Combinatorial Mathematics

Mong-Jen Kao (高孟駿)

Monday 18:30 – 20:20

Outline

- The Maximum Matching Problem
  - A Generic Algorithm and the Berge’s Theorem
  - The Augmenting Path Problem in Bipartite Graphs
    - A simple DFS-like recursive algorithm
- Concluding Notes
  - The best algorithms for Maximum Matching

The Maximum Matching Problem

To compute a maximum-size matching for the input graph.

The maximum matching problem

- Input :
  - A graph $G = (V, E)$.
- Output :
  - A matching $M \subseteq E$ that has the maximum size among all possible matchings.

Maximal matching v.s. Maximum matching

- A matching M is called maximal,
  if there exists no other matching M' that contains M.

A maximal matching

- A matching M is called maximum,
  if its size is at least the size of all other matchings.

A maximum matching

- Note that,
  a maximal matching is not necessarily a maximum matching.

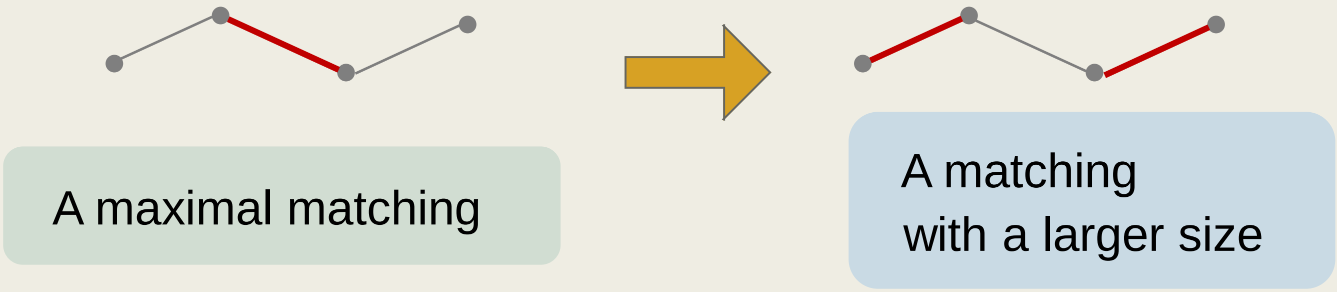
Local maximum vs global maximum

How Can We Enlarge the Size of a Matching?

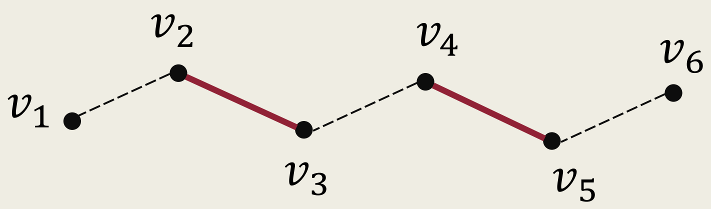
- To enlarge the size of a matching,
  we can add edges to the current matching until it becomes maximal.

- However,
  a maximal matching is not necessarily a maximum matching.

- What can we do?



Alternating Path & Augmenting Path

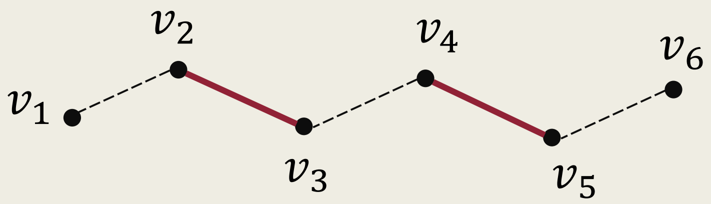


- Given a matching M,
  - an M-alternating path is a path
    that alternates between edges in M and edges not in M.
  - an $\boldsymbol{M}$**-augmenting path** is an M-alternating path
    that both starts and ends at unmatched vertices.

$v_1, v_2, v_3, v_4, v_5, v_6$ is
an M-augmenting paths.

v_1, v_2, v_3 and v_2, v_3, v_4, v_5 are
both M-alternating paths.

