19 Bipartite Matching

In this chapter we will look at the problem of computing maximum matchings which is an important application of max-flow algorithms. We say that a graph G is bipartite if it can be partitioned into two disjoint sets L and R such that every edge in the graph crosses the cut L, R (i.e. every edge has one endpoint in L and one endpoint in R). Bipartite graphs arise naturally in problems where the goal is to match two different kinds of things (e.g. students and courses). In this chapter we will explore algorithms for solving matching problems in bipartite graphs.

19.1 Matchings: Maximum and Perfect

For a (not necessarily bipartite) graph G, we say that a **matching** is a subset M of the edges of G such that no two edges in M share an endpoint. We say that a vertex v is **matched** by M if one of the edges in M has v as an endpoint. We say that M is a:

- **maximal matching** if there is no matching M' in G with $M \subsetneq M'$ (i.e. every edge not in M shares an endpoint with one of the edges in M)
- **maximum matching** if there is no matching M' in G with $|M'| > |M|$ (i.e. M is a matching with the maximum possible number of edges)
- **perfect matching** if every vertex in G is matched by M (i.e. if $|M| = n/2$)

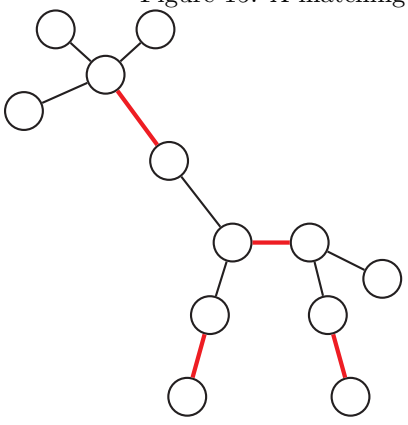
Note that every perfect matching is maximum, but not every maximum matching is perfect. Every graph has a maximum matching (even a graph with no edges has the empty matching $M = \emptyset$), but not every graph has a perfect matching.

At first the distinction between a maximal and maximum matching may not be clear. If we cannot add any more edges to M doesn't that mean that M uses the maximum possible number of edges? Not necessarily, as the following figure illustrates!

The red edges in Figure 15 form a matching of size 4 in the G-raph. This matching is maximal, because every edge not in the matching shares an endpoint with an edge from the matching (and therefore couldn't be added to the matching). This is not a maximum matching however, because the G-raph has a matching of size 5 (can you find it?).

The maximum matching in the G-raph is still not a perfect matching. We can see that this graph has no perfect matching from the fact that no matching could match both of the ossicone vertices (they both share a single neighbor, the head vertex, and only one of the two can be matched with that neighbor, meaning that any matching must leave at least one of the two unmatched).

Figure 15: A matching in the G-raph, which is maximal, but not maximum



The fact that we can have maximal matchings which are not maximum implies that the obvious greedy algorithm for finding a matching "while there exists an edge which shares no endpoints with

any edge in the matching, add such an edge to the matching", does not find a maximum matching in every graph (it can get stuck in a locally optimal matching like the one in Figure 15). Therefore if we want to compute a maximum matching we need to be a little bit more clever.

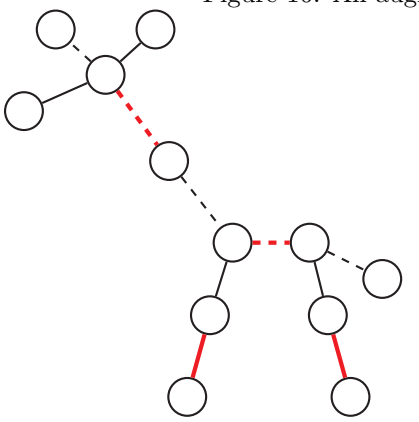
19.2 Characterizing Maximum Matchings

In this section we will prove a pair of theorems that will allow us to characterize when a matching is maximum and when a graph will have a large maximum matching.

