# Fundamental Algorithms

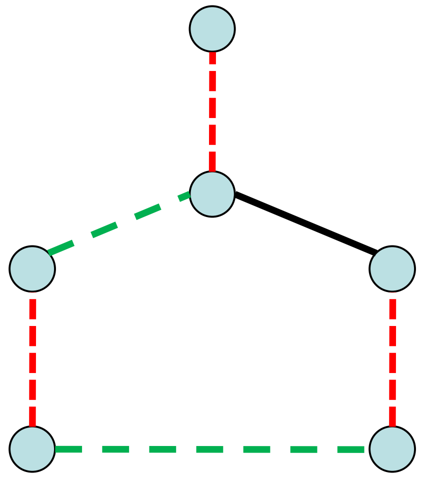
# Chapter 5: Matchings

Christian Scheideler

WS 2017

# Basic Notation

Definition 5.1: Let G=(V,E) be an undirected graph. A matching M in G is a subset of E in which no two edges share a common node.



Matching:

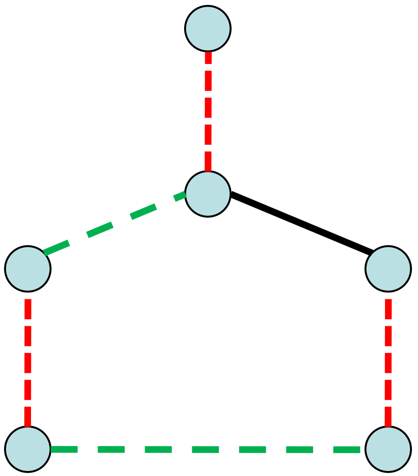
- - - Variant 1

---- Variant 2

# Basic Notation

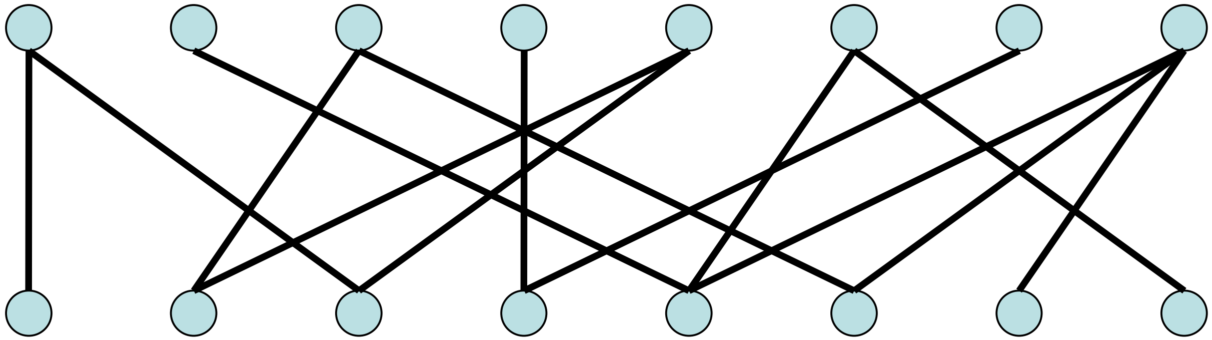
Definition 5.2:

- A matching $M$ in $G=(V,E)$ is called perfect if $|M|=|V|/2$.
- A matching $M$ in is called a maximum matching if there is no matching $M'$ in $G$ with $|M'|>|M|$ (example: red edges)
- A matching $M$ is called maximal if it is maximal w.r.t. „$\subseteq$“, i.e., it cannot be extended (example: green edges)



# Basic Notation

Definition 5.3: Let $G=(V,E)$ be an undirected graph. If $V$ can be partitioned into two non-empty subsets $V_1$ and $V_2$ (i.e., $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$) so that $E \subseteq V_1 \times V_2$, then $G$ is called bipartite (in this case, $G$ may also be defined as $G=(V_1,V_2,E)$ ).



# Foundations

Theorem 5.4: A graph $G=(V,E)$ has a perfect matching if and only if $|V|$ is even and there is no $S\subseteq V$ so that the subgraph induced by $V\setminus S$ contains more than $|S|$ connected components (CC) of odd size.

Proof:

„$\Rightarrow$“: (only direction we prove here)

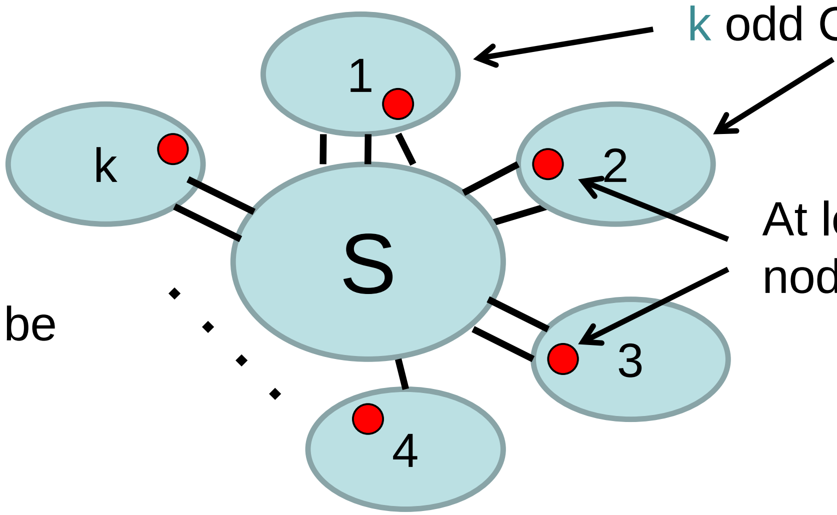
- $|V|$ is odd: certainly, no perfect matching possible
- there is an $S\subseteq V$ so that the subgraph induced by $V\setminus S$ contains more than $|S|$ connected components of odd size

# Foundations

Theorem 5.4: A graph G=(V,E) has a perfect matching if and only if |V| is even and there is no $S \subseteq V$ so that the subgraph induced by V\S contains more than |S| connected components (CC) of odd size.

Proof:

„⇒“:



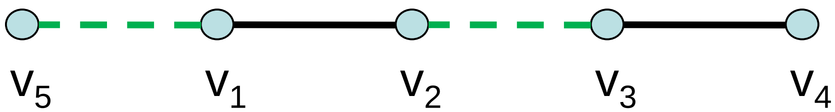

Not all ● can be matched by S.

# Foundations

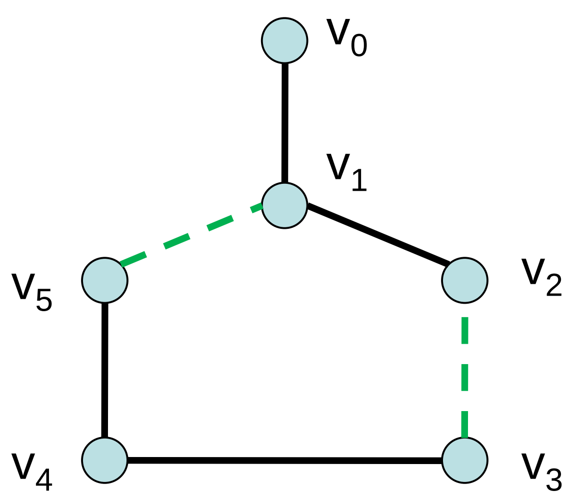
Definition 5.5: A simple path (cycle) $v_0, v_1, \ldots, v_k$ is called alternating w.r.t. a matching $M$ if the edges $\{v_i, v_{i+1}\}$ are alternately in $M$ and not in $M$.

even length:

green edges: in M



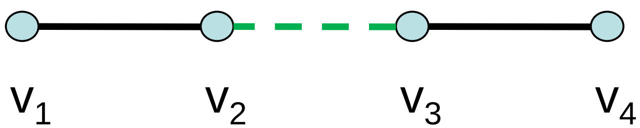

odd length:

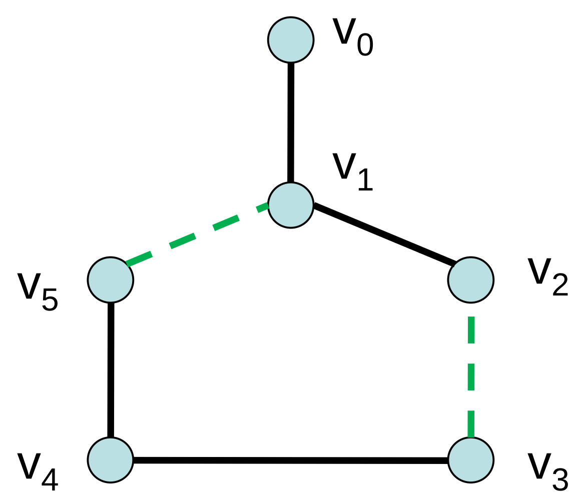
# Foundations

Definition 5.6: An alternating path w.r.t. a matching M is called augmenting if it contains unmatched nodes at both ends and does not form a cycle.

not augmenting ($v_1$ matched):

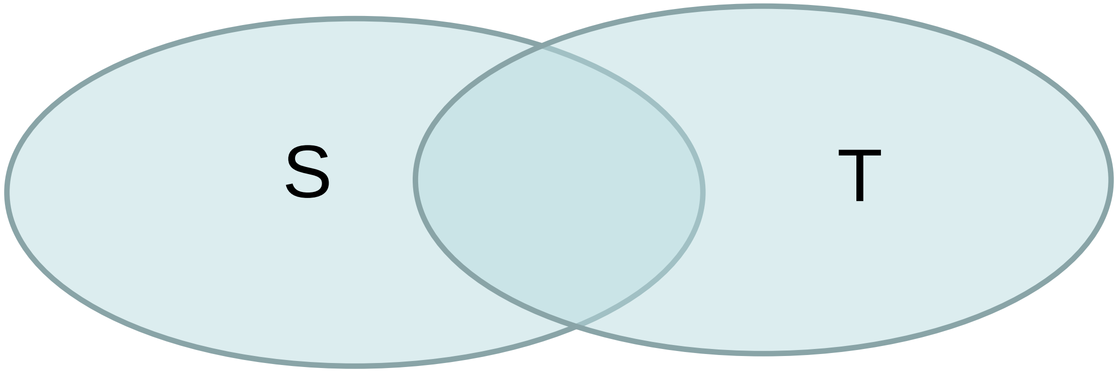



augmenting:

# Foundations

Definition 5.7: Let S and T be two sets. Then $S \ominus T$ denotes the symmetric difference of S and T, i.e., $S \ominus T = (S \setminus T) \cup (T \setminus S)$.



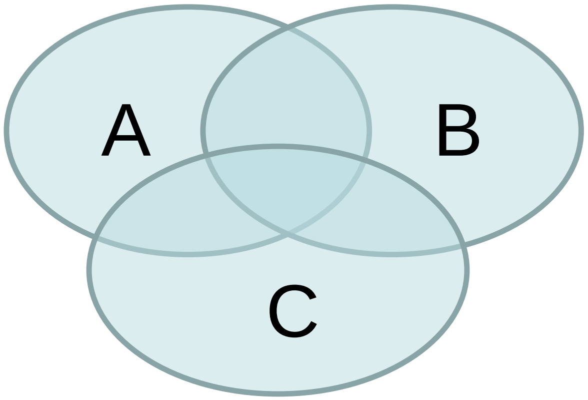

$S \ominus T$: all elements in S and T not in $S \cap T$

# Foundations

Definition 5.7: Let $S$ and $T$ be two sets. Then $S \ominus T$ denotes the symmetric difference of $S$ and $T$, i.e., $S \ominus T = (S \setminus T) \cup (T \setminus S)$.

