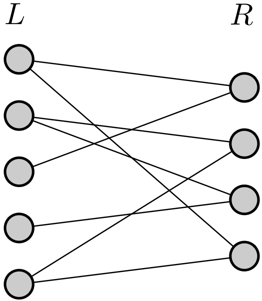
Bipartite Graphs and Partially Ordered Sets

Zachary Friggstad

Programming Club Meeting

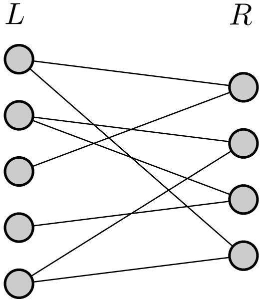
Bipartite Recognition

An undirected graph $G = (V, E)$ is bipartite if V can be partitioned into two sides L, R where all edges have an endpoint in both L and R.



Bipartite Recognition

An undirected graph $G = (V, E)$ is bipartite if V can be partitioned into two sides L, R where all edges have an endpoint in both L and R.



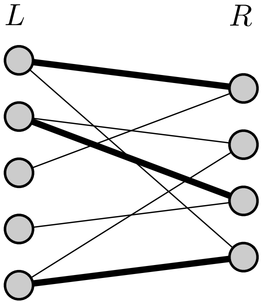
How to recognize bipartite graphs? Pick a vertex v, add v to L, add its neighbours to R, add their unprocessed neighbours to L, etc.

Now check if any edge has both endpoints on the same side.

```
typedef vector<vector<int>> graph;

graph g; //assume is already filled
vector<int> mark(g.size(), -1); //-1 means not processed

// uses a depth-first search
bool partition(int v, int side) {
   if (left[v] != -1) return mark[v] == side;
   mark[v] = side;
   for (auto u : g[v])
       if (!partition(u, 1-side)) return false;
   return true;
}

bool bip = true;
// run the dfs in each component
for (int u = 0; u < n && bip; ++u)
   bip &= (mark[u] != -1 || partition(u, 0));
//if bip == true, then {u : mark[u] = 0} is the left side
```

Bipartite Matchings

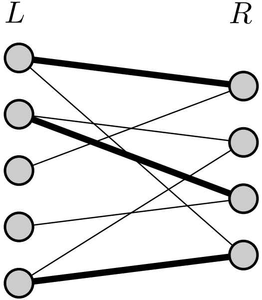
Let $G = (V, E)$ be a graph. A matching is a subset $M \subseteq E$ such that each vertex $v \in V$ is the endpoint of *at most* one edge in M.



i.e. M pairs up some vertices ($M \equiv$ thick edges).

Bipartite Matchings

Let $G = (V, E)$ be a graph. A matching is a subset $M \subseteq E$ such that each vertex $v \in V$ is the endpoint of *at most* one edge in M.



i.e. M pairs up some vertices ($M \equiv$ thick edges).

Optimization Question

Find the largest possible matching M.

Can find in polynomial time for **any** graph, but the algorithm is a bit too intricate for the contest setting.

However, the question is frequently asked in a programming contest if G is bipartite.

Can find in polynomial time for **any** graph, but the algorithm is a bit too intricate for the contest setting.

However, the question is frequently asked in a programming contest if G is bipartite.

Basic Approach
Let $n = |V|, m = |E|$.

There is a $O(n + m)$-time algorithm that does the following:

Given a matching M, either finds a larger matching or else correctly determines M is a maximum-size matching.

Can find in polynomial time for **any** graph, but the algorithm is a bit too intricate for the contest setting.

However, the question is frequently asked in a programming contest if G is bipartite.

Basic Approach
Let $n = |V|, m = |E|$.

There is a $O(n + m)$-time algorithm that does the following:

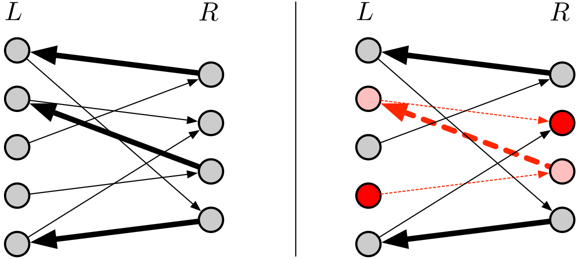
Given a matching M, either finds a larger matching or else correctly determines M is a maximum-size matching.

