Since all heights are non-negative, we have $\Phi_f \geqslant 0$ at all times. Initially, all non-*st* vertices have height 0, so $\Phi_f = 0$. Finally, there are no active vertices, so $\Phi_f = 0$ again. There are at most $n-2$ active vertices and $h(u) \leqslant 2n-1$, so the maximum value of Φ_f can be $\leqslant (n-2)(2n-1) \leqslant 2n^2$. But its increments and decrements may be interleaved. So we need to count how different operations affect its value.

A relabeling of u increases Φ_f by exactly the same amount by which $h(u)$ increases. Since $n-2$ vertices can experience height gains from 0 to at most $2n-1$, the total increase in $\Phi(n)$ caused by all relabeling operations must be $\leqslant 2n^2$.

Now, consider a saturating push along the edge (u,v) in G_f. If v is s or t, $h(v)$ does not change at all. Otherwise v becomes active (it may have been so earlier). So Φ_f can increase by $h(v) \leqslant 2n-1$. After the saturating push, u may become inactive and leads to a reduction in the potential Φ_f. In all cases, the maximum possible increase during a saturating push is bounded from above by $2n-1$. We have already seen that the maximum number of saturating pushes is $\leqslant 2mn$, so the total increase in Φ_f caused by all saturating pushes is no larger than $2mn \times (2n-1) \leqslant 4mn^2$.

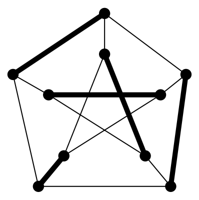
It follows that relabeling steps and saturating pushes increase Φ_f by at most the quantity $2n^2 + 4mn^2 \leqslant 6mn^2$. I now show that each non-saturating push decreases the potential by at least one, implying that at most $6mn^2$ non-saturating pushes are possible.

Consider a non-saturating push along the edge $(u,v) \in E_f$. Since the push is non-saturating, we must have $\delta_f(u,v) = e_f(u)$, that is, after the push, u no longer remains active. If v is s or t, it never contributes to the potential, and Φ_f decreases by $h(u) \geqslant 1$ (we must have $h(u) = h(v)+1$ and $h(v) \geqslant 0$). If v was active before the push, the decrease in Φ_f is again by $h(u) \geqslant 1$. Finally, if $v \neq s,t$ and v was not active before the push, the decrease in Φ_f by the push is $h(u) - h(v) = 1$.

The total effort associated with all non-saturating pushes is therefore $\mathrm{O}(mn^2)$. Consequently, the generic push-relabel algorithm can be implemented to run in $\mathrm{O}(n^3 + n^2 m)$ time. We have argued that $m = \Omega(n)$, so the running time of the algorithm is $\mathrm{O}(n^2 m) = \mathrm{O}(|V|^2|E|)$.

21.6 Maximum matching

Let $G = (V,E)$ be an undirected graph. A subset $M \subseteq E$ is called a *matching* in G if every vertex of G belongs to at most one edge in M. Equivalently, M is a matching if and only if no two distinct edges of M share a common endpoint. A vertex $v \in V$ is said to be *matched* by M if v is the endpoint of a (unique) edge of M. An unmatched vertex is also called *free*. M is called a *perfect matching* in G if every vertex $v \in V$ is matched by M. The bold edges in the following figure constitute a perfect matching in the Petersen graph.



The (unweighted) *maximum-matching problem* deals with the computation of a matching M in G containing the maximum possible number of edges. In a more generic setting, we assign weights

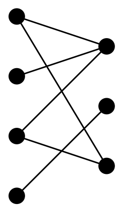
$w(e)$ (usually positive) to each edge $e \in E$. The *weighted maximum-matching problem* deals with the computation of a matching M in G such that the sum $w(M) = \sum_{e \in M} w(e)$ is as large as possible. The unweighted variant of the matching problem is the special case in which $w(e) = 1$ for all $e \in E$.

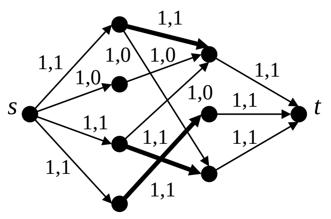
21.6.1 Unweighted bipartite matching

Let $G = (V, E)$ be a bipartite graph with a bipartition of V into V_1 and V_2. We may think of V_1 as a set of men, and V_2 as a set of women. There exists an edge between $v_1 \in V_1$ and $v_2 \in V_2$ if and only if v_1 and v_2 like each other (and can marry). The maximum-matching problem in this case is the determination of the maximum number of pairs that can marry. A prefect matching is possible only if $|V_1| = |V_2|$.