Rules: for all sets $A,B,C$,

- $A \ominus A = \emptyset$
- $A \ominus B = B \ominus A$
- $(A \ominus B) \ominus C = A \ominus (B \ominus C)$



# Foundations

Definition 5.7: Let $S$ and $T$ be two sets. Then $S \ominus T$ denotes the symmetric difference of $S$ and $T$, i.e., $S \ominus T = (S \setminus T) \cup (T \setminus S)$.

Lemma 5.8: Let $M$ be a matching and $P$ be an augmenting path w.r.t. $M$. Then also $M \ominus P$ is a matching, and it holds that $|M \ominus P| = |M|+1$.

Proof:

change w.r.t. augmenting path $P$:

# Foundations

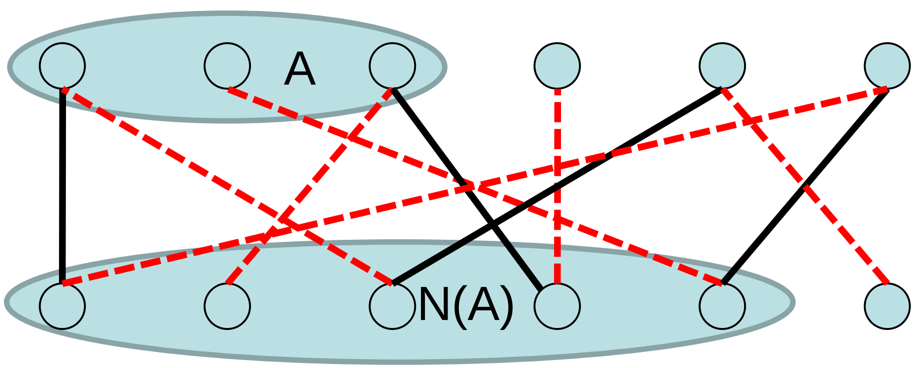
**Theorem 5.9: (Hall´s Theorem)**

Let $G=(U,V,E)$ be a bipartite graph. $G$ contains a matching of cardinality $|U|$ if and only if:

$$\forall A \subseteq U: |N(A)| \geq |A|$$

**Proof:**

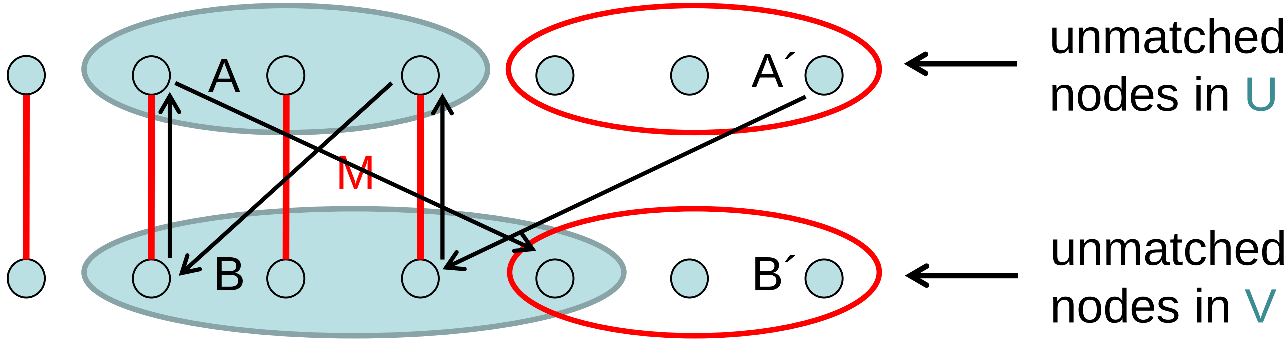
„$\Rightarrow$“: clear due to matching edges



# Foundations

Proof:

„⇐“: Let $M$ be a maximum matching in $G$ with $|M|<|U|$.




$A\subseteq U$: nodes reachable via alternating paths starting in $A´$

$B\subseteq V$: nodes reachable via alternating paths starting in $A´$

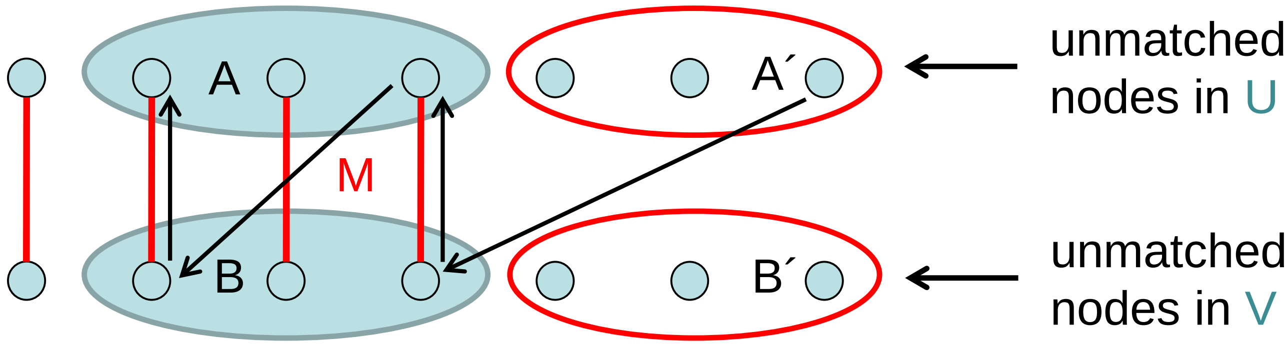
Observations:

- $A\cap A´=\emptyset$ because a node in $U$ can only be reached by an alternating path from $A´$ if it has an edge in $M$
- $B\cap B´=\emptyset$ because if $B\cap B´\neq\emptyset$ then there is an augmenting path (see picture), so $M$ is not maximum, leading to a contradiction!

# Foundations

Proof:

„$\Leftarrow$“: Let $M$ be a maximum matching in $G$ with $|M|<|U|$.




$A\cap A´=\emptyset$ and $B\cap B´=\emptyset$:

- $|A|=|B|$ since $A=\{ u\in U \mid \exists v\in B$ with $\{u,v\}\in M \}$
- $N(A´)\subseteq B$ and $N(A)\subseteq B$ because otherwise $B$ would be extendible
- Hence, $|N(A\cup A´)|\leq|B|=|A|<|A\cup A´|$ since $|A´|>0$

# Foundations

Alternative proof for „$\Leftarrow$“:

- Suppose that $\forall A \subseteq U$: $|N(A)| \geq |A|$.
- Let $M$ be a matching in $G$ with $|M|<|U|$, and let $u_0 \in U$ be an unmatched node.
- Since $|N(\{u_0\})| \geq 1$, $u_0$ has a neighbor $v_1 \in V$. If $v_1$ is unmatched, we are done because we have already found an augmenting path.
- Otherwise let $u_1 \in U$ be the node matched with $v_1$. Since $u_1 \notin \{u_0\}$ and $|N(\{u_0,u_1\})| \geq 2$, there is a node $v_2 \notin \{v_1\}$ that is adjacent to $u_0$ or $u_1$. If $v_2$ is unmatched, we are done because we have already found an augmenting path.
- Otherwise, let $u_2 \in U$ be the node matched with $v_2$. Since $u_2 \notin \{u_0,u_1\}$ and $|N(\{u_0,u_1,u_2\})| \geq 3$, there is a node $v_3 \notin \{v_1, v_2\}$ that is adjacent to a node in $\{u_0,u_1,u_2\}$. If $v_3$ is unmatched, then we are done, otherwise we continue as above.
- Since $|M|<|V|$ and $|V|<\infty$, we finally have to get to an unmatched node $v_k$, and we can increase the matching.

# Foundations

Theorem 5.10: (Berge´s theorem, bipartite graphs)
A matching in a bipartite graph is a maximum matching if and only if there is no augmenting path for that matching.

Proof:

„$\Rightarrow$“: (also holds for arbitrary graphs)

- Suppose that there is an augmenting path $P$ for some matching $M$.
- Then it follows from Lemma 5.8 that $|M \ominus P| = |M|+1$, which implies that $M$ cannot be a maximum matching.

# Foundations

Theorem 5.10: (Berge´s theorem, bipartite graphs)
A matching in a bipartite graph is a maximum matching if and only if there is no augmenting path for that matching.

Proof:

„$\Leftarrow$“:

- Certainly holds for bipartite graphs that satisfy Hall´s theorem.
- We will show the general validity later.

# Foundations

Theorem 5.10: (Berge´s theorem, bipartite graphs)
A matching in a bipartite graph is a maximum matching if and only if there is no augmenting path for that matching.

This theorem also holds for general graphs:

Theorem 5.11: (Berge´s theorem)
A matching in an arbitrary graph is a maximum matching if and only if there is no augmenting path for that matching.

# Foundations

Berge´s theorem, if correct, implies the following algorithm for computing a maximum matching:

$M := \emptyset$
while $\exists$augmenting $P$ w.r.t. $M$ do
    $M := M \ominus P$
output $M$

Runtime:

- The while-loop is executed at most $n$ times.
- The search for an augmenting path can be done in $O(n+m)$ time in general graphs, as we will see later.

Therefore, a runtime of $O(n \cdot (n+m))$ is possible.

# Matching in Bipartite Graphs

Berge´s theorem, if correct, implies the following algorithm for computing a maximum matching:

$M := \emptyset$
while $\exists$augmenting $P$ w.r.t. $M$ do
    $M := M \ominus P$
output $M$

Remarks:

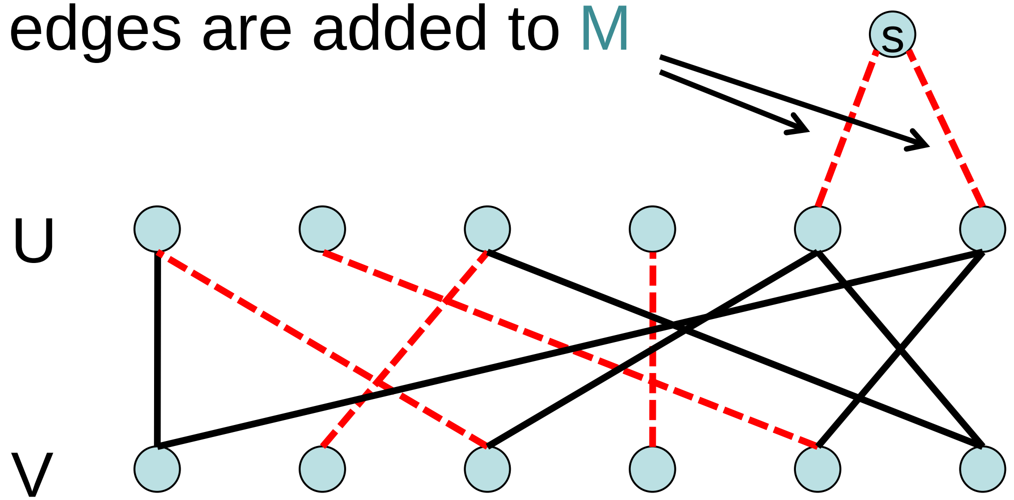
- In a bipartite graph $G=(U,V,E)$ it suffices to search for augmenting paths starting from unmatched nodes in $U$ because every augmenting path must have one unmatched node in $U$ and one in $V$.
- In bipartite graphs we can use an alternating DFS approach to find augmenting paths.

