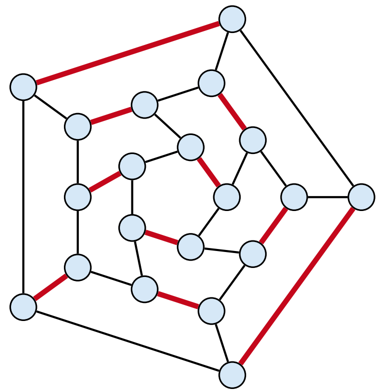
# Part V

## Matchings



## Matching

- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in $M$.
- Maximum Matching: find a matching of maximum cardinality



## 16 Bipartite Matching via Flows

**Which flow algorithm to use?**

- Generic augmenting path: $\mathcal{O}(m \operatorname{val}(f^*)) = \mathcal{O}(mn)$.
- Capacity scaling: $\mathcal{O}(m^2 \log C) = \mathcal{O}(m^2)$.
- Shortest augmenting path: $\mathcal{O}(mn^2)$.

For unit capacity simple graphs shortest augmenting path can be implemented in time $\mathcal{O}(m\sqrt{n})$.



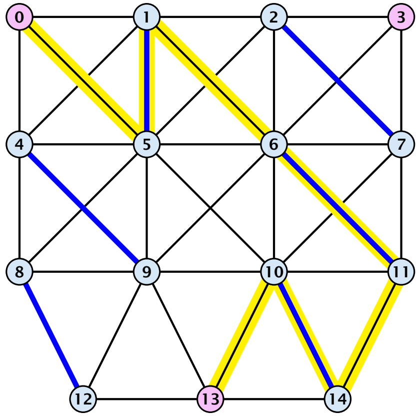
## 17 Augmenting Paths for Matchings

**Definitions.**

- Given a matching $M$ in a graph $G$, a vertex that is not incident to any edge of $M$ is called a free vertex w. r. .t. $M$.
- For a matching $M$ a path $P$ in $G$ is called an alternating path if edges in $M$ alternate with edges not in $M$.
- An alternating path is called an augmenting path for matching $M$ if it ends at distinct free vertices.

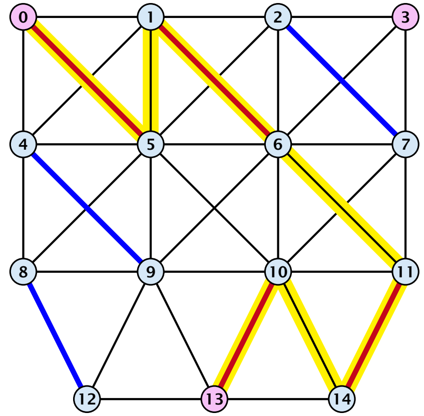
**Theorem 1**
*A matching $M$ is a maximum matching if and only if there is no augmenting path w. r. t. $M$.*

## Augmenting Paths in Action






## Augmenting Paths in Action






## 17 Augmenting Paths for Matchings

**Proof.**

- ⇒ If $M$ is maximum there is no augmenting path $P$, because we could switch matching and non-matching edges along $P$. This gives matching $M' = M \oplus P$ with larger cardinality.
- ⇐ Suppose there is a matching $M'$ with larger cardinality. Consider the graph $H$ with edge-set $M' \oplus M$ (i.e., only edges that are in either $M$ or $M'$ but not in both).

  Each vertex can be incident to at most two edges (one from $M$ and one from $M'$). Hence, the connected components are alternating cycles or alternating path.

  As $|M'| > |M|$ there is one connected component that is a path $P$ for which both endpoints are incident to edges from $M'$. $P$ is an alternating path.



## 17 Augmenting Paths for Matchings

**Algorithmic idea:**
As long as you find an augmenting path augment your matching using this path. When you arrive at a matching for which no augmenting path exists you have a maximum matching.

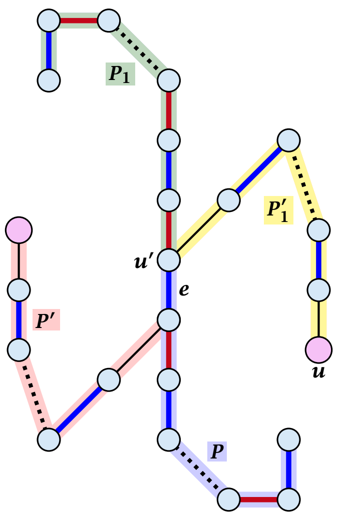
**Theorem 2**
*Let $G$ be a graph, $M$ a matching in $G$, and let $u$ be a free vertex w.r.t. $M$. Further let $P$ denote an augmenting path w.r.t. $M$ and let $M' = M \oplus P$ denote the matching resulting from augmenting $M$ with $P$. If there was no augmenting path starting at $u$ in $M$ then there is no augmenting path starting at $u$ in $M'$.*

The above theorem allows for an easier implementation of an augmenting path algorithm. Once we checked for augmenting paths starting from $u$ we don't have to check for such paths in future rounds.

