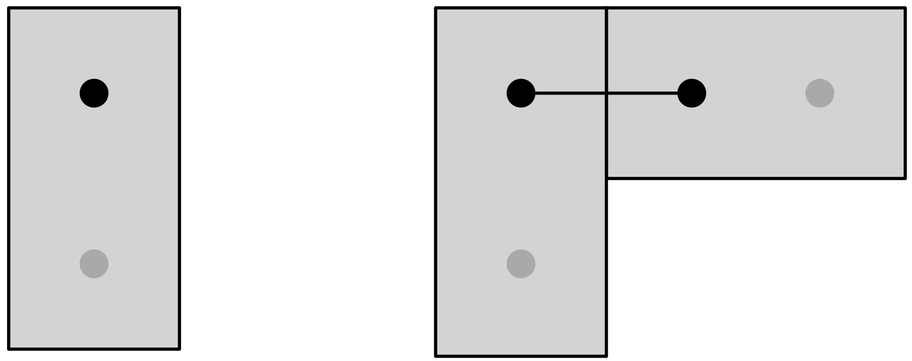
Let us find out using an example...



What do domino graphs look like? — II

Question: What is the maximum degree Δ in the domino graph?

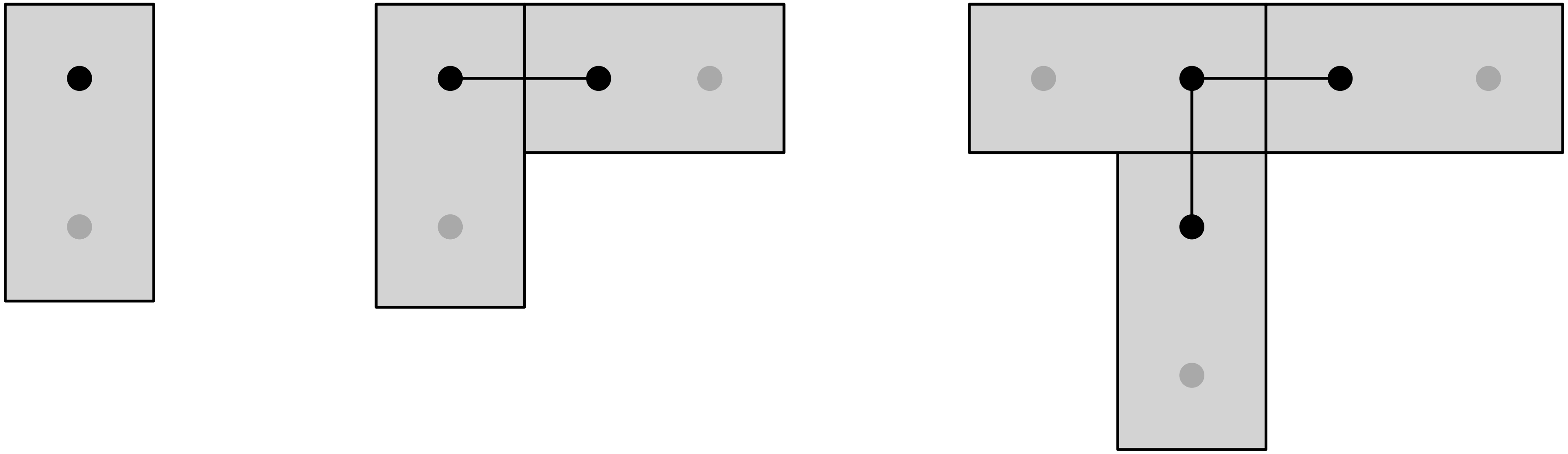
Let us find out using an example...



What do domino graphs look like? — II

Question: What is the maximum degree Δ in the domino graph?

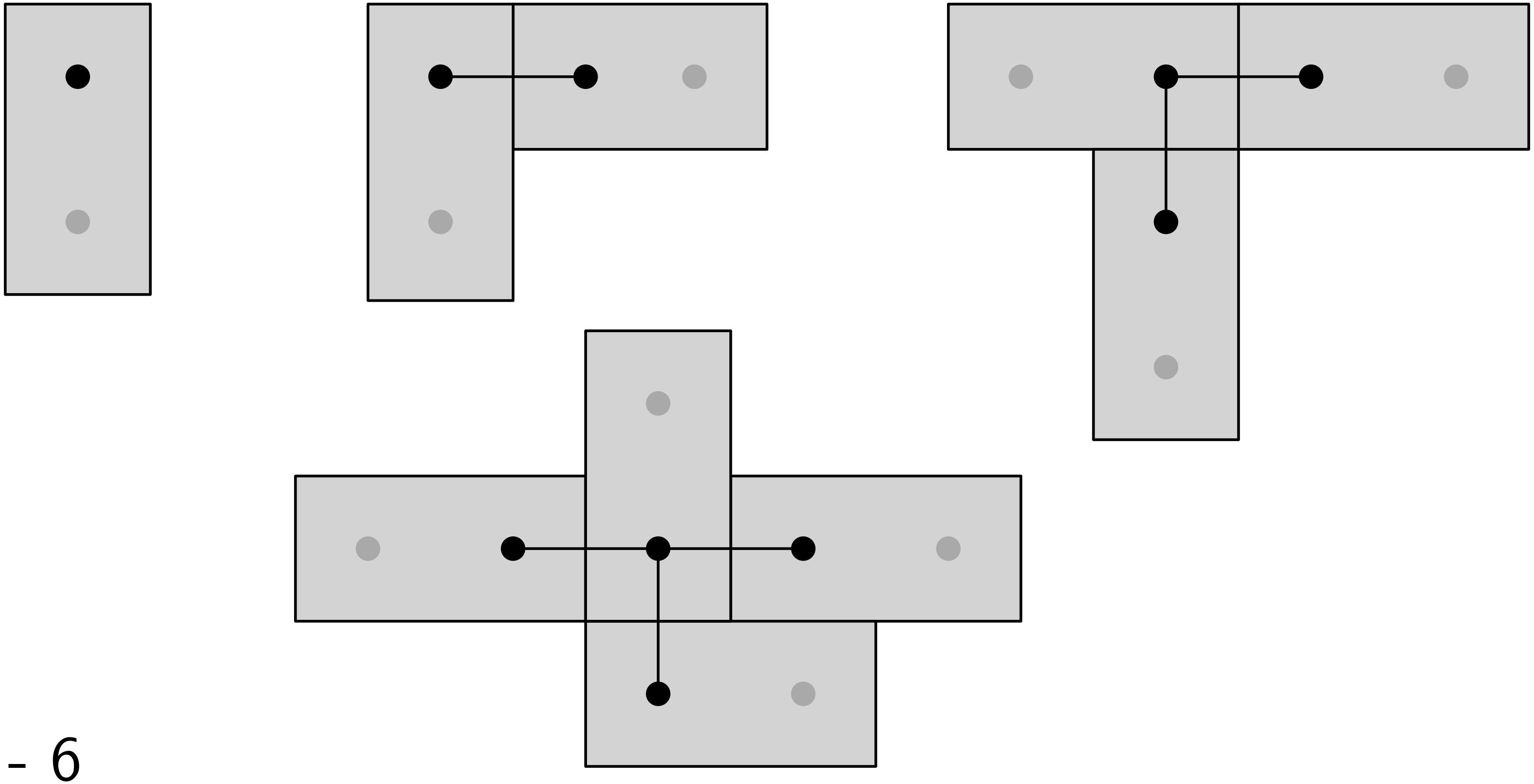
Let us find out using an example...



What do domino graphs look like? — II

Question: What is the maximum degree Δ in the domino graph?

