アルゴリズムとデータ構造II Ex03 DFS・BFS

深さ優先探索(Depth First Search)と幅優先探索(Breadth First Search)でグラフを探索し全域木を作る方法を理解することが目的である。しかし、これはアルゴリズムとデータ構造Iで既出なので、アルゴリズムの面で言えば実質「復習回」になる。

とは言え、忘れている部分も多いと思うので、ここでプログラムを書いたりして思い出して欲しい。

- 提出課題はプログラムを書かなくても手計算で求めることができるが、手計算で求めたものと、プログラムで求めたものが一致するのかの確認をして欲しい。
- 提出課題では以下の1.と2.だけで済む。
- 3.はDFSを再帰を使わずに実装する方法である。
- 4.は隣接リストを使ったDFSの実装になる。
    - 次回以降の演習で隣接リストを使ったプログラムを扱うので、4.も別途取り組むように。

1. 深さ優先探索DFS(再帰)

スケルトンコード`dfs.c`は再帰によって深さ優先探索(DFS)を行うプログラムである。

探索開始前に入力として

1. データ数
2. 隣接行列の入力
3. 探索開始ノード

の順番で、標準入力を行なっている。

変数`A[][]`は隣接行列、`visited[]`は探索済みかどうかのフラグを表してる。

「そのノードはまだ探索していないか」どうかの判定条件は空欄にしてあるので、そこを埋めてプログラムを実行し、DFSアルゴリズムの動作を理解すること。

- グラフ上の頂点番号は$1 \sim n$としているが、プログラムでは配列の添字に合わせて$0 \sim n-1$となっているので注意すること。

2. 幅優先探索BFS

スケルトンコード`bfs.c`は幅優先探索(BFS)を行うプログラムである。

BFSは探索方針の性質上、再帰を使うことができないので、キューやスタックを使うことになる。

`bfs.c`はキューを用いた実装になっている。配列`Q[]`がキューを表している。

探索したノードを追加していき、現在、頂点をどこまで辿っていたかを記憶する。

- キューはFIFOの構造であることがポイント
  - スケルトンコード中に定義された関数プロトタイプ
    - `void queue_init()`・・・キューを初期化
    - `void enqueue(int)`・・・指定したノードをキューに入れる
    - `int dequeue()`・・・キューの先頭ノードを取り出す
    - `int ifqueue_empty()`・・・キューが空かどうか

3. 深さ優先探索DFS(非再帰)

1.で再帰を使った深さ優先探索をやったが、再帰を使わなくても、スタックを用いて深さ優先探索の実装はできる。スタックはFILOである。

スケルトンコード`dfs_nr.c`

int型変数`pointer`を使って、`stack[]`を操作している。

- スケルトンコード中に定義された関数プロトタイプ
  - `void Visit(int)`・・・非再帰的なDFSを行う関数本体
  - `void push(int)`・・・スタックに指定したノードを積む
  - `int pop()`・・・スタックの先頭ノードを取り出す
  - `int top()`・・・スタックの先頭ノードを調べる
  - `void stack_init()`・・・スタックを初期化する
  - `int ifstack_empty()`・・・スタックが空かどうか

4. 深さ優先探索DFS(隣接リスト)

スケルトンコード`dfs_list.c`はグラフの頂点間の接続状況を隣接行列ではなく、連結リストによる隣接リストでDFSを実装したものである。

隣接行列で表現するためには2次元配列を用いて$|V| \times |V|$個の値を保存する必要があるが、多くの場合、隣接行列は**疎行列**(行列の成分の多くが0)になっていることが多い。

0は加法における単位元($a + 0 = 0 + a = a$)であり、乗法については$0 \cdot a = a \cdot 0 = 0$という性質があるため、通常の演算についても、今回のアルゴリズムの挙動においても、自明で無価値な処理を繰り返し、時間や記憶領域のロスになる。

実際の科学技術計算での疎行列を扱う場合においても、そのまま2次元行列に格納することはせず、何らかのアルゴリズムによって0ではない要素のみを保持して計算することが多い。

隣接リストでグラフを表現すると、ある頂点に対して隣接している頂点のみを記憶するため、頂点数$|V|$が少ない場合は、記憶領域を節約でき、処理時間の短縮に繋がる。

1.のDFSと比べると、隣接リストでは「始点の頂点から繋がっている頂点のみ」がリストに記録されているので、リストを順に辿って出てきた番号の頂点がすでに訪問済みかどうかだけを調べれば良いことになる。

Prog Cでも連結リストをやっているが、忘れている人も多いと思う。連結リストに関わる部分の処理や変数・関数の名前はできるだけProg Cの内容に合わせているが、複数の連結リストを同時に扱うため一部異なる部分もある。

- 隣接リストの構成について(main関数部分)
    1. 頂点数と同じだけ、連結リストの先頭ノード(heads)の領域を確保する(malloc)
    2. i番目の頂点から接続する頂点のリストの先頭ノード(heads[i])を作成する
    3. 入力された隣接行列を読み取り、0なら接続していないので無視、1(あるいはそれ以外)ならば接続しているので、頂点番号をリストへ加える。
        - 頂点数の分だけ上記を繰り返す。

スケルトンコードは上記にも書いてあるように、`dfs_list.c`である。次回以降の演習でこの隣接リストを使うので、取り組むように。

実行例

6×6の隣接行列を隣接リストで保持し、DFSで頂点1から探索した結果を示すので、作ったプログラムの確認に使うと良い。

入力データは`6by6matrix.txt`

```
$ ./a.out < 6by6matrix.txt
Input the number of data: 6
Input the Adjacency matrix:
0 1 0 1 1 0
0 0 0 1 0 1
0 1 0 0 0 0
1 0 0 0 0 0
0 1 0 0 0 1
1 0 1 0 1 0
V1  is adjacent to 2  4  5
V2  is adjacent to 4  6
V3  is adjacent to 2
V4  is adjacent to 1
V5  is adjacent to 2  6
V6  is adjacent to 1  3  5
Input the initial node: 1
The edges in the DFS tree with root 1 are:
( 1, 2)
( 2, 4)
```

```
( 2, 6)
( 6, 3)
( 6, 5)
```