

A Fast Matching Algorithm

Algorithm 53 Bimatch-Hopcroft-Karp(G)

```
1:  $M \leftarrow \emptyset$ 
2: repeat
3:   let  $\mathcal{P} = \{P_1, \dots, P_k\}$  be maximal set of
4:   vertex-disjoint, shortest augmenting path w.r.t.  $M$ .
5:    $M \leftarrow M \oplus (P_1 \cup \dots \cup P_k)$ 
6: until  $\mathcal{P} = \emptyset$ 
7: return  $M$ 
```

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

Analysis

Lemma 4

Given a matching M and a maximal matching M^* there exist $|M^*| - |M|$ *vertex-disjoint augmenting paths* 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

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

Lemma 5

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

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 ℓ .

Analysis

Lemma 6

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, $|A| \leq k\ell + |P| - 1$.
- ▶ The lower bound on $|A|$ gives $(k + 1)\ell \leq |A| \leq k\ell + |P| - 1$, and hence $|P| \geq \ell + 1$.

Analysis

If the shortest augmenting path w.r.t. a matching M has ℓ 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.

Lemma 7

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 $\lfloor \sqrt{|V|} \rfloor + 1 \geq \sqrt{|V|}$.
- ▶ Hence, there can be at most $|V| / (\sqrt{|V|} + 1) \leq \sqrt{|V|}$ additional augmentations.

