Lecture 2: Matching Algorithms

Stefano Leonardi leon@dis.uniroma1.it
Piotr Sankowski sankowski@dis.uniroma1.it

Theoretical Computer Science

Lecture Overview

- Matchings: Problem Definition
- Augmenting Paths Algorithm
- Ascending Price Matching Algorithm
- Hopcroft-Karp Algorithm

Problem Definition

Let $G = (V, E)$ be an undirected graph.

Matching in G is a subset of edges $M \subseteq E$ such that at most one edge is incident to each vertex in V.

We say that a vertex is *matched* if it is incident to some edge in M.

Otherwise we say that a vertex is *free*.

Similarly if an edge e is in the matching we say it is *matched*, and otherwise we say it is *free*.

Problem Definition

A matching is said to be *maximum* if it has the maximum size.

A matching is *maximal* if there is no matching that includes it as a strict subset.

A *perfect matching* is a matching which matches all vertices of the graph.

A graph is *bipartite* if

- it's vertex set can be divided into $V = V_1 \cup V_2$, where V_1 are V_2 disjoint,
- all edges in E go between V_1 and V_2.

Augmenting Paths

Given a matching M,

- an *alternating path* is a path in which the edges belong alternatively to the matching and not to the matching,
- an *augmenting path* is an alternating path that starts from and ends on free vertices.

We can easily notice that if there exists an augmenting path p with respect to M, then M is not maximum.

Using the path p we can construct a bigger matching by taking $M = M \oplus p$, i.e., by switching free edges to matched edges and matched edges to free edges.

Augmenting Paths

More importantly the contrary is true as well:

Theorem 1 (Berge) *The matching M is maximum if and only if there is no augmenting path with respect to M.*

We have shown the only if direction on the previous slide.

For the if direction let us assume the contrary, i.e., let us assume that there exists a bigger matching M'.

Augmenting Paths

Consider the graph $G' = (V, M \oplus M')$.

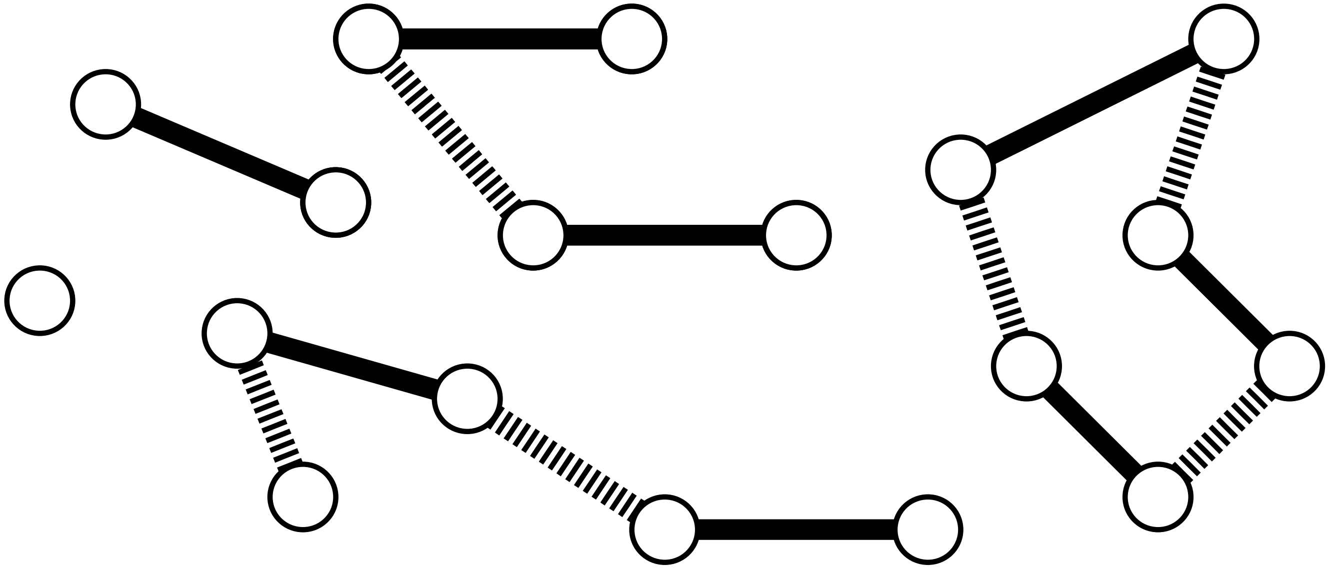
Note that the degree of every vertex in G' is at most 2.

Hence, the graph G' is composed of disjoint paths and cycles.

There are equally many edges from M and M' on each cycle.

Whereas on any path there might be at most one edge more from one of the matchings.

Augmenting Paths



In the graph there are more edges from M' then from M, so there must exist a path containing one more edge from M'. This needs to be an augmenting path.

Augmenting Paths

We will now try to check whether a bipartite graph does not contain an augmenting path or when there is one then we show how to find it.

For a bipartite graph $G = (V_1 \cup V_2, E)$ and for a matching M let us define a directed graph $G_M = (V_1 \cup V_2, E_M)$ as:

$$\begin{aligned} E_M = & \{(v_1, v_2) : v_1 v_2 \in E, v_1 \in V_1, v_2 \in V_2\} \\ & \cup \{(v_2, v_1) : v_1 v_2 \in M, v_1 \in V_1, v_2 \in V_2\}. \end{aligned}$$

Augmenting Paths

FIND-AUGMENTING-PATH$(G = (V_1 \cup V_2, E), M)$

- $V_1' =$ a set of free vertices in V_1
- $V_2' =$ a set of free vertices in V_2
- construct the directed graph $G_M = (V_1 \cup V_2, E_M)$
- find a simple path p from V_1' to V_2' in G_M
- if p does not exists *then*
  - return NIL *(no augmenting paths)*
- else
  - return p *(p is an augmenting path in G)*

Augmenting Paths

Lemma 1 *The algorithm* `FIND-AUGMENTING-PATH` *finds a path p if and only if in G there exists an augmenting path with respect to M. Moreover, the returned path p is an augmenting path.*

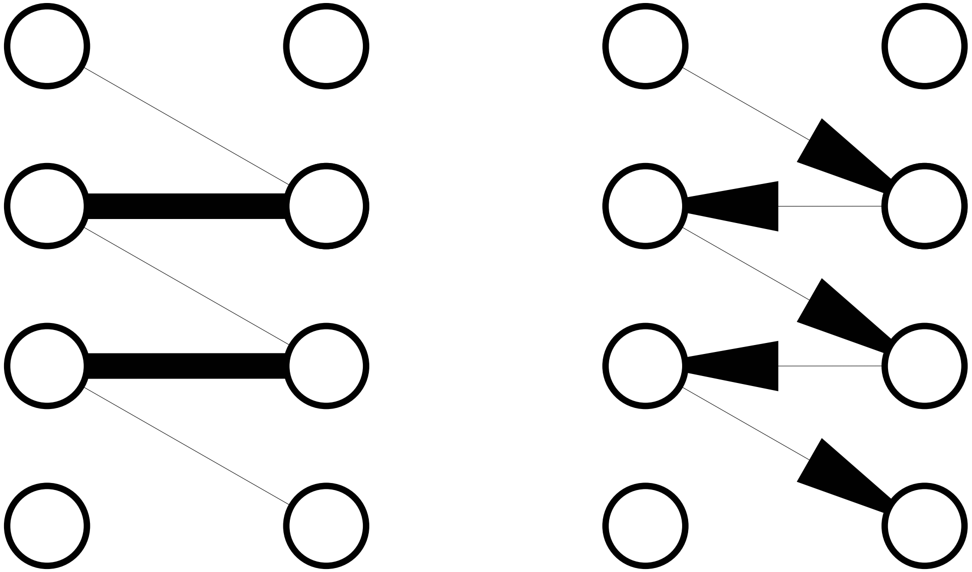
Let us assume that the path p was found. By the construction of the graph G_M we know that it:

- starts from free vertex in V_1,
- from V_1 to V_2 goes using free edge,
- from V_2 to V_1 returns using matched edge,
- ends at free vertex in V_2.

Hence the path p is an augmenting path.

Augmenting Paths

On the other hand if there is an augmenting path in M we can translate it directly into a path in G_M.



Augmenting Paths

We are now ready to give the algorithm MAXIMUM-MATCHING$(G = (V_1 \cup V_2, E))$ for finding maximum matchings in bipartite graphs.

- $M = \emptyset$
- repeat
  - $p =$FIND-AUGMENTING-PATH(G, M)
  - if $p \neq NIL$ then
    $M = M \oplus p$
- until $p = NIL$
- return M

Augmenting Paths

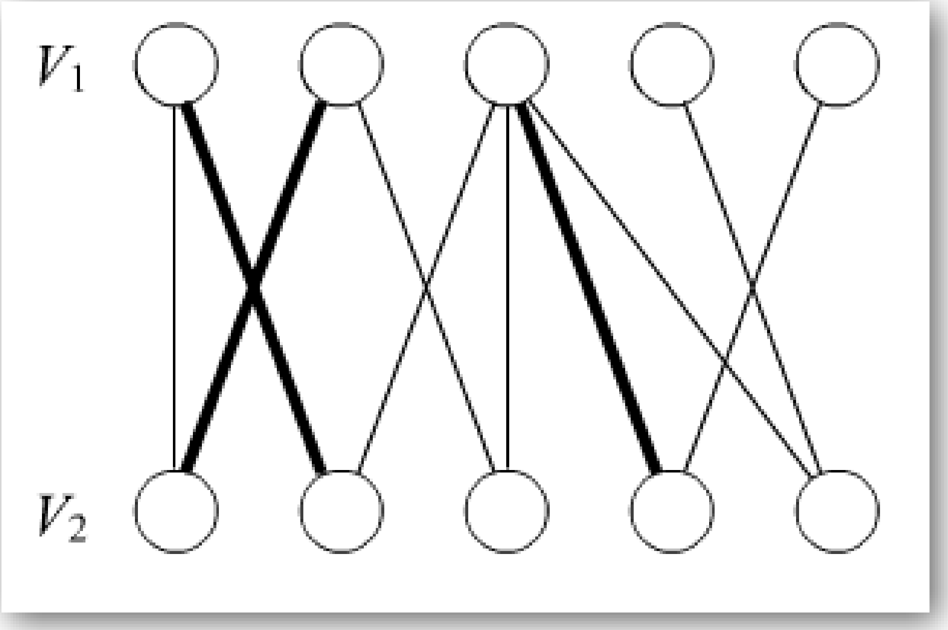
The correctness of the algorithm is a direct result of Lemma 1 and Theorem 1.

The size of maximum matching is upper bounded by $\frac{|V|}{2}$, and in each step of the loop the size of the matching grows by 1, so the loop will be executed at most $O(|V|)$ times.

We need $O(|E|)$ time to find each augmenting path, so the algorithm works in $O(|V||E|)$.

