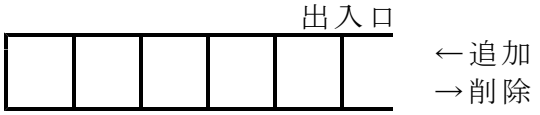
スタック

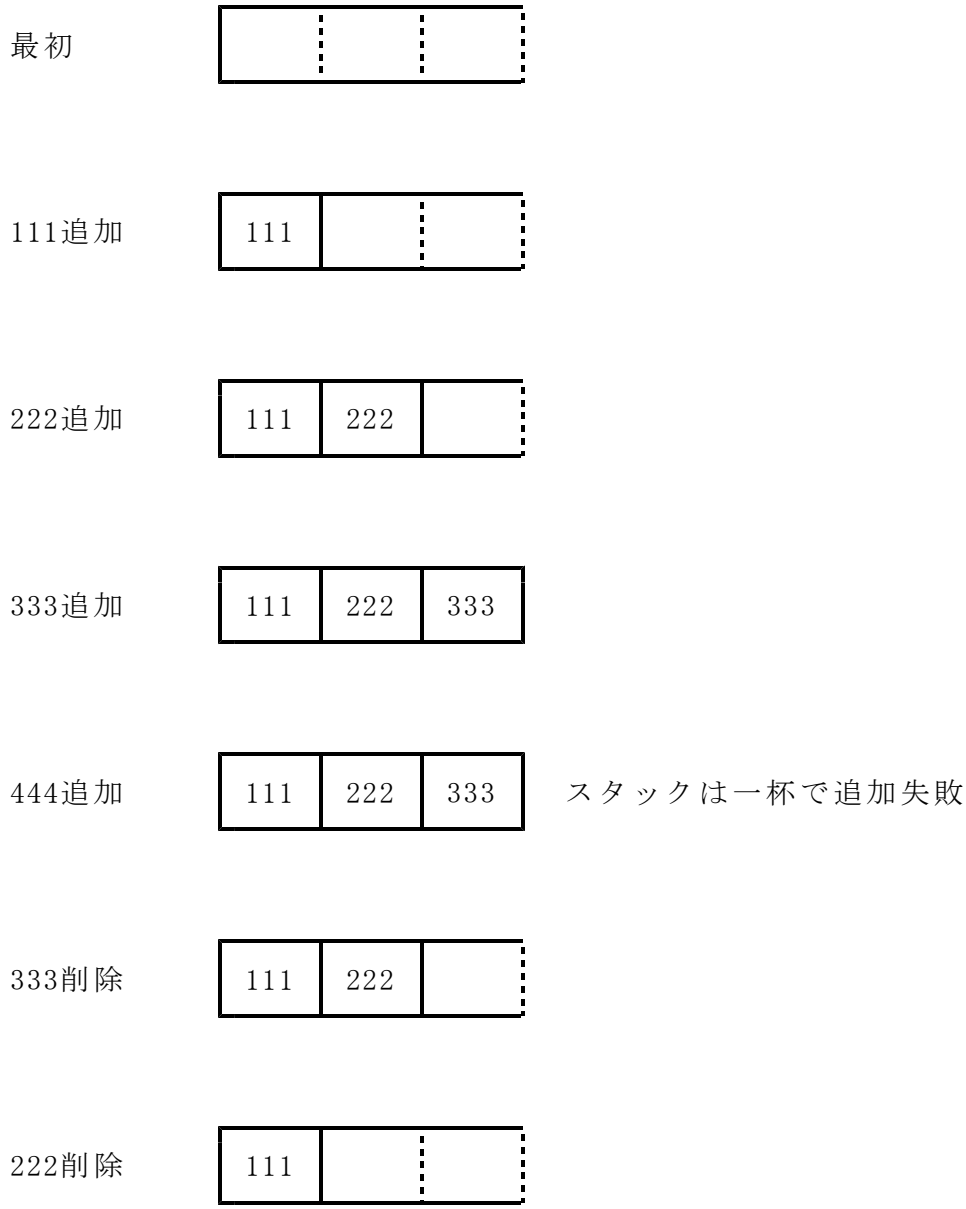
０．目次

１．スタック

スタックは、データの出し入れが一カ所で行われ、操作は追加と削除ができるデータ構造をいう。

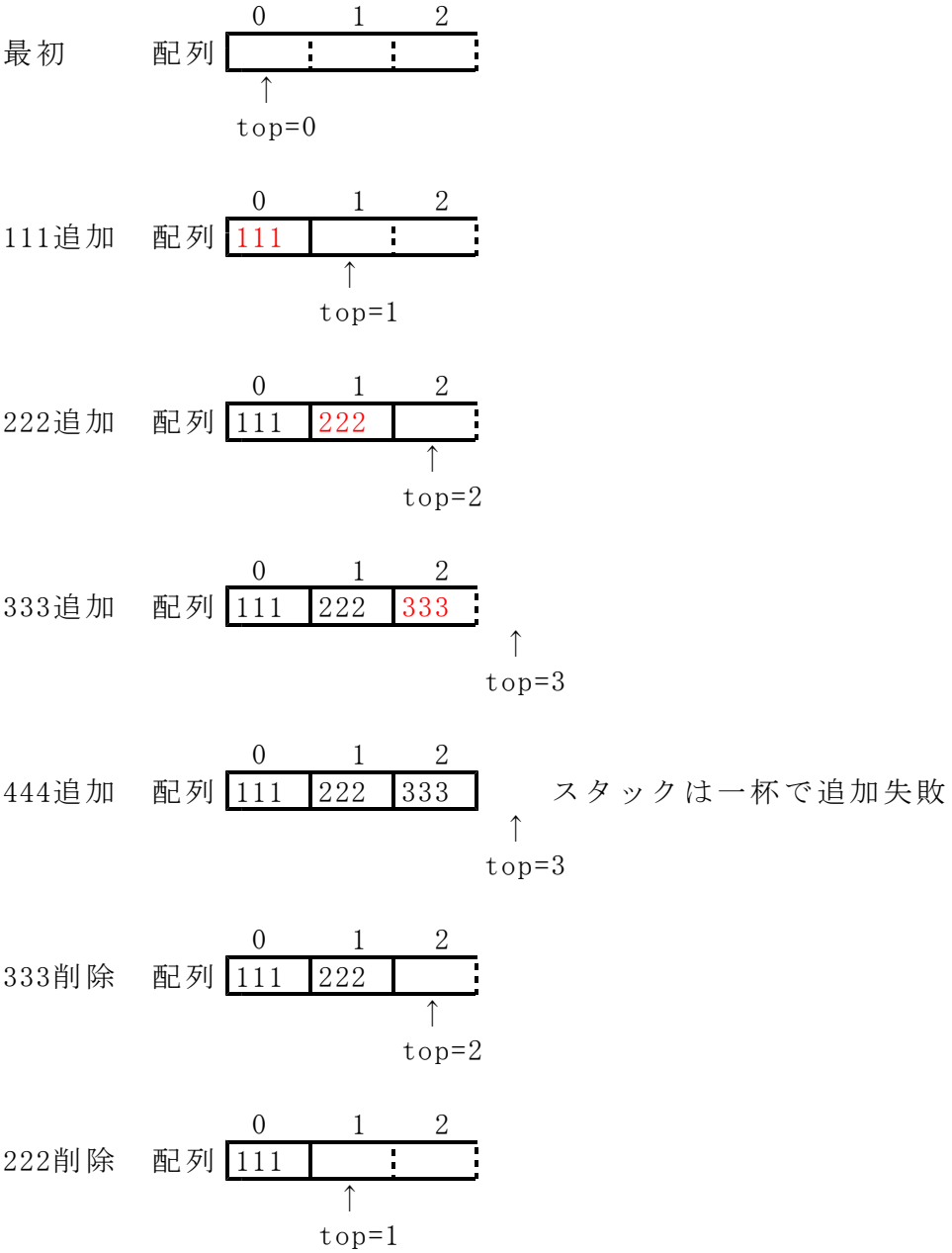


●スタック操作



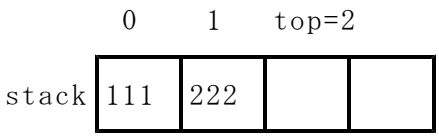
１．１　配列によるスタックの実現

配列での実現：変数topでスタックに保存されるデータの位置を示す。
変数topの初期値は、0とする。
データを保存する場合、変数topが指す位置にデータが保存され、変数topの値を1増やす。
データを取り出す場合、変数topの値を1減らし、変数topの指す位置からデータが取り出される。
変数topの値は、要素の数でもある。



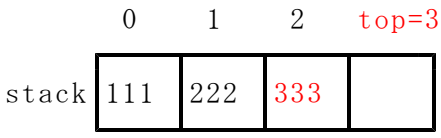
●スタック（配列表現）へ要素の追加

追加前



追加データ：333

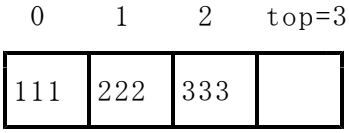
追加後




```
stack[top] = x; top++;
```

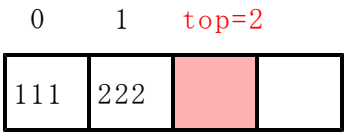
●スタック（配列表現）から要素の削除

削除前



削除データ：333

削除後




```
top--;
```

●プログラム（st111.c）

スタックの操作を行うプログラム

操作1：正整数を読み込みスタックに追加。
操作0：スタックからデータを取り出す。
操作-1：終了。

```
1  /* << st111.c >> */
2  #include <stdio.h>
3  #define SMAX 2 /* スタックに保存できるデータ数。*/
4
5  int main() {
6    int i,
7        op,           /* 操作。*/
8        stack[SMAX], /* スタックを実現する配列。*/
9        top,          /* スタックに保存されるデータの位置。*/
10       x;            /* データ。*/
11
12   /* スタックの初期化。*/
13   top = 0;
14
15   /* 操作。*/
16   while( 1 ) {
17     /* スタックの表示。*/
18     printf("スタック：");
19     for( i=0; i<top; i++ ) { printf("%d ",stack[i]); }
20     printf("¥n");
21
22     /* 操作入力。*/
23     printf("操作："); scanf("%d",&op);
24     switch( op ) {
25
26       case 1: /* 追加データを読み込む。*/
27           printf("データ:"); scanf("%d",&x);
28           /* スタックにデータを追加。*/
29           if( top < SMAX ) {
30             stack[top] = x; top++;
31           } else {
32             printf("スタックが一杯です¥n");
33           }
34           break;
35
36        case 0: /* スタックからデータを削除。*/
37           if( top > 0 ) {
38             top--;
39           } else {
40             printf("スタックが空です¥n");
41           }
42           break;
```

```
43
44          case -1: /* 終了。*/
45              exit(0);
46      }
47    }
48  }
```

実行結果

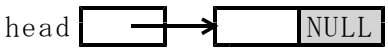
```
% cc st111.c
% ./a.out
スタック：
操作：1
データ:111
スタック：111

操作：1
データ:222
スタック：111 222

操作：1
データ:333
スタックが一杯です
スタック：111 222

操作：0
スタック：111
操作：0
スタック：

操作：0
スタックが空です
スタック：

操作：-1
```

１．２　再帰的なデータ構造によるスタックの実現

構造体を用いた再帰的なデータ構造（構造体のメンバに他の構造体を指すポインタを含む）で実現する。headは、スタックの先頭を指す。
NULLはリストの最後を意味する。

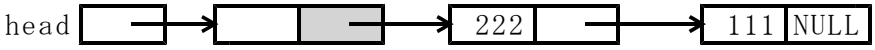
●スタック操作

最初



111追加

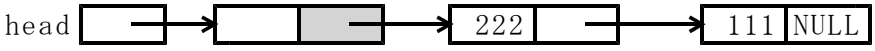
222追加



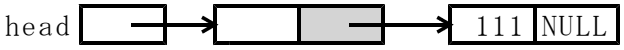
333追加

(注意)構造体が確保できる間、追加操作が可能。

333削除

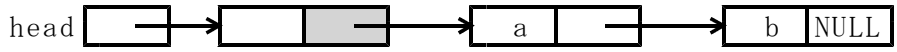


222削除

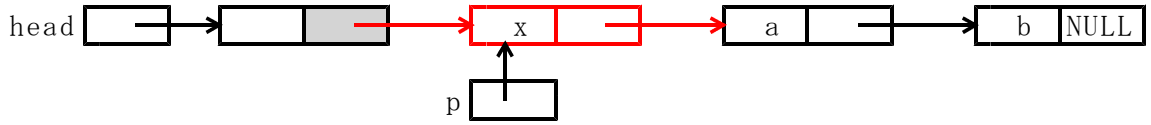


●スタック（再帰的表現）へ要素の追加

追加前



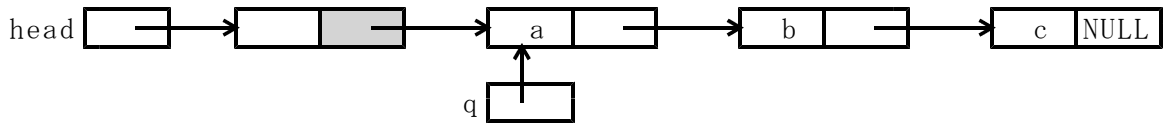
追加後　追加データ：x




```
p = (struct NODE *)malloc(sizeof(struct NODE));
p->info = x;
p->next = head->next;
head->next = p;
```

●スタック（再帰的表現）から要素の削除

削除前



削除後


```
q = head->next;
head->next = q->next;
```

●プログラム（st121.c）

スタックの操作を行うプログラム
操作1：正整数を読み込みスタックに追加。
操作0：スタックからデータを取り出す。
操作-1：終了。

```
1  /* << st121.c >> */
2  #include <stdio.h>
3  #include <stdlib.h>
4  struct NODE {
5    int info;
6    struct NODE *next;
7  };
8
9  int main() {
10   int op, /* 操作。*/
11       x;  /* データ。*/
12   struct NODE *head, /* スタックの先頭を指すポインタ変数。*/
13               *p,    /* 作業用ポインタ変数。*/
14               *q;    /* 作業用ポインタ変数。*/
15
16   /* リストヘッドの作成。*/
17   head = (struct NODE *)malloc(sizeof(struct NODE));
18   head->next = NULL;
19
20   while( 1 ) {
21     /* スタックの表示。*/
22     printf("スタック：");
23     p = head->next;
24     while( p != NULL ) { printf("%d ",p->info); p = p->next; }
25     printf("¥n");
26
27     /* 操作入力。*/
28     printf("操作："); scanf("%d",&op);
29     switch( op ) {
30
31       case 1: /* 追加データを読み込む。*/
32           printf("データ:"); scanf("%d",&x);
33           /* スタックにデータを追加。*/
34           p = (struct NODE *)malloc(sizeof(struct NODE));
35           p->info = x;
36           p->next = head->next;
37           head->next = p;
38           break;
39
40        case 0: /* スタックからデータを削除。*/
41           if( head->next != NULL ) {
42             q = head->next;
43             head->next = q->next;
```

```
44              } else {
45                printf("スタックは空です¥n");
46              }
47              break;
48
49          case -1: /* 終了。*/
50              exit(0);
51      }
52    }
53  }
```

実行結果

```
% cc st121.c
% ./a.out
スタック：

操作：1
データ:111
スタック：111

操作：1
データ:222
スタック：222 111

操作：1
データ:333
スタック：333 222 111

操作：0
スタック：222 111

操作：0
スタック：111

操作：0
スタック：

操作：0
スタックは空です
スタック：

操作：-1
```

１．３　地図情報処理

0と1で与えられた地図情報から島(1が上下左右に隣接する領域)の個数を求める問題を考察する。なお、0は海を表し、1は陸を表す。

地図情報

```
000001100000
000111111000
000110000000
011000111100
000000001100
011100000000
000100000000
```

この場合、島の個数は４となる。

●考え方

① 地図のすべての点について②を行う。

② 点の値が1の場合、隣接する点で値が1となる点をひとつの島として登録する。登録した島の点は、再度登録されないように値を変更しておく。

●動作

（１）地図情報

```
000001100000
000111111000
000110000000
011000111100
000000001100
011100000000
000100000000
```

（２）地図情報

```
000002200000
000222222000
000220000000
011000111100
000000001100
011100000000
000100000000
```

（３）地図情報

```
000002200000
000222222000
000220000000
033000111100
000000001100
011100000000
000100000000
```

（４）地図情報

```
000002200000
000222222000
000220000000
033000444400
000000004400
011100000000
000100000000
```

（５）地図情報

```
000002200000
000222222000
000220000000
033000444400
000000004400
055500000000
000500000000
```

●プログラム（st131.c）

```
/* << st131.c >> */
#include <stdio.h>
#include <stdlib.h>
#define M 30 /* 行の最大値。*/
#define N 30 /* 列の最大値。*/

int main() {
  int a[M+2][N+2], /* 地図情報を保存する配列。*/
      i,j,k,l,
      m,           /* 行数。*/
      n,           /* 列数。*/
      num,         /* 島の番号。*/
      ti,tj;
  int si[M*N],     /* スタック関連の配列。*/
      sj[M*N],     /* スタック関連の配列。*/
      top;         /* スタック関連の変数。*/

  /* 地図情報（m:行数、n:列数）の読み込み。*/
  scanf("%d%d",&m,&n);
  if( (m <= 0)||(m > M)||
      (n <= 0)||(n > N) ) { exit; }
  for( i=1; i<=m; i++ ) {
    for( j=1; j<=n; j++ ) { scanf("%1d",&a[i][j]); }
  }

  /* 与えられた地図情報の周囲を海としておくと、処理が簡潔になる。*/
  for( j=0; j<=n+1; j++ ) {
    a[0][j] = 0; a[m+1][j] = 0;
  }
  for( i=1; i<=m; i++ ) {
    a[i][0] = 0; a[i][n+1] = 0;
  }

  /* 地図情報の表示。*/
  printf("探索前¥n");
  for( i=1; i<=m; i++ ) {
    for( j=1; j<=n; j++ ) { printf("%d",a[i][j]); }
    printf("¥n");
  }

  /* 初期設定。*/
  num = 1;

  /* 地図情報のすべての点について、探索する。*/
  for( i=1; i<=m; i++ ) {
    for( j=1; j<=n; j++ ) {
      /* 点(i,j)が0、すなわち、海の場合、スキップ。*/
      if( a[i][j] == 0 ) { continue; }
```

```
49
50        /* 点(i,j)が1以外、すなわち、すでに探索済みの場合、スキップ。*/
51        if( a[i][j] > 1 ) { continue; }
52
53        /* 点(i,j)が1、すなわち未探索の場合、島を確定する。*/
54        num++;
55
56        /* スタックの初期化。*/
57        si[0] = i; sj[0] = j; top = 1;
58
59        /* スタック(si[*],sj[*])を使って、点(ti,tj)に隣接する陸の
60           点をすべて同じ島の番号に設定する。*/
61        while( top > 0 ) {
62          ti = si[top-1]; tj = sj[top-1]; top--;
63          a[ti][tj] = num;
64
65          if( a[ti-1][tj] == 1 ) {
66            si[top]=ti-1; sj[top]=tj; top++;
67          }
68          if( a[ti+1][tj] == 1 ) {
69            si[top]=ti+1; sj[top]=tj; top++;
70          }
71          if( a[ti][tj-1] == 1 ) {
72            si[top]=ti; sj[top]=tj-1; top++;
73          }
74          if( a[ti][tj+1] == 1 ) {
75            si[top]=ti; sj[top]=tj+1; top++;
76          }
77        }
78
79        /* 途中結果の表示。*/
80        printf("途中結果¥n");
81        for( k=1; k<=m; k++ ) {
82          for( l=1; l<=n; l++ ) { printf("%1d",a[k][l]); }
83          printf("¥n");
84        }
85      }
86    }
87
88    /* 最終結果の表示。*/
89    printf("探索後¥n");
90    for( i=1; i<=m; i++ ) {
91      for( j=1; j<=n; j++ ) { printf("%1d",a[i][j]); }
92      printf("¥n");
93    }
94    printf("島の個数：%d¥n",num-1);
95  }
```

実行結果

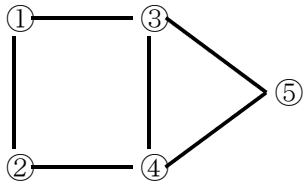
```
% cat st131.dat
7 12
0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 1 0 0
0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
% cc st131.c
% ./a.out < st131.dat
探索前
000001100000
000111111000
000110000000
011000111100
000000001100
011100000000
000100000000
途中結果
000002200000
000222222000
000220000000
011000111100
000000001100
011100000000
000100000000
途中結果
000002200000
000222222000
000220000000
033000111100
000000001100
011100000000
000100000000
途中結果
000002200000
000222222000
000220000000
033000444400
000000004400
011100000000
000100000000
途中結果
000002200000
000222222000
000220000000
033000444400
```

```
000000004400
055500000000
000500000000
探索後
000002200000
000222222000
000220000000
033000444400
000000004400
055500000000
000500000000
島の個数：4
```

