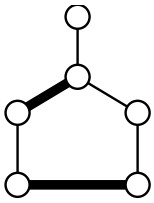
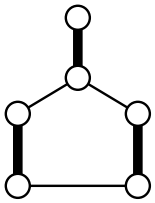
Matchings in graphs

Let $G = (V, E)$ be an undirected graph. A *matching* of G is a subset $M \subseteq E$ of the edges such that no two edges of M share an adjacent node. A matching M is called *maximal*, if G doesn't have a larger matching $M' \supset M$. A matching M is called *maximum matching* if G doesn't have a matching M' such that $|M'| > |M|$. We are interested in an algorithm which, given a graph, efficiently computes a maximum matching.



(a) Maximal matching



(b) Maximum matching

Matching problems occur in practice mostly in form of assignment problems. An example could be the assignment of professors to courses, if each professor wants to be in charge of at most one course: the nodes of the graph represent the professors and the courses. An edge between a professor and a course indicates that the professor is able to give the lecture of that course. A maximum matching corresponds to an assignment of professors to courses such that as much courses as possible can be given.

For a graph $G = (V, E)$ and a matching $M \subseteq E$ we call the edges of M *matched*, while edges of $E \setminus M$ are called *unmatched*. A node is called *matched* if one of its incident edges is matched, otherwise it is called *free*. A helpful concept is that of alternating and augmenting paths w.r.t. a matching M:

(i) A simple path v_0, v_1, ..., v_r is called *alternating*, if the edges (v_{i-1}, v_i) alternate between being part of M and being part of $E \setminus M$.

(ii) An alternating path is called *augmenting*, if the first and the last edge of the path are unmatched such that the path can't be extended (i.e., the first and the last node of the path are free).

Keep in mind that an augmenting path can also consist of a single unmatched edge connecting two free nodes.

It is easy to see that a matching M can be enlarged to a matching M' by using an augmenting path: if the matched edges of the path are removed from M and the unmatched edges are added to M, we obtain a matching whose cardinality is larger by exactly one compared to M. This process is called *inverting* an augmenting path. The matching (b) illustrated above can be obtained from the matching (a) by inverting such an augmenting path.

Satz 1 *A matching M of $G = (V, E)$ is a maximum matching if and only if there is no augmenting path.*

Beweis: The claim of this proposition is equivalent to: A matching M is not a maximum matching if and only if there is an augmenting path. We prove this claim. The direction $\Leftarrow$ obviously is correct. It remains to prove the direction $\Rightarrow$.

Let M be a matching in G which isn't a maximum matching. Let M' be a maximum matching in G. Consider the graph $G' = (V, M \oplus M')$ where $M \oplus M' = (M \cup M') \setminus (M \cap M')$ denotes the symmetric difference of M and M'. It is easy to see that each node of G' has degree at most two. Therefore G' consists of a collection of simple paths and cycles which are alternating w.r.t. M. M' can be obtained from M by inverting all these paths and cycles. Since inverting a cycle or paths that are not augmenting can't increase the matching, there must be $|M'| - |M| \geq 1$ augmenting paths (w.r.t. M). □

From this proposition even follows that G contains exactly $|M'| - |M|$ node disjoint augmenting paths where M' is a maximum matching. Since $|M|$ matched edges can be distributed over these $|M'| - |M|$ augmenting paths we also find that there must be an augmenting path of length at most $2 \cdot \lfloor |M|/(|M'| - |M|) \rfloor + 1$.

This proposition also shows that we can find a maximum matching by starting with an arbitrary matching M, e.g. $M = \emptyset$, and iteratively search for and invert augmenting paths until no augmenting path exists anymore. We only have to clarify how exactly we should search for augmenting paths.

It turns out to be pretty beneficial to not simply search for an arbitrary augmenting path but for a *maximal set of shortest* augmenting paths. Let M be the current matching and let ℓ be the length (number of edges) of a shortest augmenting path w.r.t. M. Then we search for a set $p_1, p_2, \ldots, p_r$ of pairwise node disjoint augmenting paths of length ℓ such there are no more such node disjoint paths exist, and invert all these paths $p_1, p_2, \ldots, p_r$. The next section describes this procedure for bipartite graphs in more detail.

1 Matchings in bipartite graphs

A graph $G = (V, E)$ is *bipartite* if its set of nodes V can be divided into two disjoint subsets V_1 and V_2 such that each edge is adjacent to one node of V_1 and one node of V_2. Many graphs which model assignment problems in practice are indeed bipartite, for instance the assignment problem of professors and courses mentioned above.