# Matching in Bipartite Graphs

Simplification for alternating DFS in bipartite graphs:

artificial source s with edges to all unmatched nodes in U

Artificial edges are added to M



# Matching in Bipartite Graphs

- E(u): edge set of node u

```
Procedure AlternatingBipartiteDFS(s: Node, M: Matching)
  d = <∞,…, ∞>: Array [1..n] of IN
  parent = <⊥,…,⊥>: Array [1..n] of Node
  d[key(s)]:=0              // s has distance 0 to itself
  parent[key(s)]:=s         // s is its own parent
  q:=<s>: Stack of Node
  while q ≠<> do            // as long as q is not empty
    u:= q.pop()             // process nodes according to LIFO rule
    if (d[key(u)] is even) then A:=M else A:=E\M
    if A∩E(u)=∅ and (d[key(u)] is even) then   // u unmatched?
      return augmenting path (via parent[])
    else
      foreach {u,v}∈A∩E(u) do
        if parent(key(v))=⊥ then  // v not visited so far?
          q.push(v)               // add v to q
          d[key(v)]:=d[key(u)]+1
          parent[key(v)]:=u
```

# Matching in Bipartite Graphs

Correctness of AlternatingBipartiteDFS:

- Suppose that there is an augmenting path $p=(s,u_1,v_1,u_2,v_2,...,v_k)$ w.r.t. $M$ but AlternatingBipartiteDFS does not find any.
- Let $w$ be the last node in $p$ that was explored by the algorithm. Certainly, $w\neq v_k$ because otherwise the algorithm would have found an augmenting path.
- Suppose that $w=v_i$ for some $i<k$. Then the algorithm would have also explored $u_i$ via the matching edge, leading to a contradiction.
- So suppose that $w=u_i$ for some $i<k$. Then the algorithm would have also explored $v_{i+1}$ via a non-matching edge, also leading to a contradiction.

# Foundations

Proof of Theorem 5.11: „$\Leftarrow$“ follows from the following lemma.

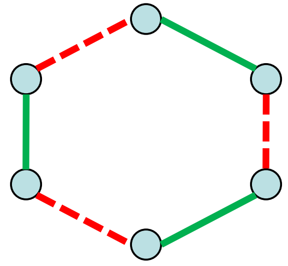
Lemma 5.12: Let $M$ and $N$ be matchings in $G$, and let $|N|>|M|$. Then $N\ominus M$ contains at least $|N|-|M|$ node-disjoint augmenting paths w.r.t. $M$.

Proof:

The degree of a node in $(V, N\ominus M)$ is at most $2$. The connected components of $(V,N\ominus M)$ are either

- isolated nodes,

- simple cycles (of even length), or



- alternating paths

# Foundations

Proof of Lemma 5.12:

- Let $C_1,\ldots,C_k$ be the connected components in $(V,N\ominus M)$.
- Then it follows from the rules for $\ominus$ that

$$M\ominus\underbrace{C_1\ominus\ldots\ominus C_k}_{N\ominus M} = N$$

- Note that the $C_i$´s are node-disjoint, so they can be applied independently to $M$.
- It is easy to check that if $C_i$ is a simple cycle or an alternating path that is not augmenting, then $|M\ominus C_i|\leq|M|$.
- Hence, only those $C_i$´s that are augmenting paths w.r.t. $M$ can increase the matching, and this by exactly $1$.
- Therefore, there must be at least $|N|-|M|$ $C_i$´s that are augmenting (and node-disjoint) paths w.r.t. $M$.

# Foundations

Consequence: approach for finding a maximum matching in bipartite graphs also works for arbitrary graphs.

We will study the following refined approach:

$M := \emptyset$
while $\exists$ augmenting path $P$ w.r.t. $M$ do
- determine a shortest augmenting path $P$ w.r.t. $M$
- $M := M \ominus P$

output $M$

In the following let

- $P_i$: augmenting path found in round $i$
- $M_i$: matching at the end of round $i$

# Shortest augmenting Paths

Lemma 5.13: Let $M$ be a matching of cardinality $r$ and let $s$ be the maximum cardinality of a matching in $G=(V,E)$, $s>r$. Then there is an augmenting path w.r.t. $M$ of length $\leq 2\lfloor r/(s-r)\rfloor+1$.

Proof:

- Let $N$ be a maximum matching in $G$, i.e., $|N|=s$.
- By Lemma 5.12, $N \ominus M$ contains $\geq s-r$ augmenting paths w.r.t. $M$, which are node-disjoint and therefore also edge-disjoint.
- At least one of these paths contains $\leq\lfloor r/(s-r)\rfloor$ edges from $M$.

# Shortest augmenting Paths

Lemma 5.14: Let $s$ be the maximum cardinality of a matching in $G=(V,E)$. Then the sequence $|P_1|, |P_2|, \ldots$ of shortest augmenting paths computed by the refined algorithm contains at most $2\sqrt{s}+1$ different values.

Proof:

- Let $r := \lfloor s-\sqrt{s} \rfloor$. By construction, $|M_i|=i$, and therefore $|M_r|=r$. From Lemma 5.13 it follows that

$$|P_r| \le 2\left\lfloor \frac{\lfloor s-\sqrt{s}\rfloor}{s - \lfloor s-\sqrt{s}\rfloor} \right\rfloor + 1 \le 2\lfloor s/\sqrt{s}\rfloor + 1 \le 2\lfloor\sqrt{s}\rfloor + 1$$

- Thus, for $i \le r$, $|P_i|$ is one of the odd numbers in $[1, 2\sqrt{s}+1]$, and therefore one of $\lfloor\sqrt{s}\rfloor+1$ odd numbers.
- $P_{r+1},\ldots,P_s$ contribute at most $s-r<\sqrt{s}+1$ additional lengths.

# Shortest augmenting Paths

Lemma 5.15: Let $P$ be a shortest augmenting path w.r.t. $M$ and $P'$ be an augmenting path w.r.t $M \ominus P$. Then it holds that:

$$|P'| \geq |P| + 2|P \cap P'|$$

Proof:

- Let $N = M \ominus P \ominus P'$, so $|N| = |M| + 2$.
- By Lemma 5.12, $M \ominus N$ contains at least 2 node-disjoint augmenting paths w.r.t. $M$, called $P_1$ and $P_2$.
- It holds: $|M \ominus N| = |P \ominus P'| = |(P \setminus P') \cup (P' \setminus P)|$
  $= |P| + |P'| - 2|P \cap P'|$

  and $|M \ominus N| \geq |P_1| + |P_2| \geq 2|P|$ (by def. of $P$)
- Therefore, $|P| + |P'| - 2|P \cap P'| \geq 2|P|$
  $\Rightarrow \quad |P'| \geq 2|P| - |P| + 2|P \cap P'|$

# Shortest augmenting Paths

Recall our refined matching algorithm:

$M:=\emptyset$

**while** $\exists$augmenting path w.r.t. $M$ **do**

- determine a shortest augmenting path $P$ w.r.t. $M$
- $M:=M\ominus P$

output $M$

- Let $P_1$, $P_2$,... be the sequence of shortest augmenting paths constructed by the algorithm.
- Lemma 5.15: $|P_{i+1}| \geq |P_i|$ for all $i$.

# Shortest augmenting Paths

Lemma 5.16: For every sequence $P_1$, $P_2$,... of shortest augmenting paths it holds for all $P_i$ and $P_j$ with $|P_i|=|P_j|$ that $P_i$ and $P_j$ are node-disjoint.

Proof:

- Suppose that there is a sequence $(P_k)_{k\geq 1}$ with $|P_i|=|P_j|$ for some $j>i$ so that $P_i$ and $P_j$ are not node-disjoint, where $j-i$ is minimal.
- Then the paths $P_i,\ldots,P_{j-1}$ resp. $P_{i+1},\ldots,P_j$ are node-disjoint.
- Therefore, $P_j$ is an augmenting path w.r.t. the matching $M$ after the augmentations by $P_1,\ldots,P_i$.
- From Lemma 5.15 it follows that $|P_j|\geq|P_i|+2|P_i\cap P_j|$, and since $|P_i|=|P_j|$, $P_i$ and $P_j$ must be edge-disjoint.
- The matching edges created by $P_i$ are still in $M\ominus P_{i+1}\ominus P_{i+2}\ominus\ldots\ominus P_{j-1}$ because $P_i,\ldots,P_{j-1}$ are node-disjoint.
- Since $P_j$ has a node in common with $P_i$, $P_j$ has to have an edge (namely, a matching edge) in common with $P_i$ as well.
- However, this cannot be, so $P_i$ and $P_j$ must be node-disjoint.

# Shortest augmenting Paths

Hopcroft-Karp Algorithm:

$M := \emptyset$

while $\exists$ augmenting path w.r.t. $M$ do

- $l :=$ length of shortest augmenting path w.r.t. $M$
- determine w.r.t. „$\subseteq$“ a maximal set of node-disjoint augmenting paths $Q_1,\ldots,Q_k$ w.r.t. $M$ that have length $l$
- $M := M \ominus Q_1 \ominus \ldots \ominus Q_k$

Corollary 5.17: The while-loop above is executed at most $O(\sqrt{n})$ times.

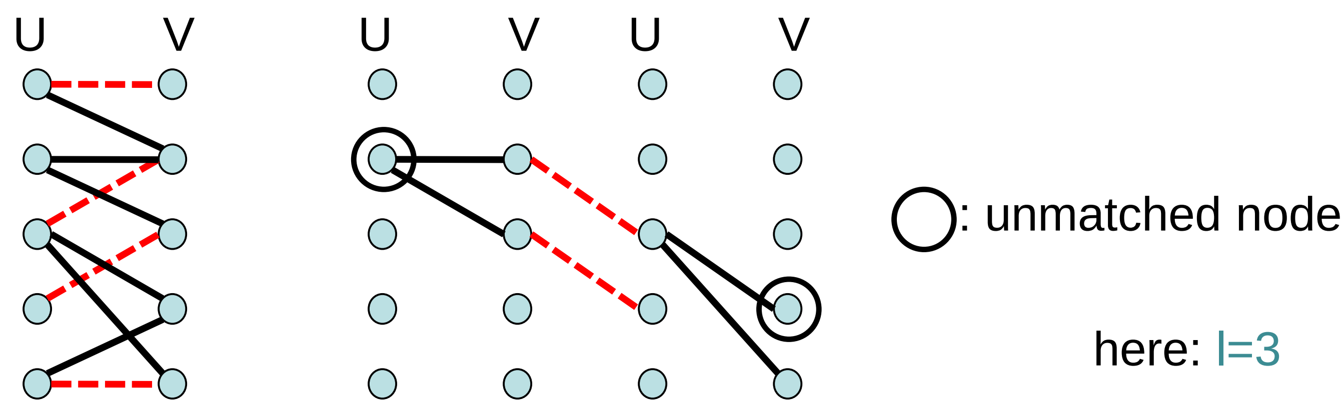
Proof: follows from Lemmas 5.14-5.16

# Shortest augmenting Paths

Question: How can we quickly find a set of shortest augmenting paths w.r.t. matching M?

Graph G bipartite, i.e., G=(U,V,E):

- Determining the shortest length l: alternating BFS, starting with all unmatched nodes in U, until an unmatched node is found in V



# Shortest augmenting Paths

- s: artificial node (see Slide 21), E(u): edge set of node u

```
Procedure AlternatingBipartiteBFS(s: Node, M: Matching)
    d = <∞,…, ∞>: Array [1..n] of IN
    parent = <⊥,…,⊥>: Array [1..n] of Node
    d[key(s)]:=0              // s has distance 0 to itself
    parent[key(s)]:=s         // s is its own parent
    q:=<s>: Queue of Node
    while q ≠<> do            // as long as node is not empty
        u:= q.dequeue()       // process nodes according to FIFO rule
        if (d[key(u)] is even) then A:=M else A:=M\E
        if A∩E(u)=∅ and (d[key(u)] is even) then
            augmenting path (via parent[]), stop
        else
            foreach {u,v}∈A∩E(u) do
                if parent(key(v))=⊥ then  // v not visited so far?
                    q.enqueue(v)  // add v to the queue q
                    d[key(v)]:=d[key(u)]+1
                    parent[key(v)]:=u
```

# Shortest augmenting Paths

Graph G bipartite, i.e., G=(U,V,E):

- Determining the shortest length $l$: alternating BFS, started with all unmatched nodes in $U$, until an unmatched node is found in $V$ or all nodes have been found.
- Remember the BFS-depth of each node.
- Determining a maximal set of shortest augmenting paths: Initially, the nodes are unmarked. Perform one after the other from each unmatched node in $U$ an alternating DFS along unmarked nodes of increasing BFS-depth up to depth $l$ until we have found an augmenting path $Q_i$ or all edges have been explored.
- For every found path $Q_i$, all nodes in $Q_i$ are marked and we continue to execute DFS from another unmatched node in $U$.
- Every node at which DFS backtracks (i.e., no augmenting path was found) will be marked.

