強連結成分分解＆トポロジカルソート

北海道大学情報理工学コースB3　大泉翼

目次

・グラフの話

-基礎

-dfs

・トポロジカルソート

・強連結成分分解

・問題への応用

目次

・グラフの話

-基礎

-dfs

・トポロジカルソート

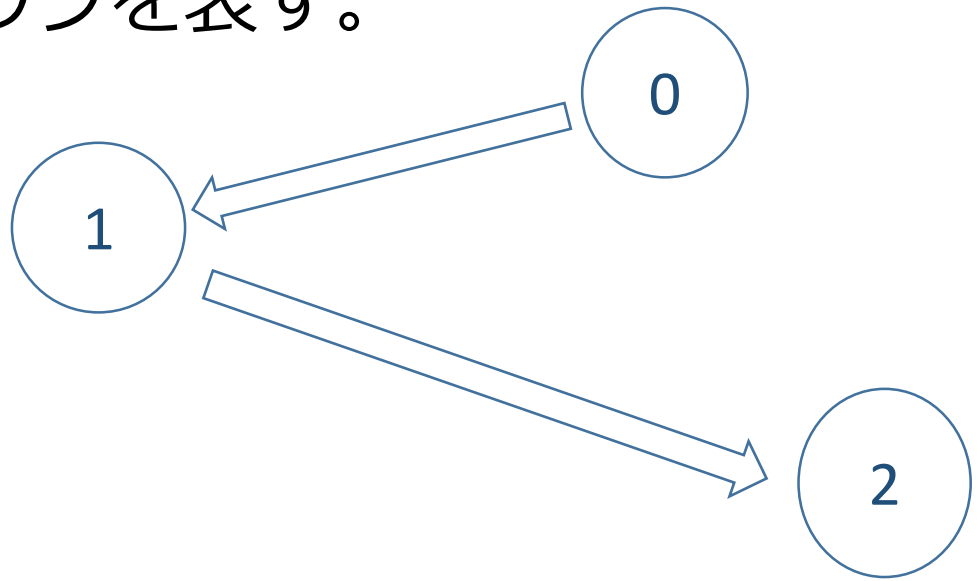
・強連結成分分解

・問題への応用

グラフの基礎

- 点の集合Vと2点間を結ぶ辺の集合Eのペアで、G = (V, E) と表す。
- 例は有向グラフである。

ex) V = {0, 1, 2} E = {(0, 1), (1, 2)}とすると以下のようなグラフを表す。



グラフの基礎

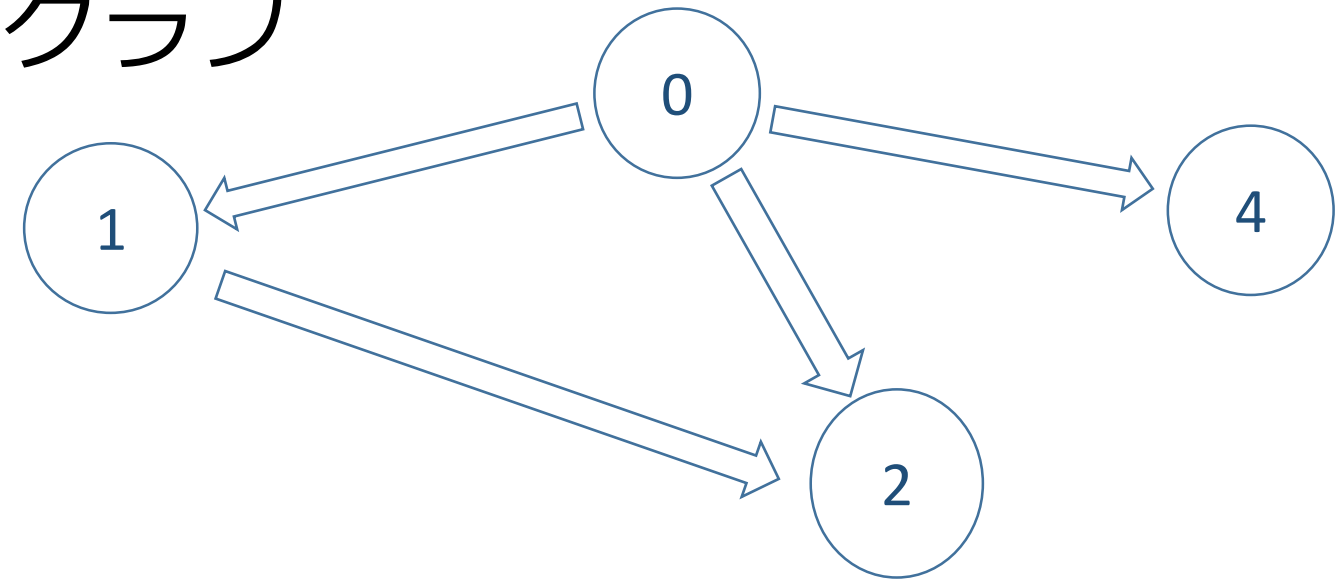
- 閉路

--始点と終点が同じ路のこと

- 有効非巡回グラフ(有効無閉路グラフ)(DAG)

--閉路のない有向グラフ

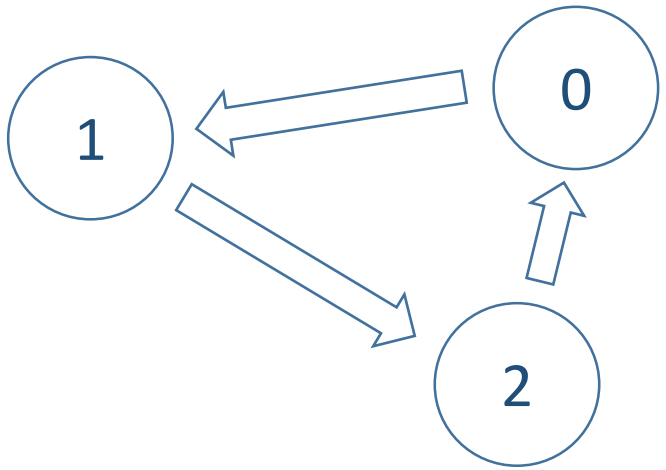
ex)



グラフの基礎

- 連結
- 無向グラフにおいて、 すべての頂点間で互いに行き来できる
- 強連結
- 有向グラフにおいて、 すべての頂点間で互いに行き来できる

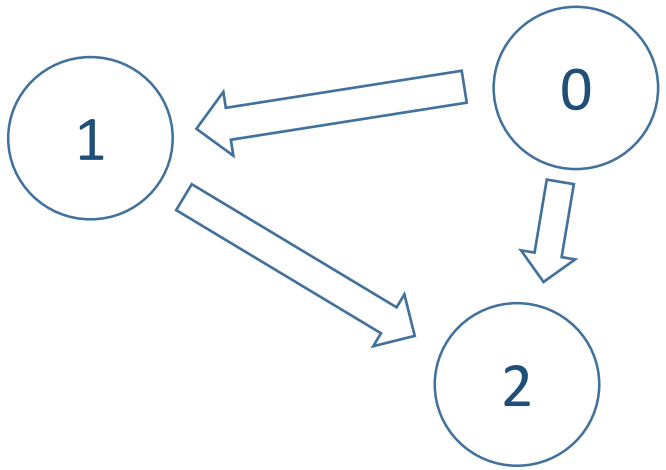
ex)



競技プログラミングにおけるグラフの表現

- 大きく分けて二種類ある
- 隣接行列vs隣接リスト
- 以下のグラフをプログラム上で表現することを考える

V = {0, 1, 2}, E = {{0, 1}, {0, 2}, {1, 2}}

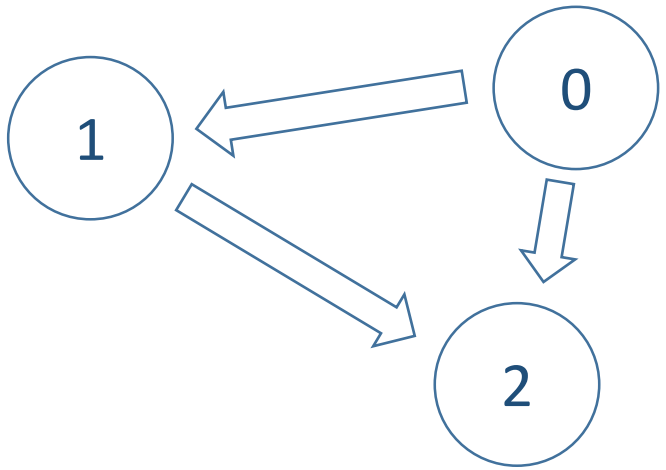


隣接行列

・二次元行列(C++なら二次元配列もしくは二次元vector)で表現

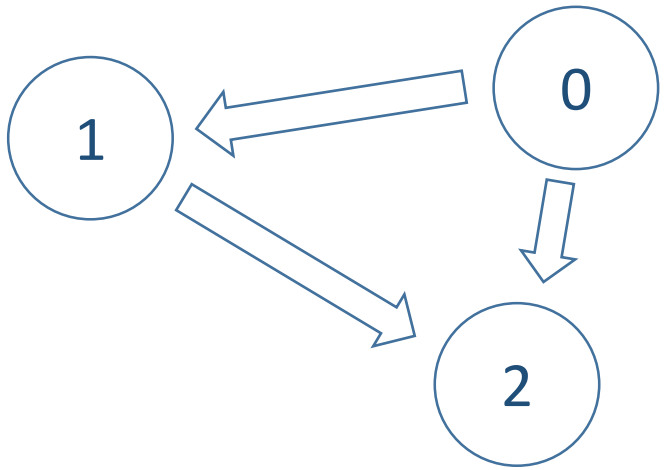
$$G = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$g_{i,j} = \begin{cases} 0\ (i \to j\text{に辺がある}) \\ 1\ (i \to j\text{に辺がある}) \end{cases}$$



ソースコード

```
bool G[3][3];
memset(G, false, sizeof(G));

G[0][1] = true;
G[0][2] = true;
G[1][2] = true;
```



隣接リスト

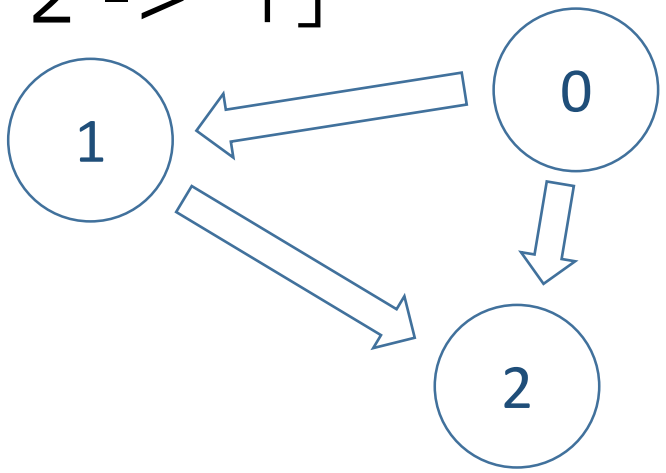
・リストで表現(C++ならSTLのvectorなどを使う)

・各頂点について、自分から出ている辺の行先をリストに格納する

0 -> 「1, 2」

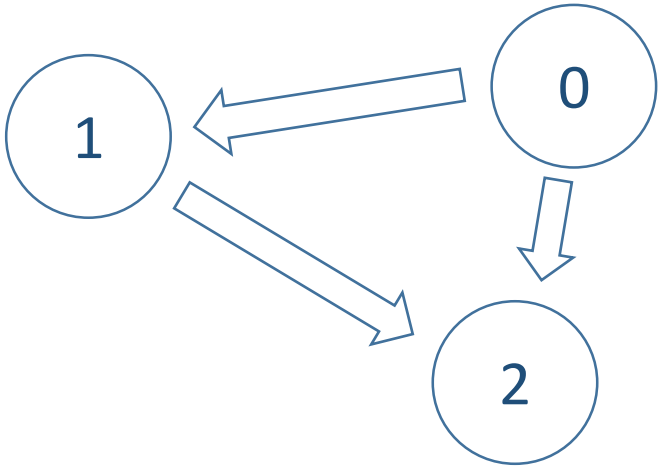
1 -> 「2」

2 -> 「」



ソースコード

```
vector<vector<int> > G(3);

G[0].push_back(1);
G[0].push_back(2);
G[1].push_back(2);
```



