情報工学実験 II 手順書

ソフトウェア—離散系のシミュレーション

version 1.0

愛媛大学工学部情報工学科

2020 年度

目次

1　ソフトウェア—離散系のシミュレーション

1.1　はじめに

計算機を用いて行なわれる様々な計算を、その対象となる集合の性質により「離散 」的な計算と「連続」的な計算とに分けて考えることがある。本節では 2 種類の計算のうち「離散」に属する計算におけるプログラミングの基礎を学ぶことを目的とする。また、そのための題材として、離散計算の重要な部分を成すグラフ理論に基づく計算機プログラムを取り上げる。具体的には、重み付き無向グラフを対象に最短路検索の問題に対する代表的な計算アルゴリズムである Dijkstra の方法を用いたプログラムを作成し、その評価をすることで離散的な計算における計算機プログラムの実際を体験・学習する。

本節の内容は先行する講義科目「情報数学 II」における学習成果を基礎に置いており、また履修にあたって学生がその内容を修得していることを前提とする。

1.2　実験の進行について

十分な学習成果を得るために、提示された実験課題は漏らすことなく解決する必要がある。

2 回の実験実施日への課題の配分は概ね自由で構わないが、1 回目には課題 3 までを必ず終了させること。

また、以下に出てくるサンプルプログラムは、授業情報のウェブサイトからダウンロードできるので、それを使うこと。

1.2.1　中間レポート

各実験実施日の終りには担当者による進捗状況の確認を受けなくてはならない。その日の実験を終える前に、取り組んだ課題に関する必要十分な報告書を提出すること。

報告書に併せて、必要に応じて口頭試問を行うので、報告書または試問の結果について、不十分であると指摘された場合は、時間内もしくは自習時間中に報告書の修正、内容に関する再学習を行うこと。全ての課題への取り組みが十分に成されているという担当者の確認を経ずに、その日の実験を終了することはできない。

進捗状況の確認は学習の円滑な進行のために行うものである。必要な作業・考察の全てを授業時間中に終えるためのアドバイスを得る機会であり、試験・審査の類ではない。単位の認定は実験の実施と提出課題の評価によってのみ行う。報告書に要した時間や修正指示の回数、口頭試問の良否は評価には影響しない。

1.3　グラフと対応する数理モデル

「情報数学 II」の講義でも学んだ通り、グラフとは集合とその要素間の二項関係を併せたものであり、要素を頂点、要素間の関係を辺で置き変えた図形による表現である。

図 1 は 5 つの要素とその間の二項関係を 5 つの頂点 $\{1,2,3,4,5\}$ と 6 つの辺 $\{\{1,2\},\{1,5\},\{2,3\},\{2,5\},\{3,4\},\{3,5\}\}$ で表したものである。例えば、1:JR 2:堀端 3:市駅 4:温泉 5:本町 などと頂点に名前をつけてみれば、どこかの町の鉄道路線図のように見えてくる。

このようなグラフをデータとして C のプログラム中で扱うことを考える。

グラフの頂点には順番に番号を振ることにすれば、頂点に関してはその数だけを記憶しておけばよい。あとは、どのような辺が存在するか、すなわちどの要素間に関連があるかを記憶することが必要になる。

グラフの辺は頂点を端点に選んだものであり、頂点の数を n とすると、$n \times n$ 通りの辺が考えられる。これを 2 次元の配列を使って保持することを考える。

プログラムリスト 1(inputgraph.c) は標準入力から頂点の数と各辺の端点となる頂点の番号を読み込むサブルーチンと、そのサブルーチンを使って読み込んだ配列の内容を表示するプログラムの例である。

プログラムリスト 1

無向グラフの入力プログラム例	inputgraph.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <limits.h>
 4
 5  #define NMAX 10                   /* 頂点の数の上限 */
 6  int graph[NMAX + 1][NMAX + 1];    /* 辺（関連）の有無 */
 7  int n;                            /* 頂点の数 */
 8
 9  #define FALSE  0
10  #define TRUE   1
11
12  void inputgraph(void)             /* グラフデータ読込み */
13  {
14    int i, j;
15
16    puts("頂点の数を入力してください");
17    if (scanf("%d%*[^\n]", &n) != 1 || n > NMAX)
18      {
19        n = 0;  return;
20      }
21
22    for (i = 1; i <= n; i++)
23      for (j = 1; j <= n; j++)
24        graph[i][j] = FALSE;
25
26    puts("各辺の両端点を番号で入力してください");
27    while (scanf("%d%d%*[^\n]", &i, &j) == 2)
28      {
29        graph[i][j] = TRUE;
30        graph[j][i] = TRUE;
31      }
32  }
33
34  int main()
35  {
36    int i, j;
37
38    inputgraph();  /* 点の数 n，辺の有無 graph[][] を入力 */
39
40    puts("入力データ graph(i,j)=F/T");
```

