1 Matching

- *Algorithm:* maximum matching (algo. 1)
- *Input:* A graph with a set of Edges E and a set of Vertices V
- *Complexity:* $\mathcal{O}(|V|^2 \cdot |E|)$
- *Data structure compatibility:* Graph (Bipartite graph supported)
- *Common applications:* matching skeleton, chemistry analysis, marriage problem,

Matching

Given a graph G with a set of vertices V and a set of edges E, find the maximum number of edges satisfying that no two edges share common vertices.

Description

1. Problem Clarification: Suppose we have a connected undirected graph, and our purpose is to find the maximum number of edges in the graph that any two do not share a same vertex.
2. Algorithm Clarification: To solve this problem, we will use an algorithm called **Blossom algorithm**, which was developed by *Jack Edmond*. The main idea of this algorithm is to improve the previous or initial matching along newly-defined 'augmenting paths' in the graph.
3. Augmenting Path: Before introducing augmenting path, there are several definitions worthy of introducing.
   (a) **Exposed**: a vertex is called exposed if no edge incident with it.
   (b) **Alternating Path**: a path that the edge is alternately in the match or not in the match. *E.g. The path starts with an edge in the match, and the next edge following is not in the match, and the next following edge is in the match, so on and so forth.* It could be taken as a locally optimized match satisfying the match condition.

   Then it comes to the definition of **Augmenting Path**: An alternating path starting and ending with two Exposed vertices.

   To augment along an augmenting path, simply replace *all the edges in the match* with *all the edges not in the edge*.

   Now the hard point here is to efficiently find the augmenting path according to definition.

Algorithm 1: find max matching

Input : a Graph G, initial matching M
Output: maximum matching

```
1 Function Find_max_matching(graph G, matching M):
2     while exists an augmenting path P do
3         M ← Augment M on P;
4         return Find_max_matching(G, M);
5     end while
6     return M;
7 end
```

Finding augmenting path

To find augmenting path, there are two operations, which is called blossoms and contractions.

1. **Blossoms**: a blossom is defined as a cycle with odd edges $(2k + 1)$ inside, and exactly k edges are in the match.

2. **Contractions**: a contraction means to take the whole blossoms as a simple node.

The algorithm of finding an augmenting path is then given as:

Algorithm 2: find augmenting path

Input : a Graph G
Output: augmenting path in G

```
1  Function Find_max_matching_including_augPath(graph G):
2      F ← all nodes in graph;
3      while F is not empty do
4          pick an e from F;
5          Add e to Q;
6          T ← empty set;
7          Add e to T;
8          while Q is not empty do
9              v ← pop an element from Q;
10             for All neighbors of v do
11                 Denote the neighbor as w;
12                 if w is not in T and w is matched then
13                     Add w to T;
14                     Add u(the other node connected to w) to T;
15                     push u to Q;
16                 else if w is in T and the cycle is even length then
17                     Do nothing;
18                 else if w is in T and the cycle is odd length then
19                     Contract;
20                 else
21                     push w and v to the match;
22                     All the nodes that contracted be expanded and construct augmenting path;
23                     invert augmenting path;
24                 end if
25             end for
26         end while
27     end while
28     return The final match
29 end
```

Something more about time complexity: The time complexity for the algorithm is $\mathcal{O}(|V| \cdot |E|^2)$. which is easy to derive by checking the iterations in the loop number in the algorithm.

References

- TUM, Edmond's Blossom Algorithm https://www-m9.ma.tum.de/graph-algorithms/matchings-blossom-algorithm/index_en.html
- Lszl Lovsz; M. D. Plummer (1986), Matching Theory, North-Holland