Maximum Cardinality Matching

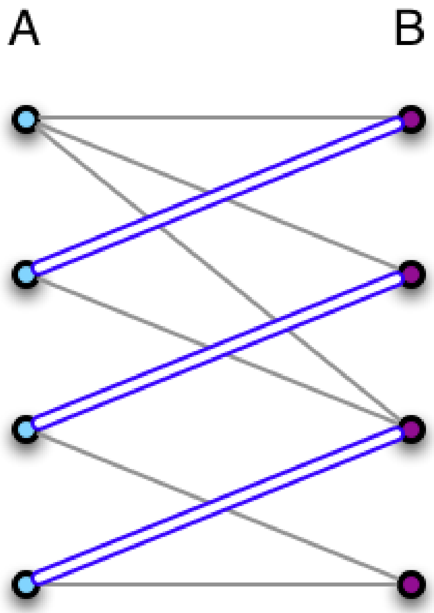
Ran Duan (Postdoc at MPI)

In this lecture

- Concepts of matchings
- Hopcroft-Karp’s algorithm for maximum cardinality matching in bipartite graphs
- Edmonds’s algorithm for maximum cardinality matching in general graphs

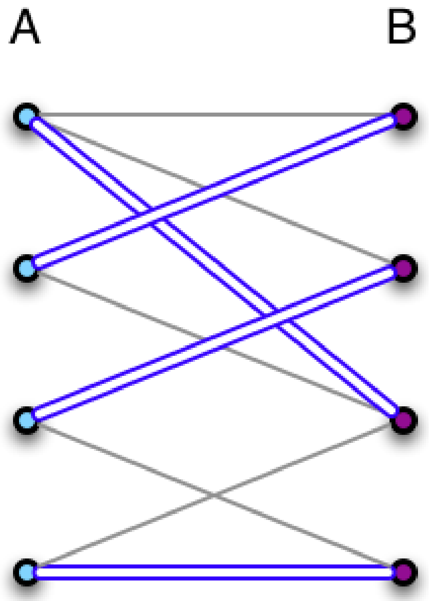
Basic Concepts and Notations

- In graph G=(V,E), a matching M:
  - A set of vertex-disjoint edges
  - Matched vertices: the vertices associated with an edge in M
  - Free vertices: unmatched vertices



Basic Concepts and Notations

- Matching M:
    - A set of vertex-disjoint edges
    - Matched vertices: the vertices associated with an edge in M
    - Free vertices: unmatched vertices
- Perfect Matching M:
    - No free vertices



Basic Concepts and Notations

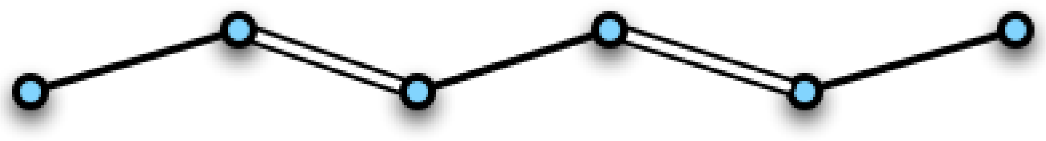
- Matching M:
  - A set of vertex-disjoint edges
  - Matched vertices: the vertices associated with an edge in M
  - Free vertices: unmatched vertices
- Perfect Matching M:
  - No free vertices
- Maximum Cardinality Matching (MCM)
  - Maximize |M|
- Maximum Weighted Matching (MWM)
  - Maximize $\sum_{e \in M} w(e)$

Applications of matching problems

- Assignment
  - Minimizing costs in job assigments
- Image Feature Matching
  - Match image features (lines, points) between two images
- Building blocks of other algorithms
  - 1.5-Approximate travelling salesman problem
    - Christofides algorithm
    - Find a minimum spanning tree and a minimum weight perfect matching

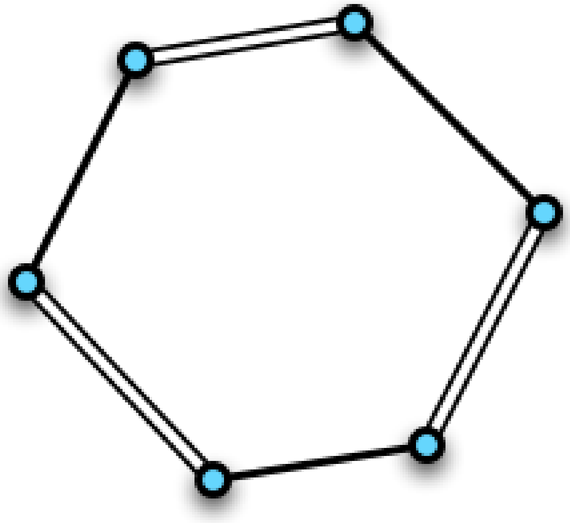
Augmenting paths

- Alternating paths:
  - The edges alternate between M and E\M
- Augmenting paths
  - The alternating paths whose both ends are free vertices
  - One more non-matching edges than matching edges



Alternating Cycles

- A cycle in which edges alternate between M and E\M



- Given two matchings M_1 and M_2, then the subgraph $(V, M_1 \oplus M_2)$ is composed of
    - isolating vertices
    - alternating paths
    - alternating cycles

    - all alternate between M_1 and M_2
    - "$A \oplus B$" represents $(A \setminus B) \cup (B \setminus A)$

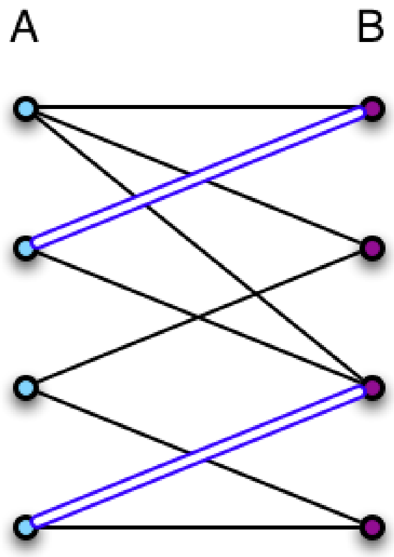
- Given two matching M_1 and M_2, then the subgraph $(V, M_1 \oplus M_2)$ is composed of
  - isolating vertices
  - alternating paths
  - alternating cycles

  - all alternate between M_1 and M_2
  - "$A \oplus B$" represents $(A \setminus B) \cup (B \setminus A)$
- Because every vertex only associated with at most one edge in each M_i

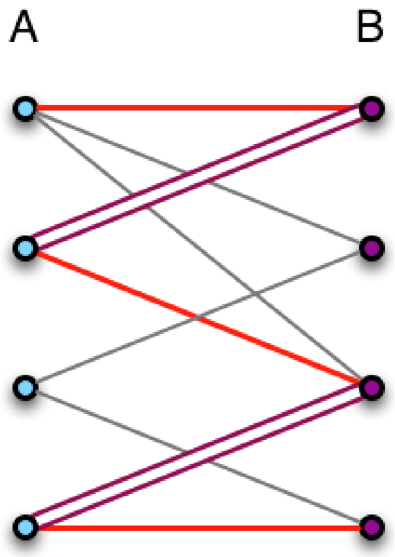
Augmenting Paths

- Augmenting paths
  - The alternating paths whose both ends are free vertices
- If P is an augmenting path w.r.t. M, then $M \oplus P$ is also a matching M', and |M'|=|M|+1

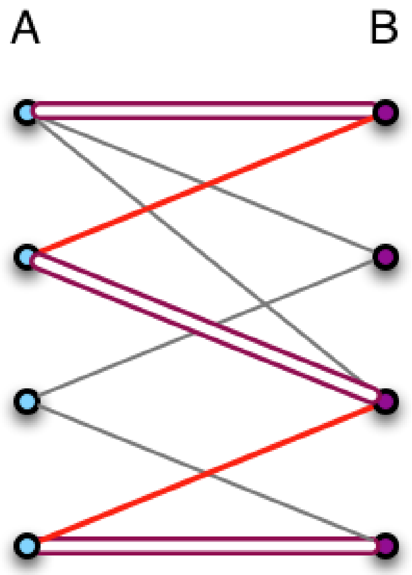
An Example



M



P



$M' = M \oplus P$

Basic Algorithm for Maximum Cardinality Matching

- Start from the empty matching
- Repeat
  - Find an augmenting paths
  - Augment along that path (non-matching edges ⇔ matching edges)
- Until there is no augmenting paths

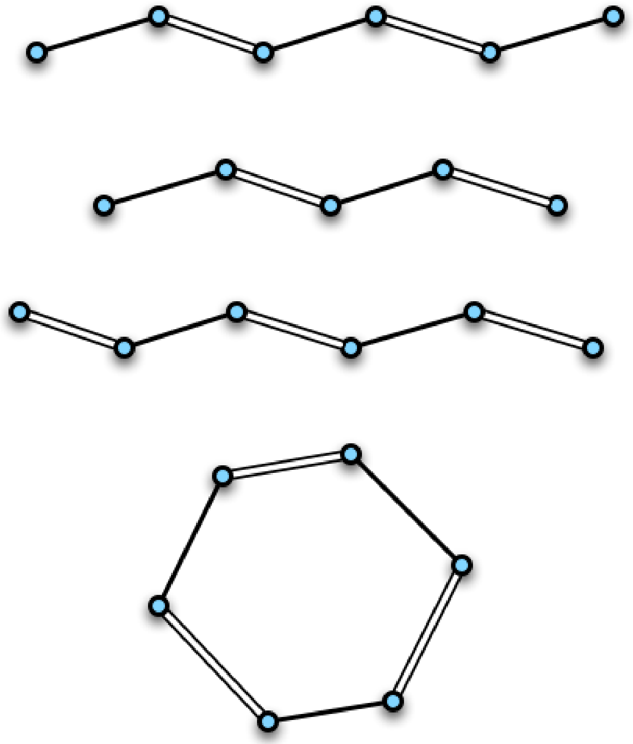
Basic Algorithm for Maximum Cardinality Matching

- Start from the empty matching
- Repeat
  - Find an augmenting paths
  - Augment along that path (non-matching edges ⇔ matching edges)
- Until there is no augmenting paths

- At most n iterations

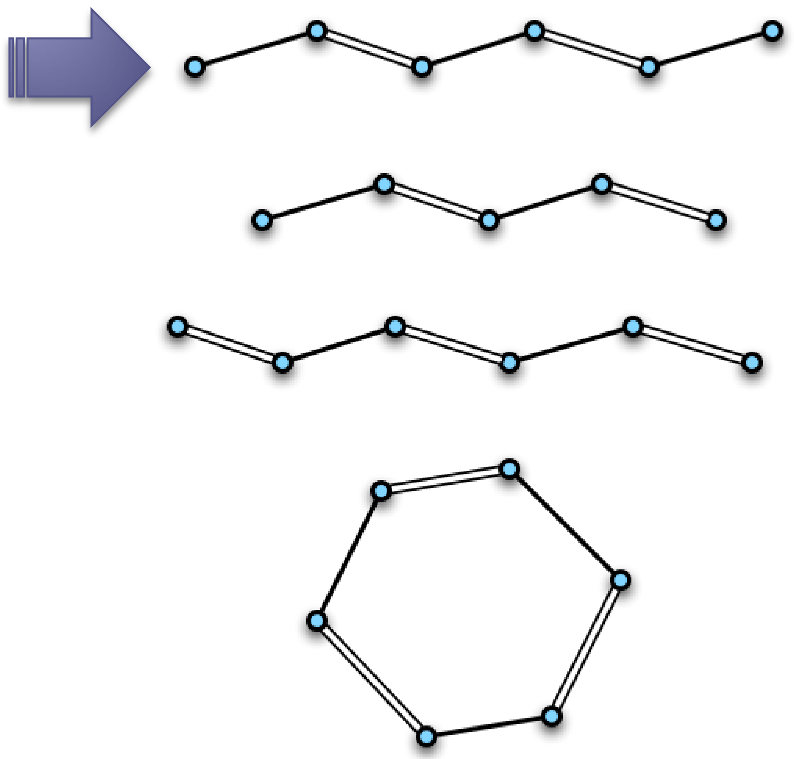
Correctness:
No augmenting path=>Maximum Matching

- Let M be the current matching and M* be the maximum matching,
- In M⊕M*, every vertex is incident with at most one edge in M and one edge in M*
- M⊕M* consists of:
    - single line: M*-edge
    - double line: M-edge