Observation

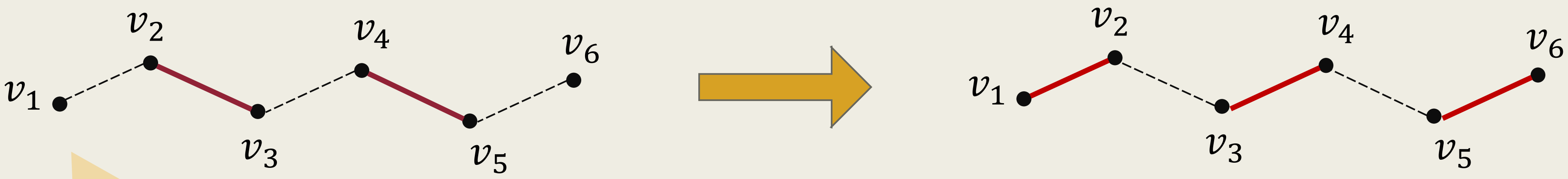


- We can see that,

  each M-augmenting path P is a way to enlarge the size of M by 1.

  - This is done by swapping the status of the edges on the path.
    - Matched edges $\Longrightarrow$ *unmatched*
    - Unmatched edges $\Longrightarrow$ *matched*

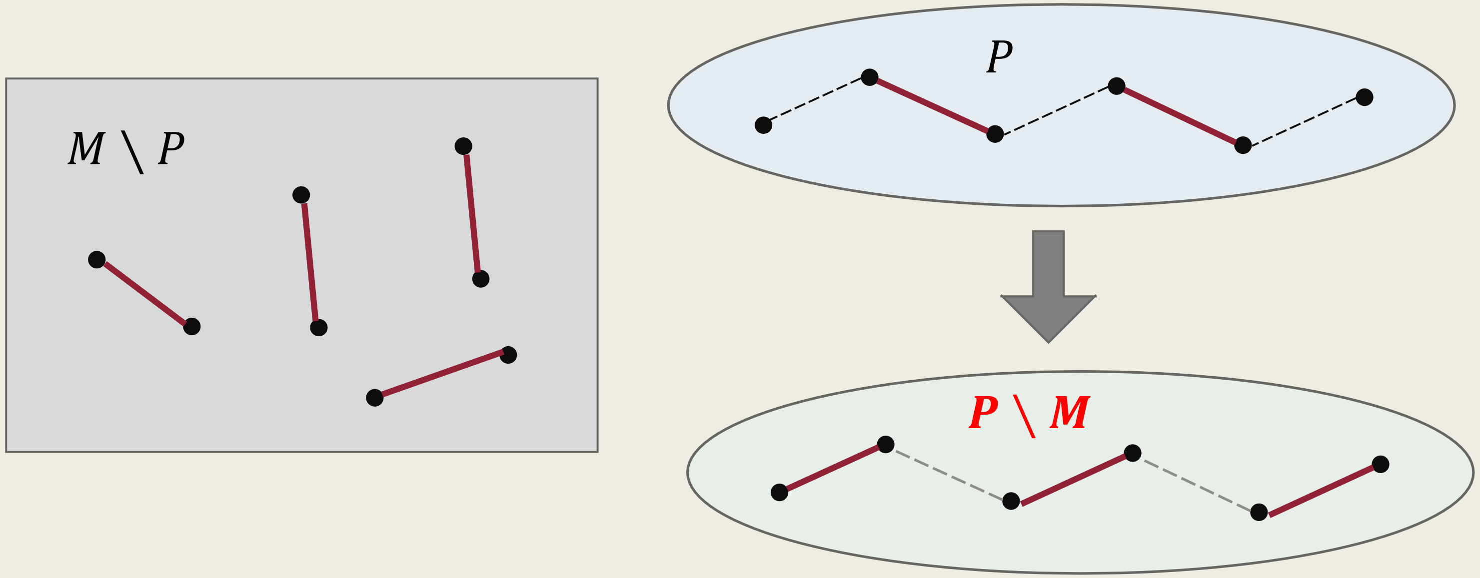
So, this is still a valid matching with size increased by 1.



v_1 and v_6 were unmatched.

All internal vertices are matched only by edges on the path.

Observation



- We can see that,

  each M-augmenting path P is a way to enlarge the size of M by 1.

- $M' \coloneqq (M \setminus P) \cup (P \setminus M)$ is a valid matching with $|M'| = |M| + 1$.