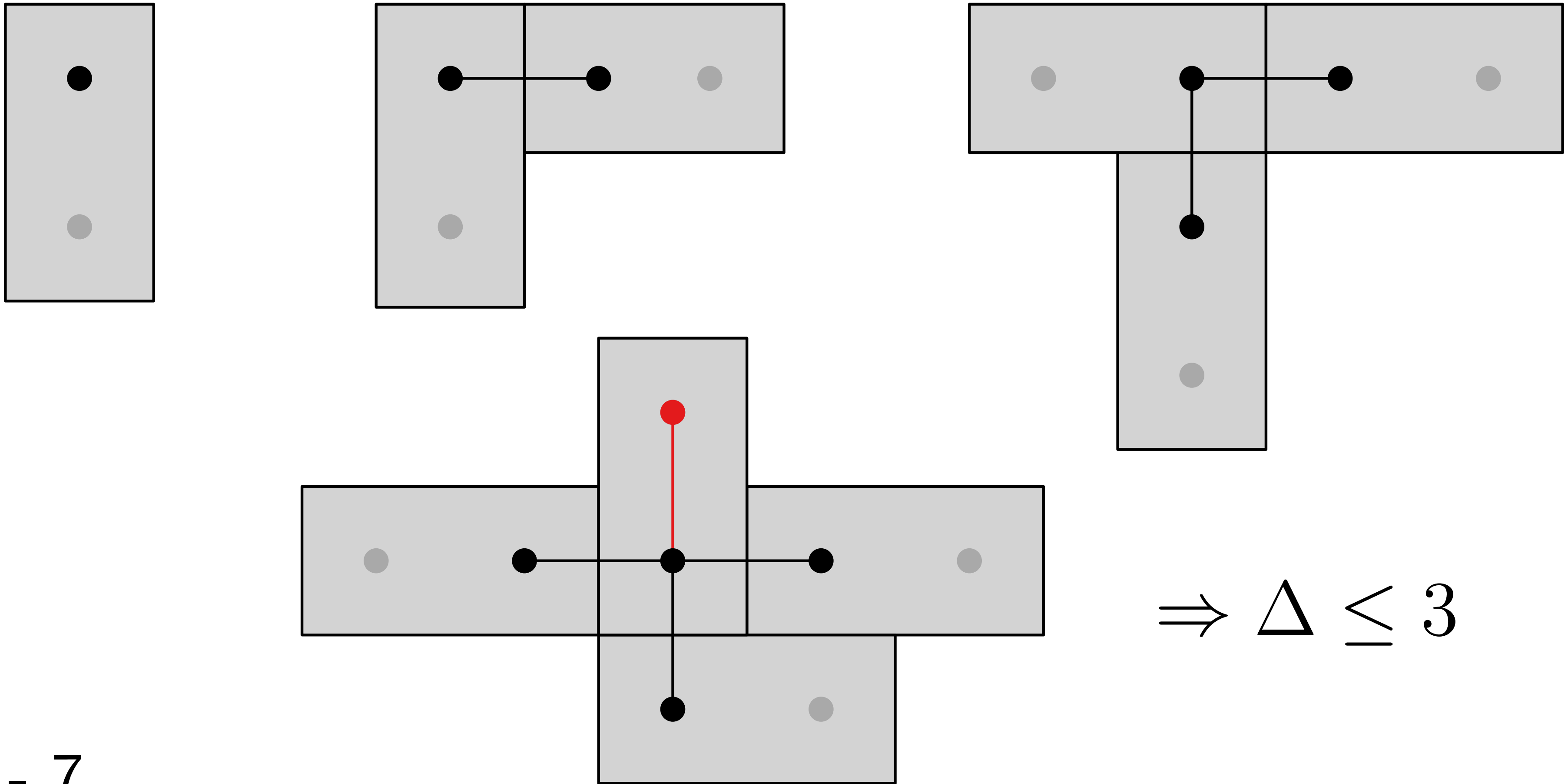
Let us find out using an example...



What do domino graphs look like? — II

Question: What is the maximum degree Δ in the domino graph?

Let us find out using an example...



$\Rightarrow \Delta \leq 3$

Max. Matchings in General Graphs

- How do we compute maximum (cardinality) matchings in general graphs?

Max. Matchings in General Graphs

- How do we compute maximum (cardinality) matchings in general graphs?
- *Algorithmic Graph Theory*:

Max. Matchings in General Graphs

- How do we compute maximum (cardinality) matchings in general graphs?
- *Algorithmic Graph Theory*:
    - Edmonds' 1965 Algorithm – $\mathcal{O}(V^3)$, too slow and too complicated!

Max. Matchings in General Graphs

- How do we compute maximum (cardinality) matchings in general graphs?
- *Algorithmic Graph Theory*:
  - Edmonds' 1965 Algorithm – $\mathcal{O}(V^3)$, too slow and too complicated!
  - Micali-Vazirani Algorithm – $\mathcal{O}(\sqrt{V}E)$, *way* too complicated!

Max. Matchings in General Graphs

- How do we compute maximum (cardinality) matchings in general graphs?
- *Algorithmic Graph Theory*:
  - Edmonds' 1965 Algorithm – $\mathcal{O}(V^3)$, too slow and too complicated!
  - Micali-Vazirani Algorithm – $\mathcal{O}(\sqrt{V}E)$, *way* too complicated!
- We know that in our domino graphs $\Delta \leq 3$. Can we specialise them further?

Max. Matchings in General Graphs

- How do we compute maximum (cardinality) matchings in general graphs?
- *Algorithmic Graph Theory*:
  - Edmonds' 1965 Algorithm – $\mathcal{O}(V^3)$, too slow and too complicated!
  - Micali-Vazirani Algorithm – $\mathcal{O}(\sqrt{V}E)$, *way* too complicated!
- We know that in our domino graphs $\Delta \leq 3$. Can we specialise them further?
- Hopefully, such a specialisation will give us faster and/or simpler algorithms!

Domino Graph is Bipartite

Theorem. Any domino graph $D = (V, E)$ is *bipartite*.

Domino Graph is Bipartite

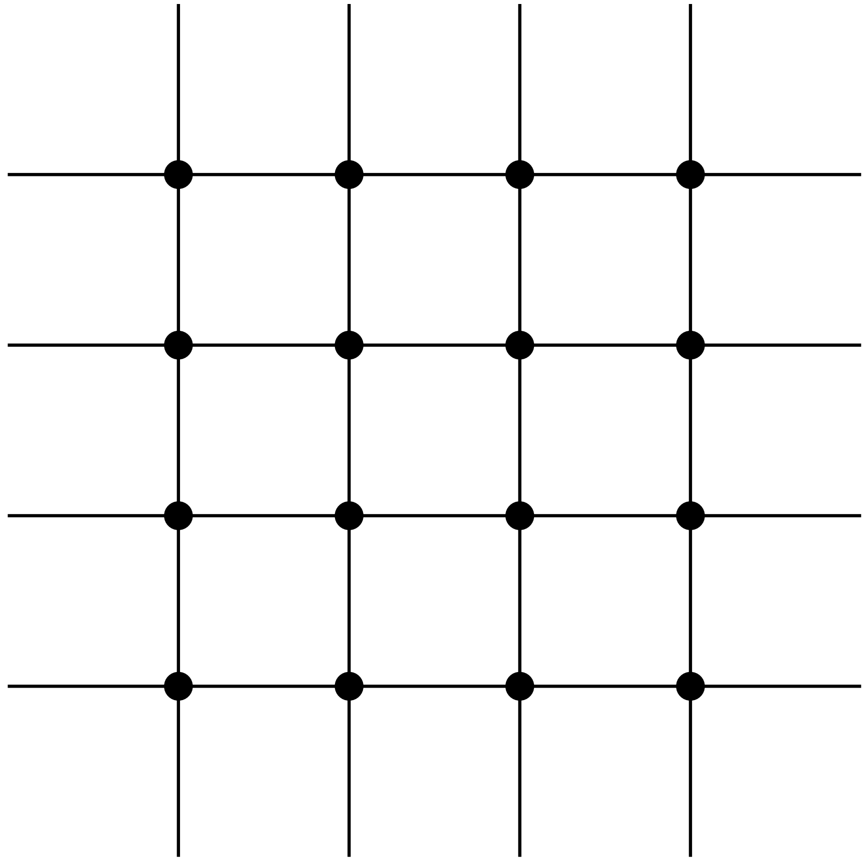
Theorem. Any domino graph $D = (V, E)$ is *bipartite*.

Proof. Domino graphs are subgraphs of the *infinite grid graph*.

Domino Graph is Bipartite

Theorem. Any domino graph $D = (V, E)$ is *bipartite*.

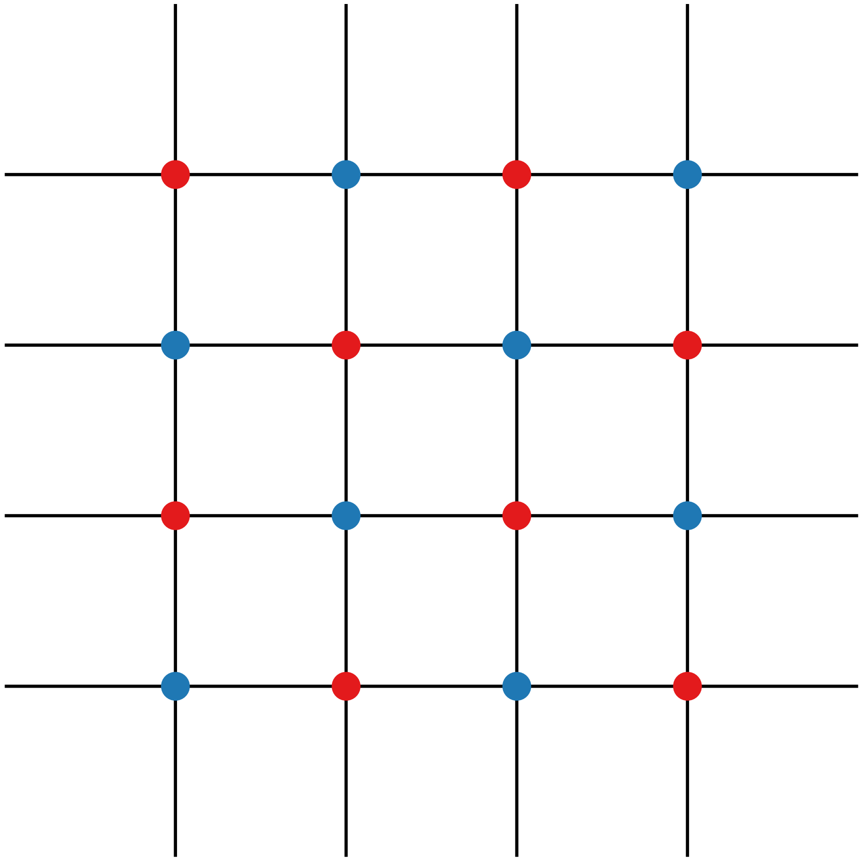
Proof. Domino graphs are subgraphs of the *infinite grid graph*.