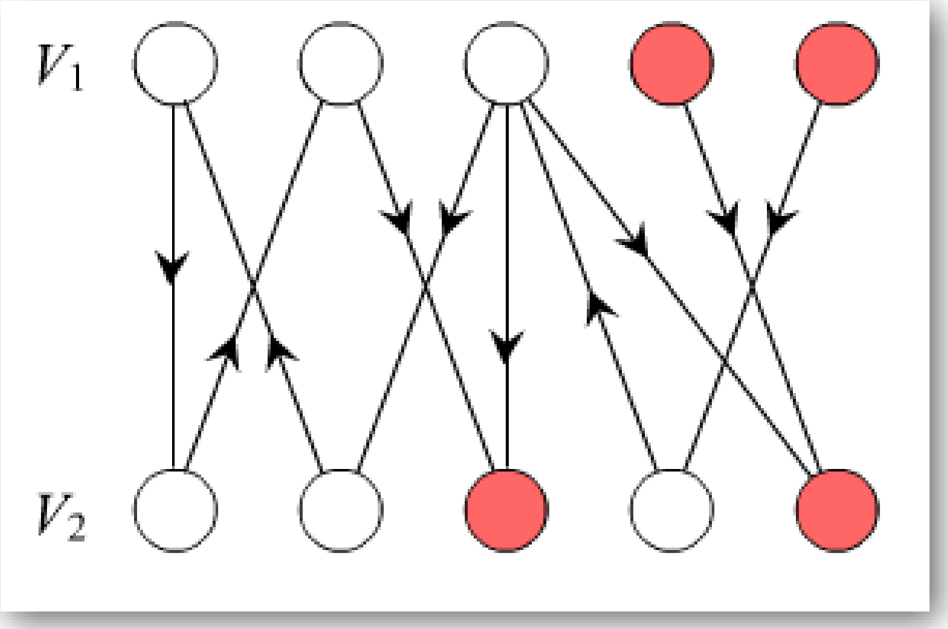
Augmenting Paths

One step of the algorithm:



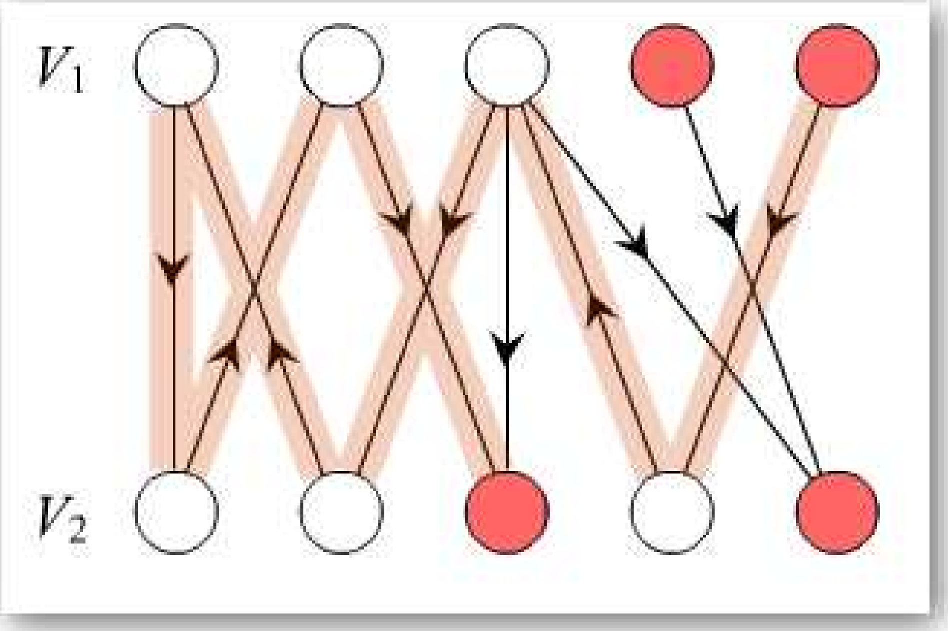
Augmenting Paths

One step of the algorithm:



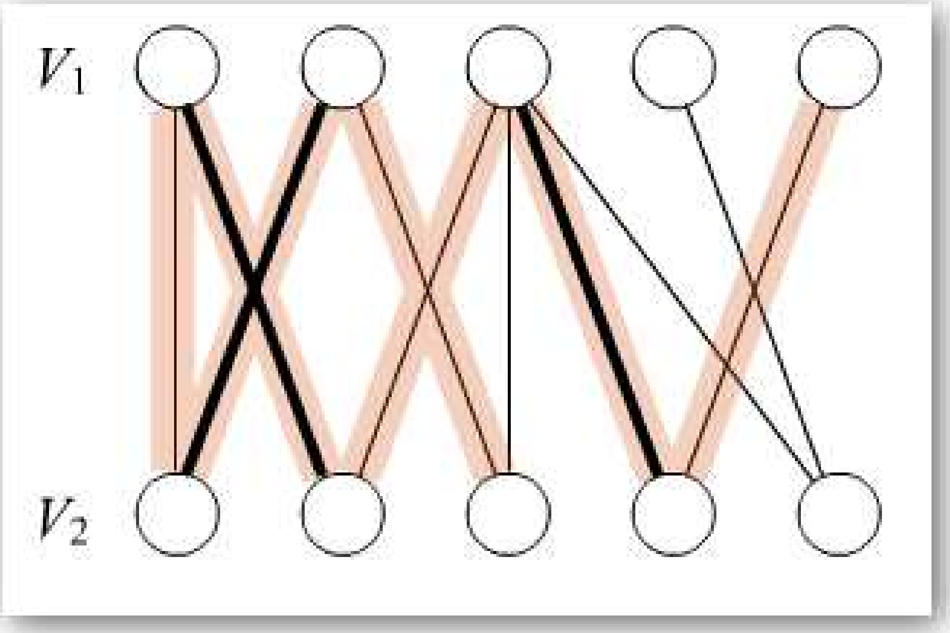
Augmenting Paths

One step of the algorithm:



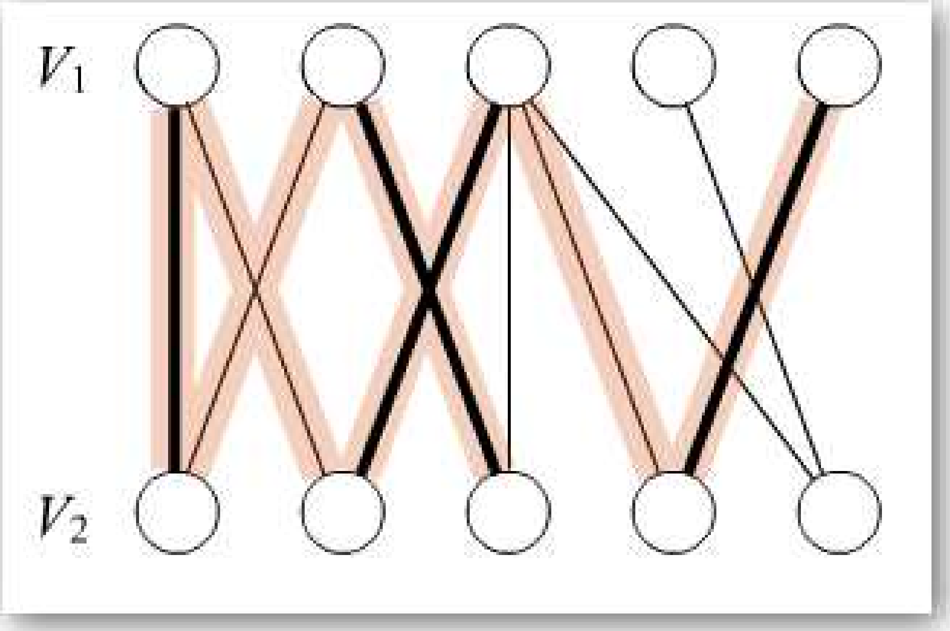
Augmenting Paths

One step of the algorithm:



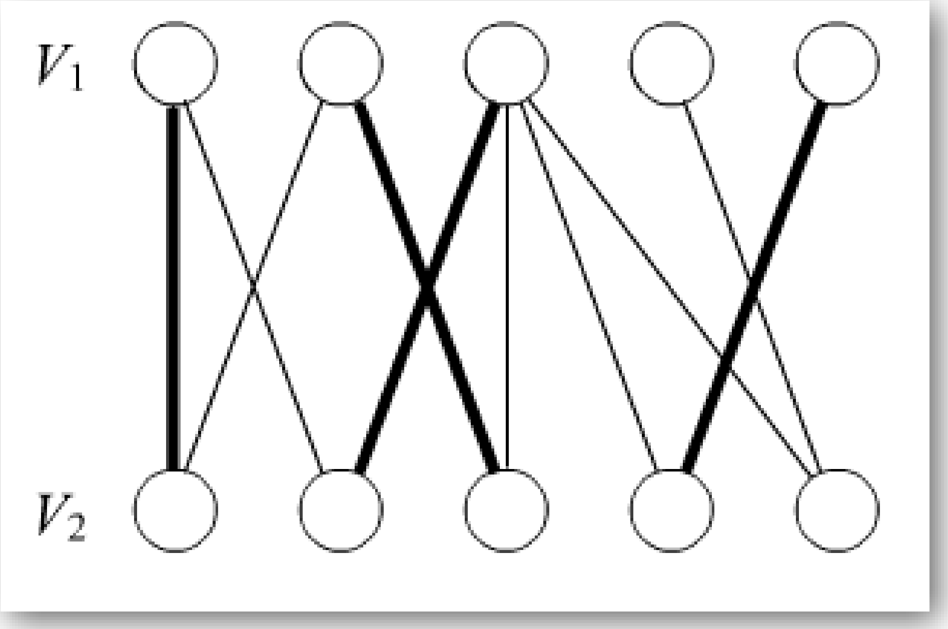
Augmenting Paths

One step of the algorithm:



Augmenting Paths

One step of the algorithm:



Ascending Price Matching

This algorithm is due to Noam Nisan. I was told about this algorithm from Renato Paes-Leme.

Denote by $G = (L, R, E)$ a bipartite graph with vertex set L, R and edge set $E = \{(v_i, v_j) : v_i \in L, v_j \in R\}$. Denote by $N(i) = \{v_j \in R : (v_i, v_j) \in E\}$ the neighbor set of $v_i \in L$. Similarly, define the neighbor set of $v_j \in R$.

Denote by $r(i) \in R$ the vertex matched to $v_i \in L$, and by $l(j) \in L$ the vertex matched to $v_j \in R$.

Ascending Price Matching

The Algorithm:

- Initially, set $\texttt{cost}(j) = 0, j \in R, r_i = \emptyset, v_i \in L,$ $l_j = \emptyset, v_j \in R.$
- Assign budget $b_i = 1$ to every vertex $v_i \in L.$
- Let $\delta < 1$ be a constant.
- Repeat
- If $\exists v_i \in L : r(i) = \emptyset$ and a vertex $v_j \in N(i) : \texttt{cost}(j) \leq b_j - \delta$ with $v_j = \texttt{argmin}_{\texttt{v}_\texttt{k} \in \texttt{N}(\texttt{i})} \texttt{cost}(\texttt{k})$
    - then $r(i) = v_j, l(j) = \emptyset, \texttt{cost}(j) = \texttt{cost}(j) + \delta$
    - else break

The analysis

Unmatched vertices $v_i \in L$ are organized in a priority queue by minimum $\texttt{cost}(j)$ of a neighbor $v_j \in N(i)$.

