Hopcroft-Karp Algorithm

Rakib Abdullah-1905047
Al-Amin Sany-1905048
Bijoy Saem-1905052

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

February 23, 2023

Table of Contents

Introduction

Hopcroft-Karp Algorithm

The Hopcroft-Karp algorithm is a graph algorithm that finds the maximum cardinality matching in a bipartite graph.

Introduction

Hopcroft-Karp Algorithm

The Hopcroft-Karp algorithm is a graph algorithm that finds the maximum cardinality matching in a bipartite graph.

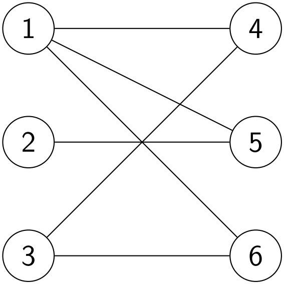


Figure: Bipartite graph

Introduction

Hopcroft-Karp Algorithm

The Hopcroft-Karp algorithm is a graph algorithm that finds the maximum cardinality matching in a bipartite graph.

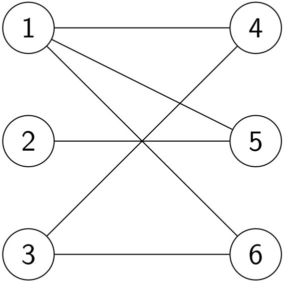


Figure: Bipartite graph

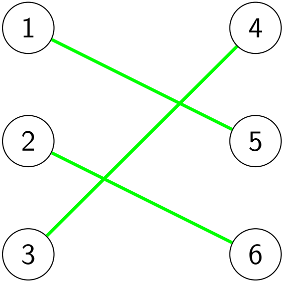


Figure: Maximum cardinality matching

Table of Contents

Definitions

Bipartite Graph

A graph is bipartite if its vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set.

Definitions

Bipartite Graph

A graph is bipartite if its vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set.

Matching

A matching in a graph is a set of edges such that no two edges share a common vertex.A matching is said to be maximum if it contains the maximum number of edges possible

Definitions

Bipartite Graph

A graph is bipartite if its vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set.

Matching

A matching in a graph is a set of edges such that no two edges share a common vertex.A matching is said to be maximum if it contains the maximum number of edges possible

Free vertex

a free vertex refers to a vertex in the left part of the bipartite graph that is not yet matched with any vertex in the right part of the graph.

Table of Contents

Algorithm

Hopcroft-Karp(G)

1: Initialize the matching $M = \phi$

Algorithm

Hopcroft-Karp(G)

1: Initialize the matching $M = \phi$

2: **while** there exists an augmenting path P with respect to M **do**

Algorithm

Hopcroft-Karp(G)

```
Initialize the matching M = φ
while there exists an augmenting path P with respect to M do
    Use a breadth-first search to find an augmenting path P with respect to M.
```

Algorithm

Hopcroft-Karp(G)

1: Initialize the matching $M = \phi$
2: **while** there exists an augmenting path P with respect to M **do**
3: Use a breadth-first search to find an augmenting path P with respect to M.
4: Use a depth-first search to augment the matching M along P.

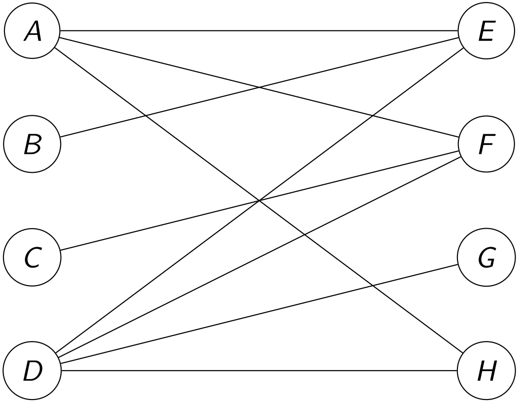
Algorithm

Hopcroft-Karp(G)

1: Initialize the matching $M = \phi$
2: **while** there exists an augmenting path P with respect to M **do**
3: Use a breadth-first search to find an augmenting path P with respect to M.
4: Use a depth-first search to augment the matching M along P.
5: **end while**
6: Output the matching M as the maximum cardinality matching.