```
41    for (i = 1; i <= n; i++) {
42      printf("%3d",i);
43      for (j = 1; j <= i; j++) {
44        if (graph[i][j])
45  fputs("  T",stdout);
46        else
47  fputs("  F",stdout);
48      }
49      puts("");
50    }
51    fputs("   ",stdout);
52    for (i = 1; i <= n; i++) printf("%3d",i);
53    puts("");
54
55    return EXIT_SUCCESS;
56  }
```

inputgraph.c において、関数 inputgraph は標準入力から頂点数 n と各辺の端点番号を読み込む。グラフは配列 graph[][] に辺の有無を真 (1) 偽 (0) の整数値として格納される。すなわち、辺 $\{i, j\}$ が存在すれば graph[i][j] は真 (1) となり、存在しなければ偽 (0) となる。

また、inputgraph 関数の使用例として、main 関数では入力されたグラフの内容を一覧表示している。

課題 1 プログラムリスト 1 または同等のプログラムを作成し、図 1 のグラフに対応するデータの入力と確認を行え。

課題 2 プログラムリスト 1 では扱うグラフは無向グラフであると仮定され、要素 i と j を結ぶ辺の入力に対して、配列 graph の成分 graph[i][j] と graph[j][i] の両方を真にしている。

有向グラフを扱う方法を検討し、考案した方法に則して課題 1 で作成したプログラムを変更せよ。

プログラムリスト 1 ではグラフのデータを格納する配列はデータの入力に先立って定義されている。したがって、利用できるグラフの頂点数は配列の大きさに制限され、プログラムの実行時には最大頂点数は変更できない。

十分に大きな配列を定義しておくという解決策は不必要なメモリの占有を生じ、結果として、他のデータの取り扱いのみならず、同時に実行されている別のプログラムの処理にも悪影響を与えてしまうかもしれない。そこで、配列の大きさを必要に応じて動的に変更することを考える。C 言語では他の言語に比べ、配列はポインタという概念と強く結びついている。したがって、配列をうまく扱うためにはポインタの理解が必要となる。

全ての変数はメモリ上のどこかにその値を格納する。変数の値が格納されるメモリ上の位置を、ここでは便宜上、アドレスと呼ぶ。特定の変数のアドレスを取り出すためにはアドレス演算子 & が用いられる。すなわち、&a は変数 a のアドレスを返す。

一方、ポインタとはポインタ型の変数のことであり、ポインタ型変数は、その値としてアドレスを持つ変数である。

また、ポインタ型変数はどのような型の変数のためのアドレスを持つのか宣言時に指定する必要がある。このとき、宣言されたポインタ型変数を指定された変数型から派生したポインタ型変数と呼ぶ。例えば、型宣言中に、次のように宣言された変数 b は int 型変数から派生したポインタ型変数である。

```
int *b
```

このとき、プログラム中ではポインタ型変数 b の持つアドレスに格納された int 型の値を *b で参照できる。
すなわち、int 型変数 a と int 型から派生したポインタ型変数 b が

```
int a, *b
```

と宣言されたとき、b=&a として、b に a のアドレスを代入すれば、*b は &a の指し示すメモリに格納された int 型変数値を与えるので a==*b が成立する。b や a のアドレス &a に変化が無ければ、変数 *b や a によっても a==*b が成立する。注意すべきは、他の変数と同様、ポインタ型変数は宣言されただけでは、その値は不定ということである。

さらに、ポインタに格納されたアドレスの値を使って連続して確保された複数の領域を参照し値を格納することができる。例えば、int 型から派生したポインタ型変数 a

```
int *a
```

に対して十分に領域が確保され、そのアドレスが代入されているとき、a+1 は a の次の int 型の領域のアドレスを示す。つまり、

```
*a = 1; *(a+1) = 3; *(a+2) = 5;
```

などとして、それぞれ独立した変数として異なる値を格納し読み出すことができる。このとき、ポインタの指し示す領域に格納された値を参照する演算子*が加算演算子 + よりも優先順位の高い演算子であることに注意する必要がある。例えば上述の代入の後で、*a+*(a+1)、*(a+*a) で読み出される値はそれぞれ 4 と 3 である。

このようなアドレスの加算を行なう表現には次のような [] を用いる方式もある。