結局どっちがいいの

場合による

目次

・グラフの話

-基礎

-dfs

・トポロジカルソート

・強連結成分分解

・問題への応用

グラフ上のDFS

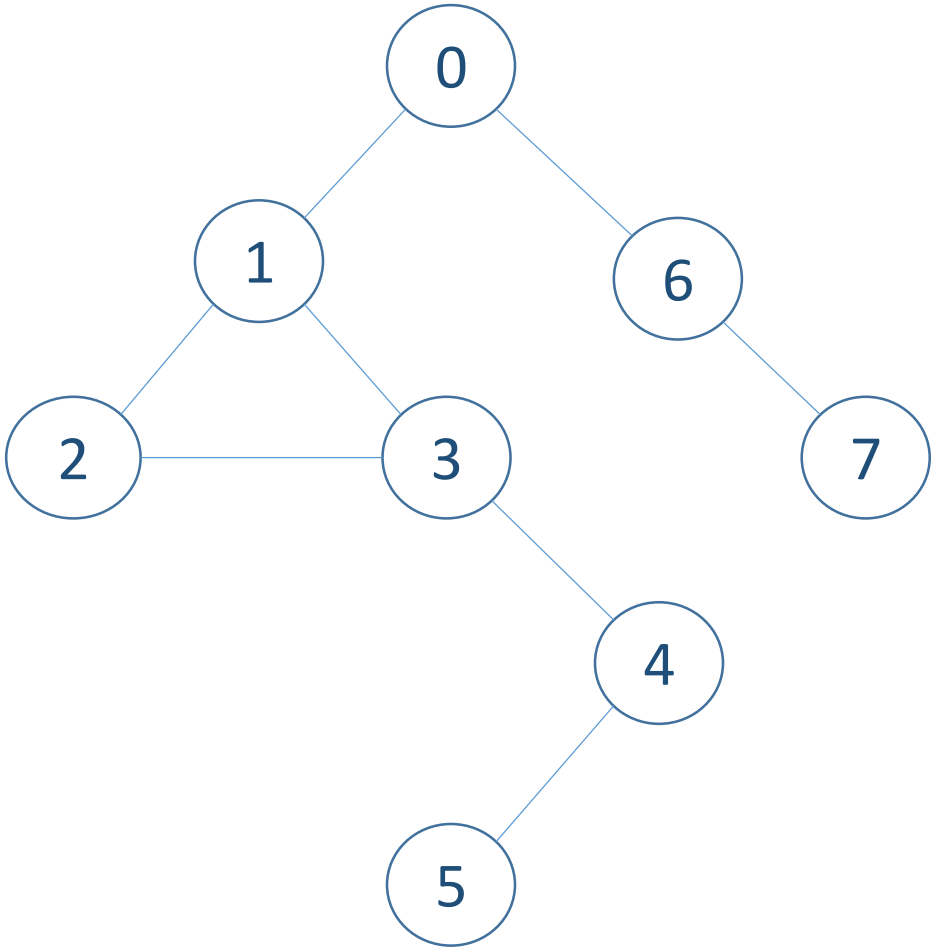
・与えられたグラフ上の頂点をちょうど一度ずつ訪問することを考える

・訪れた頂点順にその頂点番号を出力することを考える

・順番とは？

グラフ上のDFS

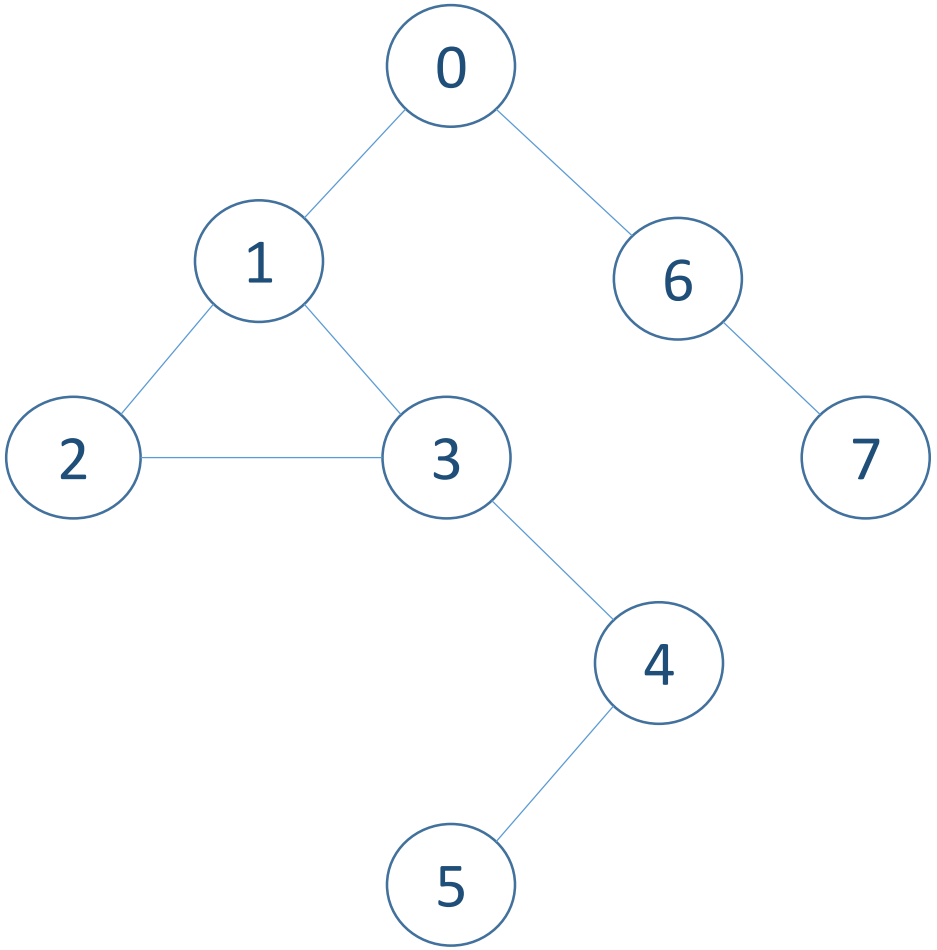
頂点0からDFSを始めることを考える



- 行きがけ順(preorder)
  - 最初の訪問時に出力
- 帰りがけ順(postorder)
  - 最後の訪問時に出力

グラフ上のDFS

頂点0からDFSを始めることを考える

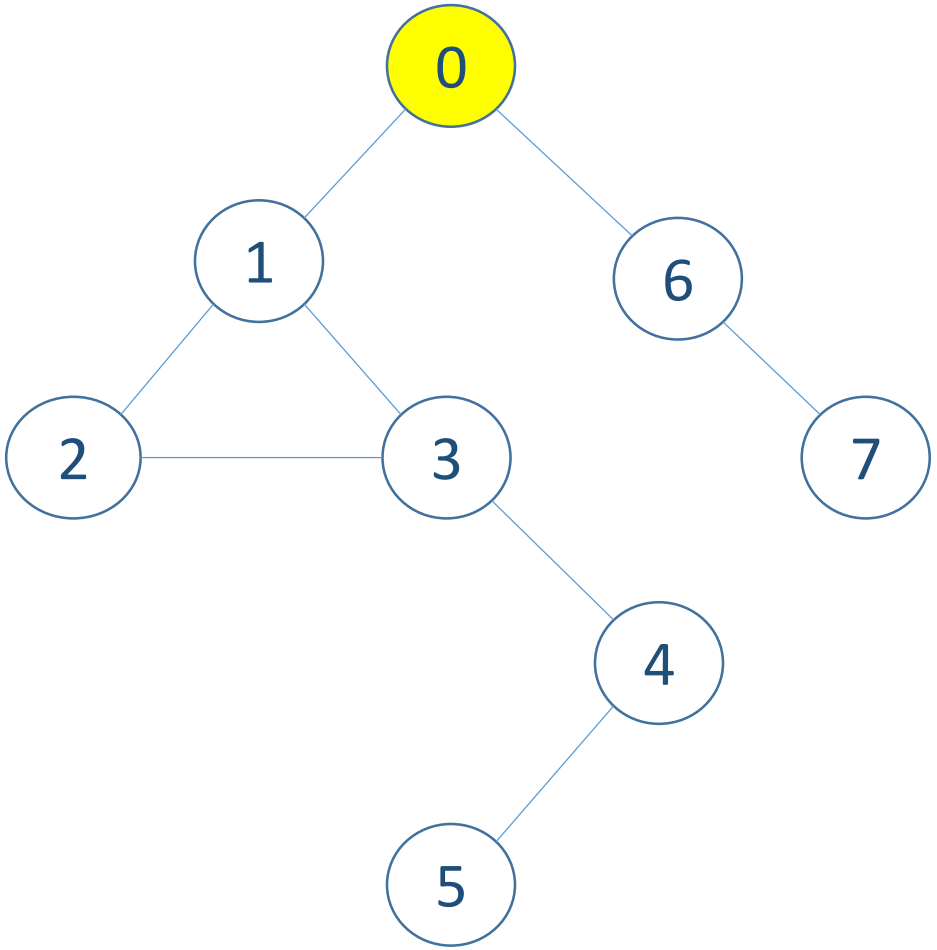


- 行きがけ順(preorder)
  - 最初の訪問時に出力
- 帰りがけ順(postorder)
  - 最後の訪問時に出力

それでは
ノーカットで
ご覧ください

preorder

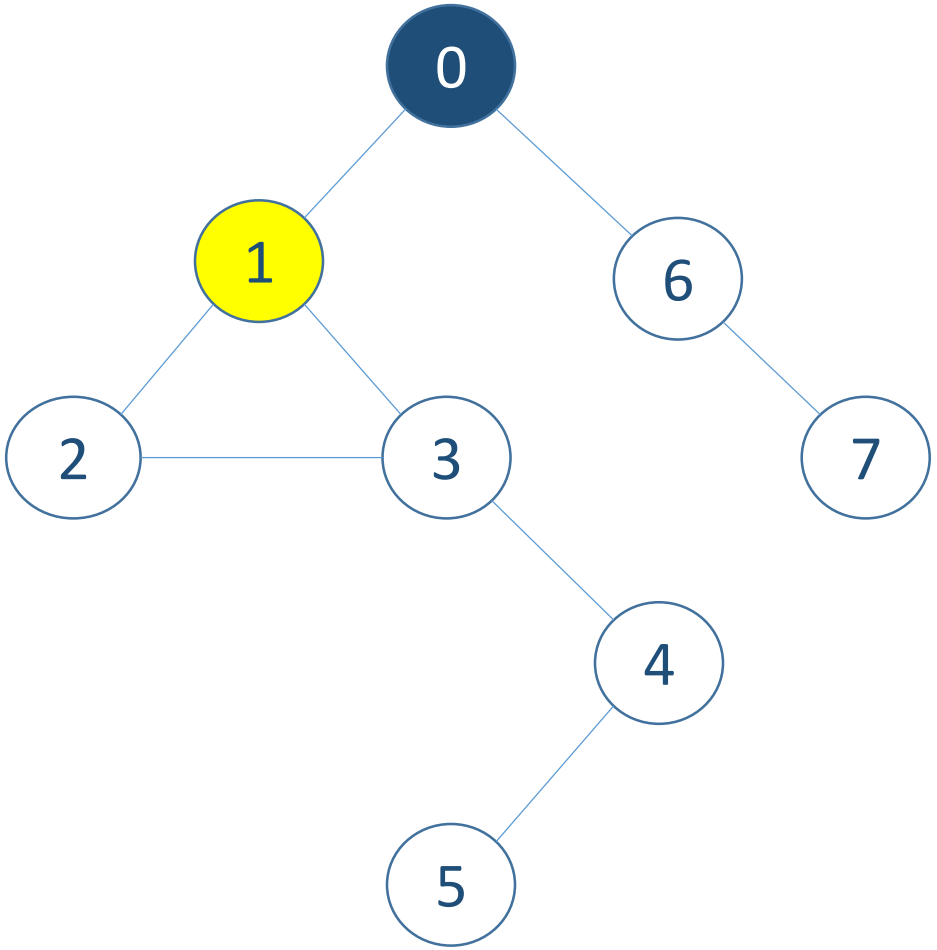
初めて頂点0を訪れたので0を出力



- 行きがけ順(preorder)
- 0

preorder

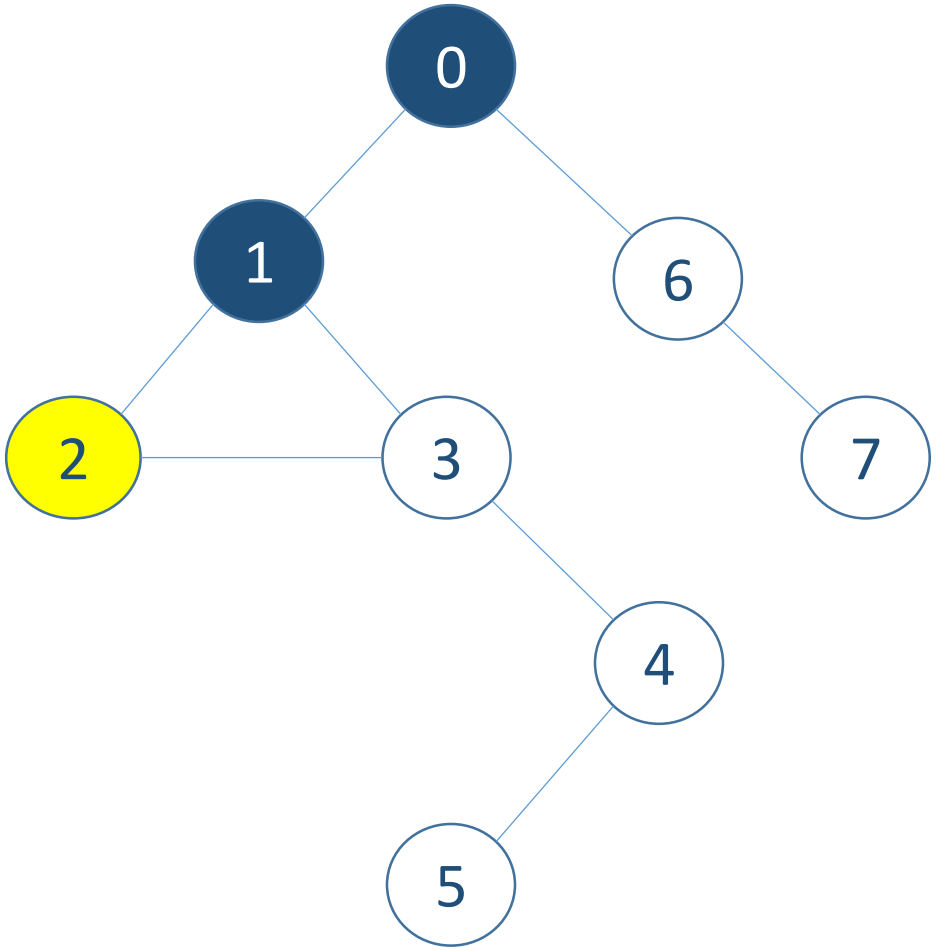
初めて頂点1を訪れたので1を出力



- 行きがけ順(preorder)
