情報処理論2a

10章「最短経路探索」

(Dijkstra法)

中島康彦

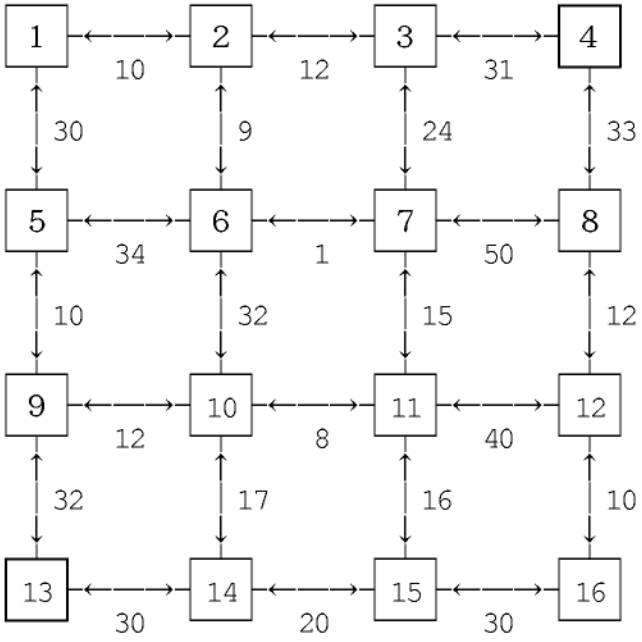
§10. 1 今日の作業ディレクトリを作る

1. % cd ⇒ ホームディレクトリへ移動
2. % mkdir chap22 ⇒ ディレクトリchap22を作成
3. % cd chap22 ⇒ ディレクトリchap22へ移動
4. % netscapeを使ってdata22をchap22へダウンロード
5. % tar xvf data22 ⇒ サンプルデータの複写

```
dijkstra.c
dijkstra.in
```

§10. 2 最短経路探索

ノードと区間距離から構成される以下の有向グラフにおいて，【13】から【4】までの最短経路を求める



§10. 3 Dijkstra法

(1)始点Nsおよび終点Ngを入力する.

```
13 4
```

(2)存在する区間データ(Ni Nj 距離)を全て入力.

```
1 2 10
2 1 10
2 3 12
3 2 12
```

(3)区間データ(Ni Nj 距離)からノードデータ(Ni Nsからの最短距離)を生成し，未調査リストAに登録. 初期値として最短距離を∞としておく.

未調査リストA: (1 ∞) (2 ∞) (3 ∞) … (16 ∞)

§10. 3 Dijkstra法(続き)

(4)始点Nsに関するノードデータを未調査リストAから調査済リストCへ移動する. その際, ノードデータを(Ns 0)に更新する.

未調査リストA:(1 ∞) (2 ∞) (3 ∞)…(16 ∞)
調査中リストB:
調査済リストC:(13 0)

(5)区間データを元に, 始点Nsから直接到達可能なノードNiを調べ, Niに関するノードデータを未調査リストAから調査中リストBに移動する. その際, ノードデータを(Ni NsからNiへの距離)に更新する.

未調査リストA:(1 ∞) (2 ∞) (3 ∞)…(16 ∞)
調査中リストB:(9 32) (14 30)
調査済リストC:(13 0)

§10. 3 Dijkstra法(続き)

(6)以下を調査中リストBが空になるまで繰り返す.

- (6a)Bから最短距離最小のノードNmを選びCへ移動.

  未調査リストA:(1 ∞) (2 ∞) (3 ∞)…(16 ∞)
  調査中リストB:(9 32)
  調査済リストC:(13 0) (14 30)

- (6b)Nmから直接到達可能なノードNiを探し, NiがAにあればBへ移動.

  未調査リストA:(1 ∞) (2 ∞) (3 ∞)…(16 ∞)
  調査中リストB:(9 32) (10 ∞) (15 ∞)
  調査済リストC:(13 0) (14 30)

- (6c)NiがBにあれば, 最短距離Ns→Nmに区間距離Nm→Niを加えた値と, 既知の最短距離Ns→Niを比べ, より小さい方を新たな最短距離Ns→Niとする.

  未調査リストA:(1 ∞) (2 ∞) (3 ∞)…(16 ∞)
  調査中リストB:(9 32) (10 30+17) (15 30+20)
  調査済リストC:(13 0) (14 30)

§10. 3 Dijkstra法(続き)

(7)Bが空になった時点で，以下の各リストが得られる.

未調査リストA:
調査中リストB:
調査済リストC: (4 120) (8 102) (12 90) (3 89) (16 80)
(2 77) (1 72) (6 68) (7 67) (11 52) (15 50)
(10 44) (5 42) (9 32) (14 30) (13 0)

- 始点Nsから各ノードへの最短距離が入っている.
- ノード13から4までの最短距離が120であることが判明.

(8)さらに，ノードデータに対して，一歩手前のノード番号を記憶させておき，終点から順に遡ることにより，最短距離だけでなく，最短経路を求めることができる.

§10. 4 データ構造の決定

入力データの構造

始点ノード番号 終点ノード番号
区間データ(FROM TO 距離)の並び …

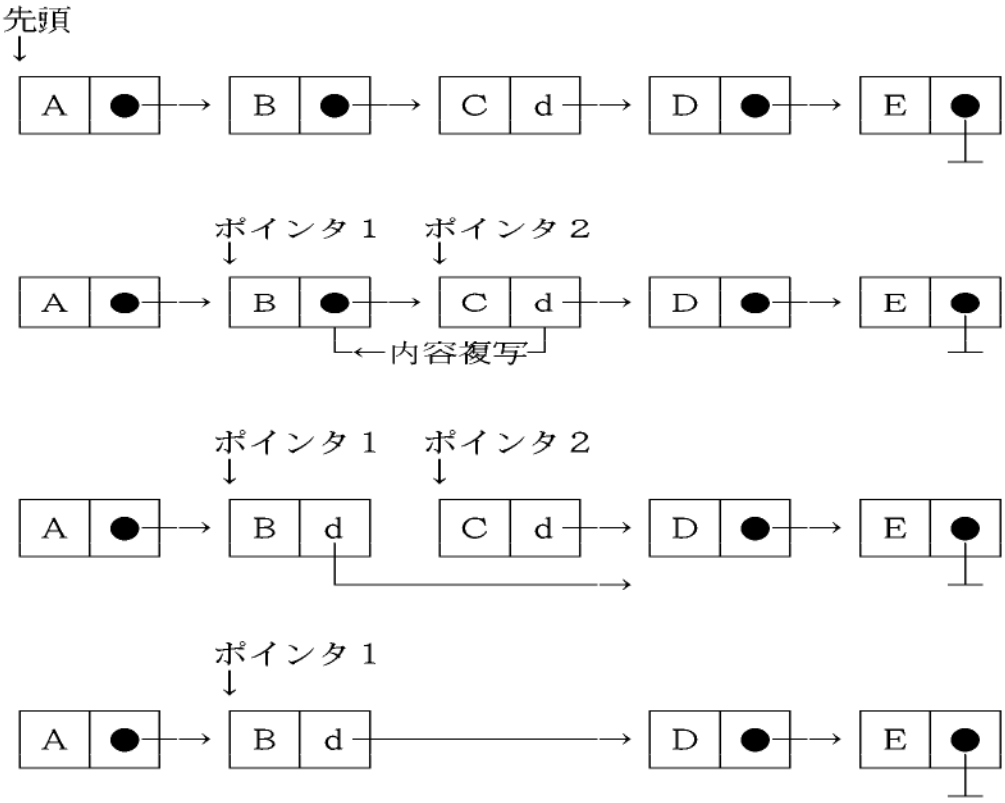
- 各項目には0以上の整数を用いる.
- ノード番号は連続していなくてもよい.

