**Complete Matching Bipartite Graphs**

Diego Mackay, Sun Pak, Karan Dhanoa

University of the Fraser Valley

Comp 359 - ON1: Analysis and Design

Dr. Russell Campbell

Sept 20th, 2024

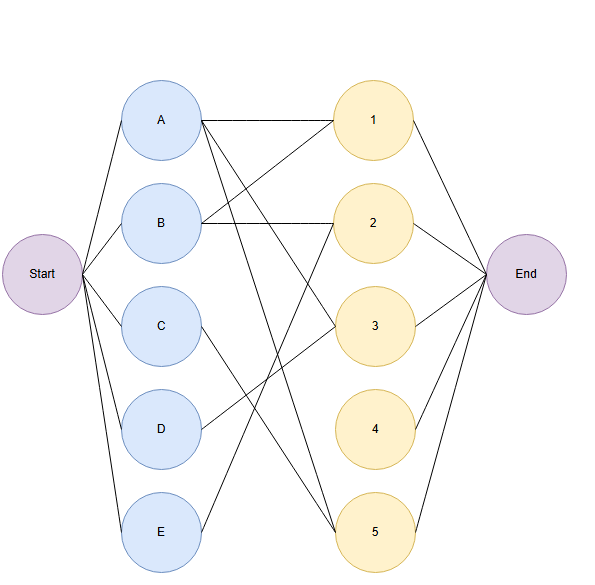
**Abstract**

A bipartite graph is a type of graph where the set of vertices can be divided into two disjoint sets. For example, if one group is called group X and the other as group Y, in this bipartite graph, the direction of all edges is from X to Y. This means that every edge in the graph connects a vertex in group X to a vertex in group Y. Each edge in the graph has a capacity of 1, meaning that each edge can only support one connection at a time. This structure is commonly used to represent relationships between two different types of entities, such as jobs and workers, or buyers and sellers.

**Introduction**

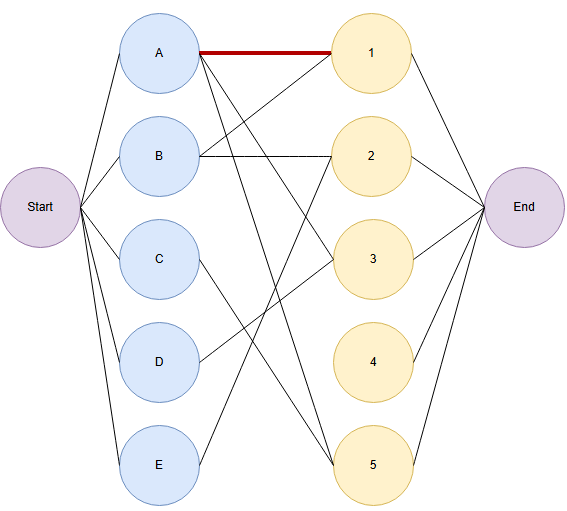
In a bipartite graph, edges always point from vertices in group X to vertices in group Y, though this direction is not always visually represented. The goal of bipartite matching is to find the maximum number of edges that can be selected such that no two edges share a common vertex, paring vertices from the two groups in a way that maximizes the number of matchings.

A matching in a graph refers to a set of edges where no two edges share a common vertex. In the context of bipartite graphs, a matching involves pairing vertices from group X with vertices from group Y, ensuring that each vertex from both groups is matched with at most one vertex from the other group. In this case, the capacity of 1 for each edge means that no vertex can be part of more than one matching simultaneously. When this is represented as a flowchart, bipartite matching can be understood as a network flow problem with edge capacity of 1. However, bipartite matching can be solved more efficiently using Depth-First Search(DFS) compared to using the Edmonds-Karp algorithm.



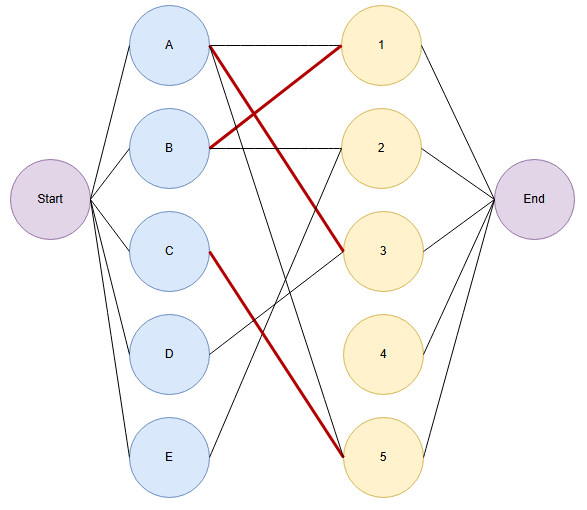
*Figure 1. Bipartite Graph Representation*

The direction of the edges begins from the left, and is matched to the right.

