SUN Yilin
520030910361

Edmonds' Blossom Algorithm

Algorithms
June 22, 2022

1

1.1

Proof of M is a maximum matching $\implies$ no M-augmenting path exists:
If there is an M-augmenting path denoted by $P = e_1e_2e_3 \dots e_n$, where $e_i = (v_{i-1}, v_i), \forall i \in [n]$. Then by definition of augmenting path we know that $e_1 \notin M, e_n \notin M$. It follows that, by definition of M-alternating path, $E_{even} = \{e_2, e_4, \dots, e_{n-1}\} \in M$ and $E_{odd} = \{e_1, e_3, e_5, \dots, e_{n-2}, e_n\} \notin M$. Then simply by switching the edges between E_{even} and E_{odd} and construct another matching $M' = (M \backslash E_{even}) \cup E_{odd}$, we would be able to construct a larger matching M', because obviously $|E_{even}| < |E_{odd}|$.

1.2

Proof of M is a maximum matching $\impliedby$ no M-augmenting path exists:
If M is not a maximum matching, following the hint, let M' be a maximum matching such that $|M \cap M'|$ is maximized, we now consider the subgraph $H = M \cup M'$ of G. Here when constructing the subgraph we do a subtle but important modification that **for any edge that is both in M and M' we count it twice.** We want to analyse the connected components of H. The degree of any vertex in H is at most 2 because any vertex has at most two edges incident to it. Thus the connected components of H is either a single vertex, a path, or a cycle. We now consider the edge distribution of these cases respectively.
Case 1: Single vertex.
Trivial since no edges in these components.
Case 2: Cycle.
Any cycle of H must be even length otherwise there will be two edges adjacent to each other and from the same matching. It follows that any cycle must contain exactly the same number of edges from M and M'.
Case 3: Path.
The paths can be either even length or odd length. For those even length path, like the cycles, they also contain same number of edges from M and M'. For those odd length path denoted by $P = e_1e_2e_3 \dots e_n$, we define $E_{even} = \{e_2, e_4, \dots, e_{n-1}\}$ and $E_{odd} = \{e_1, e_3, e_5, \dots, e_{n-2}, e_n\}$. Then either $E_{even} \in M, E_{odd} \in M'$ or $E_{even} \in M', E_{odd} \in M$.

From the above three cases, the only case where a connected component contains more edges from M' than M is in case 3 where there is a path P with odd length and $E_{odd} \in M', E_{even} \in M$. Since we know M' is a maximum matching, $|M'| > |M|$, then at least one

connected component is in the form of such a path P with odd length. But such a path P is actually an $M-augmenting$ path because, as $E_{odd} \in M'$ the endpoints of P are not covered by matching M. Thus we can conclude that if M is not a maximum matching, there will be an M-augmenting path. So if no M-augmenting path exists M must be a maximum matching.

2

First we introduce some notations that will be used in the proof.
Denote $M \backslash C$ by M'.
Denote vertices in a cycle C in G with $2k+1$ vertices by $\{v_1, v_2, \ldots, v_{2k+1}\}$.
Denote the vertex added by the contraction process in the new graph G' by u.

2.1

Proof of M is a maximum matching of $G \implies M'$ is a maximum matching of G':
Suppose M' is not a maximum matching of G', then there is an M'-augmenting path P' in G' by what we have proven in problem 1. Since C contains exactly k edges in M and meets no other edges in M, any edge in G' incident with u must not be in M'. Then u must be an endpoint of P', otherwise P' would also be an M-augmenting path in G. And clearly the other endpoint s of P' must not be in C.
Now consider the path P' in G. Denote the vertex in C which is not covered by edge in M by v^*, clearly v^* is unique. Denote the vertex in C which is an endpoint of P' by v_i, clearly v_i is also unique. (Note that v_i and v^* may be equal but that does not matter.)
For v_i, we starts from the edge that is both incident with v_i and in $C \cap M$ and moves along the cycle till we meet v^*, denote such a path by P^*(P^* is empty when $v_i = v^*$ but that does not affect our proof). Then clearly $P' \cup P^*$ with endpoints s and v^* is an M-augmenting path in G, which means that M cannot be a maximum matching, hence contradicts with our assumption. Thus M' must be a maximum matching of G'.