内部データ構造(区間データとノードデータ)

```
struct path {
  int from;            /* 区間の始点 */
  int to;              /* 区間の終点 */
  int dist;            /* 区間距離 */
  struct path *link;   /* 経路リストを繋ぐリンク */
};
struct node {
  int from;            /* 手前のノード */
  int id;              /* ノード番号 */
  int dist;            /* 最短距離 */
  struct node *link;   /* 同一リストを繋ぐリンク */
};
```

§10. 5 一次元リスト(一方向リスト)

一次元リスト構造(要素の削除と追加)



§10. 6 必要な関数を作る

ノード番号=idを探し，現最短距離よりもdistが小さければ書き換える．見つからなければ，malloc()によりノードデータ1個分の領域を取得してリストに追加．

```
addnode(struct node **n, int id, int from, int dist)
{
  struct node *w;
  for (w=*n; w; w=w->link) {
    if (w->id == id) { /* already registered */
      if (w->dist > dist) {
        w->from = from;
        w->dist = dist;
      }
      return;
    }
  }
  w = *n;
  if (!(*n = (struct node *)malloc(sizeof(struct node)))) {
    fprintf(stderr, "addnode: not enough memory\n");
    exit(1);
  }
  (*n)->id = id; (*n)->from = from; (*n)->dist = dist; (*n)->link = w;
}
```

§10. 6 必要な関数を作る(続き)

ノード番号=idを探し，リストmからリストnへ移動する．見つからなければ-1を戻す．

- vは1つ手前のノードを記憶する．
- v=NULLの場合，対象ノードはリストの先頭なので，mが次ノードを指すように変更する．

```
movenode(struct node **m, struct node **n, int id)
{
  struct node *v, *w;
  for (v=NULL, w=*m; w; v=w, w=w->link) {
    if (w->id == id) {
      if (v==NULL)
        *m = w->link;
      else
        v->link = w->link;
      w->link = *n;
      *n = w;
      return (0); /* found */
    }
  }
  return (-1); /* not found */
}
```

§10. 6 必要な関数を作る(続き)

ノード番号=idを探し現最短距離distを戻す．見つからなければ-1を戻す．

```
getdist(struct node *n, int id)
{
  for (; n; n=n->link) {
    if (n->id == id)
      return (n->dist); /* found */
  }
  return (-1); /* not found */
}
```

ノード番号=idを探し，手前のノード番号を戻す．見つからなければ-1を戻す．

```
getfrom(struct node *n, int id)
{
  for (; n; n=n->link) {
    if (n->id == id)
      return (n->from); /* found */
  }
  return (-1); /* not found */
}
```

§10. 6 必要な関数を作る(続き)

標準入力から始点/終点ノード番号と区間リストを入力
初期値∞として整数最大値(MAXINT)を用いる

```
readfile(int *id1, int *id2, struct path **p, struct node **n)
{
  int f, t, d;
  struct path *w;
  if (scanf("%d %d", id1, id2) != 2) { ...(1)
    fprintf(stderr, "readfile: start & goal node is required first\n");
    exit(1);
  }

  while (scanf("%d %d %d", &f, &t, &d) == 3) { ...(2)
    w = *p;
    if (!(*p = (struct path *)malloc(sizeof(struct path)))) {
      fprintf(stderr, "readfile: not enough memory\n");
      exit(1);
    }
    (*p)->from = f; (*p)->to = t; (*p)->dist = d; (*p)->link = w;
    addnode(n, f, -1, MAXINT); ...(3)
    addnode(n, t, -1, MAXINT); ...(3)
  }
}
```

§10. 6 必要な関数を作る(続き)

区間リストの内容を表示する.

```
printpath(struct path *p)
{
  for (; p; p=p->link)
    printf("(%d->%d %d)", p->from, p->to, p->dist);
  printf("\n");
}
```