Let $G = (V, E)$ be a graph and M be a matching in G. We say that a path P in G is **M-alternating**, if P alternates between using edges in M and edges in $E \setminus M$. We say that P is **M-augmenting** if it is an M-alternating path which begins and ends with vertices which are not matched by M.

Just as an augmenting path in the residual graph of a flow network implies that we can obtain a larger flow, an M-augmenting path implies that we can obtain a larger matching than M. In particular, we can increase the size of our matching by 1 by toggling which edges from the augmenting path P are part of the matching (swapping the edges from $E(P) \cap M$ with the ones from $E(P) \setminus M$). The dashed lines in the figure below form an augmenting path in the matching from Figure 15. Note that we could increase the size of our matching from 4 to 5 by swapping the two matching edges (the dashed red lines) on the augmenting path with the three non-matching edges on the path (the dashed black edges).

Figure 16: An augmenting path for the matching from Figure 15



In fact, any time that our matching is not maximum, we can increase its size by finding an augmenting path, as the following theorem states. The first proof of this theorem was published by Claude Berge in 1957, so it is known as Berge's theorem.

Theorem 89. *M is a maximum matching for a graph G if and only if G has no M-augmenting paths.*

Proof. To prove an "if and only if" statement we need to show two directions:

- First we show that if M is a maximum matching then M has no augmenting paths. We prove the contrapositive: If M has an augmenting path, then M is not a maximum matching.

  Let M be a matching which has an augmenting path, and let P be an M-augmenting path. Then, as noted above, we can swap all of the edges from M which are part of P for the edges of P which are not part of M to obtain a larger matching. More formally, let $M' := (M \setminus (E(P) \cap M)) \cup (E(P) \setminus M)$. We note that since P alternates between edges in $E \setminus M$ and edges in M, but begins and ends with unmatched vertices, P contains one more edge from $E \setminus M$ than from M, so $|M'| = |M| + 1$. Furthermore, M' is still a matching: It matches all of the vertices that M matched, as well as two additional vertices that were unmatched by M

(the start and end of P), so each edge of M' is matching two distinct vertices. Thus, since there is a matching larger than M, M is not a maximum matching.

- Now we show that if G has no M-augmenting paths, then M is a maximum matching for G. Again we prove the contrapositive: If M is not a maximum matching then G has an M-augmenting path.

  Suppose that M is a matching for G which is not a maximum matching. Then there exists some matching M' with $|M'| > |M|$. Consider the subgraph $G' := (V(G), M \cup M')$. Since each vertex in G has at most one edge in M and at most one edge in M' (since they are matchings), G' consists of disjoint paths and cycles. Furthermore, since edges in the same matching do not share an endpoint, each of these paths and cycles must alternate between edges of M and edges of M'. In particular, this means that all of the cycles must have an equal number of edges from M and M'. Thus, since $|M'| > |M|$, there must be some path P in G' which has more edges from M' than from M. This P must begin and end with vertices unmatched by M, so it is an M-augmenting path in G, and thus G has an M-augmenting path.

□

Berge's theorem gives us a characterization of maximum matchings in arbitrary graphs. It also suggests an algorithm for finding a maximum matching "start with an empty matching, then repeatedly find an augmenting path and toggle its edges until there are no augmenting paths." One detail of this algorithm is rather underspecified, however. How do we find an augmenting path? This turns out to be a fairly non-trivial problem. While it is possible to design an algorithm for computing a maximum matching in an arbitrary graph by repeatedly finding augmenting paths (look up Edmonds' Blossom Algorithm if you're curious), we will not discuss algorithms for maximum matching in arbitrary graphs in this course. Instead we will restrict our attention to a special class of graphs where finding a maximum matching is significantly easier.

Recall that a graph G is bipartite if we can divide the vertex set into two (not-necessarily equally sized) parts L and R such that all of the edges have one endpoint in L and the other in R (we often draw a bipartite graph with all the vertices of L on the left and all the vertices of R on the right). Note that for any matching M in a bipartite graph we must have $|M| \leq |L|$ and $|M| \leq |R|$, so G cannot have a perfect matching if $|L| \neq |R|$.