# 17 Augmenting Paths for Matchings

**Proof**

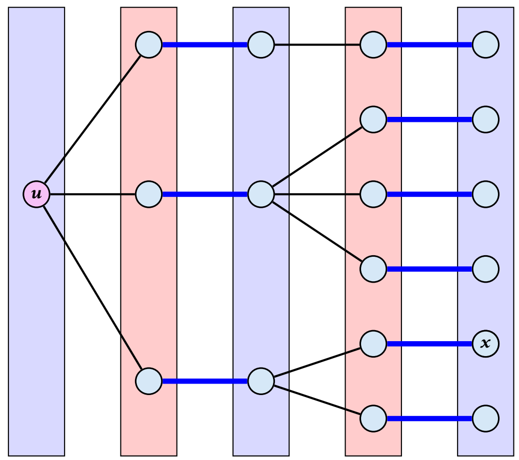
- Assume there is an augmenting path $P'$ w.r.t. $M'$ starting at $u$.
- If $P'$ and $P$ are node-disjoint, $P'$ is also augmenting path w.r.t. $M$ ($\lightning$).
- Let $u'$ be the first node on $P'$ that is in $P$, and let $e$ be the matching edge from $M'$ incident to $u'$.
- $u'$ splits $P$ into two parts one of which does not contain $e$. Call this part $P_1$. Denote the sub-path of $P'$ from $u$ to $u'$ with $P_1'$.
- $P_1 \circ P_1'$ is augmenting path in $M$ ($\lightning$).






# How to find an augmenting path?

**Construct an alternating tree.**




even nodes
odd nodes

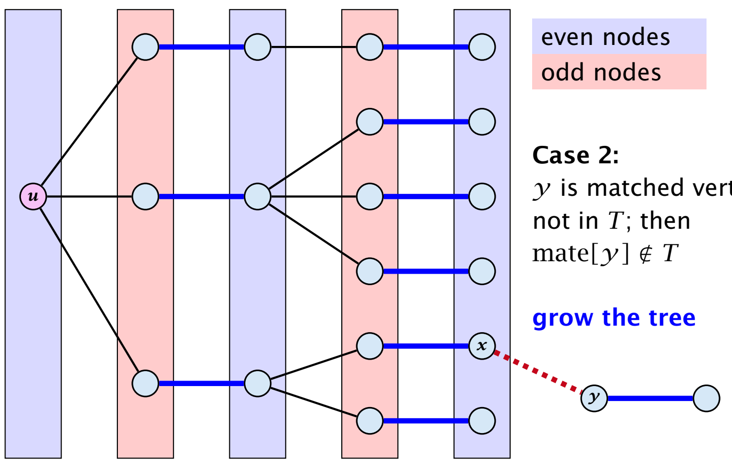
**Case 1:**
$y$ is free vertex not contained in $T$

**you found alternating path**



# How to find an augmenting path?

**Construct an alternating tree.**




even nodes
odd nodes

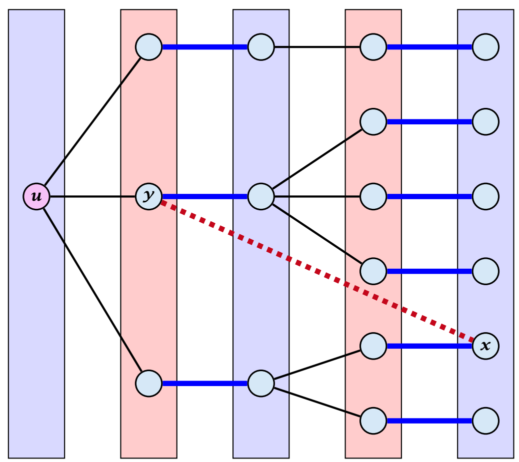
**Case 2:**
$y$ is matched vertex not in $T$; then $\text{mate}[y] \notin T$