Iterating the procedure starting with $M = \emptyset$ finds a maximum matching in $O(mn + n^2)$ time.

Augmenting Paths

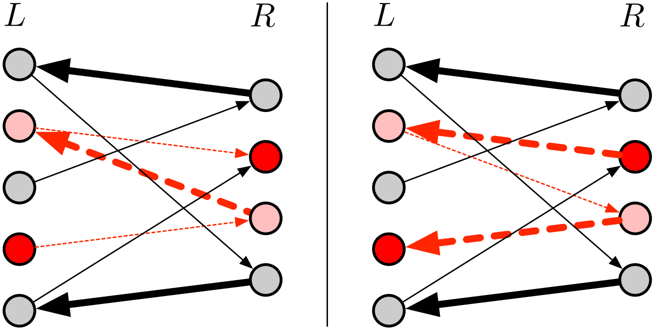
Given a matching M, direct all edges e as follows:

- From L to R if $e \notin M$
- From R to L if $e \in M$



Find an M-alternating path P: a path from an unmatched vertex in L to an unmatched vertex in R.

Flip the directions of the edges in P: the matching is larger.



Lemma

If M is not a maximum matching, there is an M-alternating path.

Idea:

Let M^* be any matching. Then $M \cup M^*$ (keep doubles) is comprised of paths and cycles that alternate between M and M^*.

If $|M| < |M^*|$, then some path starts and ends with an edge in M^*. This is an M-alternating path.

Lemma

If M is not a maximum matching, there is an M-alternating path.

Idea:
Let M^* be any matching. Then $M \cup M^*$ (keep doubles) is comprised of paths and cycles that alternate between M and M^*.

If $|M| < |M^*|$, then some path starts and ends with an edge in M^*. This is an M-alternating path.

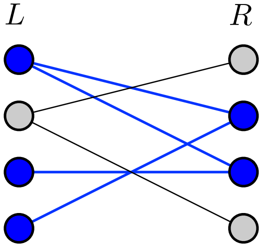
Pseudocode:

- $M = \emptyset$
- While there is an M-alternating path P
- Update $M \leftarrow M \oplus P$ (toggle/flip edges of P)
- Return M

```
int left; //# nodes on left
graph g;  //say L is indexed from 0 to left-1
vector<int> match(g.size(), -1), seen(left, -1);

bool augment(int u, int cno) { //find a path via dfs
  if (seen[u] == cno) return false;
  seen[u] = cno;
  for (auto v : g[u])
    if (match[v] == -1 || augment(match[v], cno)) {
      match[v] = u; match[u] = v; // flip the edges
      return true;
    }
  return false;
}

int match() {
  int cnt = 0;
  // can show we only need to search from each vertex once
  for (int u = 0; u < left; ++u)
     if (augment(u, u)) ++cnt;
  return cnt;
}
```

Hall's Theorem.

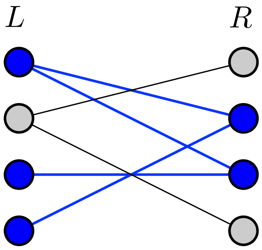
There is a matching M of size $|L|$ if and only if for every $X \subseteq L$, the number of nodes in R adjacent to some vertex in X is at least $|X|$.



Picture: No way to match all three highlighted nodes in L.

Hall's Theorem.

There is a matching M of size $|L|$ if and only if for every $X \subseteq L$, the number of nodes in R adjacent to some vertex in X is at least $|X|$.

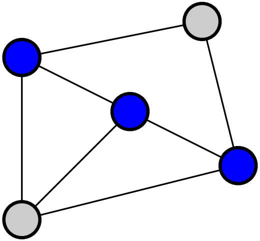


Picture: No way to match all three highlighted nodes in L.

To find such an X, if max matching $|M| < |L|$ let X be all nodes on L reachable from M-alternating paths from unmatched nodes.

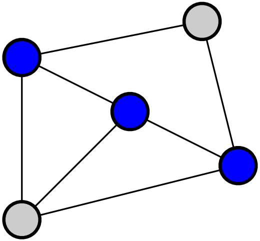
Exercise: Prove such X has $< |X|$ neighbours.

A vertex cover is a set C of nodes so each edge e has at least one endpoint in C.



Finding a minimum-size vertex cover is NP-hard in general, but easy in bipartite graphs.

