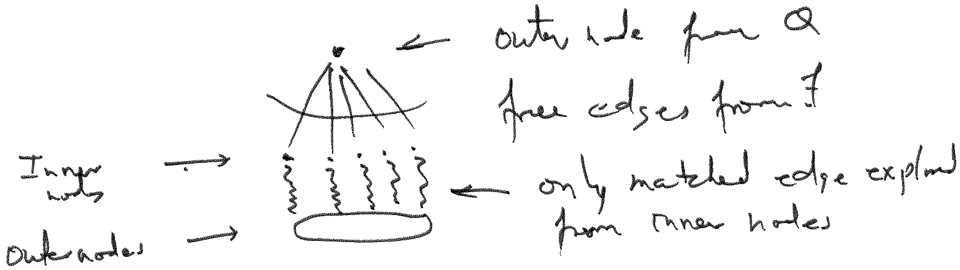
Summary of matching algorithm for bipartite graphs

1. Nodes are classified into $\underbrace{\text{Outer / Inner}}_{\text{Bipartition}}$ nodes

2. start searching for augmenting paths from <u>all</u> free outer nodes.

3. Built alternating forest from these nodes.
   (Queue contains only outer nodes)
   - outer nodes that are free
   - outer nodes that are added to Queue by their mate.



4. An augmenting path is found if there is a free inner node that is added to the alternating forest.

5. When algorithm terminates: alternating forest = ~~Hungarian~~ F
   Hungarian Forest
   - there are no free inner nodes in a Hungarian Forest.

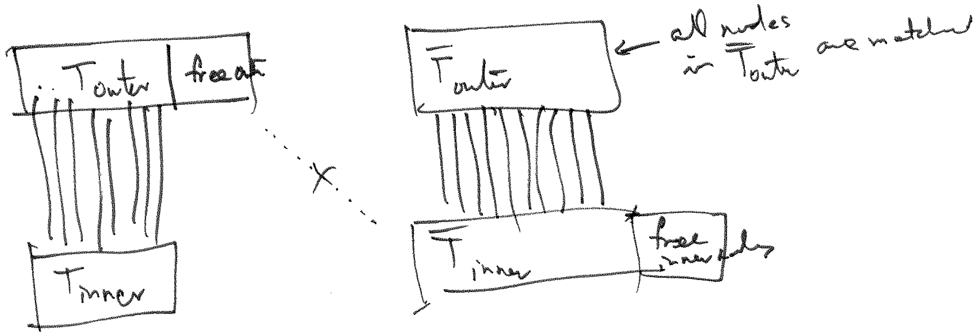
<u>Correctness of algorithm:</u>

$$\text{Outer} \;—\; T_{outer} \cup \overline{T}_{outer}$$

T_{outer}: outer nodes in H. ~~Tree~~ Forest

$\overline{T}_{outer}$: other outer nodes.

$$\text{Inner} \;—\; T_{inner} \cup \overline{T}_{inner}.$$



- No edges of G between T_{outer} and $\bar{T}_{inner}$.
  (Otherwise, if $(a, b) \in E$ with $a \in T_{outer}$, $b \in \bar{T}_{inner}$, then when a was processed in Q, b would have been added to alternating forest with $b.parent = a$.
  $\Rightarrow b \in T_{inner}$).

- $T_{inner} \cup \bar{T}_{outer}$ is a vertex cover of G.
  (all edges of G have at least one end in $T_{inner} \cup \bar{T}_{outer}$)

- Size of matching $|M| = |T_{inner}| + |\bar{T}_{outer}|$.

$\Rightarrow$ M is a max matching
(because no matching can be bigger than a Vertex Cover.
  - ~~no two edges of M share a common node~~.

edges of M: $(\underline{u}, v)$ $u \in VC$

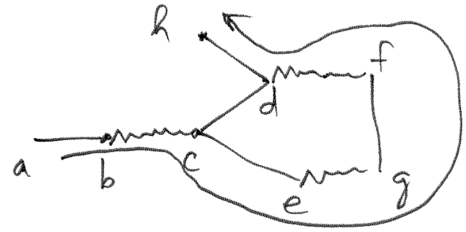
$(a, \underline{b})$ $b \in VC$

(p, q) $p, q \in VC$.

