情報処理論2a

11章「最大流問題」

(多層ネットワークとKarzanov法)

中島康彦

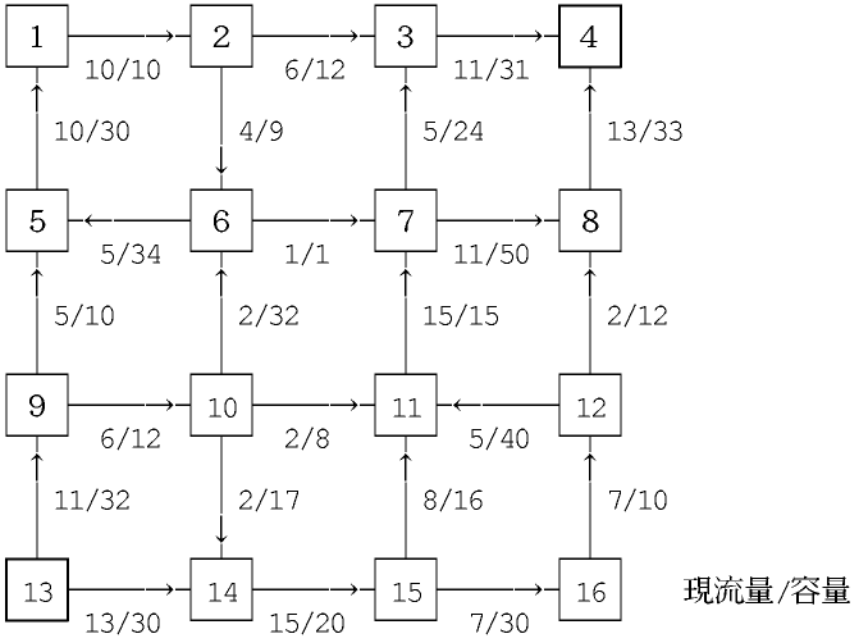
§11. 1 今日の作業ディレクトリを作る

1. % cd ⇒ ホームディレクトリへ移動
2. % mkdir chap23 ⇒ ディレクトリchap23を作成
3. % cd chap23 ⇒ ディレクトリchap23へ移動
4. % netscapeを使ってdata23をchap23へダウンロード
5. % tar xvf data23 ⇒ サンプルデータの複写

```
karzanov.c
karzanov.in
```

§11. 2 最大流問題

ノードと区間容量から構成される以下の有向グラフにおいて，【13】から【4】へ流すことのできる最大流量を求める



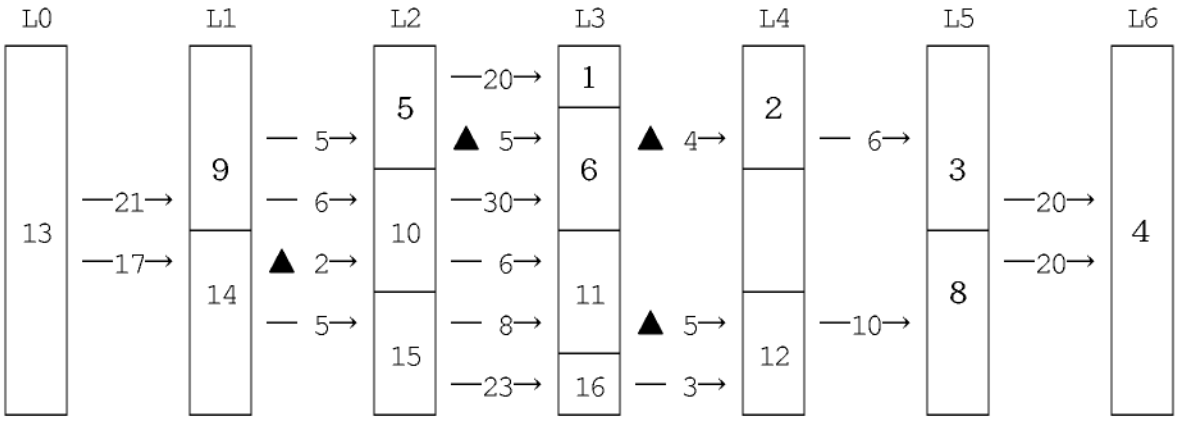
§11. 3 多層ネットワーク

ノードNiからノードNjへ増量できるのは以下の場合

- ▶ Ni→Njについて現流量<容量(順流を増加できる)
- ▶ Nj→Niについて現流量>0　(逆流を削減できる)

「順流の増加余地」と「逆流の削減余地▲」から生成.