- 0, 1

preorder

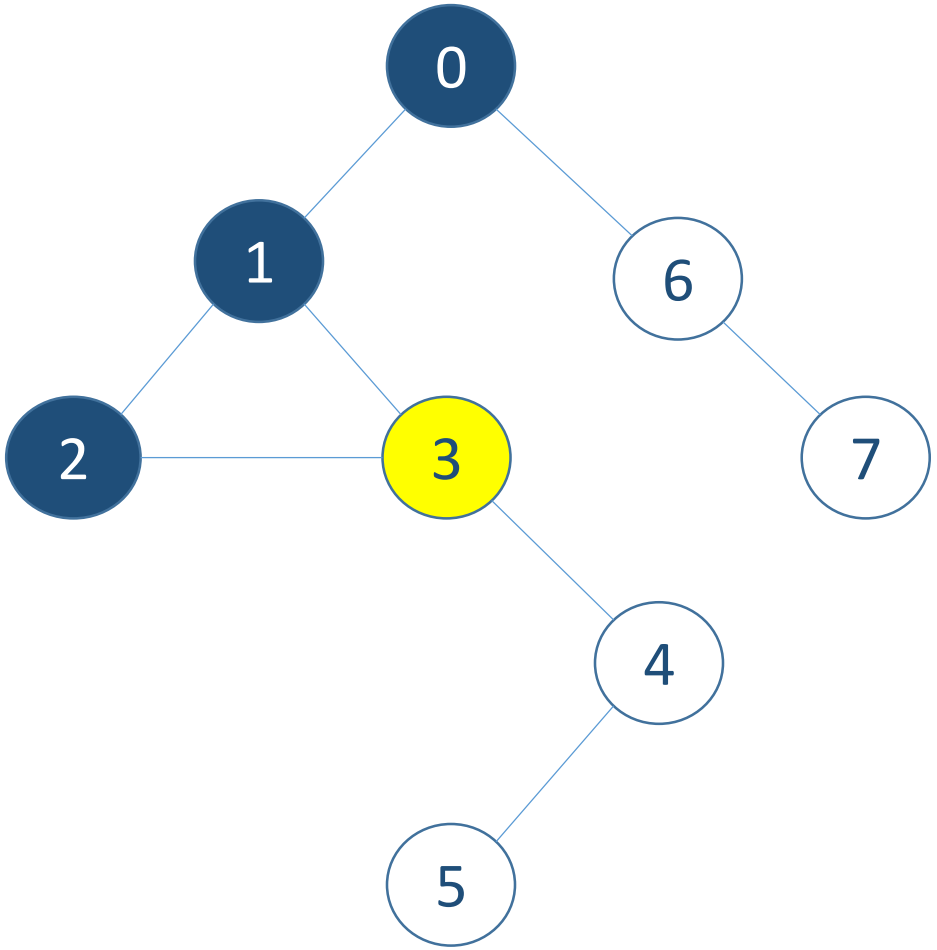
初めて頂点2を訪れたので2を出力



- 行きがけ順(preorder)
- 0, 1, 2

preorder

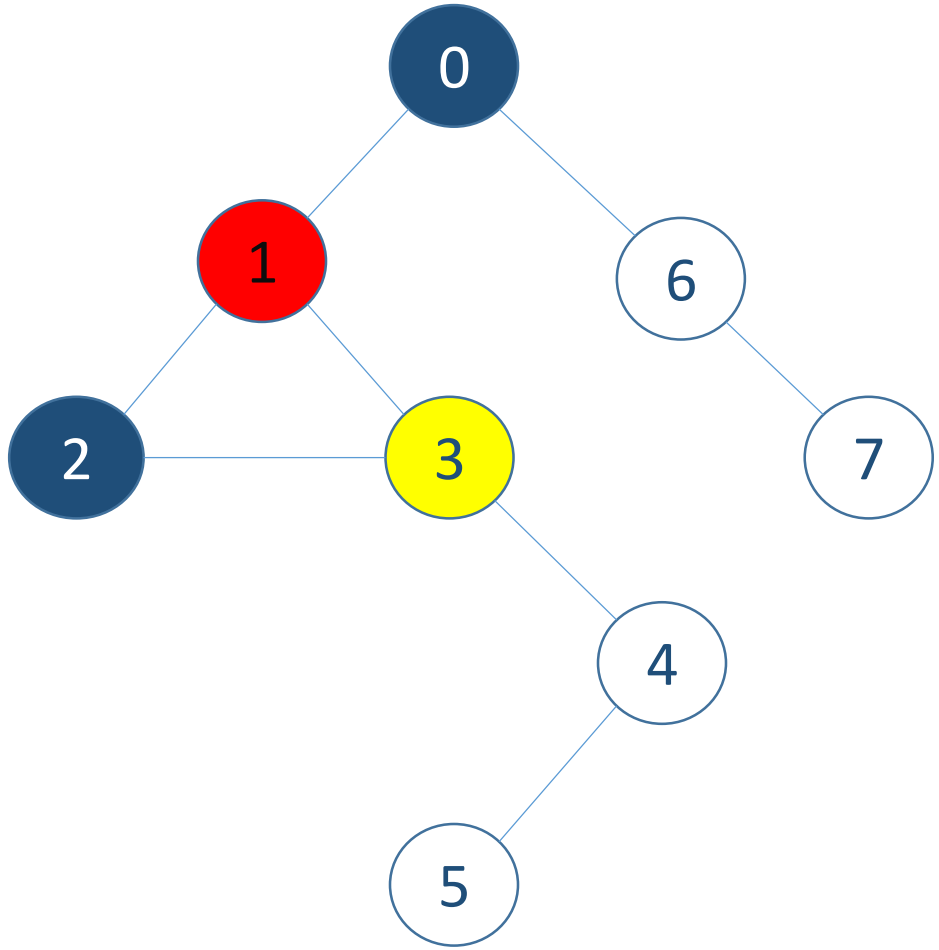
初めて頂点3を訪れたので3を出力



- 行きがけ順(preorder)
- 0, 1, 2, 3

preorder