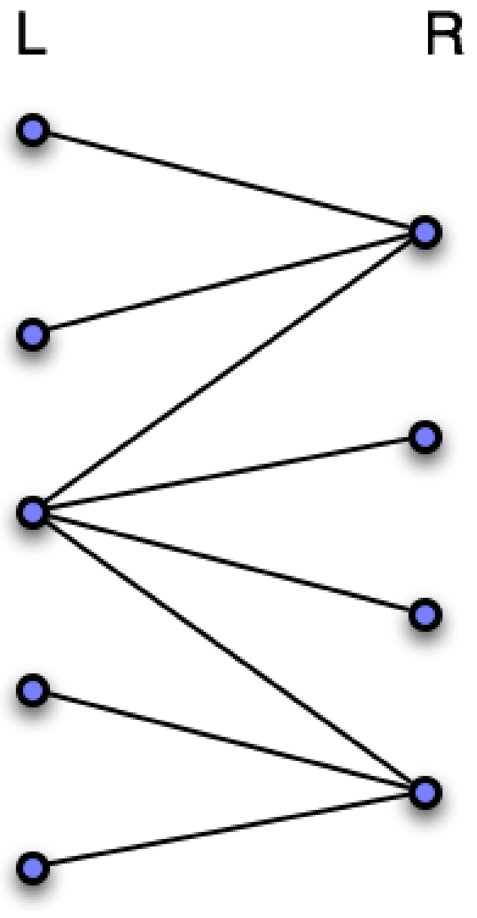


Correctness:
No augmenting path=>Maximum Matching

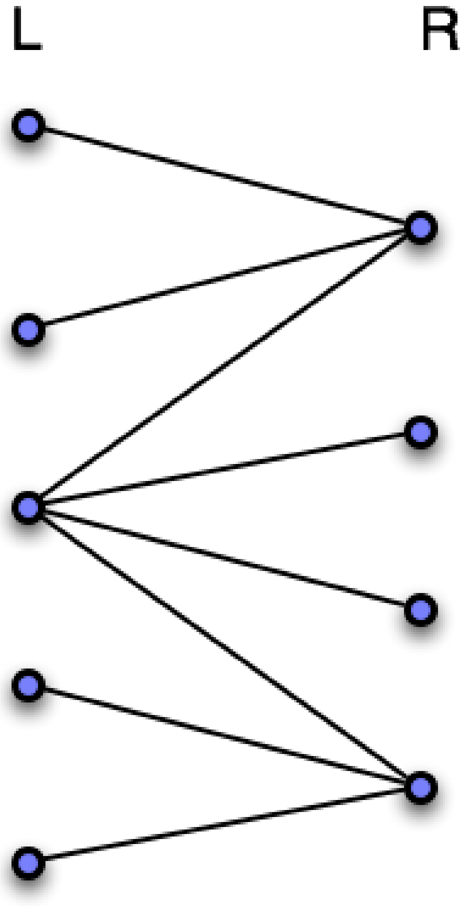
- If |M*|>|M|, there must a path of the first type
- It is an augmenting path w.r.t. M