The following theorem, which was proven in 1935 by Philip Hall and is known as Hall's theorem, gives us a useful characterization of when a bipartite graph has a matching which matches all of the vertices on one side of the bipartition.

Theorem 90. *Let $G = (L \cup R, E)$ be a bipartite graph. Then G has a matching of size $|L|$ if and only if $\forall\ S \subseteq L$, $|S| \leq |N(S)|$ (Here $N(S) := \{y \in R : \exists\ x \in S, \{x, y\} \in E\}$).*

Proof. To prove an "if and only if" statement we need to show two directions:

- First we show that $(G$ has a matching of size $|L|) \Rightarrow (\forall\, S \subseteq L, |S| \leq |N(S)|)$. Let M be a matching in G of size $|L|$. Since every edge of G has one of its endpoints in L and no two edges in M can share an endpoint, we know that each vertex in L must be matched under M.

  Let $S \subseteq L$ be arbitrary. Let $T = \{y \in R : \exists\ x \in S, \{x, y\} \in M\}$. Note that since $M \subseteq E$, $T \subseteq N(S)$. We have shown above that M must match all vertices of L. In particular M matches all vertices in S. Since M is a matching, no two of its edges share an endpoint, so each vertex of S is matched to a distinct vertex in R. Therefore $|S| = |T| \leq |N(S)|$.

- Now we show that $(\forall\, S \subseteq L, |S| \leq |N(S)|) \Rightarrow (G$ has a matching of size $|L|)$. We will prove the contrapositive: If the size of a maximum matching in G is less than $|L|$, then there exists a set $S \subseteq L$ for which $|S| > |N(S)|$.

  Let M be a maximum matching in G and assume $|M| < |L|$. Then M does not match all of the vertices in L. Let $x \in L$ be a vertex which is not matched by M. Let

  $$T := \{v \in V : v \text{ can be reached through an } M\text{-alternating path starting from } x\},$$

By Theorem 89, since M is a maximum matching, there are no M-augmenting paths in G. Therefore x cannot reach an unmatched vertex along an alternating path, so every vertex in $T - \{x\}$ is matched. Let $T_L := T \cap L$ and $T_R := T \cap R$.

We note that every M-alternating path starting from x must begin with an edge of $E \setminus M$ (since x is unmatched) so it will travel from L to R on edges in $E \setminus M$ and from R to L on edges in M. Therefore, for every vertex $y \in T_R$, there is an M-alternating path from x to y which ends with an edge in $E \setminus M$. Let $z \in L$ be the vertex that y is matched with. Adding the edge y, z to the M-alternating path from x to y gives an M-alternating path from x to z, so $z \in T_L$. This means that every vertex of T_R is matched to a vertex of T_L, but T_L also contains the extra unmatched vertex x, so $|T_L| \geq |T_R| + 1$

Now we will show that $N(T_L) \subseteq T_R$. Let $v \in T_L$ be arbitrary. We need to show that $N(v) \subseteq T_R$. If $v = x$, this is true, since every neighbor w of v can be reached by the length one M-alternating path (v, w). Otherwise, v is matched by M, let $u \in R$ be the vertex that v is matched with. Since $v \in T$, there exists an M-alternating path P from x to v. As noted above, this path reaches vertices in L by following edges in M, so the last edge of P must be $\{u, v\}$. Removing this edge from P gives an M-alternating path from x to u, so $u \in T_R$. Similarly, any other vertex in R which is visited by P must be in T_R (since we can simply remove all the edges of P after reaching that vertex to get an alternating path from x to that vertex). Let w be a neighbor of v which is not visited by P. Since M can match v with only one other vertex, $\{v, w\} \notin M$. Therefore adding $\{v, w\}$ to P gives an M-alternating path from x to w, so $w \in T_R$ as well. Thus, $N(v) \subseteq T_R$, so $N(T_L) \subseteq T_R$.

Putting together these facts we have that

$$|T_L| > |T_L| - 1 \geq |T_R| \geq |N(T_L)|.$$

So G has a set $S \subseteq L$ for which $|S| > |N(S)|$.

□

Hall's theorem is useful for proving that a graph has a perfect matching without needing to explicitly construct one. For example, Hall's theorem allows us to easily prove the following theorem:

Theorem 91. *If k is a positive integer and $G = (L \cup R, E)$ is a k-regular bipartite graph, then G has a perfect matching.*

Proof. We note that since every vertex in L has degree k and every edge has one endpoint in L, $m = |L|k$. Similarly since every vertex in R has degree k and every edge has one endpoint in R, $m = |R|k$. Therefore, $|L| = |R|$, so a matching which matches all of the vertices of L would be a perfect matching.

To show that G has a matching which matches all vertices of L we will show that it satisfies Hall's condition: $\forall\, S \subseteq L, |S| \leq |N(S)|$. Let $S \subseteq L$ be arbitrary. Since every vertex in L has degree k, there are exactly $|S|k$ edges incident to one of the vertices in S. Since each vertex in R has degree k, no vertex in R may be incident to more than k of these edges. So the edges of vertices in S are incident to at least $|S|k/k = |S|$ vertices in R. Thus, $|N(S)| \geq |S|$, so by Theorem 90, G has a perfect matching. □

19.3 Finding Maximum Matching in Bipartite Graphs

So how do we find a maximum matching in a bipartite graph? The fastest and easiest way is via a reduction to max-flow!

Given a bipartite graph $G = (V, E)$ with bipartition $V = L \cup R$, we construct a flow network $G' = (V', E')$ as follows:

1. $V' = V \cup \{s, t\}$, i.e. we add two new vertices to G to serve as our source and sink.
2. E' contains the following edges:

- For each $u \in L$, E' contains an edge (s, u)
- For each $\{u, v\} \in E$, with $u \in L$, $v \in R$ E' also contains (u, v) (i.e. we direct the edges from L to R).
- For each $v \in R$, E' contains an edge (v, t).
- All edges in E' have capacity 1.

We note that every integral flow in G' corresponds to a matching in G and vice-versa:

- Since the edges (s, u) each have capacity 1, an integer flow in G' can send flow out of $u \in L$ through at most 1 edge, and since the edges (v, t) also have capacity 1, an integer flow in G' can send flow into a vertex $v \in R$ through at most 1 edge, so for any integer flow f, the edges of G that f sends positive flow through form a matching in G.
- Conversely, given a matching M in G, we can create a corresponding flow in G' by sending 1 unit of flow from s to each vertex in L which is matched by M, sending 1 unit of flow through each edge of M, and sending 1 unit of flow from each vertex of R which is matched by M to t.

Since all the edge capacities in G' are integers, we know that G has an integral maximum flow. G' has $O(n)$ vertices and $O(m + n)$ edges, so we can compute a max flow in G' (and thus find a maximum matching in G) using the algorithm of CKLPGS in $O((m + n)^{1+o(1)})$ time.

Note that if we computed this maximum flow using the Ford-Fulkerson (or Edmonds-Karp) algorithm, then the augmenting paths in the residual graph that we push flow through while running the algorithm would correspond to M-augmenting paths in G, demonstrating that our two definitions of "augmenting paths" in fact have a close relationship.

19.4 Stable Matchings

In the United States, doctors who have graduated from medical school must go through a residency program, where they obtain hands-on experience working in a hospital, before they can obtain a medical license.

This process can be fiercely competitive, from the perspective of both the young doctors and the hospital residency programs. By 1950, hospitals desperate to secure the top students were issuing "exploding" residency offers, where they would call a final-year medical student to offer them admission to a residency program and require them to decide whether or not to accept the offer on the spot before hanging up the phone.