ノード7への流入は飽和. もしノード2，12も飽和の場合は終点に至る増量可能経路が存在しない.

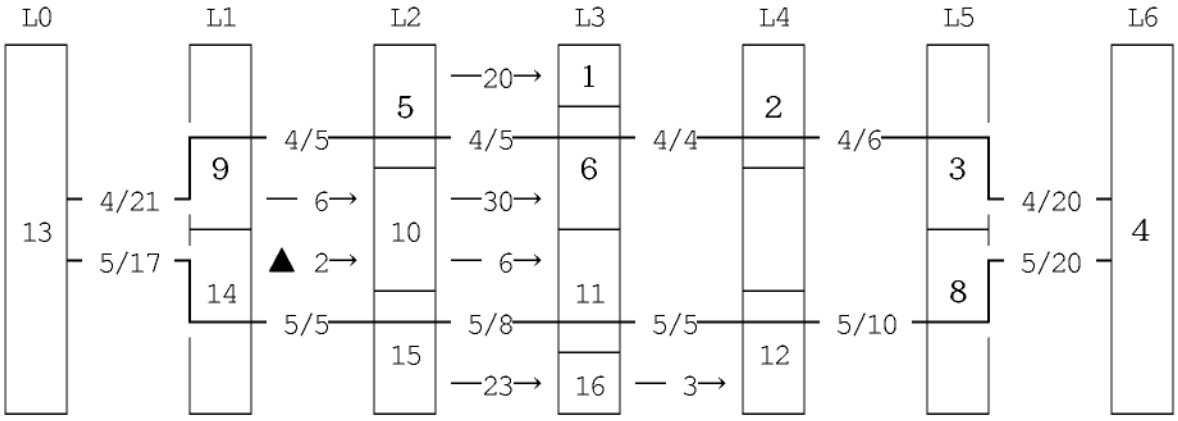


§11. 4 極大フロー

ある増量を仮定したとき，全ての増量可能経路において，少なくとも1つの区間が飽和している状態

以下は区間6→2，14→15，11→12が飽和しており極大

- ただし，極大は最大ではない．14→10→11→12を流量2，14→15→11→12を流量2，14→15→16→12を流量3とすれば，現流量5を7まで増量できる．



§11. 5 最大流問題の解法

最大流問題は，以下の方法により解くことができる．

- 多層ネットワークを構築する．
- 構築できなければ最大流を実現している．
- 構築できれば，次に極大フローを求める
- 極大フローを元のグラフに反映する．
- 以上を繰り返す．

Karzanov法

- 極大フローを効率良く求める方法の一つ

§11. 6 Karzanov法

(1)始点へ∞が流入すると仮定する. 各ノードでは流入量−流出量を変数ovfへ記憶しておく. 各区間は全て開いている. 次層L1から(2)を行う.
(2)層の各ノードへ, 開いている区間を使って順に可能な限り流入させ, 飽和した区間は閉じる. ノード毎に流入元と流入量を記憶する. 流入量を流入元のovfから減じ流入先のovfに加える. (2)を最終層まで繰り返す.
(3)最終層から遡って, ovfが0でないノードを含む最初の層Liについて, 各ノードのovfが0となるよう流入分を流入元へ押し戻す. 押し戻した区間は閉じる. 戻した量を流入元のovfに加え流入先から減じる. 手前の層Li−1が始点でなければ, Ni−1から(2)を繰り返す.
(4)Li−1が始点であれば, 流入超過は解消されている. 極大フローの解が各区間に格納されている.

§11. 7 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 現流量 容量)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, ノードデータ, 層データ)

- 区間データ構造は, 入力区間データを格納するplistの他に, 多層ネットワークblistにおける区間の表現に用いる.