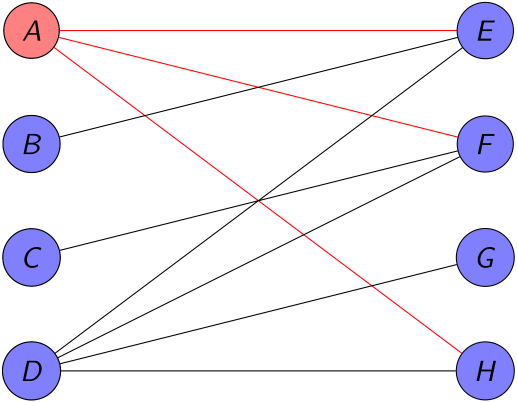
Example

A bipartite graph is given below where we will find its maximum cardinality mathching:



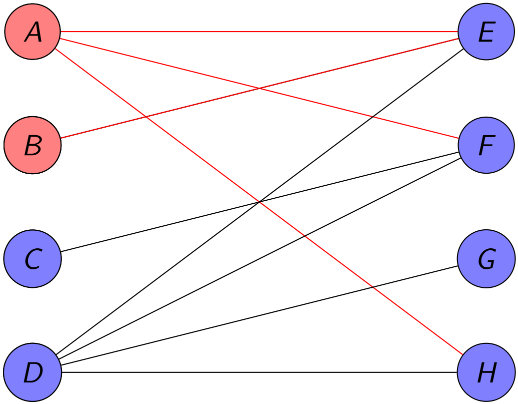
First iteration

Running BFS on the free vertices of the left side...



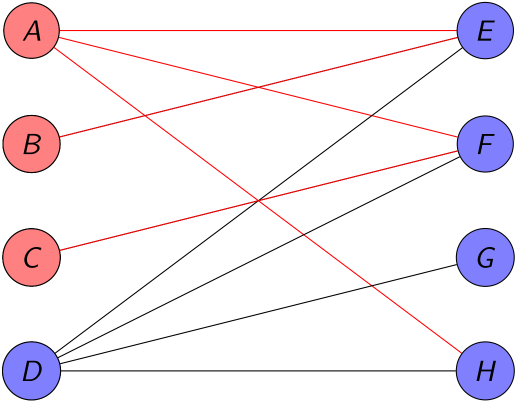
First iteration

Running BFS on the free vertices of the left side...



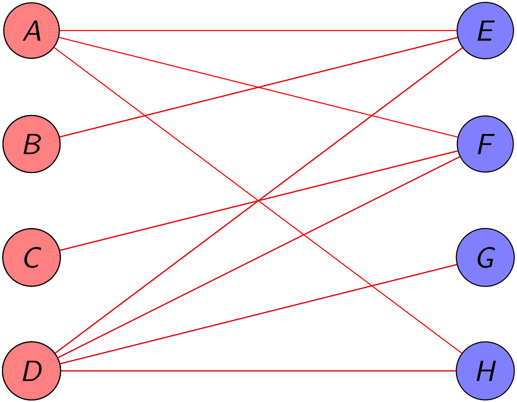
First iteration

Running BFS on the free vertices of the left side...



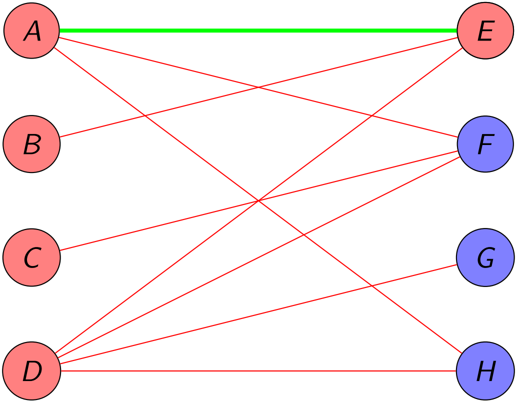
First iteration

Running BFS on the free vertices of the left side...