Frustrated by these conditions, students appealed to their schools, and in 1952 a non-profit known as the National Resident Matching Program was created to fairly match doctors and hospitals for residency.

Every spring, hospitals submit a ranked list of students they would like to admit, and students submit ranked lists of hospitals where they would like to go for residency after graduation. On the third Friday in March (match day), the NRMP releases an assignment of students to hospitals, avoiding months of stressful negotiations between individual students and programs.

But how does the NRMP compute an assignment that keeps everyone happy? How do we even define what a "fair" assignment means? That is the problem that we'll discuss in this section.

The algorithm that we discuss in this section is essentially the algorithm that the NRMP actually uses to solve a massive doctor-hospital pairing problem in the real world. However, in order to simplify the algorithm and analysis a bit, we will focus on an unrealistic version of the problem with the following assumptions:

- We have n doctors and n hospitals (for some positive integer n).
- Each hospital is looking to hire only a single doctor.
- Each doctor has a preference list ranking all the hospitals from the one that they would most like to work at to the one that they would least like to work at. We will assume that these are represented as a 2-D array D where $D[i, j]$ is Doctor #i's j'th favorite hospital.

- Each hospital has a preference list ranking all of the doctors from the one that they would most like to hire to the one that they would least like to hire. We will assume that these are represented as a 2-D array H where $H[i, j]$ is Hospital $\#i$'s j'th favorite doctor.

Under these assumptions, we can view the problem as finding a perfect matching in a bipartite graph, where L is the set of doctors and R is the set of hospitals. We don't want just any perfect matching, though (otherwise we could just pair the i'th doctor with the i'th hospital for each $1 \leq i \leq n$), we want to find a *stable* perfect matching.

For a matching M between doctors and hospitals, we say that a doctor-hospital pair (d, h) is an **unstable pair** (you may also see the term "blocking pair" used) if d and h are not matched by M, but they each prefer each other over their matches under M. We say that a matching is **stable** if it has no unstable pair. Stability is a desirable property for a matching since it ensures that no doctor and hospital have an incentive to ignore the NRMP's assignment and cut their own backroom deal. Note that a stable matching does not mean that everyone is happy with their assignment. Some doctor d may get their last choice hospital, but as long as every other hospital other than the one d is matched with has been assigned a doctor they like better than d, none of them have an incentive to take d, and there is no unstable pair.

It is not obvious from the definition that a stable perfect matching always exists. We will prove that one does, however, by giving an efficient algorithm to find one. This algorithm has been in use by the NRMP since 1952, but it is named after David Gale and Lloyd Shapley, who proved in 1962 that the algorithm always produces a stable matching. This work contributed to Shapley being awarded the Nobel prize in economics in 2012.

```
GaleShapley(D, H)
  M ← ∅
  While |M| ≠ n:
      h ← a hospital not matched by M
      d ← the top doctor on h's preference list
      If d is not matched by M:
          Add (d, h) to M
      Else if d prefers h to the hospital h' they are currently matched with:
          Remove (d, h') from M
          Add (d, h) to M
      Remove d from h's preference list
  Return M
```

We can view this algorithm as hospitals making a sequence of offers to doctors. When a doctor receives her first offer, they tentatively accept it. Then, when they receive a new offer, if it is better than their current offer, they reject the previous hospital and tentatively accept the new one. Once every doctor has an offer, the doctors all accept their current offers.

Theorem 92. GaleShapley(D, H) *returns a stable perfect matching. Furthermore,* GaleShapley *can be implemented to run in* $O(n^2)$ *time.*

Proof. We first show that GaleShapley returns a perfect matching. We note that at every point during the algorithm M is always a valid matching, since we only add an edge (d, h) to M when d and h are both currently unmatched (possibly because we just discarded an edge that was previously used to match d). We also note that once a doctor is matched by M, they stay matched in every future iteration of the while loop (a doctor only discards their current hospital to accept a better offer). An unmatched doctor accepts any offer they received, becoming matched. Thus, if every doctor eventually receives an offer, then every doctor is eventually matched.