```
a[0] = 1;  a[1] = 3;  a[2] = 5;
```

それぞれ、a に格納されたアドレスに続く「[] 内の値」番目の領域に格納された値を読み出すことができる。つまり、a[0] と*a、a[1] と*(a+1)、a[2] と*(a+2) が全て同等になる。

C の配列は、このようなポインタ型変数とそのアドレスに対する演算によって実現されている。例えば、int 型配列が

```
int c[3]
```

等として宣言された場合、メモリ上に int 型の変数 3 つ分の領域が確保され、その位置がアドレスとして特定される。このとき、c[0]==*c, c[1]==*(c+1), c[2]==*(c+2), が成立する。つまり、*(c+j) ($j = 0, \ldots, 2$) により c[j] のアドレスに格納された値が参照できる。

このことは、配列の宣言が指定された個数の値をメモリ上に格納するためのアドレスの確保と、対応するポインタ型変数の宣言とアドレスの代入にほぼ等しいということを示している。実際、

```
int *d


d=malloc(3*sizeof(int))
```

として、int 型から派生したポインタ型変数 d を宣言し、その値として、malloc 関数で int 型変数 3 個分を確保したアドレスを代入すれば、配列と同様の表現 d[0], d[1], d[2] を使って int 型の値が参照できる。

つまり、ヒープ上に値を格納する領域を確保し、そのアドレスを返す関数、malloc を用いて、予め配列の大きさを定めてしまうことなく、動的に必要な大きさの配列を利用することができる。[*1]

課題 3 ポインタを使い動的に配列を確保する方法を利用して、プログラム実行時に頂点数を柔軟に指定できるように課題 1 のプログラムを改良せよ。

また、改良されたプログラムにより、頂点数はどのくらいまで増やせるのか、理論的な限界を考察し、実際に確保できる限界を比較せよ。

さらに、データの格納方法をよく検討し、なるべく多くの頂点数を扱える方法を工夫せよ。

- ヒント

1. $n \times n$ 個の値を格納するためには、int graph[$n \times n$] として宣言された配列があれば良い。

   例えば、3 個の頂点 A, B, C からなるグラフの辺を考える。辺の両端点に、それぞれ 3 通りの頂点が選ばれる可能性があり、全体で $3 \times 3 = 9$ 通りの辺が存在し得る。そこで、この全ての頂点について、その存在ん真偽を変数に格納するならば、9 個の領域を確保すれば良いことになる。int v[3][3] 等として 2 次元配列を利用することも可能ではあるがより直接的には、次のようにポインタ型変数を 1 つ用いて必要な領域を確保することも考えられる。

   ```
   int *v, n;
   n = 3;
   v = malloc(n*n*sizeof(int));
   ```

   この方法であれば、 n の値をユーザが入力する等の仕組みを作って柔軟性を高めることも容易である。
2. 無向グラフであれば、$n \times n$ の配列の約半分は不要である。
3. ループした辺を認めないのであれば、対角成分は常に偽となる。

   前項の例では全ての組合せを認めているので、次の 9 通り全ての辺の存否について領域を確保することになってしまう。

   $$\{A,A\},\{A,B\},\{A,C\},\{B,A\},\{B,B\},\{B,C\},\{C,A\},\{C,B\},\{C,C\}.$$

   無向グラフを想定しているのであれば、例えば {A,B} と {B,A} の辺は一方だけが存在し、他方が存在しないということはあり得ず、両方の状態を別々に格納しておく必要は無い。また、ある点を出てまたその点に戻る自己ループにあたる辺を考える必要が無い場合も多い。上記の例であれば {A,A}、{B,B}、{C,C} がこれにあたる。これを併せて、自己ループの存在しない無向グラフのデータであれば、頂点数の 2 乗分ではなく、その半分以下の状態だけを格納しておけば全てのグラフの状態を表現できることが分かる。3 頂点の例で考えれば、次の 3 通りの辺の真偽だけを格納すれば良い。

   $$\{A,B\},\{A,C\},\{B,C\}.$$

[*1] 詳しくは malloc 関数のマニュアルページを参照すること。

1.4 最短経路検索と Dijkstra のアルゴリズム

前節までに取り扱ったグラフの各辺毎に固有の重みをかけたグラフを重みつきグラフと呼ぶ。図 1 のような路線図の各辺に、辺に対応する区間の運賃や所要時間、道のり等を重みとして付加すれば、さらに様々なモデルをグラフとして取り扱うことができるようになる。

重みつきグラフは、頂点の集合 V、辺の集合 E に加えて E の要素の関数 $w = w(e \mid e \in E)$ を併せたものである。例えば、図 2 は 9 つの頂点と 16 の辺からなる重みつき無向グラフである。

課題 4 課題 3 で作成したプログラムを改良し、重みつきグラフデータを入力するプログラムを作成せよ。

- ヒント

  配列 `graph` の値を辺の有無ではなく、辺の重みとすれば良い。辺の存在しない部分の値をどうするかを工夫せよ。

重みつきグラフ中の頂点を 2 つ選び、それぞれ s 点、t 点とする。s 点から出発して、次々と隣接する点へ各辺を辿って移動して t 点へ辿りつく経路を考えたとき、その方法は 1 通りとは限らない。経路のバリエーションがいく通りも考えられるとき、多くのバリエーションのうち、経路上の辺の重み全ての合計値が最も小さくなるようなものを最短経路と呼ぶ。

重みつきグラフと出発点 s 、終着点 t が与えられたとき、s から t への最短経路を求める問題を考える。重みつき無向グラフの最短経路を求める方法には Dijkstra のアルゴリズムと呼ばれるよく知られた方法が存在する。

以下では Dijkstra のアルゴリズムを図 2 で表されるグラフを用いて説明する。

