計算機科学実験及演習 1 報告書
課題 12

安済 翔真

提出日: 2022 年 6 月 29 日

1　最短経路探索アルゴリズム

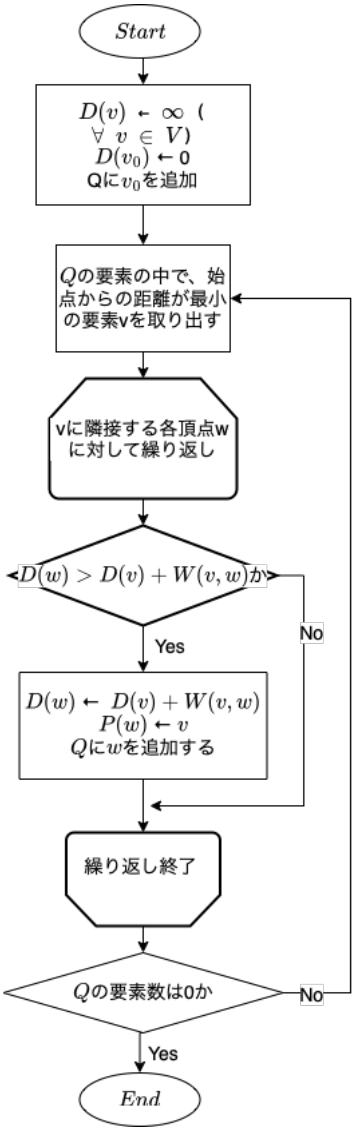
1.1　アルゴリズムの説明

以下に、実装したダイクストラ法のアルゴリズムを示す。

1. 距離が確定した頂点の集合 A、始点から頂点 v までの距離を返す関数 D(v)、最短経路において頂点 v の 1 つ前に通る頂点を返す関数 P(v)、少なくとも 1 回訪れた頂点の集合 Q、頂点 v, w 間の辺の重みを返す関数 $W(v, w)$ を用意する。はじめは $\forall v \in V$(頂点全体の集合), $D(v) = \infty$ としておく。
2. 始点 v_0 からの距離 $D(v_0) = 0$ とする。また、Q に v_0 を追加する。
3. Q の要素数が 0 になるまで以下を繰り返す。
   (a) Q の要素の中で、始点からの距離が最小の要素 v を取り出す。
   (b) v に隣接する各頂点 w に対して、$D(w) > D(v)+W(v, w)$ ならば $D(w) = D(v)+W(v, w)$ とし、Q に w を追加する。また、$P(w) = v$ とする。

以上の操作によって定められた $P(v)$ の値を辿ることで始点から任意の点までの最短経路を求めることができる。

1.2　アルゴリズムの流れ図



1.3　実行例

```
$ java MyGraphTest
graph.txt
3
```

2　クラス仕様

この章では実装したクラスの仕様を説明する。

2.1 MyEdge クラス

2.1.1 役割

このクラスはグラフ内の辺を表す。辺の両端の頂点と辺の重みを変数として保持し、このクラスのインスタンスメソッドを呼び出すことでそれらを取得することができる。

2.1.2 メンバ変数

Integer node1 辺の両端の頂点のうち、頂点番号が小さい方の頂点の番号を保持する。

Integer node2 辺の両端の頂点のうち、頂点番号が大きい方の頂点の番号を保持する。

Integer weight 辺の重みを保持する。

2.1.3 getNodes メソッド

機能 辺の両端の頂点番号を配列として返す。

インタフェース

1. 引数: なし
2. 返り値の型: Integer[]

2.1.4 getWeight

機能 辺の重みを返す。

インタフェース

1. 引数: なし
2. 返り値の型: Integer[]