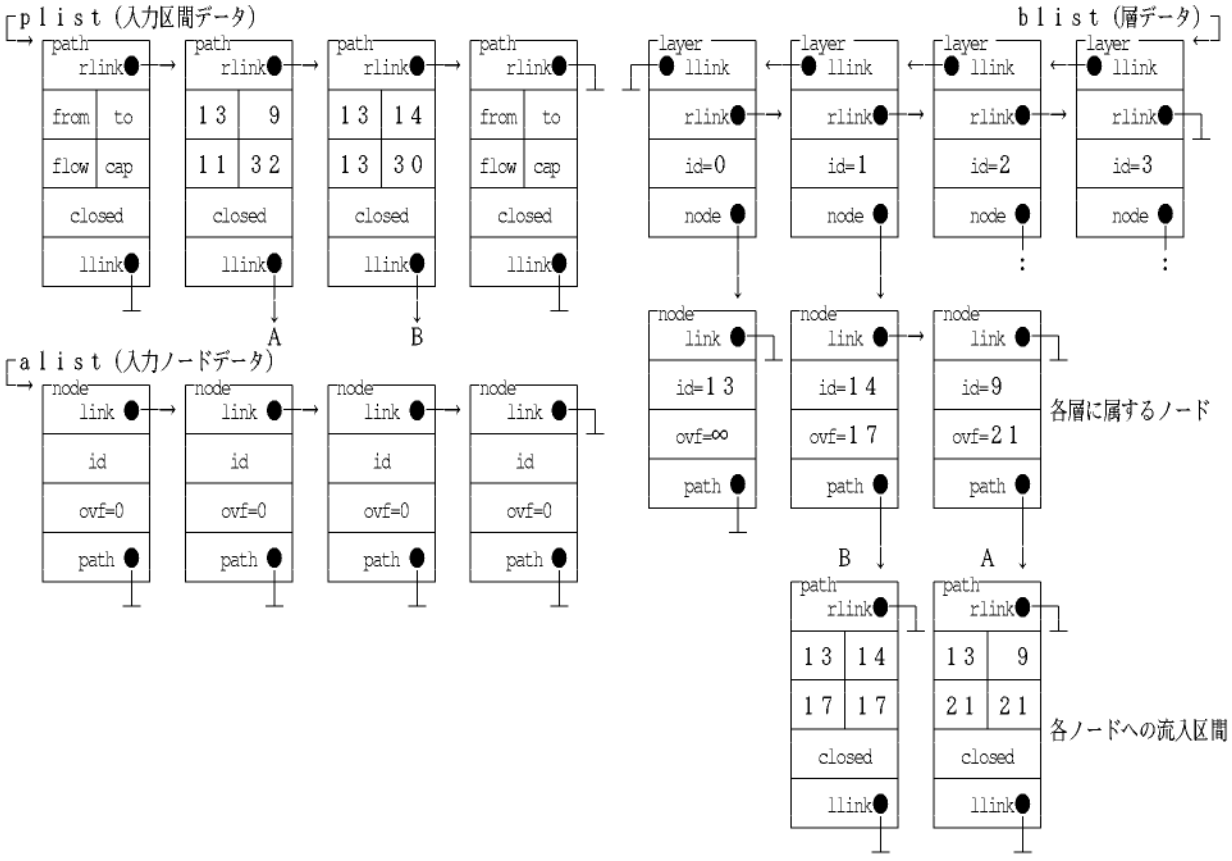
```
struct path {
  int from;             /* 区間の始点 */
  int to;               /* 区間の終点 */
  int flow;             /* 区間流量 */
  int cap;              /* 区間容量 */
  int closed;           /* ﾒ切 */
  struct path  *llink;/* plistからlayer内要素を指すリンク */
  struct path  *rlink;/* リスト内を繋ぐリンク */
};
```

§11. 7 データ構造の決定(続き)

- 各層データlayerはblistに繋がれる.(双方向リスト)
- 最初alistに繋がれるノードは,多層ネットワーク構築後はlayer内のnodeに繋がれる.同一リストにおけるノード間の繋ぎにはlinkを使用する.
- layerに繋がれた各ノードのpathには,多層ネットワークにおけるノードへの流入区間が繋がれる.

```
struct node {
  int id;              /* ノード番号 */
  int ovf;             /* 流入超過分 */
  struct path  *path;  /* 流入リストを繋ぐリンク */
  struct node  *link;  /* 同一リストを繋ぐリンク */
};
struct layer {
  int id;              /* 層番号 */
  struct node  *node;  /* 含むノード */
  struct layer *llink;/* start_idへ向かうリンク */
  struct layer *rlink;/* goal_idへ向かうリンク */
};
```

§11. 8 データ構造の決定(続き)



§11. 9 必要な関数を作る

```
/* 標準入力から開始ノード番号と区間リストを読み込む */
readfile(int *id1, int *id2, struct path **p, struct node **n)
{
  int f, t, fl, c;
  struct path *w;

  if (scanf("%d %d", id1, id2) != 2) {
    fprintf(stderr, "readfile: start & goal node is required first\n");
    exit(1);
  }

  while (scanf("%d %d %d %d", &f, &t, &fl, &c) == 4) {
    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)->flow   = fl;  (*p)->cap    = c;
    (*p)->closed = 0;   (*p)->llink  = NULL;
    (*p)->rlink  = w;
    addnode(n, f);
    addnode(n, t);
  }
}
```