Dijkstra のアルゴリズムの進行には、各点毎に付加される、s 点からその点までに辿る経路上の重み全ての和のうち最小のもの「最小距離」と、最小距離を実現する経路上の頂点リスト「最小経路」を用いる。各辺に付加される「重み」に対応して、このような各点に付加されるデータを「ラベル」と呼ぶ。このとき、以下のような手順で s 点から t 点への最短経路を求めることができる。

- Dijkstra のアルゴリズム

1. V を頂点、E を辺、w を辺の関数である重みとする。
2. V の部分集合 $V_1 = \{s\}$ とし、s に最小距離を示すラベル $l_s = 0$ を付加する。
3. V から V_1 を除いたものを V_2 とする。
4. E のうち、V_1 の要素だけを端点に持つものを E_1 とし、V_2 の要素だけを端点に持つものを E_2 とする。
5. E から E_1, E_2 を除いたものを B とする。
6. B の端点のうち、V_2 に属するものの順番に注目し v とする。
7. v の反対側の端点と、その最小距離ラベルを参照し、s から v までの最小距離 l_v と最小距離を与える経路上で v の直前の頂点 u を求める。
8. 最小の l_v を与える頂点を $v_{\min}$ とし、その直前の頂点を $u_{\min}$ とする。
9. $v_{\min}$ に最小距離を示すラベル $l_{v_{\min}}$ と直前の頂点 $u_{\min}$ を付加し、$v_{\min}$ を V_1 に加える。
10. t 点が V_1 に含まれていなければ、3 に戻る。
11. t 点に付加された最小距離のラベルが最小経路の距離となる。また、直前の頂点を示すラベルを t 点か

ら逆順に s 点まで辿れば最小経路を構成する頂点のリストになる。

課題 5 図 2 の頂点 1 を s 点、頂点 9 を t 点とする。s 点から t 点への最短経路を Dijkstra のアルゴリズムを用いて求めよ。(プログラムは必要ない。)

プログラムリスト 2(dijkstra.c) は inputgraph 関数を用いて読み込んだグラフデータにおいて頂点 1 を出発点とした場合の各点への最短経路を Dijkstra のアルゴリズムを用いて求めるプログラムの例である。

ただし、グラフデータを表す int 型配列 graph[][] はその大きさが固定で、値が各辺の重みを表し、辺が存在しない場合には int 型の最大値 INT_MAX を値に持つことを前提としている。また、出発点は頂点 1 に固定で、先に示した Dijkstrat のアルゴリズムの説明と異なり、V_2 が空になるまで反復を繰り返し、全ての点への最小距離と最短経路がその点のラベルとして求まるように構成されている。

プログラムリスト 2

Dijkstra のアルゴリズムプログラム例 dijkstra.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <limits.h>
 4
 5  #define NMAX 10                   /* 点の数の上限 */
 6  int graph[NMAX + 1][NMAX + 1];    /* 辺（関連）の有無 */
 7  int n;                            /* 点の数 */
 8
 9  #define START  1  /* 出発点 */
10  #define FALSE  0
11  #define TRUE   1
12
13  void inputgraph(void);             /* グラフデータ読込み */
14
15  int main()
16  {
17    int i, j, next, min;
18    static char visited[NMAX + 1];
19    static int leastdistance[NMAX + 1], previous[NMAX + 1];
20
21    inputgraph();  /* グラフデータ graph[][] の読込み */
22    for (i = 1; i <= n; i++) {
23      visited[i] = FALSE;  leastdistance[i] = INT_MAX;
24    }
25    leastdistance[START] = 0;  next = START;
26    do {
27      i = next;  visited[i] = TRUE;  min = INT_MAX;
28      for (j = 1; j <= n; j++) {
29        if (visited[j]) continue;
30        if (graph[i][j] < INT_MAX &&
31    leastdistance[i] + graph[i][j] < leastdistance[j]) {
32  leastdistance[j] = leastdistance[i] + graph[i][j];
```

```
33  previous[j] = i;
34        }
35        if (leastdistance[j] < min) {
36  min = leastdistance[j];  next = j;
37        }
38      }
39    } while (min < INT_MAX);
40    printf("点　直前の点　最短距離\n");
41    for (i = 1; i <= n; i++)
42      if (i != START && visited[i])
43        printf("%2d%10d%10d\n", i, previous[i], leastdistance[i]);
44    return EXIT_SUCCESS;
45  }
```