A simple greedy algorithm

- The observation suggests the following algorithm.
  - Let $G = (V, E)$ be the input graph.

1. $M \longleftarrow \emptyset$.
2. Repeat until there is no M-augmenting path in G.
   a. Find an M-augmenting path P.
   b. Set $M \longleftarrow (M \setminus P) \cup (P \setminus M)$.
3. Output M.

1. $M \longleftarrow \emptyset$.
2. Repeat until there is no M-augmenting path in G.
   a. Find an M-augmenting path P.
   b. Set $M \longleftarrow (M \setminus P) \cup (P \setminus M)$.
3. Output M.

The philosophy behind the algorithm is very simple :

"Make the current matching larger until no augmenting path exists."

- A very natural question is that,

  "Does it always output the maximum matching?"

Theorem 1. (Berge 1957).

A matching M in a graph G is a maximum matching if and only if G has no M-augmenting path.

- Theorem 1 assures the correctness of the previous algorithm.

"Yes, the algorithm always outputs a maximum matching for G."

- The next question is,

"is the algorithm efficient?"

That is, can we ***efficiently*** *determine the existence* of augmenting paths and *compute one* if it exists?

We will address this problem later.

Symmetric Difference

- Let $G = (V, E)$ be a graph, and $A, B \subseteq E$ be two edge sets.
  - The symmetric difference of A and B is defined as

$$A \triangle B \coloneqq (A \setminus B) \cup (B \setminus A).$$

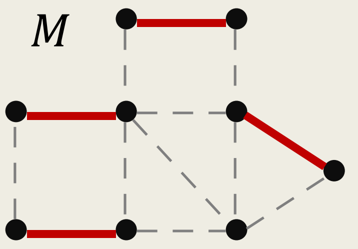
  - That is, the set of edges that appear exactly once in A and B.

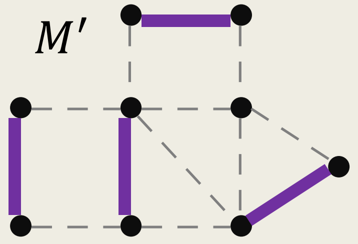
Lemma 2.

Let M, M' be two matchings for a graph G.

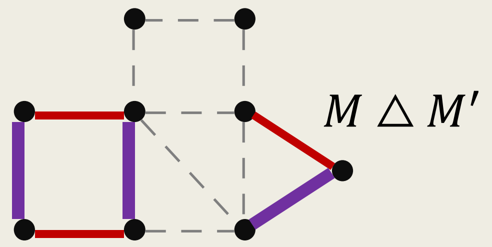
Every component of $M \triangle M'$ is a path or a cycle with an even length.

- Let $F \coloneqq M \triangle M'$.
  - Each vertex in G is incident to at most two edges in F.
  - Hence, each component in F is either a path or a cycle.





- Consider any cycle in F.
  - The cycle alternates between edges in M and M'.
  - It must have an even length.



Theorem 1. (Berge 1957).

A matching M in a graph G is a maximum matching if and only if G has no M-augmenting path.

- Let us prove Theorem 1.
  - The direction $\Longrightarrow$ is clear.
  - It suffices to prove that,
    if G has no M-augmenting path, then M is a maximum matching.
- We will prove the contrapositive of the above, i.e.,

  if M' is a matching with $|M'| > |M|$,
  then G has an M-augmenting path.

It suffices to prove that, if M' is a matching with $|M'| > |M|$, then G has an M-augmenting path.

- Let $F \coloneqq M \triangle M'$.
  - By Lemma 2, F is a union of paths and even cycles.
- Since $|M'| > |M|$,

  there must be a component in F that has more edges from M' than M.
  - The component must be a path.

    Furthermore, it must start and ends with edges in M'.
  - The path is then an M-augmenting path.

The Augmenting Path Problem

in Bipartite Graphs

The augmenting path problem in bipartite graphs can be solved by DFS in $O(n+m)$ time!