2.2 MyGraph クラス

2.2.1 役割

このクラスはグラフそのものを表す。入力ファイルからグラフを読み取って辺をリストとして保持する。グラフに対する種々の操作を行うためのメソッドが用意されている。

2.2.2 メンバ変数

final int MAX_NODES_NUM 許容可能な頂点の個数の最大値を表す。

final int MAX_EDGES_NUM 許容可能な辺の個数の最大値を表す。

final integer MAX_WEIGHT 許容可能な辺の重みの最大値を表す。

final integer MIN_WEIGHT 許容可能な辺の重みの最小値を表す。

LinkedList<MyEdge>[ ] 許容可能な辺の重みの最小値を表す。

2.2.3 readFromFile メソッド

機能 入力ファイルからグラフを読み取って辺をリストとして保持する

インタフェース

1. 引数: String filename
2. 返り値: なし

2.2.4 getEdges メソッド

機能 引数に与えられた頂点につながる辺のリストを返す。

インタフェース

1. 引数: int id
2. 返り値の型: LinkedList<MyEdge>

2.2.5 getLinkedNodes メソッド

機能 引数に与えられた頂点につながる頂点の配列を返す。

インタフェース

1. 引数: int id
2. 返り値の型: Integer[]

2.2.6 getWeight メソッド

機能 引数に与えられた 2 つの頂点の間の辺の重みを返す。

インタフェース

1. 引数: Integer v, Integer w
2. 返り値の型: Integer

2.2.7 getNodeSize メソッド

機能 グラフ内の頂点の個数を返す。

インタフェース

1. 引数: なし
2. 返り値の型: int

2.2.8 getShortestPath メソッド

機能 ダイクストラ法により始点から終点までの最短経路を求める。返り値には、始点から拡張点に至るまでのパスにおいて、各頂点の１つ前の頂点番号が格納された配列が返される。

インタフェース

1. 引数: Integer start, Integer end
2. 返り値の型: Integer[]

2.3 Node クラス

2.3.1 役割

このクラスは頂点そのものを表す。ダイクストラ法において、すでに訪れた頂点の集合 $found$ の中から始点からの距離が最も小さい頂点を取り出す際、通常の配列を用いると $O(n)$ かかってしまうため、$PriorityQueue$ を用いて実装をした。ヒープは始点からの距離を基準として構成する必要があるため、頂点の大小を始点からの距離で測るよう $compareTo$ メソッドが $Override$ して実装されている。

2.3.2 メンバ変数

Integer id 頂点番号を保持する。

Integer distance 始点からの距離を保持する。

2.3.3 compareTo メソッド

機能 頂点の順序関係を表す。自身と引数で与えられた頂点を比べて、始点からの距離が自身の方が小さければ負の値を、大きければ正の値を返し、等しければ 0 を返す。

インタフェース

1. 引数: Node node
2. 返り値の型: int

2.4 MyGraphTest クラス

2.4.1 役割

このクラスは、MyGraph クラスの getShortestPath メソッドをテストするのに用いる。

2.4.2 メンバ変数

なし。

2.4.3 main メソッド

機能 MyGraph クラスの *getShortestPath* をテストする。標準入力から、入力ファイルの名前と始点・終点の頂点番号を受け取り、始点から終点までの最短経路とそれにかかるコストを標準出力に出力する。

インタフェース

1. 引数: String[] args
2. 返り値の型: なし

3 プログラムの評価

本プログラムにおいて工夫した点は、すでに訪れた頂点の集合を表す $found$ を $PriorityQueue$ で表すことで、$found$ の中で始点からの距離が最も小さい頂点を取り出すのにかかる時間を $O(\log n)$ に抑えたことである。ヒープは始点からの距離を基準として構成する必要があるため、Node クラスを用意し、*compareTo* メソッドを Override することで頂点の順序関係を自身で作成した。これによりプログラム全体の計算量は $O(n \log n)$ (n : 頂点の個数) 程度に抑えられている。テストにおいては、連結グラフにおいて正しくパスが出力されること・入力ファイルに不適切な文字が入っていた場合に正常に例外処理をしてプログラムを終了すること・始点と終点が繋がっていなかった場合に正常に終了することを確認した。

4 プログラム開発の経過

1. 問題の分析と解法の検討
   課題 12 の wiki を読んで要求仕様を確認した。この段階には、ダイクストラ法のアルゴリズムの確認などを含めて 1 時間ほど費やした。
2. クラス設計
   PriorityQueue を用いるのに必要な Node クラスの設計などに時間がかかり、2 時間ほど費やした。
3. クラス内論理設計
   約 1 時間。
4. プログラムテスト・デバッグ
   約 30 分。
5. 仕様書の作成
   約 5 時間。