Algorithm based on flow

The bipartite matching problem can be solved by reduction to the maximum-flow problem. First, convert G to a directed graph $G' = (V', E')$ as follows. V' consists of all vertices of G and two new vertices s (the source) and t (the sink). Convert each undirected edge $(v_1, v_2) \in E$ with $v_1 \in V_1$ and $v_2 \in V_2$ to the directed edge $(v_1, v_2) \in E'$. Also add directed edges (s, v_1) and (v_2, t) to E' for all $v_1 \in V_1$ and $v_2 \in V_2$. Finally, take the capacity $c(e') = 1$ for all $e' \in E'$. The following figure illustrates this construction along with a solution of the maximum-flow problem.


Bipartite matching



Network flow

It is easy to see that this construction works. We may apply the Ford–Fulkerson algorithm for finding a maximum flow in G'. We start with the zero flow, and note that the residual capacity of an augmenting path is always an integer in this case. The maximum matching size in G can be no larger than $\min(|V|/2, |E|)$, that is, the number of flow-augmentation step can be at most $\min(|V|/2, |E|)$. Since each augmenting path can be discovered in $O(|V'| + |E'|) = O(|V| + |E|)$ time, it follows that this bipartite-matching algorithm runs in $O(|E||V|)$ time.

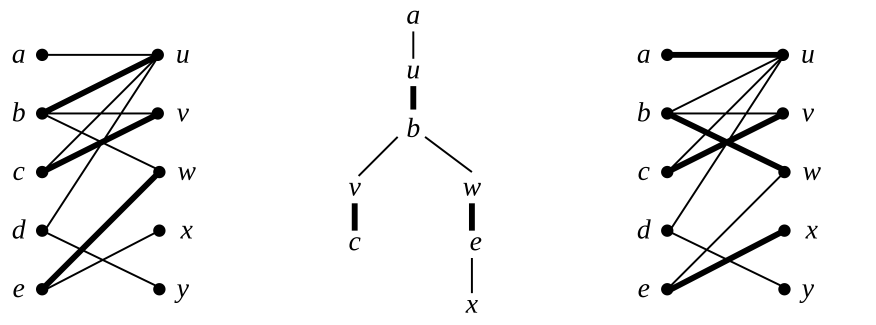
Algorithm based on traversal

Another approach for computing maximum unweighted bipartite matchings is based on traversal. Let M be a matching already available to us. If all the vertices of V_1 are matched, then M is already a maximum matching. So assume that there are free vertices in V_1. Pick any free vertex $v_1 \in V_1$. Start a DFS traversal on G from v_1. Let $v_2 \in V_2$ be a neighbor of v_1. If v_2 is free, add the edge (v_1, v_2) to M to obtain a larger matching. So suppose that v_2 is matched, say, to a vertex $v_3 \in V_1$. Continue the DFS traversal from v_3. Let $v_4 \neq v_2$ be a neighbor of v_3 in V_2. Since v_3 is already matched to v_2, the edge (v_3, v_4) does not belong to M. If v_4 is free, replace $(v_2, v_3) \in M$ by (v_1, v_2) and (v_3, v_4) to get a matching larger than M. Continue this process until the current matching can be augmented or no such possibilities are located before the DFS ends. We then look at another free vertex in V_1, and repeat the same process.

Here, we use a restricted form of DFS. We always use an edge not in M for going from V_1 to V_2, and an edge in M from coming back to V_1 from V_2. For example, if we find v_2 matched to v_3, we take the edge (v_2, v_3) in M. The vertex v_2 may have other neighbors v_3' in V_1, which may or may not be matched. But (v_2, v_3') is not in M, and we do not take the edge (v_2, v_3') in the DFS traversal. In other words, in the DFS path $v_1, v_2, v_3, v_4, \ldots$, all edges (v_{2i-1}, v_{2i}) are not in M, whereas all edges (v_{2i}, v_{2i+1}) are in M. Such a path in G where the edges of $E \setminus M$ and M alternate is called an *alternating path*, and the DFS tree thus produced is called an *alternating tree*.

If we find a DFS path $v_1, v_2, v_3, v_4, \ldots, v_{2k-1}, v_{2k}$ with v_{2k} free, then replacing (v_2, v_3), (v_4, v_5), $\ldots$, (v_{2k-2}, v_{2k-1}) in M by (v_1, v_2), (v_3, v_4), $\ldots$, (v_{2k-1}, v_{2k}) increases the size of M by 1. In this case, we say that the matching is augmented. When there are no futher augmenting possibilities, the algorithm stops.