§11. 9 必要な関数を作る(続き)

```
/* 区間を追加する */
addpath(struct node *n, int id, int from, int flow, int cap, struct path *p)
{
  struct path **v, *w;

  for (v=NULL; n; n=n->link) {
    if (n->id == id) { /* already registered */
      v = &n->path;
      break;
    }
  }
  if (!v) /* not found */
    return;
  w = *v;
  if (!(*v = (struct path *)malloc(sizeof(struct path)))) {
    fprintf(stderr, "addpath: not enough memory\n");
    exit(1);
  }
  (*v)->from   = from;  (*v)->to     = id;
  (*v)->flow   = flow;  (*v)->cap    = cap;
  (*v)->closed = 0;     (*v)->llink  = NULL;
  (*v)->rlink  = w;
     p->llink  = *v;
}
```

§11. 9 必要な関数を作る(続き)

```
/* ノード番号=idを探し,ノードが見つからなければ追加する */
addnode(struct node **n, int id)
{
  struct node *w;

  for (w=*n; w; w=w->link) {
    if (w->id == id) /* already registered */
      return;
  }
  w = *n;
  if (!(*n = (struct node *)malloc(sizeof(struct node)))) {
    fprintf(stderr, "addnode: not enough memory\n");
    exit(1);
  }
  (*n)->id   = id;
  (*n)->ovf  = 0;
  (*n)->path = NULL;
  (*n)->link = w;
}
```

§11. 9 必要な関数を作る(続き)

```
/* ノード番号=idを探し移動する     */
/* id=-1の時は最初の1つを移動する */
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 (id == -1 || w->id == id) {
      if (v==NULL)
        *m = w->link;
      else
        v->link = w->link;
      w->link = *n;
      *n = w;
      return; /* found */
    }
  }
  /* not found */
}
```

§11. 9 必要な関数を作る(続き)

```
/* 層番号=idを探し,層が見つからなければ追加する */
addlayer(struct layer **l, int id)
{
  struct layer *w;

  for (w=*l; w; w=w->llink) {
    if (w->id == id) /* already registered */
      return;
  }
  w = *l;
  if (!(*l = (struct layer *)malloc(sizeof(struct layer)))) {
    fprintf(stderr, "addlayer: not enough memory\n");
    exit(1);
  }
  if (w) w->rlink = *l;
  (*l)->id    = id;
  (*l)->node  = NULL;
  (*l)->llink = w;
  (*l)->rlink = NULL;
}
```

§11. 9 必要な関数を作る(続き)

```
/* ノード番号=idを探し,ovfにdiffを加えて結果を戻す */
/* ノードが見つからなければ-1を戻す                 */
getovf(struct node *n, int id, int diff)
{
  for (; n; n=n->link) {
    if (n->id == id) {
      n->ovf += diff;
      return (n->ovf); /* found */
    }
  }
  return (-1); /* not found */
}

/* 区間リストの内容を表示する */
printpath(struct path *p, int flag)
{
  for (; p; p=p->rlink)
    printf("(%d->%d %d %d %c)",
            p->from, p->to, p->flow, p->cap, p->closed?'X':'O');
  if (flag)
    printf("\n");
}
```

§11. 9 必要な関数を作る(続き)

```
/* ノードリストの内容を表示する */
printnode(struct node *n, int flag)
{
  for (; n; n=n->link) {
    printf("[%d %d", n->id, n->ovf);
    printpath(n->path, 0);
    printf("]");
  }
  if (flag)
    printf("\n");
}

/* 層リストの内容を表示する */
printlayer(struct layer *l, int flag)
{
  for (; l; l=l->llink) {
    printf("(%d", l->id);
    printnode(l->node, 0);
    printf(")");
  }
  if (flag)
    printf("\n");
}
```

§11. 10 プログラム(Karzanov.c)

```
#include <stdio.h>
#define MAXINT (~(1<<(sizeof(int)*8-1)))        整数最大値の定義
#define min(a,b) ((a<b)?(a):(b)) より小さい方を選択するマクロ

main()
{
  struct path  *plist = NULL; /* 区間リスト    */
  struct node  *alist = NULL; /* ノードリスト */
  struct layer *blist = NULL; /* 多層ネットワーク */
  struct path  *p;
  struct node  *n;
  struct layer *l;
  int start_id, goal_id;
  int layer_id, diff, flag;

  readfile(&start_id, &goal_id, &plist, &alist);
  printf("plist:"); printpath(plist, 1);        入力区間データの表示

  printf("==layer start==\n");    多層ネットワーク構築前のデータ構造を表示
  /* alistおよびblistの表示 */
```