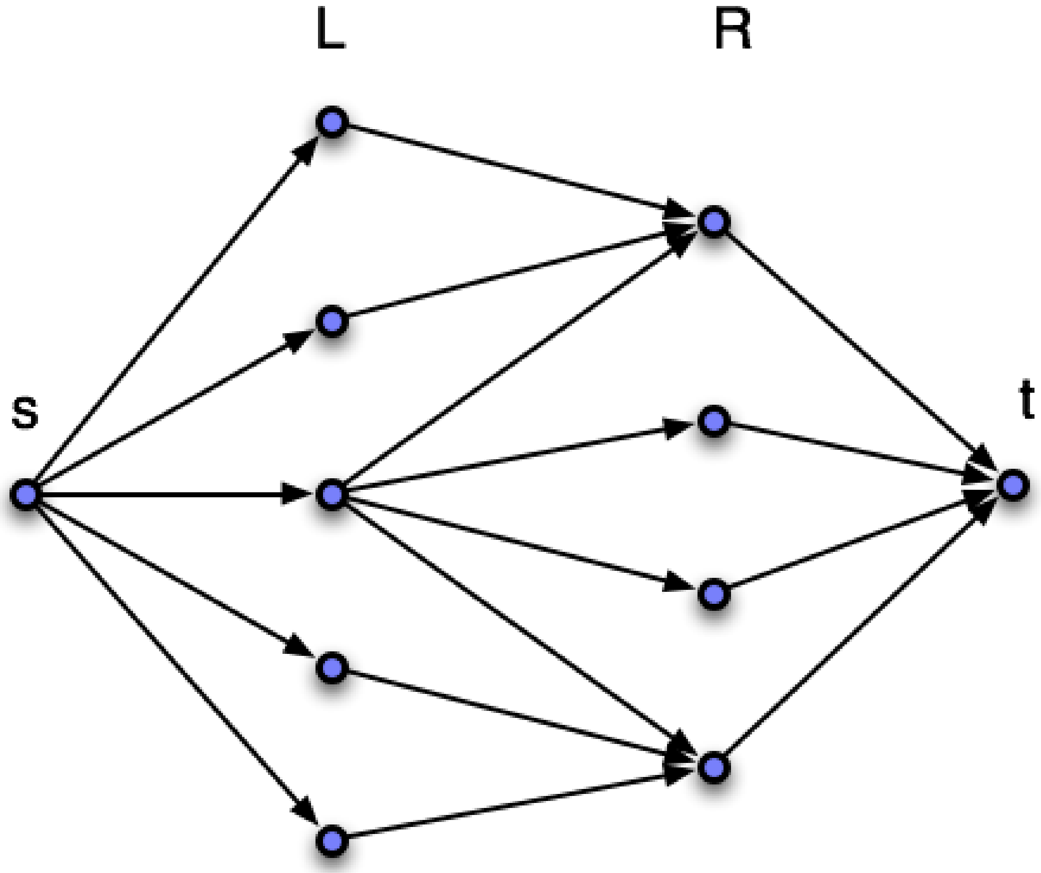


Bipartite Graphs

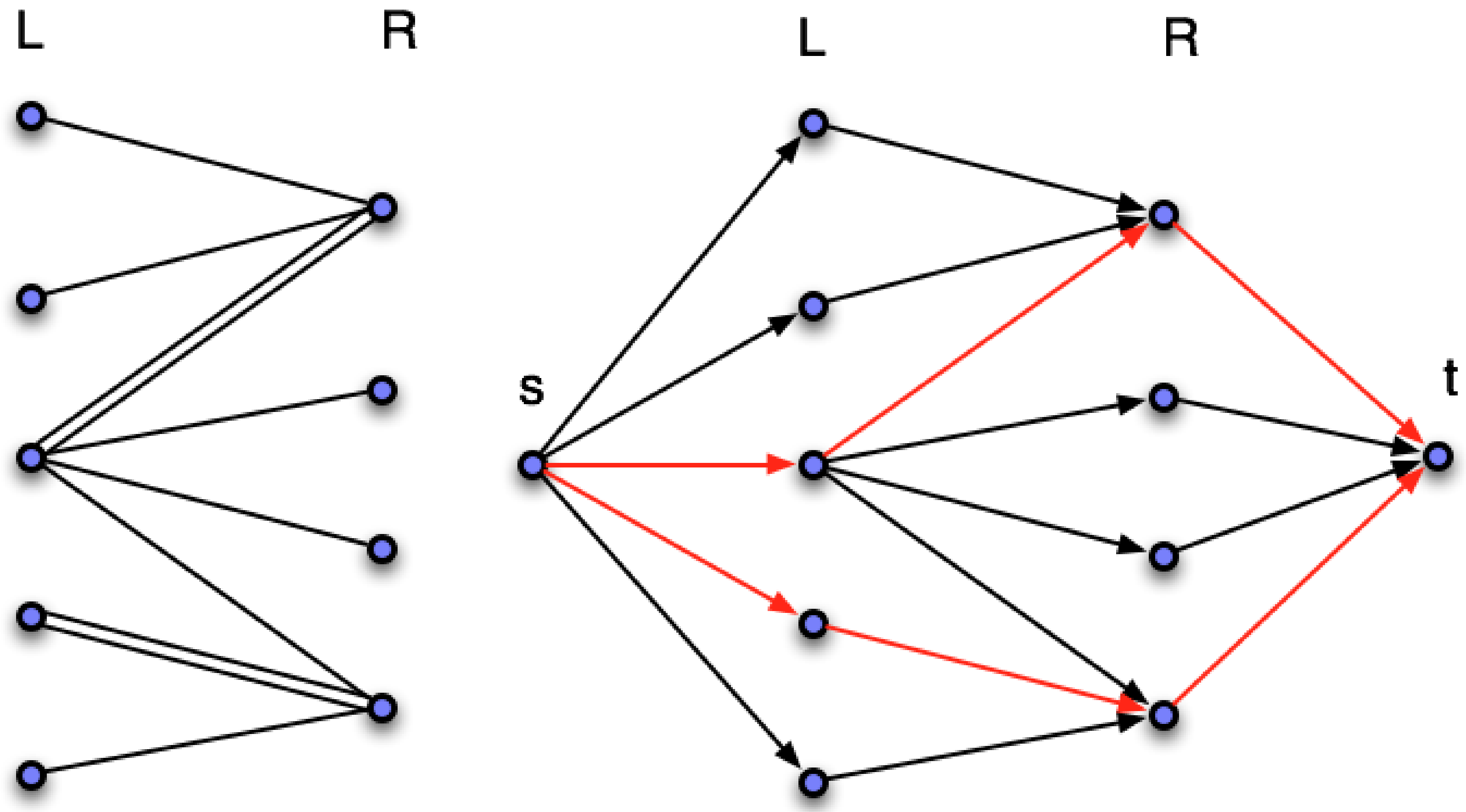


Reduce Bipartite Matching to Flow

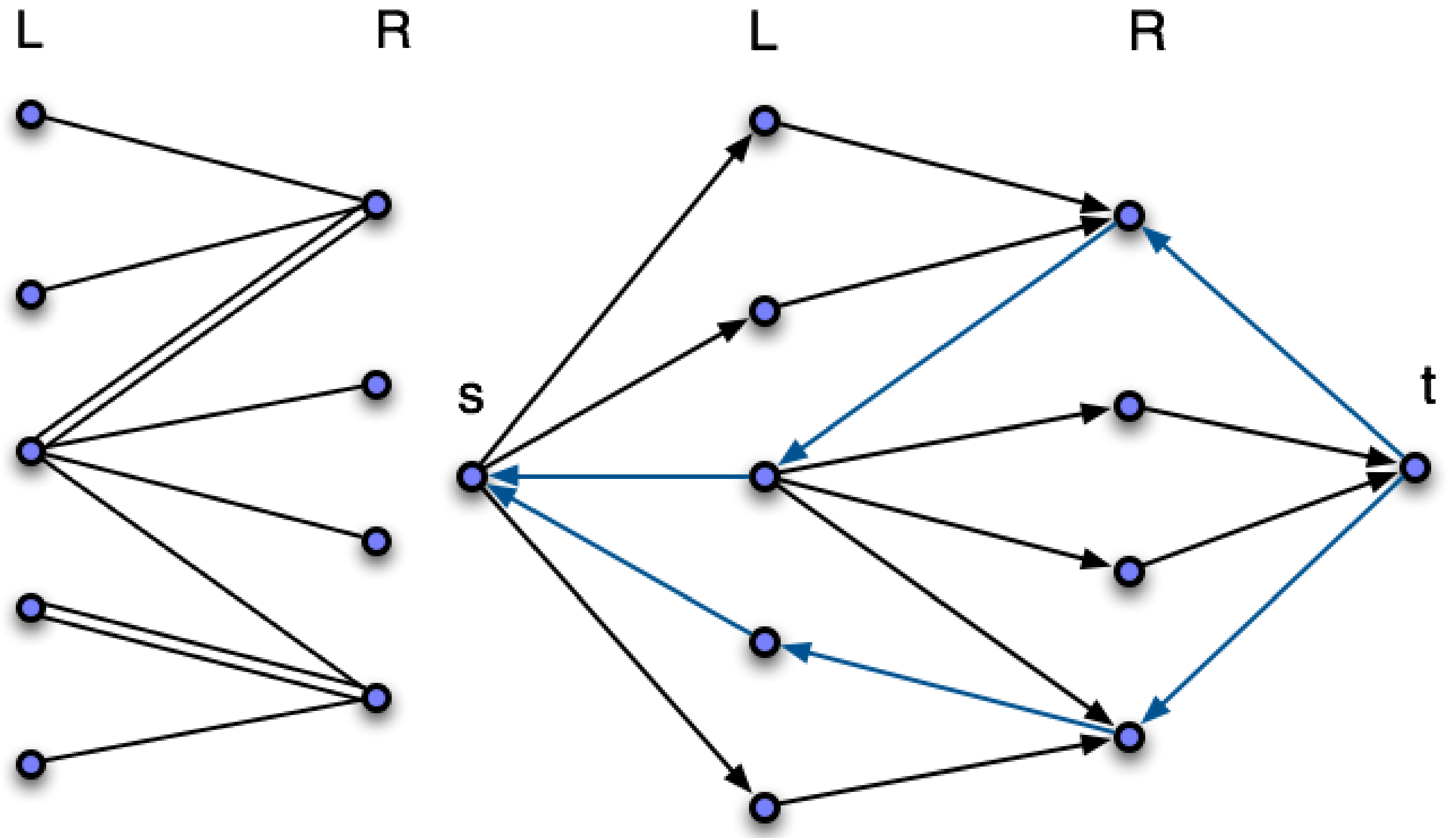




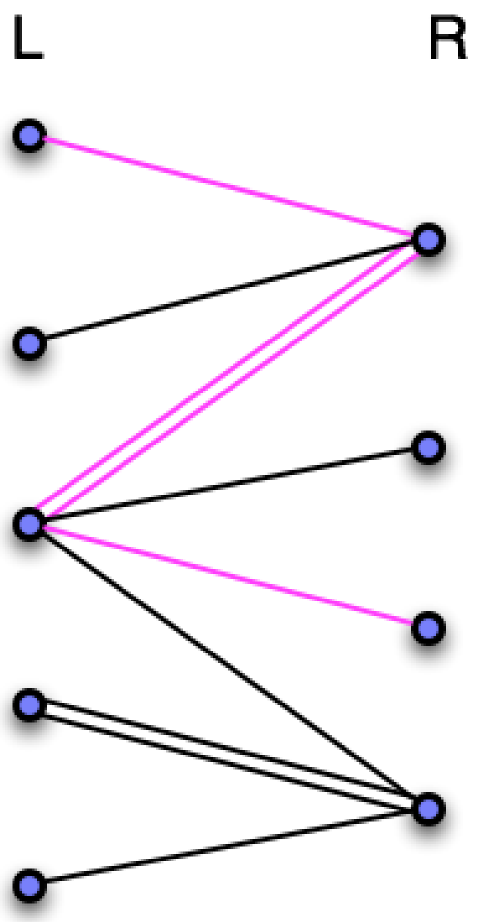
Reduce Bipartite Matching to Flow

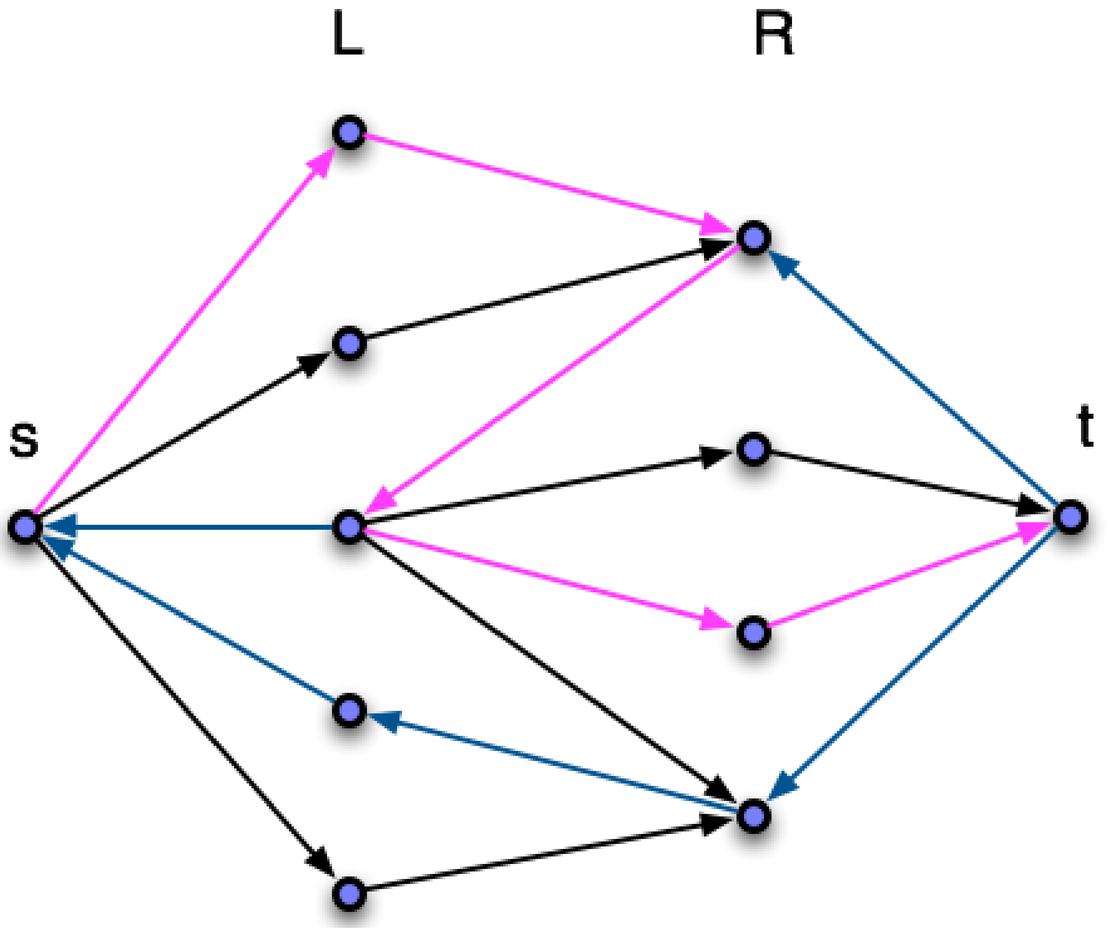


The residual graph



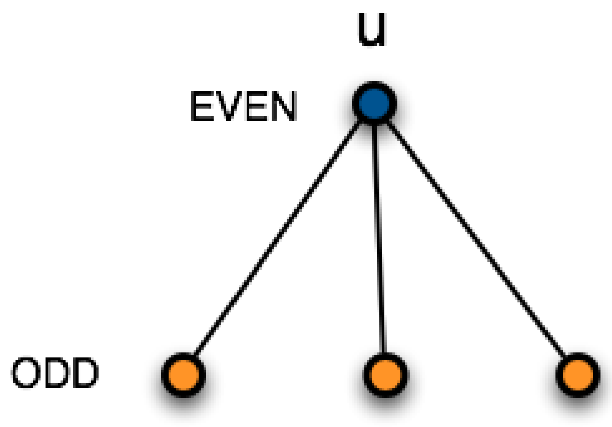
Augmenting path





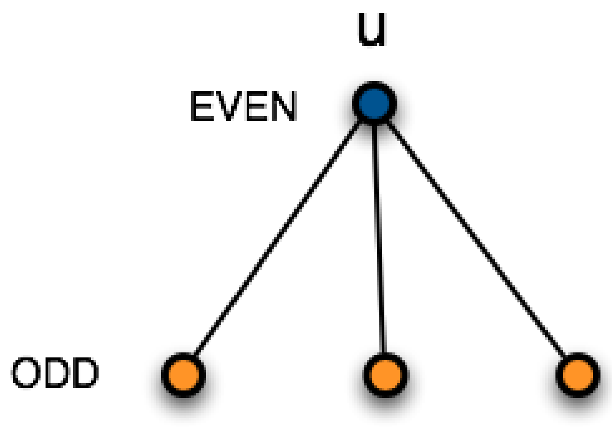
Alternating trees

- Start from a free vertex on the left side, mark it as "EVEN"
- Search all the unmatched edges associated with it, mark the vertices on the other sides by "ODD"



Alternating trees

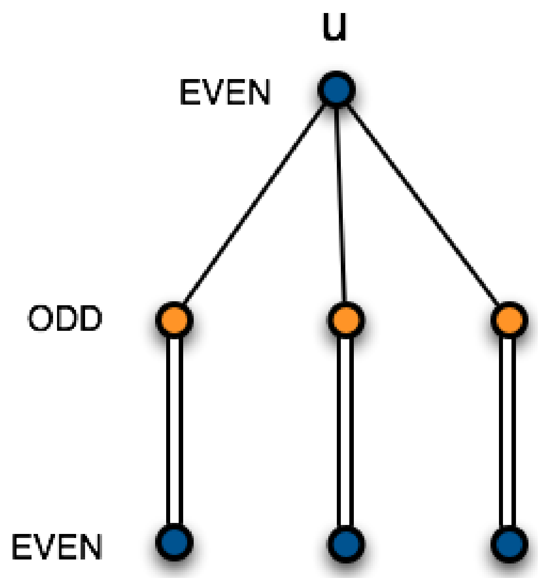
- Start from a free vertex on the left side, mark it as "EVEN"
- Search all the unmatched edges associated with it, mark the vertices on the other sides by "ODD"



Note that EVEN vertices are in L, and ODD vertices are in R.

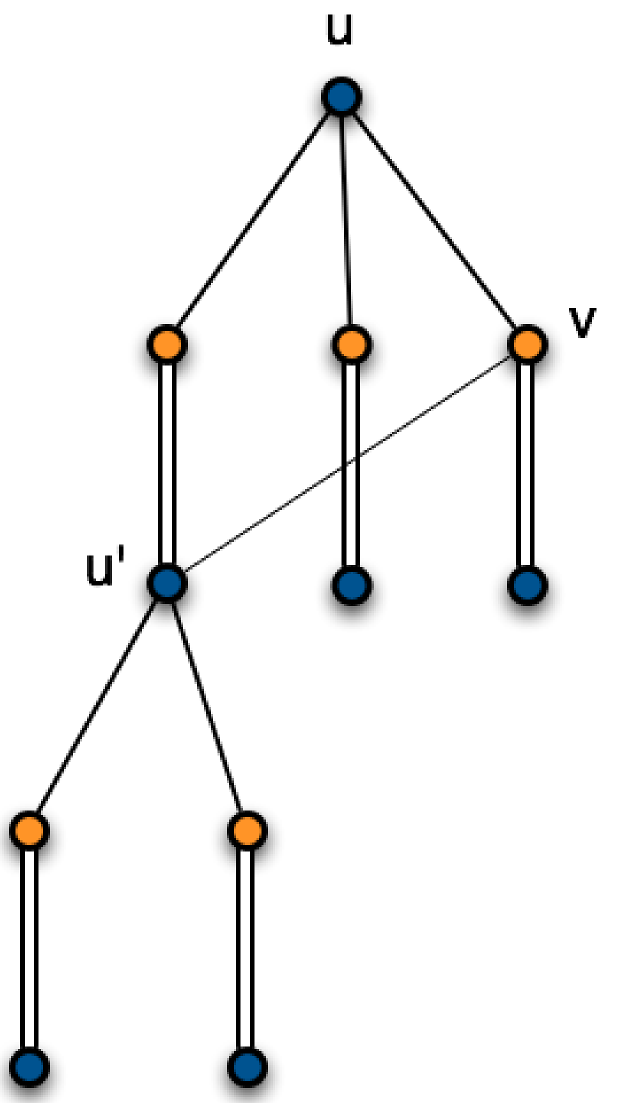
Alternating Trees

- If we find an ODD vertex that is free, then we have found an augmenting path
- If an ODD vertex v is marked and matched, mark its matched vertex w by "EVEN", then start from w.



Continue

- If v is already marked "ODD", do nothing



- When there is no augmenting path from a free vertex, which means all the vertices on the alternating tree of that free vertex are not on an augmenting paths, so we can erase the entire tree and start from other free vertices.

- When there is no augmenting path from a free vertex, which means all the vertices on the alternating tree of that free vertex are not on an augmenting paths, so we can erase the entire tree and start from other free vertices.

- Every edge is scanned only once.
- The time for finding an augmenting path is O(m).

