lecture 9

- bipartite graphs
- matching
- stable marriage problem

Resources for today

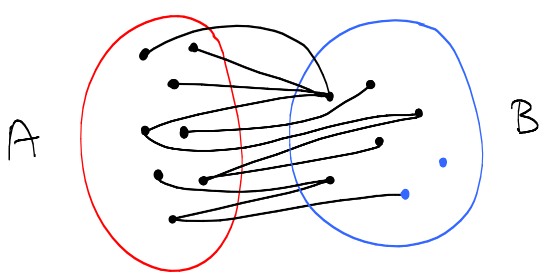
Kleinberg and Tardos textbook

(that was my main resource)

Bipartite Graphs

(directed or <u>undirected</u>)

TODAY



Vertices are partitioned into two sets.
All edges are crossing edges.

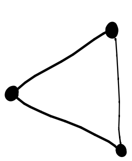
Many graphs are <u>designed</u> to be bipartite.

A		B
men	traditional marriage	women
people	employment	companies
students	registered	courses
profs	teach	courses
people	have seen/read	films/books
words	contains	documents
⋮		⋮

(Later today)

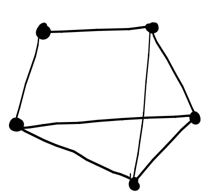
Examples of graphs that are not bipartite.

1



easy to see if is not bipartite

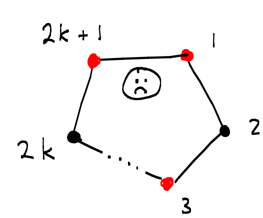
2



not so easy to see if it is bipartite

<u>Claim:</u> if a graph is bipartite then it does <u>not</u> contain an odd cycle.
[Exercise: the converse also holds. Prove it.]

<u>Proof:</u> (sketch) If we have an odd cycle, then we cannot color alternating vertices of the cycle red-black.

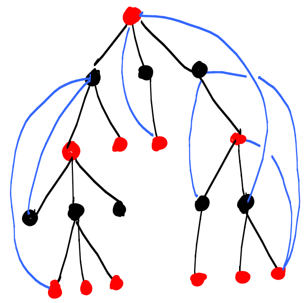


How to test if an undirected connected graph is bipartite?

- run DFS and build a DFS tree
- color vertices by layer

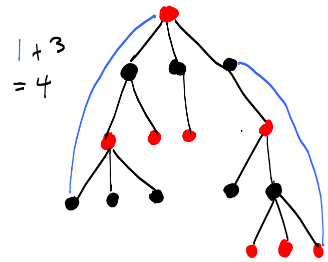
  e.g. even layers red, odd layers black
- non-tree edges in the graph are between an ancestor and descendent and span two or more levels.

  Why?

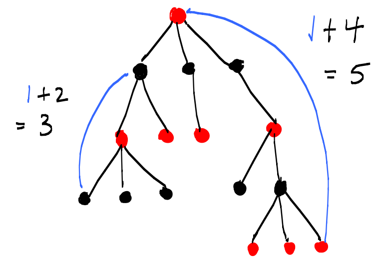


When DFS discovers a non-tree edge, check if its two vertices have the same color (red or black). If all **non-tree edges** join vertices of different color then the graph is bipartite. (Note that all tree edges, by the definition of the coloring, join vertices of the different color).

Note that this test is basically the same as the test in the claims above: two vertices in each non-tree edge also define a path in the tree. That path needs to be of odd length for the graph to be bipartite, since otherwise (the path is of even length), there would be a cycle of odd length.



bipartite



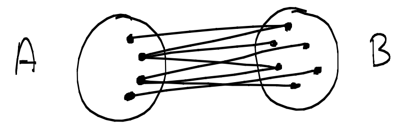
not bipartite

lecture 9

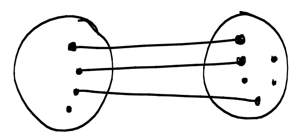
- bipartite graphs
- matching
- stable marriage problem

Bipartite Matching

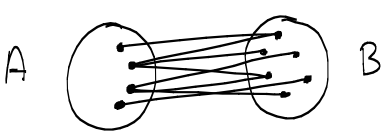
Consider an undirected bipartite graph.



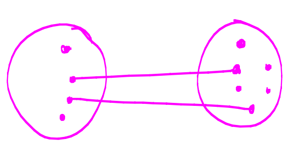
A matching is a subset of the edges $\{(\alpha, \beta)\}$ such that no two edges share a vertex.

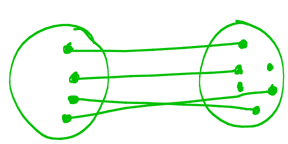


bipartite graph



Examples of matchings

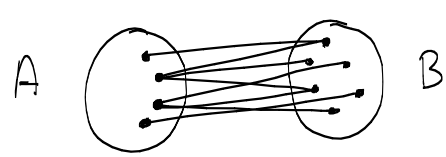




Yes, this too is a matching, though not such an interesting one.