The algorithm extracts $v_i \in L$ from the queue at most $O(1/\delta)$ times for a total running time of $O(n/\delta \times \log n)$.

At every increase by δ of $\texttt{cost}(j)$ we update the priority for each $v_i \in N(j)$ with a total running time of $O\left(\sum_{v_j \in R} 1/\delta |N(j)|\right) = O(m \times 1/\delta)$

The Analysis

Denote the OPT the set of vertices of L that are matched in the optimal solution and by ALG the set of vertices of L that are matched by `Ascending Price Matching`. We prove in the next slides:

Lemma 2 $|OPT| \leq |ALG| + n \times \delta$.

If we choose $\delta = 1/\sqrt{n}$, `Ascending Price Matching` has running time $O(m\sqrt(n))$. By the previous lemma we need to find additional $\sqrt{n}$ augmenting paths to reach the cardinality of an optimal matching. This also needs at most $O(m\sqrt{n})$.

Proof of Lemma 2

All vertices in ALG are matched to vertices in R that have cost larger than the minimum cost by at most δ. It follows:

$$\sum_{v_i \in ALG \cap OPT} \left[1 - \texttt{cost}(r^{ALG}(i))\right] + \sum_{v_i \in OPT \setminus ALG} \left[1 - \texttt{cost}(r^{OPT}(i))\right]$$

$$\geq \sum_{v_i \in ALG \cap OPT} \left[1 - \texttt{cost}(r^{ALG}(i)) - \delta\right] \sum_{v_i \in OPT \setminus ALG} \left[1 - \texttt{cost}(r^{OPT}(i))\right]$$

We rewrite with

$$|OPT| \leq |ALG \cap OPT| + \sum_{v_i \in OPT} \texttt{cost}(r^{OPT}(i)) - \sum_{v_i \in OPT \cap ALG} \texttt{cost}(r^{ALG}(i)) \qquad (1)$$

$$+ \sum_{v_i \in ALG \cap OPT} \delta + \sum_{v_i \in OPT \setminus ALG} \left[1 - \texttt{cost}(r^{OPT}(i))\right] \qquad (2)$$

Proof of Lemma 2

Since every $v_j \in R$ of $\texttt{cost}(\texttt{v}_\texttt{j}) > 0$ is matched from ALG to a $v_i \in L$, we have

Claim 1

$$\sum_{v_i \in OPT} \texttt{cost}(r^{OPT}(i)) - \sum_{v_i \in OPT \cap ALG} \texttt{cost}(r^{ALG}(i))$$
$$\leq \sum_{v_i \in ALG \setminus OPT} \texttt{cost}(r^{ALG}(i)) \leq |ALG \setminus OPT|$$

The second claim follows from the observation that a vertex $v_i \in OPT \setminus ALG$ has each vertex $v_j \in N(i)$ with $\texttt{cost}(v_j) > 1 - \delta$.

Claim 2

$$\sum_{v_i \in ALG \cap OPT} \delta + \sum_{v_i \in OPT \setminus ALG} \left[1 - \texttt{cost}(r^{OPT}(i))\right] \leq |OPT| \times \delta$$

Proof of Lemma 2

We conclude from Equation 1, Claim 1 and Claim 2:

$$\begin{aligned} |OPT| &\leq |ALG \cap OPT| + |ALG \setminus OPT| + |OPT| \times \delta \\ &\leq |ALG| + n \times \delta \end{aligned}$$

Hopcroft-Karp Algorithm

In order to guarantee that the length of the paths grows in each phase we will in each phase construct a maximal set of disjoint augmenting paths P.

We will show now that when the matching is augmented using these paths the length of the shortest augmenting path increases.

Let us denote by $M \oplus P = M \oplus \bigoplus_{p \in P} p$.

Hopcroft-Karp Algorithm

Lemma 3 *Let k be the length of the shortest augmenting path with respect to M and let P be a maximal set of shortest disjoint augmenting paths with respect to M, then the length of the shortest augmenting path with respect to $M \oplus P$ is larger then k.*

Let us consider the shortest path π' with respect to $M \oplus P$.

If π' does not intersect any path from P its length has to be larger then k.

Otherwise its existence contradicts the assumption that P are shortest or maximal.

Hopcroft-Karp Algorithm

Now, let us consider the case when π' intersects some path π_1 from the set P.

We will show that $|\pi'| \geq |\pi_1| + 1$.

Actually, the path π' can intersect more then one path in P.

Let us consider that π' intersect paths $\pi_1, \pi_2, \ldots, \pi_t$ from P in the given order.

Hopcroft-Karp Algorithm

Using these paths and π' we can construct a set of $t+1$ new augmenting paths.

The path R_1 is constructed by taking the beginning of π' and then a piece of π_1.

The path R_i, for $i = 2, \ldots, t$, is constructed by taking a piece of π_i, then a piece of π', and finally a piece of π_{i+1}.

The last path R_{t+1} is constructed by taking a piece of π_t and the ending of π'.

Hopcroft-Karp Algorithm

The total length of the paths R_i is shorter by at least one than the total length of the paths π_i plus the length of π'.

$$1 + \sum_{i=1}^{t+1} |R_i| \leq |\pi'| + \sum_{i=1}^{t} |\pi_i|.$$

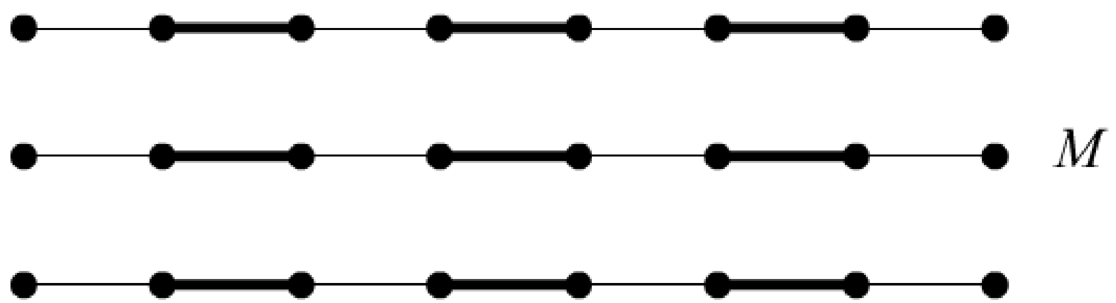
The paths R_i are augmenting with respect to M. Path R_i cannot be shorter than the pathsπ_i, so:

$$1 + \sum_{i=1}^{t} |\pi_i| + |\pi_1| \leq 1 + \sum_{i=1}^{t+1} |R_i| \leq |\pi'| + \sum_{i=1}^{t} |\pi_i|$$

that implies $1 \leq -|\pi_1| + |\pi'|$, what proves the

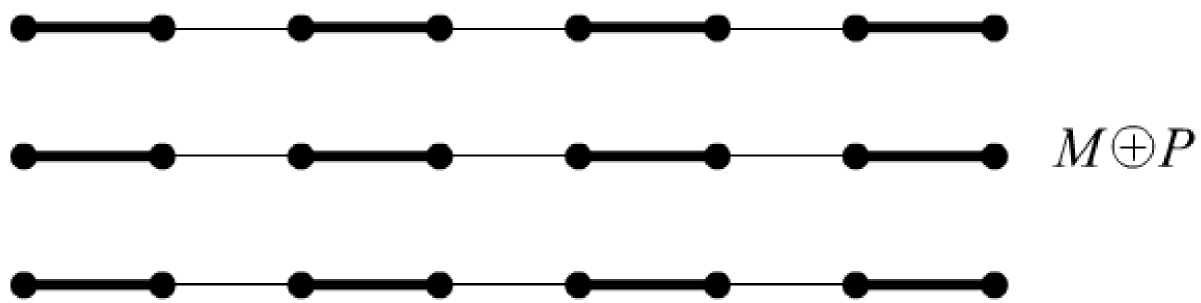
Hopcroft-Karp Algorithm

The steps of the proof:



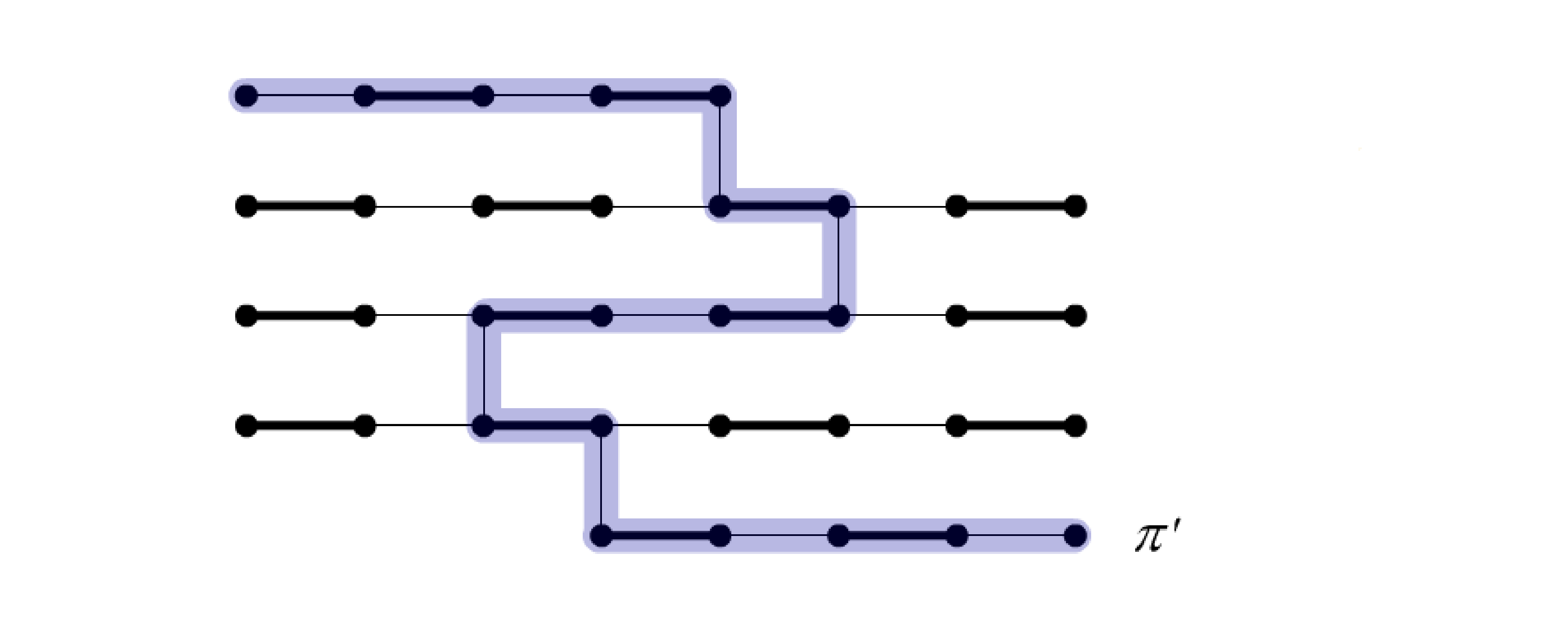
Hopcroft-Karp Algorithm

The steps of the proof:



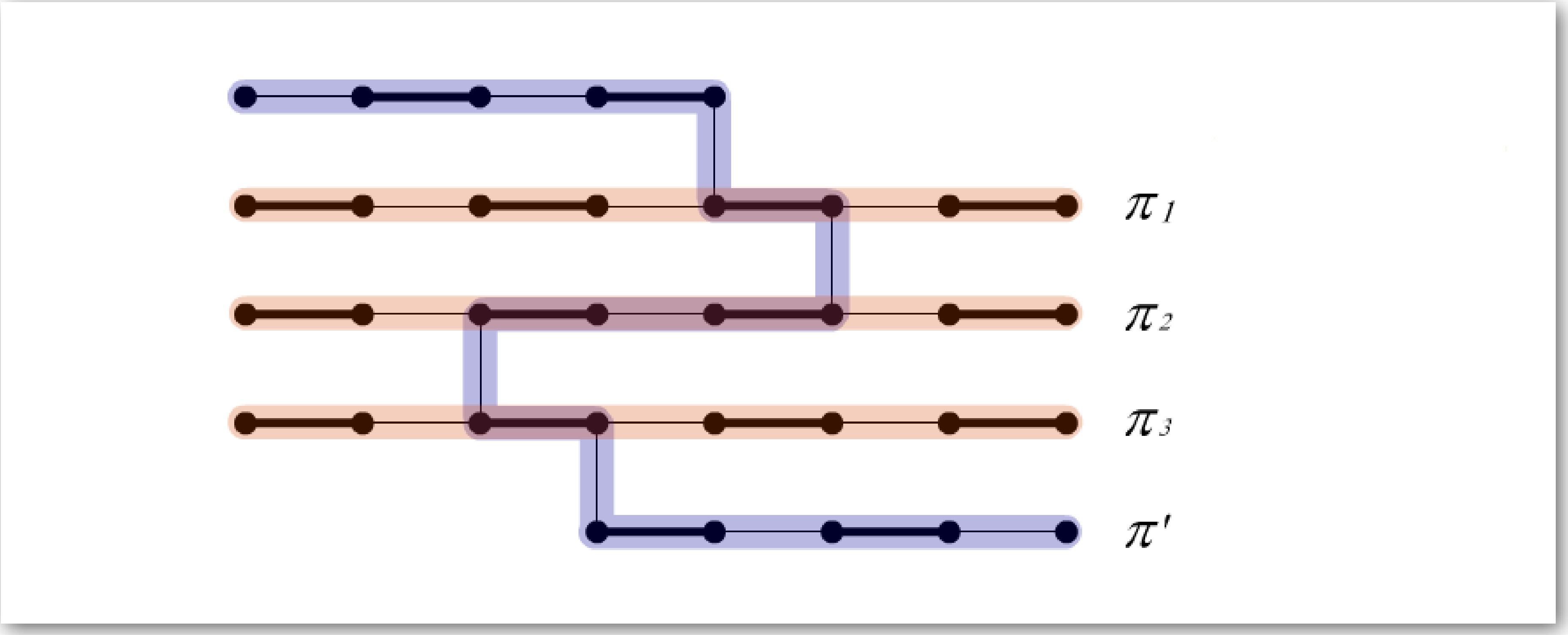
Hopcroft-Karp Algorithm

The steps of the proof:



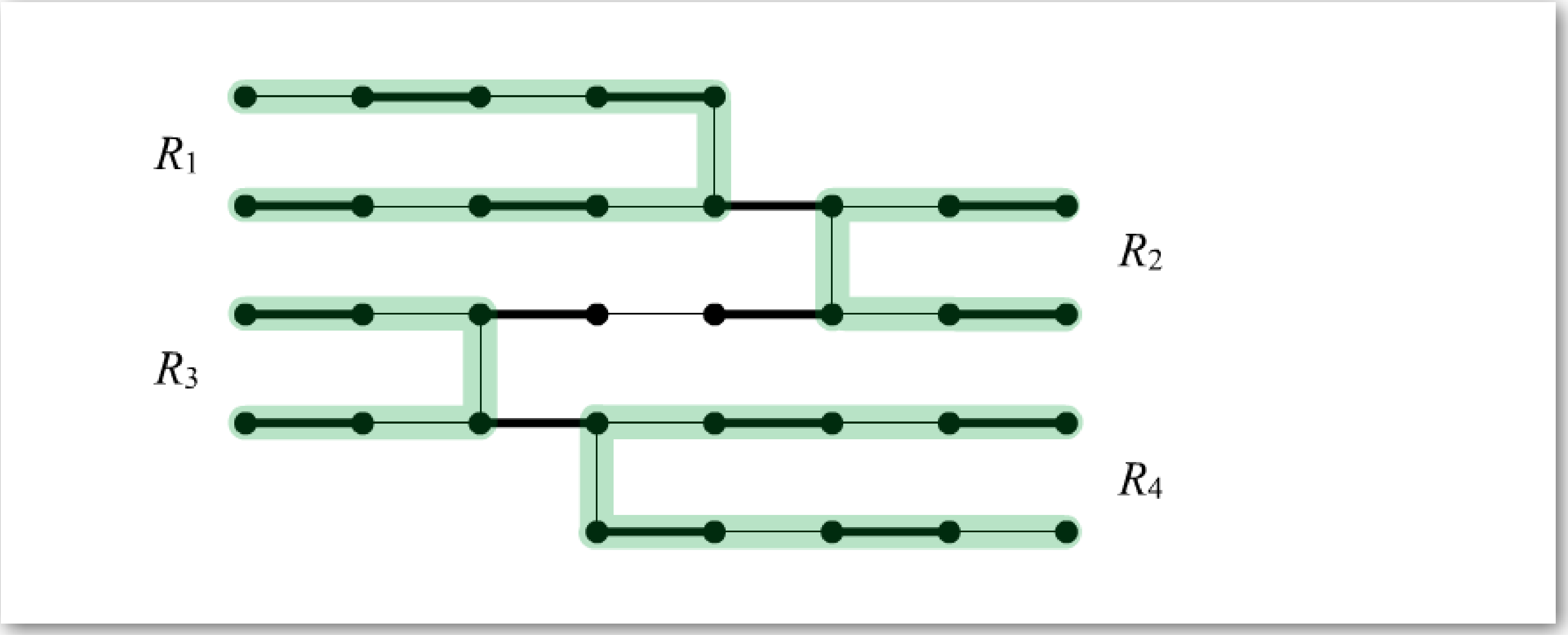
Hopcroft-Karp Algorithm

The steps of the proof:



Hopcroft-Karp Algorithm

The steps of the proof:



Hopcroft-Karp Algorithm

Let us now give a procedure `PARTIAL-DFS`(G, v, T) for constructing the set of paths P. The procedure is based on the DFS search.

- run `DFS`(G, v) till you find the first vertex from T
- remove all visited vertices during DFS from graph G
- `if` there exists a path p from v to T `then`
  - `return` p
- `else`
  - `return` NIL

Hopcroft-Karp Algorithm

This procedure is different from the standard DFS because:

- carries over the search till it finds the first vertex from the set T,
- after the search is finished it removes from the graph all visited vertices.

Removing visited vertices assures that the path that will be found later on will not intersect.

Hopcroft-Karp Algorithm

We will use this procedure to the layered graph $\overline{G}_M$ that is constructed out of G_M.

Let V_1' be the set of free vertices in V_1.

Let $d : V \to \mathcal{N}$ be the distance $d(v)$ of a vertex v from the vertices in V_1'.

The graph $\overline{G}_M = (V_1 \cup V_2, \overline{E}_M)$ contains the following edges:

$$\overline{E}_M = \{(u,v) : (u,v) \in E_M \mid d(u) + 1 = d(v)\}.$$

Hopcroft-Karp Algorithm

We can now prove the following simple lemma:

Lemma 4 *Every path in $\overline{G}_M$, that start in V_1', is the shortest path in G_M.*

The lemma follows directly from the definition of the shortest paths, i.e., the path is shortest if its length is equal to the distance from its beginning to its end.

Hopcroft-Karp Algorithm

MAXIMAL-SET-OF-PATHS$(G = (V_1 \cup V_2, E), M)$.

- $P = \emptyset$
- construct the graph $\overline{G}_M = (V_1 \cup V_2, \overline{E}_M)$
- let V_1' be the set of free vertices in V_1
- for $v \in V_1'$ do
- begin
  - $p =$PARTIAL-DFS(G, v, V_2')
  - if $p \neq NIL$ then
    - $P = P \cup p$
- end
- return P

Hopcroft-Karp Algorithm

Lemma 5 *The procedure* `MAXIMAL-SET-OF-PATHS` *finds a maximal set of shortest vertex disjoint augmenting paths with respect to* M *in* $O(|E|)$ *time.*

$O(|E|)$ running time is the result of the construction of the `PARTIAL-DFS` procedure that considers each vertex, and each edge only once.

The visited vertices are removed, so the paths in P are disjoint.

By Lemma 4 the returned path are the shortest.