2.2

Proof of M is a maximum matching of $G \impliedby M'$ is a maximum matching of G':
Suppose M is not a maximum matching of G, then there is an M-augmenting path P in G. If P does not intersect any of the vertices in C, then obviously P is an M'-augmenting path in G'. If P intersects cycle C, we can always find an endpoint of P which is not in C since C contains exactly one vertex that is not covered by M. Denote such an endpoint outside C by w. In G', we starts from w and moves along the path P till we reach u, as we have shown in part 2.2, u is not covered by M', so we have found a path P' starting from w and ending at u which is an M'-augmenting path in G'. In either case there is an M'-augmenting path in G', which contradicts with our assumption, hence M must be a maximum matching of G.

3

Firstly, if all the vertices are already covered by the matching M, there will be no valid root for the tree, now the M-alternating tree for this special case is an empty set. Actually, even if there are uncovered vertices for a matching M, there can be valid root in a possible M-alternating tree. Yet we can still choose the empty set as an M-alternating tree, which completes the proof of the existence of M-alternating tree.

What's important here is to find a maximal one in polynomial-time, we can carry out this task by a modified DFS process with certain search rules.

Step 1:

Find the set of all uncovered vertices $R = \{r_1, r_2, \ldots, r_n\}$.

Step 2:

Pick an element that has not been traversed r^*, r^* will be the root of a component of the maximal M-alternating tree.

Step 3:

Start a DFS from root r^* by the following rule:

In this DFS, never traverse a vertex that has already been visited before.

If current vertex is a root or an outer vertex, we must do DFS along an edge that connects a vertex that is covered.

If current vertex is an inner vertex, we must do DFS along the edge that covers the very inner vertex.

Step 4:

If all vertices have been traversed, then we have already find a maximal M-alternating forest. If not, go to step 2.

Clearly all the steps mentioned above can be done in polynomial time, Even though step 2 is a loop, it will be carried out at most $|V|$ times where V is the set of all vertices. So this process can be done in polynomial time.

We now prove that such an M-alternating forest is maximal. To prove this, we only need to show that adding an edge will make this forest not an M-alternating forest any more.

For those edges that are within a component of such a forest, clearly we cannot add them because the components must be a tree. For those edges that are between two components of such a forest, we cannot add them because once we add them, we merge two trees and generate a tree which has two vertices that are not covered, which violates the definition 1 of an M-alternating forest. So such a forest is maximal.

4

Firstly, since M is a maximal matching, there will be no edges between roots of components of F. (Otherwise we can add this edge to M, M is not maximal)
Also, any M-augmenting path P must have two endpoints which are not covered by M, thus the endpoints can only be the roots of components of F.

We first consider the situation where the M-augmenting path contains only one edge denoted by w which connets two components of forest F. Now consider the vertices incident with w, there are three possible cases.
Case 1: $w = (u, v)$ connects two outer vertices from two trees.
In this case u, v are the two consecutive outer vertices we want.
Case 2: $w = (u, v)$ connects an inner vertex u and an outer vertex v.
We consider the tree T where $u \in T$. Since w connects two components of F, $w \notin E$. Then the subset of augmenting path P that passes through vertices in T must start with the root of T and end with an edge $e \in M$ where $e = (s, u)$ and s is an outer vertex and s is a child of u. Since s is a child of u, the path starting from root of T must first path another branch and then reaches s. For an augmenting path, the edge between two branches of an alternating tree must be connecting two outer vertices, so there are two consecutive outer vertices in the augmenting path.
Case 3: $w = (u, v)$ connects two inner vertices from two trees.
Like case 2, here the proof is the same other than that both the trees containing u and v will have an edge which is connecting two outer vertices.
To sum up, in the above three cases, there will always be two consecutive outer vertices.

