情報工学実験II

テーマ03
グラフ・ネットワークプログラム

令和５年０７月０６日

イマム　カイリ　ルビス

学籍番号：214071

目 次

1 概要 **3**
1.1 グラフ理論とは 3
1.2 スタックとは 3
1.3 キューとは 3
1.3.1 リングバッファによるキュー 3
1.4 実行環境 4

2 深さ優先検索と幅優先検索を用いて検索 **4**
2.1 深さ優先検索 4
2.1.1 深さ優先検索のプログラム 4
2.1.2 深さ優先検索のプログラムの動作 4
2.2 幅優先検索 4
2.2.1 幅優先検索のプログラム 5
2.2.2 深さ優先検索のプログラムの動作 5
2.3 深さ優先検索と幅優先検索の結果 5
2.4 深さ優先検索と幅優先検索の考察 5

3 連結成分数 **5**
3.1 連結成分数のプログラム 5
3.2 連結成分数のプログラムの動作 6
3.3 連結成分数のプログラムの実行結果 6
3.4 連結成分数の結果の考察 6

4 最大クリーク問題 **6**
4.1 最大クリークのプログラム 6
4.2 最大クリークのプログラムの動作 6
4.3 最大クリークの結果 7
4.4 最大クリークの考察 7

5 発表者の感想 **7**

1　概要

1.1　グラフ理論とは

数学においてグラフ理論とは、グラフを研究する学問であり、グラフはオブジェクト間の対関係をモデル化するために用いられる数学的構造である。グラフを構成するためには、点（節点またはノードとも呼ばれる）と辺（枝またはエッジとも呼ばれる）が必要である [?]。グラフ理論には、辺が方向を持っているかどうかによって分れている

有向グラフ　：辺の方向が決まっている一方向性のグラフである。

無向グラフ　：辺が特定の方向を持たず、双方向性を持つグラフである。

本実験では、使用したグラフはすべて無向グラフである。更に、

1.2　スタックとは

スタックは、データを一時的にた蓄えるためのデータ構造の一つ。データの出し入れは**後入れ先出し**（*LIFO / Last In First Out*）で行われる。すなわち、最後に入れられたデータが最初に取り出される [?]。

なお、スタックにデータを入れる操作を**プッシュ**（*push*）と呼び、スタックからデータを取り出す操作を**ポップ**（*pop*）と呼びます [?]。

しかし本実験では、実際のスタック機能を模倣するため、ノード数分の大きさを持つ配列を使ってスタックデータ構造を作った。

1.3　キューとは

キューは、データを一時的に蓄えるための基本的なデータ構造の一つである。最初に入れられたデータが最初に取り出されるという**先入れ先出し**（*FIFO / First In First Out*）の機構である。[?]

なお、キューにデータを追加する操作を**エンキュー**（*enqueue*）と呼び、データを取り出す操作を**デキュー**（*dequeue*）と呼ぶ。また、データが取り出される側を**先頭**（*front*）と呼び、データが押し込まれる側を**末尾**（*rear*）と呼ぶ [?]。

しかし本実験では、実際のキュー機能を模倣するため、ノード数分の大きさを持つ配列を使ってスタックデータ構造を作った。

1.3.1　リングバッファによるキュー

リングバッファとは、配列の末尾が先頭につながっているとみなすデータ構造である [?]。エンキューとデキューを行うと *front* と *rear* の値は変化する。

1.4 実行環境

本実験で使用される実行環境：

- プロセッサ：AMD Ryzen 5 5600X
- メモリー：16.0 GB
- OS：Windows 11 Pro
- コンパイラ：gcc

2 深さ優先検索と幅優先検索を用いて検索

2.1 深さ優先検索

深さ優先探索は、木やグラフのデータ構造を探索するアルゴリズムである。このアルゴリズムは、根（始点）から開始し、バックトラックする前に各辺に沿って可能な限り探索する。

指定した辺に沿ってこれまでに発見されたノードを追跡し、グラフのバックトラックに役立てるために、スタックが必要となる。

2.1.1 深さ優先検索のプログラム

以下は深さ優先検索のプログラムである.