§11. 10 プログラム(Karzanov.c続き)

```
/* 多層ネットワークの生成 */
layer_id = 0;
addlayer(&blist, layer_id++);        多層ネットワークに層0を生成
movenode(&alist, &blist->node, start_id);        始点を層0へ移動する.
while (n = blist->node) {    層Lに属するノードに関して以下を繰り返す.
  if (getovf(n, goal_id, 0) != -1) { /* completed */ 終点に到達したので終了
    while (blist->node)
      movenode(&blist->node, &alist, -1);        最終層から一旦ノードを追い出し,
    movenode(&alist, &blist->node, goal_id);    終点のみを登録する.
    break;        多層ネットワークの生成を終了.
  }
  else {
    addlayer(&blist, layer_id++); 次層L+1を追加する.
    for (; n; n=n->link) {    現層Lに属するノードNiの流出先Njを順に調べる.
      for (p=plist; p; p=p->rlink) {      入力区間データを順に調べ,
        if (p->from == n->id && p->flow < p->cap) {   Ni→Njについて現流量<容量であれば層L+1に登録
          movenode(&alist, &blist->node, p->to);
          addpath(blist->node, p->to, p->from, 0, p->cap-p->flow, p);
        }
        else if (p->to == n->id && p->flow > 0) {      Nj→Niについて現流量>0であれば層L+1に登録
          movenode(&alist, &blist->node, p->from);
          addpath(blist->node, p->from, p->to, 0, p->flow, p);
        }
      }
    }
  }
  printf("==layer%d==\n", blist->id);      層L+1の層番号を表示し, 現状をレポート
  /* alistおよびblistの表示 */
}
if (!blist->node) {    最終層にノードが1つもなければ, 多層ネットワークを構築できなかった. この場合, 最大流を実現しているので終了.
  printf("==full==\n");
  exit(0);
}

printf("==karzanov start==\n");    Karzanov法適用前のデータ構造を表示
/* alistおよびblistの表示 */
```

§11. 10 プログラム(Karzanov.c続き)

```
/* Karzanov法により極大流を求める */
for (l=blist; l && l->llink; l=l->llink);        始点を探す
l->node->ovf = MAXINT;        ...(1)始点へ∞を流入させる.
for (l=l->rlink; l && l->llink; l=l->llink) {
  /* 流出 */
  for (; l; l=l->rlink) {    ...(2)次層以降の各ノードへ, 開いている区間を使って順に可能な限り流入させる.
    for (n=l->node; n; n=n->link) {        ...(2)各ノード毎に調べる.
      /* 流出可能分を順流 */
      for (p=n->path; p; p=p->rlink) {    ...(2)流入区間を順に調べる. 区間が開いており, diff(>0)だけ流入可能であれば,
        if (!p->closed && (diff = min(getovf(l->llink->node,p->from,0),p->cap-p->flow)) > 0) {
          getovf(l->llink->node, p->from, -diff);        ...(2)流入元のOVFからdiffを減じる.
          n->ovf   += diff;                    ...(2)流入先のOVFにdiffを加える.
          p->flow  += diff;                    ...(2)区間流量を更新する.
          if (p->flow == p->cap) p->closed = 1;        ...(2)限度一杯であれば区間を閉じる.
        }
      }
    }
  }
  /* 流入超過ノード探索 */
  for (flag=0,l=blist->llink; l && l->llink; l=l->llink) {  ...(3)最終層から遡り, 最初の流入超過層に関して超過分を押し戻す.
    for (n=l->node; n; n=n->link) {        ...(3)各ノード毎に調べる.
      /* 流入超過分を逆流 */
      for (p=n->path; p; p=p->rlink) {    ...(3)流入区間を順に調べる. 流入超過であり, diff(>0)だけ押し戻し可能であれば,
        if ((diff = min(n->ovf,p->flow)) > 0) {
          getovf(l->llink->node, p->from, diff);        ...(3)流入元のOVFにdiffを加える.
          n->ovf   -= diff;                    ...(3)流入先のOVFからdiffを減じる.
          p->flow  -= diff;                    ...(3)区間流量を更新する.
          p->closed = 1;                       ...(3)区間を閉じる.
          flag      = 1;                       ...(3)現在の層において押し戻しを行った印を付ける.
        }
      }
    }
    if (flag) break;  ...(3)現在の層において押し戻しを行った場合, (2)から再開する.
  }
  printf("==layer%d==\n", l->id); 現在の層番号を表示する.
  /* alistおよびblistの表示 */
}  ...(4)現在の層が始点であれば, 流入超過は解消されている. 極大フローの解が各区間に格納されている.
```

