Hopcroft-Karp Algorithm

1905047-Rakib Abdullah
1905048-Al-Amin Sany
1905052-Bijoy Ahmed Saiem

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

February 25, 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.

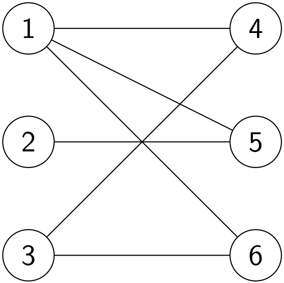


Figure: Bipartite graph

Introduction

Hopcroft-Karp Algorithm

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

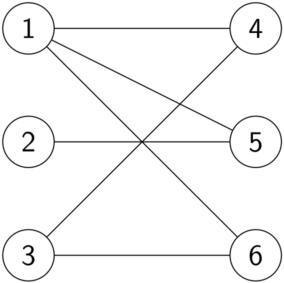


Figure: Bipartite graph

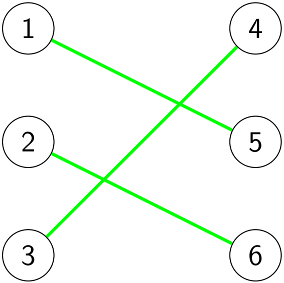


Figure: Maximum cardinality matching

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.

Maximum Cardinality

A matching in a graph is a set of edges such that no two edges share a common vertex.Maximum cardinality means maximum possible matching.

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.

Maximum Cardinality

A matching in a graph is a set of edges such that no two edges share a common vertex.Maximum cardinality means maximum possible matching.

Free vertex

A free vertex is a vertex with no matching edge connected to it.

Table of Contents

Algorithm

Hopcroft-Karp(G)

1: $M = \phi$

Algorithm

Hopcroft-Karp(G)

1: $M = \phi$

2: **while** $P \neq \phi$ **do**

Algorithm

Hopcroft-Karp(G)

$M = \phi$

while $P \neq \phi$ **do**

    $P = \{p_1, p_2, \ldots, p_k\}$ //vertex-disjoint shortest augmenting paths

Algorithm

Hopcroft-Karp(G)

```
M = φ
while P ≠ φ do
    P = {p_1, p_2, ..., p_k} //vertex-disjoint shortest augmenting paths
    M := M ⊕ {p_1 ∪ p_2 ∪ ··· ∪ p_k}
```

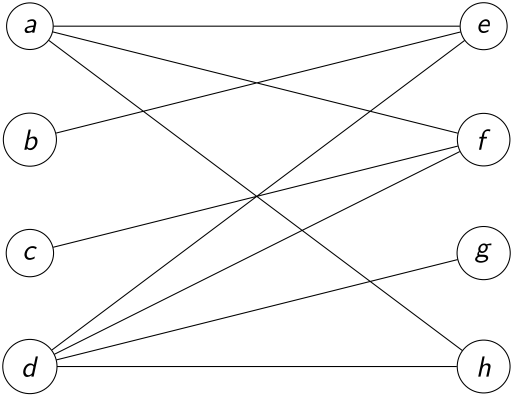
Algorithm

Hopcroft-Karp(G)

1: $M = \phi$
2: **while** $P \neq \phi$ **do**
3: $\quad P = \{p_1, p_2, \ldots, p_k\}$ //vertex-disjoint shortest augmenting paths
4: $\quad M := M \oplus \{p_1 \cup p_2 \cup \cdots \cup p_k\}$
5: **end while**
6: Output the matching M as the maximum cardinality matching.