Domino Graph is Bipartite

Theorem. Any domino graph $D = (V, E)$ is *bipartite*.

Proof. Domino graphs are subgraphs of the *infinite grid graph*.

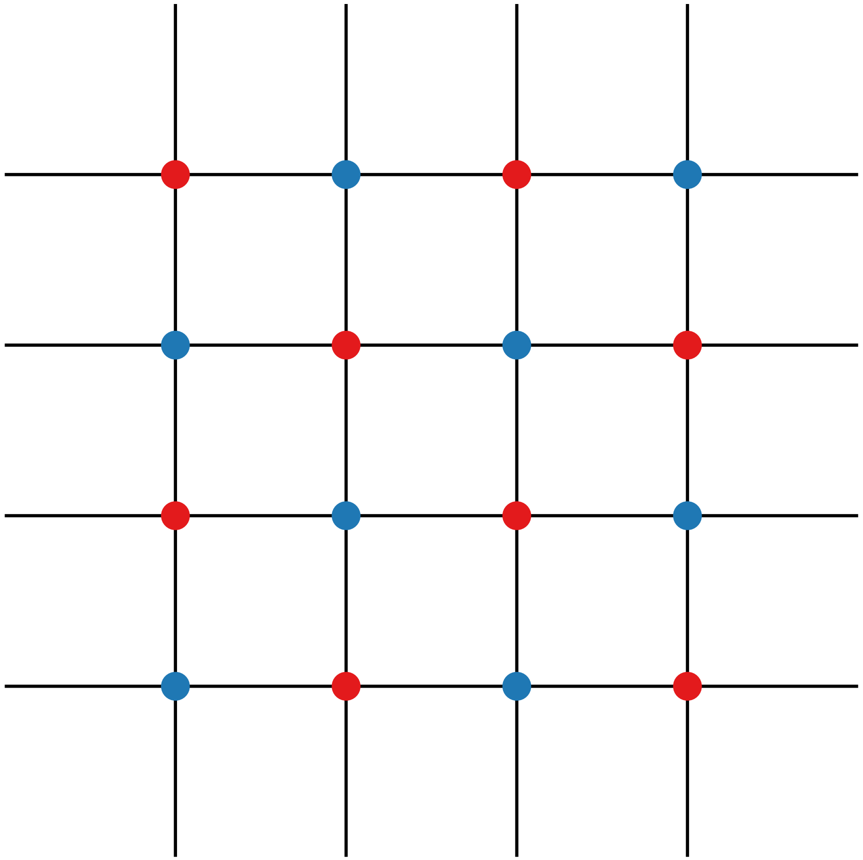


Domino Graph is Bipartite

Theorem. Any domino graph $D = (V, E)$ is *bipartite*.

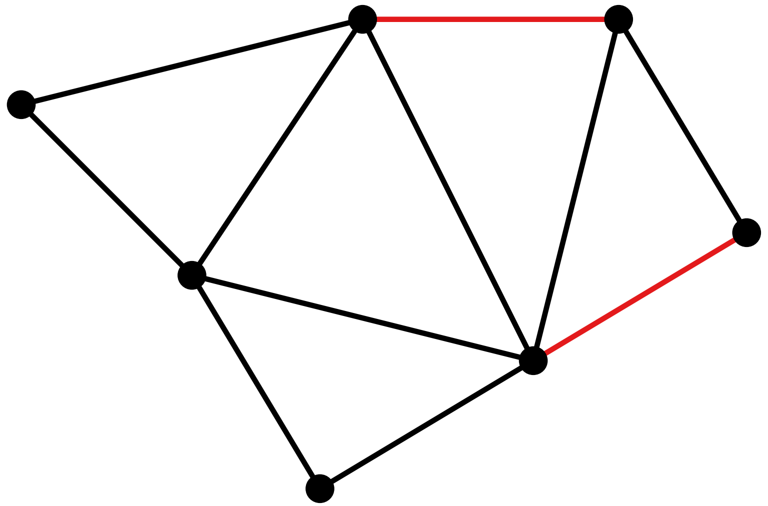
Proof. Domino graphs are subgraphs of the *infinite grid graph*.

The infinite grid graph can be two-coloured. Thus, we can divide V into two edge-disjoint sets A and B.



Berge's Theorem on Maximum Matchings

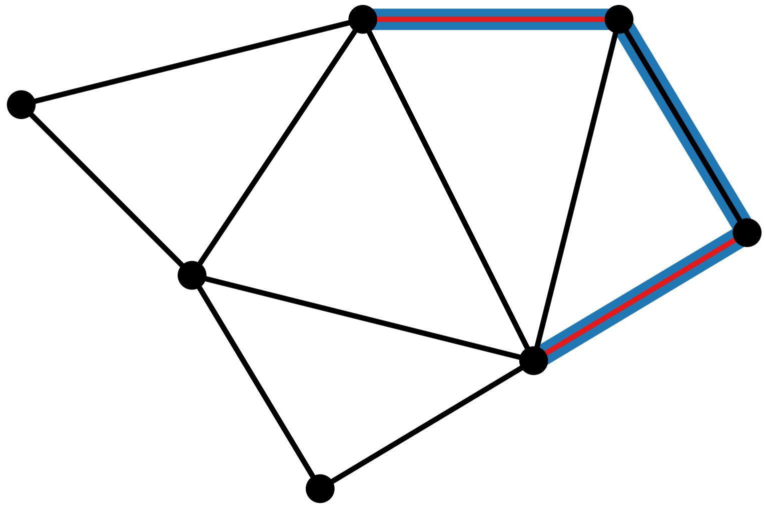
Let M be a (not necessarily maximal) matching...



Berge's Theorem on Maximum Matchings

Let M be a (not necessarily maximal) matching...

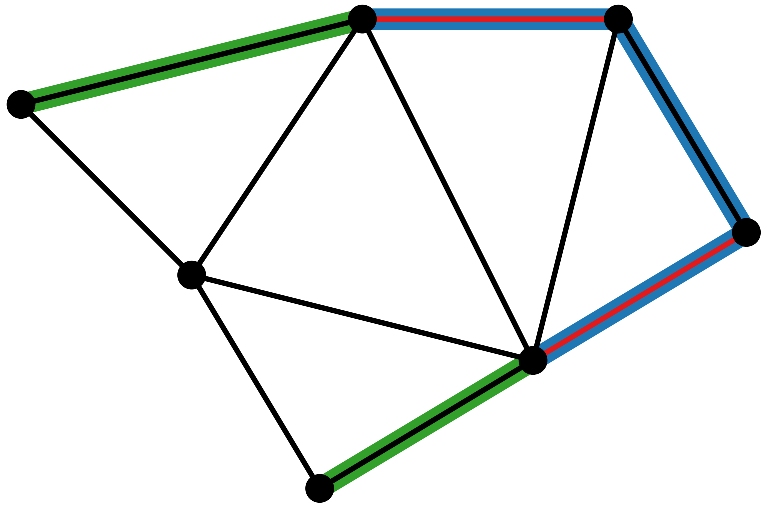
An *alternating path* switches between matching and non-matching edges.



Berge's Theorem on Maximum Matchings

Let M be a (not necessarily maximal) matching...

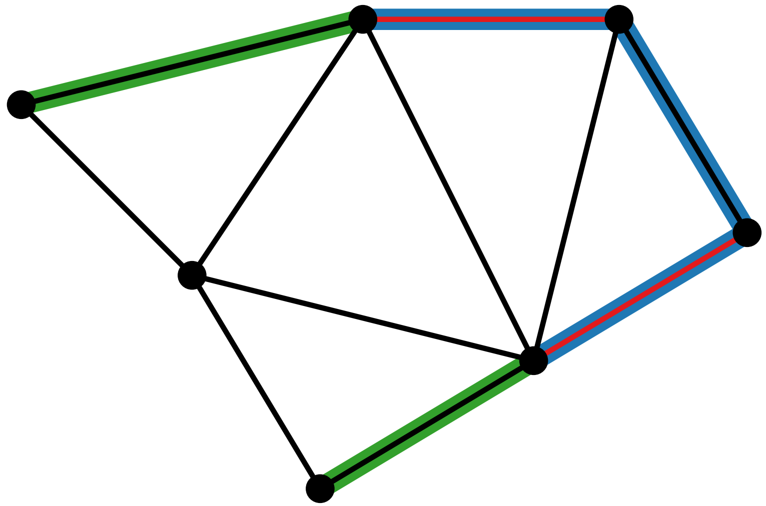
An *alternating path* switches between matching and non-matching edges.



An *augmenting path* is an *alternating path* that starts and ends in an M-free node.

Berge's Theorem on Maximum Matchings

Let M be a (not necessarily maximal) matching...



An *alternating path* switches between matching and non-matching edges.