A vertex cover is a set C of nodes so each edge e has at least one endpoint in C.



Finding a minimum-size vertex cover is NP-hard in general, but easy in bipartite graphs.

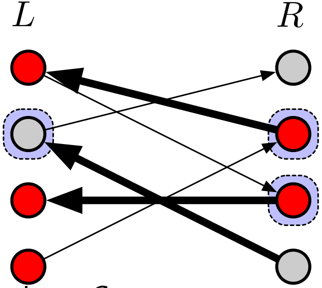
Theorem (Vizing)

If C is a min vertex cover and M a max matching in a bipartite graph, then $|C| = |M|$.

Again, let M be a max-matching.

Let S be the set of all vertices reachable by an M-alternating path starting at the unmatched nodes on the left.

Let $C = (L - S) \cup (R \cap S)$. This is a vertex cover.

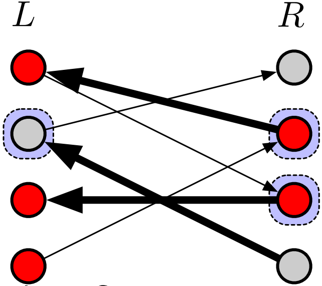


red $\equiv$ reachable, blue halo $\equiv$ C

Again, let M be a max-matching.

Let S be the set of all vertices reachable by an M-alternating path starting at the unmatched nodes on the left.

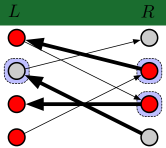
Let $C = (L - S) \cup (R \cap S)$. This is a vertex cover.



red $\equiv$ reachable, blue halo $\equiv$ C

Side Note: Can show $L \cap S$ is a "Hall Set" (i.e. fewer than $|L \cap S|$ neighbours) if $|M| < |L|$.

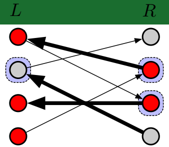
Steps to the proof.



Showing C is a vertex cover

Show no edge $e = uv$ has $u \in L \cap S, v \in R - S$ using definition of "reachability" for S (two cases, if $e \in M$ or $e \notin M$).

Steps to the proof.



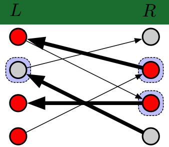
Showing C is a vertex cover

Show no edge $e = uv$ has $u \in L \cap S, v \in R - S$ using definition of "reachability" for S (two cases, if $e \in M$ or $e \notin M$).

Each $e \in M$ is covered by only one vertex in C

Otherwise a contradiction to reachability for S.

Steps to the proof.

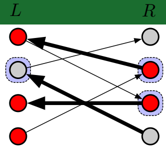


Showing C is a vertex cover
Show no edge $e = uv$ has $u \in L \cap S, v \in R - S$ using definition of "reachability" for S (two cases, if $e \in M$ or $e \notin M$).

Each $e \in M$ is covered by only one vertex in C
Otherwise a contradiction to reachability for S.

No unmatched $u \in L$ is in C
Because unmatched $u \in L$ are trivially reachable.

Steps to the proof.



Showing C is a vertex cover
Show no edge $e = uv$ has $u \in L \cap S, v \in R - S$ using definition of "reachability" for S (two cases, if $e \in M$ or $e \notin M$).

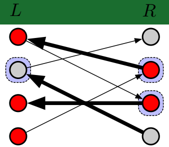
Each $e \in M$ is covered by only one vertex in C
Otherwise a contradiction to reachability for S.

No unmatched $u \in L$ is in C
Because unmatched $u \in L$ are trivially reachable.

No unmatched $v \in R$ is in C
Otherwise there is an M-augmenting path, so M is not maximum.

Therefore $|C| = |M|$.

Steps to the proof.



Showing C is a vertex cover
Show no edge $e = uv$ has $u \in L \cap S, v \in R - S$ using definition of "reachability" for S (two cases, if $e \in M$ or $e \notin M$).

Each $e \in M$ is covered by only one vertex in C
Otherwise a contradiction to reachability for S.

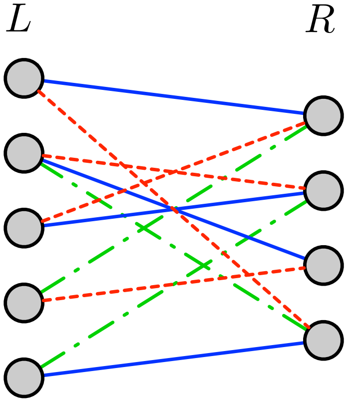
No unmatched $u \in L$ is in C
Because unmatched $u \in L$ are trivially reachable.