Since every node and edge is only processed once in the BFS and DFS, the runtime is $O(n+m)$.

# Shortest augmenting Paths

Correctness of the algorithm for determining a maximal set of shortest augmenting paths (here called refined AlternatingBipartiteDFS):

- Suppose that there is an augmenting path $p=(u_1,v_1,u_2,v_2,\ldots,v_{2k+1})$ w.r.t. $M$ of length $l=2k+1$ that is not discovered by the refined AlternatingBipartiteDFS algorithm.
- This can only happen if the nodes of $p$ do not have a consecutive BFS-depth.
- Suppose w.l.o.g. that BFS-depth($v_i$) $\neq$ BFS-depth($u_i$)+1 for some $i$.
- Case 1: BFS-depth($v_i$) > BFS-depth($u_i$)+1. Then the alternating BFS algorithm would not have worked correctly because it should have reached $v_i$ from $u_i$, so that cannot happen.
- Case 2: BFS-depth($v_i$) < BFS-depth($u_i$)+1. Then it is possible to construct an augmenting path of length less than $l$ (go along the shortest alternating path from an unmatched node $u$ to $v_i$ instead of using $p$ to reach $v_i$), also contradicting our assumption that the alternating BFS algorithm works correctly.

# Shortest augmenting Paths

Corollary 5.18: In bipartite graphs, a maximum matching can be computed in $O(\sqrt{n}\,(n+m))$ time.

Is this also possible for arbitrary graphs?

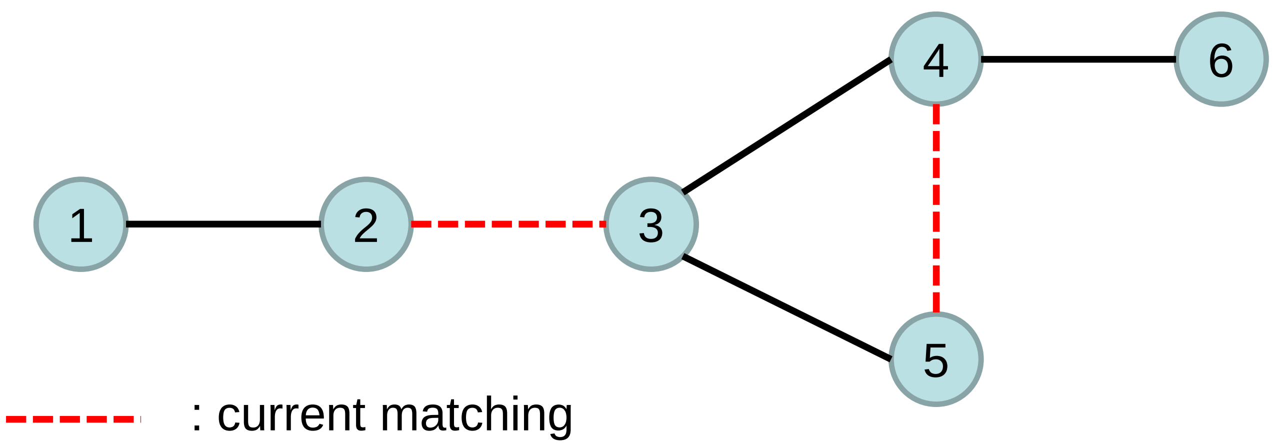
Yes, but it´s much more complicated:

- Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}\,E)$ general graph maximum matching algorithm. Combinatorica 14(1), pp. 71-109 (1994).

# Matching in arbitrary Graphs

Problem: BFS in bipartite graph is not applicable in general graphs.

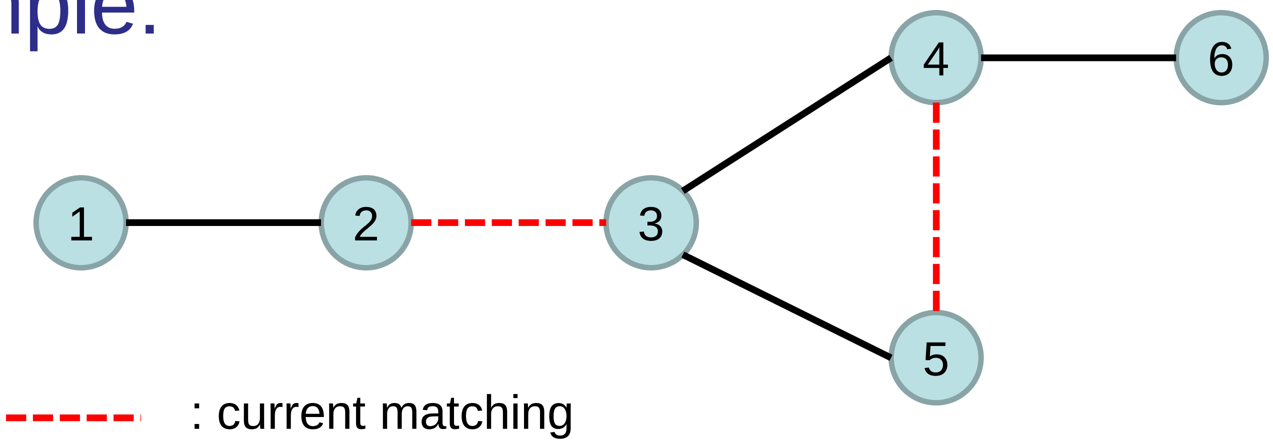
Example:



# Matching in arbitrary Graphs

Alternating BFS from 1 via node 4:
  misses augmenting path 1-2-3-5-4-6 since
  4 has already been visited

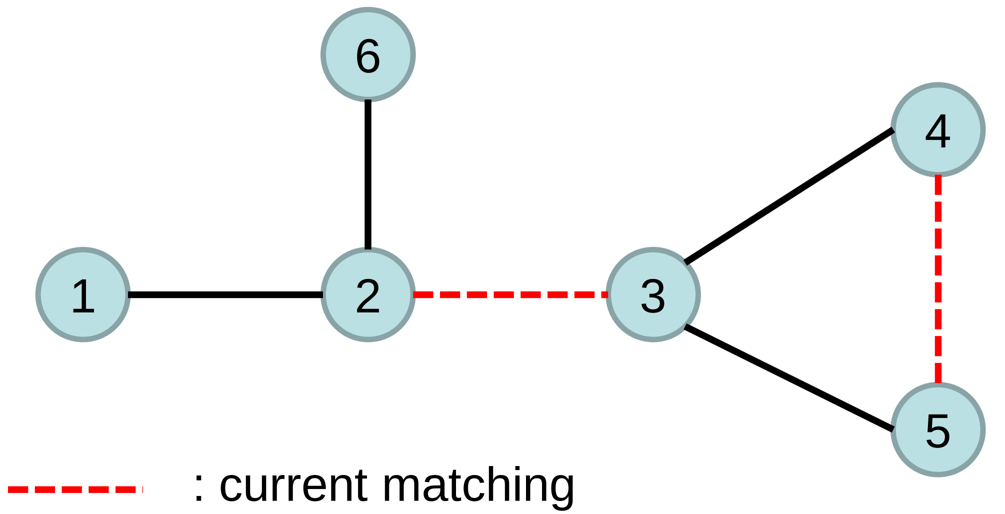
Example:



# Matching in arbitrary Graphs

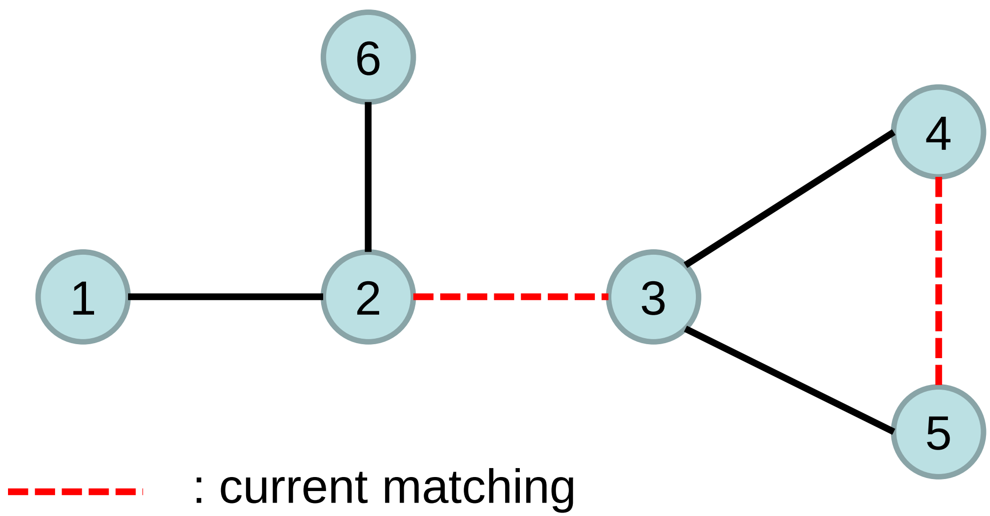
If we allow nodes to be visited multiple times, then there are other problems

Example:



# Matching in arbitrary Graphs

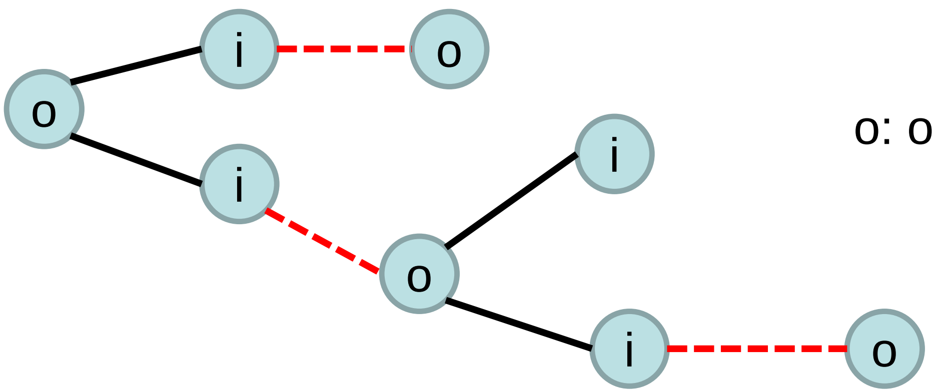
Then it seems that 1-2-3-4-5-3-2-6 is an augmenting path although the example below does not contain any.



# Matching in arbitrary Graphs

Edmonds´ Algorithm:

- Build a tree of alternating paths via alternating BFS.
- The root and all nodes of even distance from the root are the outer nodes.
- The other nodes are the inner nodes.

