# Class 31

MAXIMUM BIPARTITE MATCHING

## Exam 3

Time: Monday, February 6, 7-8:30pm

Location:
- Crapo G219, section 1
- Crapo G220, section 2
- Crapo G221, section 3
- Crapo G222, section 4

Covers material from January 13 - 27: Chapters 8 and 9

You may use a 1-page cheat sheet during the exam

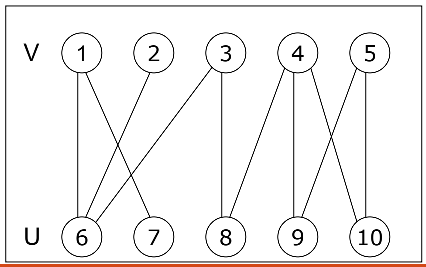
No class Tuesday, February 7, to compensate for evening exam

# Pairing Elements

Sometimes, it is important to match up elements of two sets.

We will use a graph to represent the elements of the two sets,

Edges are used to indicate which elements among the two sets can be matched.

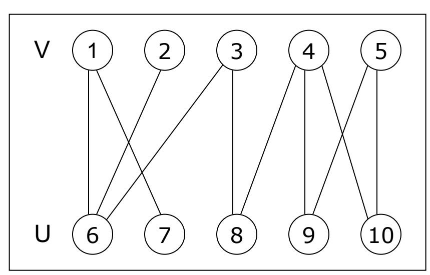



# Bipartite Graphs

In a *bipartite graph*, vertices can be partitioned into two disjoint sets U and V, not necessarily of the same size, so that every edge has one end in U and the other end in V.

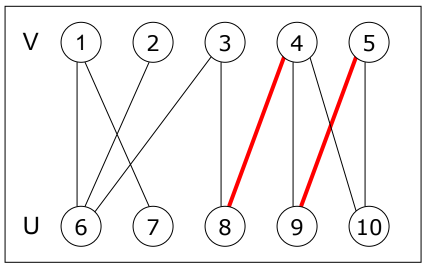
For a graph G, the following are equivalent:

- G is bipartite
- G has no odd-length cycle
- The vertices of G are *2-colorable*



# Matching in a Graph

- A *matching* in a graph is a subset of its edges with the property that no two edges share a vertex.
- In a matching M, a vertex is either *matched* or *free*



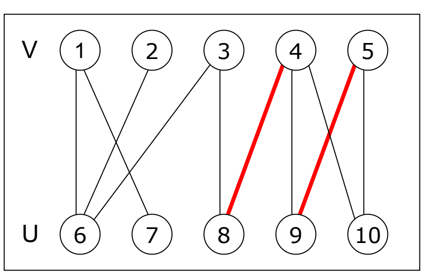

example matching:
M = {(4,8), (5,9)}
Vertices 4, 5, 8 and 9
Are matched.
Vertices 1, 2, 3, 6, 7
and 10 are free.

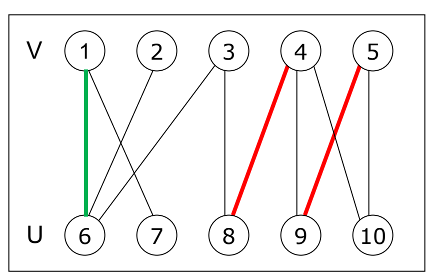
# Improving a Matching

A *maximum* (or *maximum cardinality*) matching is a matching with the largest number of edges.

Always exists, not necessarily unique.

One option: add an edge between a free vertex in U to a free vertex in V

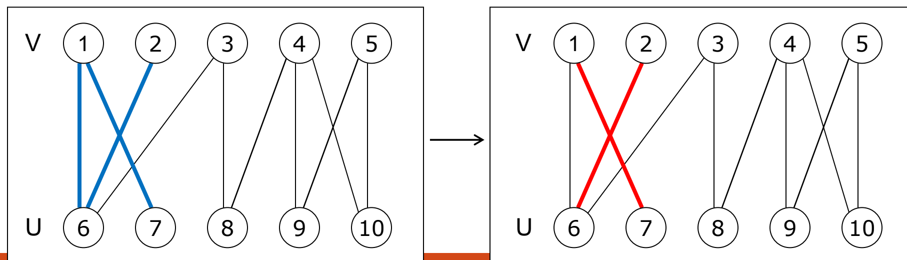


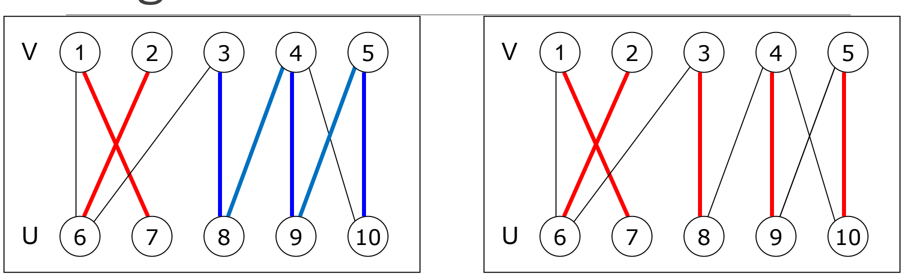
# Augmenting Paths

*Augmenting path* for matching M: path from free vertex to free vertex whose edges alternate: not in M, in M, etc.

- The length of an augmenting path is always odd.
- Remove every edge appearing in an even numbered position.




# Augmentation Continued




Now a *perfect matching* (all vertices matched), a maximum.

