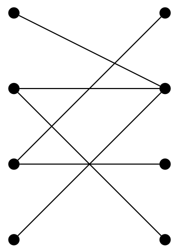
Maximum Matchings in Graphs (Edmonds' Blossom Algorithm)

Source: Chapter 10.1, 10.5

Let $G = (V, E)$ be a graph. A **matching** M a subset of E, such that graph (V, M) has maximum degree one.

Problem: For a graph G, find a matching M of maximum size.

1: Find a maximum matching in the following graph.



2: Formulate the maximum matching problem using integer programming (IP).

Solution: To every edge $e \in E$ assign variable $x_e \in \{0,1\}$. For every vertex, sum of edges is ≤ 1. Maximize the sum over all edges.

$$(IP)\begin{cases}\text{maximize} & \sum_{e\in E} x_e\\ \text{subject to} & A\mathbf{x} \leq \mathbf{1}\\ & \mathbf{x} \in \{0,1\}^{|E|},\end{cases}$$

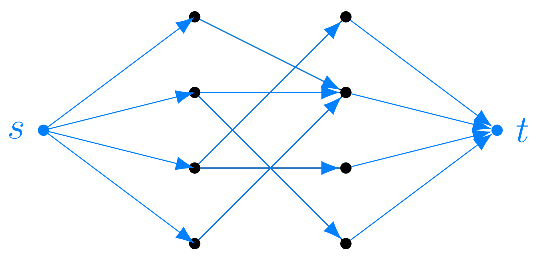
where A is the incidence matrix of G.

3: Is there a condition on G that guarantees that a relaxation of (IP) to a linear program always has an integral optimal solution? Notice that the matrix A is an incidence matrix of G.

Solution: If G is a bipartite graph, then the incidence matrix is totally unimodular. That guarantees the relaxation to LP will give an integral solution. See the worksheet on unimodularity.

4: Show that finding a maximum matching in a bipartite graph G can be done using maximum flow algorithm. Use the graph below to do the reduction from maximum matching to maximum flow. Is there a *certificate* that a matching is maximum?

Solution: Let $G = (V, E)$, where $V = V_1 \cup V_2$. Add s adjacent to all vertices in V_1 and t adjacent to all vertices in V_2. Orient all edges from s to t and add capacities 1 everywhere.



Notice that the maximality can be shown using minimum cut.

Definition: Let M be a matching in a graph $G = (V, E)$. A vertex $v \in V$ is **covered** if exists $e \in M$ such that $v \in e$, otherwise v is **exposed**. A matching is called a **perfect matching** if all vertices are covered.

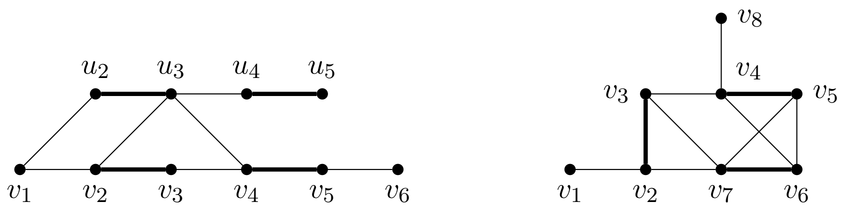
Perfect matchings are covered in the book but we are skipping them.

Notice: maximizing $|M|$ is the same as minimizing the number of exposed vertices.

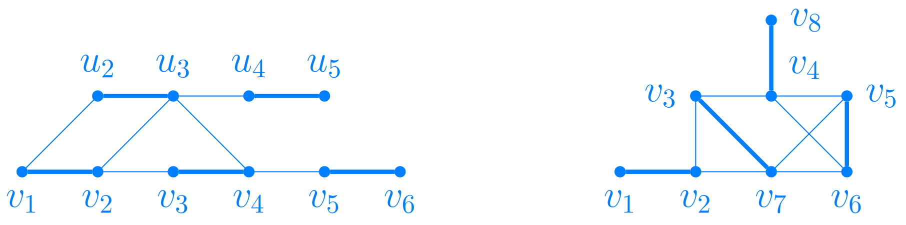
Inspiration by flow: use *augmenting paths.*

A path P is M**-alternating** if $E(P) \setminus M$ is a matching. An M**-alternating** path P is M**-augmenting** if P has positive length and both its endpoints are exposed in M. Augmented $M' = M \Delta E(P)$.

5: Assume there is a matching M (thick lines). Find M-augmenting path(s) and augment M.



Solution: An M-augmenting path is a path P with endpoints exposed, inner points covered and the edges of the matching are alternating on P. On the left for example $v_1, v_2, v_3, v_4, v_5, v_6$. On the right it is $v_1, v_2, v_3, v_7, v_6, v_5, v_4, v_8$.



Theorem 10.7 Let G be a graph with a matching M. Then M is maximum if and only if there is no M-augmenting path.

6: Prove Theorem 10.7

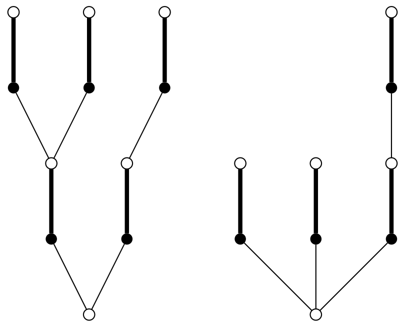
Solution: $\Rightarrow$: Augmenting path increases the size of the matching
$\Leftarrow$. Consider M' being a matching with more edges than M. Take the symmetric difference of M and M'. It is a graph of maximum degree two, which gives set of even cycles and paths. Implies one of the paths must be M-augmenting.

