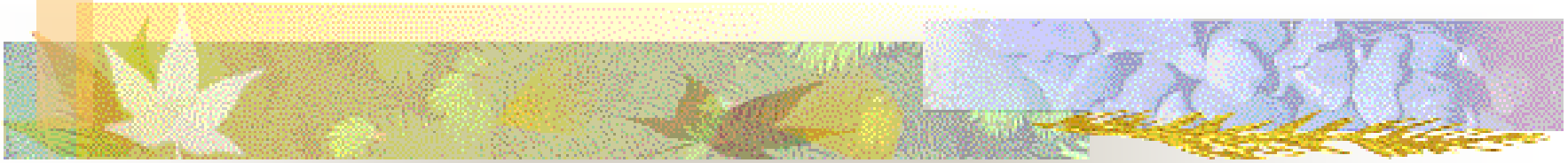
アルゴリズムとデータ構造b
9 - グラフの探索(深さ優先探索)



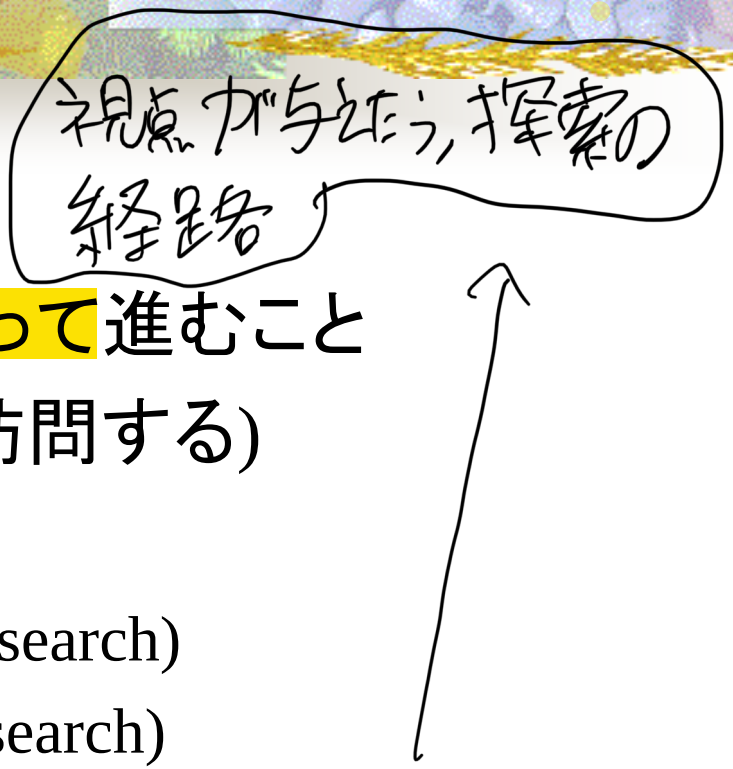
大見　嘉弘

2020/11/16

本日の内容

- グラフの探索
  - グラフの探索とは
  - 深さ優先探索

グラフの探索Search

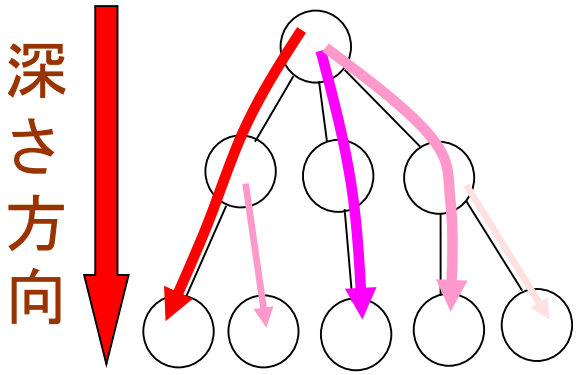


- グラフの各頂点を辺を辿って進むこと
- 通常全ての頂点を辿る(訪問する)
- グラフの探索方法
  - 深さ優先探索(depth first search)
  - 幅優先探索 (breath first search)
- 各頂点を訪問する際に頂点に何らかの操作をすることが多い(例：頂点の値を変更する)
- 典型的な目的:ある頂点(始点)からある頂点(目標点)までの経路があるかどうか調べる