§11. 10 プログラム(Karzanov.c続き)

```
  /* 差分を出力する */最終層のOVFが極大フローを保持している.
  printf("==difference:%d==\n", blist->node->ovf);
  for (p=plist; p; p=p->rlink) { 入力区間データから多層ネットワークの区間を辿り,
    if      (!p->llink)                    diff =  0;
    else if (p->from == p->llink->from) diff =  p->llink->flow;
    else                                   diff = -p->llink->flow;
    if (diff) 増減があれば表示する.
      printf("(%d->%d %d)",p->from,p->to,diff);
  }
  printf("\n");

  /* 元の区間流量を修正して出力する */
  printf("==total flow==\n");
  printf("%d %d\n", start_id, goal_id);
  for (p=plist; p; p=p->rlink) { 入力区間データから多層ネットワークの区間を辿り,
    if      (!p->llink)                    diff =  0;
    else if (p->from == p->llink->from) diff =  p->llink->flow;
    else                                   diff = -p->llink->flow;
    printf("%d %d %d %d\n",p->from,p->to,p->flow+diff,p->cap);
  } 反映後の区間データを出力する. これを用いて再度プログラムを実行することができる.

  exit(0);
}
```

§11. 11 コンパイルと実行

1. コンパイルする.

```
% gcc karzanov.c -o karzanov
```

2. ファイルの確認

```
% cat karzanov.in

13 4
1 2 10 10
2 3 6 12
:
12 8 2 12
16 12 7 10
```

3. ファイルから入力し, 結果を表示する.

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

§11. 11 コンパイルと実行(続き)

```
plist:(16->12 7 10 O)(12->8 2 12 O)(8->4 13 33 O)(15->11 8 16 O)(11->7 15 15 O)(7->3 5 24 O)(10->14 2 17
O)(10->6 2 32 O)(2->6 4 9 O)(13->9 11 32 O)(9->5 5 10 O)(5->1 10 30 O)(15->16 7 30 O)(14->15 15 20 O)(13->14 13
30 O)(12->11 5 40 O)(10->11 2 8 O)(9->10 6 12 O)(7->8 11 50 O)(6->7 1 1 O)(6->5 5 34 O)(3->4 11 31 O)(2->3 6 12
O)(1->2 10 10 O)

==layer start==
alist:[16 0][15 0][14 0][13 0][12 0][11 0][10 0][9 0][8 0][7 0][5 0][6 0][4 0][3 0][2 0][1 0]
blist:
==layer1==
alist:[16 0][15 0][12 0][11 0][10 0][8 0][7 0][5 0][6 0][4 0][3 0][2 0][1 0]
blist:(1[14 0(13->14 0 17 O)][9 0(13->9 0 21 O)])(0[13 0])
==layer2==
alist:[16 0][12 0][11 0][8 0][7 0][6 0][4 0][3 0][2 0][1 0]
blist:(2[5 0(9->5 0 5 O)][15 0(14->15 0 5 O)][10 0(9->10 0 6 O)(14->10 0 2 O)])(1[14 0(13->14 0 17 O)][9
0(13->9 0 21 O)])(0[13 0])
==layer3==
alist:[12 0][8 0][7 0][4 0][3 0][2 0]
blist:(3[16 0(15->16 0 23 O)][11 0(10->11 0 6 O)(15->11 0 8 O)][6 0(10->6 0 30 O)(5->6 0 5 O)][1 0(5->1 0 20
O)])(2[5 0(9->5 0 5 O)][15 0(14->15 0 5 O)][10 0(9->10 0 6 O)(14->10 0 2 O)])(1[14 0(13->14 0 17 O)][9 0(13->9
0 21 O)])(0[13 0])
==layer4==
alist:[8 0][7 0][4 0][3 0]
blist:(4[2 0(6->2 0 4 O)][12 0(11->12 0 5 O)(16->12 0 3 O)])(3[16 0(15->16 0 23 O)][11 0(10->11 0 6 O)(15->11 0
8 O)][6 0(10->6 0 30 O)(5->6 0 5 O)][1 0(5->1 0 20 O)])(2[5 0(9->5 0 5 O)][15 0(14->15 0 5 O)][10 0(9->10 0 6
O)(14->10 0 2 O)])(1[14 0(13->14 0 17 O)][9 0(13->9 0 21 O)])(0[13 0])
==layer5==
alist:[7 0][4 0]
blist:(5[8 0(12->8 0 10 O)][3 0(2->3 0 6 O)])(4[2 0(6->2 0 4 O)][12 0(11->12 0 5 O)(16->12 0 3 O)])(3[16
0(15->16 0 23 O)][11 0(10->11 0 6 O)(15->11 0 8 O)][6 0(10->6 0 30 O)(5->6 0 5 O)][1 0(5->1 0 20 O)])(2[5
0(9->5 0 5 O)][15 0(14->15 0 5 O)][10 0(9->10 0 6 O)(14->10 0 2 O)])(1[14 0(13->14 0 17 O)][9 0(13->9 0 21
O)])(0[13 0])
==layer6==
alist:
blist:(6[7 0(3->7 0 5 O)(8->7 0 11 O)][4 0(3->4 0 20 O)(8->4 0 20 O)])(5[8 0(12->8 0 10 O)][3 0(2->3 0 6
O)])(4[2 0(6->2 0 4 O)][12 0(11->12 0 5 O)(16->12 0 3 O)])(3[16 0(15->16 0 23 O)][11 0(10->11 0 6 O)(15->11 0 8
O)][6 0(10->6 0 30 O)(5->6 0 5 O)][1 0(5->1 0 20 O)])(2[5 0(9->5 0 5 O)][15 0(14->15 0 5 O)][10 0(9->10 0 6
O)(14->10 0 2 O)])(1[14 0(13->14 0 17 O)][9 0(13->9 0 21 O)])(0[13 0])
```

