CMSC 651 Advanced Algorithms	Lecture 6
Lecturer: Samir Khuller	Th. Feb 14, 2002

Notes by Samir Khuller.

4 Hopcroft-Karp Matching Algorithm

The original paper is by Hopcroft and Karp [SIAM J. on Computing, 1973].

We present the Hopcroft-Karp matching algorithm that finds a maximum matching in bipartite graphs.

The main idea behind the Hopcroft-Karp algorithm is to augment along a *set* of shortest augmenting paths *simultaneously*. (In particular, if the shortest augmenting path has length k then in a single phase we obtain a *maximal* set S of vertex disjoint augmenting paths all of length k.) By the maximality property, we have that *any* augmenting path of length k will intersect a path in S. In the next phase, we will have the property that the augmenting paths found will be strictly longer (we will prove this formally later). We will implement each phase in linear time.

We first prove the following lemma.

Lemma 4.1 *Let M be a matching and P a shortest augmenting path w.r.t M. Let P' be a shortest augmenting path w.r.t $M \oplus P$ (symmetric difference of M and P). We have*

$$|P'| \geq |P| + 2|P \cap P'|.$$

$|P \cap P'|$ refers to the edges that are common to both the paths.

Proof:

Let $N = (M \oplus P) \oplus P'$. Thus $N \oplus M = P \oplus P'$. Consider $P \oplus P'$. This is a collection of cycles and paths (since it is the same as $M \oplus N$). The cycles are all of even length. The paths may be of odd or even length. The odd length paths are augmenting paths w.r.t M. Since the two matchings differ in cardinality by 2, there must be two odd length augmenting paths P_1 and P_2 w.r.t M. Both of these must be longer than P (since P was the shortest augmenting path w.r.t M).

$$|M \oplus N| = |P \oplus P'| = |P| + |P'| - 2|P \cap P'| \geq |P_1| + |P_2| \geq 2|P|.$$

Simplifying we get

$$|P'| \geq |P| + 2|P \cap P'|.$$

$\square$

We still need to argue that after each phase, the shortest augmenting path is strictly longer than the shortest augmenting paths of the previous phase. (Since the augmenting paths always have an odd length, they must actually increase in length by two.)

Suppose that in some phase we augmented the current matching by a maximal set of vertex disjoint paths of length k (and k was the length of the shortest possible augmenting path). This yields a new matching M'. Let P' be an augmenting path with respect to the new matching. If P' is vertex disjoint from each path in the previous set of paths it must have length strictly more than k. If it shares a vertex with some path P in our chosen set of paths, then $P \cap P'$ contains at least one edge in M' since every vertex in P is matched in M'. (Hence P' cannot intersect P on a vertex and not share an edge with P.) By the previous lemma, P' exceeds the length of P by at least two.

Lemma 4.2 *A maximal set S of disjoint, minimum length augmenting paths can be found in $O(m)$ time.*

Proof:

Let $G = (U, V, E)$ be a bipartite graph and let M be a matching in G. We will grow a "Hungarian Tree" in G. (The tree really is not a tree but we will call it a tree all the same.) The procedure is similar to BFS and very similar to the search for an augmenting path that was done in the previous lecture. We start by putting the free vertices in U in level 0. Starting from even level $2k$, the vertices at level $2k + 1$ are obtained by following free edges from vertices at level $2k$ that have not been put in any level as yet. Since the graph is bipartite the odd(even) levels contain only vertices from $V(U)$. From each odd level $2k+1$, we simple add the matched neighbours of the vertices to level $2k+2$. We repeat this process until we encounter a free vertex in an odd level (say t). We continue the search only to discover all free vertices in level t, and stop when we have found all such vertices. In this procedure clearly each edge is traversed at most once, and the time is $O(m)$.

We now have a second phase in which the maximal set of disjoint augmenting paths of length k is found. We use a technique called *topological erase*, called so because it is a little like topological sort. With *each* vertex x (except the ones at level 0), we associate an integer counter initially containing the number of edges entering x from the previous level (we also call this the indegree of the vertex). Starting at a free vertex v at the last level t, we trace a path back until arriving at a free vertex in level 0. The path is an augmenting path, and we include it in S.