5　感想

入力ファイルに不適切な文字が含まれていた場合の例外処理を考えるのに苦労した。また、計算量を抑えるために、$found$ を $PriorityQueue$ で表すために、Node クラスを作成して $compareTo$ メソッドを Override するというのが難しく、時間がかかった。グラフの表し方は他にもあるそうなので、別の表し方でプログラムを書くことも試してみたいと感じた。

付録

ソースコード 1: MyEdge.java

```
1  public class MyEdge {
2      private Integer node1;
3      private Integer node2;
4      private Integer weight;
5
6      public MyEdge(Integer node1, Integer node2, Integer weight) {
7          this.node1 = Math.min(node1, node2);
8          this.node2 = Math.max(node1, node2);
9          this.weight = weight;
10     }
11
12     public Integer[] getNodes() {
13         Integer[] nodes = { this.node1, this.node2 };
14         return nodes;
15     }
16
17     public Integer getWeight() {
18         return this.weight;
19     }
20 }
```

ソースコード 2: Node.java

```
1  public class Node implements Comparable<Node> {
2      Integer id;
3      Integer distance;
4
5      public Node(Integer id, Integer distance) {
6          this.id = id;
7          this.distance = distance;
8      }
9
10     @Override
11     public int compareTo(Node node) {
12         return this.distance - node.distance;
13     }
14 }
```

ソースコード 3: MyGraph.java

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Arrays;
4  import java.util.LinkedList;
5  import java.util.PriorityQueue;
6  import java.util.Scanner;
```

```
7
8  public class MyGraph {
9      private final int MAX_NODES_NUM = 50;
10     private final int MAX_EDGES_NUM = 100;
11     private final Integer MAX_WEIGHT = 9999;
12     private final Integer MIN_WEIGHT = 1;
13
14     private LinkedList<MyEdge>[] edges;
15
16     public void readFromFile(String filename) {
17         try {
18             File file = new File(filename);
19             Scanner scanner = new Scanner(file);
20             int nodeNum = Integer.parseInt(scanner.nextLine());
21             int edgeNum = Integer.parseInt(scanner.nextLine());
22
23             if (nodeNum > MAX_NODES_NUM) {
24                 String message = String.format(
25                     "The number of nodes must be less than %d.",
26                     MAX_NODES_NUM
27                 );
28                 scanner.close();
29                 throw new RuntimeException(message);
30             }
31             if (edgeNum > MAX_EDGES_NUM) {
32                 String message = String.format(
33                         "The number of edges must be less than %d.",
34                         MAX_EDGES_NUM);
35                 scanner.close();
36                 throw new RuntimeException(message);
37             }
38
39             this.edges = new LinkedList[nodeNum];
40
41             for (int i = 0; i < nodeNum; i++) {
42                 this.edges[i] = new LinkedList<MyEdge>();
43             }
44             for (int i = 0; i < edgeNum; i++) {
45                 String[] chars = scanner.nextLine().split(" ");
46                 Integer[] nums = new Integer[chars.length];
47
48                 for (int j = 0; j < chars.length; j++) {
49                     nums[j] = Integer.parseInt(chars[j]);
50                 }
51
52                 if (nums[2] > MAX_WEIGHT || nums[2] < MIN_WEIGHT) {
53                     String message = String.format(
54                         "The weight value must be between %d and %d.",
55                         MIN_WEIGHT, MAX_WEIGHT
56                     );
```