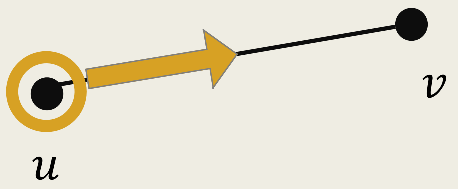
The Augmenting Path Problem in Bipartite Graphs

- Input :
  - A bipartite graph $G = (V, E)$ and a matching M for G.
- Goal :
  - An M-augmenting path for G, or asserts that there exists no such paths.
- We will present an $O(n + m)$ algorithm for this problem.

This leads to an $O(nm)$ algorithm for the maximum bipartite matching problem.

A Simple DFS-like Algorithm

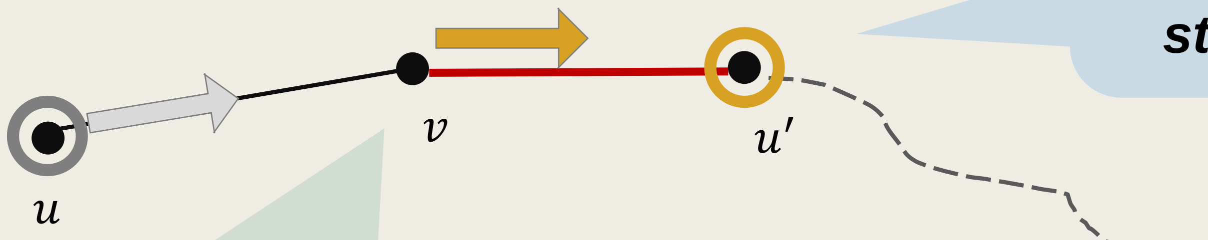
- Finding an M-augment path problem in a bipartite graph can be done by a simple & intuitive DFS-like algorithm.
  - We start with an unmatched vertex, say, u.
    - The goal is to find an M-augmenting path starting from u.
  - Consider each neighbor of u, say, v.



If v is *unmatched*,
then u, v is an M-augmenting path,
and we're done.

- We start with an unmatched vertex, say, u.
  - Our goal is to find an M-augmenting path starting from u.

- Consider each neighbor of u, say, v.



Then, the goal becomes finding an M-augmenting path ***starting from*** u'.

This is a recursive problem that starts at the vertex u'.

If v is ***matched***, then to form an M-augmenting path that passes v, we must follow the matched edge to some u'.

- We start with an unmatched vertex, say, u.
  - Our goal is to find an M-augmenting path starting from u.

- Consider each neighbor of u, say, v.

Then, the goal becomes finding an M-augmenting path ***starting from*** u'.

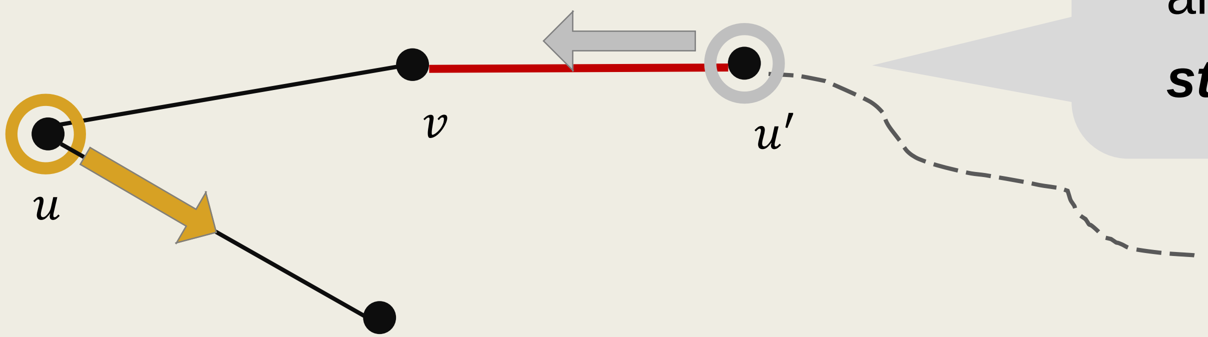
This is a recursive problem starting at the vertex u'.

If v is ***matched***, then
to form an M-augmenting path,
we must follow the matched edge to some u'.

If the recursion succeeds,
we have an augmenting path for u.

- We start with an unmatched vertex, say, u.
  - Our goal is to find an M-augmenting path starting from u.

- Consider each neighbor of u, say, v.