１．４　問題

問題１　グラフ探索問題

与えられたグラフにおいて、開始節点から到達できる節点をすべて求める問題を考察する。



●考え方

節点に隣接する節点を隣接節点とよぶ。

（手順a）始点を訪問済節点とし、スタックに格納する。
他の節点は未訪問節点とする。

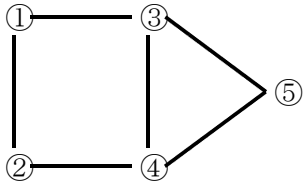
（手順b）スタックから節点をひとつ取り出す。
この節点の隣接節点が未訪問の場合、
訪問済節点とした後、スタックに格納する。
訪問済の場合、何もしない。

（手順c）手順bをスタックが空になるまで繰り返す。

（手順d）訪問済節点を表示する。

●動作　開始節点を節点1とする。

（１）手順a



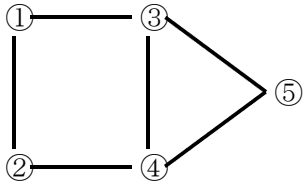
スタック

①

節点	
①	訪問済
②	
③	
④	
⑤	

未訪問節点は空欄とする。

（２）手順b



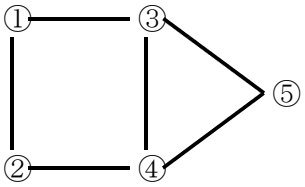
スタック

③
②

節点	
①	訪問済
②	訪問済
③	訪問済
④	
⑤	

未訪問節点は空欄とする。

（３） 手順b



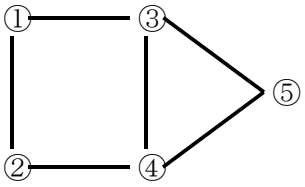
スタック

⑤
④
②

節点	
①	訪問済
②	訪問済
③	訪問済
④	訪問済
⑤	訪問済

未訪問節点は空欄とする。

（４） 手順b



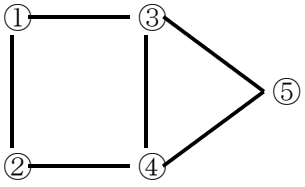
スタック

④
②

節点	
①	訪問済
②	訪問済
③	訪問済
④	訪問済
⑤	訪問済

未訪問節点は空欄とする。

（５） 手順b



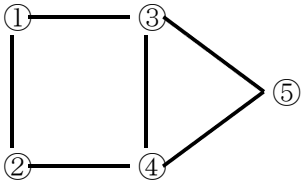
スタック

②

節点	
①	訪問済
②	訪問済
③	訪問済
④	訪問済
⑤	訪問済

未訪問節点は空欄とする。

（６） 手順b　終了



スタック

節点	
①	訪問済
②	訪問済
③	訪問済
④	訪問済
⑤	訪問済

未訪問節点は空欄とする。

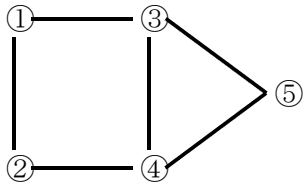
●プログラム（st141.c）

```
1  /* << st141.c >> */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #define N 99 /* 節点の最大数。*/
5  struct NODE {
6    int info;
7    struct NODE *next; /* nextはポインタ変数。*/
8  };
9
10 int main() {
11   struct NODE *head[N+1],   /* 節点iに隣接する節点のリスト。*/
12               *p,           /* 作業用ポインタ変数。*/
```

```
13              *q;             /* 作業用ポインタ変数。*/
14  int d0,d1,      /* (d0,d1)は辺を意味する。*/
15      i,
16      n,          /* 節点の個数。*/
17      s,          /* 開始節点。*/
18      v[N+1];     /* v[i]=0 節点iが未訪問、
19                     v[i]=1 節点iが訪問済を意味する。*/
20  int stack[99], /* スタック関係の配列。*/
21      top;        /* スタック関係の変数。*/
22
23  /* 節点の個数nを入力。*/
24  scanf("%d",&n);
25  if( (n <= 0)||(n > N) ) { exit; }
26
27  /* 各節点のリストヘッドの作成。*/
28  for( i=1; i<=n; i++ ) {
29    head[i] = (struct NODE *)malloc(sizeof(struct NODE));
30    head[i]->info = i;
31    head[i]->next = NULL;
32  }
33
34  /* グラフの作成。*/
35  while( 1 ) {
36    /* 辺の入力。(d0,d1)は辺を意味する。*/
37    scanf("%d%d",&d0,&d1);
38    if( (d0 <= 0)||(d0 > n)||(d1 <= 0)||(d1 > n) ) { break; }