**grow the tree**



# How to find an augmenting path?

**Construct an alternating tree.**



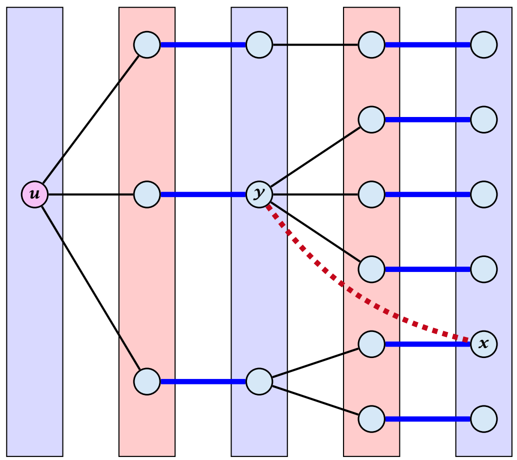

even nodes
odd nodes

**Case 3:**
$y$ is already contained in $T$ as an odd vertex

**ignore successor $y$**

## How to find an augmenting path?

**Construct an alternating tree.**




even nodes
odd nodes

**Case 4:**
$y$ is already contained in $T$ as an even vertex

**can't ignore $y$**

**does not happen in bipartite graphs**



**Algorithm 23** BiMatch$(G, match)$

**for** $x \in V$ **do** $mate[x] \leftarrow 0$;
$r \leftarrow 0$; $free \leftarrow n$;
**while** $free \geq 1$ **and** $r < n$ **do**
  $r \leftarrow r + 1$
  **if** $mate[r] = 0$ **then**
    **for** $i = 1$ **to** $n$ **do** $parent[i'] \leftarrow 0$
    $Q \leftarrow \emptyset$; $Q.\text{append}(r)$; $aug \leftarrow \text{false}$;
    **while** $aug = \text{false}$ **and** $Q \neq \emptyset$ **do**
      $x \leftarrow Q.\text{dequeue}()$;
      **for** $y \in A_x$ **do**
        **if** $mate[y] = 0$ **then**
          $\text{augm}(mate, parent, y)$;
          $aug \leftarrow \text{true}$;
          $free \leftarrow free - 1$;
        **else**
          **if** $parent[y] = 0$ **then**
            $parent[y] \leftarrow x$;
            $Q.\text{enqueue}(mate[y])$;

graph $G = (S \cup S', E)$

$S = \{1, \ldots, n\}$

$S' = \{1', \ldots, n'\}$

The lecture version of the slides contains a step-by-step explanation of the algorithm.

## 18 Weighted Bipartite Matching

**Weighted Bipartite Matching/Assignment**

- Input: undirected, bipartite graph $G = L \cup R, E$.
- an edge $e = (\ell, r)$ has weight $w_e \geq 0$
- find a matching of maximum weight, where the weight of a matching is the sum of the weights of its edges

**Simplifying Assumptions (wlog [why?]):**

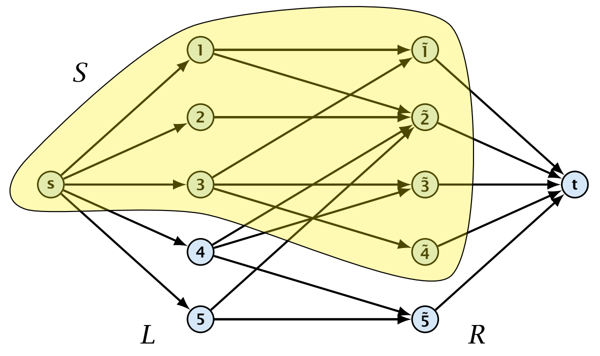
- assume that $|L| = |R| = n$
- assume that there is an edge between every pair of nodes $(\ell, r) \in V \times V$
- can assume goal is to construct maximum weight perfect matching



## Weighted Bipartite Matching

**Theorem 3 (Halls Theorem)**
*A bipartite graph $G = (L \cup R, E)$ has a perfect matching if and only if for all sets $S \subseteq L$, $|\Gamma(S)| \geq |S|$, where $\Gamma(S)$ denotes the set of nodes in $R$ that have a neighbour in $S$.*

# 18 Weighted Bipartite Matching




# Halls Theorem

**Proof:**

- $\Leftarrow$ Of course, the condition is necessary as otherwise not all nodes in $S$ could be matched to different neigbhours.
- $\Rightarrow$ For the other direction we need to argue that the minimum cut in the graph $G'$ is at least $|L|$.
  - Let $S$ denote a minimum cut and let $L_S \stackrel{\text{def}}{=} L \cap S$ and $R_S \stackrel{\text{def}}{=} R \cap S$ denote the portion of $S$ inside $L$ and $R$, respectively.
  - Clearly, all neighbours of nodes in $L_S$ have to be in $S$, as otherwise we would cut an edge of infinite capacity.
  - This gives $R_S \geq |\Gamma(L_S)|$.
  - The size of the cut is $|L| - |L_S| + |R_S|$.
  - Using the fact that $|\Gamma(L_S)| \geq L_S$ gives that this is at least $|L|$.



# Algorithm Outline

**Idea:**
We introduce a node weighting $\vec{x}$. Let for a node $v \in V$, $x_v \in \mathbb{R}$ denote the weight of node $v$.

- Suppose that the node weights dominate the edge-weights in the following sense:

$$x_u + x_v \geq w_e \text{ for every edge } e = (u, v).$$

- Let $H(\vec{x})$ denote the subgraph of $G$ that only contains edges that are tight w.r.t. the node weighting $\vec{x}$, i.e. edges $e = (u, v)$ for which $w_e = x_u + x_v$.
- Try to compute a perfect matching in the subgraph $H(\vec{x})$. If you are successful you found an optimal matching.