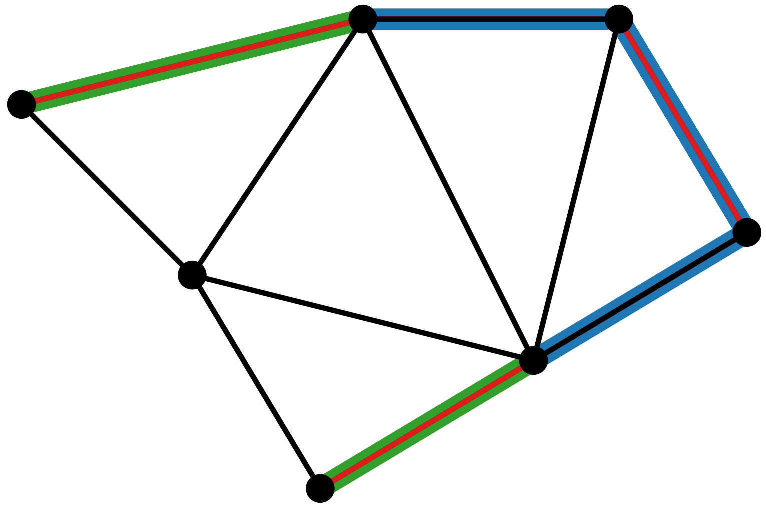
An *augmenting path* is an *alternating path* that starts and ends in an M-free node.

By switching the parity of the matching along an augmenting path, we can extend the matching by 1 edge.

Berge's Theorem on Maximum Matchings

Let M be a (not necessarily maximal) matching...

An *alternating path* switches between matching and non-matching edges.



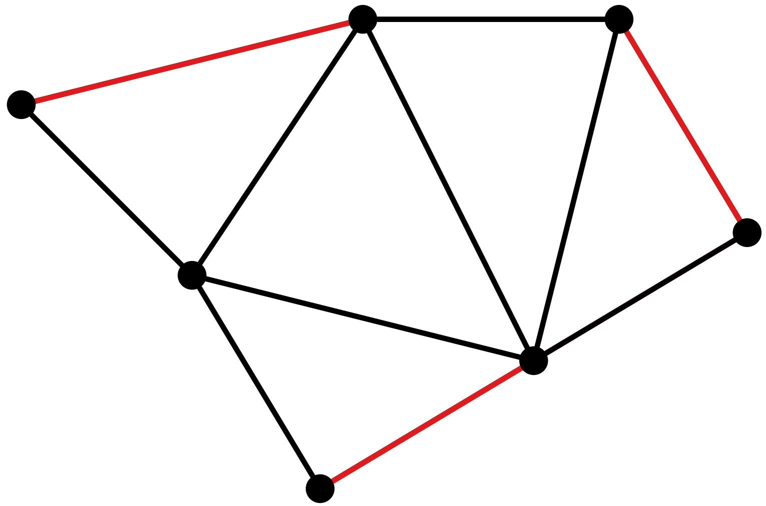
An *augmenting path* is an *alternating path* that starts and ends in an M-free node.

By switching the parity of the matching along an augmenting path, we can extend the matching by 1 edge.

Berge's Theorem on Maximum Matchings

Let M be a (not necessarily maximal) matching...

An *alternating path* switches between matching and non-matching edges.



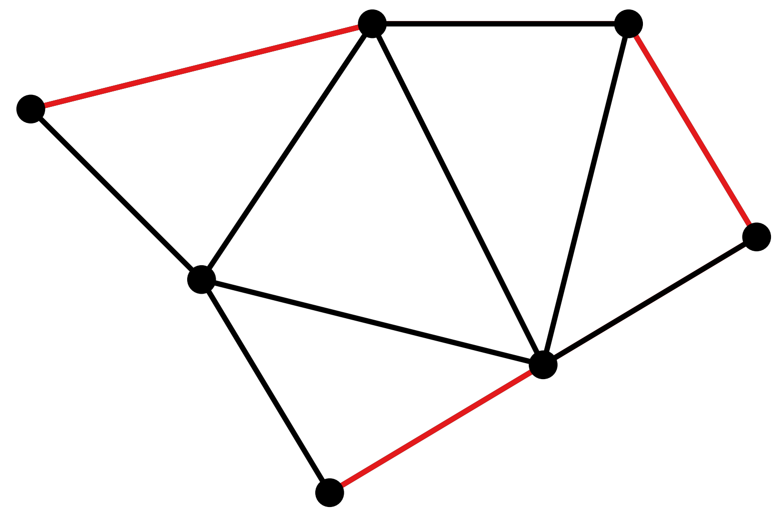
An *augmenting path* is an *alternating path* that starts and ends in an M-free node.

By switching the parity of the matching along an augmenting path, we can extend the matching by 1 edge.

Berge's Theorem on Maximum Matchings

Let M be a (not necessarily maximal) matching...

An *alternating path* switches between matching and non-matching edges.



An *augmenting path* is an *alternating path* that starts and ends in an M-free node.

By switching the parity of the matching along an augmenting path, we can extend the matching by 1 edge.

Theorem. *(Berge)*

$$M \text{ is maximum matching} \Leftrightarrow \nexists \text{Augmenting path}$$

Matching Algos using Berge's Theorem

- Berge's theorem immediately gives us an outline for a general maximum matching algorithm:

Matching Algos using Berge's Theorem

- Berge's theorem immediately gives us an outline for a general maximum matching algorithm:

MaxMatching($G = (V, E)$)
  $M = \emptyset$
  while $\exists$ Augmenting path P in G **do**
    Augment M along P
  return M

Matching Algos using Berge's Theorem

- Berge's theorem immediately gives us an outline for a general maximum matching algorithm:

$\textsc{MaxMatching}(G = (V, E))$
  $M = \emptyset$
  while $\exists$ Augmenting path P in G **do**
    Augment M along P
  return M

- Why can we not implement this algorithm "directly"?

Matching Algos using Berge's Theorem

- Berge's theorem immediately gives us an outline for a general maximum matching algorithm:

```
MaxMatching(G = (V, E))
    M = ∅
    while ∃ Augmenting path P in G do
        Augment M along P
    return M
```

- Why can we not implement this algorithm "directly"?
- There are *many* paths that could be augmenting!

Matching Algos using Berge's Theorem

- Berge's theorem immediately gives us an outline for a general maximum matching algorithm:

$\textsc{MaxMatching}(G = (V, E))$
    $M = \emptyset$
    while $\exists$ Augmenting path P in G **do**
        Augment M along P
    return M

- Why can we not implement this algorithm "directly"?
- There are *many* paths that could be augmenting!
- Solution: Specialise the algorithm for bipartite graphs.

Reduction to Maximum Flow

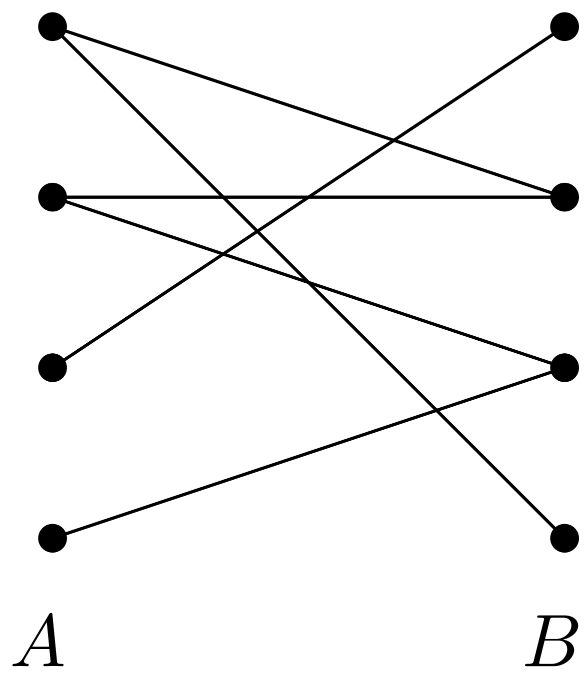
- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.

Reduction to Maximum Flow

- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.
- We can convert the problem of bipartite maximum matchings into a maximum flow problem...

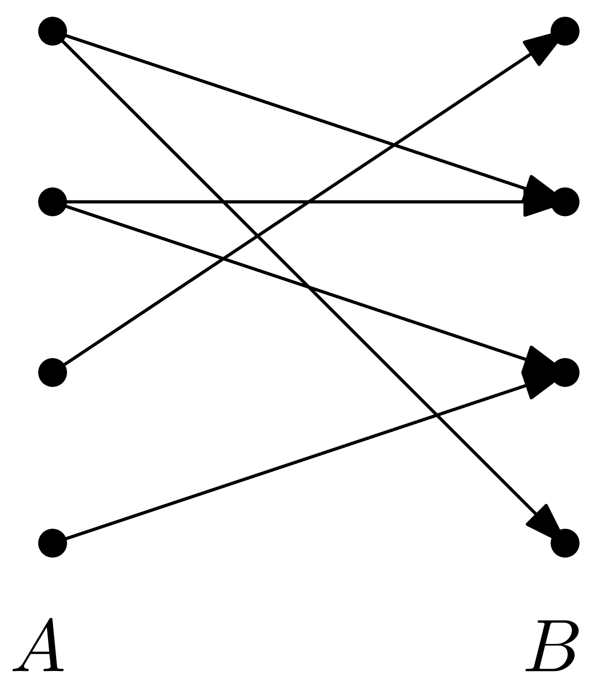
Reduction to Maximum Flow

- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.
- We can convert the problem of bipartite maximum matchings into a maximum flow problem...