No unmatched $v \in R$ is in C
Otherwise there is an M-augmenting path, so M is not maximum.

Therefore $|C| = |M|$. But any vertex cover needs $\geq |M|$ vertices just to cover M, **so C is a minimum vertex cover**.

Edge Colouring

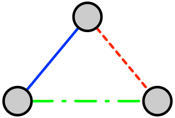
Problem: Colour each edge of a graph so every colour is a matching.



Degree Bounds

Obvious Bound: If Δ is the maximum "degree" of a vertex (# of incident edges), then we need $\geq \Delta$ colours.

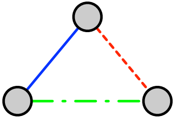
Sometimes more, here $\Delta = 2$ but 3 colours are needed:



Degree Bounds

Obvious Bound: If Δ is the maximum "degree" of a vertex (# of incident edges), then we need $\geq \Delta$ colours.

Sometimes more, here $\Delta = 2$ but 3 colours are needed:

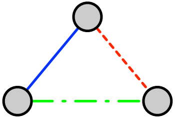


Vizing's Theorem: $\Delta + 1$ colours always suffice.

Degree Bounds

Obvious Bound: If Δ is the maximum "degree" of a vertex (# of incident edges), then we need $\geq \Delta$ colours.

Sometimes more, here $\Delta = 2$ but 3 colours are needed:



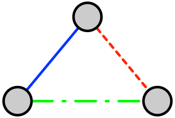
Vizing's Theorem: $\Delta + 1$ colours always suffice.

Unfortunately it is **NP**-hard to determine if Δ colours suffice.

Degree Bounds

Obvious Bound: If Δ is the maximum "degree" of a vertex (# of incident edges), then we need $\geq \Delta$ colours.

Sometimes more, here $\Delta = 2$ but 3 colours are needed:



Vizing's Theorem: $\Delta + 1$ colours always suffice.

Unfortunately it is **NP**-hard to determine if Δ colours suffice.

König: In a bipartite graph, Δ colours suffice.

Algorithm
Colour the edges one at a time.

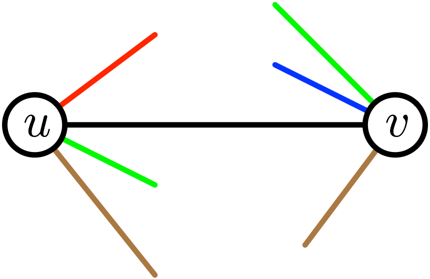
When processing $e = uv$, if there is an available colour, use it!

- That is, if some colour has neither u nor v matched by that colour so far, then use that colour for e.

Otherwise, we will modify the current colouring in $O(n)$ time to ensure there is an available colour.

Modifying the Colouring

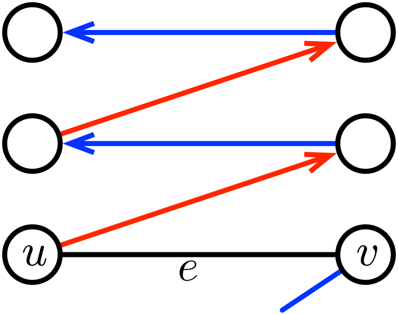
Say we are trying to colour $e = uv$ but none of the Δ colours are free on both u and v.



We still know some colour, say *blue*, is not used on u and some colour, say *red*, is not used on v.

Modifying the Colouring

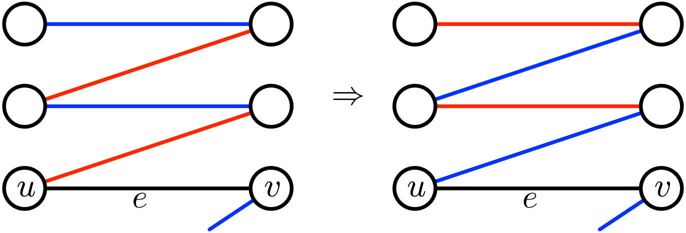
Consider the *maximal* path from u that alternates between *blue* and *red* edges.