# Algorithm Outline

**Reason:**

- The weight of your matching $M^*$ is

$$\sum_{(u,v)\in M^*} w_{(u,v)} = \sum_{(u,v)\in M^*} (x_u + x_v) = \sum_v x_v \enspace .$$

- Any other perfect matching $M$ (in $G$, not necessarily in $H(\vec{x})$) has

$$\sum_{(u,v)\in M} w_{(u,v)} \leq \sum_{(u,v)\in M} (x_u + x_v) = \sum_v x_v \enspace .$$

## Algorithm Outline

**What if you don't find a perfect matching?**

Then, Halls theorem guarantees you that there is a set $S \subseteq L$, with $|\Gamma(S)| < |S|$, where $\Gamma$ denotes the neighbourhood w.r.t. the subgraph $H(\vec{x})$.

**Idea:** reweight such that:

- the total weight assigned to nodes decreases
- the weight function still dominates the edge-weights

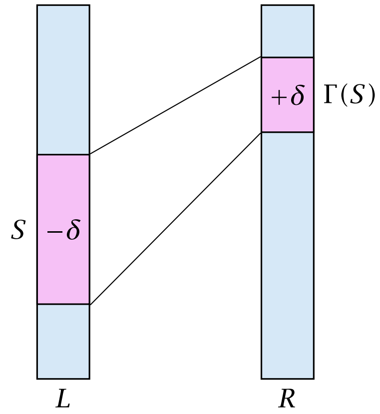
If we can do this we have an algorithm that terminates with an optimal solution (we analyze the running time later).



## Changing Node Weights

Increase node-weights in $\Gamma(S)$ by $+\delta$, and decrease the node-weights in $S$ by $-\delta$.

- Total node-weight decreases.
- Only edges from $S$ to $R - \Gamma(S)$ decrease in their weight.
- Since, none of these edges is tight (otw. the edge would be contained in $H(\vec{x})$, and hence would go between $S$ and $\Gamma(S)$) we can do this decrement for small enough $\delta > 0$ until a new edge gets tight.

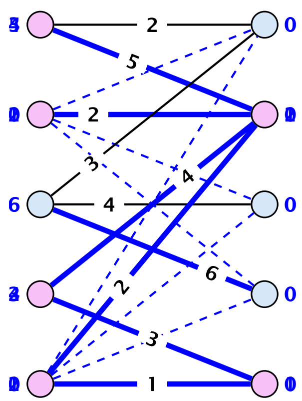





## Weighted Bipartite Matching

Edges not drawn have weight $0$.

$\delta = 1$ $\delta = 1$






## Analysis

**How many iterations do we need?**

- One reweighting step increases the number of edges out of $S$ by at least one.
- Assume that we have a maximum matching that saturates the set $\Gamma(S)$, in the sense that every node in $\Gamma(S)$ is matched to a node in $S$ (we will show that we can always find $S$ and a matching such that this holds).
- This matching is still contained in the new graph, because all its edges either go between $\Gamma(S)$ and $S$ or between $L - S$ and $R - \Gamma(S)$.
- Hence, reweighting does not decrease the size of a maximum matching in the tight sub-graph.

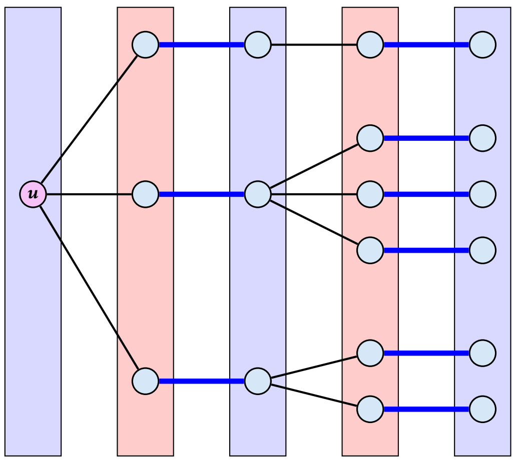
## Analysis

- We will show that after at most $n$ reweighting steps the size of the maximum matching can be increased by finding an augmenting path.
- This gives a polynomial running time.



## How to find an augmenting path?

**Construct an alternating tree.**






## Analysis

**How do we find $S$?**

- Start on the left and compute an alternating tree, starting at any free node $u$.
- If this construction stops, there is no perfect matching in the tight subgraph (because for a perfect matching we need to find an augmenting path starting at $u$).
- The set of even vertices is on the left and the set of odd vertices is on the right and contains all neighbours of even nodes.
- All odd vertices are matched to even vertices. Furthermore, the even vertices additionally contain the free vertex $u$. Hence, $|V_{\text{odd}}| = |\Gamma(V_{\text{even}})| < |V_{\text{even}}|$, and all odd vertices are saturated in the current matching.

