Assignment 2: Maximum Bipartite Matching

Jainil Shah, Harsh Jain, Harsh Kushwaha,
Dipin Garg, Rahul Kumar

6th February 2023

1 Introduction

For a graph $G = (V, E)$, a matching M is defined a as subset of the edge-set E such that no two edges are coincident on the same vertex. A maximum matching is a matching with the largest possible size in the graph.

For a unweighted bipartite graph, the maximum matching can be computed using the augmenting path algorithm or by the Hopcroft-Karp Algorithm.

2 Augmenting Path Algorithm

Let M be a matching of the graph $G = (V, E)$. An **alternating path** defined w.r.t. M is a path which has edges alternating in M and $\bar{M}$. An **augmenting path** defined w.r.t. M is an alternating path which starts and end on an unmatched vertex.

It can be noticed that for every augmenting path P w.r.t. to the matching M, we can take the symmetric difference of M with P to increase the cardinality of M by one. Hence, we can formulate the following algorithm.

Algorithm 1 Augmenting Path Algorithm

```
    Input G = (X ∪ Y, E)
    Output A maximum matching M
1: M = φ
2: while ∃ path P|P is augmenting w.r.t. M do
3:     M = M△P                              ▷ Symmetric Difference
4: end while
```

For the implementation, we can find augmenting paths using BFS/DFS after converting G to a directed graph $G(M)$. First, for all edges $e \in \bar{M}$, we assign them directions from X to Y, while for edges in M we assign direction from Y to X. Now we can run a multisource BFS/DFS from all the unmatched edges in X. As soon as we reach an unmatched edge in Y, we have found an augmenting path.

Algorithm 2 Algorithm to find an Augmenting Path for a Matching

```
    Input G = (X ∪ Y, E), matching M ⊂ E
    Output An Augmenting Path P
V' = X ∪ Y
E' = φ
P = φ
digraph G(M) = (V', E')
for edges e ∈ E do
    if e = u, v ∈ M then
        E' = E' ∪ (v, u)                ▷ Direction from Y to X
    else
        E' = E' ∪ (u, v)                ▷ Direction from X to Y
    end if
end for
V' = V' ∪ {s}                           ▷ Source for BFS
for vertices v ∈ X do
    if v is not matched then
        E' = E' ∪ (s, v)
    end if
end for
DFS(s, P)
create path from vertices
return P

procedure DFS(v, P)
    if v ∈ Y and is not matched then
        P = P ∪ {v}
        return True
    end if
    for Neighbours to of v do
        if BFS(to, P) then
            P = P ∪ {v}
            return True
        end if
    end for
    return False
end procedure
```

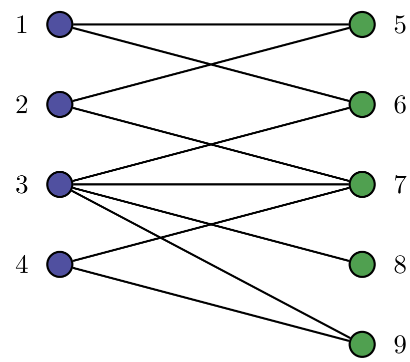
3 Complexity Analysis

For a bipartite graph with n vertices with a maximum matching M, $|M| \leq \frac{n}{2}$, hence the while loop on line 2 of Algorithm 1 will run atmost $\frac{n}{2}$ times because each loop will increase the cardinality of the matching by atleast 1.

Next, the Algorithm 2 runs a linear time BFS/DFS and hence runs in $O(V+E)$ time. Hence the time complexity of the algorithm is $O(V \cdot E)$.

4 Results

We run the code for the follwing graph:



The results obtained are as follows:

```
masterchief410@warthog410:advanced-algorithms-lab/tut2$ g++ code.cpp
masterchief410@warthog410:advanced-algorithms-lab/tut2$ ./a.out
1 6
2 5
3 8
4 7
```

Hence the matching is as follows: