We will now see algorithms for solving the *matching problem*. Most of you already seen it but this is the sample problem we will use throughout this course to demonstrate techniques used.

Problem 1 (Maximum matching)**.** Given a graph G, we want to find a *matching*, that is a set of edges with no common endpoint, of maximum size.

First let's see a combinatorial algorithm and then we will describe this problem using linear constraints and analyse its polytope.

1 A combinatorial algorithm for maximum matching

1.1 A combinatorial algorithm for the bipartite case

Recall the definition of an augmenting path

Definition 1.1. *A path P is* alternating *with respect to a matching M if edges of P alternative between being in M and not in M.*

Definition 1.2. *An alternating path P is* augmenting *with respect to a matching M if its endpoints are not incident to M.*

An algorithm for a bipartite graph G:

- Start with an empty matching M.
- Repeatedly find an augmenting path P and swap on it (i.e., $P = M \Delta E(P)$).
- When no augmenting path exist, output M.

By Hall's theorem, this algorithm produces a certificate of maximality: a stable set $S \subseteq V(G)$ with $|N(S)| < |S|$.

For general matching, there's isn't always a certificate.

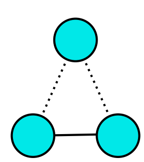

Figure 1: No Hall certificate

But the augmenting path characterization still holds: a matching is maximum if and only if there are no augmenting path (with respect to it). This fact will not be proved here.

1.2 The blossom algorithm

However, an augmenting path is also harder to find in the general case. Sometimes the algorithm may also find a blossom.

1.3 Flowers

A flower in a graph is an odd cycle alternating between matched an unmatched edges (*blossom*) together with an alternating path (*stem*) from an unmatched vertex of the graph to the vertex of this cycle incident two unmatched edges.

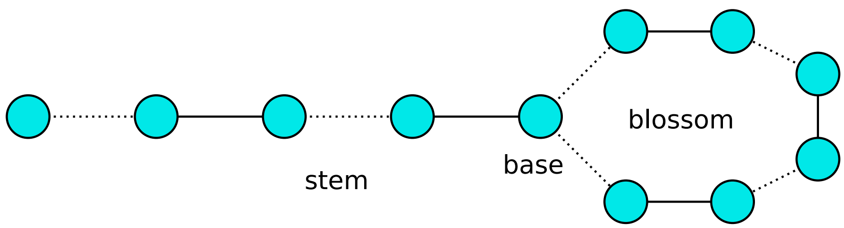

Figure 2: A flower

1.4 Contraction

In case, we find a blossom, the algorithm will contract it (we will explain why later).

Definition 1.3 (Contraction)**.** *For an edge uv of G, G contract uv (written G/uv) is the graph obtained from G by deleting both u and v, and adding a new vertex adjacent to all vertices u and v are adjacent to the new vertex.*

Similarly, for a set S of vertices of G, G contract S (written G/S) is the graph obtained from G by deleting S and adding a new vertex s adjacent to all neighbours of vertices in S.

In the usual definition, the subgraph induced by S is required to be connected (normally, we can only contract a single edge at a time).

1.5 Main algorithm

The main steps of the general algorithm (called the "blossom algorithm") can now be stated.

- Start with an empty matching.
- Repeat
  - Search for an augmenting path or blossom.
  - If a blossom is found, contract it.

  - If an augmenting path is found, swap on it. (Actually uncontract all contracted blossoms and find a corresponding augmenting path where all blossoms are uncontracted and swap on that path.)
  - If neither exist, claim the matching is maximum and return it.

In a bipartite graph, there's no odd cyles so this algorithm would simplify.

The correctness of this algorithm depends on the following lemma which we will prove later.

Lemma 1.4. *For a blossom C with a trivial stem, there is an augmenting path for M in G if and only if there is an augmenting path for $M - C$ in G/C.*

First let's describe how we can find an augmenting path or blossom.

1.6 Finding an augmenting path or blossom

We will mark vertices according to whether they are accessible from an unmatched vertex (a vertex not incident to M) by an odd length path and from an unmatched vertex by an even length path. (A vertex that is both odd and even implies a blossom in the union of these two paths.)

All vertices are unlabelled at the beginning. Technically, we keep track of 3 arrays all indexed by vertices:

- an array of markings which takes value in $\{unmarked, odd, even\}$,
- an array *root* of unmatched "roots" which takes value a vertex, and
- an array *pred* of predecessors on the path to the root, which takes value a vertex or undefined.