As long as some doctor is unmatched, some hospital will also be unmatched and will make offer to the top doctor they have yet to scratch off their preference list. Since every hospital begins with all n doctors on its preference list, this process can continue for only a finite number of iterations until either all doctors are matched and the algorithm terminates or some hospital reaches the end of its preference list. If a hospital h reaches the last doctor on its preference list it means that every

other doctor must already be matched to one of the other $n-1$ hospitals (otherwise they would not have rejected h's earlier offer), so the final doctor must be unmatched and will accept h's offer, terminating the algorithm.

Next we show that the perfect matching that GaleShapley returns is stable. Let (d, h) be any doctor-hospital pair that are *not* matched by M. We need to show that (d, h) is not an unstable pair. Let d' be the doctor that h is matched with by M. We consider two cases:

- If h prefers d' over d, then (d, h) is not unstable, because h would not prefer to switch to d.
- If h prefers d over d', then h must have made an offer to d before making its offer to d'. Since d and h did not end up matched in M, d must at some point have rejected h for a hospital h' that d prefers over h. After rejecting h for h', d would only reject h' for another hospital that they prefer even more. Thus, whichever hospital d ends up matched with under M must be a hospital that they prefer to h. Therefore, d would not prefer to switch to h and (d, h) is not an unstable pair.

Thus in both cases, (d, h) is not unstable. Since (d, h) was an arbitrary unmatched pair, we conclude that M has no unstable pairs and thus is a stable matching.

Finally, we analyze the runtime. We note that we can implement the hospitals moving down their preference lists by storing each hospital's preference list in a linked list. We can create these n length n linked lists in $O(n^2)$ time, and using them we can get a hospital's most preferred doctor or remove that doctor from the preference list in constant time.

Since each iteration of the loop removes a doctor from a hospital's preference list, the algorithm's while loop runs for $O(n^2)$ iterations. We first show that we can find an unmatched hospital in constant time. To do this, we maintain a list of unmatched hospitals. Each iteration we pop the first hospital h from this list. If we fail to match h, we insert h back at the front of the list, otherwise we insert the hospital that h displaced (if any) into the list.

As noted above, we can get the top doctor on a hospital's preference list in constant time. We can check whether a doctor is matched in constant time by maintaining an array storing each doctor's current match (or Null if they are unmatched).

Finally, we need to show that we can determine in constant time whether d prefers h over the hospital h' that d is currently matched with. With our stated input representation, where each doctor has an array listing their hospitals from most preferred to least preferred, determining whether d prefers h over h' takes $O(n)$ time, we need to iterate through the whole list looking for the two hospitals. Thus, before the loop we preprocess the doctors' preference lists into a more convenient format. Call the hospitals $h_1, \ldots, h_n$. For each doctor d we create an array A_d where $A_d[i]$ is the rank of h_i in d's preference list (so if d's number one choice is h_5, then $A_d[5] = 1$). We can create this array A_d from the preference list in $O(n)$ time (we simply iterate through the list once, filling in the appropriate entries of A_d as we go). Thus, we can do this for all doctors in $O(n^2)$ time before the loop, and then in the loop we can determine whether d prefers h_i to h_j by simply comparing $A_d[i]$ and $A_d[j]$.

Thus, with $O(n^2)$ preprocessing, we can ensure that each iteration of the loop runs in $O(1)$ time. Since the loop runs $O(n^2)$ times, this implies that the total runtime for the algorithm is $O(n^2)$.

□

19.5 Comprehension Questions

1. True or false? If M is a maximal matching in a graph G, then G does not contain any M-augmenting path.
2. If $n \geq 4$ is an integer and G is a graph consisting of a cycle on n vertices, then G has a perfect matching.
3. Let G be a 3-regular bipartite graph with n vertices. What can we say about the size of a maximum matching in G? (select the most precise true statement)

(a) The size of a maximum matching in G is n.