At all points of time we maintain the invariant that there is a path from each free node (in V) in the last level to some free node in level 0. This is clearly true initially, since each free node in the last level was discovered by doing a BFS from free nodes in level 0. When we find an augmenting path we place all vertices along this path on a deletion queue. As long as the deletion queue is non-empty, we remove a vertex from the queue, delete it together with its adjacent vertices in the Hungarian tree. Whenever an edge is deleted, the counter associated with its right endpoint is decremented if the right endpoint is not on the current path (since this node is losing an edge entering it from the left). If the counter becomes 0, the vertex is placed on the deletion queue (there can be no augmenting path in the Hungarian tree through this vertex, since all its incoming edges have been deleted). This maintains the invariant that when we grab paths coming backwards from the final level to level 0, then we automatically delete nodes that have no paths backwards to free nodes.

After the queue becomes empty, if there is still a free vertex v at level t, then there must be a path from v backwards through the Hungarian tree to a free vertex on the first level; so we can repeat this process. We continue as long as there exist free vertices at level t. The entire process takes linear time, since the amount of work is proportional to the number of edges deleted. □

Let's go through the following example (see Fig. **??**) to make sure we understand what's going on. Start with the first free vertex v_6. We walk back to u_6 then to v_5 and to u_1 (at this point we had a choice of u_5 or u_1). We place all these nodes on the deletion queue, and start removing them and their incident edges. v_6 is the first to go, then u_6 then v_5 and then u_1. When we delete u_1 we delete the edge (u_1, v_1) and decrease the indegree of v_1 from 2 to 1. We also delete the edges (v_6, u_3) and (v_5, u_5) but these do not decrease the indegree of any node, since the right endpoint is already on the current path.

Start with the next free vertex v_3. We walk back to u_3 then to v_1 and to u_2. We place all these nodes on the deletion queue, and start removing them and their incident edges. v_3 is the first to go, then u_3 (this deletes edge (u_3, v_4) and decreases the indegree of v_4 from 2 to 1). Next we delete v_1 and u_2. When we delete u_2 we remove the edge (u_2, v_2) and this decreases the indegree of v_2 which goes to 0. Hence v_2 gets added to the deletion queue. Doing this makes the edge (v_2, u_4) get deleted, and drops the indegree of u_4 to 0. We then delete u_4 and the edge (u_4, v_4) is deleted and v_4 is removed. There are no more free nodes in the last level, so we stop. The key point is that it is not essential to find the maximum set of disjoint augmenting paths of length k, but only a maximal set.

Theorem 4.3 *The total running time of the above described algorithm is $O(\sqrt{n}m)$.*

Proof:

Each phase runs in $O(m)$ time. We now show that there are $O(\sqrt{n})$ phases. Consider running the algorithm for exactly $\sqrt{n}$ phases. Let the obtained matching be M. Each augmenting path from now on is of length at least $2\sqrt{n}+1$. (The paths are always odd in length and always increase by at least two after each phase.) Let M^* be the max matching. Consider the symmetric difference of M and M^*. This is a collection of cycles and paths, that contain the augmenting paths w.r.t M. Let $k = |M^*| - |M|$. Thus there are k augmenting paths w.r.t M that yield the matching M^*. Each path has length at least $2\sqrt{n}+1$, and they are disjoint. The total length of the paths is at most n (due to the disjointness). If l_i is the length of each augmenting path we have:

$$k(2\sqrt{n}+1) \leq \sum_{i=1}^{k} l_i \leq n.$$

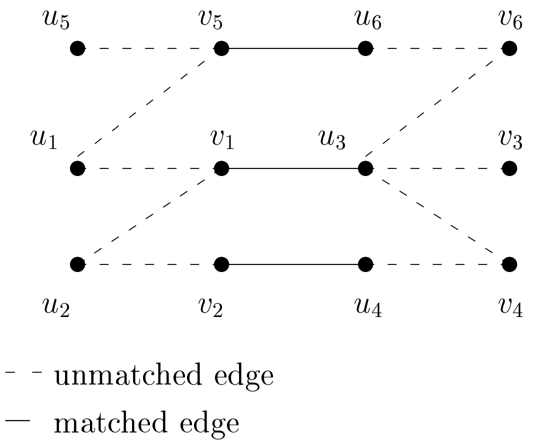

Figure 1: Sample execution of Topological Erase.

Thus k is upper bounded by $\frac{n}{2\sqrt{n}+1} \leq \frac{n}{2\sqrt{n}} \leq \frac{\sqrt{n}}{2}$. In each phase we increase the size of the matching by at least one, so there are at most k more phases. This proves the required bound of $O(\sqrt{n})$. $\square$