39
40    q = (struct NODE *)malloc(sizeof(struct NODE));
41    q->info = d1;
42    q->next = NULL;
43
44    /* 入力した辺(d0,d1)をリストの先頭に挿入する。*/
45    p = head[d0]->next;
46    head[d0]->next = ①      ;
47    q->next = ②      ;
48  }
49
50  /* グラフの表示。*/
51  printf("グラフ¥n");
52  for( i=1; i<=n; i++ ) {
53    printf("%d: ",i);
54    p = head[i]->next;
55    while( p != NULL ) {
56      printf("%d ",p->info);
57      p = p->next;
58    }
59    printf("¥n");
60  }
61
62  /* 到達節点の探索。*/
```

```
63
64    while( 1 ) {
65      /* 開始節点の入力。*/
66      scanf("%d",&s);
67      if( s == 0 ) { break; }
68
69      /* 初期設定。*/
70      for( i=1; i<=n; i++ ) { /* すべての節点を未訪問に設定。*/
71        v[i] = 0;
72      }
73      stack[0] = s; top = 1; /* topは、次に格納する位置を意味する。
74                                格納されている節点数でもある。*/
75      v[s] = 1; /* 開始節点sを訪問済にする。*/
76
77      /* スタック操作。*/
78      while( top > 0 ) {
79        /* 節点v[top-1]に隣接する節点で、未訪問であればスタックに格納。*/
80        p = head[stack[top-1]]->next; top--;
81        while( p != NULL ) {
82          /* 節点p->infoが未訪問の場合。*/
83          if( v[p->info] == ③      ) {
84            stack[top] = ④          ;
85            v[stack[top]] = ⑤      ;
86            top++;
87          }
88          p = p->next;
89        }
90      }
91
92      /* 到達節点の出力。*/
93      printf("開始節点：%d  ",s);
94      printf("到達節点：");
95      for( i=1; i<=n; i++ ) {
96        if( v[i] == 1 ) { printf(" %d",i); }
97      }
98      printf("¥n");
99    }
100 }
```

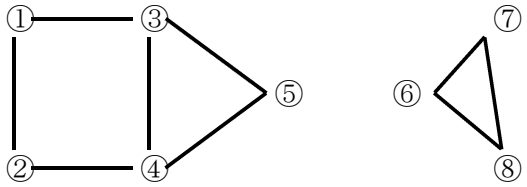
●グラフ１



実行結果

```
% cat g1.dat
5
1 2  1 3
2 1  2 4
3 1  3 4  3 5
4 2  4 3  4 5
5 3  5 4
0 0
1
5
0
% cc st141.c
% ./a.out < g1.dat
グラフ
1: 3 2
2: 4 1
3: 5 4 1
4: 5 3 2
5: 4 3
開始節点：1  到達節点： 1 2 3 4 5
開始節点：5  到達節点： 1 2 3 4 5
```

●グラフ２



実行結果

```
% cat g2.dat
8
1 2  1 3
2 1  2 4
3 1  3 4  3 5
4 2  4 3  4 5
5 3  5 4
6 7  6 8
7 6  7 8
8 6  8 7
0 0
1
8
0
% ./a.out < g2.dat
グラフ
1: 3 2
2: 4 1
3: 5 4 1
4: 5 3 2
5: 4 3
6: 8 7
7: 8 6
8: 7 6
開始節点：1  到達節点： 1 2 3 4 5
開始節点：8  到達節点： 6 7 8
```