7: Can we use augmenting walks instead of paths? It particular, examine walk
$v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_5, v_4, v_8$ in the graph on the right-hand side.

Solution: We cannot augment on it. Both v_4 and v_5 would have two matching edges.

Idea for an algorithm: Keep finding M-alternating paths and augment on them. This increases size of M and when none exists, M is maximum.

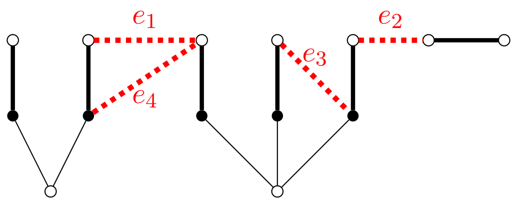
Main issue: Find an M-alternating path. We explore ways how to find it. Start from exposed vertices as roots and build **alternating forest** F. Alternate non-matching edges and matching edges. This gives layers of matching edges and non-matching edges.



Notice that not all edges of G are present in the forest F! Call vertex **outer** vertex if it is in even distance from the root (white ones).

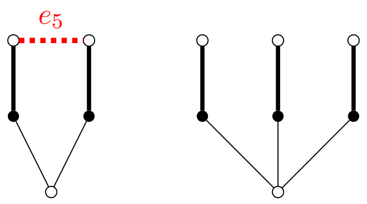
Assume building of the alternating forest by picking an edge adjacent to outer vertex and trying to extend the forest edge by edge.

8: What happens when we try to add any of the dotted edges e_1, e_2, e_3, or e_4?

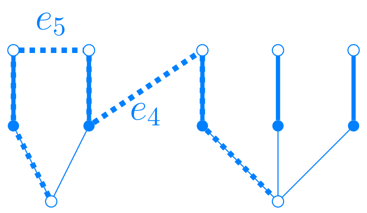


Solution: e_1 gives an augmenting path, e_2 allows the forest to grow, e_3, e_4 can be dropped - if there is an augmenting path using them, there is one avoiding them.

9: What happens with e_5? Hint: What if we have both e_5 and e_4?

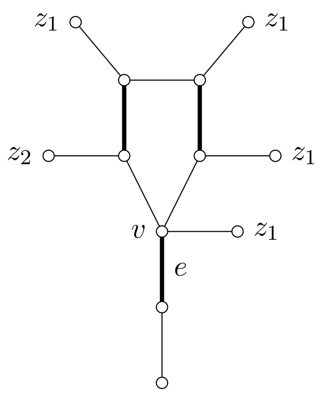


Solution: Edge e_5 is important. Consider edge e_4 from the previous. Show the augmenting paths is using e_5 and it cannot be avoided.



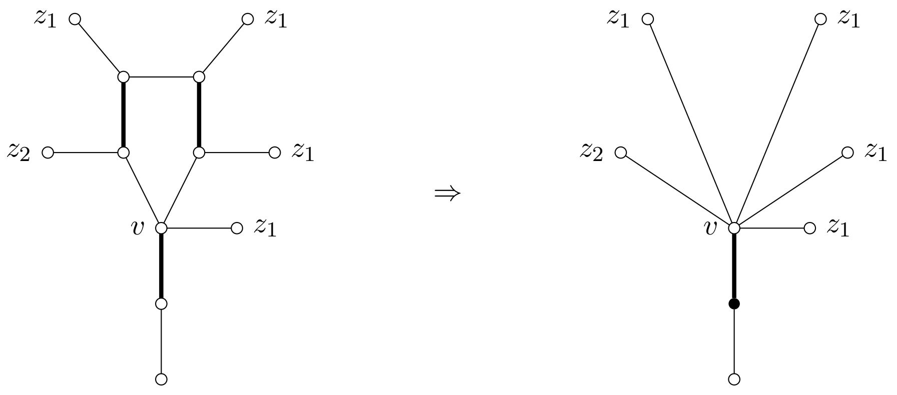
Blossom is an odd cycle on k vertices containing $\frac{k-1}{2}$ edges from M.

10: Let C be a blossom, where v is not matched with other vertex in C. (What is C in the figure below?) Show that alternating path entering C using a matching edge e containing v can leave C using an unmatched edge from **any** vertex of C.



Solution: Just try the cases.

Since blossom acts like a vertex that can be matched to anything, we contract the blossom.

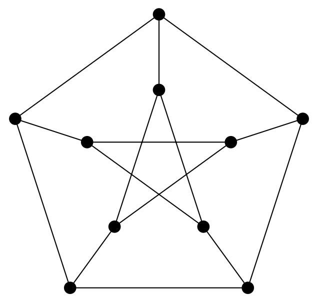


Edmonds Blossom Algorithm (sketch)
Runs on a graph $G = (V, E)$.

```
1. $M = \emptyset$
2. $F$ = uncovered vertices, all edges unexplored
3. while exists an unexplored edge $e \in E$ adjacent to an outer vertex of $F$
4.         if $e$ connects two outer vertices from different components of $F$,
5.             get an $M$-alternating path and augment $M$, go to 2.
6.         if $e$ connects two outer vertices from the same components of $F$:
7.             find a blossom and contract it, unexplore edges going out from no-outer vertices
               of the blossom.
8.         if $e$ connects and outer vertex to a vertex $x \in V \setminus V(F)$,
9.             add $x$ and its neighbor in $M$ to $F$.
```

If the algorithm is implemented carefully, it runs in $O(\sqrt{n}m)$, where $n = |V|$ and $m = |E|$.

11: Try to run the algorithm on the Petersen's graph.



Solution: A good choice of order of explored edges can show all features of the algorithm. But I don't know the right order :-). Also, good if the graph had numbered vertices.