Then, the goal becomes finding an M-augmenting path ***starting from*** u'.

This is a recursive problem starting at the vertex u'.

If it fails, we go back to u,
and continue to examine the next neighbor
until all its neighbors have been examined.

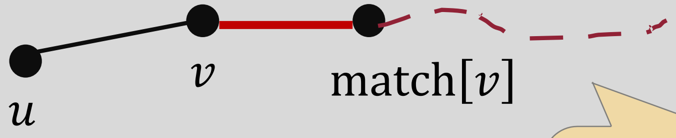
The DFS-like Recursive Algorithm

- To describe the algorithm, let’s assume the following.
- The graph is represented by adjacency lists.
- For each vertex v,
  let $\mathrm{match}[v]$ denote the vertex to which v is matched.
  - $\mathrm{match}[v] = -1$ if v is unmatched.

- The DFS-like recursive algorithm goes as follows.

<u>Procedure</u> Aug-Path(u)

1. Mark u as *visited*.
2. For each neighbor v of u, do.
   - If v is *unmatched*, or,
     if match$[v]$ is *unvisited* and Aug-Path(match$[v]$) is true, then
     a. Set match$[u] = v$ and match$[v] = u$. // match u with v
     b. Return *true.*
3. Return *false*.



Augmenting path from $\mathrm{match}[v]$ is found.

The Augmenting Path Algorithm

for Bipartite Graphs

The Augmenting Path Algorithm for Bipartite Graphs

- Let $G = (V, E)$ be the input bipartite graph and M a matching for G.
- The algorithm goes as follows.

<u>The Augmenting Path Algorithm</u> (for Bipartite Graphs).

1. Mark all the vertices as *unvisited*.
2. For each *unmatched* vertex, say, u, do
   - If Aug-Path(u) returns true, then report "Yes."
3. Report "No."

The Augmenting Path Algorithm for Bipartite Graphs

- Since each vertex is visited at most once and each edge is examined at most twice by the procedure Aug-Path(),
  - The algorithm runs in $O(n+m)$ time.
- It is clear that, if Aug-Path(u) returns true, then an M-augmenting path starting at u is found.
- To prove the correctness of the algorithm, it remains to prove that,
  - There exists no M-augmenting path in the graph when the algorithm reports "No."

The Augmenting Path Algorithm for Bipartite Graphs

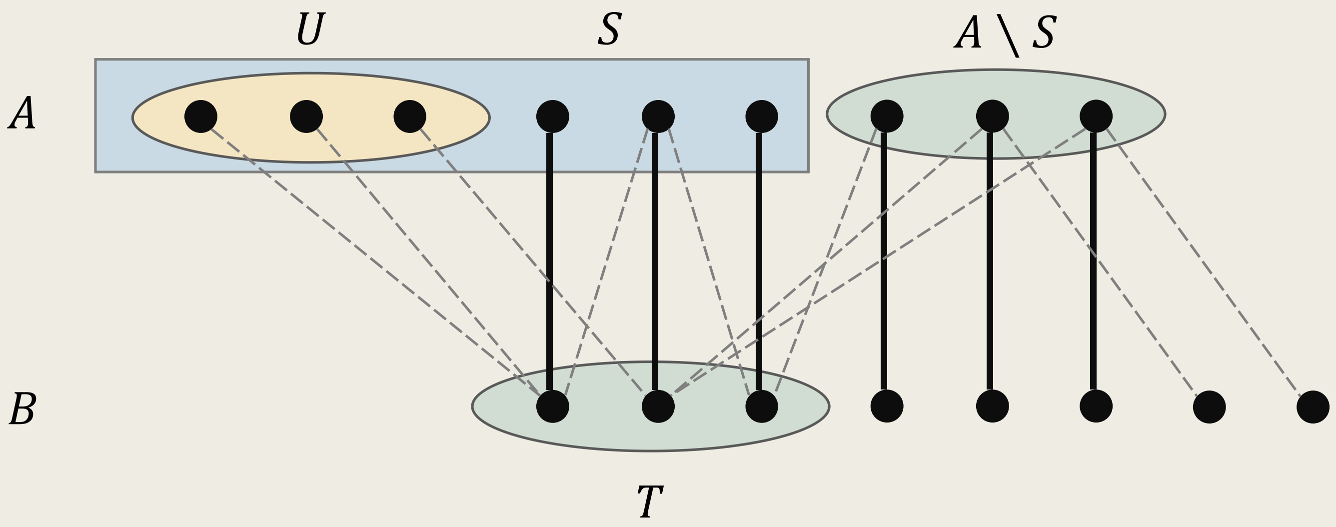
- To prove the correctness of the algorithm, it remains to prove that,
  - There exists no M-augmenting path in the graph when the algorithm reports "No."