ノードリストの内容を表示する.

```
printnode(struct node *n)
{
  for (; n; n=n->link)
    printf("(%d %d %d)", n->from, n->id, n->dist);
  printf("\n");
}
```

§10. 7 プログラム(Dijkstra.c)

```
main()
{
  struct path *plist = NULL; /* 区間リスト    */
  struct node *alist = NULL; /* 未調査リスト */
  struct node *blist = NULL; /* 調査中リスト */
  struct node *clist = NULL; /* 調査済リスト */
  struct path *p;
  struct node *n;
  int start_id, goal_id;
  int min_from, min_id, min_dist;

  readfile(&start_id, &goal_id, &plist, &alist); ...(1)(2)(3)
  printf("plist:"); printpath(plist);
  /* ここで各リスト表示 */

  movenode(&alist, &clist, start_id); ...(4)
  addnode(&clist, start_id, start_id, 0); ...(4)
  for (p=plist; p; p=p->link) {
    if (p->from == start_id) {
      movenode(&alist, &blist, p->to); ...(5)
      addnode(&blist, p->to, p->from, p->dist); ...(5)
    }
  }
  /* ここで各リスト表示 */
```

§10. 7 プログラム(Dijkstra.c続き)

```
  while (blist) { ...(6)
    min_dist = MAXINT;
    for (n=blist; n; n=n->link) {
      if (min_dist > n->dist) {
        min_id   = n->id;
        min_dist = n->dist;
      }
    }
    movenode(&blist, &clist, min_id); ...(6a)
    for (p=plist; p; p=p->link) {
      if (p->from == min_id) {
        movenode(&alist, &blist, p->to); ...(6b)
        if (getdist(blist, p->to) != -1)
          addnode(&blist, p->to, p->from, min_dist+p->dist); ...(6c)
      }
    }
    /* ここで各リスト表示 */
  } ...(7)
```

§10. 7 プログラム(Dijkstra.c続き)

実際には以下のように(5)と(6)をまとめることができる.

```
  movenode(&alist, &blist, start_id); ...(4)'
  addnode(&blist, start_id, start_id, 0); ...(4)'
  while (blist) { ...(6)
    min_dist = MAXINT;
    for (n=blist; n; n=n->link) {
      if (min_dist > n->dist) {
        min_id   = n->id;
        min_dist = n->dist;
      }
    }
    movenode(&blist, &clist, min_id); ...(6a)
    for (p=plist; p; p=p->link) {
      if (p->from == min_id) {
        movenode(&alist, &blist, p->to); ...(5)(6b)
        if (getdist(blist, p->to) != -1)
          addnode(&blist, p->to, p->from, min_dist+p->dist); ...(5)(6c)
      }
    }
    /* ここで各リスト表示 */
  } ...(7)
```

§10. 7 プログラム(Dijkstra.c続き)

この時点で，調査済リストCに，始点Nsから各ノードへの最短距離が格納されている.（調査中リストBは空）

- (8a)まずリストCからBへ終点ノードを移動する.
- (8b)fromを頼りに始点まで遡る.
- (8c)順にリストBへ詰め込んでいくと，最終的に，始点から終点までのノードが順に並ぶ.

```
  if (movenode(&clist, &blist, goal_id)) == -1) ...(8a)
    printf("==not found==\n"); ...cに終点が見つからなければ最短経路は存在しない
  else {
    while (goal_id != start_id) {
      goal_id = getfrom(blist, goal_id); ...(8b)
      movenode(&clist, &blist, goal_id); ...(8c)
    }
    printf("==found==\n");
    /* ここで各リスト表示 */
  }
  exit(0);
}
```

§10. 8 コンパイルと実行

1．コンパイルする.

```
% gcc dijkstra.c -o dijkstra
```

2．ファイルの確認

```
% cat dijkstra.in
```

```
13 4
1 2 10
2 1 10
:
12 16 10
16 12 10
```