- When there is no augmenting path from a free vertex, which means all the vertices on the alternating tree of that free vertex are not on an augmenting paths, so we can erase the entire tree and start from other free vertices.

- Every edge is scanned only once.
- The time for finding an augmenting path is O(m).
- Total running time: O(mn)

The Hopcroft-Karp Algorithm for Bipartite Graphs

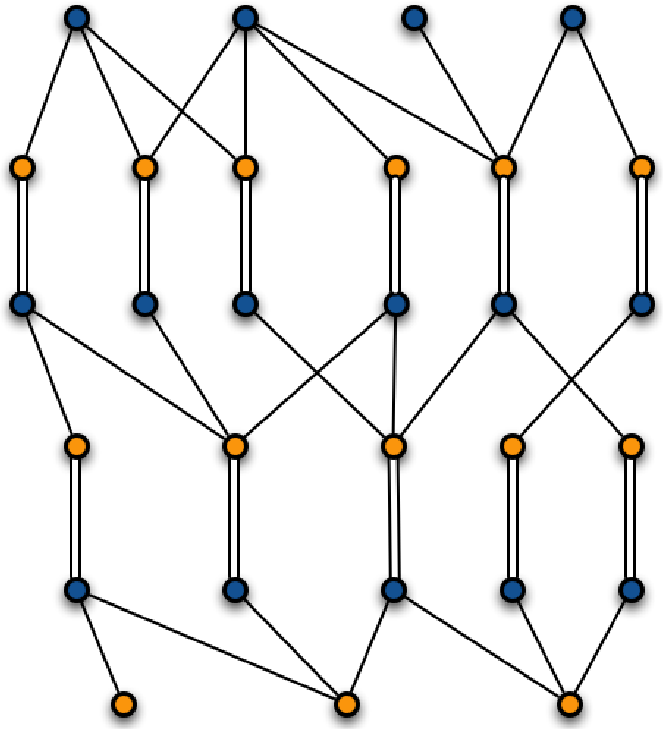
- Find a maximal set of shortest augmenting paths in one search
- The length of augmenting paths will increase after augmentation.
- After $n^{1/2}$ iterations, the length of augmenting paths will be at lease $n^{1/2}$, so there are at most $n^{1/2}$ free vertices left.
- Total time: $O(mn^{1/2})$

Level Graph

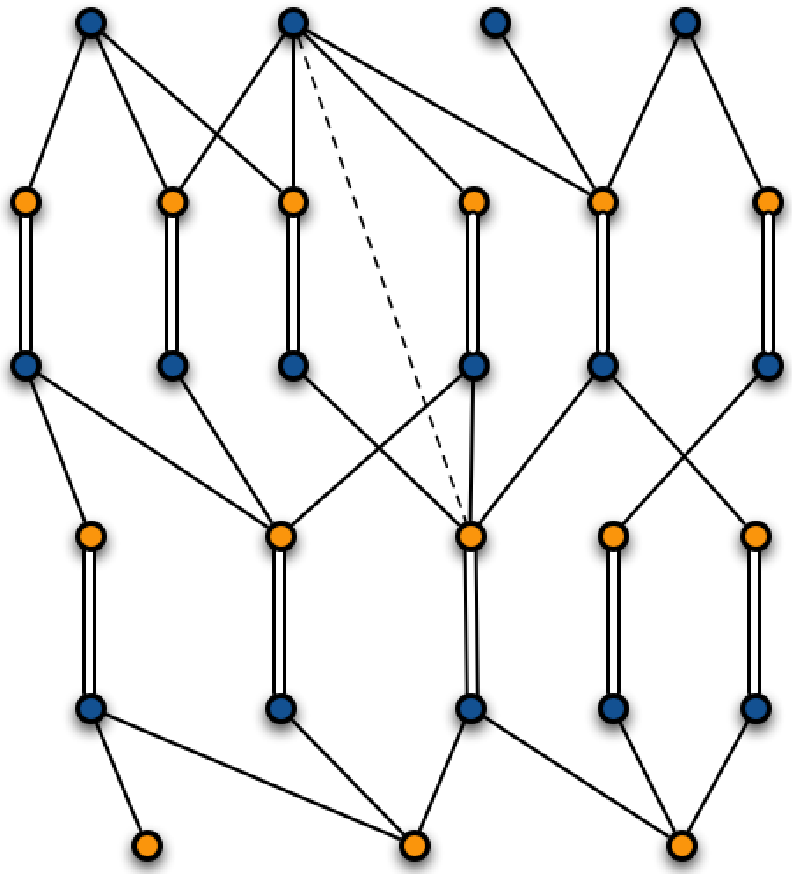
- Start from all the free vertices in L
- Find the length l of shortest augmenting paths

Level Graph

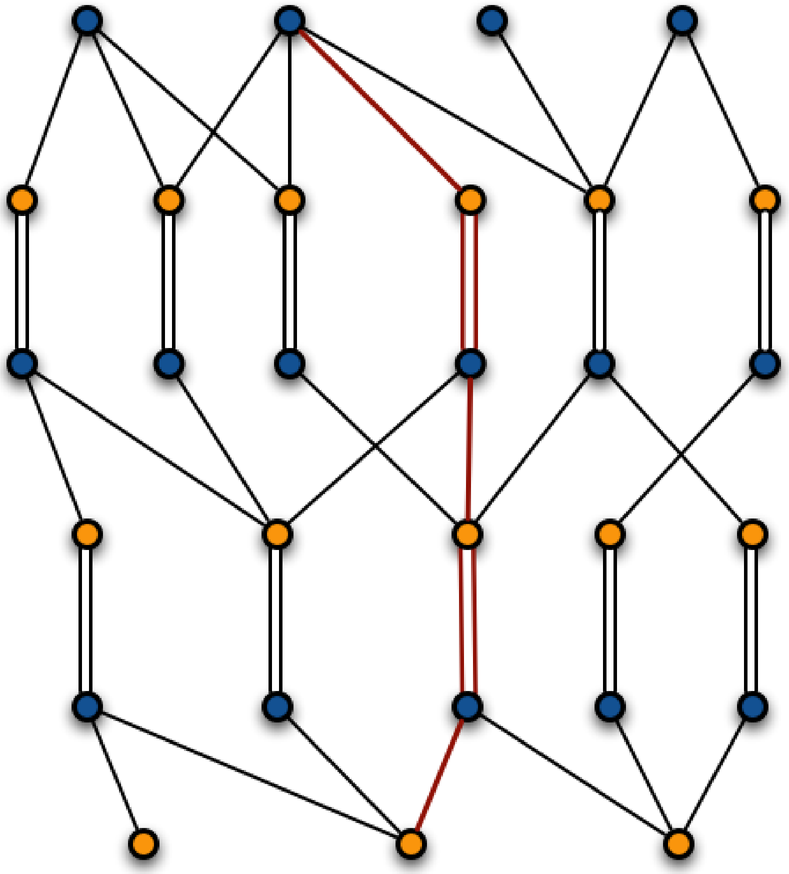
- In the level graph L, every vertex and edge is on a shortest augmenting path from s to t.
  - The edges are only between EVEN vertices and ODD vertices in adjacent levels.



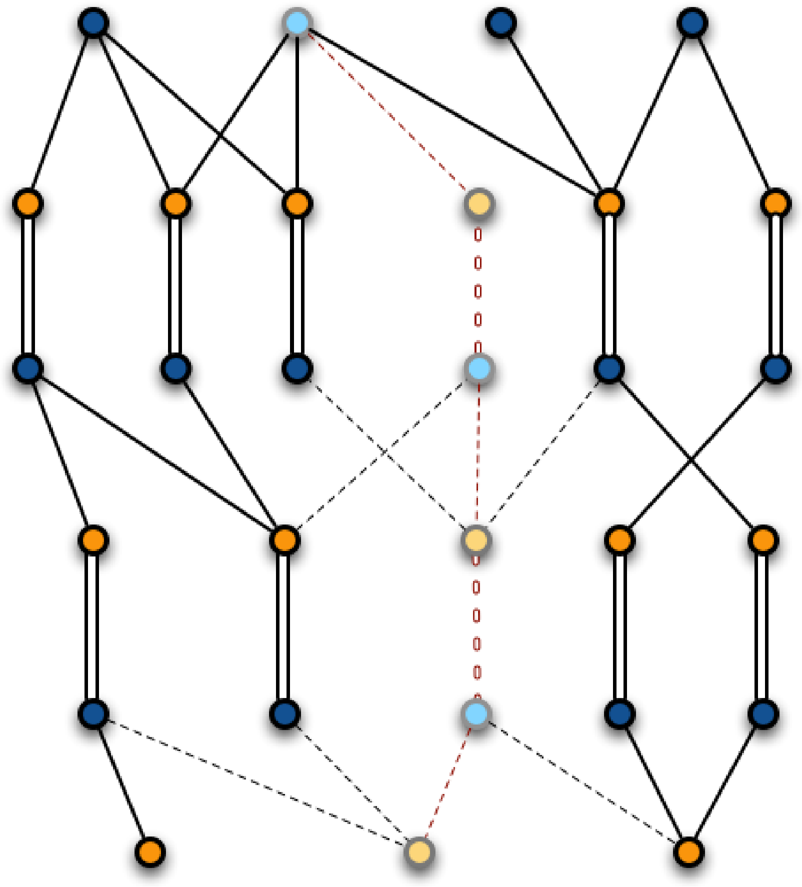
- There is no edge like this:



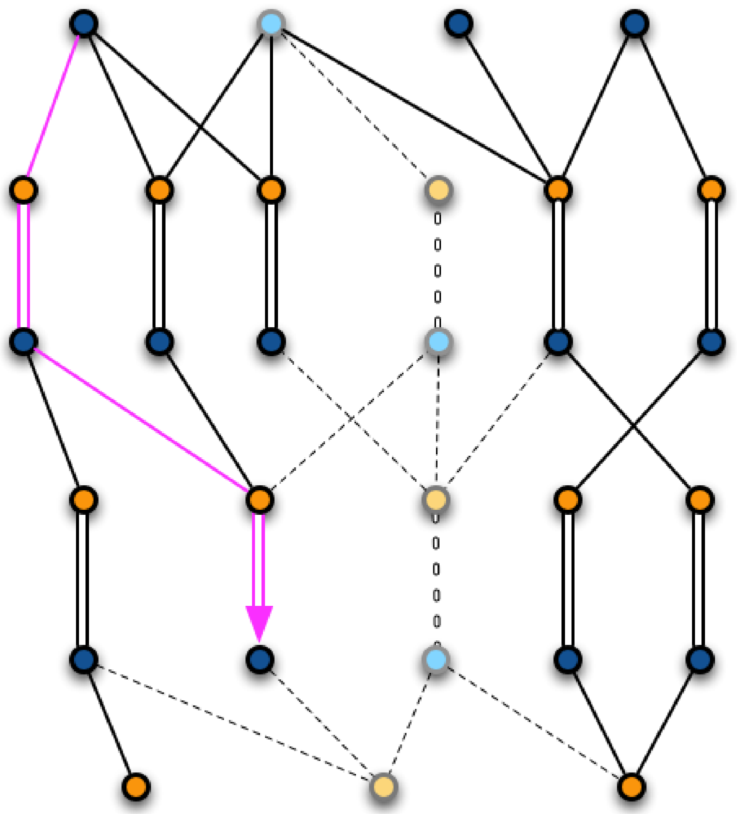
- Find all the shortest augmenting paths by a Depth-First Search in this level graph.
  - When we find an augmenting path, delete all the vertices and edges in it.