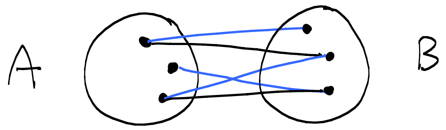
Maximal Matching Problem

(coming up in lecture 11)



Find the matching with the most edges.

Perfect Matching



Suppose we have a bipartite graph with n vertices each in A and B ($2n$ total)

A perfect matching is a matching that has n edges.

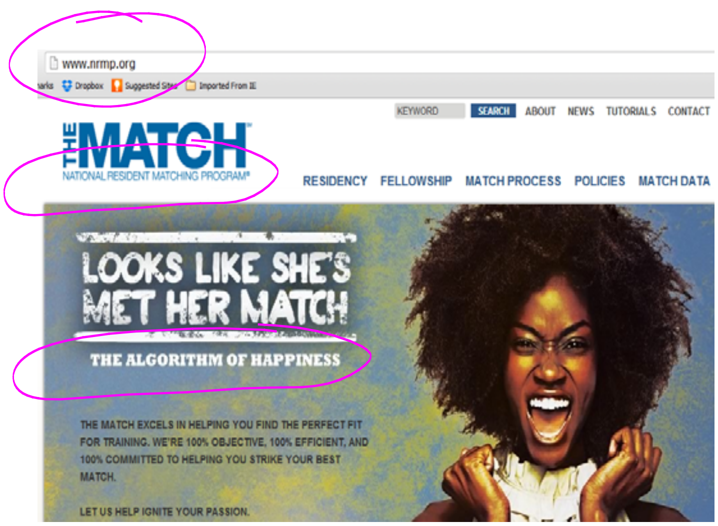
(It is not always possible to find a perfect matching)

Example

National Resident Matching Program:

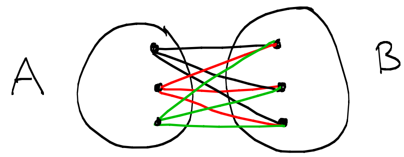
~30,000 medical school graduates per year in the U.S.

Where do they go for further training? ("residency")



Complete Bipartite Graph (definition)

For every pair of vertices (α, β) with $\alpha \in A$ and $\beta \in B$, the graph has an edge (α, β).



We will next consider an interesting perfect matching problem for which $|A| = |B|$.

"Stable Marriage" Problem

Consider a complete bipartite graph with sets A and B, each with n vertices.

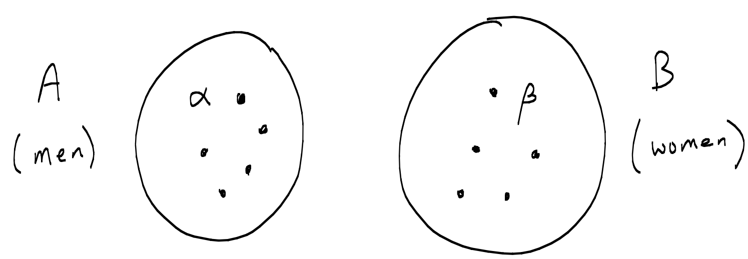
Each member of set A has a preference ordering of the members of B.
Each member of set B has a preference ordering of the members of A.

The problem is usually explained in terms of men (A) and women (B).
Think of these as preferences for marriage (matching).

<u>Algorithm for finding a matching (see next slide):</u>
Each A member proposes to a B, in order from most to least preferred. Each B member accepts the first proposal from an A, but then rejects that proposal if/when it receives a proposal from an A that it prefers more.

(Men propose to women. Woman accept the first offer made to them, but women will drop their partner when/if a prefered man proposes to them.)

Note the asymmetry between A and B.



alpha (Alan, "Al")

beta ("Betty")

Gale-Shapley algorithm (1962) for finding a perfect matching

For each α in A, let prefList(α) be the ordering of its B preferences.
For each β in B, let prefList(β) be the ordering of its A preferences.

Let **matching** be a set of crossing edges between A and B.

```
matching = empty set

while there is α in A that is not yet matched {   // use a list of unmatched A's.

    β = prefList( α ).removeFirst()
    if  β is not yet matched {              //  there is no (α', β) in matching
         matching.add( α, β )
    else                                              //  β already matched
         if  β prefers α over β's current match {          // unstable
              matching.remove(β's current match, β)
              matching.add(α, β )
         }
    }
return matching
```

Properties of the algorithm:

If β in B is not yet matched, then β has not been proposed to yet
(i.e. β must accepts the first offer that comes along).

β's match gets better (or stays the same) over time.
(After β in B is matched, β will only change its match if it is proposed to by an α that it prefers over its current match.)