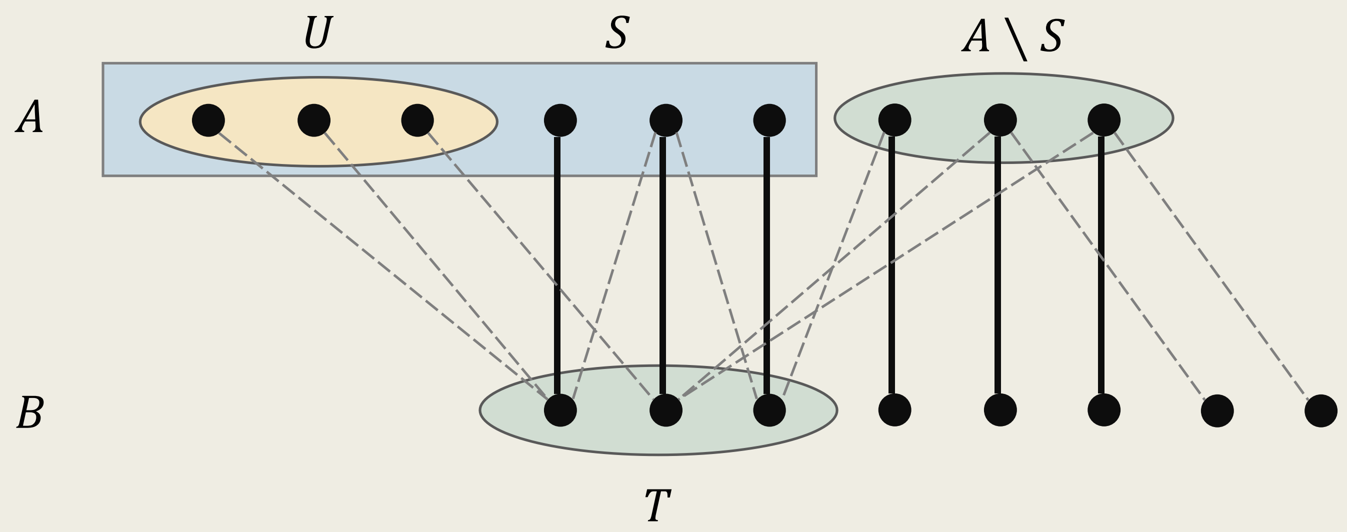
- We will prove that, if the algorithm reports "No," then G has a vertex cover C of size $|M|$.
  - Since $|C| \geq |M'|$ holds for all matching M' for G, this will imply that M is a maximum matching for G.

It takes at least one vertex to cover each edge in M.

Some Notations

- Let A and B be the two partite sets of G.
  - Let U be the set of unmatched vertices in A.
  - Let S be the vertices in A that are marked as *visited*.
  - Let T be the set of vertices in B that are matched to $S \setminus U$ by M.





Theorem 3.