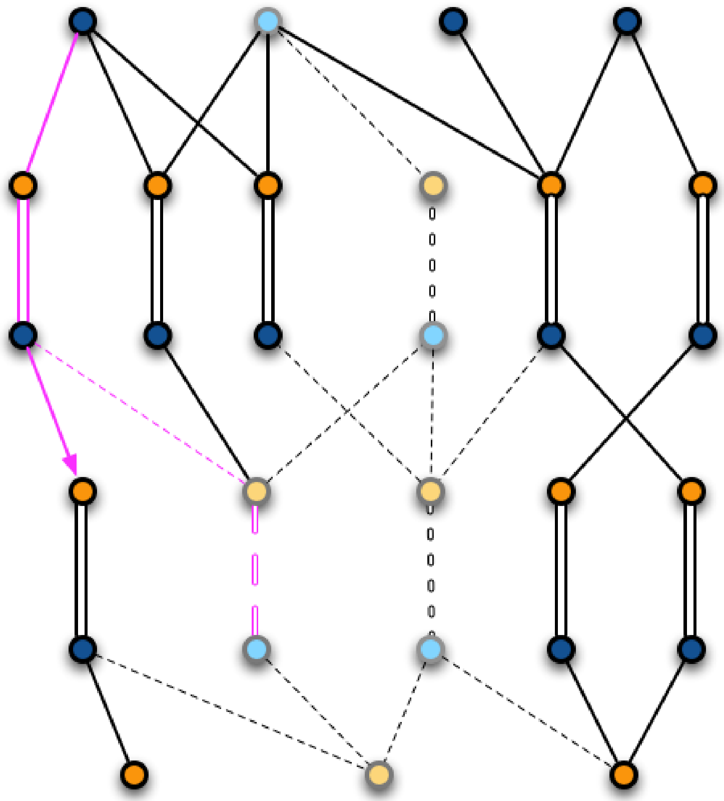
- Find all the shortest augmenting paths by a Depth-First Search in this level graph.
  - When we find an augmenting path, delete all the vertices and edges in it.



- Find all the shortest augmenting paths by a Depth-First Search in this level graph.
  - When we find an augmenting path, delete all the vertices and edges in it.
  - Delete the backtracking edges



- Find all the shortest augmenting paths by a Depth-First Search in this level graph.
  - When we find an augmenting path, delete all the vertices and edges in it.
  - Delete the backtracking edges



- Find all the shortest augmenting paths by a Depth-First Search in this level graph.
    - When we find an augmenting path, delete all the vertices and edges in it.
    - Delete the backtracking edges

- We can find a maximal set Ω of augmenting paths.
- So every edge is visited only once, and the time needed for this is O(m)

Proof

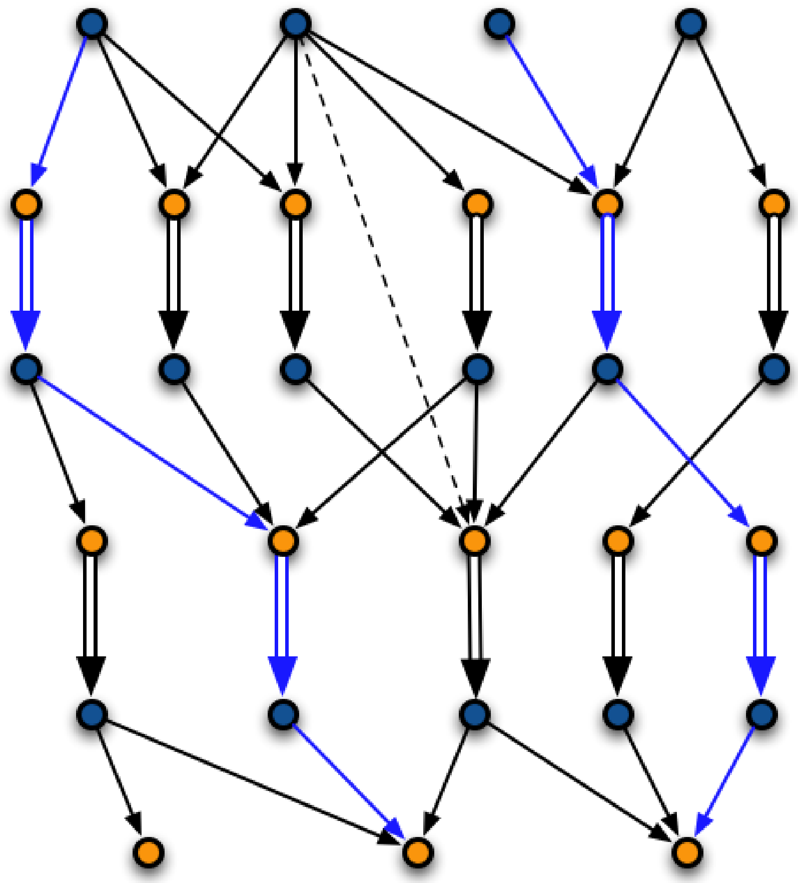
- Lemma: After augmentation of these paths, there is no augmenting paths of length at most l any more.

- We assign directions to edges and get G':
    - Unmatching edges: from L to R
    - Matching edges from R to L
- A path between free vertices of L and R in G' ⇔
  An augmenting path in G

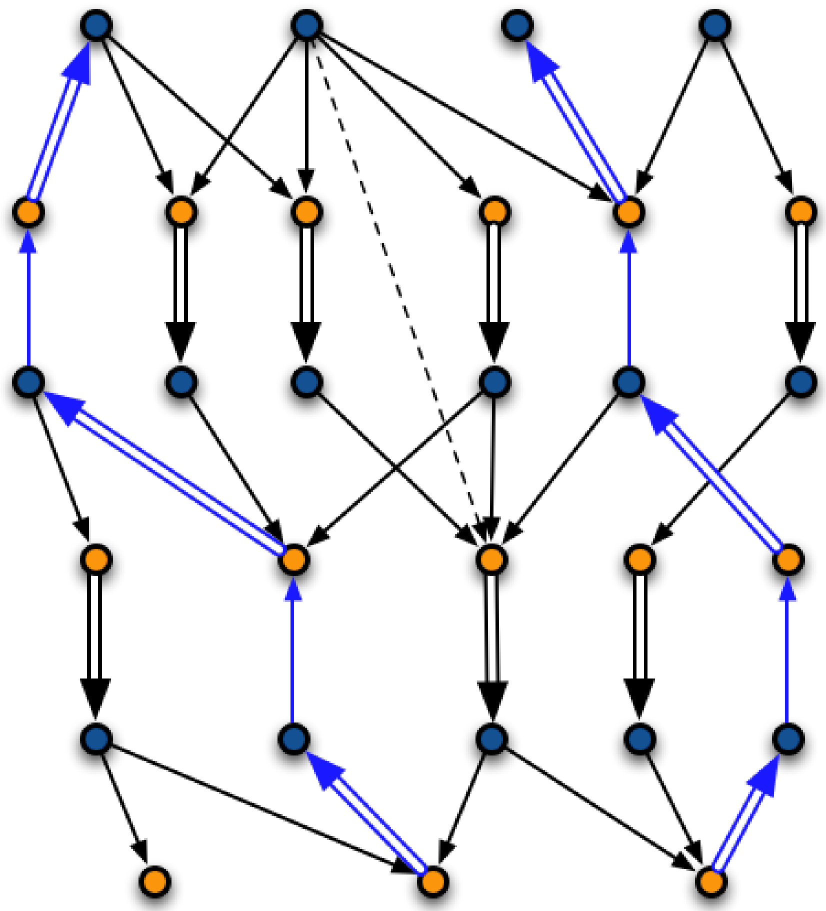
- There is no edges from level i to level i+2 or higher.
- After augmentation, the edges in the augmenting paths reverse directions
- So if there is still an augmenting path, it cannot overlap other augmenting paths, contradicting that we have found a maximal set of augmenting paths.

Example

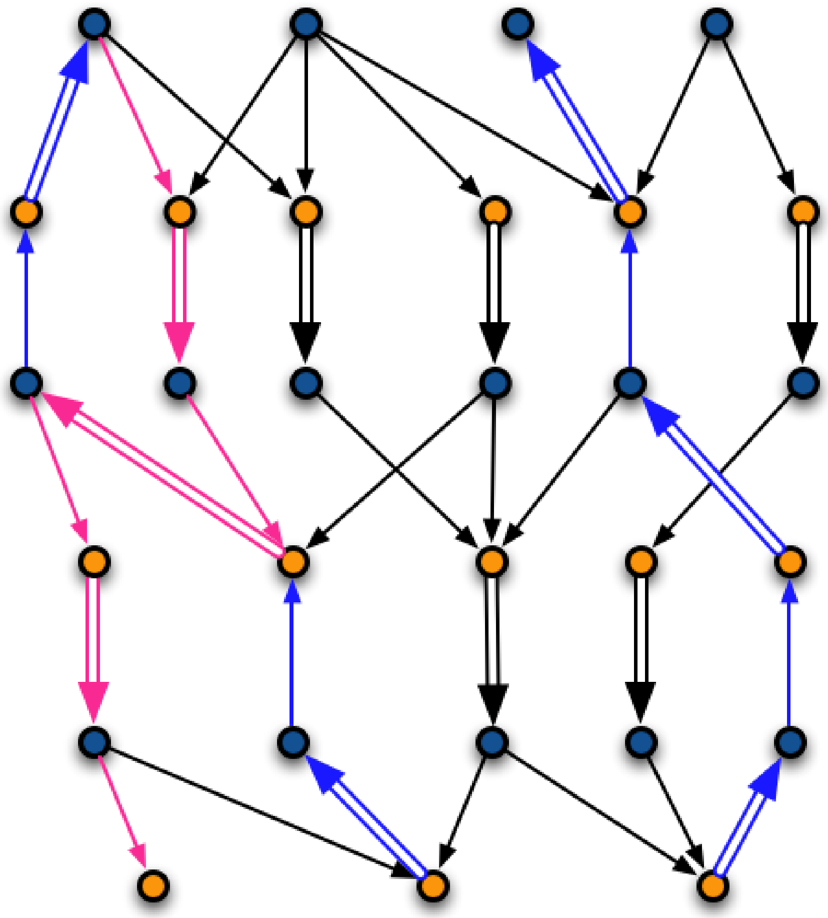
- Ω:



Example



- Then any path overlapping Ω will have longer length



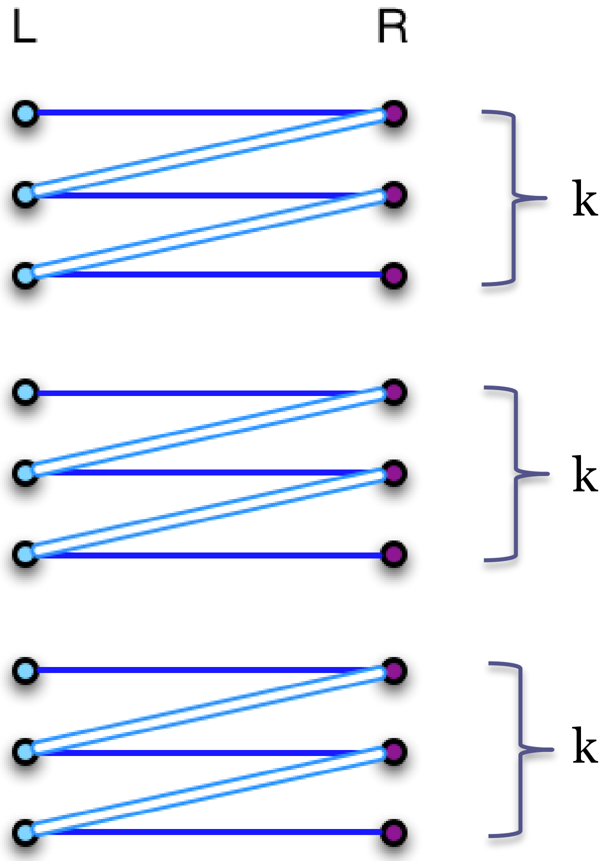
The Hopcroft-Karp Algorithm for Bipartite Graphs

- Find a maximal set of shortest augmenting paths in one search
- The length of augmenting paths will increase after augmentation.
- After $n^{1/2}$ iterations, the length of augmenting paths will be at lease $n^{1/2}$, so there are at most $n^{1/2}$ free vertices left.

The Hopcroft-Karp Algorithm for Bipartite Graphs

- Find a maximal set of shortest augmenting paths in one search
- The length of augmenting paths will increase after augmentation.
- After $n^{1/2}$ iterations, the length of augmenting paths will be at lease $n^{1/2}$, so there are at most $n^{1/2}$ free vertices left.

- After k iterations, the length of augmenting paths will be at lease k
- If we compare the current matching M and the maximum matching M*, we can get |M*|-|M| disjoint augmenting paths
- So |M*|-|M|≤|M*|/k≤n/k



- After k iterations, the length of augmenting paths will be at lease k
- If we compare the current matching M and the maximum matching M*, we can get |M*|-|M| disjoint augmenting paths
- So $|M^*|-|M| \leq |M^*|/k \leq n/k$

- When $k=n^{1/2}$, $|M^*|-|M| \leq n^{1/2}$, so we only need to find $n^{1/2}$ augmenting paths. So the running time is $O(mn^{1/2})$.