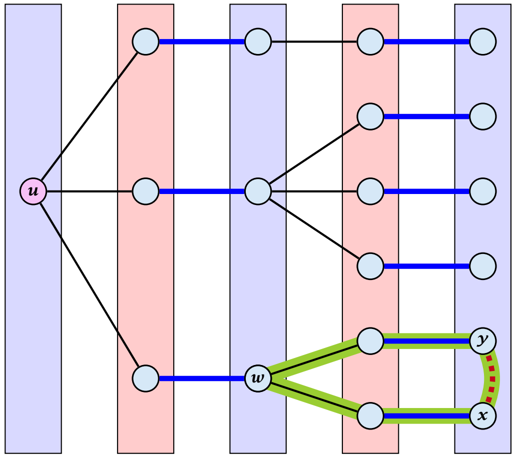


## Analysis

- The current matching does not have any edges from $V_{\text{odd}}$ to $L \setminus V_{\text{even}}$ (edges that may possibly be deleted by changing weights).
- After changing weights, there is at least one more edge connecting $V_{\text{even}}$ to a node outside of $V_{\text{odd}}$. After at most $n$ reweights we can do an augmentation.
- A reweighting can be trivially performed in time $\mathcal{O}(n^2)$ (keeping track of the tight edges).
- An augmentation takes at most $\mathcal{O}(n)$ time.
- In total we obtain a running time of $\mathcal{O}(n^4)$.
- A more careful implementation of the algorithm obtains a running time of $\mathcal{O}(n^3)$.

# How to find an augmenting path?

**Construct an alternating tree.**




even nodes
odd nodes

**Case 4:**
$y$ is already contained in $T$ as an even vertex

**can't ignore $y$**

The cycle $w \leftrightarrow y - x \leftrightarrow w$ is called a blossom.
$w$ is called the base of the blossom (even node!!!).
The path $u$-$w$ is called the stem of the blossom.



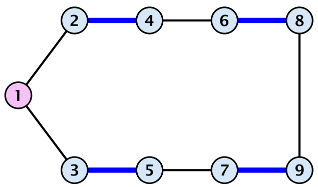
# Flowers and Blossoms

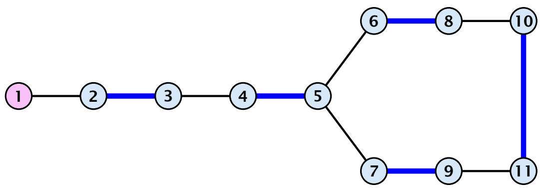
**Definition 4**
A flower in a graph $G = (V, E)$ w.r.t. a matching $M$ and a (free) root node $r$, is a subgraph with two components:

- A stem is an even length alternating path that starts at the root node $r$ and terminates at some node $w$. We permit the possibility that $r = w$ (empty stem).
- A blossom is an odd length alternating cycle that starts and terminates at the terminal node $w$ of a stem and has no other node in common with the stem. $w$ is called the base of the blossom.



# Flowers and Blossoms






# Flowers and Blossoms

**Properties:**

1. A stem spans $2\ell + 1$ nodes and contains $\ell$ matched edges for some integer $\ell \geq 0$.
2. A blossom spans $2k + 1$ nodes and contains $k$ matched edges for some integer $k \geq 1$. The matched edges match all nodes of the blossom except the base.
3. The base of a blossom is an even node (if the stem is part of an alternating tree starting at $r$).

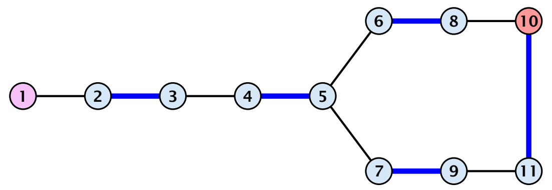
## Flowers and Blossoms

**Properties:**

4. Every node $x$ in the blossom (except its base) is reachable from the root (or from the base of the blossom) through two distinct alternating paths; one with even and one with odd length.
5. The even alternating path to $x$ terminates with a matched edge and the odd path with an unmatched edge.



## Flowers and Blossoms






## Shrinking Blossoms

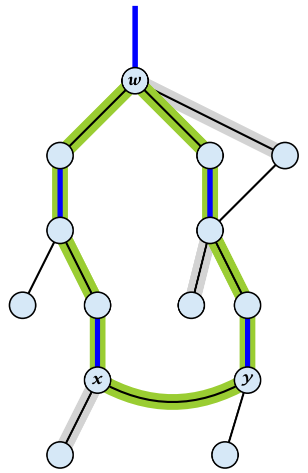
When during the alternating tree construction we discover a blossom $B$ we replace the graph $G$ by $G' = G/B$, which is obtained from $G$ by contracting the blossom $B$.