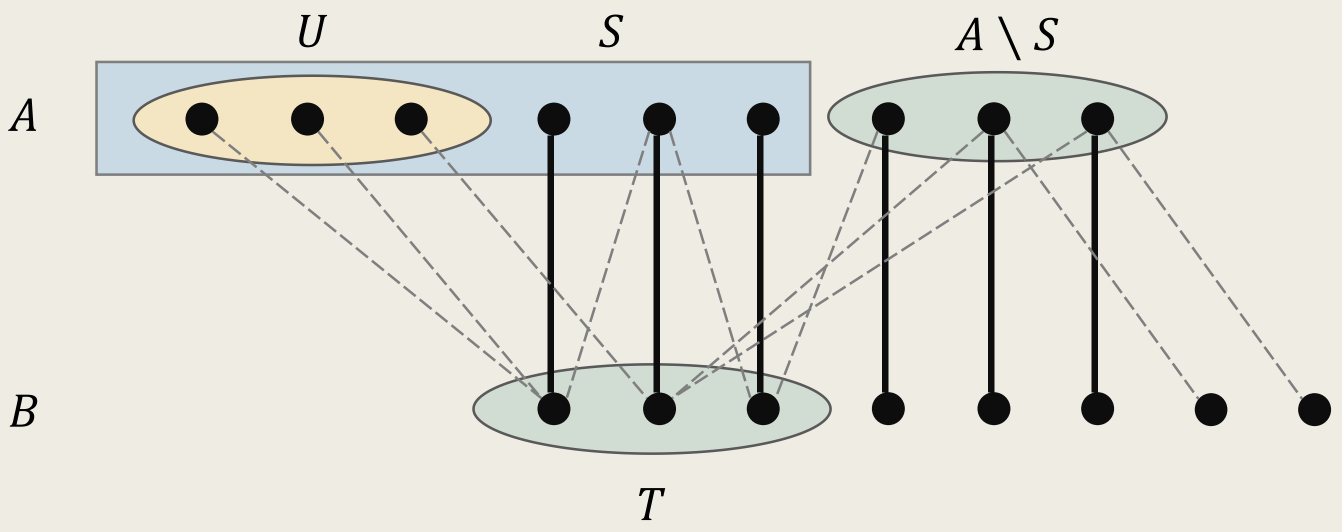
If the Augmenting Path Algorithm reports "No," then the set $C \coloneqq (A \setminus S) \cup T$ is a vertex cover for G with size M.

Note that, this is also a *constructive proof* for the König-Egeváry theorem.

Observation 1.

Since v is marked visited,
it is visited by a recursion call that
originates from some $u \in U$.

- For each $v \in S$,
  - There is an M-alternating path that starts at some $u \in U$ and ends at v with a matched edge in M.

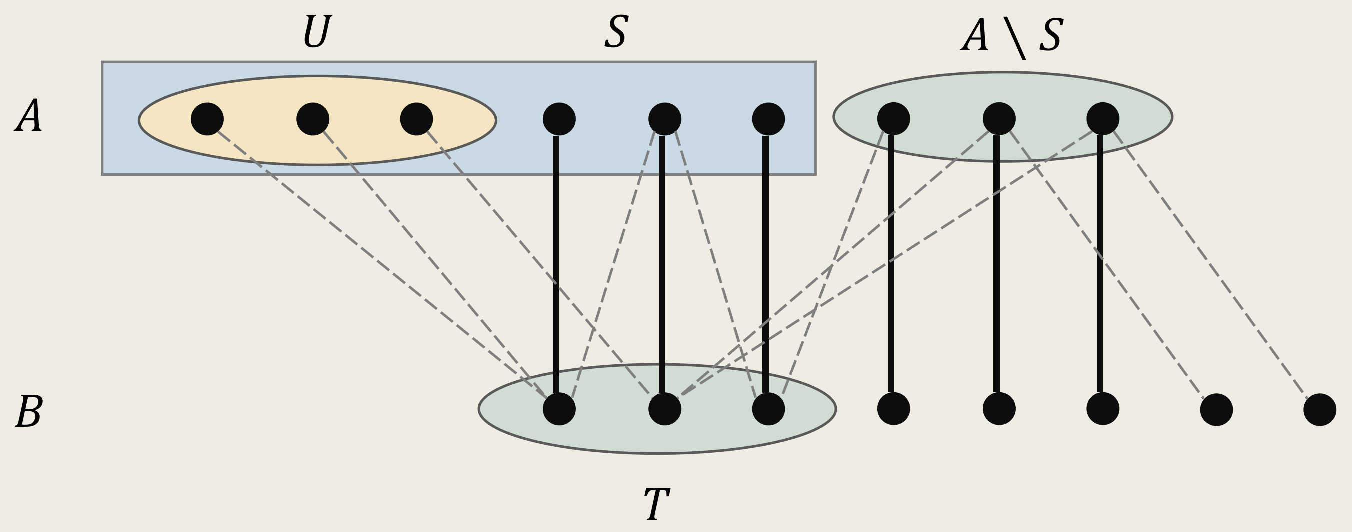


Observation 2.