Approximate Matching

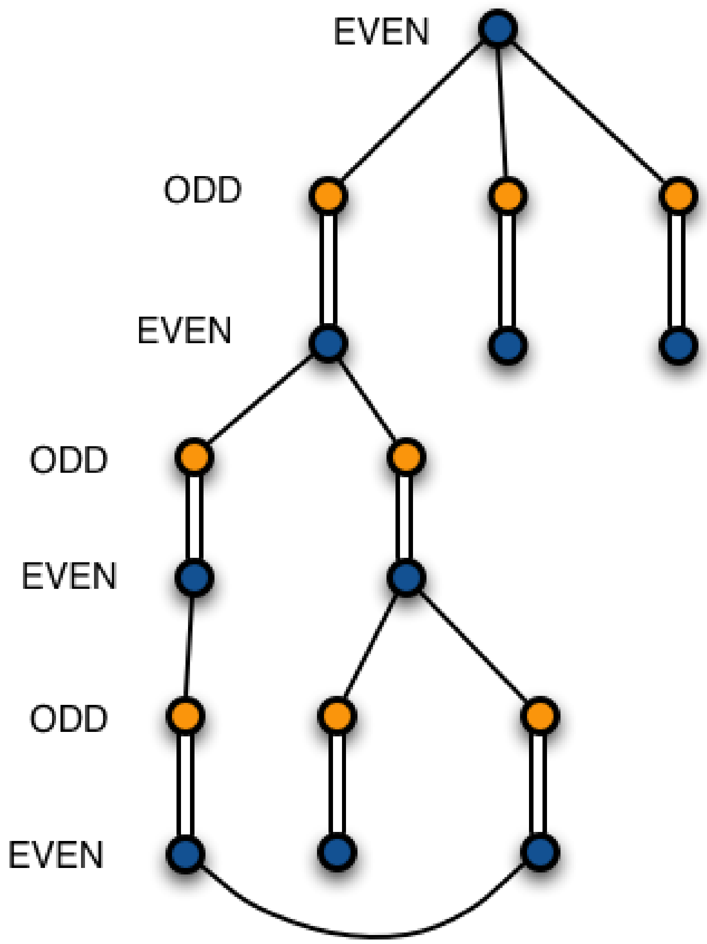
- After k iterations, the length of augmenting paths will be at lease k
- If we compare the current matching M and the maximum matching M*, we can get $|M^*|-|M|$ disjoint augmenting paths
- So $|M^*|-|M|\leq|M^*|/k$,
- and $|M|\geq|M^*|-|M^*|/k=(1-1/k)|M^*|$
- so we can get a $(1-1/k)$-approximate matching in $O(km)$ time.

Non-bipartite Matching

- New concept: Blossom

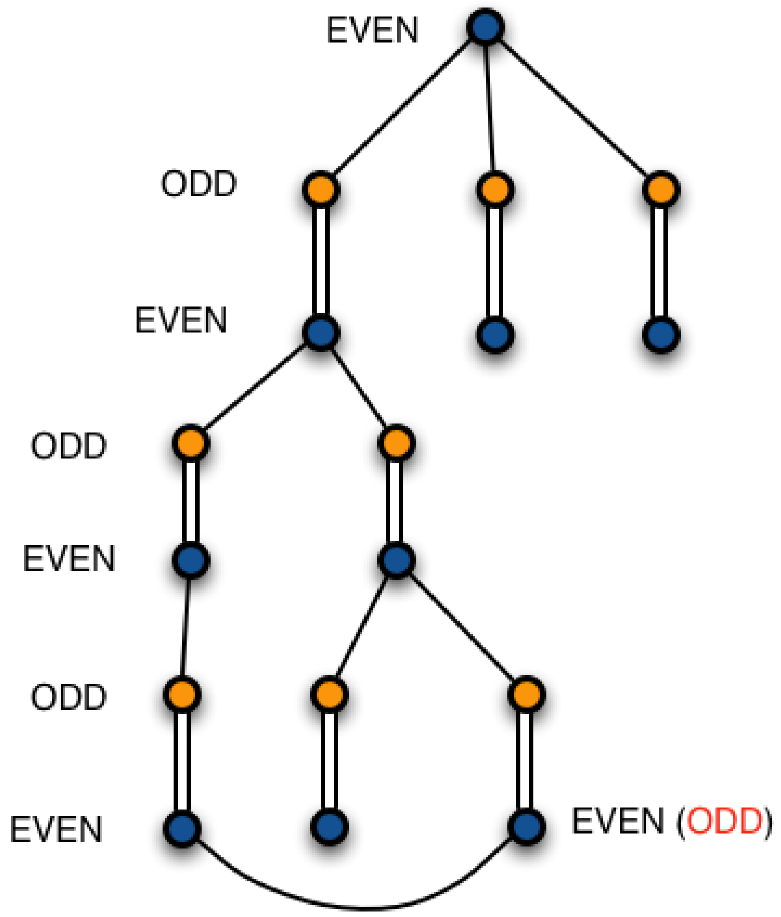
Non-bipartite Matching

- New concept: Blossom
- Why?



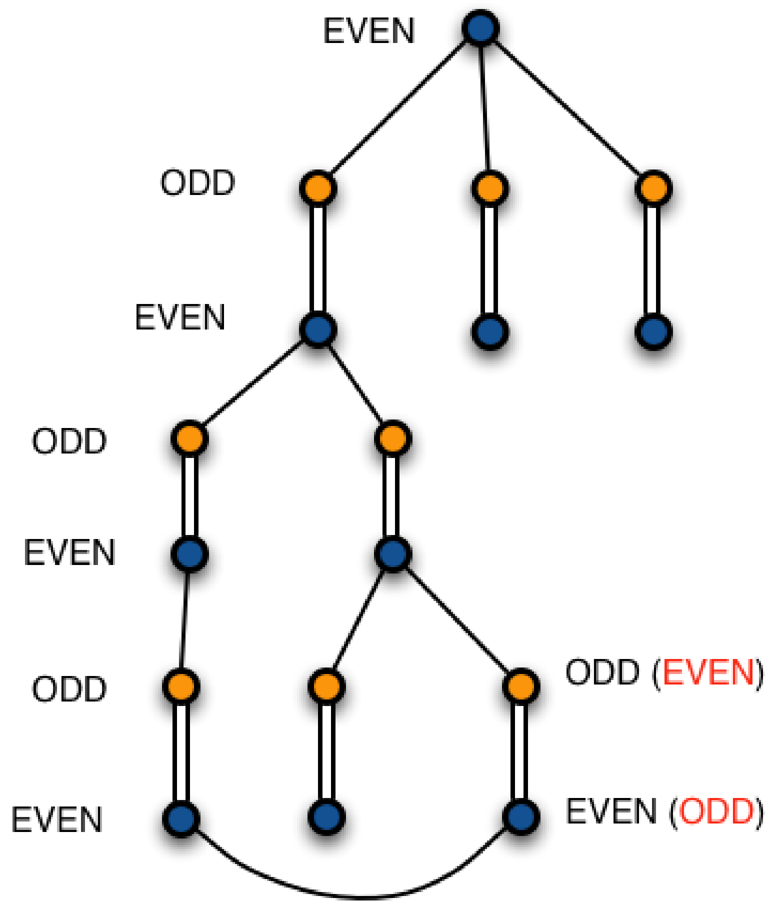
Non-bipartite Matching

- New concept: Blossom
- Why?



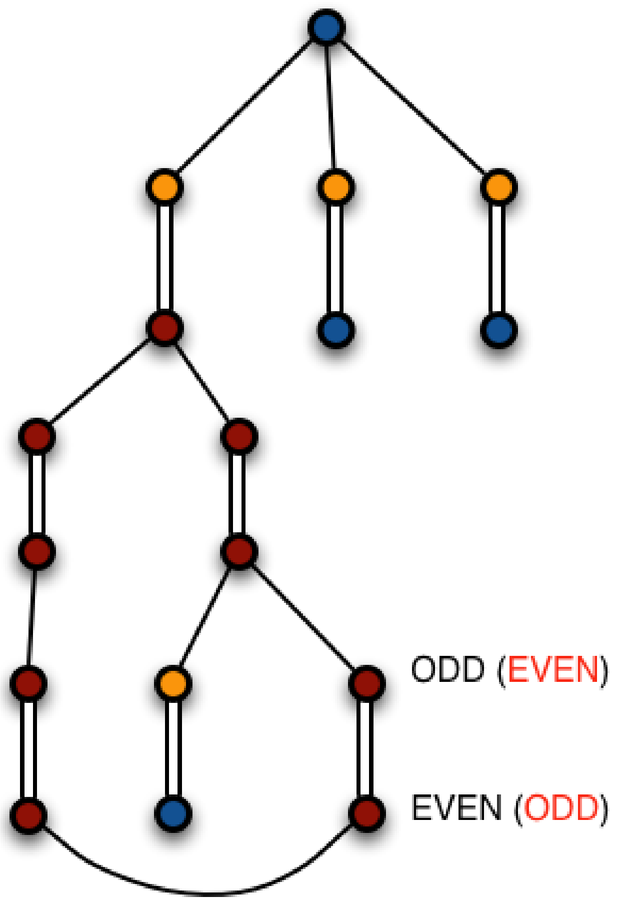
Non-bipartite Matching

- New concept: Blossom
- Why?



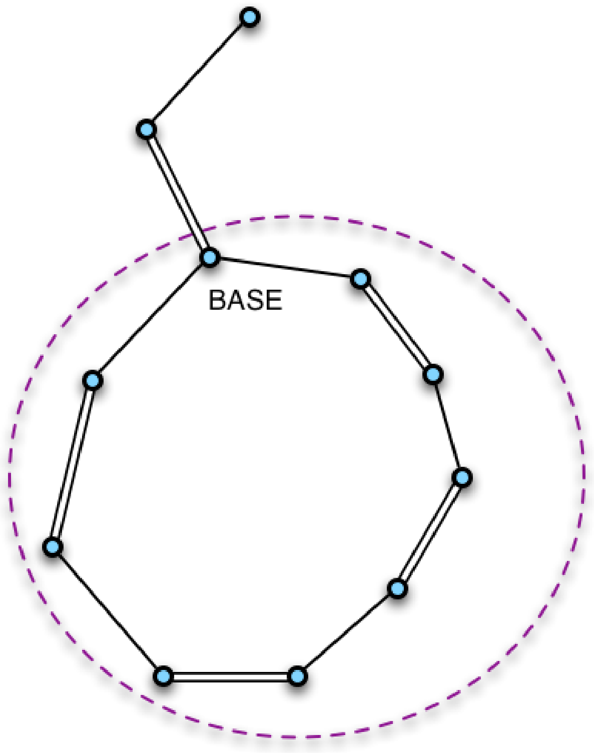
Non-bipartite Matching

- New concept: Blossom
- So all the vertices in this odd-length cycle are both EVEN and ODD, we call this kind of cycles "blossoms"



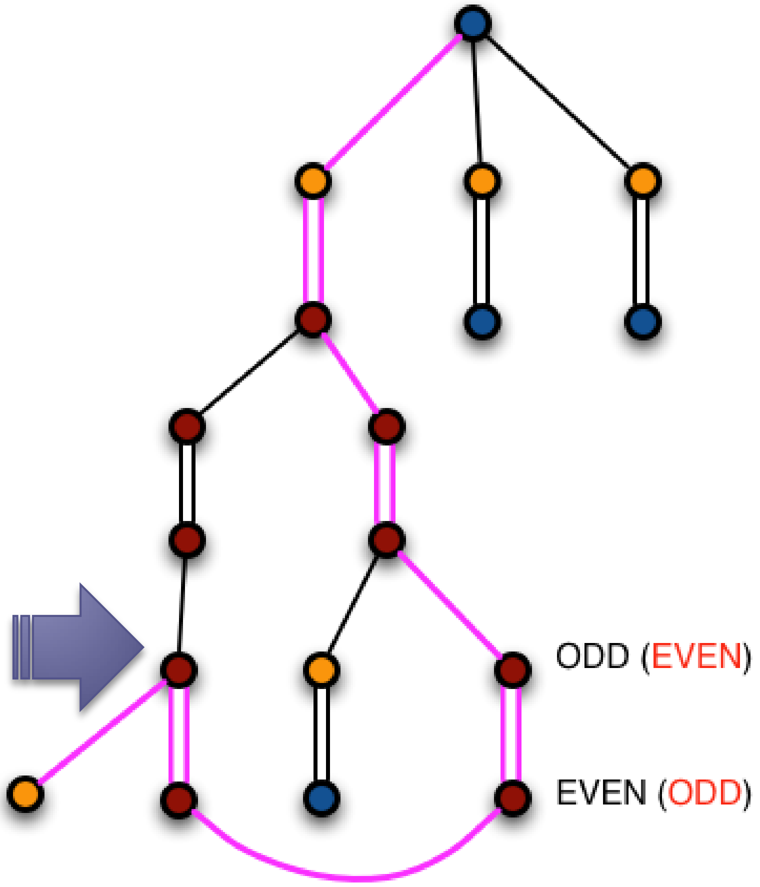
Blossom

- A blossom B is a cycle in G consisting of 2k + 1 edges of which exactly k belong to M.
- The only vertex whose matching edge is not in B is called the base.