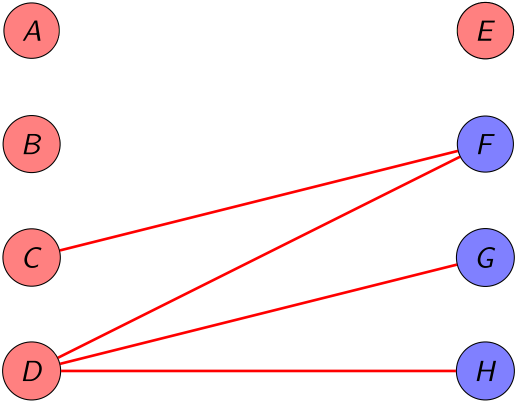
First iteration

Running DFS on the free vertices of the right side...



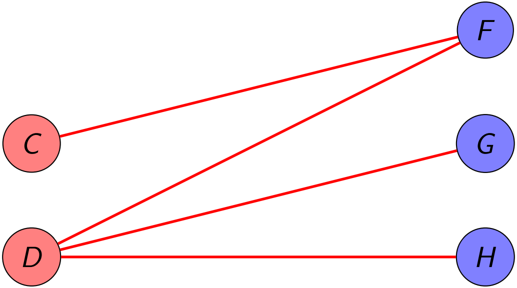
First iteration

Running DFS on the free vertices of the right side...



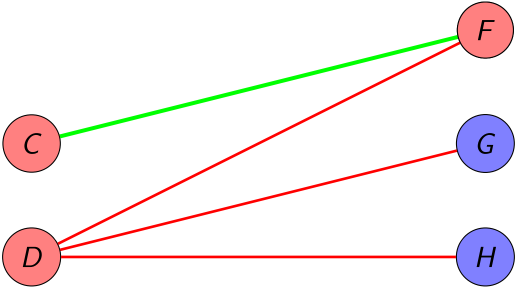
First iteration

Running DFS on the free vertices of the right side...



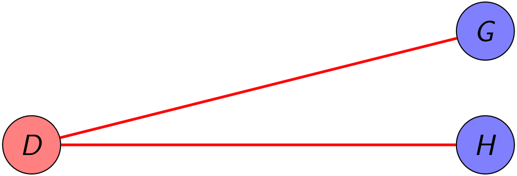
First iteration

Running DFS on the free vertices of the right side...



First iteration

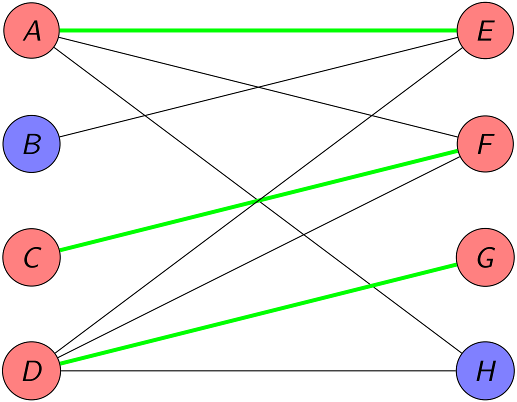
Running DFS on the free vertices of the right side...



After first iteration

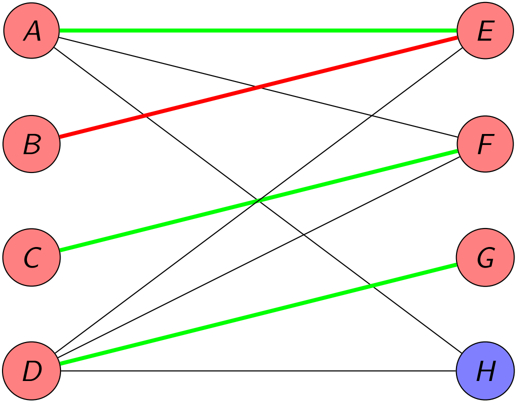
After running the DFS on the remaining part,