This path does not contain v since it goes $L \to R$ along *red* edges.

Modifying the Colouring

Swap the colours of the *blue* and *red* on the path from u.



Now we can colour e *red*.

Running Time: $O(m \cdot n)$.
The colour "swapping" along the path takes $O(n)$ time. Done at most once per edge added.

Partially Ordered Sets

A set of items X with a binary relation $\preceq$ is a POSET if:

- **Reflexivity**: For $a \in X$, $a \preceq a$.
- **Antisymmetry**: For $a, b \in X$, $(a \preceq b$ and $b \preceq a) \Rightarrow a = b$.
- **Transitivity**: For $a, b, c \in X$ $(a \preceq b$ and $b \preceq c) \Rightarrow a \preceq c$.

Write $a \prec b$ for $(a \preceq b$ and $a \neq b)$.

Examples

Partially Ordered Sets

A set of items X with a binary relation $\preceq$ is a POSET if:

- **Reflexivity**: For $a \in X$, $a \preceq a$.
- **Antisymmetry**: For $a, b \in X$, $(a \preceq b \text{ and } b \preceq a) \Rightarrow a = b$.
- **Transitivity**: For $a, b, c \in X$ $(a \preceq b \text{ and } b \preceq c) \Rightarrow a \preceq c$.

Write $a \prec b$ for $(a \preceq b \text{ and } a \neq b)$.

Examples

1. The usual order $\leq$ on numbers, or the lexicographic order on strings (these are total orderings).

Partially Ordered Sets

A set of items X with a binary relation $\preceq$ is a POSET if:

- **Reflexivity**: For $a \in X$, $a \preceq a$.
- **Antisymmetry**: For $a, b \in X$, $(a \preceq b$ and $b \preceq a) \Rightarrow a = b$.
- **Transitivity**: For $a, b, c \in X$ $(a \preceq b$ and $b \preceq c) \Rightarrow a \preceq c$.

Write $a \prec b$ for $(a \preceq b$ and $a \neq b)$.

Examples

1. The usual order $\leq$ on numbers, or the lexicographic order on strings (these are total orderings).
2. A set of boxes where $a \prec b$ means a fits in b.

Partially Ordered Sets

A set of items X with a binary relation $\preceq$ is a POSET if:

- **Reflexivity**: For $a \in X$, $a \preceq a$.
- **Antisymmetry**: For $a, b \in X$, $(a \preceq b \text{ and } b \preceq a) \Rightarrow a = b$.
- **Transitivity**: For $a, b, c \in X$ $(a \preceq b \text{ and } b \preceq c) \Rightarrow a \preceq c$.

Write $a \prec b$ for $(a \preceq b \text{ and } a \neq b)$.

Examples

1. The usual order $\leq$ on numbers, or the lexicographic order on strings (these are total orderings).
2. A set of boxes where $a \prec b$ means a fits in b.
3. Let $G = (X; E)$ be a directed, acyclic graph. Say $u \preceq v$ if there is a $u - v$ path.

Partially Ordered Sets

A set of items X with a binary relation $\preceq$ is a POSET if:

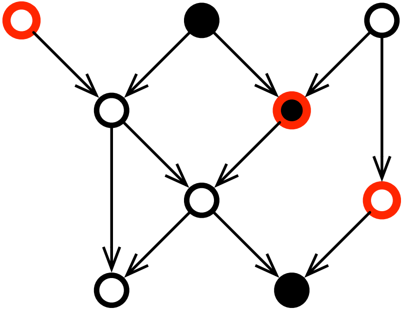
- **Reflexivity**: For $a \in X$, $a \preceq a$.
- **Antisymmetry**: For $a, b \in X$, $(a \preceq b \text{ and } b \preceq a) \Rightarrow a = b$.
- **Transitivity**: For $a, b, c \in X$ $(a \preceq b \text{ and } b \preceq c) \Rightarrow a \preceq c$.

Write $a \prec b$ for $(a \preceq b \text{ and } a \neq b)$.

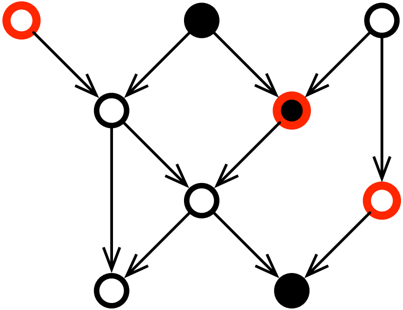
Examples

1. The usual order $\leq$ on numbers, or the lexicographic order on strings (these are total orderings).
2. A set of boxes where $a \prec b$ means a fits in b.
3. Let $G = (X; E)$ be a directed, acyclic graph. Say $u \preceq v$ if there is a $u - v$ path.
4. Let $G = (X; E)$ be a graph with nonzero edge distances. Fix $r \in V$. Say $u \preceq v$ if some shortest $r - v$ path passes through u.

Can view as a directed, acyclic graph. We can omit missing edges inferred by transitivity (a Hasse diagram).

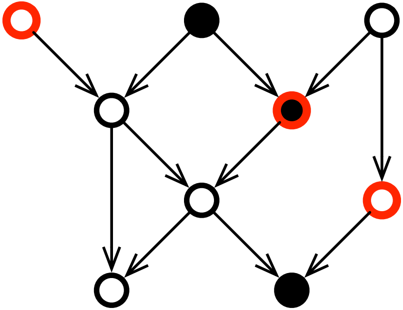


Can view as a directed, acyclic graph. We can omit missing edges inferred by transitivity (a Hasse diagram).



A chain is a set $C \subseteq X$ that can be totally ordered: i.e. every pair $a, b \in C$ has $a \prec b$ or $b \prec a$. (e.g. solid black nodes)

Can view as a directed, acyclic graph. We can omit missing edges inferred by transitivity (a Hasse diagram).

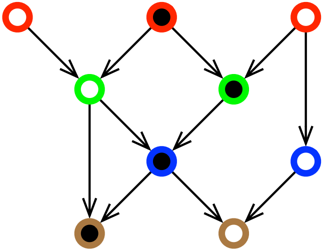


A chain is a set $C \subseteq X$ that can be totally ordered: i.e. every pair $a, b \in C$ has $a \prec b$ or $b \prec a$. (e.g. solid black nodes)

An antichain is a set $A \subseteq X$ where no two $a, b \in X$ have $a \prec b$ **or** $b \prec a$. i.e. no two items in A are comparable. (e.g. thick red outline)

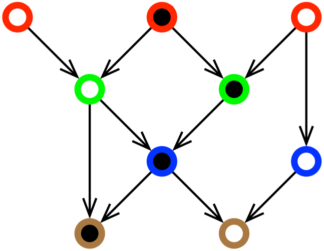
Theorem

Longest chain = minimum # of antichains to cover all nodes.



Theorem

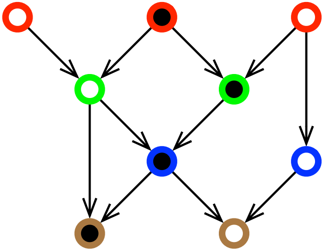
Longest chain = minimum # of antichains to cover all nodes.



Let $\ell[v]$ be the length of the longest chain starting at v (dynamic programming). **Figure**: longest chain $\equiv$ black nodes.

Theorem

Longest chain = minimum # of antichains to cover all nodes.

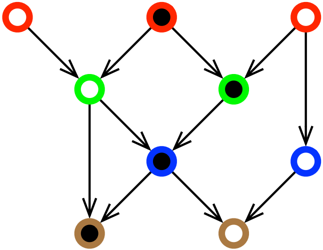


Let $\ell[v]$ be the length of the longest chain starting at v (dynamic programming). **Figure**: longest chain $\equiv$ black nodes.

Then for all $k \in \mathbb{Z}$, $\{v : \ell[v] = k\}$ is an antichain.

Theorem

Longest chain = minimum # of antichains to cover all nodes.



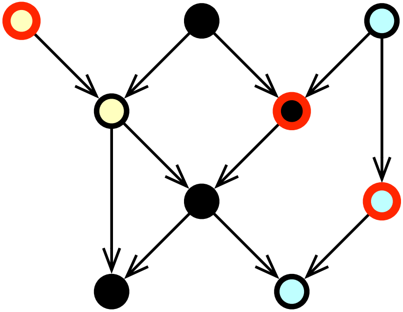
Let $\ell[v]$ be the length of the longest chain starting at v (dynamic programming). **Figure**: longest chain $\equiv$ black nodes.