(b) The size of a maximum matching in G is at most n.

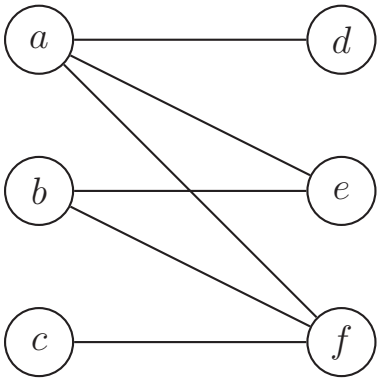
(c) The size of a maximum matching in G is $n/2$.

(d) The size of a maximum matching in G is at most $n/2$.

(e) The size of a maximum matching in G is $n/3$.

(f) The size of a maximum matching in G is at most $n/3$.

4. $M = \{\{a, e\}, \{b, f\}\}$ is a matching in the bipartite graph below. Give an example of an M-augmenting path in the graph.



5. Consider an instance of the stable matching problem where we wish to pair up 4 doctors, A, B, C, and D with 4 hospitals W, X, Y, and Z. The preference lists of the doctors and hospitals are given below (from most preferred to least preferred):

$$\begin{array}{ll} A : W, X, Y, Z & W : B, D, A, C \\ B : W, X, Y, Z & X : A, B, C, D \\ C : W, X, Y, Z & Y : B, A, D, C \\ D : W, X, Y, Z & Z : D, A, C, B \end{array}$$

(a) A-W,B-X,C-Y,D-Z is not a stable matching for these preference lists. Show why by giving an unstable pair for this matching:

(b) Give an example of a stable matching for this set of preference lists:

19.6 Exercises

1. Suppose that you are given an $n \times n$ grid where some of the cells are white and others are black. Your goal is to mark n of the *white* cells in such a way that every row contains exactly one marked cell, and every column contains exactly one marked cell. Give an algorithm which takes as input an $n \times n$ array $White[1 \ldots n][1 \ldots n]$ of booleans, where $White[i][j]$ is true if and only if the square at row i and column j is white. Your algorithm should either output a list of n (i, j) pairs indicating white cells to mark, or should correctly report that there is no valid set of cells in this grid. Your algorithm should run in polynomial time.

   For example, if $White$ is the array:

$$\begin{bmatrix} false & true & true \\ true & false & true \\ false & true & false \end{bmatrix}$$

then your algorithm should output $[(1,3),(2,1),(3,2)]$ (or some permutation of this list), corresponding to marking the top right, left middle and bottom middle cells.

2. Let $G = (V, E)$ be a (not necessarily bipartite) unweighted undirected graph. Initially none of the vertices of G are marked. Two friends, Alice and Bob, play the following game on this graph:

   - Alice begins the game by choosing any vertex of G and marking it.
   - Bob chooses an unmarked vertex adjacent to the one that Alice just marked and marks it.
   - Alice chooses an unmarked vertex adjacent to the one that Bob just marked and marks it.
   - The players continue to alternate turns, and on each turn a player marks an unmarked vertex adjacent to the vertex their opponent marked on the previous turn.
   - When a player has no legal move (because all of the vertices neighboring the one their opponent just marked have already been marked earlier in the game), they lose.

   In this problem you will help Alice and Bob develop winning strategies for this game.

   (a) Prove that if G has a perfect matching then Bob has a strategy which guarantees him a win no matter which vertices Alice chooses.

   (b) Prove that if G does not have a perfect matching then Alice has a strategy which guarantees her a win no matter which vertices Bob chooses.

3. A *cycle cover* of a directed graph G is a collection of cycles in G such that each vertex of G is in exactly one of the cycles. Give an efficient algorithm to determine whether or not a directed graph has a cycle cover (Hint: Use bipartite matching).