§11. 11 コンパイルと実行(続き)

```
==karzanov start==
alist:[7 0(3->7 0 5 O)(8->7 0 11 O)]
blist:(6[4 0(3->4 0 20 O)(8->4 0 20 O)])(5[8 0(12->8 0 10 O)][3 0(2->3 0 6 O)])(4[2 0(6->2 0 4 O)][12 0(11->12
0 5 O)(16->12 0 3 O)])(3[16 0(15->16 0 23 O)][11 0(10->11 0 6 O)(15->11 0 8 O)][6 0(10->6 0 30 O)(5->6 0 5
O)][1 0(5->1 0 20 O)])(2[5 0(9->5 0 5 O)][15 0(14->15 0 5 O)][10 0(9->10 0 6 O)(14->10 0 2 O)])(1[14 0(13->14 0
17 O)][9 0(13->9 0 21 O)])(0[13 0])

==layer3==
alist:[7 0(3->7 0 5 O)(8->7 0 11 O)]
blist:(6[4 12(3->4 4 20 O)(8->4 8 20 O)])(5[8 0(12->8 8 10 O)][3 0(2->3 4 6 O)])(4[2 0(6->2 4 4 X)][12 0(11->12
5 5 X)(16->12 3 3 X)])(3[16 0(15->16 3 23 X)][11 0(10->11 5 6 X)(15->11 0 8 O)][6 0(10->6 0 30 X)(5->6 4 5
X)][1 0(5->1 0 20 O)])(2[5 1(9->5 5 5 X)][15 2(14->15 5 5 X)][10 3(9->10 6 6 X)(14->10 2 2 X)])(1[14 10(13->14
17 17 X)][9 10(13->9 21 21 X)])(0[13 2147483609])
==layer3==
alist:[7 0(3->7 0 5 O)(8->7 0 11 O)]
blist:(6[4 12(3->4 4 20 O)(8->4 8 20 O)])(5[8 0(12->8 8 10 O)][3 0(2->3 4 6 O)])(4[2 0(6->2 4 4 X)][12 0(11->12
5 5 X)(16->12 3 3 X)])(3[16 0(15->16 3 23 X)][11 0(10->11 3 6 X)(15->11 2 8 O)][6 0(10->6 0 30 X)(5->6 4 5
X)][1 0(5->1 0 20 X)])(2[5 1(9->5 5 5 X)][15 0(14->15 5 5 X)][10 5(9->10 6 6 X)(14->10 2 2 X)])(1[14 10(13->14
17 17 X)][9 10(13->9 21 21 X)])(0[13 2147483609])
==layer2==
alist:[7 0(3->7 0 5 O)(8->7 0 11 O)]
blist:(6[4 12(3->4 4 20 O)(8->4 8 20 O)])(5[8 0(12->8 8 10 O)][3 0(2->3 4 6 O)])(4[2 0(6->2 4 4 X)][12 0(11->12
5 5 X)(16->12 3 3 X)])(3[16 0(15->16 3 23 X)][11 0(10->11 3 6 X)(15->11 2 8 O)][6 0(10->6 0 30 X)(5->6 4 5
X)][1 0(5->1 0 20 X)])(2[5 0(9->5 4 5 X)][15 0(14->15 5 5 X)][10 0(9->10 1 6 X)(14->10 2 2 X)])(1[14 10(13->14
17 17 X)][9 16(13->9 21 21 X)])(0[13 2147483609])
==layer1==
alist:[7 0(3->7 0 5 O)(8->7 0 11 O)]
blist:(6[4 12(3->4 4 20 O)(8->4 8 20 O)])(5[8 0(12->8 8 10 O)][3 0(2->3 4 6 O)])(4[2 0(6->2 4 4 X)][12 0(11->12
5 5 X)(16->12 3 3 X)])(3[16 0(15->16 3 23 X)][11 0(10->11 3 6 X)(15->11 2 8 O)][6 0(10->6 0 30 X)(5->6 4 5
X)][1 0(5->1 0 20 X)])(2[5 0(9->5 4 5 X)][15 0(14->15 5 5 X)][10 0(9->10 1 6 X)(14->10 2 2 X)])(1[14 0(13->14 7
17 X)][9 0(13->9 5 21 X)])(0[13 2147483635])
```