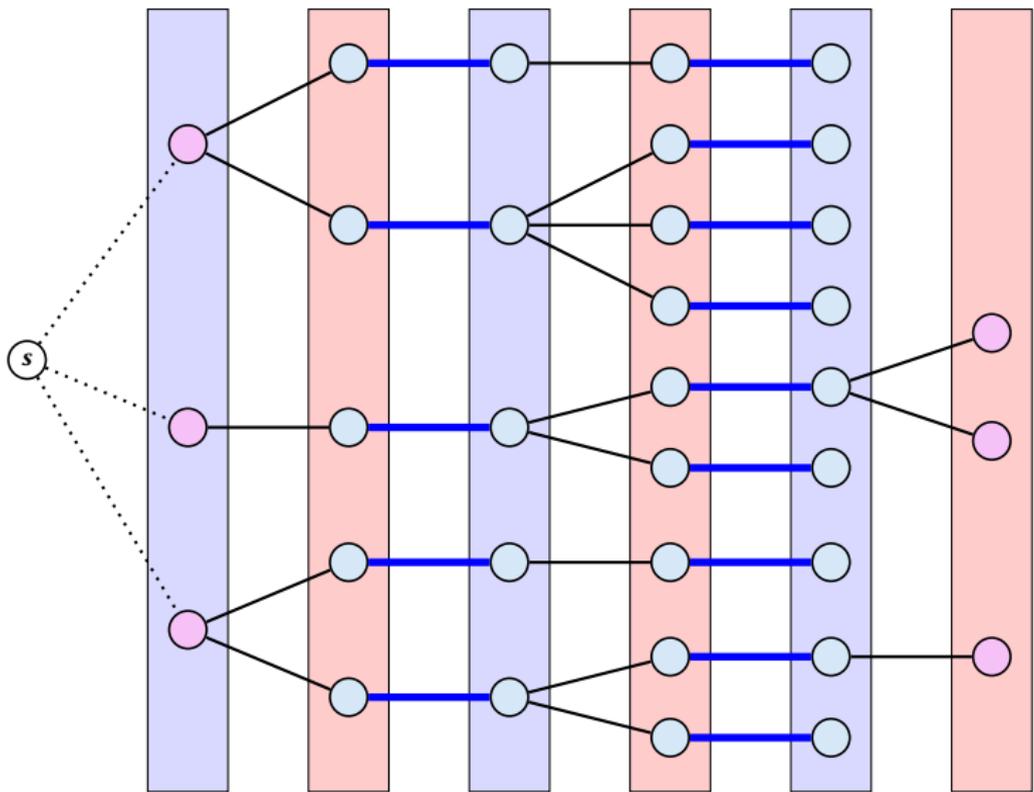
Lemma 8

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

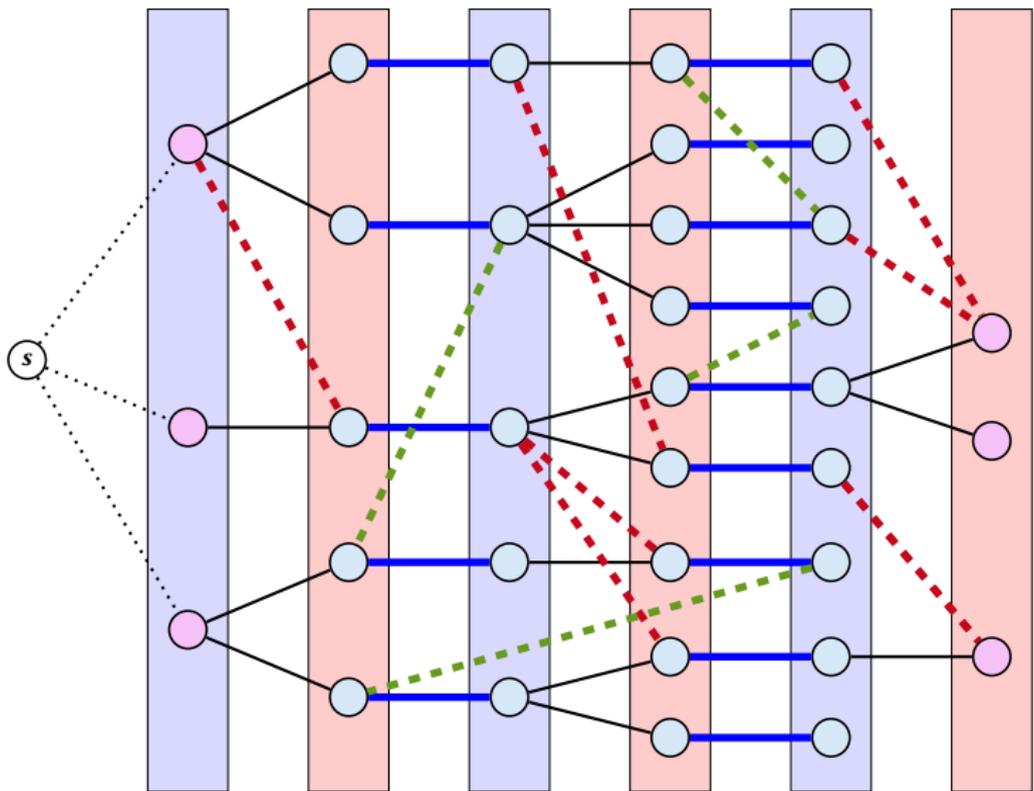
- ▶ Do a breadth first search starting at all free vertices in the left side L .
(alternatively add a super-startnode; connect it to all free vertices in L and start breadth first search from there)
- ▶ The search stops when reaching a free vertex. However, the current **level** of the BFS tree is still finished in order to find a set F of free vertices (on the right side) that can be reached via shortest augmenting paths.

Analysis

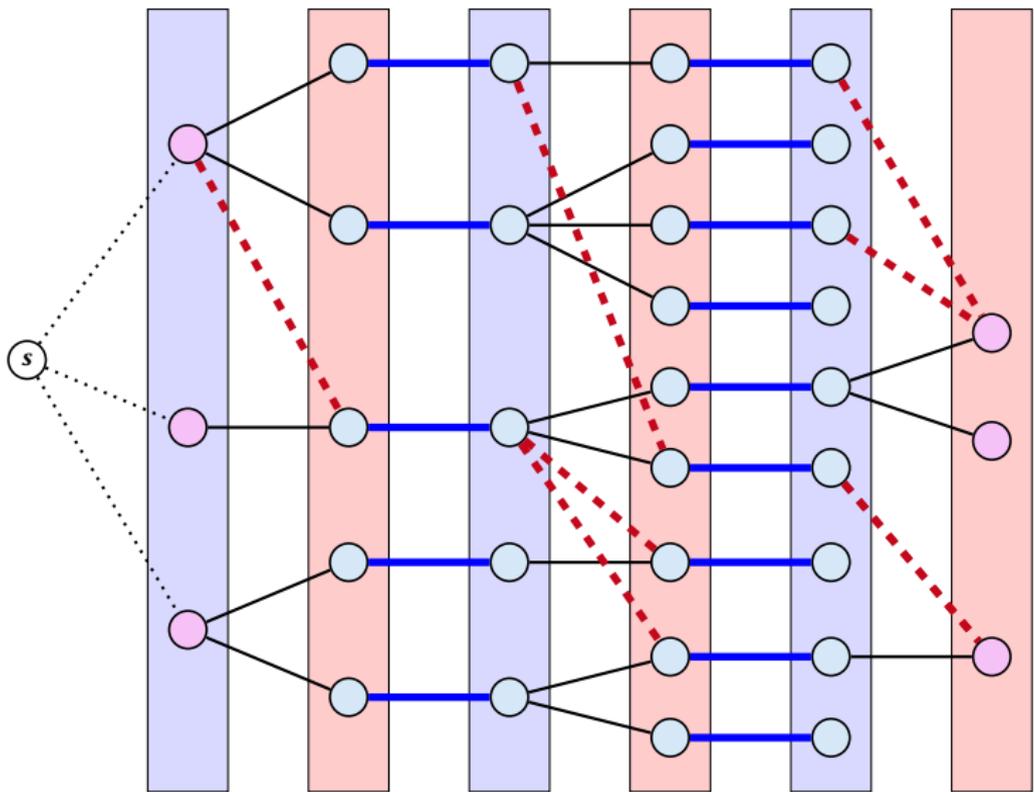
- ▶ Then a maximal set of shortest path from the leftmost layer of the tree construction to nodes in F needs to be computed.
- ▶ Any such path must visit the layers of the BFS-tree from left to right.
- ▶ To go from an odd layer to an even layer it must use a matching edge.
- ▶ To go from an even layer to an odd layer edge it can use edges in the BFS-tree **or** edges that have been ignored during BFS-tree construction.
- ▶ We direct all edges btw. an even node in some layer ℓ to an odd node in layer $\ell + 1$ from left to right.
- ▶ A DFS search in the resulting graph gives us a maximal set of vertex disjoint path from left to right in the resulting graph.