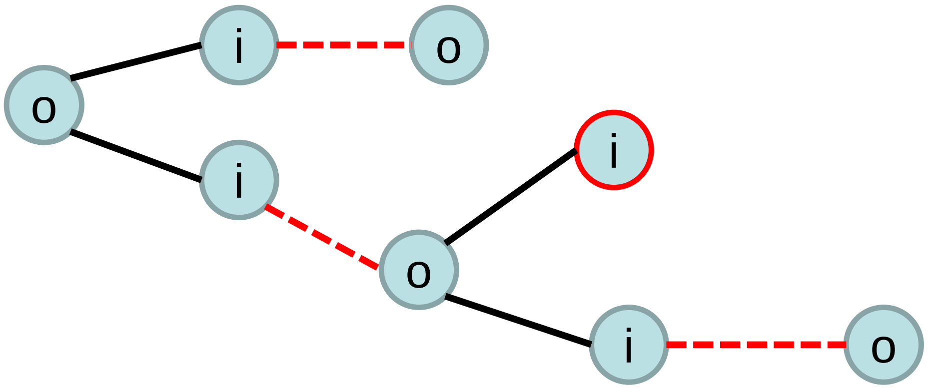



o: outer, i: inner

# Matching in arbitrary Graphs

Edmonds´ Algorithm:

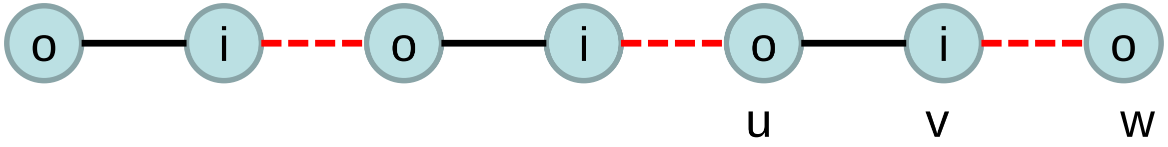
- Build a tree of alternating paths via alternating BFS.
- The root and all nodes of even distance from the root are the outer nodes.
- The other nodes are the inner nodes.
- If the search ends in an unmatched inner node, then there is an augmenting path to that node, as one can easily check.



# Matching in arbitrary Graphs

Edmonds´ Algorithm:

- Build a tree of alternating paths via alternating BFS.
- The root and all nodes of even distance from the root are the outer nodes.
- The other nodes are the inner nodes.
- If the search ends in an unmatched inner node, then there is an augmenting path to that node, as one can easily check.
- If the BFS is currently at an outer node u, then all unmatched edges {u,v} for some node v that is not already in the tree are added to the tree. Such a node v is then an inner node. If v is not matched, we have found an augmenting path. Otherwise, if w is not already in the tree, we also add the unique matching edge {v,w} to the tree and declare w an outer node.



# Matching in arbitrary Graphs

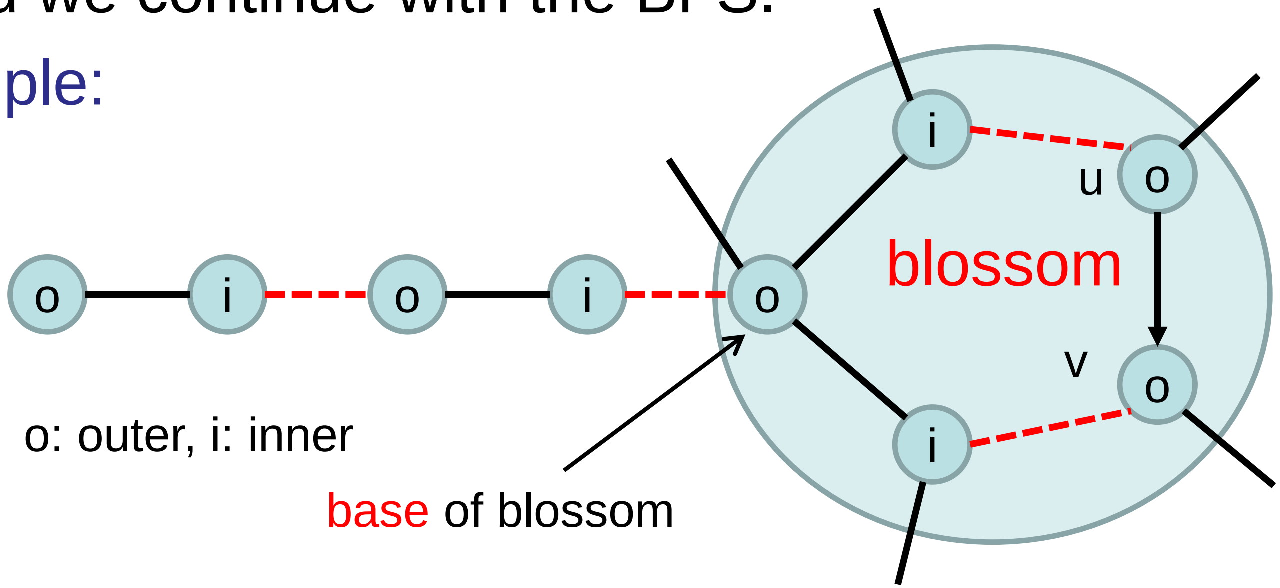
Edmonds´ Algorithm:

- Build a tree of alternating paths via alternating BFS.
- The root and all nodes of even distance from the root are the outer nodes.
- The other nodes are the inner nodes.
- If the search ends in an unmatched inner node, then there is an augmenting path to that node, as one can easily check.
- If the BFS is currently at an outer node u, then all unmatched edges {u,v} for some node v that is not already in the tree are added to the tree. Such a node v is then an inner node. If v is not matched, we have found an augmenting path. Otherwise, if w is not already in the tree, we also add the unique matching edge {v,w} to the tree and declare w an outer node.
- If for some outer node u an edge {u,v} is found where v is already an outer node, then we have a cycle.

# Matching in arbitrary Graphs

- If for some outer node u an edge {u,v} is found where v is already an outer node, then we have a cycle, which is also called a blossom.
- The cycle will then be merged into a single outer node, and we continue with the BFS.

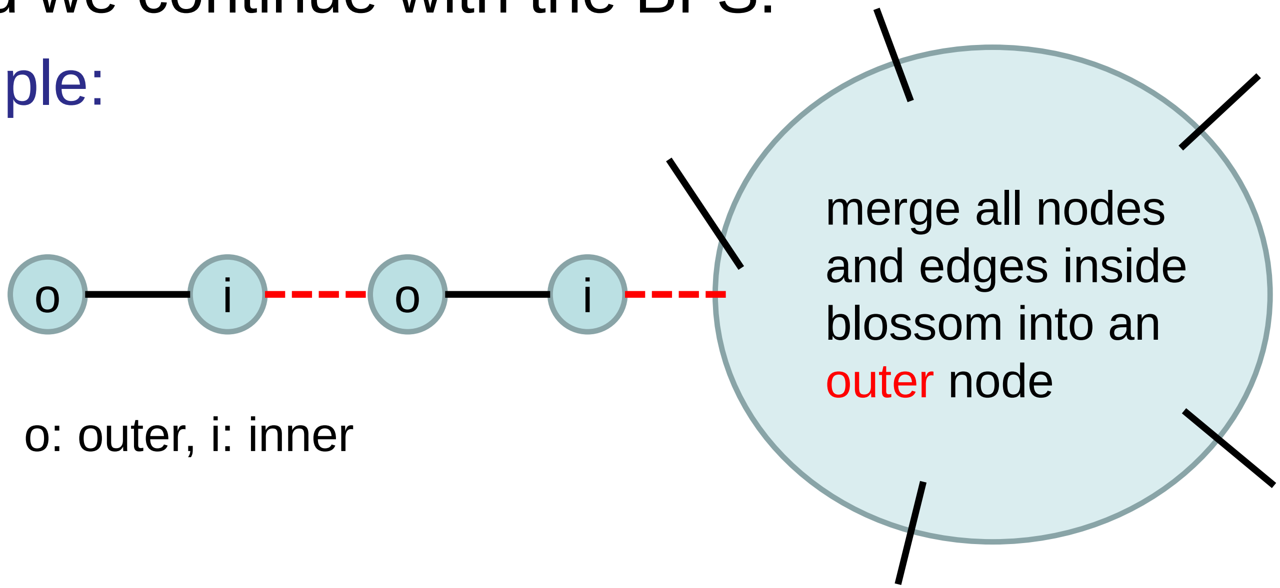
Example:



# Matching in arbitrary Graphs

- If for some outer node u an edge {u,v} is found where v is already an outer node, then we have a cycle, which is also called a blossom.
- The cycle will then be merged into a single outer node, and we continue with the BFS.

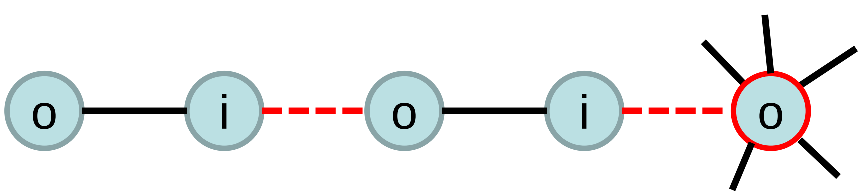
Example:



# Matching in arbitrary Graphs

- If for some outer node u an edge {u,v} is found where v is already an outer node, then we have a cycle, which is also called a blossom.
- The cycle will then be merged into a single outer node, and we continue with the BFS.

Example:




o: outer, i: inner

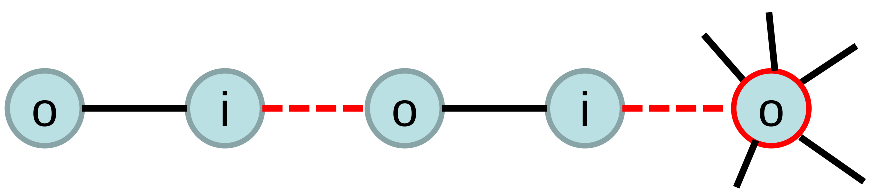
resulting graph: contracted graph

# Matching in arbitrary Graphs

Lemma 5.19: The contracted graph G´ has an augmenting path if and only if the original graph G has an augmenting path.

Proof sketch:

Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge

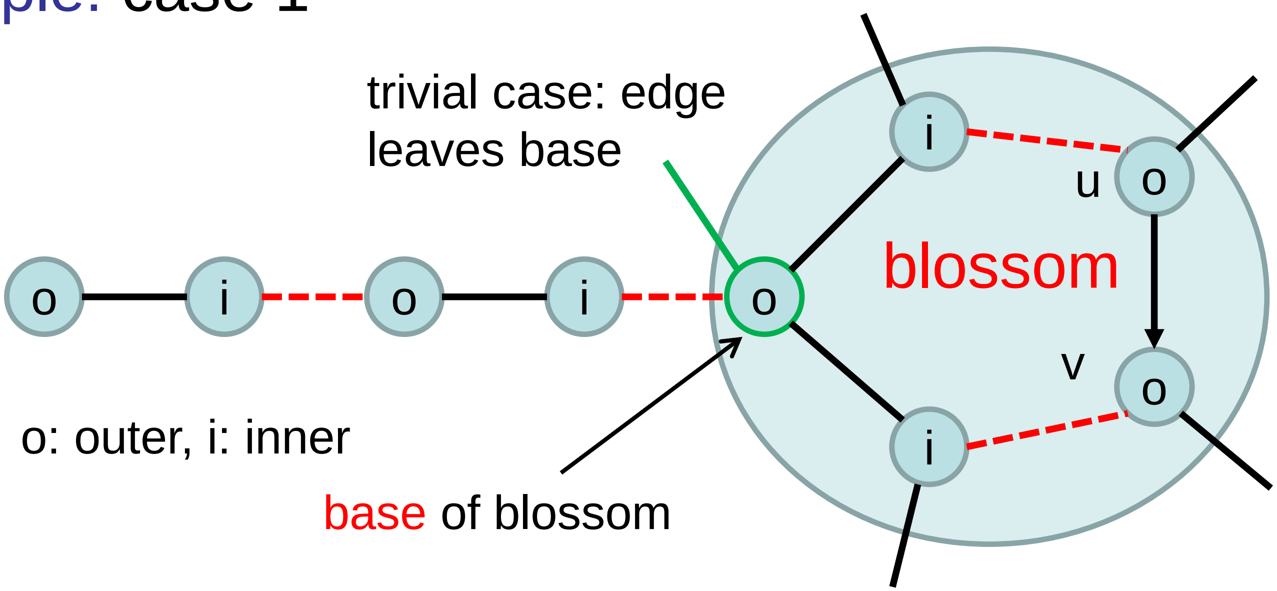



o: outer, i: inner

# Matching in arbitrary Graphs

Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge.