§11. 11 コンパイルと実行(続き)

```
==difference:12==
(16->12 3)(12->8 8)(8->4 8)(15->11 2)(10->14 -2)(2->6 -4)(13->9 5)(9->5
4)(15->16 3)(14->15 5)(13->14 7)(12->11 -5)(10->11 3)(9->10 1)(6->5 -4)(3->4
4)(2->3 4)
```

極大フローは流量12であり，そのための解は，

区間 16->12を　3だけ増加
区間 12->8を　8だけ増加
区間 8->4を　8だけ増加
区間 15->11を　2だけ増加
区間 10->14を　2だけ減少▲
区間 2->6を　4だけ減少▲
区間 13->9を　5だけ増加
区間 9->5を　4だけ増加
区間 15->16を　3だけ増加
区間 14->15を　5だけ増加
区間 13->14を　7だけ増加
区間 12->11を　5だけ減少▲
区間 10->11を　3だけ増加
区間 9->10を　1だけ増加
区間 6->5を　4だけ減少▲
区間 3->4を　4だけ増加
区間 2->3を　4だけ増加

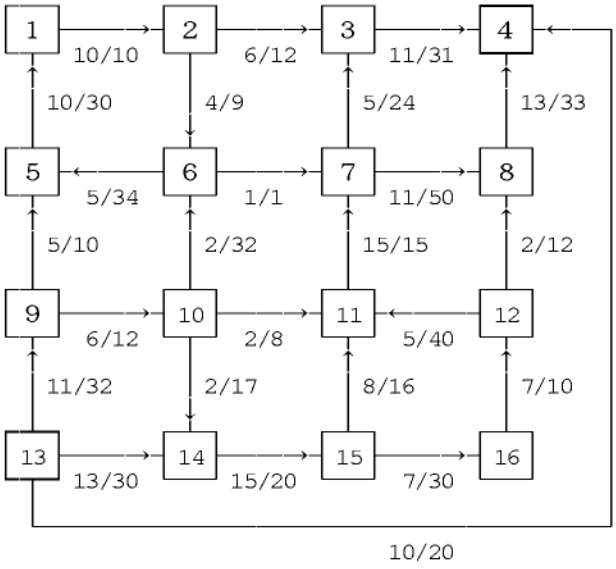
§11. 11 コンパイルと実行(続き)

```
==total flow==
13 4
16 12 10 10
12 8 10 12
8 4 21 33
:
6 5 1 34
3 4 15 31
2 3 10 12
1 2 10 10
```

最後に，極大フロー反映後の区間データが表示される.
このデータを再度プログラムに入力し，多層ネットワークが構築できなくなるまで繰り返すことにより，最大流を求めることができる.
ただし，この例では最大流を実現している.

§11. 12 例題

13→4に現流量10，容量20のバイパスを設けた．最大流が求まるまでの，各段階での極大フローを示せ．
ヒント:この場合，1回の実行では最大流が求まらない．



§11. 13 今日の課題

リテラシとしての情報教育(専らWindowsの使い方)は小中高校生でもやっている．

後期プログラミング演習は，考える訓練，知的な創造のための実習としての情報教育．

以上を踏まえ，プログラミング演習を通じて各自が得た知見を述べよ．最低文字数は1000字．

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

今日はここまで