CS300 - Fall 2024-2025 Sabancı University Homework #4

Introduction

Imagine you're working in a company with n workers and m different jobs or tasks that need to get done. You've just joined the company as a fresh graduate, and they've thrown you straight into the deep end. Your job is to figure out how to assign workers to tasks in the best way possible.

The catch? The company wants to make sure every worker is as busy as possible, so they don't waste any resources. Your task is to come up with an algorithm that assigns the maximum number of workers to individual tasks, helping the company get the most out of its team. To address this problem, you remembered taking the CS300 - Data Structures course, and you came up with the idea of representing workers and tasks in a graph data structure. In this approach, workers and tasks are represented as nodes, and edges represent the assignment of a specific worker to a task.

Just immediately, you notice that such a problem requires special case of graphs to begin with because if all workers and tasks are nodes of a graph, what happens if your algorithm incorrectly assumes one of the workers as a task and assigns a worker to a worker? That wouldn't make sense at all. To better understand the problem, you make a research and come up with the following.

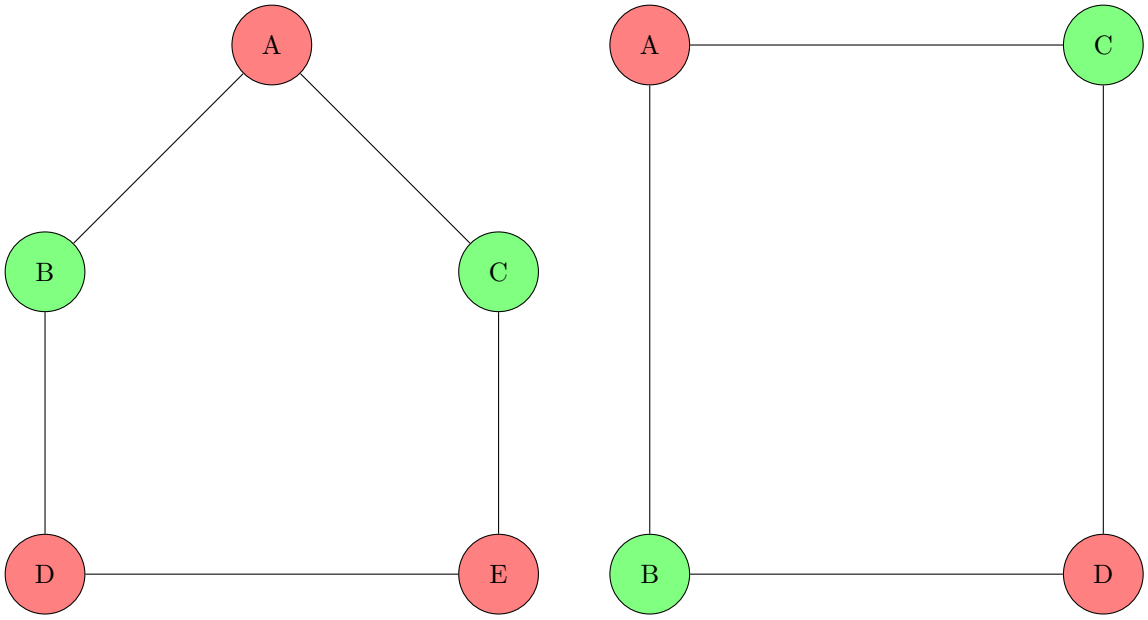
Bipartite Graphs

Bipartite graphs are a special type of graph in which the vertices can be divided into two disjoint and independent sets, G_1 and G_2. In this structure, there are no edges connecting any pair of vertices within the same set, meaning no edges exist between nodes in G_1 or between nodes in G_2. All edges connect vertices in G_1 to vertices in G_2. To express it in more formal way, we can define them as follows.
A graph $G = (V, E)$ is called a **bipartite graph** if its vertex set V can be partitioned into two disjoint subsets G_1 and G_2 such that:

1. $G_1 \cap G_2 = \emptyset$ and $G_1 \cup G_2 = V$
2. $\forall u, v \in G_1,\ (u, v) \notin E$
3. $\forall u, v \in G_2,\ (u, v) \notin E$
4. $\forall u \in G_1, \forall v \in G_2,\ (u, v) \in E$ or $(v, u) \in E$

You then observe that a bipartite graph can be colored using two colors in such a way that the neighbors of any vertex v are assigned a different color than v. For such a thing to ever exist, there should be no odd-numbered cycles in the graph.