$$M = \{A - E, C - F, G - D\}$$



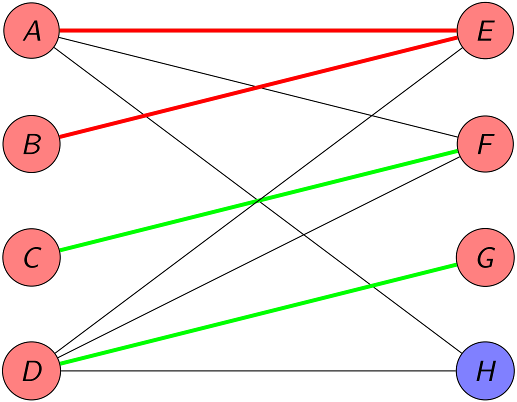
Second iteration

Running BFS on the free vertices of left side..



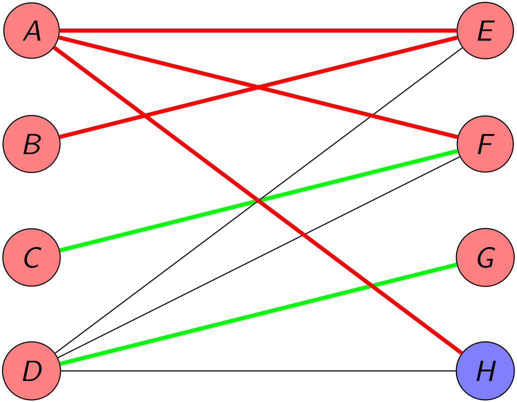
Second iteration

Running BFS on the free vertices of left side..



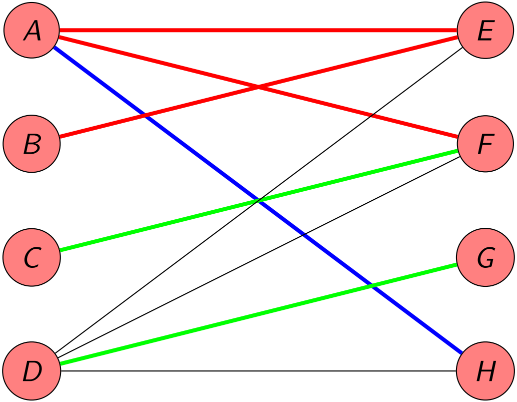
Second iteration

Running BFS on the free vertices of left side..



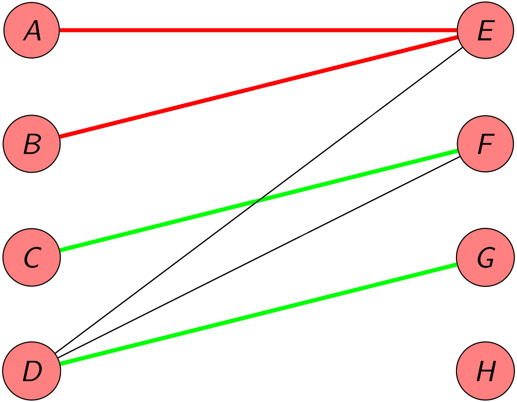
Second iteration

Running DFS on the free vertices of right side...



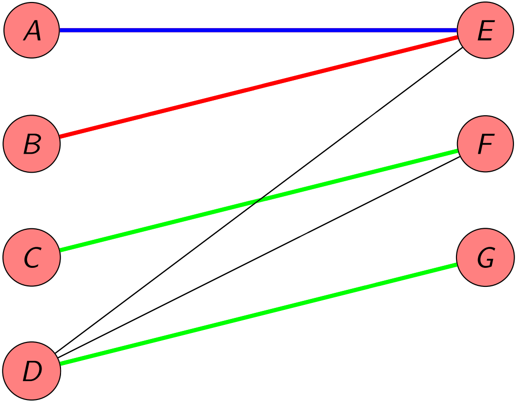
Second iteration

Running DFS on the free vertices of right side...