4. We saw that the Gale-Shapley algorithm can be used to find a stable matching between newly graduated doctors and hospitals looking to hire them, in the special case where each hospital has only one open position, each doctor has ranked all the hospitals, and each hospital has ranked all of the doctors. In this problem, you will generalize that result to a more realistic scenario.

   Suppose now that there are x doctors and y hospitals. Each hospital may have a different number of available positions. Each doctor has a preference list which is a partial ranking of the hospitals, i.e. they have ranked some *subset* of the hospitals from most wanted to least wanted (each doctor separately chooses their own set of hospitals to rank). Different doctors may rank different numbers of hospitals. Similarly, each hospital has a preference list which is a ranking for some subset of the doctors. In this context, we say that a doctor prefers a hospital that they have ranked over every hospital they have not ranked, and a doctor has no preference between hospitals they have not ranked (and vice-versa for hospitals). We say that doctor d and hospital h form an unstable pair if doctor d prefers hospital h over the hospital she is currently assigned to, and either: (i) hospital h has an open position (they are not hiring their maximum number of doctors) or (ii) hospital h prefers doctor d over at least one of the doctors they are hiring.

   Give an efficient algorithm which takes as input:

   - The number of doctors, x,
   - The number of hospitals, y,
   - A 1-D array $Positions[1, \ldots, y]$, where $Positions[i]$ is the maximum number of doctors that the ith hospital is willing to hire,
   - A 2-D array $Doctors[1, \ldots x][]$ (with rows of uneven length), where $Doctors[i]$ is the preference list of the ith doctor, and
   - A 2-D array $Hospitals[1, \ldots, y][]$ (with rows of uneven length), where $Hospitals[i]$ is the preference list of the ith hospital.

Your algorithm should output an assignment of doctors to hospitals where:

- Every doctor is assigned to a hospital,
- At most $Positions[i]$ doctors are assigned to hospital i, for every i, and
- There are no unstable pairs.

Note that you may need to pair up a doctor and a hospital which have not put each other on their preference lists. You may assume that the total number of positions is greater than or equal to the total number of doctors.

5. For a graph $G = (V, E)$ we define a **vertex cover** as a set of vertices $S \subseteq V$ such that for every edge $e \in E$, $e \cap S \neq \emptyset$ (i.e. at least one of e's two endpoints is in S). A minimum vertex cover is one which uses the minimum number of vertices (among all vertex covers in G).

   (a) Prove that if M is a matching and S is a vertex cover then $|S| \geq |M|$.

   (b) Give an example of a graph where the size of a minimum vertex cover is larger than the size of a maximum matching (Hint: your graph will not be bipartite)

   (c) It turns out that in bipartite graphs, the size of a maximum matching is equal to the size of a minimum vertex cover (this is known as Kőnig's theorem). In the remaining parts of this problem we will prove this theorem.

   Let G be a bipartite graph with bipartition L, R. From part (a) we already know that the size of the minimum vertex cover is at least the size of the maximum matching, so we just need to prove that if M is a maximum matching in a bipartite graph, then there exists a vertex cover S with $|S| = |M|$.

   Begin by proving that if M matches all of the vertices in L, then G has a vertex cover of size $|M|$.

   (d) Now suppose that M does not match all of the vertices of L. Let L_M be the set of vertices from L that M matches. We define a new *directed* graph G' on vertex set $\{s\} \cup L \cup R$ with the following edges:

   - For each edge $\{u, v\} \in M$ with $u \in L, v \in R$, G' has an edge (v, u).
   - For each edge $\{u, v\} \in E \setminus M$, G' has an edge (u, v).
   - For each vertex $u \in L \setminus L_M$, G' has an edge (s, u)

   Let $X := \text{reach}_{G'}(s)$. Prove that $S := (L \setminus X) \cup (R \cap X)$ is a vertex cover for G.

   (e) Finish the proof by showing that every vertex of S is matched by a distinct edge from M and therefore $|M| \geq |S|$.