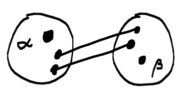
α's match worsens (or stays the same) over time.
(Once α in A is matched, it *can* become unmatched, namely if its current β match accepts the offer of a different α' in A. We'll see that α becomes matched again but, when it does, this match is less preferred.)

Claim: **the Gale-Shapley algorithm always finds a perfect matching.**

Proof:

We show that when the algorithm terminates, every α has a match.

If α in A is not matched during the algorithm, then:



- there must be some β in B that is not yet matched
  (because there are the same number of A's and B's.)

- α could not yet have proposed to this β
  (because an unmatched β will always accept a proposal offer.)

It cannot happen that the algorithm terminates in such a state, since α still must have the chance to propose to β.

Thus, every α will eventually be matched (and since there are the same number in A and B, it follows that every β will have a match too).

What is the O( ) running time of the algorithm?

| A | = | B | = n.

Each pass through the main while loop reduces the size of the (remaining) preference list of one of the α in A. Since there are n preferences lists for all the members of A, each list of length n, there are n^2 passes through the main loop.

Hence the running time is O(n^2).

When the algorithm terminates, what properties does the matching have? Is there a sense in which it is a good matching ?

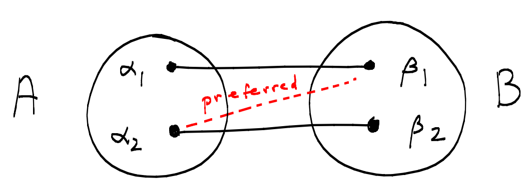
Our intuition is that the A's have the advantage here.
But in what technical sense ? How can we characterize this advantage ?

We will deal with these questions shortly, but first we need to introduce the concept of a **stable matching**.

What can happen in general with an arbitrary matching ?

Suppose there are two edges ($\alpha1$, $\beta1$) and ($\alpha2$, $\beta2$) in the matching and the preferences are such that $\alpha2$ would prefer to be matched with $\beta1$ and similarly $\beta1$ would prefer to be matched with $\alpha2$.

If the two above matches/edges are deleted so that ($\alpha2$, $\beta1$) can match up, then $\alpha1$ and $\beta2$ would become unmatched. $\alpha1$ would go down its preference list and seek another match which could lead to a chain a broken matches.



In this situation, we say that the matching is not stable (or unstable).

A **matching is stable** when there are no two elements, α and β, that prefer each other over their partners.

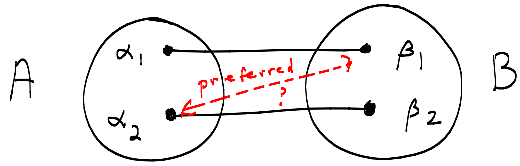
Claim: **the Gale-Shapley algorithm finds a stable matching.**

Proof: Suppose we run the algorithm and it gives us matches ($\alpha1$, $\beta1$) and ($\alpha2$, $\beta2$). Could it happen that **$\alpha2$ prefers $\beta1$** over $\beta2$ **and $\beta1$ prefers $\alpha2$** over $\alpha1$? No, that can't happen. Why not?

If **$\alpha2$ prefered $\beta1$** over $\beta2$, then $\alpha2$ would have proposed to $\beta1$ before it proposed to $\beta2$.

But since $\beta1$ is matched with $\alpha1$, we know that $\beta1$ must have rejected $\alpha2$'s proposal in favor of someone that it preferred more than $\alpha2$.

Since $\beta1$ can only improve its matches over time and it ended up matched with $\alpha1$, it follows that **$\beta1$ prefers $\alpha1$** over $\alpha2$ (not $\alpha2$ over $\alpha1$).



Example of two stable matchings

Is the match found by the Gale-Shapley algorithm the only stable matching? Not necessarily. Sometimes there is more than one.

For example, suppose that the first choices of each of the A's were different from each other. Then Gale-Shapley would give each of the A's their first choice.

Now further suppose the B's first choices (most preferred) were also different from each other, *and that the B's first choices don't correspond to the A's first choices*.

If we were to swap the roles of A and B in the algorithm and run this algorithm, then we would end up with a different matching than before, namely a matching in which each β in B would get its first choice instead of each α in A getting its first choice *(recall what we assumed above)*.

Both algorithms work with the same A and B preferences, and both give stable matchings. Thus, in this example at least, there exist two different stable matchings.

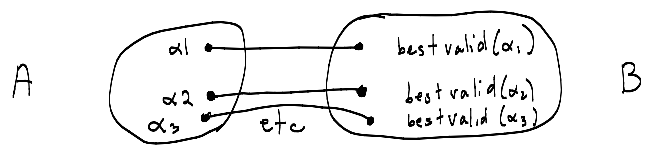
We will see next that the Gale-Shapley algorithm finds the best possible stable matching, where "best" is defined from the perspective of the A's.