3．ファイルから入力し，結果を表示する.

```
% ./dijkstra < dijkstra.in | less
```

§10. 8 コンパイルと実行(続き)

```
plist:(16->12 10)(12->16 10)…(1->2 10)
==start==
alist:(-1 16 2147483647)…(-1 1 2147483647)
blist:
clist:
========
alist:(-1 16 2147483647)…(-1 1 2147483647)
blist:(13 14 30)(13 9 32)
clist:(13 13 0)
========
alist:(-1 16 2147483647)…(-1 1 2147483647)
blist:(14 15 50)(14 10 47)(13 9 32)
clist:(13 14 30)(13 13 0)
========
alist:(-1 16 2147483647)…(-1 1 2147483647)
blist:(9 5 42)(14 15 50)(9 10 44)
clist:(13 9 32)(13 14 30)(13 13 0)
========
alist:(-1 16 2147483647)…(-1 2 2147483647)
blist:(5 6 76)(5 1 72)(14 15 50)(9 10 44)
clist:(9 5 42)(13 9 32)(13 14 30)(13 13 0)
========
alist:(-1 16 2147483647)…(-1 2 2147483647)
blist:(10 11 52)(5 6 76)(5 1 72)(14 15 50)
clist:(9 10 44)(9 5 42)(13 9 32)(13 14 30)(13 13 0)
========
alist:(-1 12 2147483647)…(-1 2 2147483647)
blist:(15 16 80)(10 11 52)(5 6 76)(5 1 72)
clist:(14 15 50)(9 10 44)(9 5 42)(13 9 32)(13 14 30)(13 13 0)
========
alist:(-1 8 2147483647)…(-1 2 2147483647)
blist:(11 12 92)(11 7 67)(15 16 80)(5 6 76)(5 1 72)
clist:(10 11 52)(14 15 50)(9 10 44)(9 5 42)(13 9 32)(13 14 30)(13 13 0)
========
alist:(-1 4 2147483647)(-1 2 2147483647)
blist:(7 8 117)(7 3 91)(11 12 92)(15 16 80)(7 6 68)(5 1 72)
clist:(11 7 67)(10 11 52)(14 15 50)(9 10 44)(9 5 42)(13 9 32)(13 14 30)(13 13 0)
```

§10. 8 コンパイルと実行(続き)

```
========
alist: (-1 4 2147483647)
blist: (6 2 77) (7 8 117) (7 3 91) (11 12 92) (15 16 80) (5 1 72)
clist: (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5 42) (13 9 32) (13 14 30) (13 13 0)
========
alist: (-1 4 2147483647)
blist: (6 2 77) (7 8 117) (7 3 91) (11 12 92) (15 16 80)
clist: (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5 42) (13 9 32) (13 14 30) (13 13 0)
========
alist: (-1 4 2147483647)
blist: (7 8 117) (2 3 89) (11 12 92) (15 16 80)
clist: (6 2 77) (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5 42) (13 9 32) (13 14 30) (13 13 0)
========
alist: (-1 4 2147483647)
blist: (7 8 117) (2 3 89) (16 12 90)
clist: (15 16 80) (6 2 77) (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5 42) (13 9 32) (13 14 30) (13 13
0)
========
alist:
blist: (3 4 120) (7 8 117) (16 12 90)
clist: (2 3 89) (15 16 80) (6 2 77) (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5 42) (13 9 32) (13 14
30) (13 13 0)
========
alist:
blist: (3 4 120) (12 8 102)
clist: (16 12 90) (2 3 89) (15 16 80) (6 2 77) (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5 42) (13 9
32) (13 14 30) (13 13 0)
========
alist:
blist: (3 4 120)
clist: (12 8 102) (16 12 90) (2 3 89) (15 16 80) (6 2 77) (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10 44) (9 5
42) (13 9 32) (13 14 30) (13 13 0)
========
alist:
blist:
clist: (3 4 120) (12 8 102) (16 12 90) (2 3 89) (15 16 80) (6 2 77) (5 1 72) (7 6 68) (11 7 67) (10 11 52) (14 15 50) (9 10
44) (9 5 42) (13 9 32) (13 14 30) (13 13 0)
```