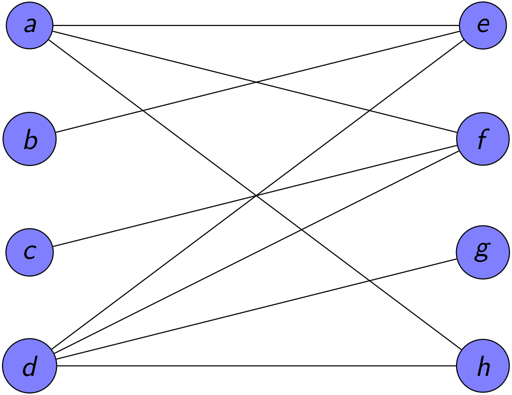
Example

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



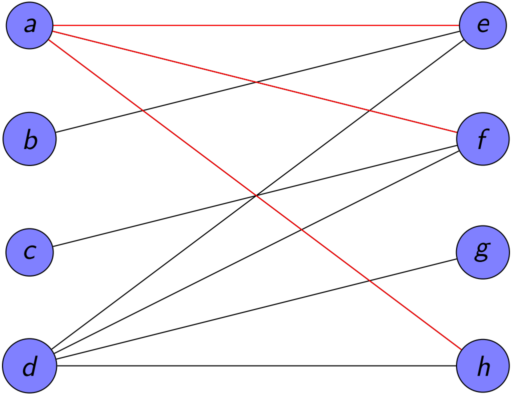
First iteration:BFS

$$F = \{\}$$



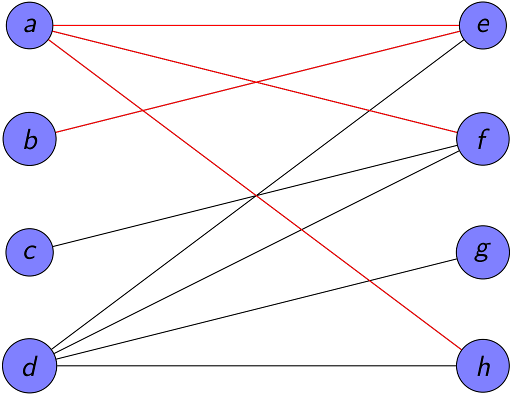
First iteration:BFS

$$F = \{e, f, h\}$$



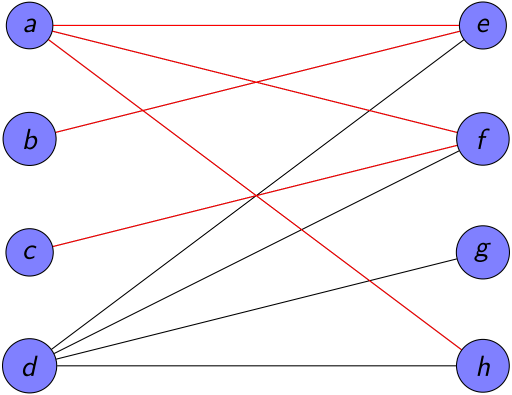
First iteration:BFS

$$F = \{e, f, h\}$$



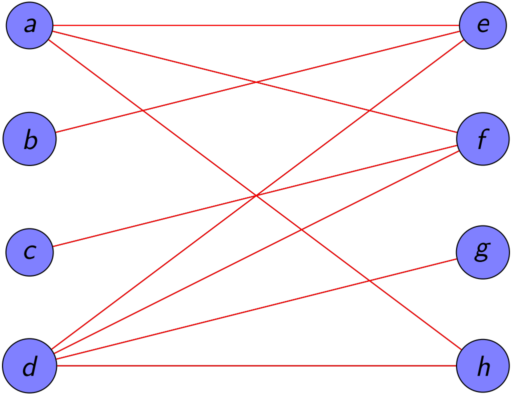
First iteration:BFS

$$F = \{e, f, h\}$$



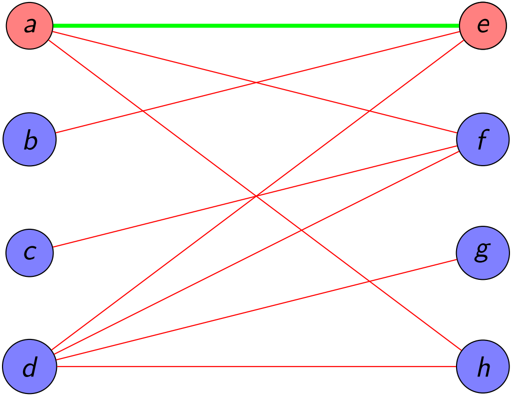
First iteration:BFS

$$F = \{e, f, h, g\}$$



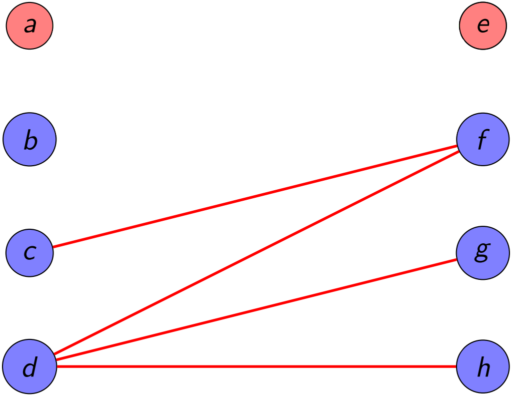
First iteration:DFS

$F = \{e, f, h, g\}$
$P = \{(e - a)\}$



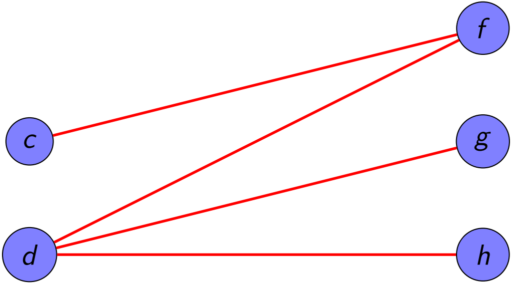
First iteration:DFS