Then for all $k \in \mathbb{Z}$, $\{v : \ell[v] = k\}$ is an antichain.

Can compute in $O(|V| + |E|)$ time.

Theorem (Dilworth's Theorem)

Largest antichain = minimum # of chains to cover all nodes.

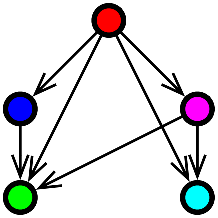


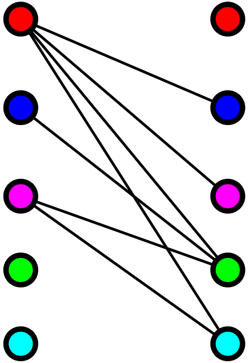
Figure

Nodes with a red outline form a maximum antichain. The colours filling the nodes partition the nodes into three chains.

Form an auxiliary bipartite graph: a copy of X on each side.

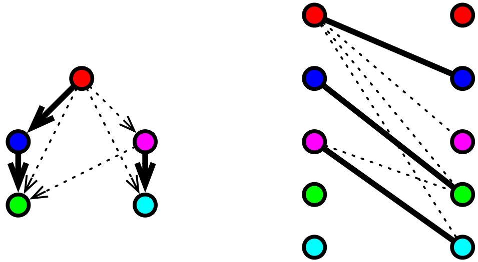
Add edge $u \in L$ to $v \in R$ if $u \prec v$ (but **NOT** $u = v$).





If starting with a Hasse diagram, don't forget edges implied by transitivity!

Bipartite matching of size $k \equiv$ chain cover of size $n - k$.



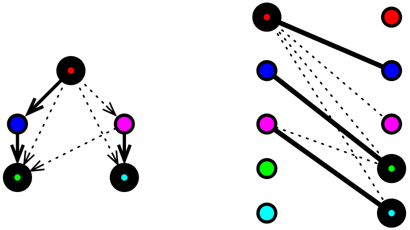
Note a singleton node $v \in X$ forms its own chain: this corresponds to no copy of v being matched.

So find a maximum bipartite matching to find minimum # of chains to cover all nodes.

Running time: $O(n \cdot m)$ where m is the number of $u \prec v$ pairs.

Finally, Maximum Antichains

In the bipartite graph, let C be a min. vertex cover (thick vertices).



Can show minimality of C and transitivity means no $v \in X$ has both copies in C.

Back in the poset, C covers all directed edges and has size $|M|$.

Therefore, $X - C$ (middle nodes in the poset above) is an antichain of size $n - |M| = (\#$ of chains in cover).

UVa 11396 - Claw Decomposition

UVa 11396 - Claw Decomposition

Draw an example of when a claw decomposition is possible.

UVa 11396 - Claw Decomposition

Draw an example of when a claw decomposition is possible.

Are there edges between centres of claws? Are there edges between "nails" of the claws?

UVa 11396 - Claw Decomposition

Draw an example of when a claw decomposition is possible.

Are there edges between centres of claws? Are there edges between "nails" of the claws?

If G is bipartite, then the set of nodes on one side form centres of claws in a decomposition. Conversely, if there is a claw decomposition then the centres of claws form one side and the "nails" form the other side of a bipartite.

UVa 11396 - Claw Decomposition

Draw an example of when a claw decomposition is possible.

Are there edges between centres of claws? Are there edges between "nails" of the claws?

If G is bipartite, then the set of nodes on one side form centres of claws in a decomposition. Conversely, if there is a claw decomposition then the centres of claws form one side and the "nails" form the other side of a bipartite.

So just check if G is bipartite!

UVa 11331 - Joys of Farming

UVa 11331 - Joys of Farming

The graph has to be bipartite: $O(a + b + c)$ check.

UVa 11331 - Joys of Farming

The graph has to be bipartite: $O(a + b + c)$ check.

In each component, there are only two ways to do it: cows on the left, bulls on the right or vice-versa.

UVa 11331 - Joys of Farming

The graph has to be bipartite: $O(a + b + c)$ check.

In each component, there are only two ways to do it: cows on the left, bulls on the right or vice-versa.

Dynamic programming!

UVa 11331 - Joys of Farming

The graph has to be bipartite: $O(a+b+c)$ check.

In each component, there are only two ways to do it: cows on the left, bulls on the right or vice-versa.

Dynamic programming!