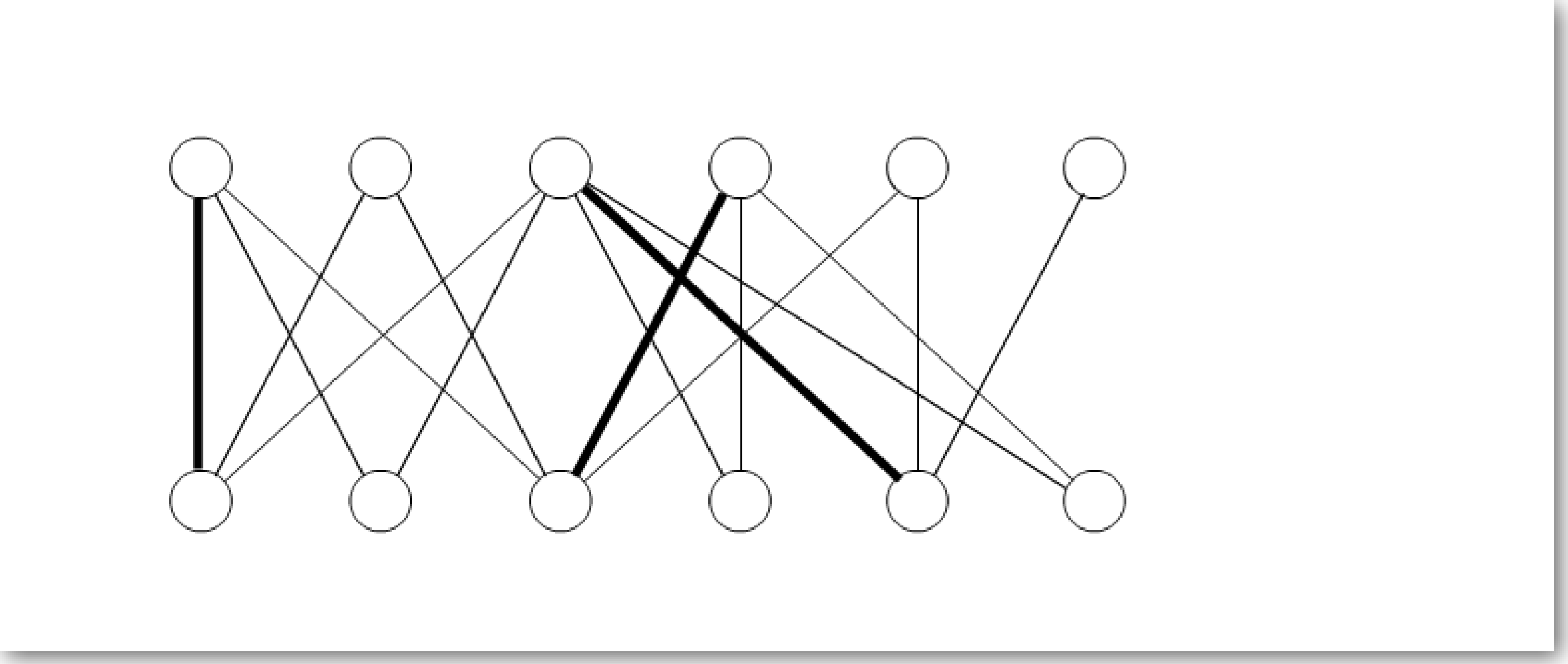
Hopcroft-Karp Algorithm

The $\texttt{HOPCROFT-KARP}(G = (V_1 \cup V_2, E))$ algorithm is given as:

- $M = \emptyset$
- `repeat`
  - $P =$`MAXIMAL-SET-OF-PATHS`$(G = (V_1 \cup V_2, E), M)$
  - `if` $P \neq NIL$ `then`
    - $M = M \oplus P$
- `until` $P = NIL$
- `return` M

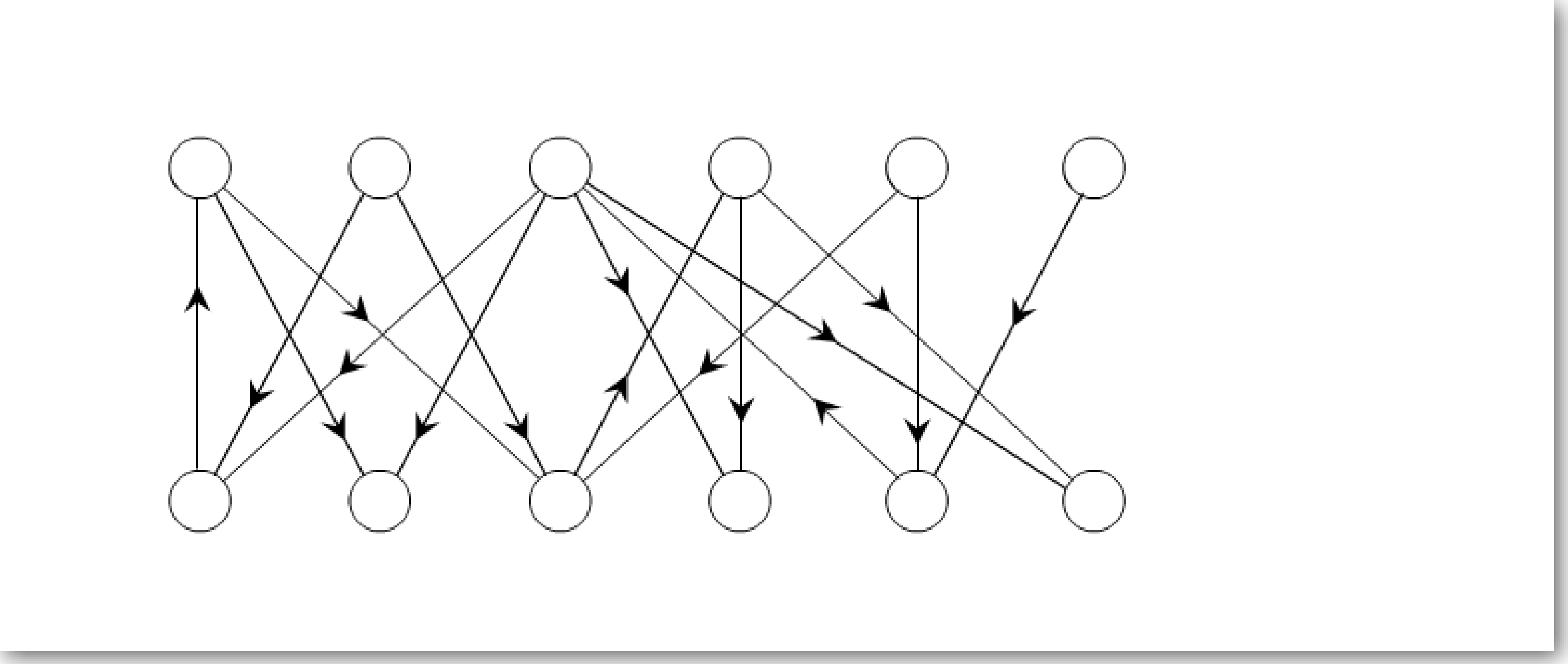
Hopcroft-Karp Algorithm

All steps of the algorithm:



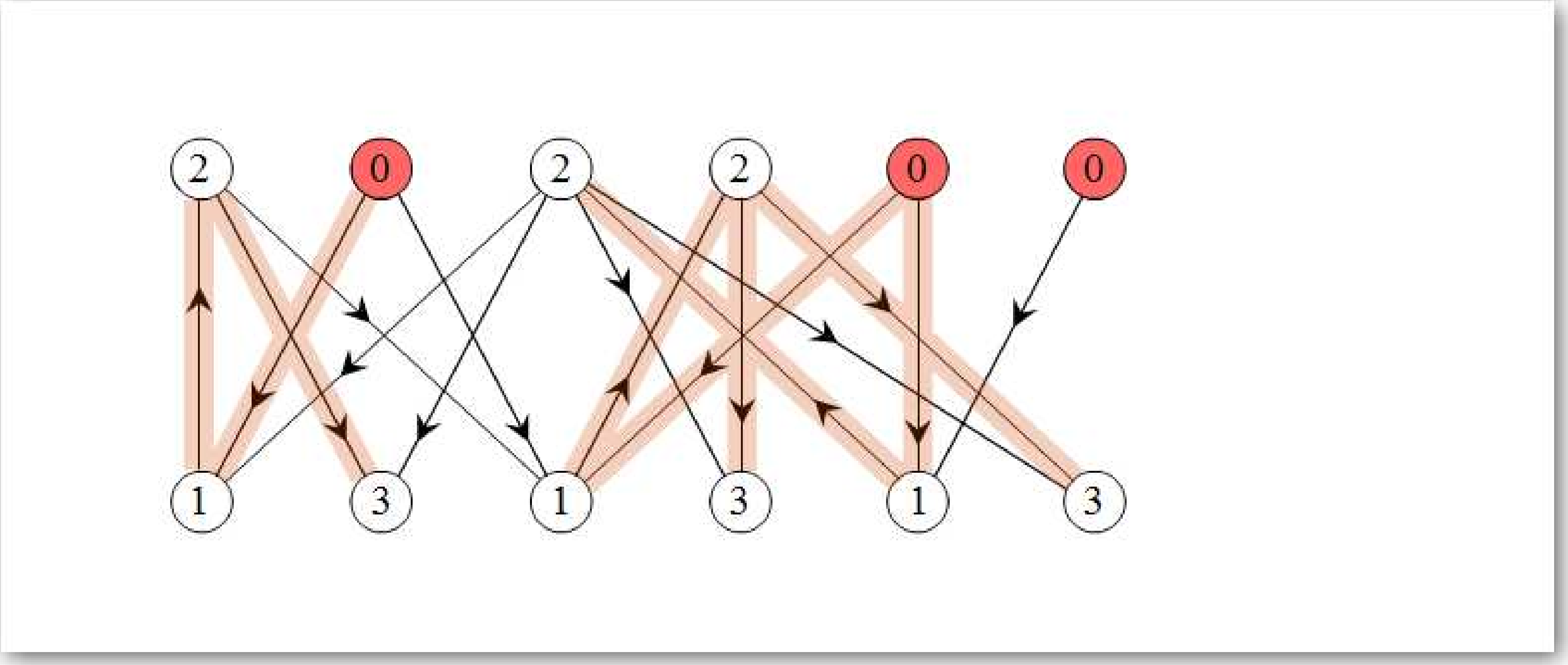
Hopcroft-Karp Algorithm

All steps of the algorithm:



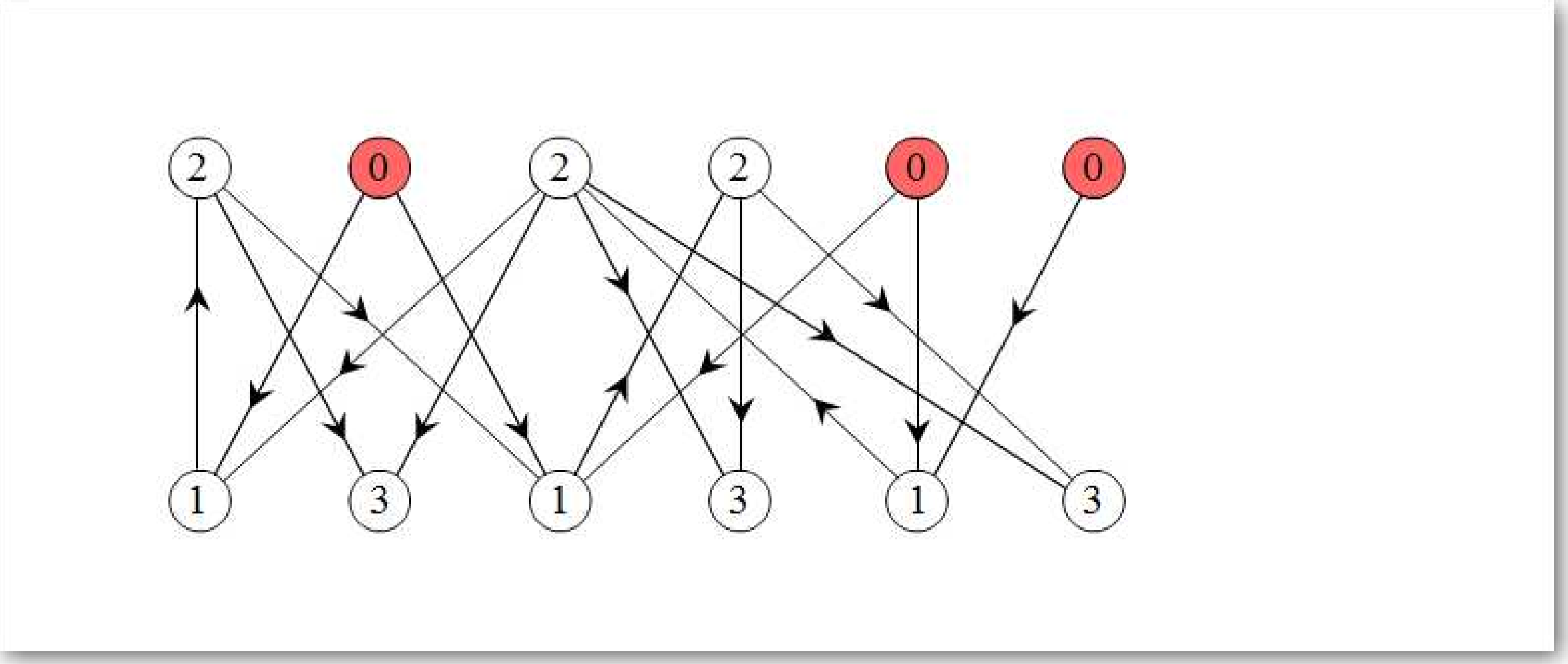
Hopcroft-Karp Algorithm

All steps of the algorithm:



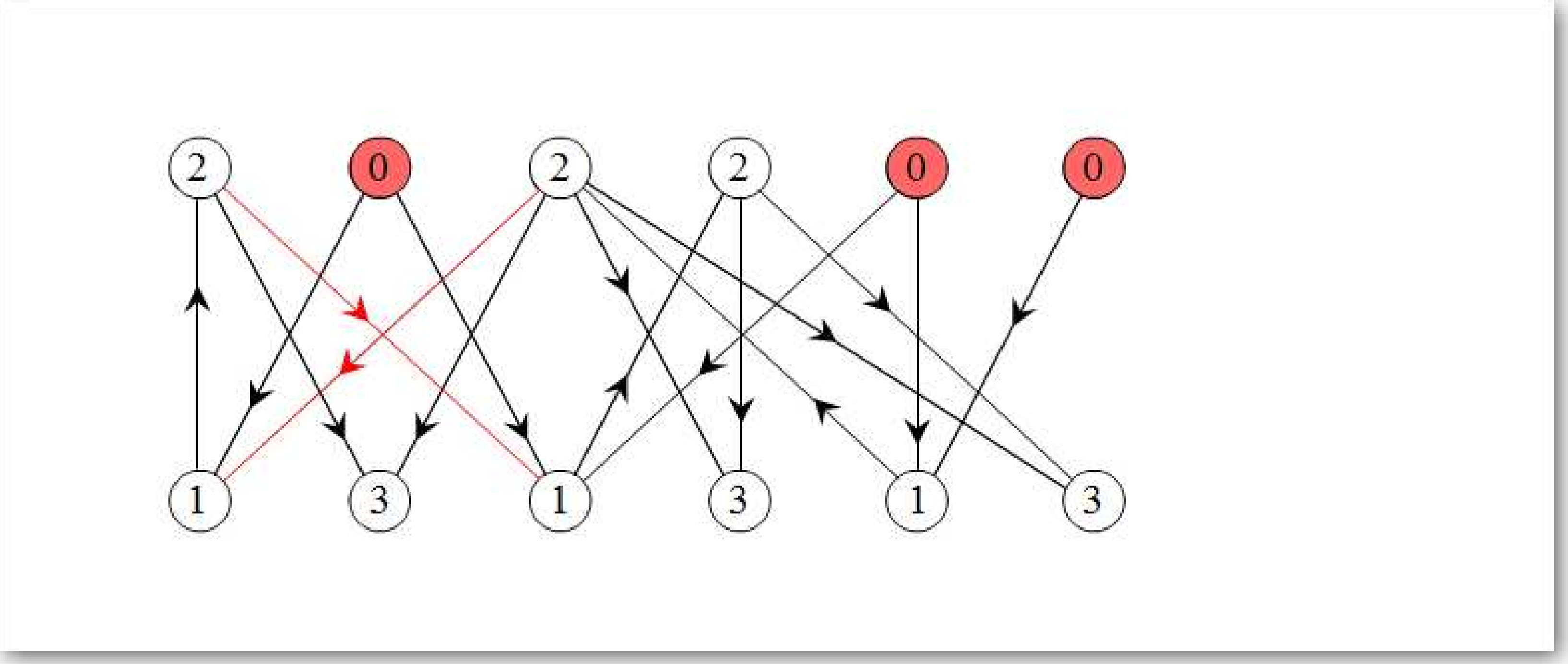
Hopcroft-Karp Algorithm

All steps of the algorithm:



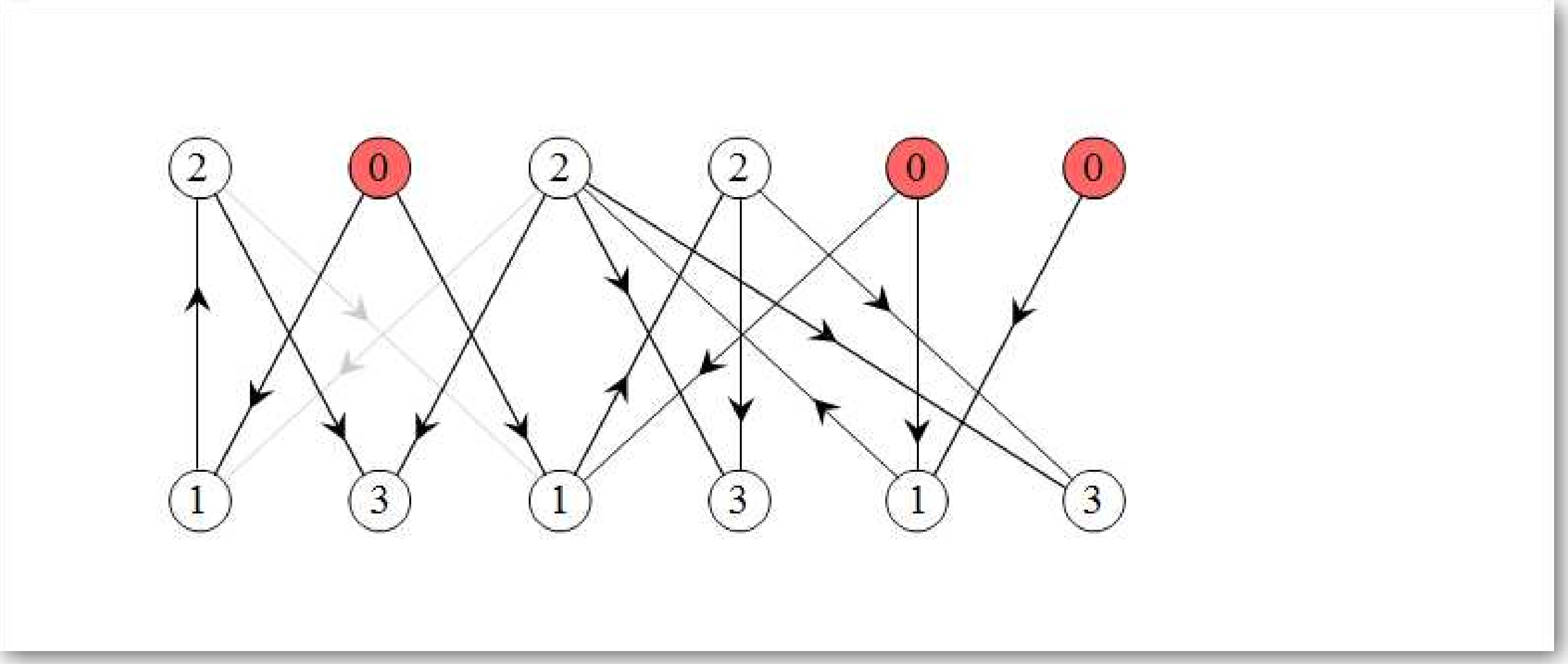
Hopcroft-Karp Algorithm

All steps of the algorithm:



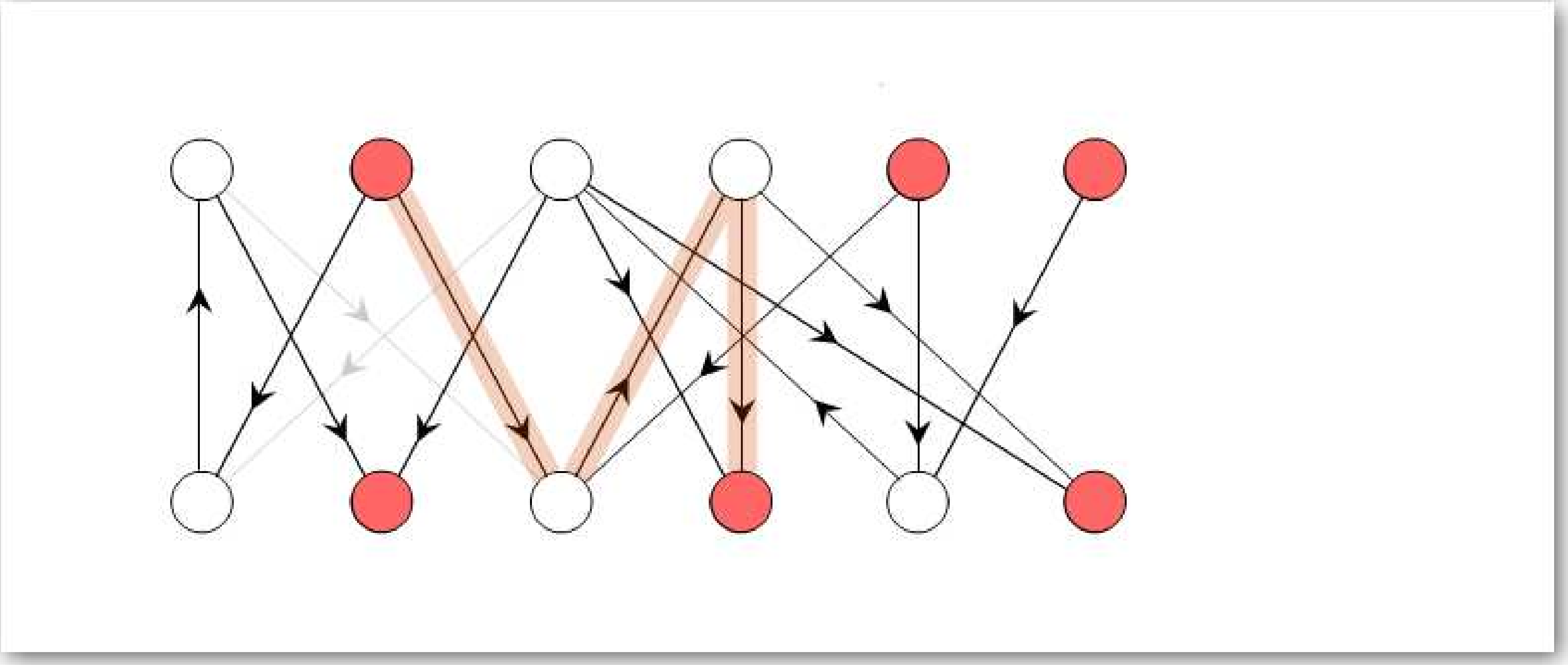
Hopcroft-Karp Algorithm

All steps of the algorithm:



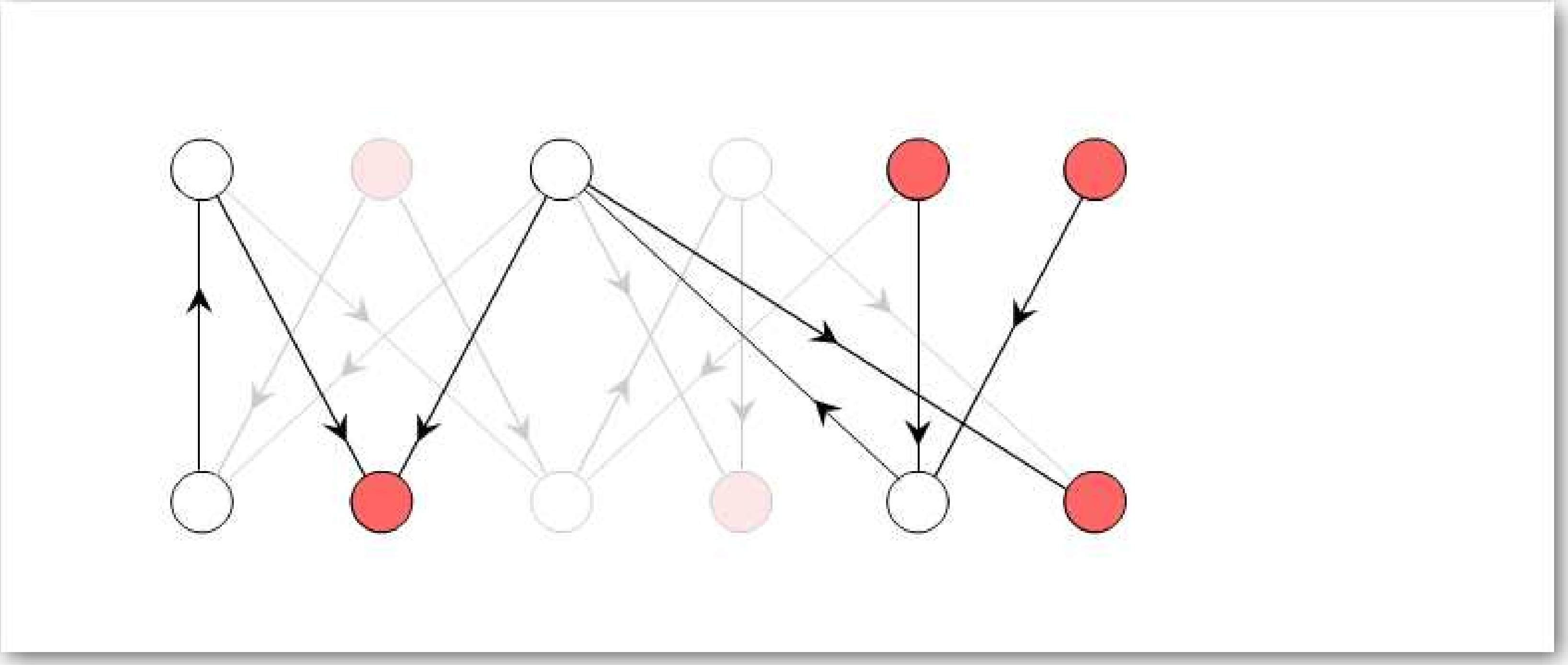
Hopcroft-Karp Algorithm

All steps of the algorithm:



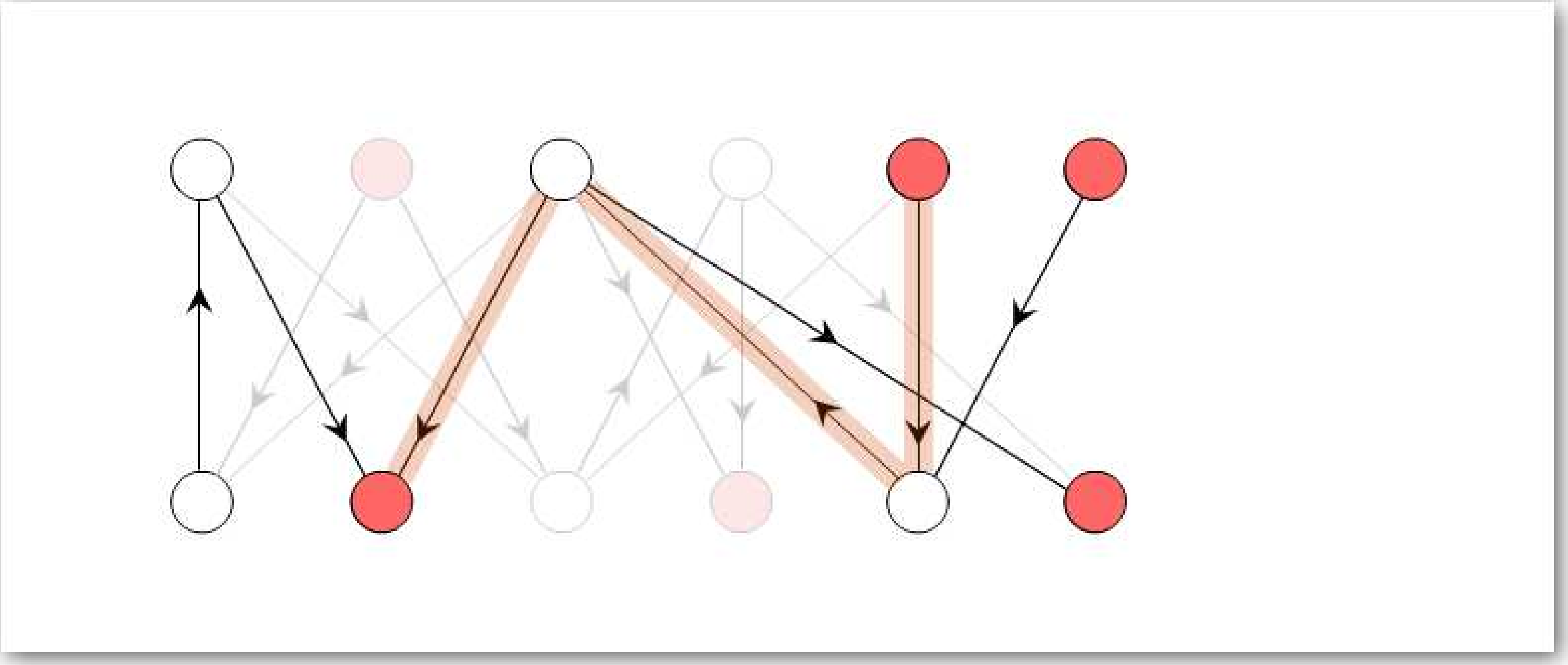
Hopcroft-Karp Algorithm

All steps of the algorithm:



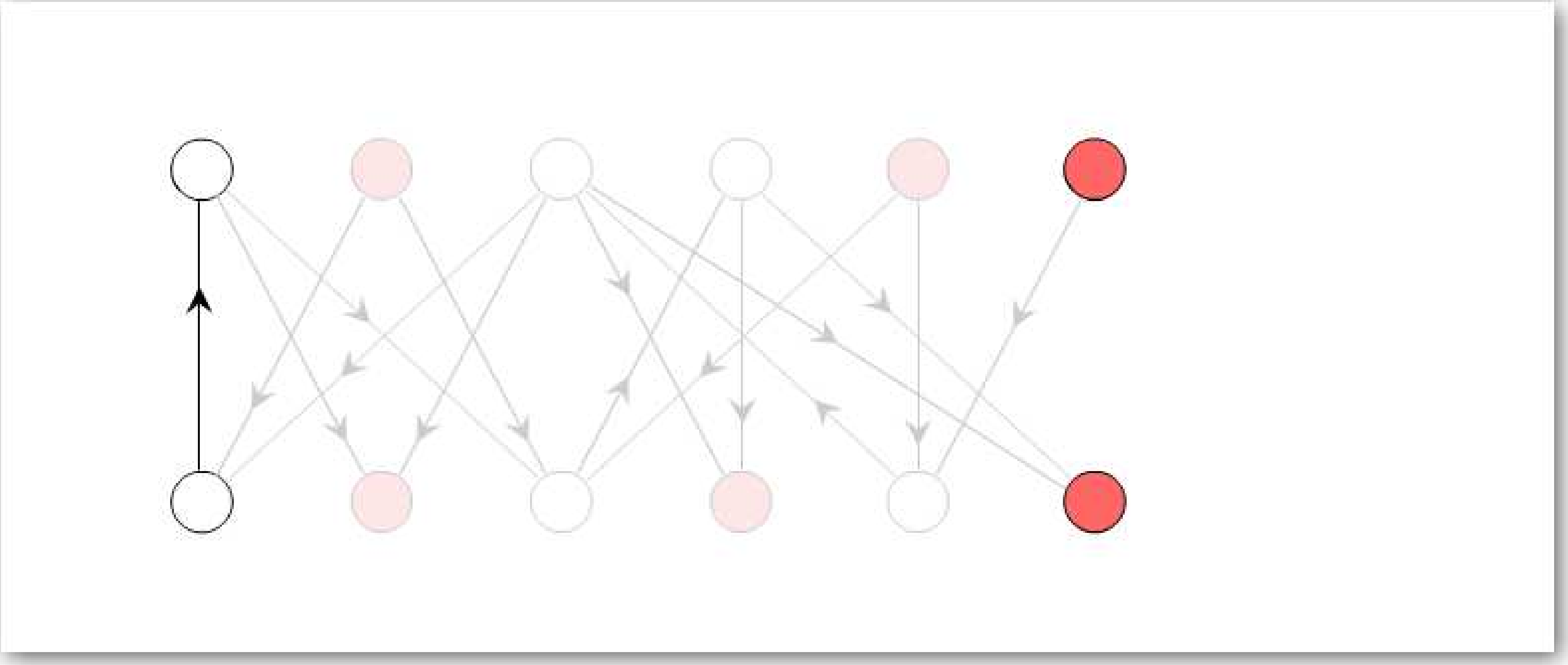
Hopcroft-Karp Algorithm

All steps of the algorithm:



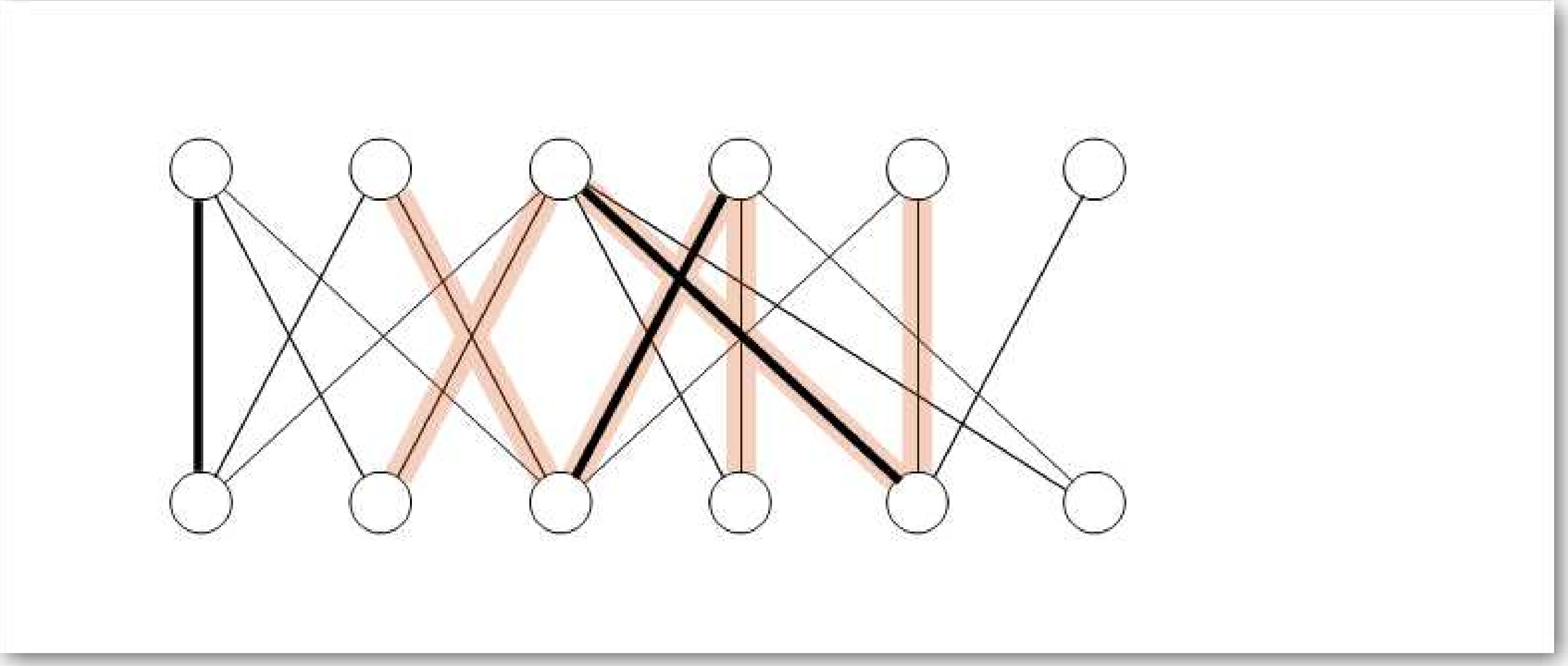
Hopcroft-Karp Algorithm

All steps of the algorithm:



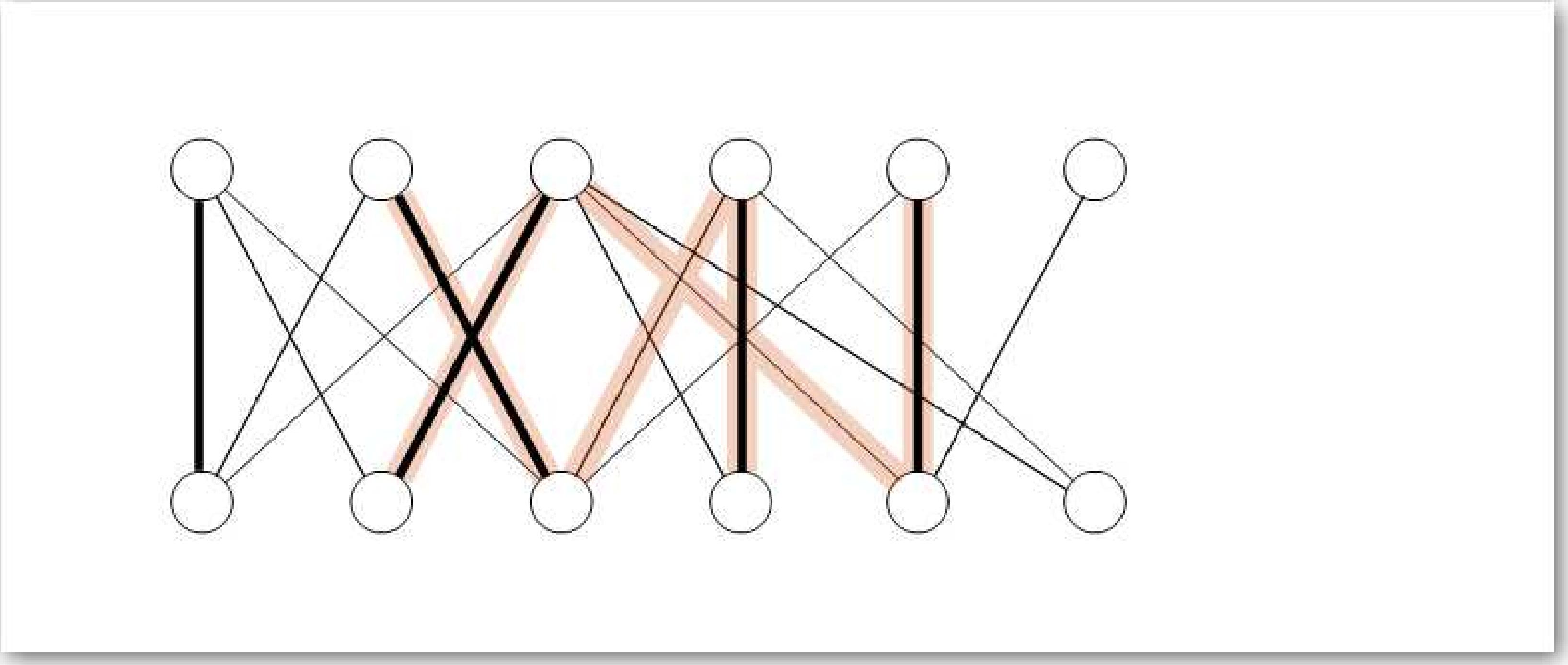
Hopcroft-Karp Algorithm

All steps of the algorithm:



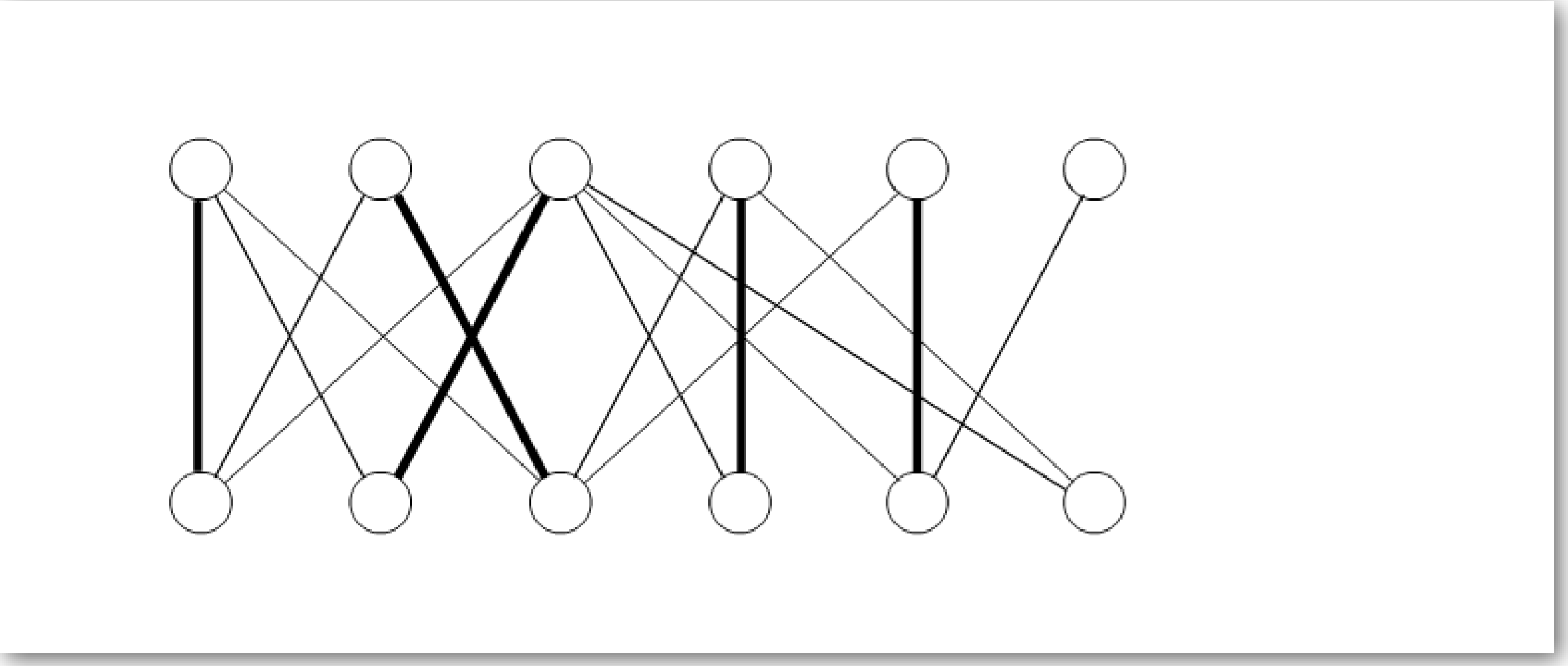
Hopcroft-Karp Algorithm

All steps of the algorithm:



Hopcroft-Karp Algorithm

All steps of the algorithm:



Hopcroft-Karp Algorithm

Theorem 2 *The Hopcroft-Karp algorithm find a maximum matching in a bipartite graph in* $O(\sqrt{|V|}|E|)$ *time.*

Recall the lemma from the very beginning:

Lemma 6 *Let* M^* *be a maximum matching, and let* M *be any matching in G. If the length of the shortest augmenting path with respect to* M *is* k*, then* $|M^*| - |M| \leq \frac{|V|}{k}$.

Hopcroft-Karp Algorithm

The correctness of the algorithm is impled by the Berge theorem, as if the graph contains an augmenting path then P will not be empty.

Lemma 3 implies that in each phase of the algorithm the length of the shortest augmenting path increases by 1.

Therefore, after $\sqrt{|V|}$ phases the length will be at least $\sqrt{|V|}$.

Hopcroft-Karp Algorithm

No from Lemma **??** we know that there are at most $\sqrt{|V|}$ augmenting paths left.

Hence, the main loop of the algorithm will be execute at most $\sqrt{|V|}$ times more.

In total the loop will be execute at most $2\sqrt{|V|}$ times.

Every execution takes $O(|E|)$, so by Lemma 5 the total running time of the algorithm is $O(\sqrt{|V|}|E|)$.