$$F = \{e, f, h, g\}$$
$$P = \{(e - a)\}$$



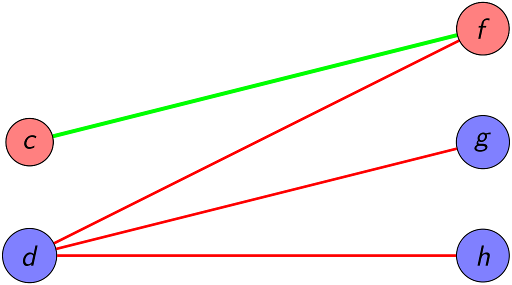
First iteration:DFS

$$F = \{f, h, g\}$$
$$P = \{(e - a)\}$$



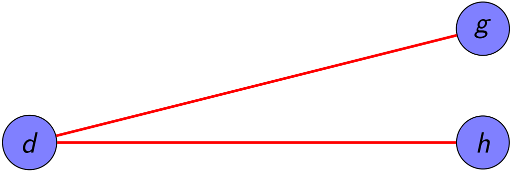
First iteration:DFS

$$F = \{f, h, g\}$$
$$P = \{(e-a), (f-c)\}$$



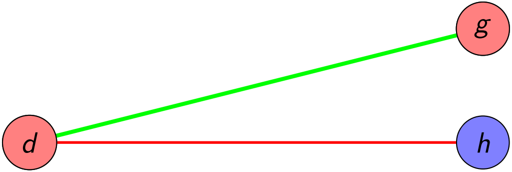
First iteration:DFS

$$F = \{h, g\}$$
$$P = \{(e - a), (f - c)\}$$



First iteration:DFS

$$F = \{h, g\}$$
$$P = \{(e - a), (f - c), (g - d)\}$$



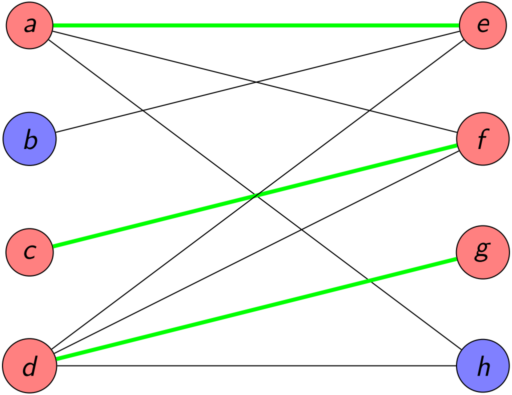
First iteration:DFS

$$F = \{h\}$$
$$P = \{(e - a), (f - c), (g - d)\}$$

After first iteration

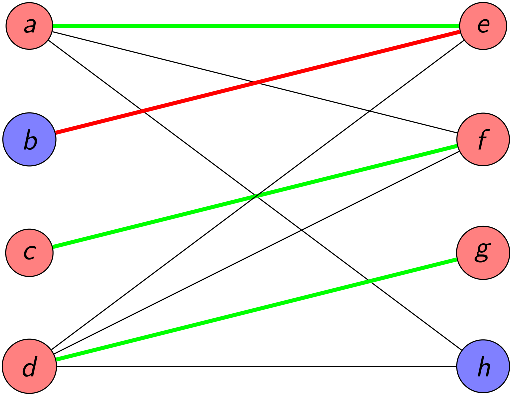
After running the DFS on the remaining part,