- Note: a perfect matching requires |V| = |U| and doesn't necessarily exist

# Method for Constructing Augmenting Path

Start with some initial matching

- e.g., the empty set

Find an augmenting path and augment the current matching along that path

- e.g., using breadth-first search like method

When no augmenting path can be found, terminate and return the last matching, which is maximum

- Follows from: **Berge's Lemma**. A matching M is maximum if and only if there exists no augmenting path with respect to M.
- We'll prove this.

# BFS-based Augmenting Path Algorithm

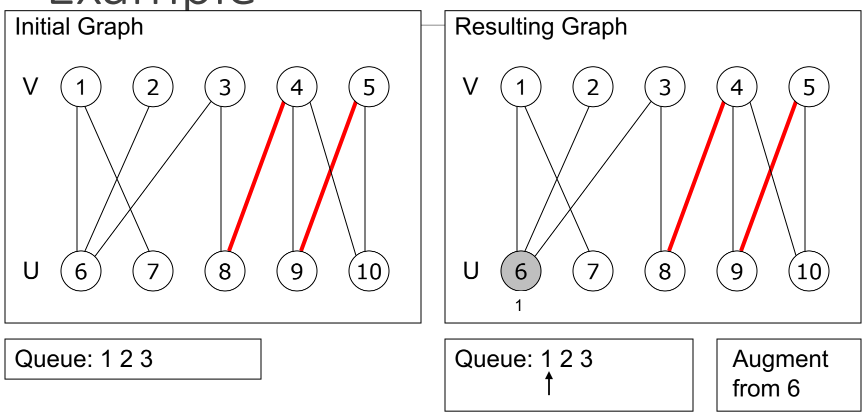
Initialize queue Q with all free vertices in one of the sets (say V)

While Q is not empty: dequeue *w* from Q.

- Case 1: *w* is in V. For each unlabeled neighbor *u*:
  - If *u* is free, augment the matching along the path ending at *u* by following labels backward until a free vertex in V is reached. Erase all labels and reinitialize Q with all the vertices in V that are still free
  - If *u* is matched (not with *w*), label *u* with *w* and enqueue *u*
- Case 2 (*w* is in U) Label its matching mate *v* with *w* and enqueue *v*

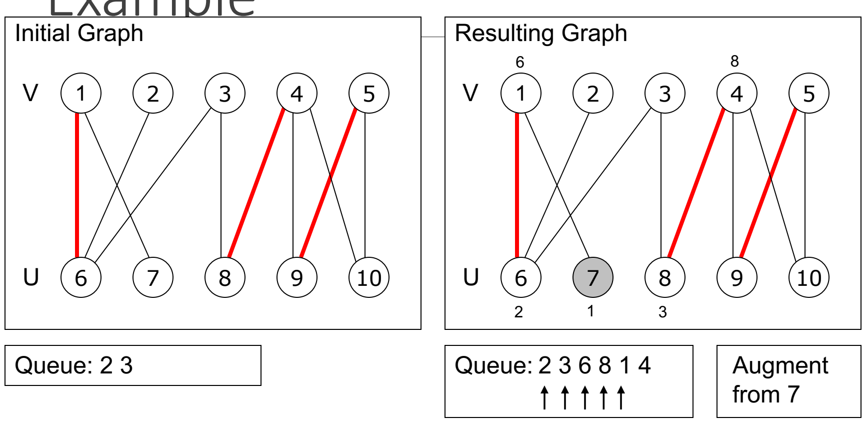
When Q becomes empty, return the last matching, which is maximum

# Example

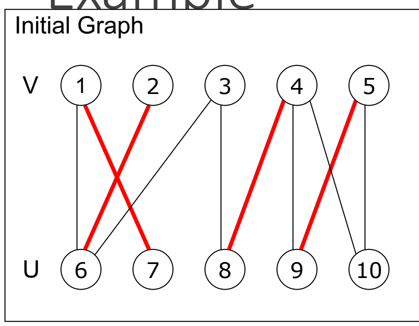


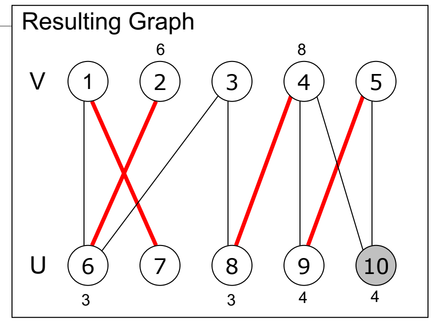

Each vertex is labeled with the vertex it was reached from. Queue deletions are indicated by arrows. The free vertex found in U is shaded and labeled for clarity; the new matching obtained by the augmentation is shown on the next slide.

# Example



## Example



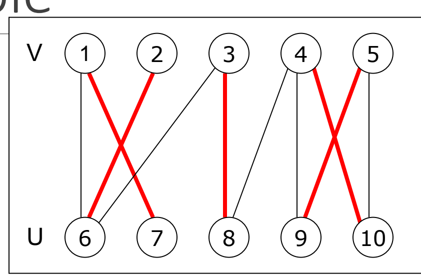




Queue: 3

Queue: 3 6 8 2 4 9

Augment from 10

## Example




maximum matching

- This matching is maximum since there are no remaining free vertices in V (the queue is empty)
- Note that this matching differs from the maximum matching found earlier

# Notes on Maximum Matching Algorithm

The number of iterations cannot exceed $\lfloor n/2 \rfloor + 1$, where *n* = #vertices. (Why?)

Time spent on each iteration is in $O(n + m)$ where *m* = #edges. Hence, the time efficiency is in $O(n(n + m))$

This can be improved to $O(\sqrt{n}(n + m))$ by combining multiple iterations to maximize the number of edges added to matching M in each search

Finding a maximum matching in an arbitrary graph is more difficult…