In bipartite graphs the search for a maximal set of node disjoint augmenting paths using a *simultaneous breadth first search* followed by a depth first search can be efficiently realized in linear time $O(|V|+|E|)$ (algorithm of Hopcroft and Karp, 1973). We describe this algorithm in the following:

- set $M := \emptyset$
- **while** (true) **do**
  simultaneous breadth first search;
  if (there is an augmenting path)
  then depth first search for node disjoint shortest augmenting paths;
  else break;
- output the maximum matching M

1.1 Simultaneous breadth first search

Given a matching M the simultaneous BFS is used in order to decide if there are augmenting paths at all, as well as assigning level values to the nodes which are used by the following DFS to find a maximal set of shortest node disjoint augmenting paths.

Each path in a bipartite graph G alternately visits nodes of V_1 and V_2. Since the length of each augmenting path is odd, it starts at a free node in V_1 and ends at a free node in V_2. In doing so they use an unmatched edge when going from V_1 to V_2 and a matched edge when going from V_2 to V_1. We assign to each node v a value level$[v]$ which is initialized by -1 at the beginning of each simultaneous BFS, and which is supposed to represent the length of a shortest augmenting path from some free node of V_1 to v at the end of the simultaneous BFS.

To calculate the length of a shortest augmenting path, we start the BFS at all free nodes of V_1 at the same time: hence, *simultaneous* BFS. To do so, we don't just initialize the queue with a single node at the beginning but add all free nodes of V_1 to the queue. Furthermore, we set level$[v] = 0$ for each of these nodes v.

The BFS is divided into *phases* which alternate between being of type 1 or type 2:

Type 1: at the beginning of the phase, all nodes in the queue are a subset of V_1; during the phase each of these nodes v are popped from the queue, and all **unmatched** incident edges $e = \{v, w\}$ are considered: if w wasn't visited before, then we set level$[w]$ = level$[v] + 1$ and append w at the end of the queue; at the end of the phase the queue only contains nodes of V_2.

Type 2: at the beginning of the phase, all nodes in the queue are a subset of V_2; during the phase each of these nodes v are popped from the queue, and all **matched** incident edges $e = \{v, w\}$ are considered: if w wasn't visited before, then we set level$[w]$ = level$[v] + 1$ and append w at the end of the queue; at the end of the phase the queue only contains nodes of V_1.

The BFS terminates, when the queue contains a free node w after a phase of type 1 (then there is an augmenting path of length level$[w]$ which ends at w, and the DFS is started), or when the queue is empty (then no augmenting path exists and the current matching is a maximum matching).

1.2 Depth first search for augmenting paths

If, during the simultaneous BFS, a free node of V_2 was labelled with ℓ, then we now that there is at least one shortest augmenting path of length ℓ, and we want to calculate a maximal set of shortest augmenting paths using DFS. To be more precise, we won't use just a single DFS but, for each free node v of V_1, one corresponding DFS to find an augmenting path of length ℓ starting at v (if possible). During these DFSs we only consider edges which satisfy the following properties (all other edges are ignored):

- If the current node u is in V_1, then we consider the edges $e = \{u, w\}$ where $e \in E \setminus M$ and level$[w]$ = level$[u] + 1$.
- If the current node u is in V_2, then we consider the edges $e = \{u, w\}$ where $e \in M$ and level$[w]$ = level$[u] + 1$.

If such a DFS starting at a free node $v \in V_1$ reaches a free node $w \in V_2$, then the corresponding path from v to w is a shortest augmenting path. This path is immediately inverted (edges which are in M are removed from M, and edges which are not in M are added to M). Furthermore, we lock all nodes u of this path by setting level$[u] = -1$ to prevent following DFSs from using these already used nodes. Then the next DFS is started at the next free node of V_1. If a DFS reaches a dead-end, that is, if the current DFS wasn't able to find an augmenting path and the current node u doesn't have an edge which satisfies one of the properties mentioned above, then we set level$[u] = -1$ and we continue at the parent node of u. The level values of the nodes are only modified if an augmenting path is found or if a dead-end is reached: if that happens, setting the level values of the nodes in question to -1 lets further DFSs ignore these nodes.