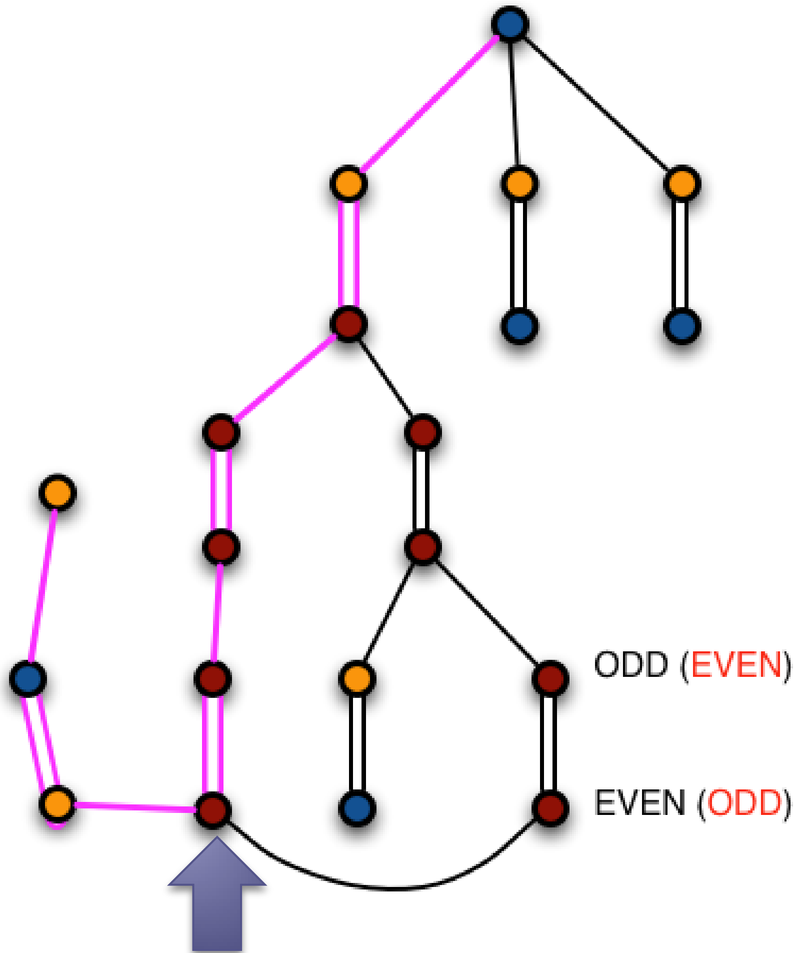
Property of blossom

- Edges associated with any vertex in the blossom can be in an augmenting path



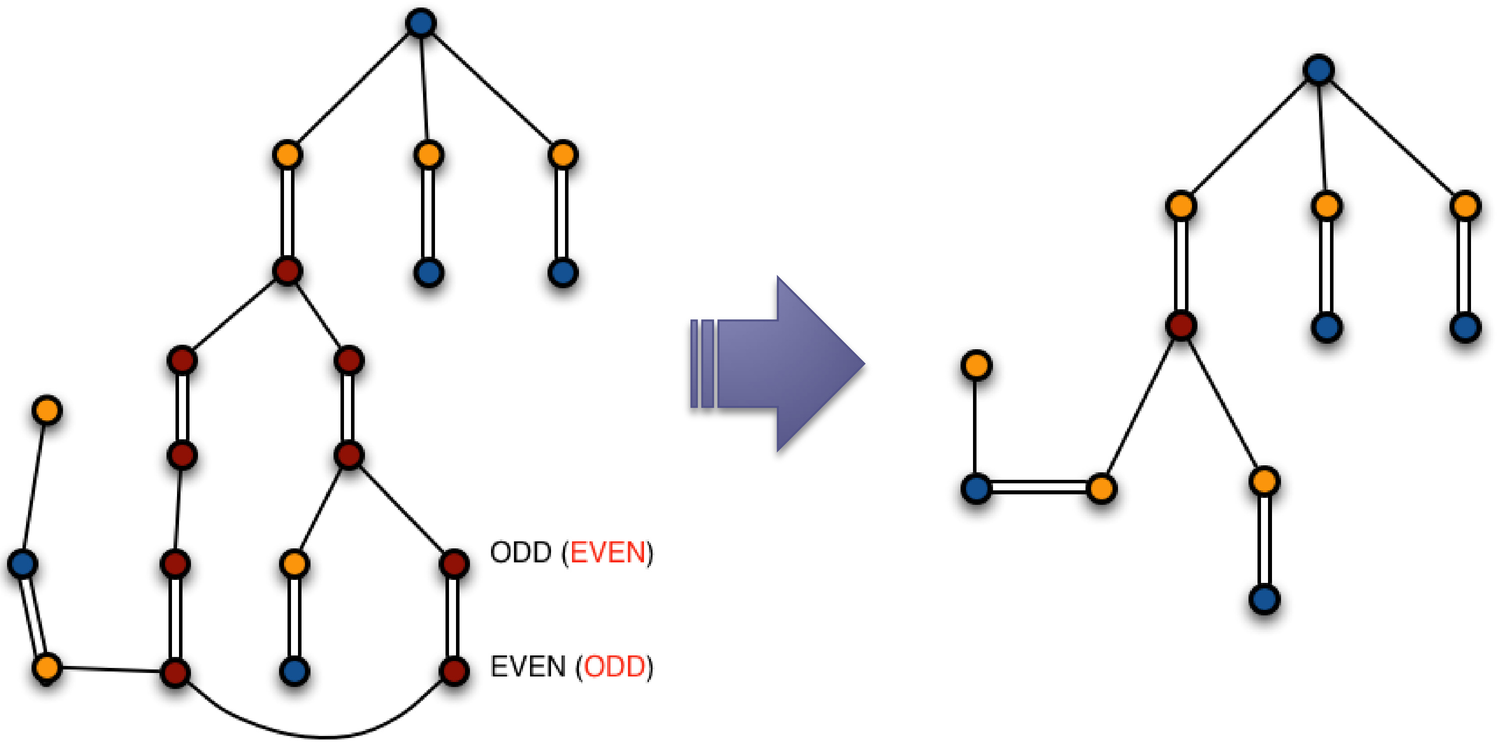
Property of blossom

- Edges associated with any vertex in the blossom can be in an augmenting path



Shrinking Blossoms

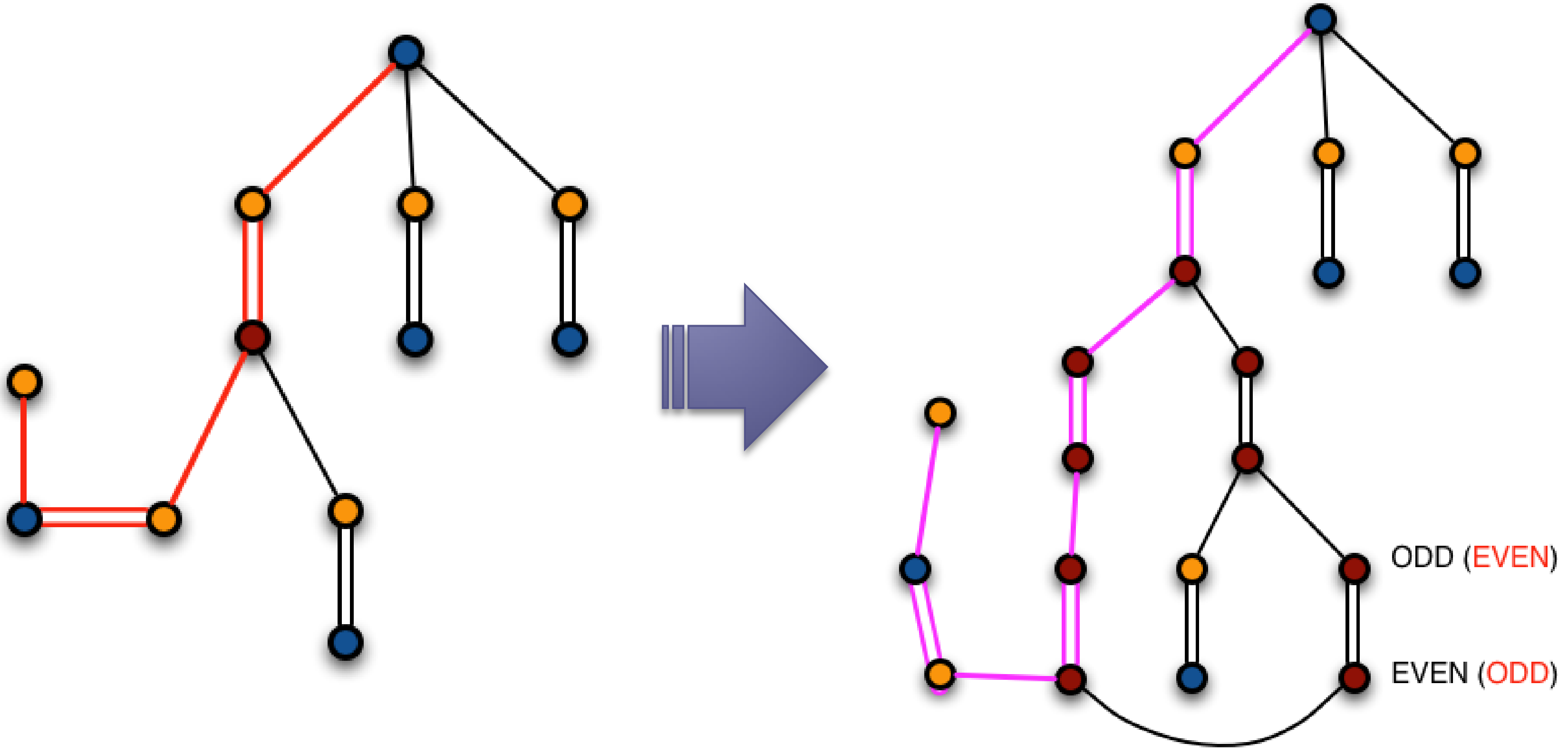
- Thus in the search blossoms can be shrunk to one vertex, we call it the contracted graph