```
57                      scanner.close();
58                      throw new RuntimeException(message);
59                  }
60
61                  this.edges[nums[0]].add(new MyEdge(nums[0], nums[1], nums[2]));
62                  this.edges[nums[1]].add(new MyEdge(nums[0], nums[1], nums[2]));
63              }
64
65              scanner.close();
66          }
67          catch (FileNotFoundException ex) {
68              System.out.println("File not found.");
69              System.exit(0);
70          }
71          catch (NumberFormatException ex) {
72              System.out.println("Input must be an integer value.");
73              System.exit(0);
74          }
75          catch (RuntimeException ex) {
76              System.out.println(ex.getMessage());
77              System.exit(0);
78          }
79      }
80
81      public LinkedList<MyEdge> getEdges(int id) {
82          return this.edges[id];
83      }
84
85      private Integer[] getLinkedNodes(int id) {
86          LinkedList<MyEdge> edges = this.getEdges(id);
87          Integer[] nodes = new Integer[edges.size()];
88          for (int i = 0; i < edges.size(); i++) {
89              Integer[] ends = edges.get(i).getNodes();
90              nodes[i] = ends[0] == id ? ends[1] : ends[0];
91          }
92
93          return nodes;
94      }
95
96      public Integer getWeight(Integer v, Integer w) {
97          LinkedList<MyEdge> edges = this.getEdges(v);
98          for (MyEdge edge : edges) {
99              if (Arrays.asList(edge.getNodes()).contains(w)) {
100                 return edge.getWeight();
101             }
102         }
103         String message = String.format("The vertices %s and %s are not connected
                .", v, w);
104         throw new RuntimeException(message);
105     }
```

```
106
107     private int getNodeSize() {
108         return this.edges.length;
109     }
110
111     public Integer[] getShortestPath(Integer start, Integer end) {
112         int size = this.getNodeSize();
113         boolean[] determined = new boolean[size];
114         Integer[] distances = new Integer[size];
115         Integer[] prevs = new Integer[size];
116
117         PriorityQueue<Node> found = new PriorityQueue<>();
118
119         for (int i = 0; i < size; i++) {
120             distances[i] = Integer.MAX_VALUE;
121         }
122         distances[start] = 0;
123
124         found.add(new Node(start, 0));
125
126         while (true) {
127             if (found.size() == 0) {
128                 break;
129             }
130
131             Node node = found.poll();
132             int minId = node.id;
133
134             if (determined[minId]) {
135                 continue;
136             }
137             determined[minId] = true;
138
139             for (int nextId : this.getLinkedNodes(minId)) {
140                 if (distances[nextId] > distances[minId] + this.getWeight(minId,
                        nextId)) {
141                     distances[nextId] = distances[minId] + this.getWeight(minId,
                            nextId);
142                     found.add(new Node(nextId, distances[nextId]));
143                     prevs[nextId] = minId;
144                 }
145             }
146         }
147
148         return prevs;
149     }
150 }
```

ソースコード 4: MyGraphTest.java

```
1  import java.util.ArrayList;
2  import java.util.InputMismatchException;
3  import java.util.NoSuchElementException;
4  import java.util.Scanner;
5
6  public class MyGraphTest {
7      public static void main(String[] args) {
8          try {
9              Scanner scanner = new Scanner(System.in);
10             String filename = scanner.nextLine();
11             int start = Integer.parseInt(scanner.nextLine());
12             int end = Integer.parseInt(scanner.nextLine());
13             scanner.close();
14
15             MyGraph graph = new MyGraph();
16             graph.readFromFile(filename);
17
18             Integer[] prevs = graph.getShortestPath(start, end);
19             Integer point = end;
20             Integer weightSum = 0;
21             ArrayList<Integer> path = new ArrayList<>();
22             path.add(0, end);
23
24             while (true) {
25                 if (point == start) {
26                     break;
27                 }
28
29                 if (prevs[point] == null) {
30                     String message = "The start and end points are not connected
                           .";
31                     throw new RuntimeException(message);
32                 }
33                 weightSum += graph.getWeight(point, prevs[point]);
34                 point = prevs[point];
35                 path.add(0, point);
36             }
37
38             for (int i = 0; i < path.size(); i++) {
39                 System.out.print(path.get(i));
40                 if (i == path.size() - 1) {
41                     System.out.print("\n");
42                 }
43                 else {
44                     System.out.print(" ");
45                 }
46             }
47             System.out.println(weightSum);
48         }
49         catch (InputMismatchException ex) {
```

```
50              System.out.println(ex.getMessage());
51              System.exit(0);
52          }
53          catch (NoSuchElementException ex) {
54              System.out.println("The actual number of edges given is less than
                    the number of edges specified in the file.");
55              System.exit(0);
56          }
57          catch (RuntimeException ex) {
58              System.out.println(ex.getMessage());
59              System.exit(0);
60          }
61      }
62  }
```