We then consider the situation where the M-augmenting path contains more than one edge connecting components of forest F. In this situation, if there is an edge that is connecting two outer vertices, then we are done. If not, like the proof in case 2, we can always find a tree of F where the M-augmenting path connects two consecutive outer vertices of the tree.

To sum up, any M-augmenting path must contain two consecutive outer vertices.

5

5.1

If belong to distinct components of F, then denote the root of u by r_u, denote the root of v by r_v, denote the path from r_u to u in the component of F by P_u, denote the path from r_v to v in the component of F by P_v, then clearly $P_u + (u, v) + P_v$ is an M-augmenting path with endpoints r_u, r_v.

5.2

If belong to the same component of F, then there is a unique path P from u to v in the very component. By definition of M-alternating forest the lowest common ancestor of u and v must also be an outer vertex and the length of the unique path between any two outer vertices must be even. Then we know the length of path P is also even. Then path P plus the extra edge that connects u and v in G but not in F forms an odd cycle.

6

Algorithm Description:
Step 1:
Pick the edges whose endpoints have not been covered one by one, till we have got a maximal matching M.
Step 2:(M-alternating forest building)
Build a maximal M-alternating forest using the method we described in problem 3. Meanwhile, mark all the edges that are not in this M-alternating forest, i,e, mark all the edges in $E\backslash F$ as we need to check these edges in the next step.
Step 3:(Blossom contraction)
For all those edges in $E\backslash F$, if there is still an edge (u, v) which connects two outer vertices in F belonging to the same component in F. start a DFS **with only edges in** F from either u or v until we find a cycle(the Blossom), then contracts this Blossom.
Step 4:(M-augmenting path finding)
For all those edges in $E\backslash F$, find an edge connecting two outer vertices belonging to different components of F then DFS **with only edges in** F from either u or v to find an M-augmenting path.
If there is no such edges, go to step 7.
Repeatedly find all such edges and all disjoint M-augmenting paths, denote the set of disjoint M-augmenting paths by P.
Step 5:
For all paths in P, switch those edges that are in M and not in M to update out matching M, after which we will have a new matching with larger cardinality.
Step 6:
With the new matching M, go to step 2 again(to build a new M-alternating forest corresponding with updated M).
Step 7:
We have got us a maximum matching in the graph with all the blossoms contracted, then by openning all those blossoms we can get the maximum matching in the original graph.

Proof of Correctness:
Firstly, the only way this algorithm exits at step 7 is from step 4 when there is no two consecutive outer vertices belonging to different components. Since we have contracted all blossoms in step 3, there will be no two consecutive outer vertices in the graph, then by problem 4, there will be no M-augmenting path in this graph. Then by problem 1, this matching is a maximum matching.
Also, py problem 2, the blossom contracting process will not affect the maximum matching

in the graph generated or the original graph. This means that at step 3 we can safely contract those blossoms and use the maximum matching in the contracted graph to generate a maximum matching in the original graph.
As we have shown in problem 3, steps 1 and 2 can always be finished within polynomial time and we can always find such maximal matching and maximal forests.
At step 5 when we are updating the maximal matching, since we are replacing edges from all those disjoint augmenting paths, we can guarantee that we will still get a maximal matching.
By the above statements, we have completed our proff of correctness or our algorithm.

Time complexity:
Here we use V, E to represent the cardinality of set of all vertices and edges respectively.
Step 1 can be done in $O(E)$ time, we only need to initialize once, so actually this time is not that important.
Since each vertex can only be covered by one edge, the forest building process at step 2 only takes $O(V)$ times.
At each iteration after step 2, it takes us $O(V)$ time to find all those consecutive outer vertices, either this leads to a blossom or an augmenting path, it takes us $O(E)$ time to check and also $O(E)$ time to contract the blossom or update matching M according to the augmenting paths.
So the total time complexity is $O(V)$ iterations times $O(V)$ outer vertices finding times $O(E)$ checking, which is $O(EV^2)$.