2.1.2 深さ優先検索のプログラムの動作

訪問したすべての点はスタックにプッシュされ、その点から先に行けない場合はスタックトップがポップされる。深さ優先検索のプログラムの主な流れは，以下の通りである：

1. 根をスタックにプッシュする。
2. スタックトップのデータを現在点になるようにピークする。
3. 現在の点に隣接している最も低い点に訪問する。
4. 現在点から行き場所はない場合、スタックからポップする。
5. 全ての点を訪問されるまで繰り返す。

2.2 幅優先検索

幅優先探索は、木やグラフのデータ構造を探索するアルゴリズムである。このアルゴリズムは、根（始点）から開始し、各点に隣接している点を訪問する。

キューは、訪問されたがまだ探索されていない点を追跡するために必要である。

2.2.1 幅優先検索のプログラム

以下は深さ優先検索のプログラムである.

2.2.2 深さ優先検索のプログラムの動作

訪問したすべての点はキューの *rear* にエンキューされ、その点から先に行けない場合はキューの *front* からデキューされる。幅優先検索のプログラムの主な流れは，以下の通りである：

1. 根をキューにエンキューする。
2. キューの *front* のデータを現在点になるようにピークする。
3. 現在の点に隣接している全ての点を訪問する。
4. 現在点から行き場所はない場合、キューからデキューする。
5. 全ての点を訪れるまで繰り返す。

2.3 深さ優先検索と幅優先検索の結果

この問題では、検索対象となるデータが複数用意されている。両プログラムを実行した結果、下表のような結果が得られました。

2.4 深さ優先検索と幅優先検索の考察

表を見ればわかるように、どちらのアルゴリズムも木を生成するが、その特性は異なる。すべてをまとめるために、この表の特徴を以下に記す。

結論として、深さ優先検索は木が長くなるが、各点の子数は少なくなる。一方、BFS はツリーは短くなりますが、各点の子数は多くなります。

3 連結成分数

連結成分とは、点の各対が辺で結ばれている部分グラフのことである。この問題では、与えられたデータがどれだけの連結成分を持つかを数えるというものである。

3.1 連結成分数のプログラム

この問題に対して、新しいプログラムはないので、プログラムのある部分だけに焦点を当てるつもりである。

3.2　連結成分数のプログラムの動作

検索問題でも同じプログラムを利用する。このデータには複数の連結要素があるので、スタックやキューが空になるが、この処理はまだ終わっていない。なので、未訪問の点を追加しなければならない。両アルゴリズムの主な流れは，以下の通りである：

1. データ構造が空になるまでプログラムを実行する
2. すべての点を訪問したかどうかをチェックする。
3. そうでない場合は、最下位の未訪問点をデータ構造に追加する。
4. すべての点を訪問するまで繰り返す

言い換えれば、連結成分数を数えるには、この処理が何回繰り返されたかを数えればいい。

3.3　連結成分数のプログラムの実行結果

これらのプログラムを実行した結果、以下のような結果が得られた。

3.4　連結成分数の結果の考察

表からわかるように、どのようなアルゴリズムで連結成分をチェックしても結果は同じである。

4　最大クリーク問題

Clique—a subgraph is a graph in which every vertex is connected by an edge to any other vertex in the subgraph; a maximal clique is a clique that cannot be extended by including an additional adjacent vertex; in other words a maximal clique is a clique that is not a subset of a larger clique.

4.1　最大クリークのプログラム

以下は深さ優先検索のプログラムである.

4.2　最大クリークのプログラムの動作

This code actually an implementation of bron kerbosch algorithm without pivoting. But there are several things that i modified to make this code. Since this algorithm utilize intersection and union of set, instead of storing all of the data as an actual number, I choose to use an array with the index represent the data it self. Which means 0 is not included and 1 is included. The algorithm it self runs like this:

4.3 最大クリークの結果

In this problem we alse prepared with some set of data. After executing this program to all of the data, we got result like below

4.4 最大クリークの考察

As we can see the more node the data has, the longer this process take. That is why this algorithm can be improve by choosing pivot in each iteration. The pivot is chosen to minimize the number of recursive calls made by the algorithm, the savings in running time compared to the non-pivoting version of the algorithm can be significant [**?**].

5 発表者の感想

Ntar ini mah