- Delete all vertices in $B$ (and its incident edges) from $G$.
- Add a new (pseudo-)vertex $b$. The new vertex $b$ is connected to all vertices in $V \setminus B$ that had at least one edge to a vertex from $B$.



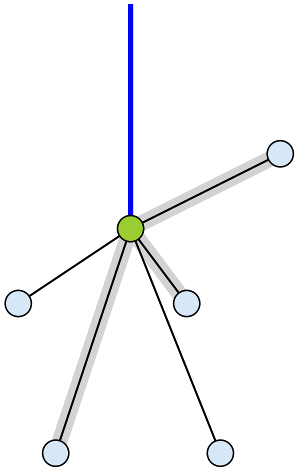
## Shrinking Blossoms

- Edges of $T$ that connect a node $u$ not in $B$ to a node in $B$ become tree edges in $T'$ connecting $u$ to $b$.
- Matching edges (there is at most one) that connect a node $u$ not in $B$ to a node in $B$ become matching edges in $M'$.
- Nodes that are connected in $G$ to at least one node in $B$ become connected to $b$ in $G'$.



## Shrinking Blossoms

- Edges of $T$ that connect a node $u$ not in $B$ to a node in $B$ become tree edges in $T'$ connecting $u$ to $b$.
- Matching edges (there is at most one) that connect a node $u$ not in $B$ to a node in $B$ become matching edges in $M'$.
- Nodes that are connected in $G$ to at least one node in $B$ become connected to $b$ in $G'$.





## Example: Blossom Algorithm

Animation of Blossom Shrinking algorithm is only available in the lecture version of the slides.



## Correctness

Assume that in $G$ we have a flower w.r.t. matching $M$. Let $r$ be the root, $B$ the blossom, and $w$ the base. Let graph $G' = G/B$ with pseudonode $b$. Let $M'$ be the matching in the contracted graph.

**Lemma 5**

*If $G'$ contains an augmenting path $P'$ starting at $r$ (or the pseudo-node containing $r$) w.r.t. the matching $M'$ then $G$ contains an augmenting path starting at $r$ w.r.t. matching $M$.*



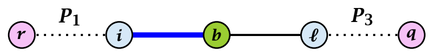
## Correctness

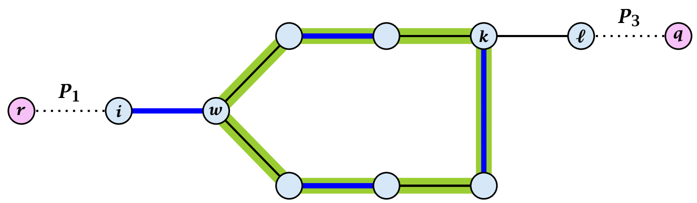
**Proof.**

If $P'$ does not contain $b$ it is also an augmenting path in $G$.

**Case 1: non-empty stem**

- Next suppose that the stem is non-empty.



## Correctness

- After the expansion $\ell$ must be incident to some node in the blossom. Let this node be $k$.
- If $k \neq w$ there is an alternating path $P_2$ from $w$ to $k$ that ends in a matching edge.
- $P_1 \circ (i, w) \circ P_2 \circ (k, \ell) \circ P_3$ is an alternating path.
- If $k = w$ then $P_1 \circ (i, w) \circ (w, \ell) \circ P_3$ is an alternating path.

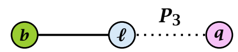


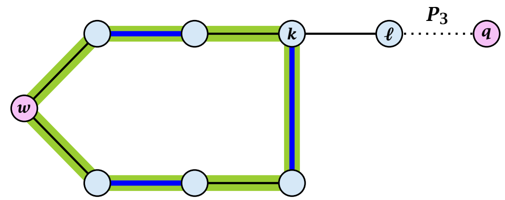
## Correctness

**Proof.**

**Case 2: empty stem**

- If the stem is empty then after expanding the blossom, $w = r$.




- The path $r \circ P_2 \circ (k, \ell) \circ P_3$ is an alternating path.



## Correctness

**Lemma 6**

*If $G$ contains an augmenting path $P$ from $r$ to $q$ w.r.t. matching $M$ then $G'$ contains an augmenting path from $r$ (or the pseudo-node containing $r$) to $q$ w.r.t. $M'$.*



## Correctness

**Proof.**

- If $P$ does not contain a node from $B$ there is nothing to prove.
- We can assume that $r$ and $q$ are the only free nodes in $G$.

**Case 1: empty stem**

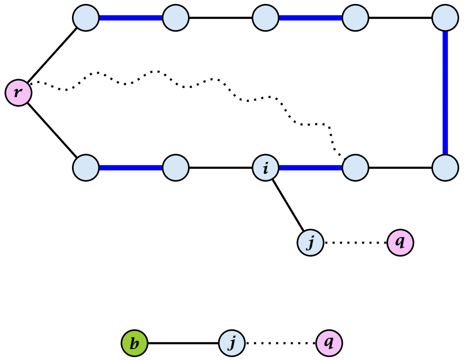
Let $i$ be the last node on the path $P$ that is part of the blossom.