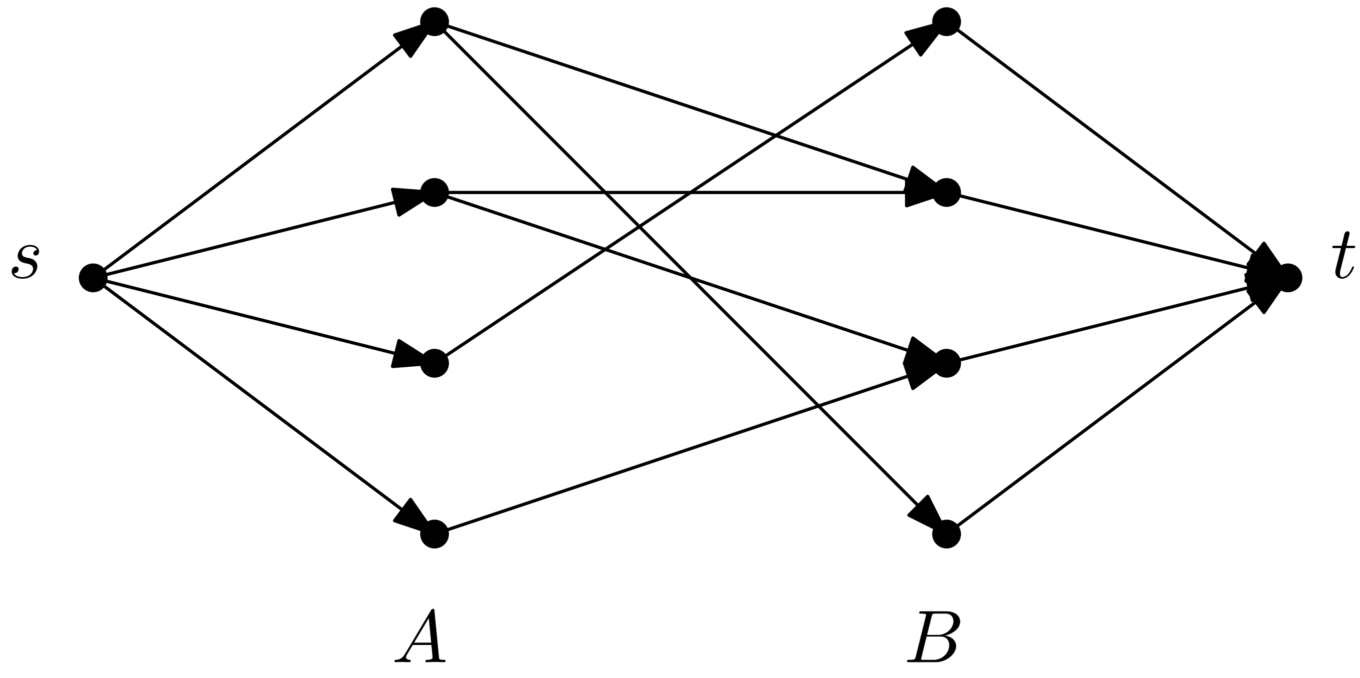
Reduction to Maximum Flow

- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.
- We can convert the problem of bipartite maximum matchings into a maximum flow problem...



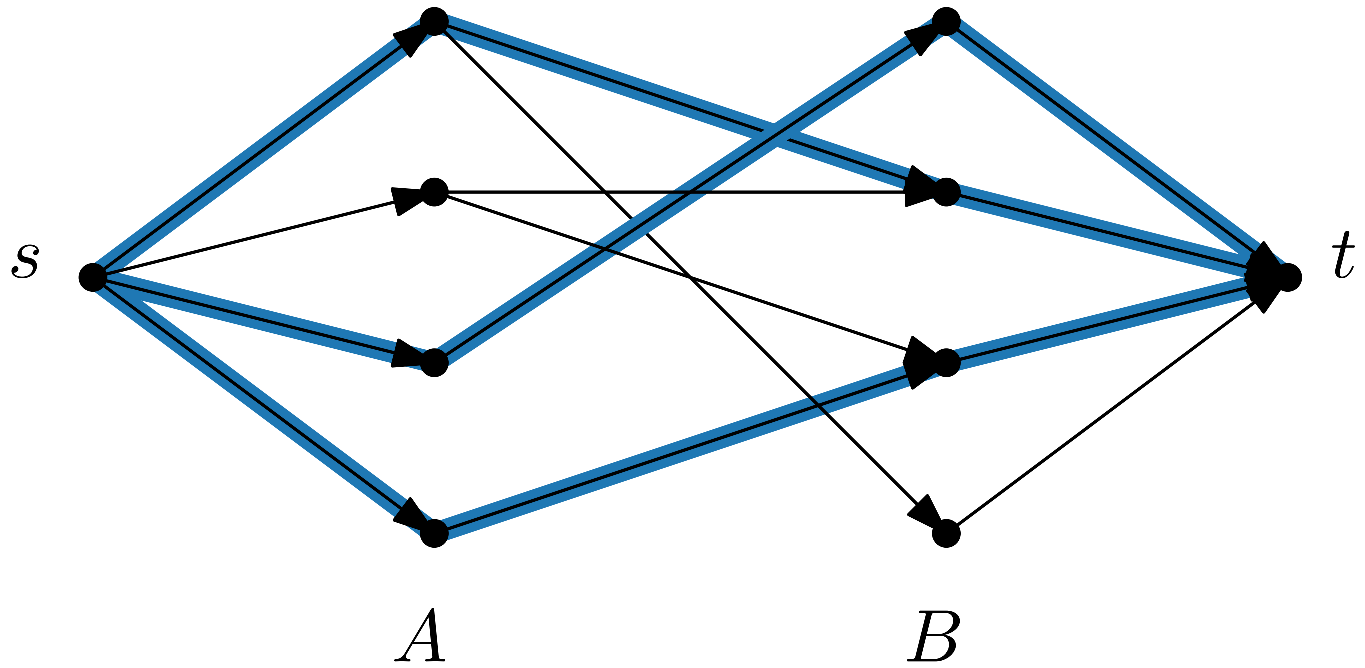
Reduction to Maximum Flow

- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.
- We can convert the problem of bipartite maximum matchings into a maximum flow problem...



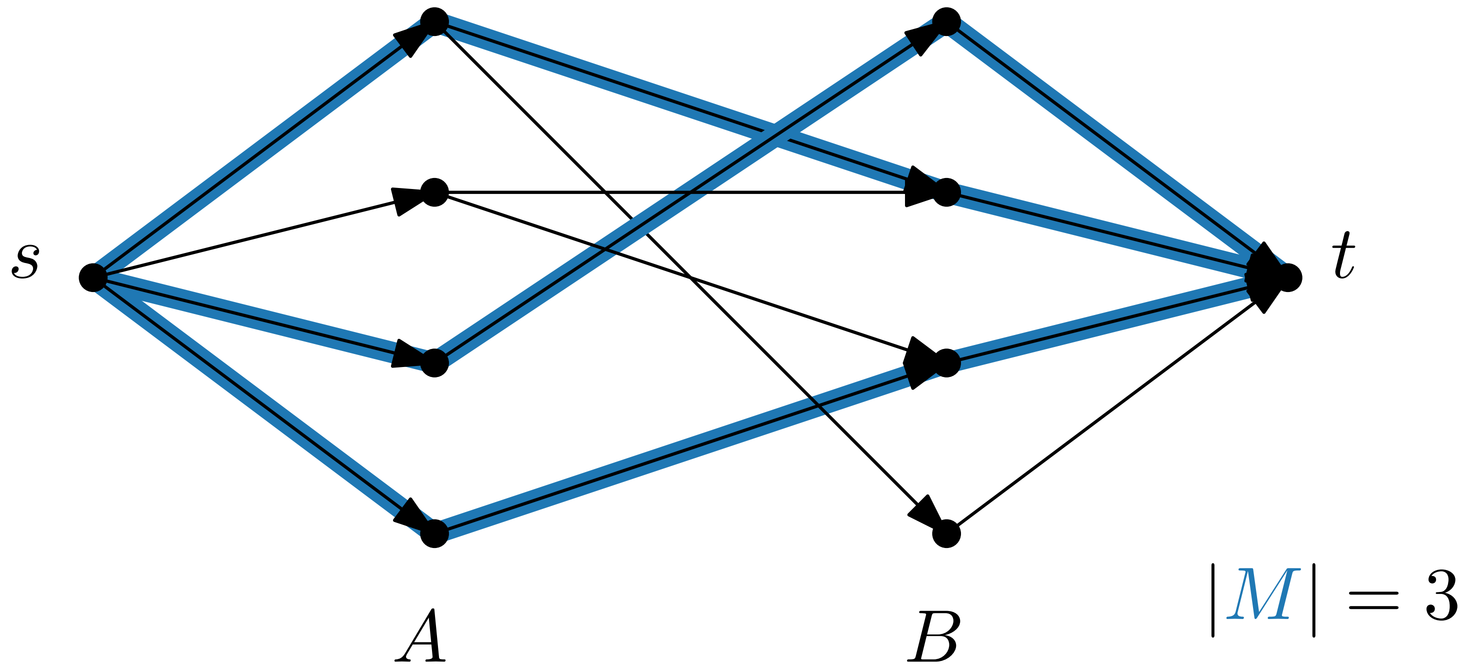
Reduction to Maximum Flow

- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.
- We can convert the problem of bipartite maximum matchings into a maximum flow problem...