Built this table:
$f[i,b] = \texttt{true}$ if it is possible to assign precise b bulls to the first i components, `false` otherwise.

Can fill in $O((b+c)^2)$ time.

UVa 10122 - Mysterious Mountain

UVa 10122 - Mysterious Mountain

A form of Bipartite matching?

UVa 10122 - Mysterious Mountain

A form of Bipartite matching?

Calculate minimum time for person p to reach endpoint e. If line pe has endpoints e' below it, rotate the line to pass through e, e' and repeat until no endpoints below it.

UVa 10122 - Mysterious Mountain

A form of Bipartite matching?

Calculate minimum time for person p to reach endpoint e. If line pe has endpoints e' below it, rotate the line to pass through e, e' and repeat until no endpoints below it.

Now find a bipartite matching minimizing the maximum edge cost.

UVa 10122 - Mysterious Mountain

A form of Bipartite matching?

Calculate minimum time for person p to reach endpoint e. If line pe has endpoints e' below it, rotate the line to pass through e, e' and repeat until no endpoints below it.

Now find a bipartite matching minimizing the maximum edge cost.

Binary search!

Running Time: $O(N^3)$ time to build the graph, $O(N^3 \cdot \log N)$ to binary search (just search over the $O(N^2)$ different edge values).

Unnecessary Exercise: Add edges in increasing order until there is a matching. A careful approach can process each edge in $O(N)$ time.

UVa 10615 - Rooks

UVa 10615 - Rooks

Again, try seeing a bipartite graph?

UVa 10615 - Rooks

Again, try seeing a bipartite graph?

Does the desired solution relate to any topic we discussed?

Let $G = (L \cup R; E)$ be a bipartite graph where:

- $L \equiv$ rows
- $R \equiv$ columns
- $E \equiv$ asterisks

UVa 10615 - Rooks

Again, try seeing a bipartite graph?

Does the desired solution relate to any topic we discussed?

Let $G = (L \cup R; E)$ be a bipartite graph where:

- $L \equiv$ rows
- $R \equiv$ columns
- $E \equiv$ asterisks

Compute a minimum edge colouring in $O(N^3)$ time.

UVa 10615 - Rooks

Again, try seeing a bipartite graph?

Does the desired solution relate to any topic we discussed?

Let $G = (L \cup R; E)$ be a bipartite graph where:

- $L \equiv$ rows
- $R \equiv$ columns
- $E \equiv$ asterisks

Compute a minimum edge colouring in $O(N^3)$ time.

See also UVa 12668 for a related problem.

UVa 11368 - Nested Dolls

UVa 11368 - Nested Dolls

This time, think partially ordered set.

UVa 11368 - Nested Dolls

This time, think partially ordered set.

For dolls with indices i, j say $i \prec j$ if doll i fits in doll j.

UVa 11368 - Nested Dolls

This time, think partially ordered set.

For dolls with indices i, j say $i \prec j$ if doll i fits in doll j.

Sequence of dolls can be nested iff they form a chain in this poset.
By Dilworths, we just find the maximum antichain size.

UVa 11368 - Nested Dolls

This time, think partially ordered set.

For dolls with indices i, j say $i \prec j$ if doll i fits in doll j.

Sequence of dolls can be nested iff they form a chain in this poset.
By Dilworths, we just find the maximum antichain size.

But the input is too large for an $O(m^3)$ algorithm!

UVa 11368 - Nested Dolls

This time, think partially ordered set.

For dolls with indices i, j say $i \prec j$ if doll i fits in doll j.

Sequence of dolls can be nested iff they form a chain in this poset. By Dilworths, we just find the maximum antichain size.

But the input is too large for an $O(m^3)$ algorithm!

If we sort the dolls in increasing order of width and break ties by descending in height, the heights of an antichain are a nonincreasing sequence.

UVa 11368 - Nested Dolls

This time, think partially ordered set.

For dolls with indices i, j say $i \prec j$ if doll i fits in doll j.

Sequence of dolls can be nested iff they form a chain in this poset. By Dilworths, we just find the maximum antichain size.

But the input is too large for an $O(m^3)$ algorithm!

If we sort the dolls in increasing order of width and break ties by descending in height, the heights of an antichain are a nonincreasing sequence.

Solution: Compute the longest nonincreasing sequence of heights in this sequence in $O(m \log m)$ time!