$$M = \{(e - a), (f - c), (g - d)\}$$



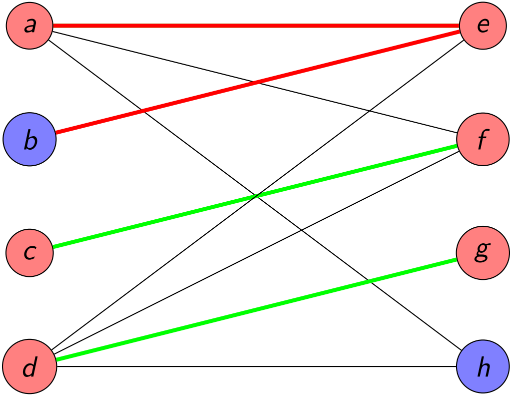
Second iteration:BFS

$F = \{\}$
Augmenting path $= b- > e$



Second iteration:BFS

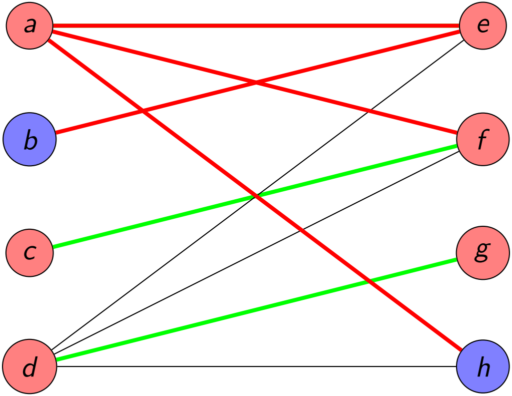
$F = \{\}$
Augmenting path $= b- > e- > a$