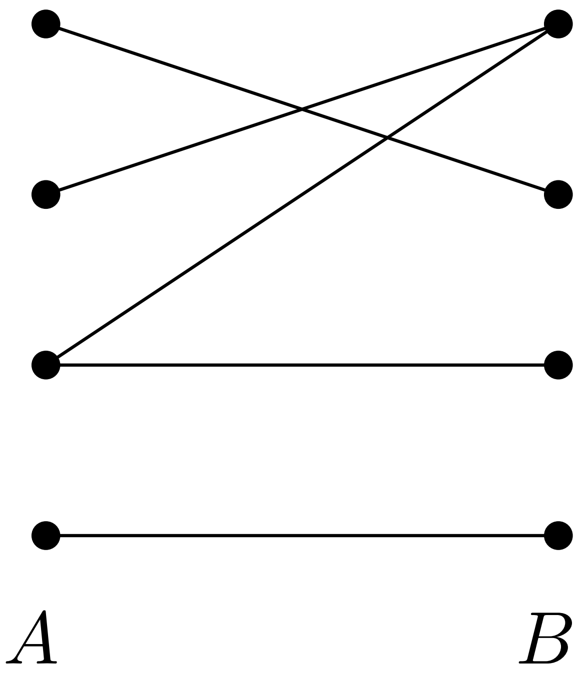
Reduction to Maximum Flow

- Recap from *Algorithmic Graph Theory*: Let $G = (A \cup B, E)$ be a bipartite graph.
- We can convert the problem of bipartite maximum matchings into a maximum flow problem...



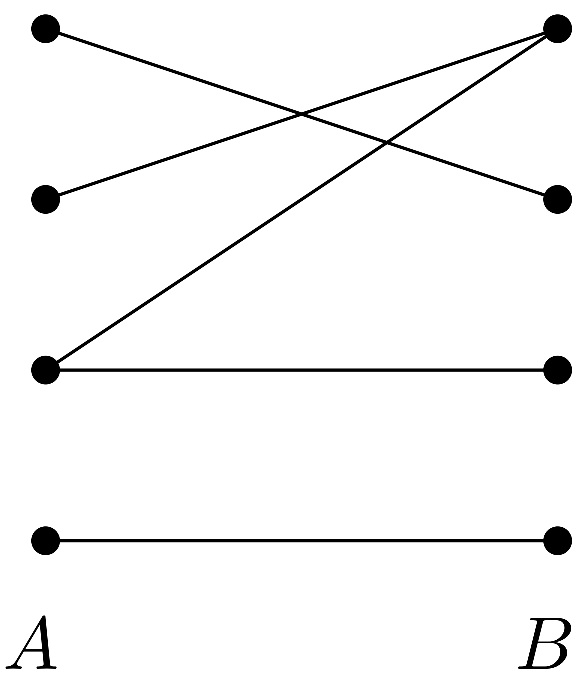
Specialising Maximum Flow

- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.