This marking algorithm behaves like DFS or BFS in that each vertex keep parent pointers *pred* along the edge used to first visits that vertex (and mark it). When *pred* is set, the value of *root* is copied to the newly visited vertex.

We will not mention these updates explicitly below, all the updates happen when a vertex is marked.

- Mark all unmatched vertex u as having an even path with $root(u) = u$ and $pred(u) = \emptyset$.
- Put all edges incident to an unmatched vertex in a list of edges to examine.
- For each edge uv to examine with u even,
  - If v is unmarked and matched,
    * Mark v as odd.
    * Mark the vertex w that v is matched to as even.

    * Add all edges incident to w except vw to edges to examine.
  - If v is even (including if v is unmatched),
    * If $root(u) \neq root(v)$ then their paths to the root do not intersect (because of how *pred* is defined), we have found an augmenting path from $root(u)$ to $root(v)$ passing through uv. Return it.
    * If $root(u) = root(v)$, we find a blossom by following *pred* from u and v until their paths coincide. Switch M along the path from u to its root to turn this blossom into one with a trivial stem (namely, u). Return this blossom.
  - Otherwise (v is odd), do nothing.
- Return "There is no augmenting path for M"

Instead of marking vertices, we could directly build (directed) paths between even vertices starting from unmarked vertices. (Then vertices are always either *unmarked* or *even*.)

1.7 Contracting a blossom

We now prove Lemma 1.4 which allows us to contract blossoms.

Proof: Say C is contracted to a vertex c. Suppose there is an augmenting path P in G/C. We will construct an augmenting path in G from P.

If P does not contain C then P is already an augmenting path in G.

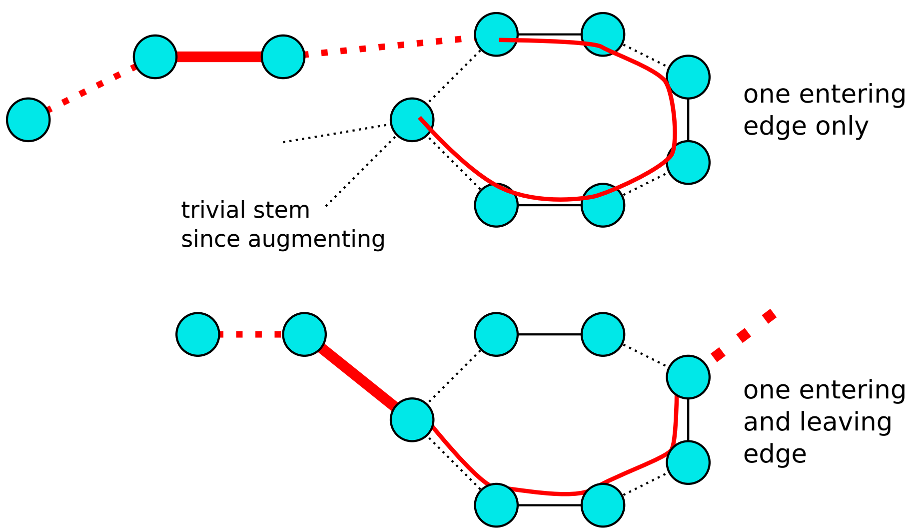

Figure 3: Two cases

Otherwise since C is a blossom with a trivial stem, c is unmatched in G/C. So c is an endpoint of P and no vertex of C is matched to a vertex outside C (or that matching edge would still be in $M-C$). So extend P by an alternating path inside C from the corresponding endpoint of P to the base of C.

Conversely, suppose there is an augmenting path P in G. Then contracting P gives an augmenting path for $M - C$ in G/C unless P enters and leaves C through non-matching edges (there is only one base so two matching edges is impossible).

In this last case, just cut the path short by going to the base (which has a trivial stem). □

1.8 Running time

We have at most $|V(G)|$ contractions between finding augmenting paths and at most $|V(G)|/2$ augmenting paths found. So we only need to know how long one iteration of the marking algorithm takes (and multiply by $|V(G)|^2$).

There are at most $O(|E(G)|)$ iterations. Each iteration where we do not return anything takes constant time. In the iteration we return, we return either a blossom or path. Finding and contracting a blossom takes $O(|V(G)|)$ as this is bound on the size of the two paths we need to intersect. Finding an augmenting path and swapping also takes $O(|V(G)|)$.

So the total time is $O(|V(G)|^2(|V(G)| + |E(G)|)$.

1.9 Speeding things up with union-find

Instead of contracting directly, we could use a union-find structure for the set of vertices.