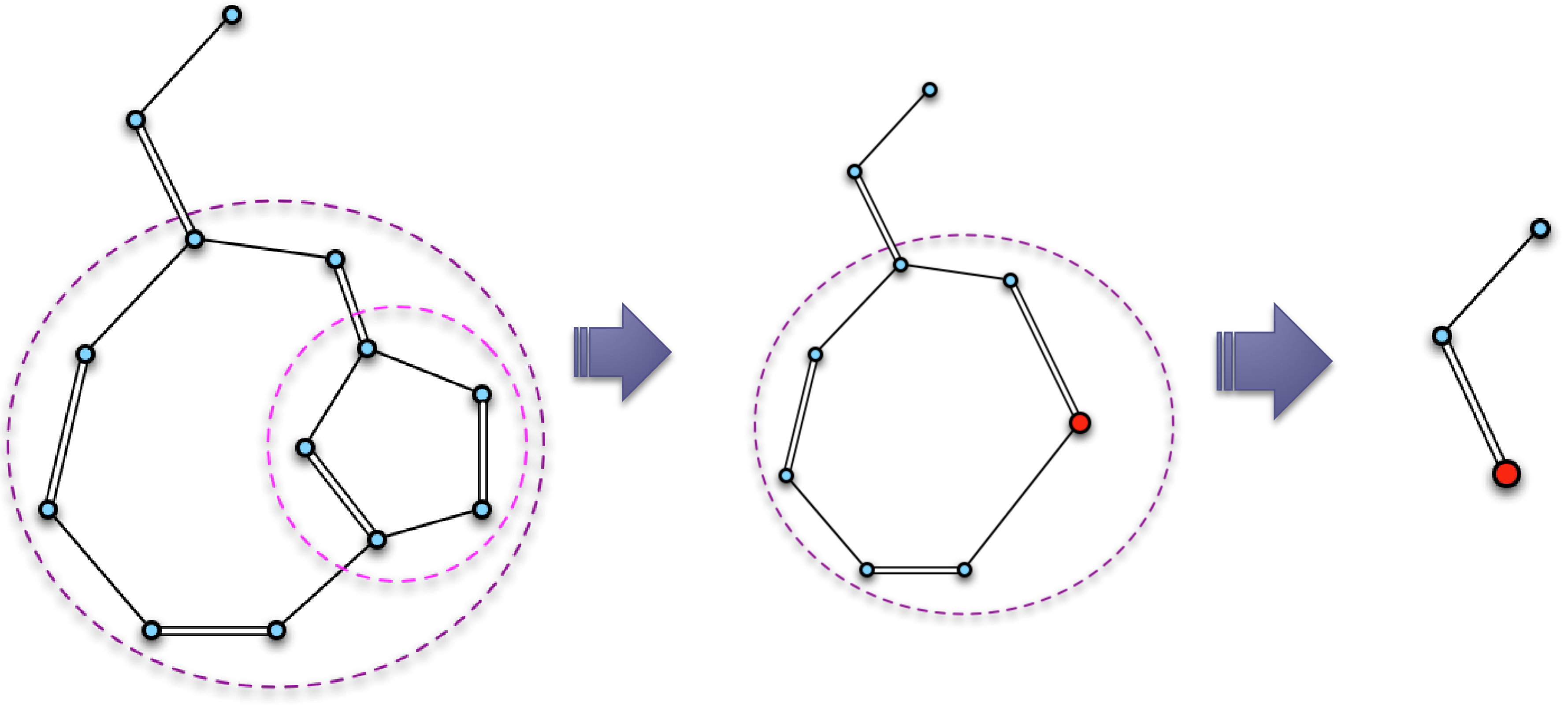
Shrinking Blossoms

- Thus in the search blossoms can be shrunk to one vertex, we call it contracted graph
- Then we find augmenting paths in the contracted graph.
- Finally we can unshrink the graph and get the real augmenting paths.

Example



- A blossom can contain other smaller blossoms

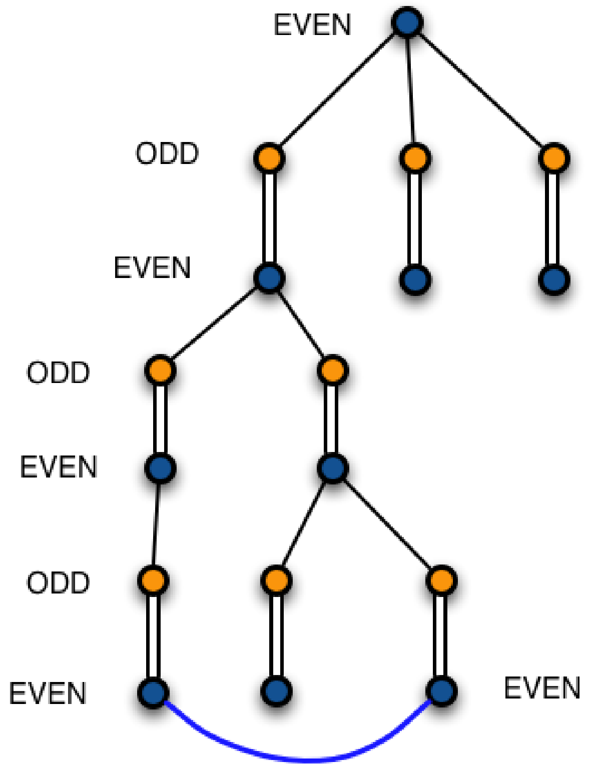


Edmonds' algorithm

- When searching an edge (v,w) where v is marked "EVEN"
  - If w is free, we have found an augmenting path
  - If w is unmarked and matched, mark it "ODD" and mark its matched vertex w' "EVEN"
  - If w is already labeled "ODD", do nothing
  - If w is already labeled "EVEN", we have found a blossom. Shrink the blossom to a single vertex and get a new graph G'. Find augmenting paths in G'.

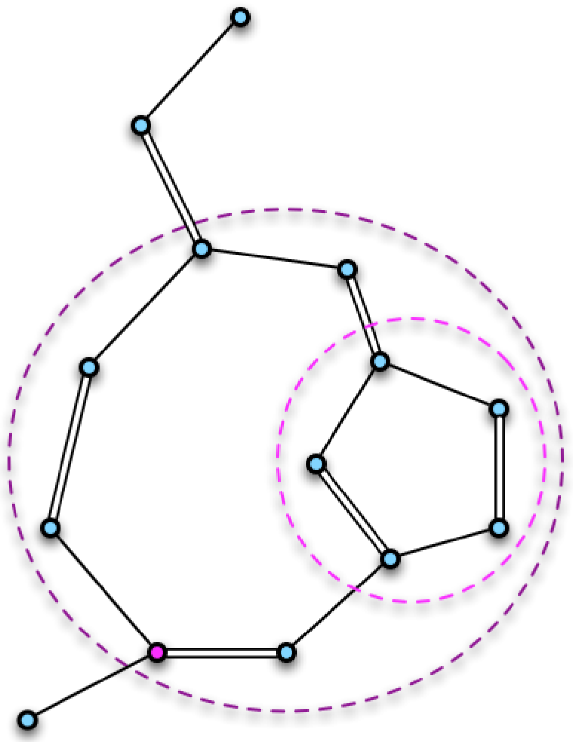
Detecting blossoms

- If w is already labeled "EVEN", we have found a blossom. Shrink the blossom to a single vertex and get a new graph G'. Find augmenting paths in G'.



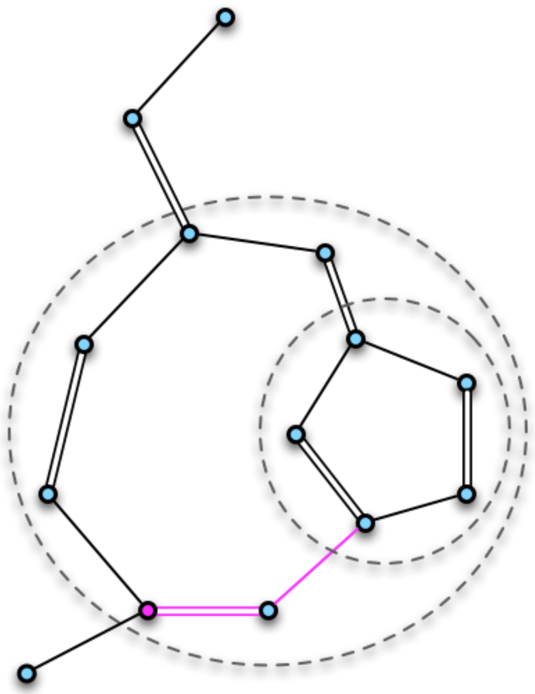
Retrieving Augmenting Path

- Retrieve the real augmenting path in G from the path in the contracted graph G':



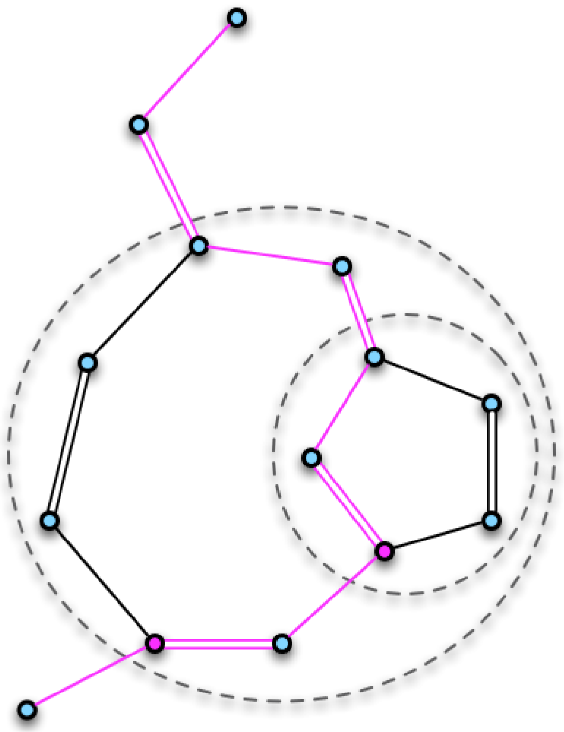
Retrieving Augmenting Path

- Retrieve the real augmenting path in G from the path in the contracted graph G':



Retrieving Augmenting Path

- Retrieve the real augmenting path in G from the path in the contracted graph G':



Running time

- In O(m) time we either find an augmenting path or a blossom.
- We may contract at most n blossoms before finding an augmenting path, so the time for an augmenting path is O(mn)

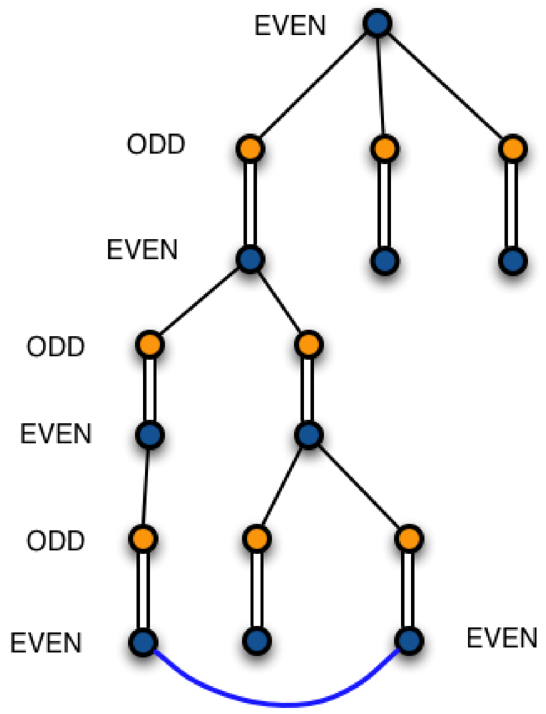
Analysis of Running Time

- Start from the empty matching
- Repeat
  - Find an augmenting paths
  - Augment along that path (non-matching edges ⇔ matching edges)
- Until there is no augmenting paths

- At most n iterations
- Every iteration takes O(mn) time
- Total time: O(mn^2)

Fast Implementation

- Using union-find structure to maintain blossoms.
- Shrinking a blossom: find the least common ancestor of two EVEN vertices.



Micali and Vazirani algorithm

- The maximum cardinality matching in general graphs can be also found in $O(mn^{1/2})$ time.
  - By S. Micali and V.V.Vazirani (1980)
  - Similar idea as Hopcroft-Karp's algorithm for bipartite graphs
  - But much more complicated

Best algorithms for maximum matching

	Bipartite Graphs	**General Graphs**
Cardinality Matching	$O(mn^{1/2})$ [Hopcroft & Karp 1973]	$O(mn^{1/2})$ [Micali & Vazirani 1980]
Weighted Matching	$O(mn^{1/2}\log(nN))$ [Gabow & Tarjan 1988]	$O(m\sqrt{n}\log(nN)\sqrt{\alpha(m,n)\log n})$ [Gabow & Tarjan 1988]

$\Uparrow$

Very Complicated

Best algorithms for maximum matching

	Bipartite Graphs	General Graphs
Cardinality Matching	$O(mn^{1/2})$ [Hopcroft & Karp 1973]	$O(mn^{1/2})$ [Micali & Vazirani 1980]
Weighted Matching	$O(mn^{1/2}\log(nN))$ [Gabow & Tarjan 1988]	$O(m\sqrt{n}\log(nN)\sqrt{\alpha(m,n)\log n})$ [Gabow & Tarjan 1988]
	$O(Nn^{\omega})$ [Sankowski 2006]	
	$O(mn^{1/2}\log N)$ [Duan & Su 2012]	

Next Class

- Weighted matching for bipartite and general graphs