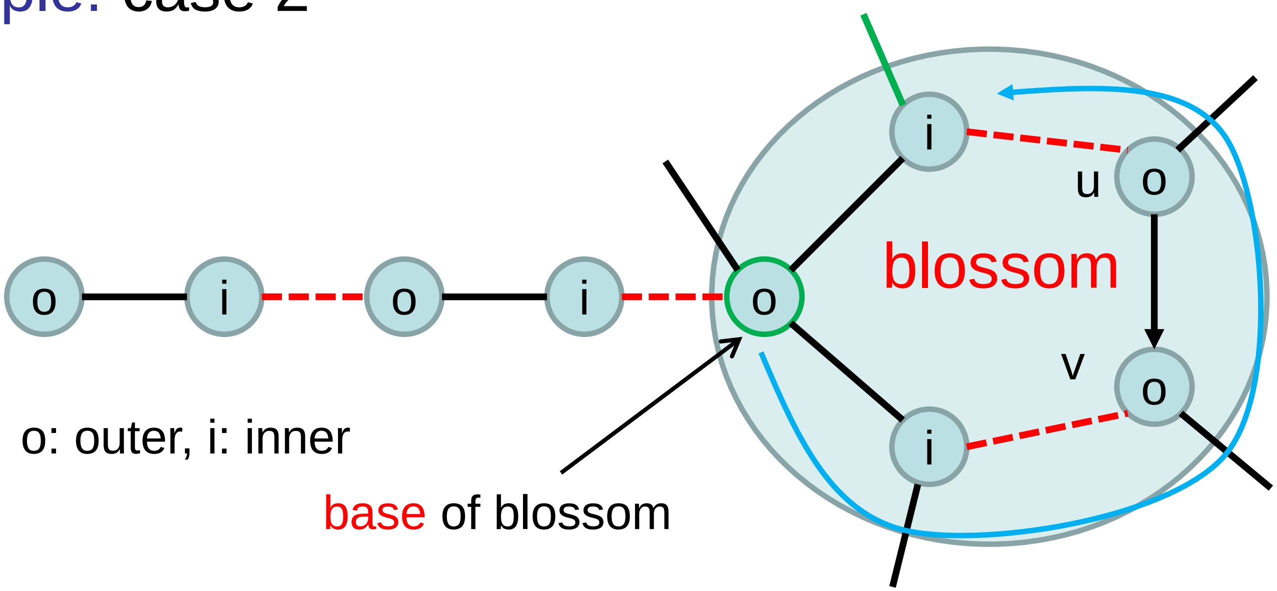
Example: case 1



# Matching in arbitrary Graphs

Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge.

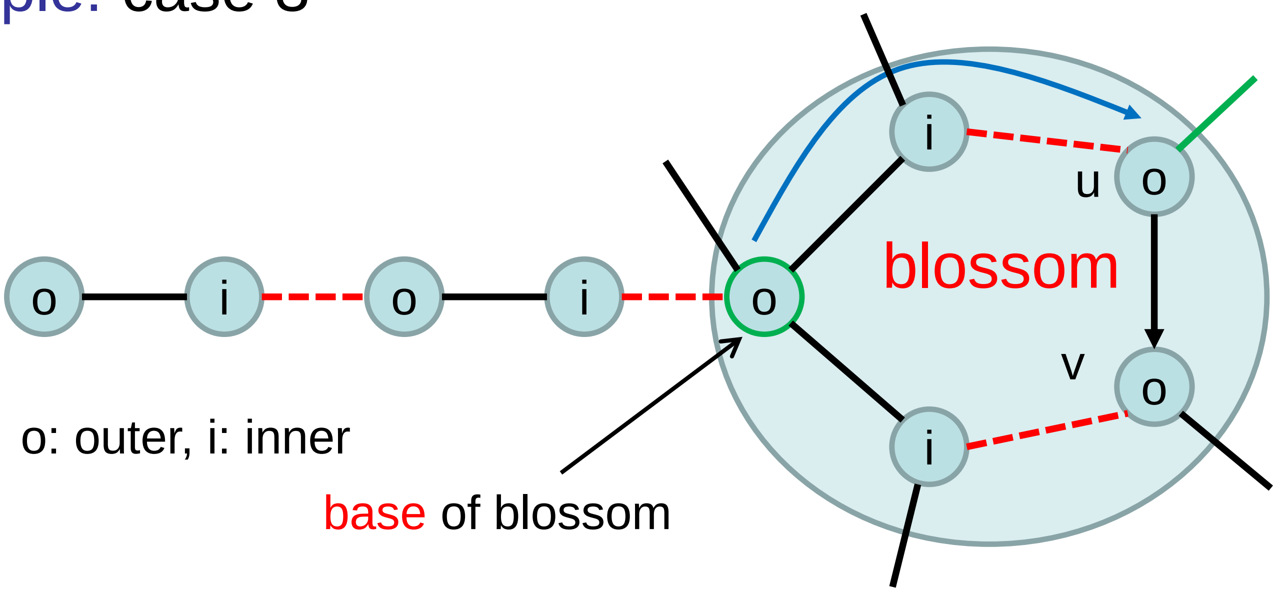
Example: case 2



# Matching in arbitrary Graphs

Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge.

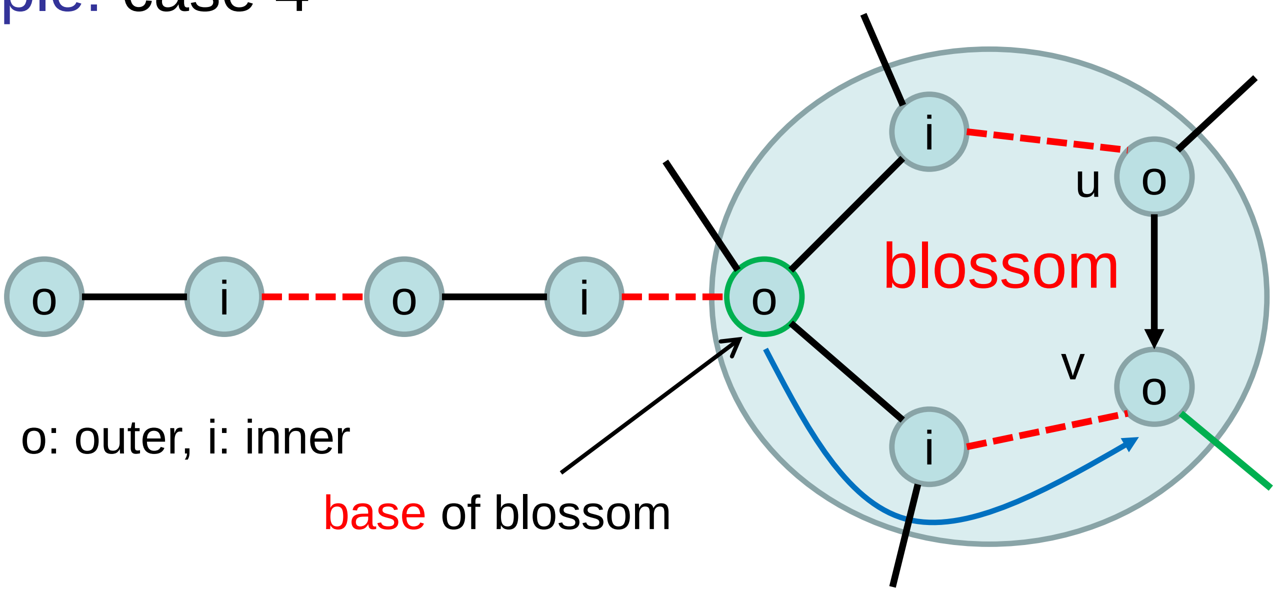
Example: case 3



# Matching in arbitrary Graphs

Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge.

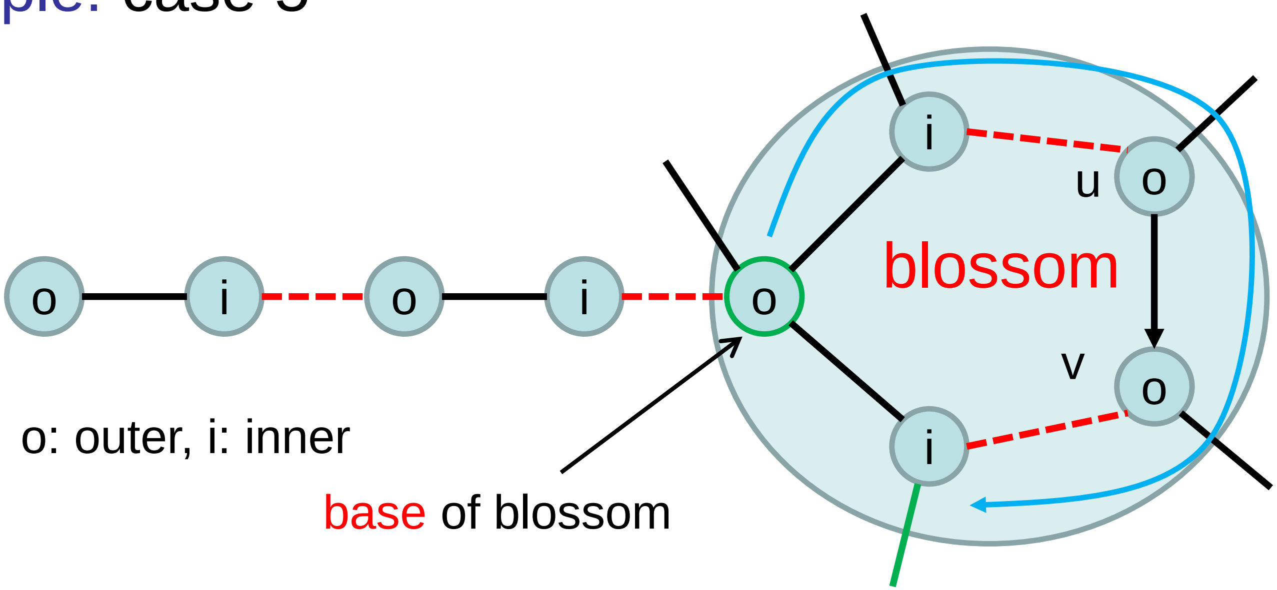
Example: case 4



# Matching in arbitrary Graphs

Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge.