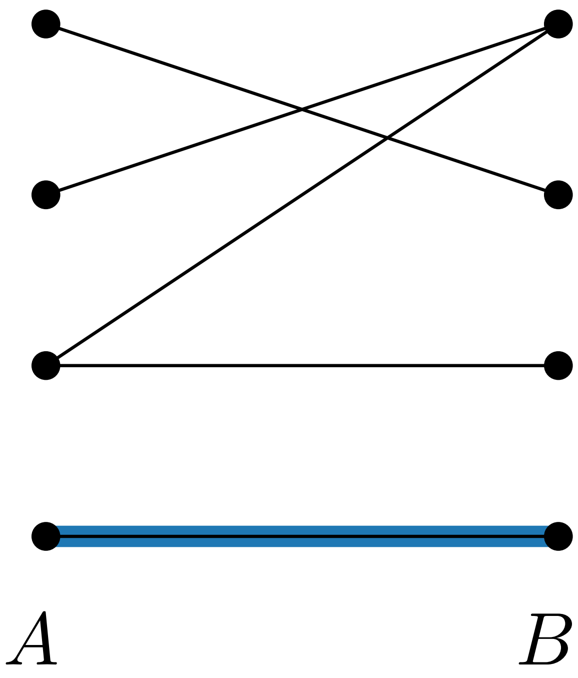
Specialising Maximum Flow

- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



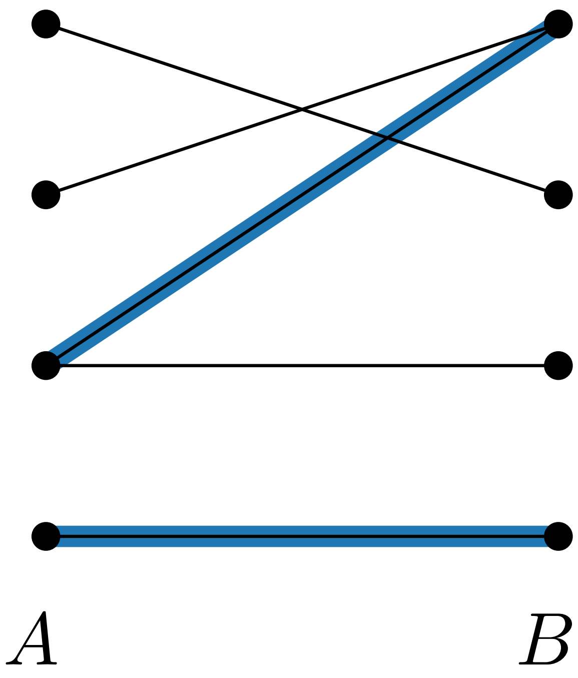
Specialising Maximum Flow

- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



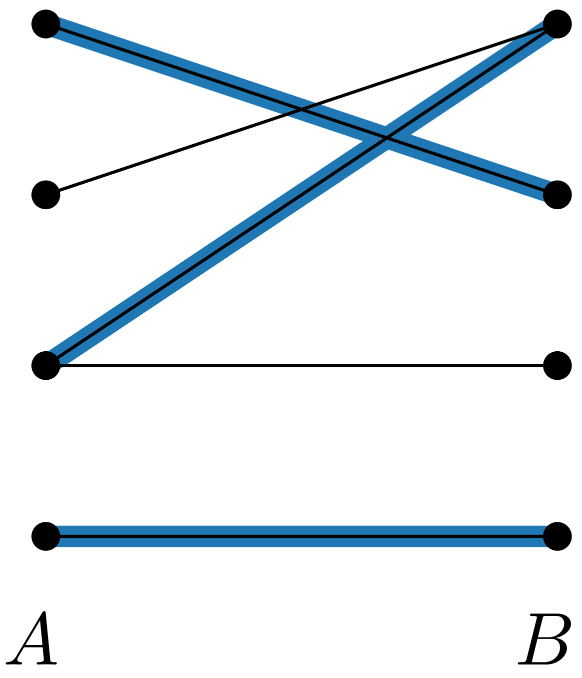
Specialising Maximum Flow

- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



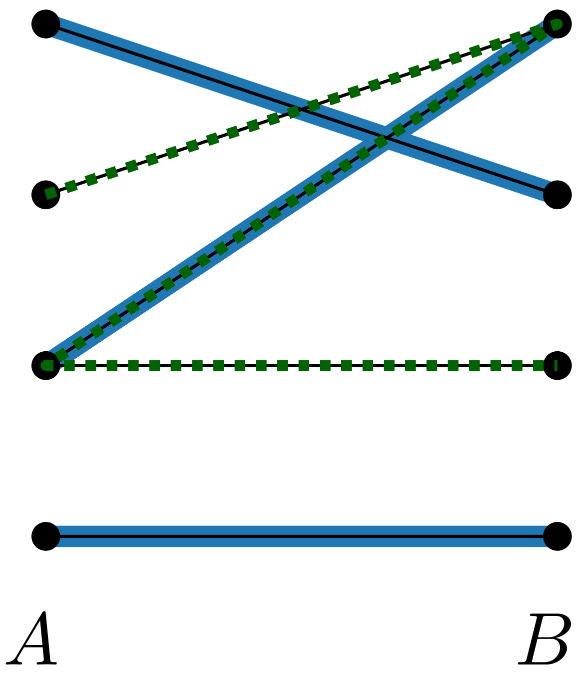
Specialising Maximum Flow

- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



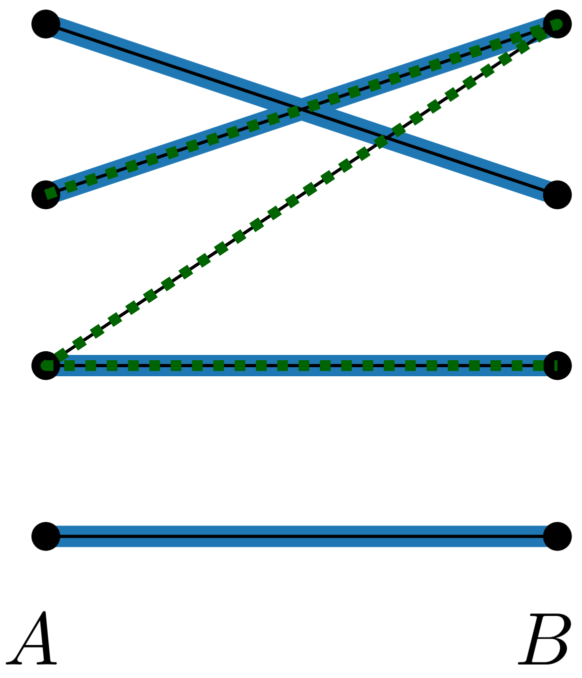
Specialising Maximum Flow

- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



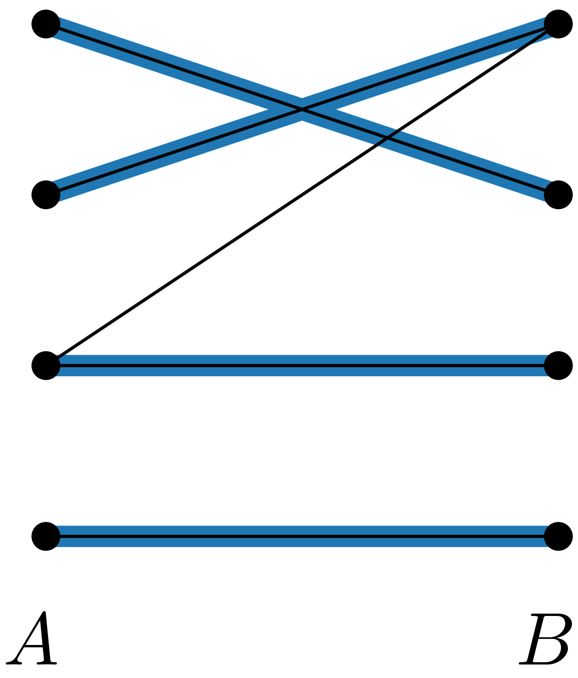
Specialising Maximum Flow

- Using EDMONDSKARP here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



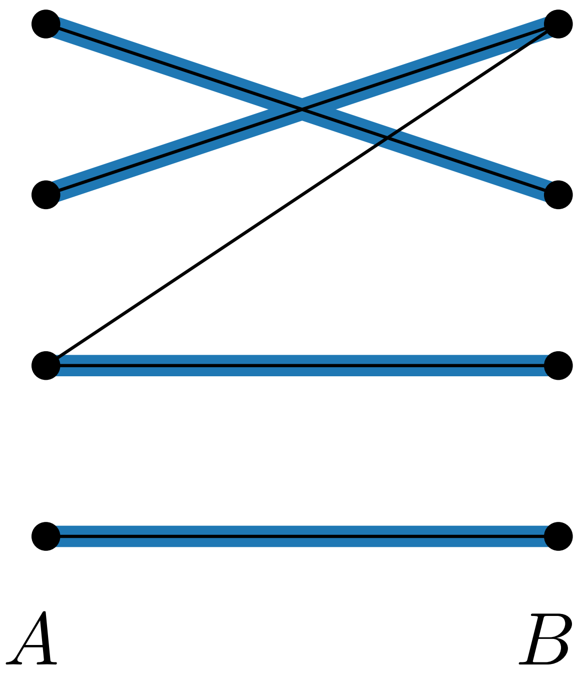
Specialising Maximum Flow

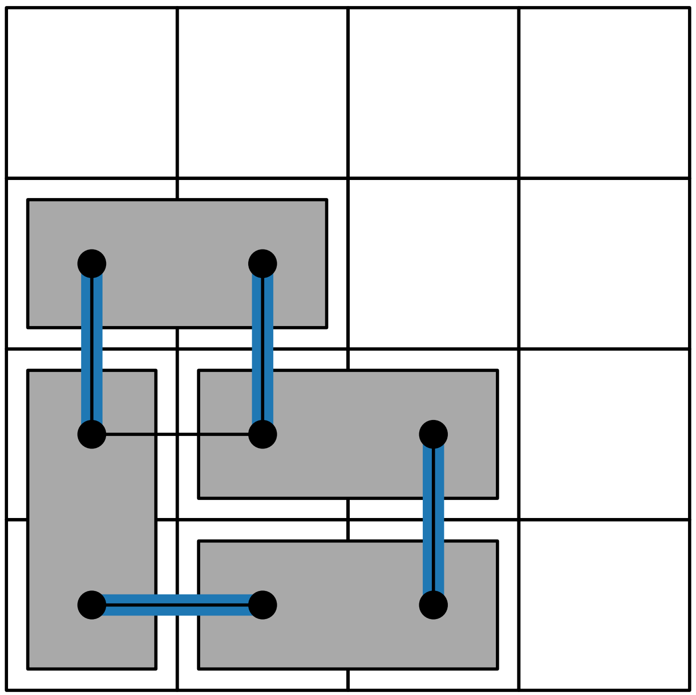
- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.



Specialising Maximum Flow

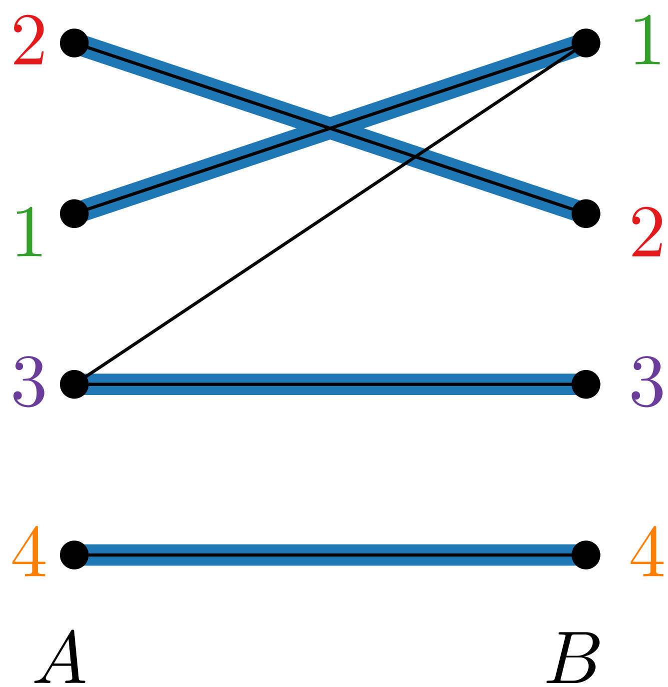
- Using EdmondsKarp here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.

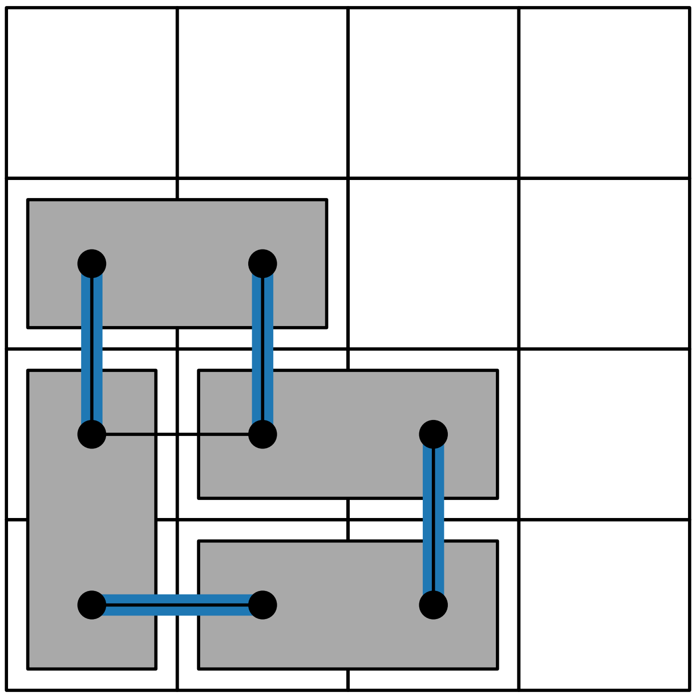




Specialising Maximum Flow

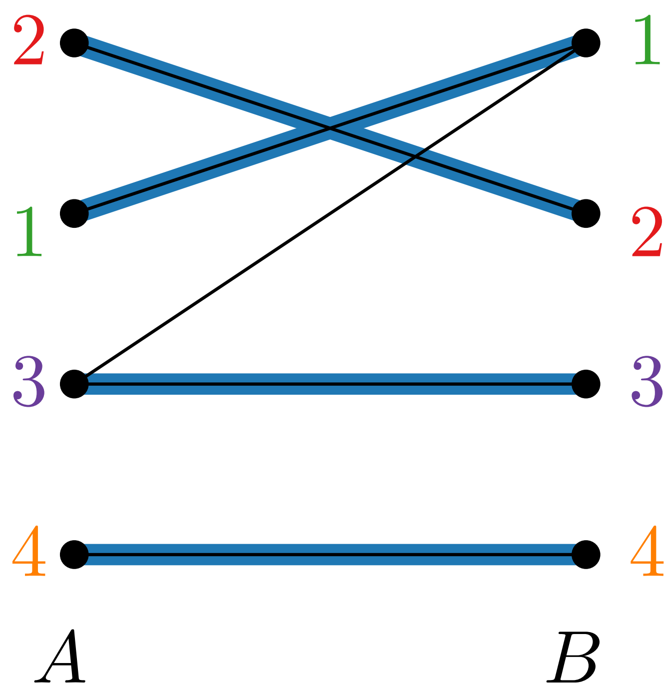
- Using EDMONDSKARP here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.