Second iteration:BFS

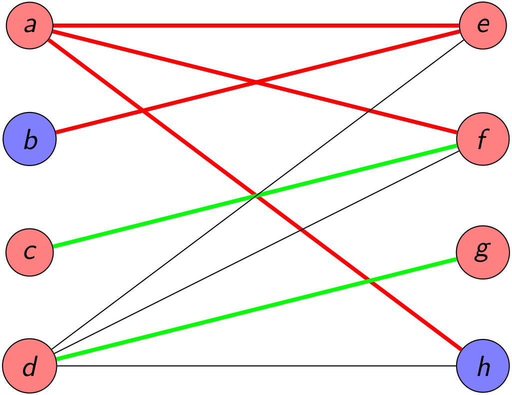
$F = \{H\}$

Augmenting path $= b- > e- > a- > h$



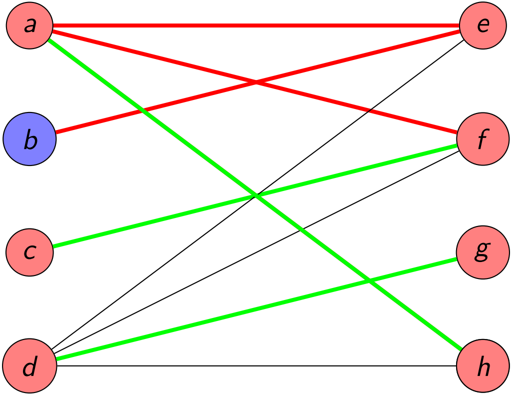
Second iteration:DFS

$$F = \{H\}$$
$$P = \{\}$$
$$M = \{(e-a), (f-c), (g-d)\}$$



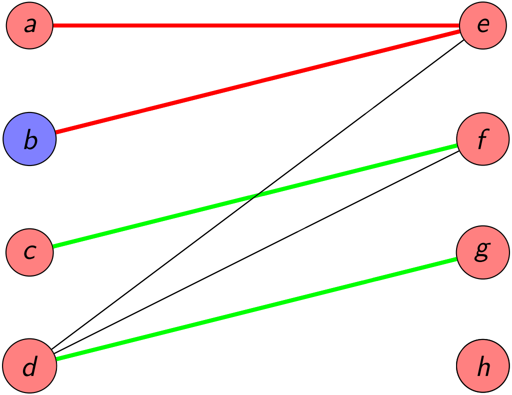
Second iteration:DFS

$F = \{H\}$
$P = \{(h - a)\}$
$M = \{(e - a), (f - c), (g - d)\}$



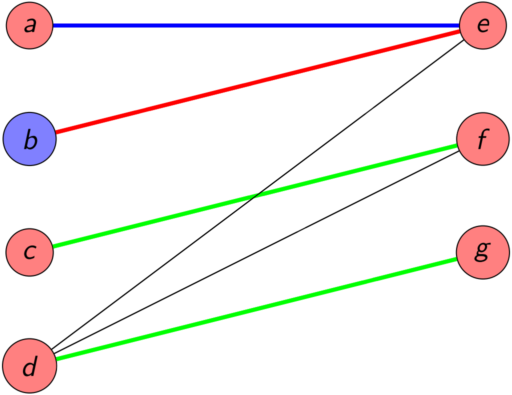
Second iteration:DFS

$F = \{H\}$
$P = \{(h - a)\}$
$M = \{(e - a), (f - c), (g - d)\}$



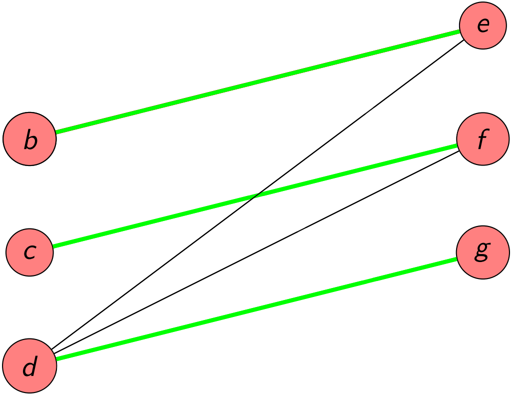
Second iteration:DFS

$F = \{\}$
$P = \{(h - a), (a - e)\}$
$M = \{(e - a), (f - c), (g - d)\}$



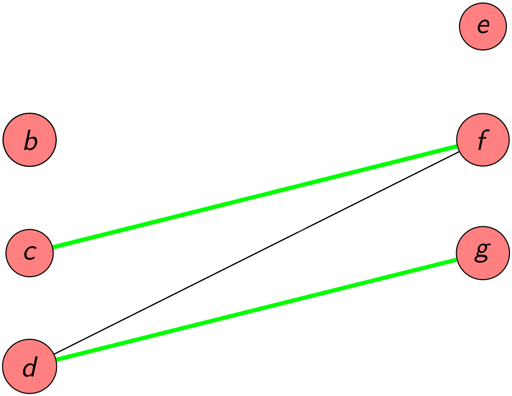
Second iteration:DFS

$$F = \{\}$$
$$P = \{(h - a), (e - b)\}$$
$$M = \{(f - c), (g - d)\}$$



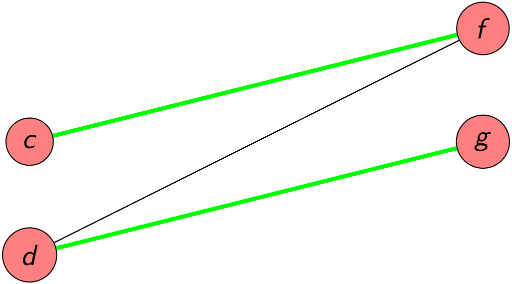
Second iteration:DFS

$$F = \{\}$$
$$P = \{(h - a), {\color{red}(e - b)}\}$$
$$M = \{(f - c), (g - d)\}$$



Second iteration:DFS

$$F = \{\}$$
$$P = \{(h - a), (e - b)\}$$
$$M = \{(f - c), (g - d)\}$$



Algorithm termination

As no more free vertex is available,the algorithm terminates and finally we get the following graph with maximum cardinaliy 4 where $M = \{(h - a), (e - b), (f - c), (g - d)\}$

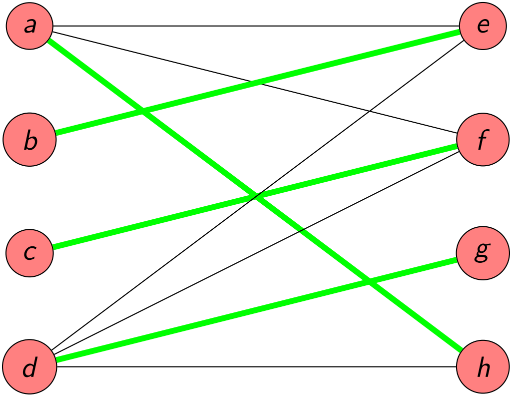


Table of Contents

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.