Figure 86: Unweighted bipartite matching based on traversal



(a) A matching (b) The alternating tree (c) Augmented matching

An augmentation step is illustrated in Figure 86. We take the matching $M = \{(b,u),(c,v),(e,w)\}$ (see Part (a)). We start the restricted DFS traversal from the free vertex a, and proceed along the edge (a,u). The vertex u has three unvisited neighbors b, c, and d. Since u is matched to b, we are not allowed to use the edges (u,c) and (u,d) in the DFS traversal. So we use the edge (u,b). The vertex b has two unvisited neighbors v and w. Let us explore v first. This vertex is matched to c, so we take the edge (v,c). From c, there are no possibilities to continue the DFS, so we backtrack to b. Notice that there is no point tracking back to v, because we have to use the edge (v,c) from v. The second neighbor of b to be explored is w which is matched to e. Finally, we find the unmatched neighbor x of e (see Part (b) of Figure 86). This DFS traversal discovers the alternating path a,u,b,w,e,x which offers an augmentation possibility. The two edges (u,b) and (w,e) in M are replaced by the three edges (a,u), (b,w) and (e,x), giving the augmented matching $\{(a,u),(b,w),(c,v),(e,x)\}$ (Part (c) of Figure 86). The reader is urged to draw a full alternating tree rooted at the free vertex d, corresponding to the augmented matching. From d, go to u first.

The final matching M obtained from this algorithm cannot be augmented. We need to show that this matching is maximum too. Suppose not. Let M^* be a maximum matching with $|M^*| > |M|$. Consider the graph $H = (V, M \oplus M^*)$, where $M \oplus M^* = (M \setminus M^*) \cup (M^* \setminus M) = (M \setminus (M \cap M^*)) \cup (M^* \setminus (M \cap M^*))$. It is easy to see that H consists of isolated vertices, and vertex-disjoint paths and cycles which are alternating with respect to both M and M^*. Such a cycle or an even-length path must contain an equal number of edges from M and M^*. Since $|M^*| > |M|$, H contains more edges from M^* than from M. Therefore there must exist an odd-length path in H with the first and the last edges from M^*. Such a path augments M, a contradiction.

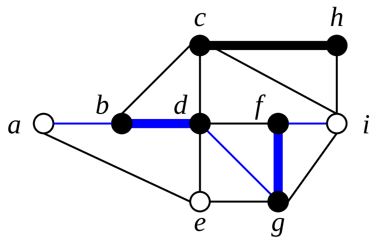
Let us now deduce the running time of this algorithm. We start with the empty matching, and keep on augmenting it until there are no further augmentation possibilities. The maximum number of augmenting steps is $\min(|V_1|, |V_2|) = O(|V|)$. Moreover, an augmenting path is discovered by a multi-DFS traversal which runs in $O(|V| + |E|)$ time. We may remove isolated vertices from G (these vertices do not take part in any matching), and assume that each vertex in G has at least one neighbor, which in turn implies that $|V| = O(|E|)$. So the running time of this algorithm is $O(|V||E|)$—the same as the flow-based algorithm.

Hopcroft and Karp propose a refinement of this algorithm. During each traversal, they find a maximal set of shortest augmenting paths using a combination of BFS and DFS. The matching is then augmented along all these paths. This results in an improved running time of $O(\sqrt{|V|}|E|)$.

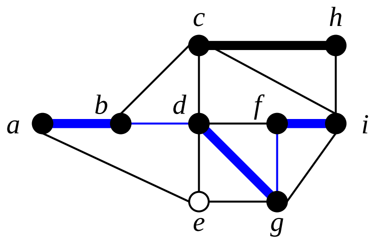
21.6.2 Unweighted matching in a general graph

For a general undirected graph, finding a maximum matching involves much more work. Let M be a matching of $G = (V, E)$. Let $u, v \in V$ be two distinct vertices. A u, v-path p on which edges alternate between M and $E \setminus M$ is called an *alternating path*. If u and v are both free (that is, not matched by M), then p is called an *augmenting path*. An *alternating cycle* can be defined likewise.

The first and last edges in an augmenting path must be outside M, that is, the length of any augmenting path (number of edges) must be odd. If we can detect an augmenting path p in G, we can replace M by the symmetric difference $M \oplus p = (M \setminus p) \cup (p \setminus M)$. That is, all edges in M outside p are kept, whereas an edge on p is kept if and only if it was not in M. The augmented matching contains one extra edge, and is therefore an iterative improvement over the previous matching. The following figure demonstrates an augmentation step. The bold edges are the edges in the matchings, and the matched vertices are shown as solid circles.



An augmenting path *a,b,d,g,f,i*



Augmented matching

Two questions need to be answered for completing the description of the algorithm. First, how to find an augmenting path (if it exists). And second, how to ensure that augmenting paths can always be found for any non-maximum matching. The following result addresses the second question.