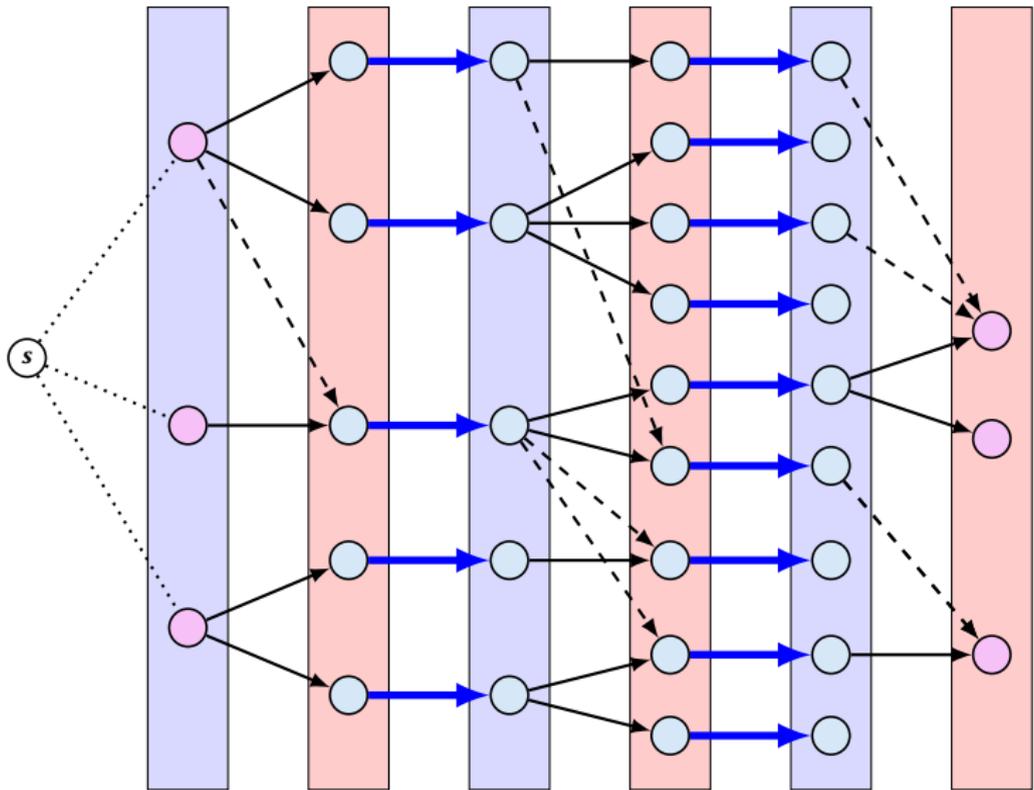
Compute an alternating tree in BFS fashion starting from all free vertices on the left (L); finish on the level where you see the first free vertex from the right set R .



This fixes length of shortest alternating path; every shortest alternating path must visit layers from left to right (green edges are not helpful)



Delete green edges and direct remaining edges from left to right;



Every shortest alternating path is a path in this graph from a left free vertex to a right free vertex; find a maximal vertex disjoint set of path by a modified DFS