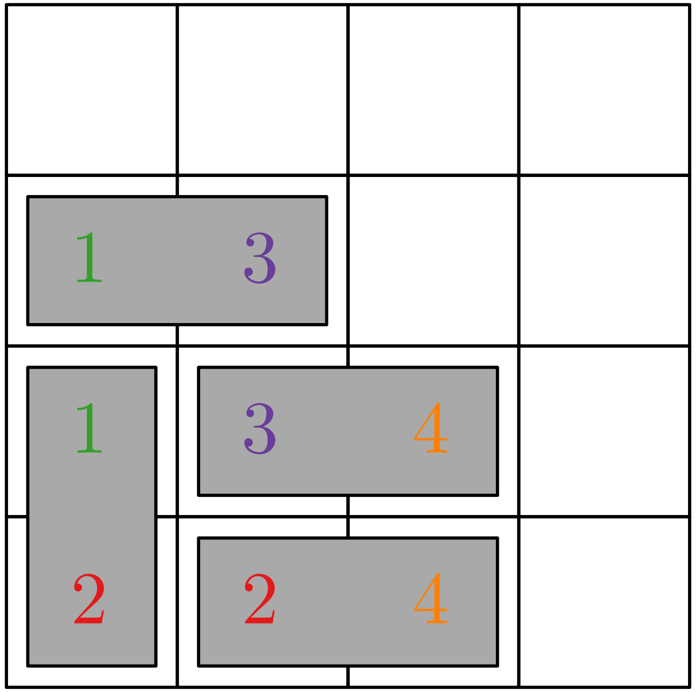




Specialising Maximum Flow

- Using EDMONDSKARP here works, but we can simplify the algo for bipartite matchings.
- Idea: Find augmenting paths from an M-free $a \in A$ to an M-free $b \in B$ until there are none left.





The Domiyes Algorithm

DOMIYES(Domino[] D)

The Domiyes Algorithm

DOMIYES(Domino[] D)

$P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$

Let $f : P \to \mathbb{N}$ be a bijection

The Domiyes Algorithm

DOMIYES(Domino[] D)

$P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$

Let $f : P \to \mathbb{N}$ be a bijection

$A = \{f(p) \mid p \in P \wedge p.x \equiv p.y \pmod 2\}$

$B = \{f(p) \mid p \in P \wedge p.x \not\equiv p.y \pmod 2\}$

The Domiyes Algorithm

DOMIYES(Domino[] D)

$P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$
Let $f : P \to \mathbb{N}$ be a bijection

$A = \{f(p) \mid p \in P \wedge p.x \equiv p.y \pmod 2\}$
$B = \{f(p) \mid p \in P \wedge p.x \not\equiv p.y \pmod 2\}$

$E = \{\{u, v\} \in \binom{P}{2} \mid$ u adj. to v of *diff.* domino$\}$

The Domiyes Algorithm

DOMIYES(Domino[] D)

$P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$

Let $f : P \to \mathbb{N}$ be a bijection

$A = \{f(p) \mid p \in P \wedge p.x \equiv p.y \pmod 2\}$

$B = \{f(p) \mid p \in P \wedge p.x \not\equiv p.y \pmod 2\}$

$E = \{\{u, v\} \in \binom{P}{2} \mid$ u adj. to v of *diff.* domino$\}$

$M = \textsc{MaxBipartiteMatching}(A, B, E)$

The Domiyes Algorithm

$\textsc{Domiyes}$(Domino[] D)

- $P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$
- Let $f : P \to \mathbb{N}$ be a bijection
- $A = \{f(p) \mid p \in P \wedge p.x \equiv p.y \pmod 2\}$
- $B = \{f(p) \mid p \in P \wedge p.x \not\equiv p.y \pmod 2\}$
- $E = \{\{u, v\} \in \binom{P}{2}$ | u adj. to v of *diff.* domino}
- $M = \textsc{MaxBipartiteMatching}(A, B, E)$
- $k = 0$
- **foreach** $\{a, b\} \in M$ **do**
  - $f^{-1}(a)$.number $= k$; $f^{-1}(b)$.number $= k$
  - $k = k + 1$

The Domiyes Algorithm

$n := D$.length

DOMIYES(Domino[] D)

$P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$ — $\mathcal{O}(n)$

Let $f : P \to \mathbb{N}$ be a bijection

$A = \{f(p) \mid p \in P \wedge p.x \equiv p.y \pmod 2\}$

$B = \{f(p) \mid p \in P \wedge p.x \not\equiv p.y \pmod 2\}$ — $\mathcal{O}(n)$

$E = \{\{u, v\} \in \binom{P}{2} \mid$ u adj. to v of *diff.* domino$\}$ — $\mathcal{O}(n^2)$

$M = \text{MaxBipartiteMatching}(A, B, E)$ — $\mathcal{O}(VE)$

$k = 0$

foreach $\{a, b\} \in M$ **do** — $\mathcal{O}(n)$

    $f^{-1}(a)$.number $= k$; $f^{-1}(b)$.number $= k$

    $k = k + 1$

The Domiyes Algorithm

$n := D.\text{length}$

$\textsc{Domiyes}(\text{Domino[]}\ D)$

- $P = \{p_1 \mid (p_1, p_2) \in D\} \cup \{p_2 \mid (p_1, p_2) \in D\}$ — $\mathcal{O}(n)$
- Let $f : P \to \mathbb{N}$ be a bijection
- $A = \{f(p) \mid p \in P \wedge p.x \equiv p.y \pmod 2\}$
- $B = \{f(p) \mid p \in P \wedge p.x \not\equiv p.y \pmod 2\}$ — $\mathcal{O}(n)$
- $E = \{\{u, v\} \in \binom{P}{2} \mid$ u adj. to v of *diff.* domino$\}$ — $\mathcal{O}(n^2)$
- $M = \textsc{MaxBipartiteMatching}(A, B, E)$ — $\mathcal{O}(VE)$
- $k = 0$
- **foreach** $\{a, b\} \in M$ **do** — $\mathcal{O}(n)$
  - $f^{-1}(a).\text{number} = k;\ \ f^{-1}(b).\text{number} = k$
  - $k = k + 1$

$\overline{\mathcal{O}(n^2 + VE)}$

MaxBipartiteMatching

MaxBipartiteMatching$(A, B, E \subseteq \binom{A}{2} \cup \binom{B}{2})$

MaxBipartiteMatching

$\textsc{MaxBipartiteMatching}(A,\ B,\ E \subseteq \binom{A}{2} \cup \binom{B}{2})$

$M = \emptyset$

foreach M-free $a \in A$ **do**

return M

MaxBipartiteMatching

MaxBipartiteMatching$(A, B, E \subseteq \binom{A}{2} \cup \binom{B}{2})$

$M = \emptyset$

foreach M-free $a \in A$ **do**

  if $\exists$ aug. path P from a to M-free $b \in B$ **then**

return M

MaxBipartiteMatching

```
MaxBipartiteMatching(A, B, E ⊆ (A choose 2) ∪ (B choose 2))
    M = ∅
    foreach M-free a ∈ A do
        if ∃ aug. path P from a to M-free b ∈ B then
            foreach uv ∈ P do
                if {u, v} ∈ M then
                    M = M \ {{u, v}}
                else
                    M = M ∪ {{u, v}}
    return M
```

MaxBipartiteMatching

MaxBipartiteMatching$(A,\ B,\ E \subseteq \binom{A}{2} \cup \binom{B}{2})$

```
M = ∅
foreach M-free a ∈ A do
    if ∃ aug. path P from a to M-free b ∈ B then
        foreach uv ∈ P do
            if {u, v} ∈ M then
                M = M \ {{u, v}}
            else
                M = M ∪ {{u, v}}
return M
```

This still runs in $\mathcal{O}(VE)$ time.
However...

MaxBipartiteMatching

$\textsc{MaxBipartiteMatching}(A,\ B,\ E \subseteq \binom{A}{2} \cup \binom{B}{2})$

$M = \emptyset$
foreach M-free $a \in A$ **do**
  if $\exists$ aug. path P from a to M-free $b \in B$ **then**
    foreach $uv \in P$ **do**
      if $\{u,v\} \in M$ **then**
        $M = M \setminus \{\{u,v\}\}$
      else
        $M = M \cup \{\{u,v\}\}$
return M

This still runs in $\mathcal{O}(VE)$ time.
However...

$\Delta \leq 3 \Rightarrow |E| \leq 3V$

MaxBipartiteMatching

MaxBipartiteMatching$(A,\ B,\ E \subseteq \binom{A}{2} \cup \binom{B}{2})$

$M = \emptyset$
foreach M-free $a \in A$ **do**
  if $\exists$ aug. path P from a to M-free $b \in B$ **then**
    foreach $uv \in P$ **do**
      if $\{u, v\} \in M$ **then**
        $M = M \setminus \{\{u, v\}\}$
      else
        $M = M \cup \{\{u, v\}\}$
return M

This still runs in $\mathcal{O}(VE)$ time.
However...

$$\Delta \leq 3 \Rightarrow |E| \leq 3V$$

$\mathcal{O}(VE) =$
$\mathcal{O}(V \cdot 3V) = \mathcal{O}(V^2)$.