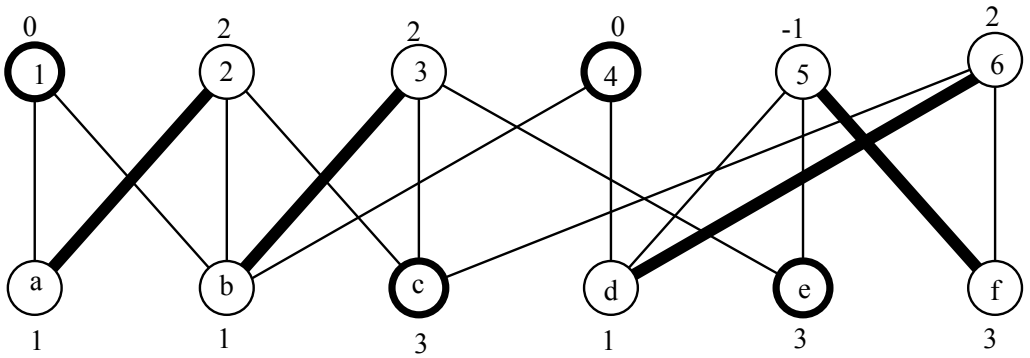
1.3 Analysis

An execution of the body of the while-loop is called *iteration*. As Hopcroft and Karp showed, we only have to do $O(\sqrt{|V|})$ iterations. (Proof idea: Each iteration increases the length of the shortest augmenting path by at least 1. After at most $\sqrt{|V|}$ phases each augmenting path has length at least $\sqrt{|V|}$. Therefore, each augmenting path of $M \oplus M'$ has length at least $\sqrt{|V|}$, where M is the current matching and M' is an arbitrary maximum matching. This implies $|M'| - |M| \leq |V|/\sqrt{|V|} = \sqrt{|V|}$ since these paths are node disjoint. Therefore, at most $\sqrt{|V|}$ augmentations can occur from this point on, hence, the remaining number of phases is at most $\sqrt{|V|}$.)

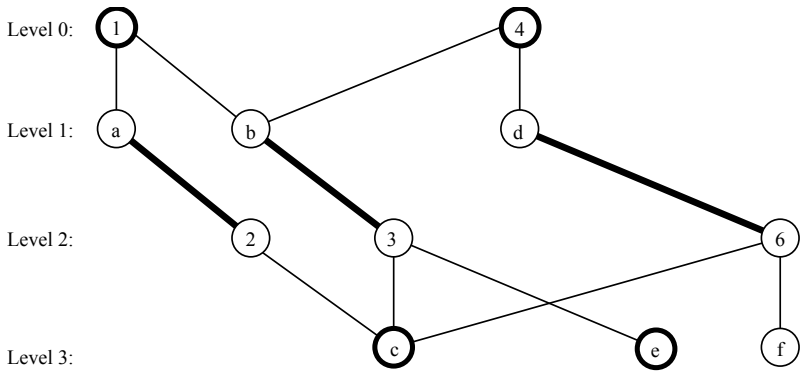
Each iteration consists of one simultaneous BFS and, if the current matching isn't a maximum matching, of multiple DFSs. The BFS alone as well as all DFSs together have running time in $O(|V| + |E|)$, respectively. Therefore, the total running time of the algorithm is $O(\sqrt{|V|}|E|)$ if we only have inputs graphs which don't contain isolated nodes. If all inputs are possible, we must remove isolated nodes first, yielding a running time of $O(|V| + \sqrt{|V|}|E|)$.

1.4 Example for the execution of one iteration

The bipartite graph $G = (V_1 \cup V_2, E)$ with $V_1 = \{1, 2, 3, 4, 5, 6\}$ and $V_2 = \{a, b, c, d, e, f\}$ as well as the current matching M are given as follows (matched edges are bold, free nodes are bold-framed):



The numbers outside the nodes are the respective level values which are obtained by the simultaneous BFS. After the third phase of the simultaneous BFS the nodes c, e, and f are in the queue. Since two of them (c and e) are free, the simultaneous BFS terminates, and for each free node of V_1, i.e. 1 and 4, a DFS is started. Hereby only the following edges are considered:



For example the paths 1, a, 2, c, as well as 4, b, 3, e can be found which constitute a maximal set of node disjoint shortest augmenting paths. (Also the path 4, b, 3, c alone would be such a set.) By inverting these two paths we obtain the following matching which already is a maximum matching.