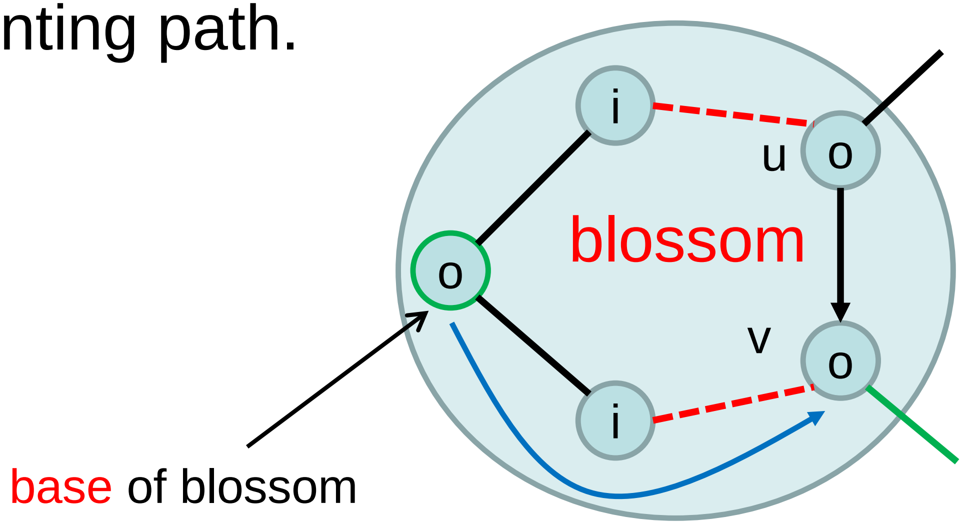
Example: case 5



# Matching in arbitrary Graphs

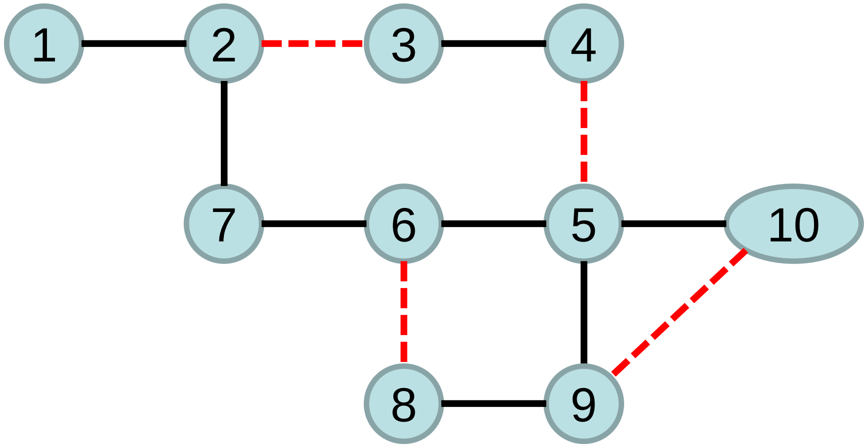
Invariant: for each contracted node, there is an internal alternating path from its base to any of its edges, starting with a non-matched and ending with a matched edge.

The base of a blossom can also be the starting point of an augmenting path.

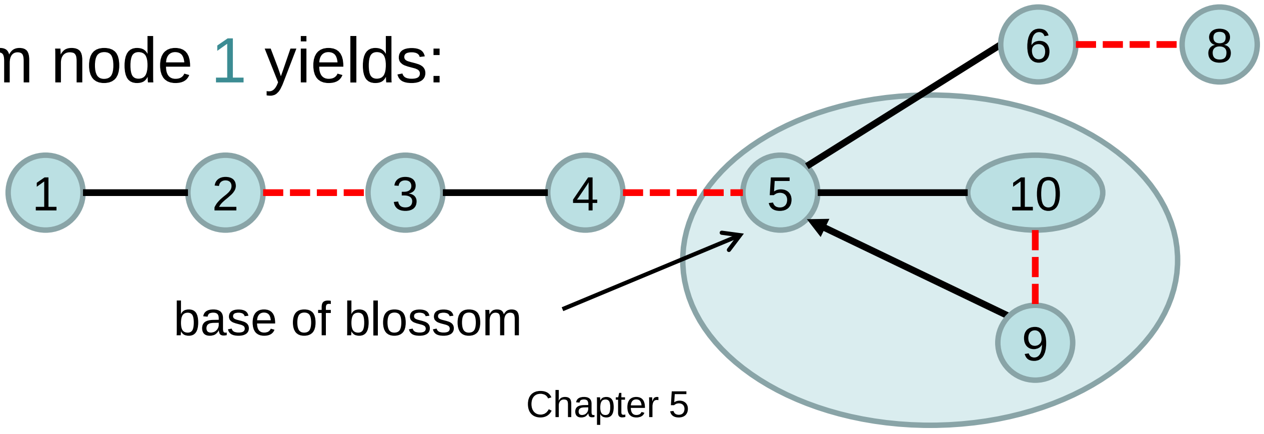


# Matching in arbitrary Graphs

Example:

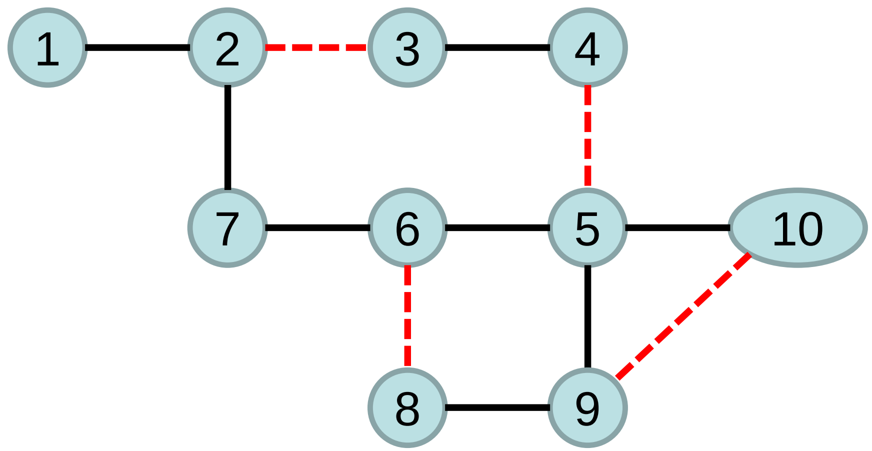



BFS from node 1 yields:

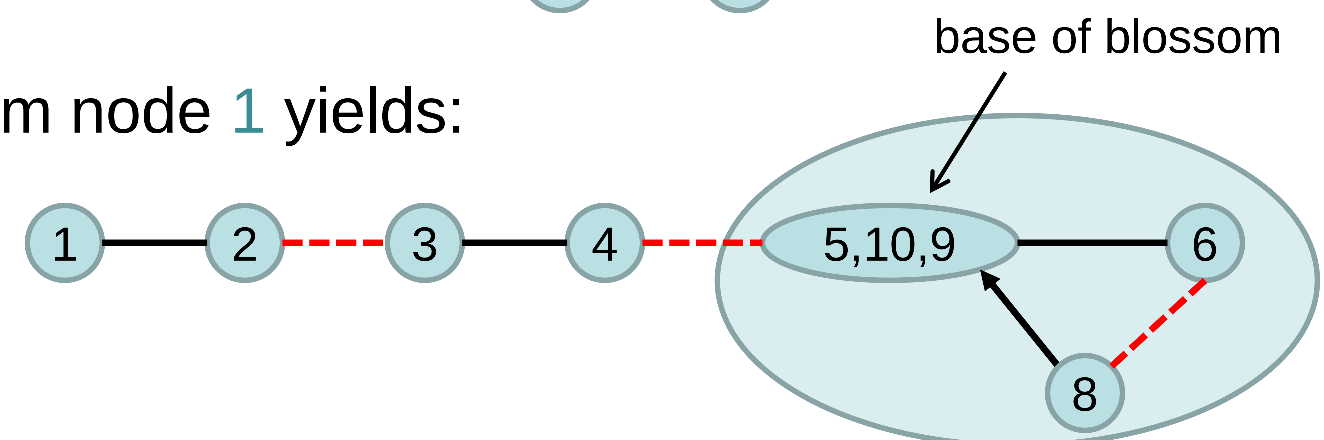


# Matching in arbitrary Graphs

Example:

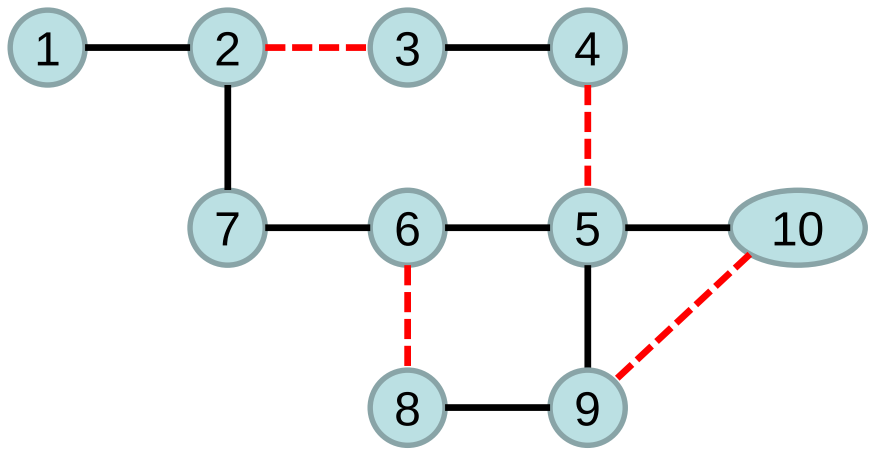



BFS from node 1 yields:

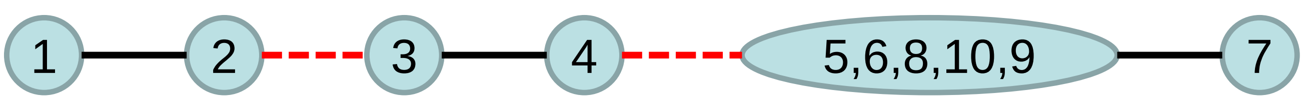


# Matching in arbitrary Graphs

Example:

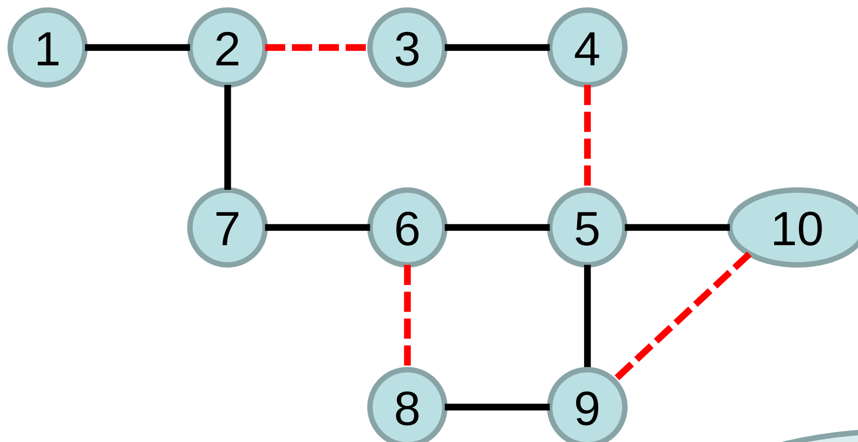



BFS from node 1 yields:

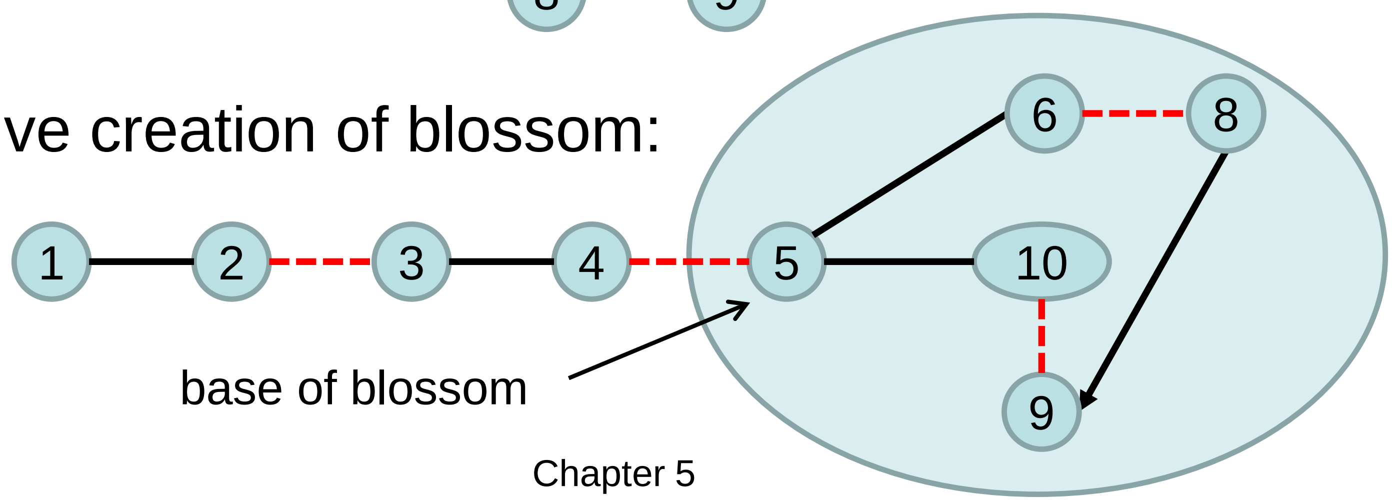


# Matching in arbitrary Graphs

Example:

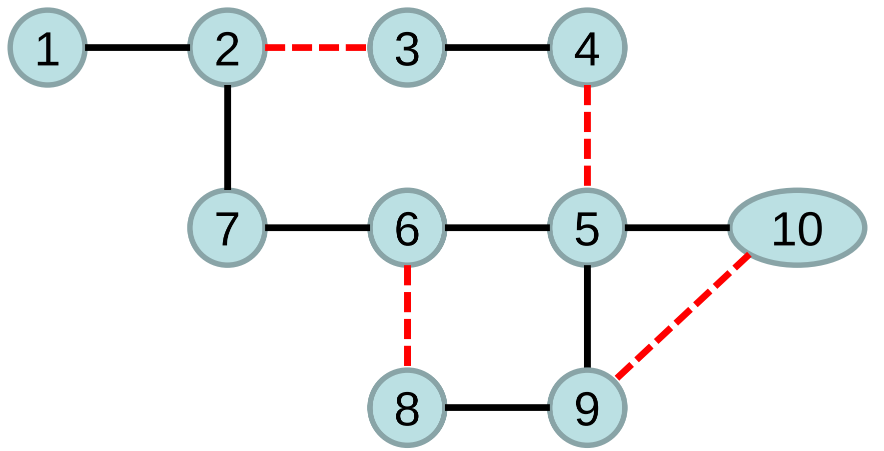



Alternative creation of blossom:

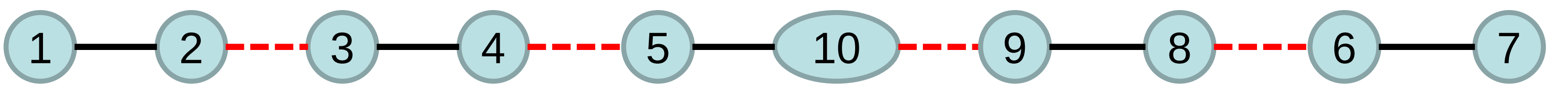


# Matching in arbitrary Graphs

Example:

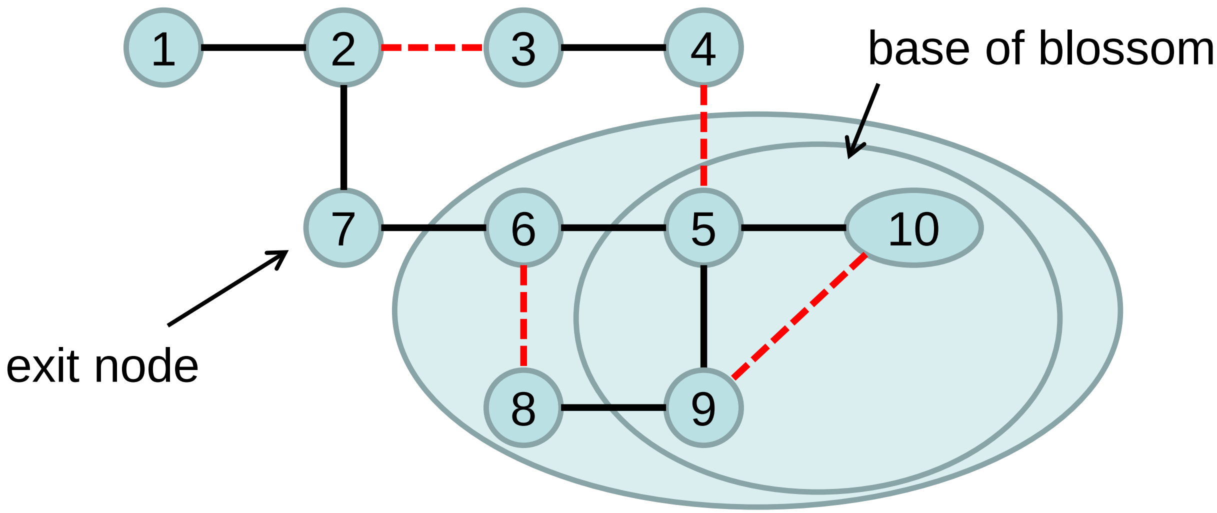



Unshrinking the nodes results in the following augm. path:



# Matching in arbitrary Graphs

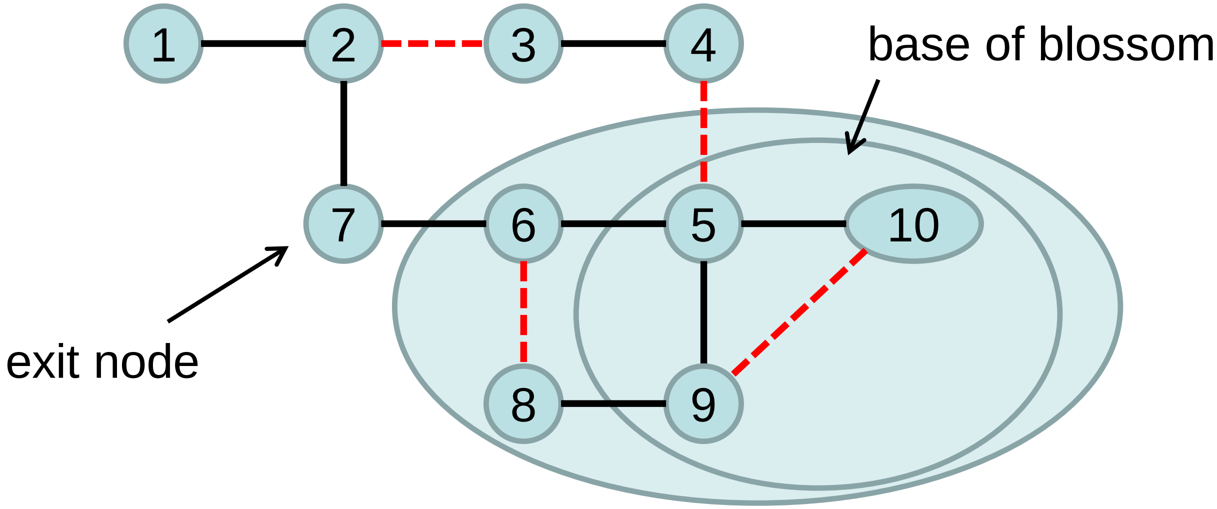
Unshrinking:




Problem: unshrink the blossoms to find augmenting path.

# Matching in arbitrary Graphs

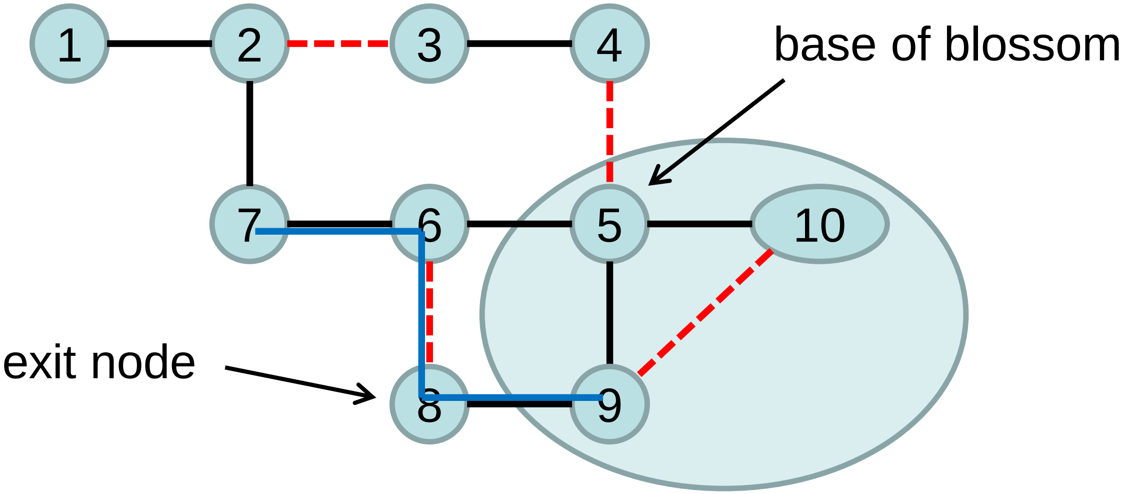
Unshrinking:




Solution: recursively find an augmenting path from base of blossom to the exit node.

# Matching in arbitrary Graphs

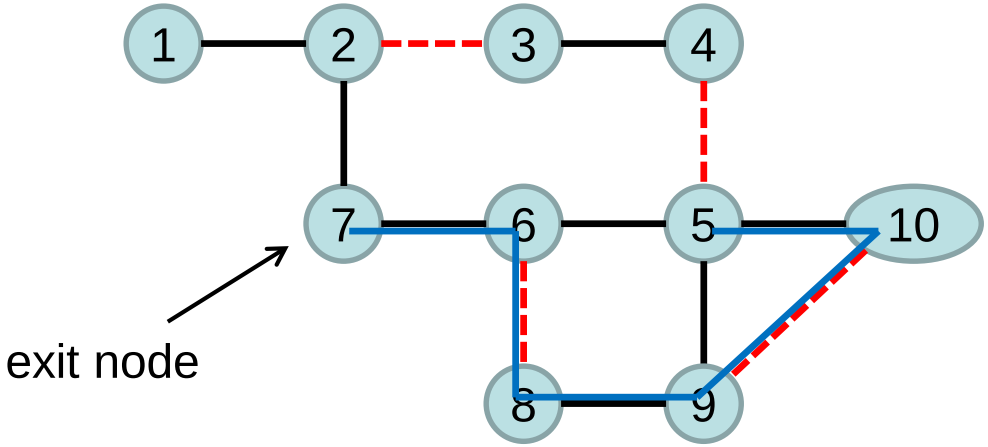
Unshrinking:




Solution: recursively find an augmenting path from base of blossom to the exit node.

# Matching in arbitrary Graphs

Unshrinking:




Solution: recursively find an augmenting path from base of blossom to the exit node.

Easy because only blossom edges need to be considered!

# Matching in arbitrary Graphs

Edmond´s algorithm:

$M := \emptyset$

repeat $\exists$augmenting $P$ w.r.t. $M$ do
  search for an augmenting path $P$ w.r.t. $M$ using Edmond`s blossom-based alternating BFS algorithm
  $M := M \ominus P$

output $M$

Runtime:

- The while-loop is executed at most $n$ times.
- The blossom-based alternating BFS algorithm can be implemented in $O(n+m)$ time.

Therefore, a runtime of $O(n \cdot (n+m))$ is possible.

# Matching in arbitrary Graphs

The Hopcroft-Karp approach can also be used for arbitrary graphs:

$M:=\emptyset$

while $\exists$augmenting path w.r.t. $M$ do

- $l:=$length of shortest augmenting path w.r.t. $M$
- determine w.r.t. „$\subseteq$" maximal set of node-disjoint augmenting paths $Q_1,\ldots,Q_k$ w.r.t. $M$ that have length $l$
- $M:=M\ominus Q_1\ominus \ldots \ominus Q_k$

- A runtime of $O(m)$ is possible per round, resulting in an overall runtime of $O(m\cdot\sqrt{n})$ .
- Details can be found, for example, in:
  Paul Peterson and Michael Loui. The general maximum matching algorithm of Micali and Vazirani. Algorithmica 3:511-533, 1988.

# Next Chapter

Network flow…