Definition: Given A and B and their preferences, a particular edge (α, β) in the complete bipartite graph is **valid** if there exists a stable perfect matching between A and B that contains (α, β).
Note: the algorithm finds stable matchings, hence it finds only valid edges.

Definition: Given A and B and their preferences, for each α, define **bestvalid**(α) to be the β in B such that (α,β) is a valid edge and it is a better match from α's perspective than any other valid edge.

Claim: **The Gale-Shapley algorithm yields the matching:**

{ (α, bestvalid(α)) : for all α in A }.



Example

α_i

i	preference list for α_i					
1	3	2	4	1	5	6
2	3	4	1	2	6	5
3	5	4	2	3	1	6
4	6	1	2	5	4	3
5	1	2	5	3	4	6
6	2	1	4	5	3	6

k best valid

k valid

Note:

- the circled values are all different (see claim on previous page)
- the β preferences are not shown, but they also affect what is valid or not

How would you prove claim?

The algorithm finds a stable matching.

Thus, all matches found are valid.

We want to show that each match is the best valid match.

Suppose not!

Then, some $\alpha \in A$ must get rejected by bestvalid(α).

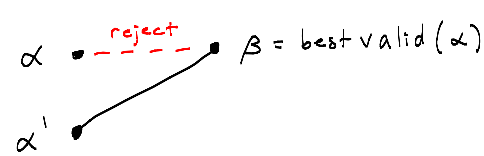
Proof:

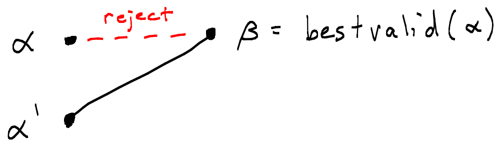
The proof is similar to the proof that the Gale-Shapley algorithm finds a stable matching.

Suppose the claim were not true. This means that, during the execution of the algorithm, some α in A would be rejected by bestvalid(α), and so α would end up matched to an element of B that it preferred less.

If such an event occurs, then there must be a ***first time*** *that it occurs.*
Assume without loss of generality that the above rejection of α by bestvalid(α) is the first time such a rejection occurs.

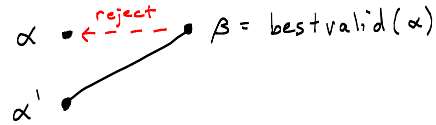
Let α' in A be the one that caused this rejection (by proposing to β) and so, at the time of rejection, we have:



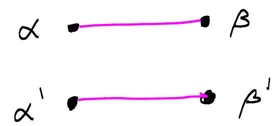


Notes

- this rejection can happen in two ways, but we don't care which:
  - (α, β) might be temporarily matched and then α' proposes to β
  - (α', β) are matched, and then α proposes to β.



Since (α, β) is a valid edge, **there must exist a stable matching that contains it**. For that stable matching, α' is matched with β', like so:



But since β prefers α' over α, and since the matching shown above is stable, it must be that α' prefers β' over β. Can this be the case?

No. If α' prefered β' over β, then α' would have proposed to β' before it proposed to β. (Recall that α' had not been rejected by any of its valid matches when it proposed to β.)

Definition: Given A and B and their preferences, for each β in B, define **worstvalid**(β) to be the valid α in A that β prefers least. (Here "valid" is relative to the particular β.)

Claim: Gale-Shapley finds the worst valid matching for each of the B's.

Proof: By contradiction. Suppose that, for some β, the algorithm finds a match (α , β) where α is not the worst possible match for β.

I didn't have time to do this proof in class.

So I have included it as an Exercise.

Since the Gale-Shapley algorithm finds the best valid matching for each of the A's and the worst valid matchings for each of the B's, it follows that Gale-Shapley finds the perfect matching:

$$\{ (\alpha,\ \text{bestvalid}(\alpha)) : \alpha \text{ in A} \} = \{ (\text{worstvalid}(\beta),\ \beta) : \beta \text{ in B}\}.$$

Related Problems

- "stable roommate problem": this is just the stable marriage problem where we allow straight and gay marriage (but no polygamy).

The problem is to put all people together into one set, and then each person now ranks all others. The problem again is to find a stable perfect matching (assuming there is an even number of people).
It can be shown that there does not necessarily exist a stable matching in this case. (Algorithms have been invented for deciding if a stable matching exists, and finding the stable matching if they do exist.)

- "hospital/residents problem" (or employer/employee problem, or college/admissions problem): An employer might have several positions open. Very similar to the stable marriage problem.
http://en.wikipedia.org/wiki/Hospital_resident#Matching_algorithm

who should get the advantage?

- the trainees? (students)
- the trainers? (programs)