Vertex A, is matched with Vertex 1. (Maximum number of matches: 1)

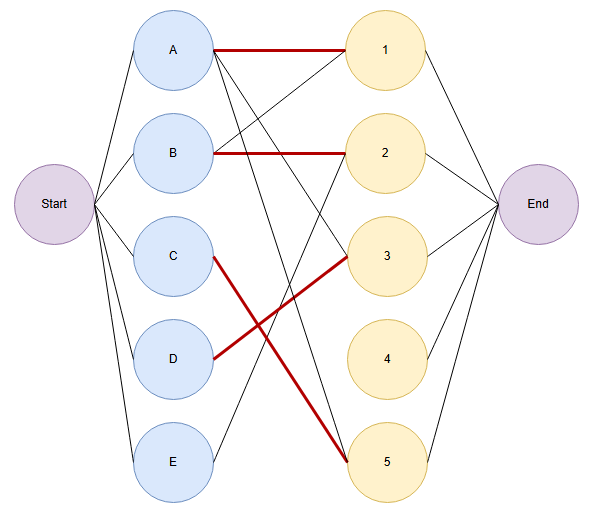
*Figure 2. Bipartite Matching Process*

Next, Vertex B tries to match with Vertex 1, but it has already been matched by Vertex A. Vertex A therefore must search for a new matching vertex. Vertex A matches with Vertex 3. (Maximum number of matches: 2). Afterwards, Vertex C matches with Vertex 5. (Maximum number of matches: 3)



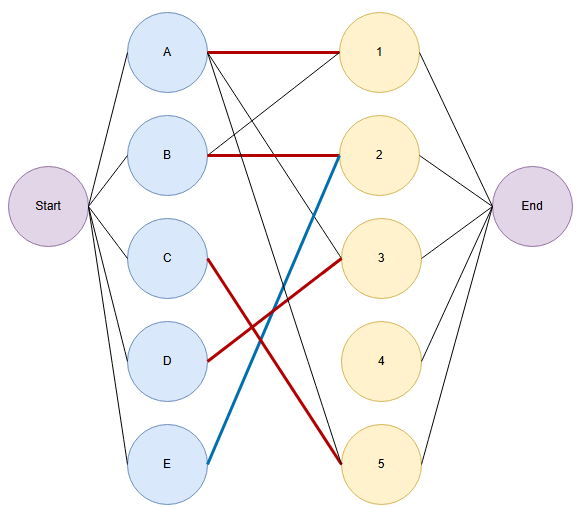
*Figure 3. Bipartite Matching Process*

Vertex D wants to match with vertex 3, but it is occupied. Vertex A however, has vertex 3, so A searches for a new direction and tries to match with vertex 1. Vertex 1, is also matched with vertex B, so B searches for a new direction and matches with vertex 2. (Maximum number of matches: 4)



*Figure 4. Bipartite Matching Process*

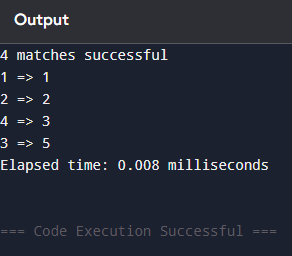
Last vertex E, wants to match with vertex 2. However, vertex E cannot match with 2, because if vertex E matches with vertex 2, B and A and D must all search for new directions. Therefore matching with vertex 2 is impossible for vertex E. The maximum number of matches remains as 4.



*Figure 5. Bipartite Matching Process*

Time complexity of the C++ algorithm, is O(VE) since bipartite matching algorithm’s time complexity is number of vertex \* number of directions.

The output executed below with C++ represents the theory of bipartite algorithms that is correctly applied to the implementation. Vertex A, in this case 1 is matched with 1, Vertex B(2) with 2. Vertex D(4) is matched with 3, and finally, Vertex C(3) with 5. E could not find a successful match, therefore the bipartite graph remains with maximum matches of four.

****

*Figure 6. Output of Bipartite Graph of C++*