MICS 実験第一 J8 課題レポート

学籍番号 2210342, 鈴木謙太郎

2024 年 4 月 15 日

1 課題 1

実験資料を参考に, コード 1 のように remove_edge 関数及び reorient_edge 関数を実装した.

Code 1: 実装した remove_edge 関数及び reorient_edge 関数のソースコード

```
1  void remove_edge(graph *g, int x, int y) {
2    if (is_edge(g, x, y)) {
3      int i;
4      for (i = 0; i < g->degree[x]; i++) {
5        // is_edge(g, x, y)が真であるので,g->degree[x][i] == yとなるiは必ず存在
6        // よって必ずbreakするしこのfor-loopは安全
7        if (g->edges[x][i] == y) {
8          break;
9        }
10     }
11     for (; i < g->degree[x] - 1; i++) {
12       g->edges[x][i] = g->edges[x][i + 1];
13     }
14     g->degree[x]--;
15     g->nedges--;
16     return;
17   }
18   fprintf(stderr, "Warning: (%d, %d) does not exist\n", x, y);
19 }
20
21 void reorient_edge(graph *g, int x, int y) {
22   if (is_edge(g, x, y)) {
23     remove_edge(g, x, y);
24     insert_edge(g, y, x);
25   } else {
26     fprintf(stderr, "Warning: (%d, %d) does not exist\n", x, y);
27   }
```

```
28    return;
29 }
```

また, 配布されたテストプログラムを用いてこれらの関数を含む処理を実行した結果は以下のようになった.

```
> ./kadai1main kadai1/input.txt
0: 1 2
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 2
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
degree[7]: 0
nedges = 11
***** insert an edge (input two numbers):
0 3
0: 1 2 3
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 3
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
```

```
degree[7]: 0
nedges = 12
***** remove an edge (input two numbers):
0 1
0: 2 3
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 2
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
degree[7]: 0
nedges = 11
***** reorient an edge (input two numbers):
0 2
0: 3
1: 3 5
2: 3 4 0
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 1
degree[1]: 2
degree[2]: 3
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
degree[7]: 0
nedges = 11
```

この操作では,0 − 3 辺を追加したあと 0 − 1 辺を削除し,0 − 2 辺を 2 − 0 辺に入れ替えている. その後の出力から, これらの操作が正しく完了していることがわかる.

次に, 異常系の入出力を試した結果は次のようになった.

```
> ./kadai1main kadai1/input.txt
0: 1 2
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 2
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
degree[7]: 0
nedges = 11
***** insert an edge (input two numbers):
0 1
Warning: (0, 1) already exists, no insertion is performed
0: 1 2
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 2
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
```

```
degree[7]: 0
nedges = 11
***** remove an edge (input two numbers):
0 3
Warning: (0, 3) does not exist
0: 1 2
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 2
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
degree[7]: 0
nedges = 11
***** reorient an edge (input two numbers):
0 3
Warning: (0, 3) does not exist
0: 1 2
1: 3 5
2: 3 4
3: 4
4: 5
5: 7
6: 4 5
7:
degree[0]: 2
degree[1]: 2
degree[2]: 2
degree[3]: 1
degree[4]: 1
degree[5]: 1
degree[6]: 2
```

```
degree[7]: 0
nedges = 11
```

この操作で, 既に存在する辺を挿入しようとする・存在しない辺を削除しようとする・存在しない辺を入れ替えようとする場合にエラーが出力され, 何も操作が行われないことが確認できた.

これら 2 つの検証によって,remove_edge 関数及び reorient_edge 関数が正しく実装されていることが確認できた.

2 課題 2

実験資料を参考に, コード 2 のように dfs 関数を実装した.

Code 2: 実装した dfs 関数のソースコード

```
1  // 資料P.16
2  void dfs(graph *g, dfs_info *d_i, int start) {
3    // startを訪問済みとする
4    d_i->visited[start] = 1;
5    for (int i = 0; i < g->degree[start]; i++) {
6      // まだ訪問していない頂点のみを対象とする
7      if (d_i->visited[g->edges[start][i]] == 0) {
8        // 再帰的にDFSを行う
9        d_i->predecessor[g->edges[start][i]] = start;
10       dfs(g, d_i, g->edges[start][i]);
11     }
12   }
13   return;
14 }
```

また, 配布された入力例 graph2_input.txt を用いて,dfs 関数をテストした結果は次のようになった.

```
> ./kadai2main kadai2/graph2_input.txt
0: predecessor[0] = -1
1: predecessor[1] = 19
2: predecessor[2] = 0
3: predecessor[3] = 4
4: predecessor[4] = 0
5: predecessor[5] = 11
6: predecessor[6] = 10
7: predecessor[7] = 15
8: predecessor[8] = 3
9: predecessor[9] = 3
10: predecessor[10] = 18
```

```
11: predecessor[11] = 12
12: predecessor[12] = 3
13: predecessor[13] = 2
14: predecessor[14] = 13
15: predecessor[15] = 10
16: predecessor[16] = 11
17: predecessor[17] = -1
18: predecessor[18] = 12
19: predecessor[19] = 5
0: visited[0] = 1
1: visited[1] = 1
2: visited[2] = 1
3: visited[3] = 1
4: visited[4] = 1
5: visited[5] = 1
6: visited[6] = 1
7: visited[7] = 1
8: visited[8] = 1
9: visited[9] = 1
10: visited[10] = 1
11: visited[11] = 1
12: visited[12] = 1
13: visited[13] = 1
14: visited[14] = 1
15: visited[15] = 1
16: visited[16] = 1
17: visited[17] = 0
18: visited[18] = 1
19: visited[19] = 1
```

このうち visited を出力している部分のみをテキストファイルに挿入し,graph2_output.txt との間に相違ないことを diff コマンドで確認した.

同様に他のすべての入力例でも出力例との間に相違ないことを確認できたので, この dfs 関数の実装に問題はないと判断した.

3 課題 3