Not Bipartite **Bipartite**



Your Task

Your task in this homework consist of 2 phases, they are as follows.

Phase 1

In this phase, your program will first check whether each input graph satisfies the properties of a bipartite graph. This check must be performed for every graph provided to your program before proceeding further. After completing this step, your program will output whether each graph is bipartite or not.

Phase 2

After determining that the graphs are bipartite, your program will attempt to maximize the number of matchings between the two sets U and V, or in our case between workers n and tasks m.

A matching in a bipartite graph is a subset of edges such that no two edges share a common vertex. In other words, each vertex in U is paired with at most one vertex in V, and vice versa. The goal is to find the maximum number of such pairings to optimize the assignment between the two sets. If given graph is bipartite and you detect its bipartite, then you should print the maximum number of matching.

Definition: Augmenting Path

To achieve the maximum matching, your program will rely on the concept of an augmenting path. An augmenting path is defined as follows:

A vertex that is not the endpoint of an edge in some partial matching M is called a **free vertex**. A path that starts at a free vertex, ends at another free vertex and throughout the path, alternates between `unmatched` and `matched`, then such paths are called **augmenting paths**.

According to this definition, all vertices in an augmenting path, except for the endpoints, must be **non-free vertices**. Additionally, an augmenting path can consist of only two free vertices connected by a single `unmatched` edge.

Using the concept of augmenting paths, your program should iteratively improve the current partial matching until no more augmenting paths can be found, thereby achieving the maximum matching. This is one way of finding the maximum matching. It is a very straightforward approach, and its time complexity is around $O(V \times E)$. If you can come up with better algorithms, you are more than welcome to use them, since some graphs can be large.

Important Remark

Due to the fact that node 0 is kept as a sentinel in the described algorithm, your code should not include it in the matching counting process. So if you find a matching that includes node 0, You should accept the answer as 1 less of that, or you can just ignore the node 0 in the traversal.

Program Flow

Your program will process input from text files named in the format `graph{i}.txt`, where i represents the graph number. For example, the files could be named `graph1.txt`, `graph2.txt`, etc. Each text file represents a graph in the Adjacency List format. They are structured as follows.

- The number of vertices (*first number*).
- The number of edges (*second number*).

Following these two numbers after first line, the file contains a series of rows, each with two numbers representing an edge. Each pair of numbers indicates the two vertices connected by that edge.

Example Input File (`graph0.txt`)(4 vertices 5 edges)

```
4 5
1 2
1 3
2 4
3 4
1 4
```

Explanation of the File Format

- The first line (`4`) specifies the number of vertices in the graph.
- The first line (`5`) specifies the number of edges in the graph.
- The following lines represent edges:
  - `1 2`: An edge between vertex 1 and vertex 2.
  - `1 3`: An edge between vertex 1 and vertex 3.
  - `2 4`: An edge between vertex 2 and vertex 4.

- `3 4`: An edge between vertex 3 and vertex 4.
- `1 4`: An edge between vertex 1 and vertex 4.

After reading this, you will check wether given graph is bipartite or not, and will print the following if its not bipartite: `The graph is not bipartite.`
If the given graph is bipartite, then your program will print 2 lines, first line stating that it is bipartite and second line stating the maximum number of matching.

```
The graph is bipartite.
Maximum matching size:  4
```

Moreover, your program will ask for input again and again untill the word `exit` is inputted, then your program will terminate.

Sample Runs

Here are some sample runs for the requested program:

Sample Run 1

```
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph0
The graph is not bipartite.
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph1
The graph is not bipartite.
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph2
The graph is bipartite.
Maximum matching size: 4
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): exit
```

Sample Run 2

```
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph1
The graph is not bipartite.
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph6
The graph is bipartite.
Maximum matching size: 49
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph5
The graph is not bipartite.
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): graph4
The graph is bipartite.
Maximum matching size: 5
Enter the graph name (e.g., 'graph1' or 'exit' to terminate): exit
```

Submission and Grading Guidelines

- Submissions will be done via SUcourse with CodeRunner in order to prevent compilation errors
- Ensure that this file is the latest version of your homework program.
- Submit via `SUCourse` ONLY.
- Submissions through e-mail or other means (e.g., paper) will receive no credit.

- Submit your own work, even if it is not fully functional. Submitting similar programs is easily detectable.

Good Luck!
CS300 Team