$P$ is of the form $P_1 \circ (i, j) \circ P_2$, for some node $j$ and $(i, j)$ is unmatched.

$(b, j) \circ P_2$ is an augmenting path in the contracted network.

## Correctness

**Illustration for Case 1:**






## Correctness

**Case 2: non-empty stem**

Let $P_3$ be alternating path from $r$ to $w$; this exists because $r$ and $w$ are root and base of a blossom. Define $M_+ = M \oplus P_3$.

In $M_+$, $r$ is matched and $w$ is unmatched.

$G$ must contain an augmenting path w.r.t. matching $M_+$, since $M$ and $M_+$ have same cardinality.

This path must go between $w$ and $q$ as these are the only unmatched vertices w.r.t. $M_+$.

For $M'_+$ the blossom has an empty stem. Case 1 applies.

$G'$ has an augmenting path w.r.t. $M'_+$. It must also have an augmenting path w.r.t. $M'$, as both matchings have the same cardinality.

This path must go between $r$ and $q$.



**Algorithm 24** search$(r, \mathit{found})$

set $\bar{A}(i) \leftarrow A(i)$ for all nodes $i$
$\mathit{found} \leftarrow$ false
unlabel all nodes;
give an even label to $r$ and initialize $\mathit{list} \leftarrow \{r\}$
**while** $\mathit{list} \neq \emptyset$ **do**
  delete a node $i$ from $\mathit{list}$
  examine$(i, \mathit{found})$
  **if** $\mathit{found} = \text{true}$ **then return**

Search for an augmenting path starting at $r$.

The lecture version of the slides has a step by step explanation.

**Algorithm 25** examine$(i, \mathit{found})$

**for** all $j \in \bar{A}(i)$ **do**
  **if** $j$ is even **then** contract$(i, j)$ and **return**
  **if** $j$ is unmatched **then**
    $q \leftarrow j$;
    $\text{pred}(q) \leftarrow i$;
    $\mathit{found} \leftarrow \text{true}$;
    **return**
  **if** $j$ is matched and unlabeled **then**
    $\text{pred}(j) \leftarrow i$;
    $\text{pred}(\text{mate}(j)) \leftarrow j$;
    add $\text{mate}(j)$ to $\mathit{list}$

Examine the neighbours of a node $i$

The lecture version of the slides has a step by step explanation.

**Algorithm 26** contract$(i, j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

Contract blossom identified by nodes $i$ and $j$



**Algorithm 26** contract$(i, j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

Get all nodes of the blossom.

Time: $\mathcal{O}(m)$



**Algorithm 26** contract$(i, j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

Identify all neighbours of $b$.

Time: $\mathcal{O}(m)$ (how?)



**Algorithm 26** contract$(i, j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

$b$ will be an even node, and it has unexamined neighbours.

**Algorithm 26** contract$(i,j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

Every node that was adjacent to a node in $B$ is now adjacent to $b$



**Algorithm 26** contract$(i,j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

Only for making a blossom expansion easier.



**Algorithm 26** contract$(i,j)$

trace pred-indices of $i$ and $j$ to identify a blossom $B$
create new node $b$ and set $\bar{A}(b) \leftarrow \cup_{x\in B}\bar{A}(x)$
label $b$ even and add to *list*
update $\bar{A}(j) \leftarrow \bar{A}(j) \cup \{b\}$ for each $j \in \bar{A}(b)$
form a circular double linked list of nodes in $B$
delete nodes in $B$ from the graph

Only delete links from nodes not in $B$ to $B$.

When expanding the blossom again we can recreate these links in time $\mathcal{O}(m)$.



## Analysis

- A contraction operation can be performed in time $\mathcal{O}(m)$. Note, that any graph created will have at most $m$ edges.
- The time between two contraction-operation is basically a BFS/DFS on a graph. Hence takes time $\mathcal{O}(m)$.
- There are at most $n$ contractions as each contraction reduces the number of vertices.
- The expansion can trivially be done in the same time as needed for all contractions.
- An augmentation requires time $\mathcal{O}(n)$. There are at most $n$ of them.
- In total the running time is at most

$$n \cdot (\mathcal{O}(mn) + \mathcal{O}(n)) = \mathcal{O}(mn^2)\ .$$

## Example: Blossom Algorithm

Animation of Blossom Shrinking algorithm is only available in the lecture version of the slides.