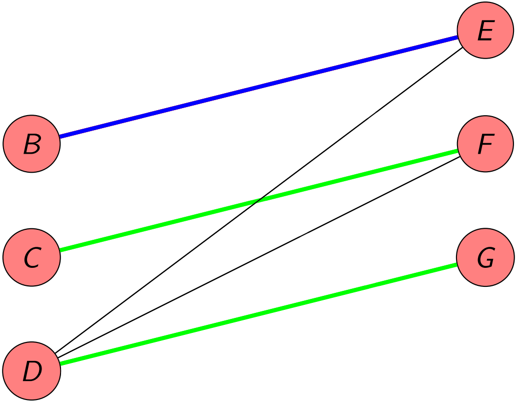
Second iteration

Running DFS on the free vertices of right side...



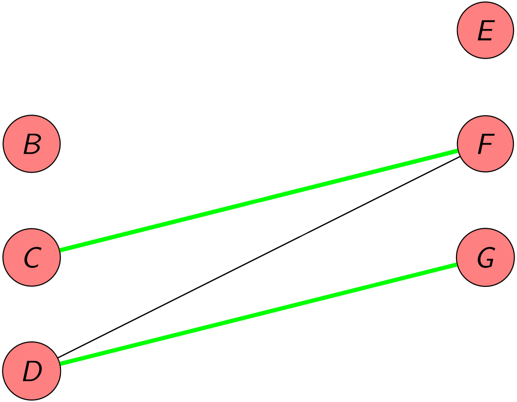
Second iteration

Running DFS on the free vertices of right side...



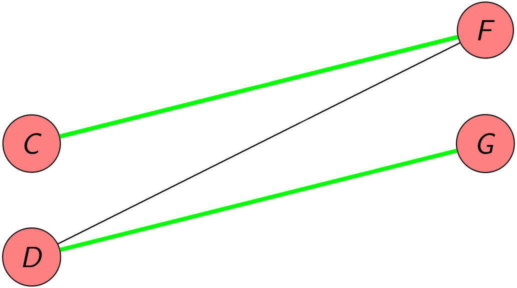
Second iteration

Running DFS on the free vertices of right side...



Second iteration

Running DFS on the free vertices of right side...



Algorithm termination..

As no more free vertex is available,the algorithm terminates and finally we get the following graph with maximum cardinaliy 4 where $M = \{A - H, B - E, C - F, D - G\}$

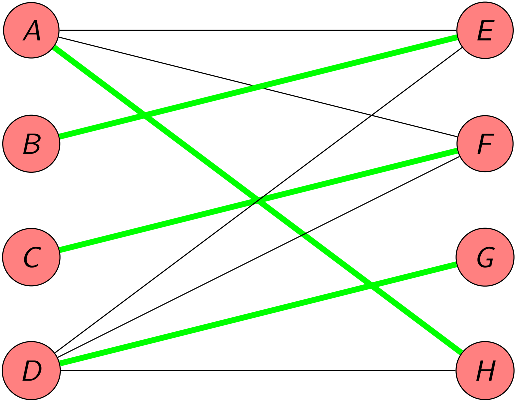


Table of Contents

Time Complexity

- The time complexity of the algorithm is O(E * sqrt(V)), where E is the number of edges and V is the number of vertices in the graph. This means that the time required to run the algorithm increases linearly with the number of edges, but is also influenced by the square root of the number of vertices.

Time Complexity

- The time complexity of the algorithm is O(E * sqrt(V)), where E is the number of edges and V is the number of vertices in the graph. This means that the time required to run the algorithm increases linearly with the number of edges, but is also influenced by the square root of the number of vertices.
- The time complexity of the algorithm is considered efficient for most bipartite graphs, but may not be the best choice for extremely large graphs.

Table of Contents

Applications

- Image segmentation - finding matches between objects in an image and a pre-defined set of object templates.

Applications

- Image segmentation - finding matches between objects in an image and a pre-defined set of object templates.
- Job scheduling - matching workers with tasks based on their skills and availability.

Applications

- Image segmentation - finding matches between objects in an image and a pre-defined set of object templates.
- Job scheduling - matching workers with tasks based on their skills and availability.
- Online advertising - matching ads with potential viewers based on demographic and behavioral data.