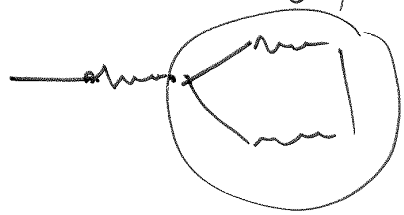
Since no two edges of M share a common node:

$$|M| \leq |VC|.$$

What about general graphs?

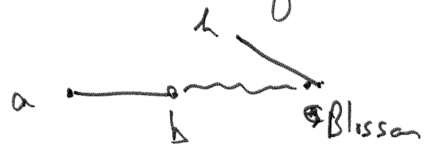


If we try bipartite matching algorithm on this example,
exploring from a: no augmenting path is found.

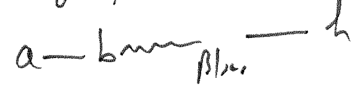


← Odd length cycle called Blossom is encountered.

shrink blossom into a single node:



Augmenting path:



When matching is augmented:

Summary of Matching algorithm in general graphs

1. start with a maximal matching (use greedy algorithm).
2. Build an alternating forest, starting from all free nodes.
   Nodes are classified as outer / inner — Q contains only outer nodes.
3. When u is removed from Q:
   search all free edges from u.
   $(u,v) \in E$: $(u,v) \notin M$,

   Case 1: v is free — v is classified as free inner node.
   augmenting path is found.

   Case 2: v is matched, but unseen. — v is inner
   — v.mate is outer
   ↳ gets added to Q

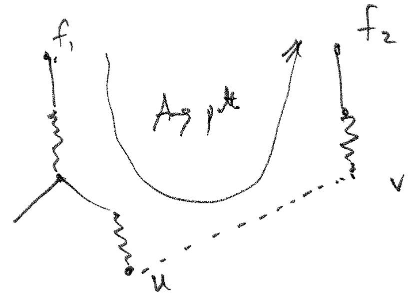
   Tree grows by 2 steps.

   v.parent ← u   v.type ← inner
   x ← v.mate   x.type ← outer   x.parent ← v
   v.seen ← true   x.seen ← true   Q.add(x)

Case 3: v is an inner node, seen before — continue

Case 4: v is an outer node, in a different tree than u.

Ex:



- Augmenting path is found:

$\underbrace{\text{root of } u\text{'s tree} \cdots\cdots}_{\text{in } u\text{'s tree}}\ u \cdots\cdot v\ \underbrace{\cdots\cdots \text{root of } v\text{'s tree}}_{v\text{'s tree}}$

Case 5: v is an outer node, in same tree as u. — Blossom.

Find lca (least common ancestor) of u and v.

Odd length cycle from lca ... u ... v ... lca

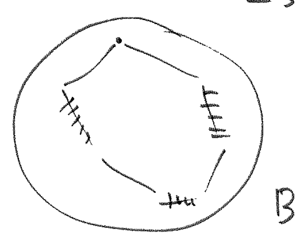
shrink cycle into a single node and continue.

Expanding a blossom — 2k+1 cycle   Edge from lca to its parent — stem of blossom.

Case 1: B is not matched

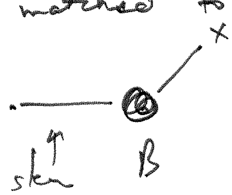
- choose matching of size k within B.

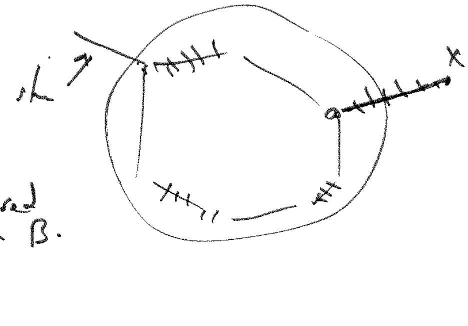
Add this to matching of other nodes.



Case 2: B is matched to its stem. Add k more edges within B. (same matching that was used when B was shrinked)

Case 3: B is matched to some other node.





Add k more edges within B based on where x connects within B.