課題 6 プログラムリスト 2 を変更して、次の仕様を満たすように改良せよ。

1. 課題 4 で作成したプログラムに対応して、実行時に頂点数を指定できる。
2. 出発点 s と終着点 t を指定して最短経路上の経由頂点リストを出力できる。

2　オンライン授業 1 回目まで進めるべき範囲について

本来、実験の第 1 日目は少なくとも課題 1、2 を完了し、課題 3 への取り組みを開始している必要があります。オンライン授業の 1 回目までにそれが実現するために必要なことを明確にし、疑問点等について質問できるようにしてください。

3　オンライン授業 2 回目以降までに進めるべき範囲について

本来、実験の第 2 日目では課題 1 から 6 までを一通り完了することが期待されます。少なくとも「離散シミュレーション実験」の課題のうちに手をつけていないもの、何をするべきか分かっていないものが無いようにしなければいけません。オンライン授業 2 回目までに、疑問点を解消するために必要な質問をまとめ、送信しておいてください。

4　備考

テキスト等に問題があればご指摘ください。問題点は適宜修正し、告知いたします。

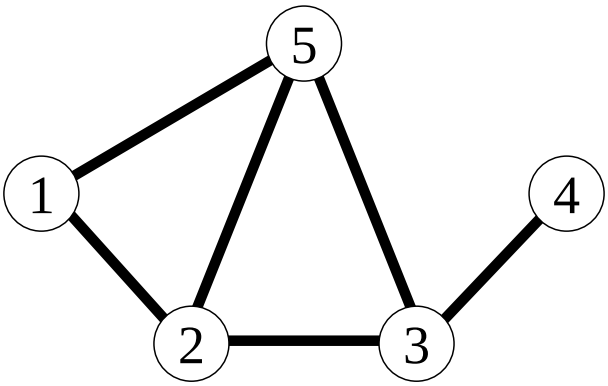

図１　無向グラフ

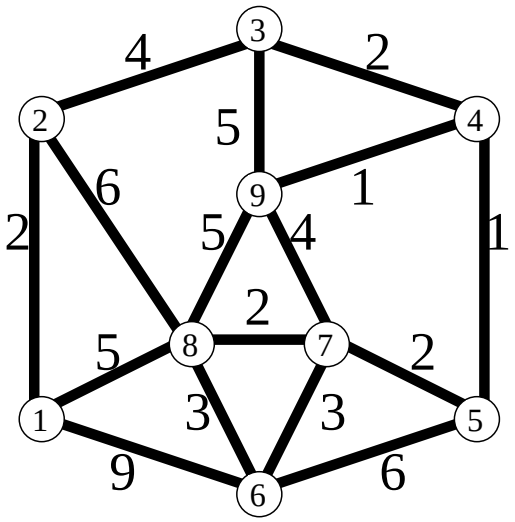

図２　重みつき無向グラフ