§10. 8 コンパイルと実行(続き)

```
==found==
alist:
blist: (13 13 0) (13 9 32) (9 10 44) (10 11 52) (11 7 67) (7 6 68) (6 2 77) (2 3
89) (3 4 120)
clist: (12 8 102) (16 12 90) (15 16 80) (5 1 72) (14 15 50) (9 5 42) (13 14 30)
```

解はblistに入っている.

経路: 13→9→10→11→7→6→2→3→4

距離: 13 … 0

13→9 … 32

13→9→10 … 44

13→9→10→11 … 52

13→9→10→11→7 … 67

13→9→10→11→7→6 … 68

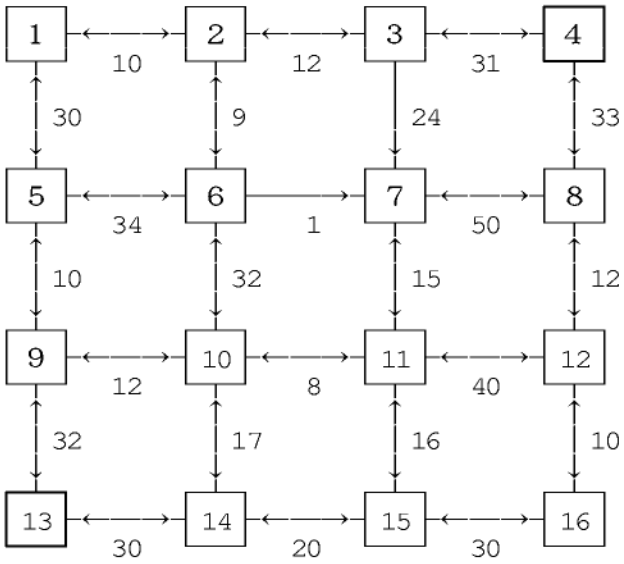
13→9→10→11→7→6→2 … 77

13→9→10→11→7→6→2→3 … 89

13→9→10→11→7→6→2→3→4 … 120

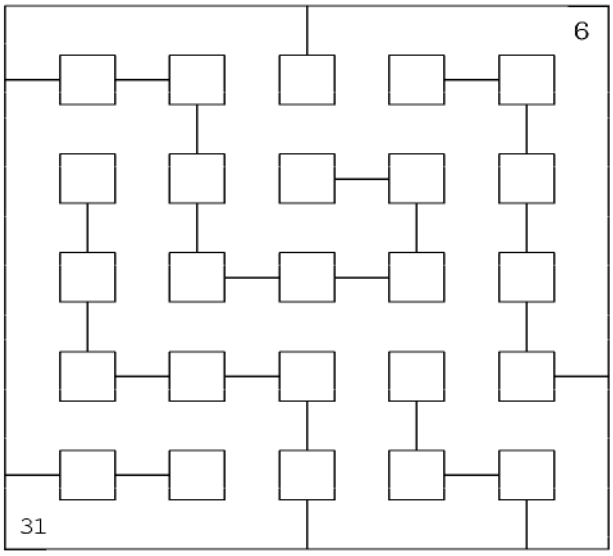
§10. 9 例題

7→6および7→3が通行不能になった場合について最短経路および経路を求めよ.
始点は13, 終点は4とする.



§10. 10 今日の課題

次の迷路(始点は31 終点は6)を表現せよ.
最短経路探索を用いて道筋と距離を求めよ.



宛先: nakashim@econ.kyoto-u.ac.jp
件名: unix2-学生番号

今日はここまで