MICS 実験第一 J8 課題レポート

学籍番号 2210342, 鈴木謙太郎

2024 年 5 月 8 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

実験資料を参考に, コード 3 のように dfs 関数を実装した.

Code 3: 実装した関数のソースコード

```
1
```

```
2  void augment(graph *g, dfs_info *d_i, graph *matching, int start, int end) {
3    int v = end;
4    int p = d_i->predecessor[v];
5
6    // 増加道の逆方向にたどる
7    while (v != start) {
8      reorient_edge(g, p, v);
9      v = p;
10     p = d_i->predecessor[v];
11
12     if ((!is_edge(matching, p, v)) && !is_edge(matching, v, p)) {
13       insert_edge(matching, p, v);
14     } else if (is_edge(matching, p, v)) {
15       remove_edge(matching, v, p);
16     }
17   }
18   return;
19 }
20
21 int find_maximum_matching(graph *g, graph *matching) {
22   int size = 0; /* the size of a current matching */
23   int source, sink;
24   dfs_info *d_i;
25
26   source = g->nvertices - 2;
27   sink = g->nvertices - 1;
28   d_i = (dfs_info *)malloc(sizeof(dfs_info));
29
30   initialize_search(g, d_i);
31   dfs(g, d_i, source);
32
33   while (d_i->visited[sink] == 1) {
34     augment(g, d_i, matching, source, sink);
35     size++;
36     initialize_search(g, d_i);
37     dfs(g, d_i, source);
38   }
39
40   free(d_i);  // メモリ解放
41   return size;
42 }
```

また, 配布された入力例の中から bigraph0_input.txt,bigraph2_input.txt,bigraph4_input.txt を用いて, 実装した関数をテストした結果はそれぞれ次のようになった.

```
> ./a.out kadai3/bigraph0_input.txt
Warning: (10, -1) does not exist
Warning: (10, -1) does not exist
Warning: (10, -1) does not exist
Warning: (10, -1) does not exist
the maximum matching size = 5
```

```
> ./a.out kadai3/bigraph2_input.txt
Warning: (15, -1) does not exist
Warning: (15, -1) does not exist
Warning: (15, -1) does not exist
Warning: (15, -1) does not exist
the maximum matching size = 5
```

```
> ./a.out kadai3/bigraph4_input.txt
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (20, 5) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
Warning: (25, 6) does not exist
Warning: (40, -1) does not exist
Warning: (28, 2) does not exist
Warning: (40, -1) does not exist
Warning: (29, 14) does not exist
Warning: (40, -1) does not exist
Warning: (40, -1) does not exist
the maximum matching size = 18
```

いずれも, 期待される出力と最大マッチングのサイズが一致しているが,augment 関数内の reorient_edge

または remove_edge において Warning が発生していた. 今回原因を突き止めることはできなかったが, その分すべての入力例に対して追加でテストを行った. その結果最大マッチングのサイズは正しく出力されていたので,2 つの関数の実装に大きな間違いはないと判断した.

4 課題 4

まず,randgen.c に適当なパラメータを入力していくつかの入力を生成し, それを用いて実行時間についての傾向を調べた. このとき利用したパラメータとその実行結果は表 1 の通りである.

表 1: randgen.c のパラメータと実行結果

集合 X の要素数	集合 Y の要素数	辺の存在確率	生成されたグラフの辺の数	実行 CPU 時間
40	20	0.4	341	0.002
40	20	0.8	330	0.002
40	40	0.4	290	0.002
160	20	0.4	1261	0.002
160	40	0.4	2615	0.002
160	60	0.4	3857	0.003
160	160	0.4	10231	0.010
160	160	0.8	20470	0.022
320	320	0.4	40839	0.059
320	320	0.8	81968	0.124

この結果から生成されたグラフの辺の数と実行 CPU 時間に正の相関があると考え, x 軸を辺の数,y 軸を実行 CPU 時間としたプロットを作成すると図 1 のようになった.

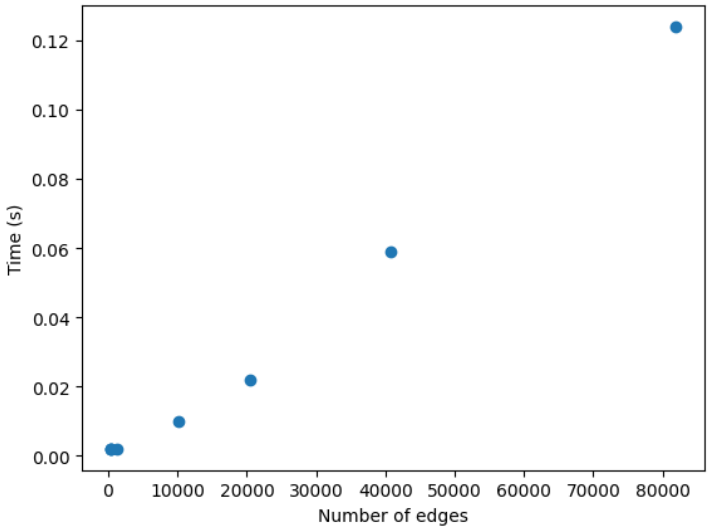

図 1: randgen.c のパラメータと実行結果のプロット

この図から, グラフの辺の数がある程度多くなると,CPU 時間がほぼ線形に増加しているように見える. これは, 増加道を見つけてそれらを反転させる操作を行う回数が, グラフの辺の数が増えていくほど増加するためであると考えられる.

また, 辺の数が少ない間はほとんど CPU 時間に差が見られないので, グラフの辺の数がある程度小さい間はファイルの中身を読み取ったりグラフを構築したりする操作にかかる時間の影響が大きいと考えられる.d