If so, that matched vertex should be classified in T.

- There exists no edge between S and $B \setminus T$.
  - By the way S is defined, there exists no edge between S and the matched vertices in B.
  - If there exists an edge between S and some unmatched vertex in B, it will be an augmenting path.

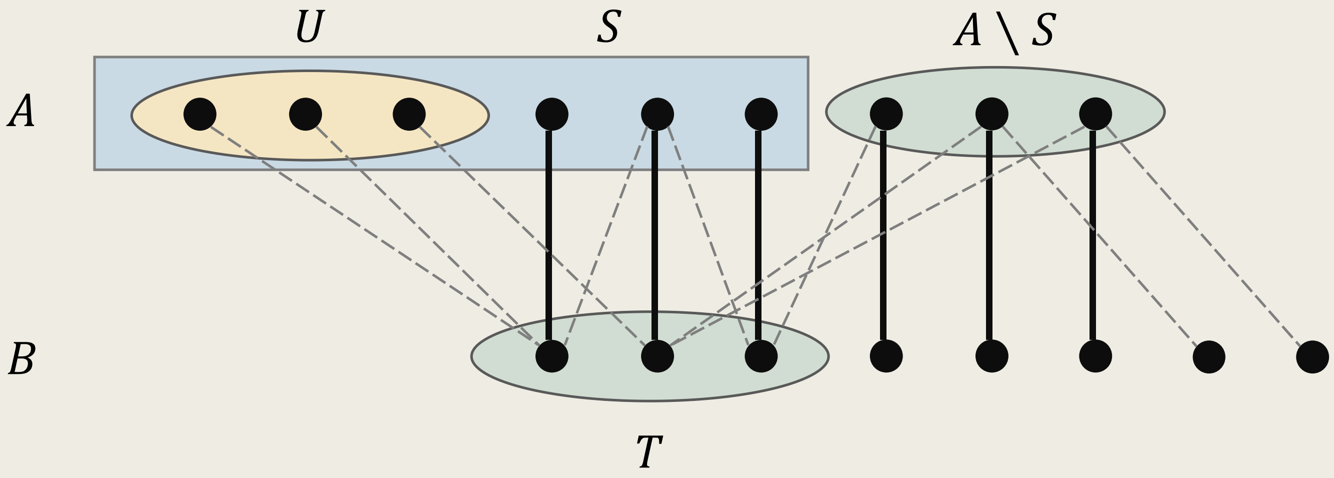
A contradiction since the algorithm reports "No."



Theorem 3.

If the Augmenting Path Algorithm reports "No," then the set $C := (A \setminus S) \cup T$ is a vertex cover for G with size M.

- The edges between S and T can be covered by T.
- By Observation 2, the remaining edges can be covered by $A \setminus S$.
- Hence, C is a vertex cover for G.



Concluding Notes

Best Algorithm for the Maximum Bipartite Matching

- In this lecture,
  we have seen an $O(nm) = O(n^3)$ algorithm for this problem.

- The best algorithm for this problem is the Hopcroft-Karp algorithm,
  which runs in $O(\sqrt{n}m) = O(n^{2.5})$.

The Hopcroft-Karp Algorithm

- The best algorithm for this problem is the Hopcroft-Karp algorithm, which runs in $O(\sqrt{n}m) = O(n^{2.5})$.
  - The idea is to perform a ***BFS*** *simultaneously* from all unmatched vertices in one partite set ***to form alternating layers*** until some unmatched vertices in the other partite set is met.
  - Then a ***layer-guided DFS*** is used to construct a maximal set of *vertex-disjoint shortest augmenting paths*.
  - It is guaranteed that, only $O(\sqrt{n})$ rounds are needed before the maximum matching is computed.

Maximum Matching in General Graphs

- For general graphs, a maximum matching can be computed by Edmonds Blossom algorithm in $O(n^2m) = O(n^4)$ time.
  - It is a beautiful algorithm.

- The best (and more complicated) algorithm, due to Micali and Vazirani, solves this problem in $O(\sqrt{n}m) = O(n^{2.5})$ time.