深さ優先と幅優先

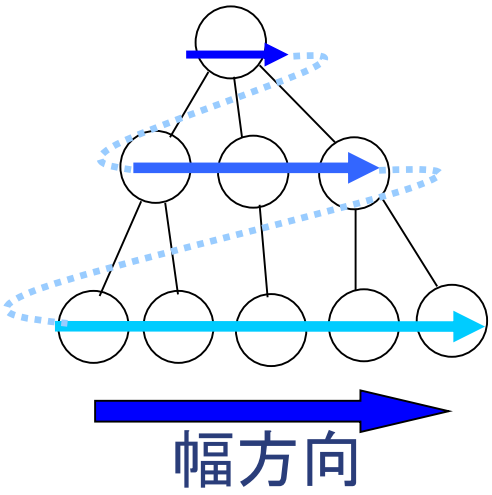
Vertical

- 縦方向(上→下)を先に辿るのが深さ優先探索
  - 縦型探索ともいう



Horizontal

- 横方向(左→右)を先に辿るのが幅優先探索
  - 横型探索ともいう

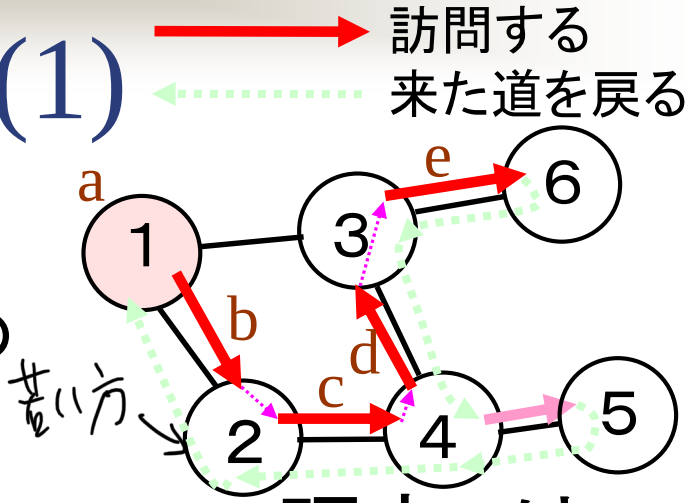


深さ優先探索

- 始点を出発し、頂点番号の若い順に縦方向(下)に行ける所まで行く
- 行き止まりになったら、元来た道を戻る
  - 他の行き先が見つかったらそこを辿る。
  - 見つからなかったら、さらに元来た道を戻る
- 始点に戻ってきて、行き場所がなくなっていたら終了する

先往下走，找最小，然后访问，访问后就去找没用，没路了就原路返回。

深さ優先探索の例(1)



a. 頂点１から開始

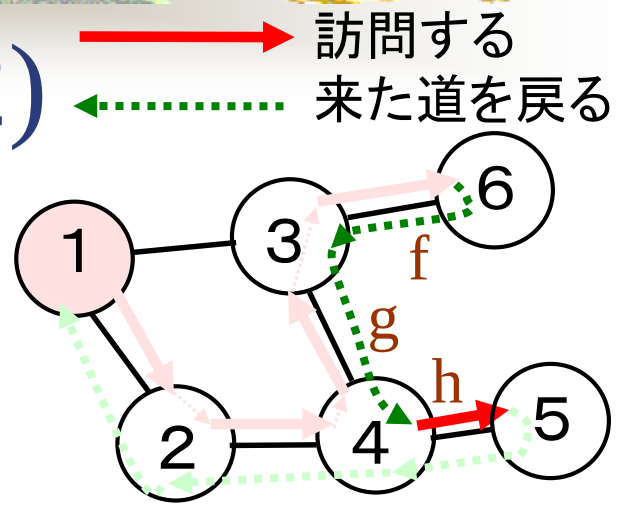
b. (頂点1に隣接していて番号の一番若い)頂点2を訪問

c. (頂点2に隣接していて番号の一番若い)頂点１は訪問済なので、次に若い頂点4を訪問

d. (頂点4に隣接していて番号の一番若い)頂点2は訪問済なので、次に若い頂点3を訪問

e. (頂点3に隣接していて番号の若い)頂点１と4は訪問済なので、次に若い頂点6を訪問

深さ優先探索の例(2)

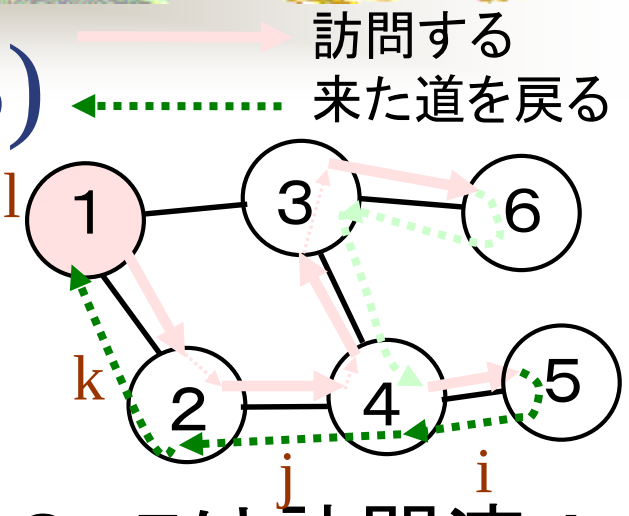


f. (頂点6に隣接している)
頂点3は訪問済なので、
訪問先がない！
そこで来た道を戻る(頂点3に戻る)

g. (頂点3に隣接している)頂点1、4、6は訪問済みで訪問先がない！
そこで来た道を戻る(頂点4に戻る)

h. (頂点4に隣接している)頂点2、3は訪問済なので、次に若い頂点5を訪問

深さ優先探索の例(3)



i. (頂点5に隣接している)頂点４は訪問済！そこで来た道を戻る(頂点4に戻る)

j. (頂点4に隣接している)頂点２、３、５は訪問済！そこで来た道を戻る(頂点2に戻る)

k. (頂点2に隣接している)頂点１、４は訪問済！そこで来た道を戻る(頂点1に戻る)

l. (頂点1に隣接している)頂点２、３は訪問済！始点に戻り、行き場所がないので終了

プログラム例(1)

- グラフは隣接行列で表す(Javaの二次元配列)
- 例:

```
int[][] g = {{0,0,0,0,0,0,0},
             {0,0,1,1,0,0,0},
             {0,1,0,0,1,0,0},
             {0,1,0,0,1,0,1},
             {0,0,1,1,0,1,0},
             {0,0,0,0,1,0,0},
             {0,0,0,1,0,0,0}};
```

添字0を使用

使わない部分

```
final int N = 6; // 頂点の数
```

- (N+1)行×(N+1)列の配列にする
  - 添字が0の部分を使わない
    (頂点1,2,3…を添字1,2,3に対応させるため)

プログラム例(2)

- 訪問フラグを設ける
  - N+1個の一次元配列(添字0は未使用)

```
int[] v = new int[N+1];
```

  - プログラムの動作途中で各頂点を訪問済がどうかを示す
    - 値が1: その頂点は訪問済
    - 値が0: その頂点は未訪問
  - 訪問フラグの初期化(最初は全部0)

```
for (int i = 1; i <= N; i++) { v[i] = 0; }
```

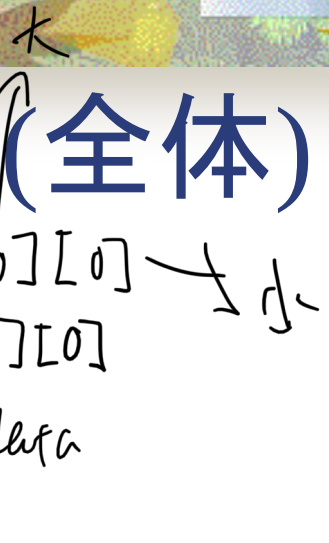
プログラム例(3)　実際の処理部

```
void visit(int i){ //頂点iを訪問
    int j;
    v[i]=1; //頂点iの訪問フラグを1に
    for(j=1;j<=N;j++) {
        if (g[i][j]==1 && v[j]==0) {
            System.out.println(
              i+"から"+j+"を訪問");
            visit(j); //頂点jを訪問
        }
    }
}
public static void main(String args[]) {
    visit(1); //まず最初に頂点1を訪問
}
```

頂点jは未訪問

頂点iと頂点jは隣接している

プログラムリスト(全体)




```
public class GraphDepthFirstSearch {
     int[][] g = {{0,0,0,0,0,0,0},
                  {0,0,1,1,0,0,0},
                  {0,1,0,0,1,0,0},
                  {0,1,0,0,1,0,1},
                  {0,0,1,1,0,1,0},
                  {0,0,0,0,1,0,0},
                  {0,0,0,1,0,0,0}};
     final int N = 6; // 頂点の数
     int[] v = new int[N+1]; // 訪問フラグ
     void visit(int i){ //頂点iを訪問
          int j;
          v[i]=1; //頂点iの訪問フラグを1に
          for(j=1;j<=N;j++) {
               if (g[i][j]==1 && v[j]==0) {
                    System.out.println(i+"から"+j+"を訪問");
                    visit(j);
               }
          }
     }
     public static void main(String args[]) {
          GraphDepthFirstSearch prog = new GraphDepthFirstSearch();
          prog.start();
     }
     public void start() {
          for (int i=1; i<=N; i++) { v[i]=0; }
          visit(1);
     }
}
```

プログラム例のトレース(1)

i	j	v	1	2	3	4	5	6	備考
			0	0	0	0	0	0	
1			1	0	0	0	0	0	頂点1訪問
	1								v[1]=1
	2								g[1][2]=1,v[2]=0 visit(2)呼ぶ
2			1	1	0	0	0	0	頂点2訪問
	1								v[1]=1
	2								v[2]=1
	3								g[2][3]=0
	4								g[2][4]=1,v[4]=0 visit(4)呼ぶ
4			1	1	0	1	0	0	頂点4訪問
	1								v[1]=1
	2								v[2]=1
	3								g[4][3]=1,v[3]=0 visit(3)呼ぶ

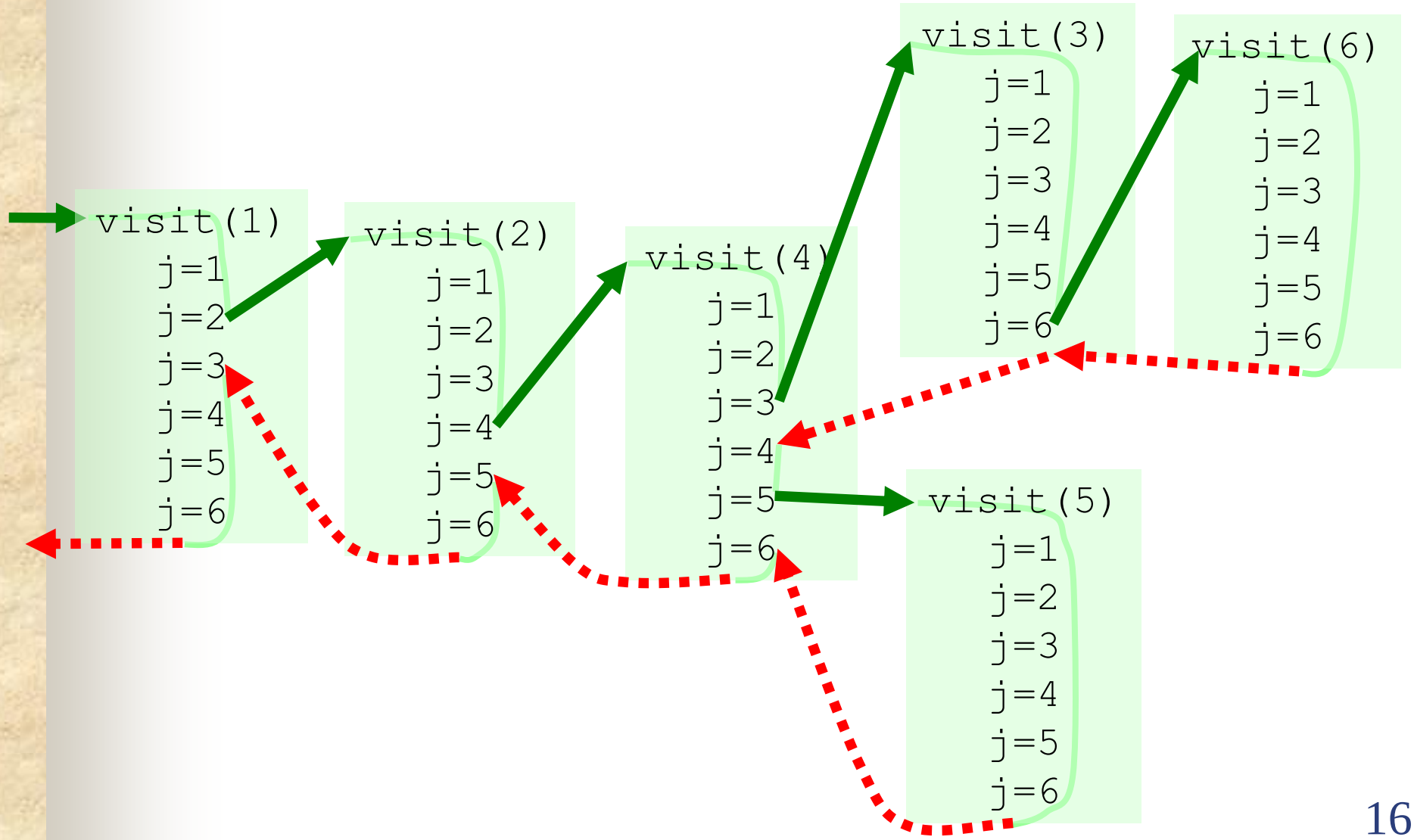
プログラム例のトレース(2)

i	j	v	1	2	3	4	5	6	備考
3			1	1	1	1	0	0	頂点3訪問
	1								v[1]=1
	2								v[2]=1
	3								v[3]=1
	4								v[4]=1
	5								g[3][5]=0
	6								g[3][6]=1,v[6]=0 visit(6)呼ぶ
6			1	1	1	1	0	1	頂点6訪問
	1								v[1]=1
	…								
	4								v[4]=1
	5								g[6][5]=0
	6								v[6]=1
									visit(6)終了,visit(3)に戻る
3									visit(3)終了,visit(4)に戻る

プログラム例のトレース(3)

i	j	v	1	2	3	4	5	6	備考
4	4								v[4]=1
	5								g[4][5]=1,v[5]=0 visit(5)呼ぶ
5			1	1	1	1	1	1	頂点5訪問
	1								v[1]=1
	…								
	6								v[6]=1
									visit(5)終了,visit(4)に戻る
4	6								v[6]=1
									visit(4)終了,visit(2)に戻る
2	5								v[5]=1
	6								v[6]=1
									visit(2)終了,visit(1)に戻る
1	3								v[3]=1
	…								
	6								v[6]=1
									visit(1)終了,プログラム終了

visitメソッドの呼び出し順序



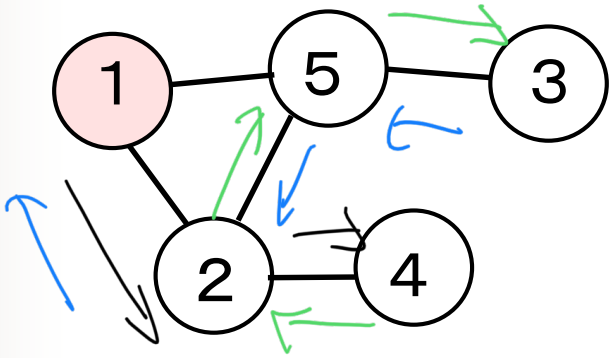
プログラムリスト2(論理型変数を使用した場合)

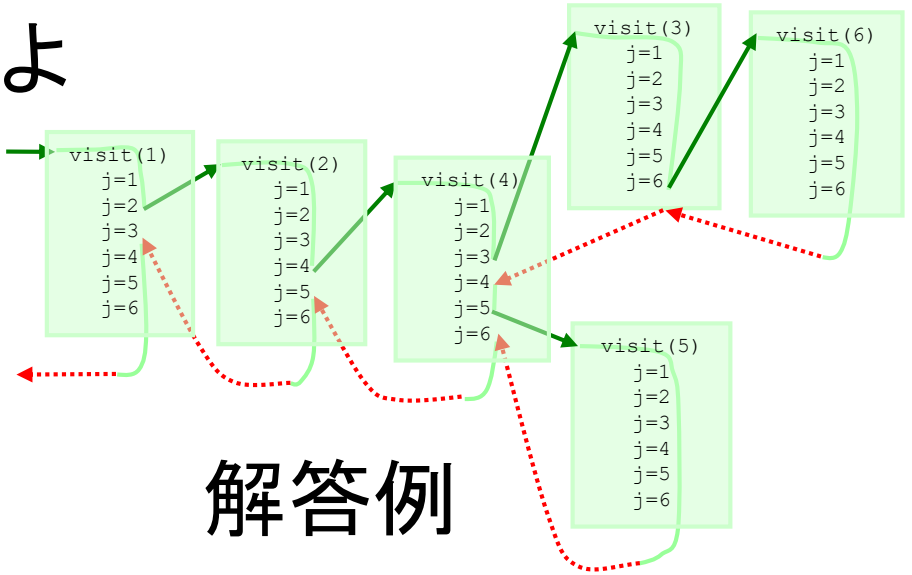
```
public class GraphDepthFirstSearch2 {
    final boolean F=false, T = true;
    boolean[][] g = {{F,F,F,F,F,F,F},
                     {F,F,T,T,F,F,F},
                     {F,T,F,F,T,F,F},
                     {F,T,F,F,T,F,T},
                     {F,F,T,T,F,T,F},
                     {F,F,F,F,T,F,F},
                     {F,F,F,T,F,F,F}};
    final int N = 6; // 頂点の数
    boolean[] v = new boolean[N+1]; // 訪問フラグ
    void visit(int i){ //頂点iを訪問
        int j;
        v[i]=true; //頂点iの訪問フラグを1に
        for(j=1;j<=N;j++) {
            if (g[i][j] && !v[j]) { // !v[j] は v[j] == false でも良い
                System.out.println(i+"から"+j+"を訪問");
                visit(j);
            }
        }
    }
    public static void main(String args[]) {
        GraphDepthFirstSearch2 prog = new GraphDepthFirstSearch2();
        prog.start();
    }
    public void start() {
        for (int i=1; i<=N; i++) { v[i]=false; }
        visit(1);
    }
}
```

宿題

	1	2	3	4	5
1	0	0	0	0	0	0
2	0	0	1	0	0	1
3	0	1	0	0	1	1
4	0	0	0	0	0	1
5	0	0	1	0	0	0
	0	1	1	1	0	0

- 下図のグラフを深さ優先探索で辿れ
  解答例: 1→5→4→3→2 1→2→4→5→3
- 下図のグラフを深さ優先探索のプログラム例で探索した場合のvisitメソッドの呼び出し順序を図示せよ





解答例