## A Fast Matching Algorithm

**Algorithm 27** Bimatch-Hopcroft-Karp$(G)$

```
M ← ∅
repeat
    let 𝒫 = {P_1, ..., P_k} be maximal set of
    vertex-disjoint, shortest augmenting path w.r.t. M.
    M ← M ⊕ (P_1 ∪ ··· ∪ P_k)
until 𝒫 = ∅
return M
```

We call one iteration of the repeat-loop a phase of the algorithm.



## Analysis Hopcroft-Karp

**Lemma 7**

*Given a matching $M$ and a maximal matching $M^*$ there exist $|M^*| - |M|$ vertex-disjoint augmenting path w.r.t. $M$.*

**Proof:**

- Similar to the proof that a matching is optimal iff it does not contain an augmenting paths.
- Consider the graph $G = (V, M \oplus M^*)$, and mark edges in this graph blue if they are in $M$ and red if they are in $M^*$.
- The connected components of $G$ are cycles and paths.
- The graph contains $k \stackrel{\text{def}}{=} |M^*| - |M|$ more red edges than blue edges.
- Hence, there are at least $k$ components that form a path starting and ending with a blue edge. These are augmenting paths w.r.t. $M$.



## Analysis Hopcroft-Karp

- Let $P_1, \dots, P_k$ be a maximal collection of vertex-disjoint, shortest augmenting paths w.r.t. $M$ (let $\ell = |P_i|$).
- $M' \stackrel{\text{def}}{=} M \oplus (P_1 \cup \cdots \cup P_k) = M \oplus P_1 \oplus \cdots \oplus P_k$.
- Let $P$ be an augmenting path in $M'$.

**Lemma 8**

*The set $A \stackrel{\text{def}}{=} M \oplus (M' \oplus P) = (P_1 \cup \cdots \cup P_k) \oplus P$ contains at least $(k+1)\ell$ edges.*

## Analysis Hopcroft-Karp

**Proof.**

- The set describes exactly the symmetric difference between matchings $M$ and $M' \oplus P$.
- Hence, the set contains at least $k+1$ vertex-disjoint augmenting paths w.r.t. $M$ as $|M'| = |M| + k + 1$.
- Each of these paths is of length at least $\ell$.



## Analysis Hopcroft-Karp

**Lemma 9**

*$P$ is of length at least $\ell + 1$. This shows that the length of a shortest augmenting path increases between two phases of the Hopcroft-Karp algorithm.*

**Proof.**

- If $P$ does not intersect any of the $P_1, \dots, P_k$, this follows from the maximality of the set $\{P_1, \dots, P_k\}$.
- Otherwise, at least one edge from $P$ coincides with an edge from paths $\{P_1, \dots, P_k\}$.
- This edge is not contained in $A$.
- Hence, $|A| \le k\ell + |P| - 1$.
- The lower bound on $|A|$ gives $(k+1)\ell \le |A| \le k\ell + |P| - 1$, and hence $|P| \ge \ell + 1$.



## Analysis Hopcroft-Karp

If the shortest augmenting path w.r.t. a matching $M$ has $\ell$ edges then the cardinality of the maximum matching is of size at most $|M| + \frac{|V|}{\ell+1}$.

**Proof.**

The symmetric difference between $M$ and $M^*$ contains $|M^*| - |M|$ vertex-disjoint augmenting paths. Each of these paths contains at least $\ell + 1$ vertices. Hence, there can be at most $\frac{|V|}{\ell+1}$ of them.



## Analysis Hopcroft-Karp

**Lemma 10**

*The Hopcroft-Karp algorithm requires at most $2\sqrt{|V|}$ phases.*

**Proof.**

- After iteration $\lfloor\sqrt{|V|}\rfloor$ the length of a shortest augmenting path must be at least $\lfloor\sqrt{|V|}\rfloor + 1 \ge \sqrt{|V|}$.
- Hence, there can be at most $|V|/(\sqrt{|V|} + 1) \le \sqrt{|V|}$ additional augmentations.

## Analysis Hopcroft-Karp

**Lemma 11**
*One phase of the Hopcroft-Karp algorithm can be implemented in time $\mathcal{O}(m)$.*

construct a "level graph" $G'$:

- construct Level 0 that includes all free vertices on left side $L$
- construct Level 1 containing all neighbors of Level 0
- construct Level 2 containing matching neighbors of Level 1
- construct Level 3 containing all neighbors of Level 2
- ...
- stop when a level (apart from Level 0) contains a free vertex

can be done in time $\mathcal{O}(m)$ by a modified BFS



## Analysis Hopcroft-Karp

- a shortest augmenting path must go from Level 0 to the last layer constructed
- it can only use edges between layers
- construct a maxima**l** set of vertex disjoint augmenting path connecting the layers
- for this, go forward until you either reach a free vertex or you read a "dead end" $v$
- if you reach a free vertex delete the augmenting path and all incident edges from the graph
- if you reach a dead end backtrack and delete $v$ together with its incident edges



## Analysis Hopcroft-Karp

See lecture versions of the slides.