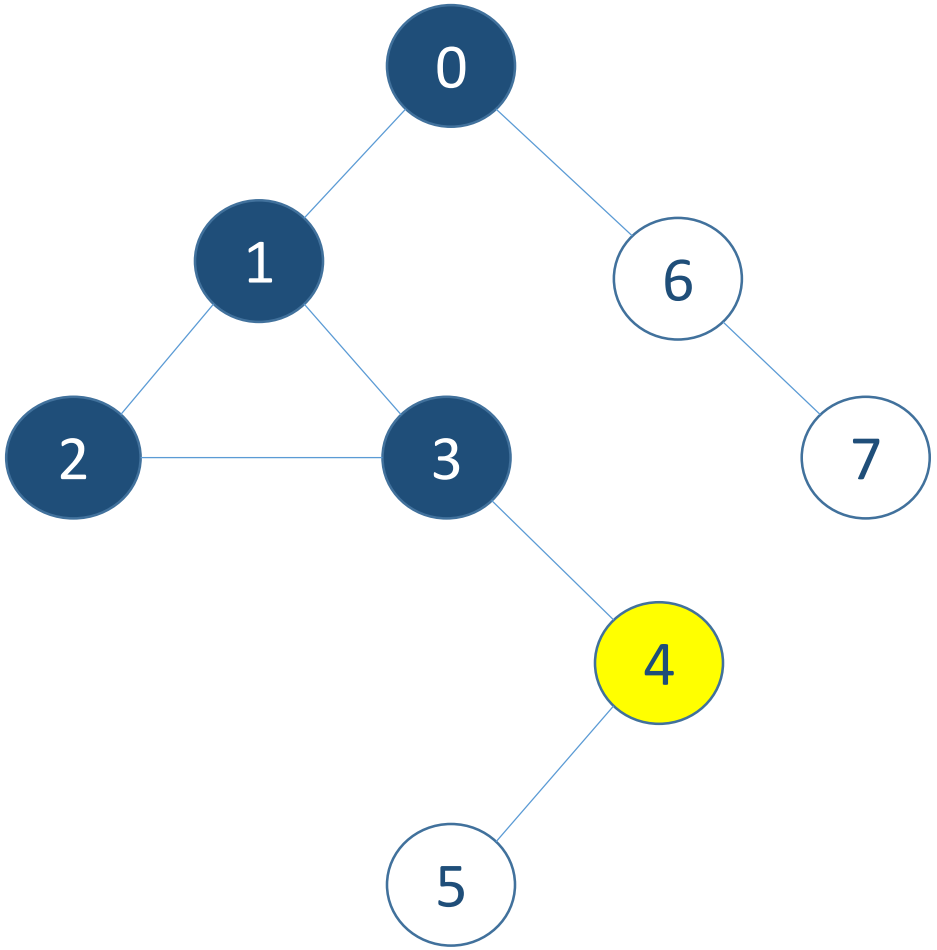
頂点1は訪問済みなので訪れない



- 行きがけ順(preorder)
- 0, 1, 2, 3

preorder

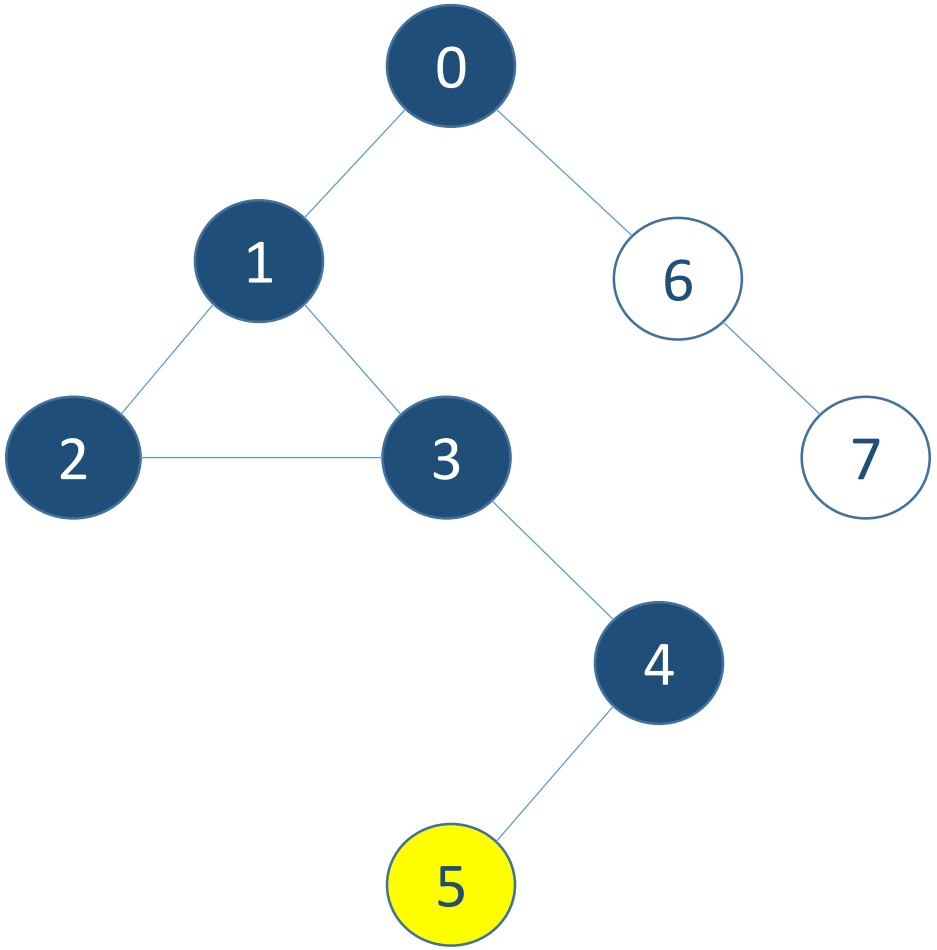
初めて頂点4を訪れたので4を出力



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4

preorder

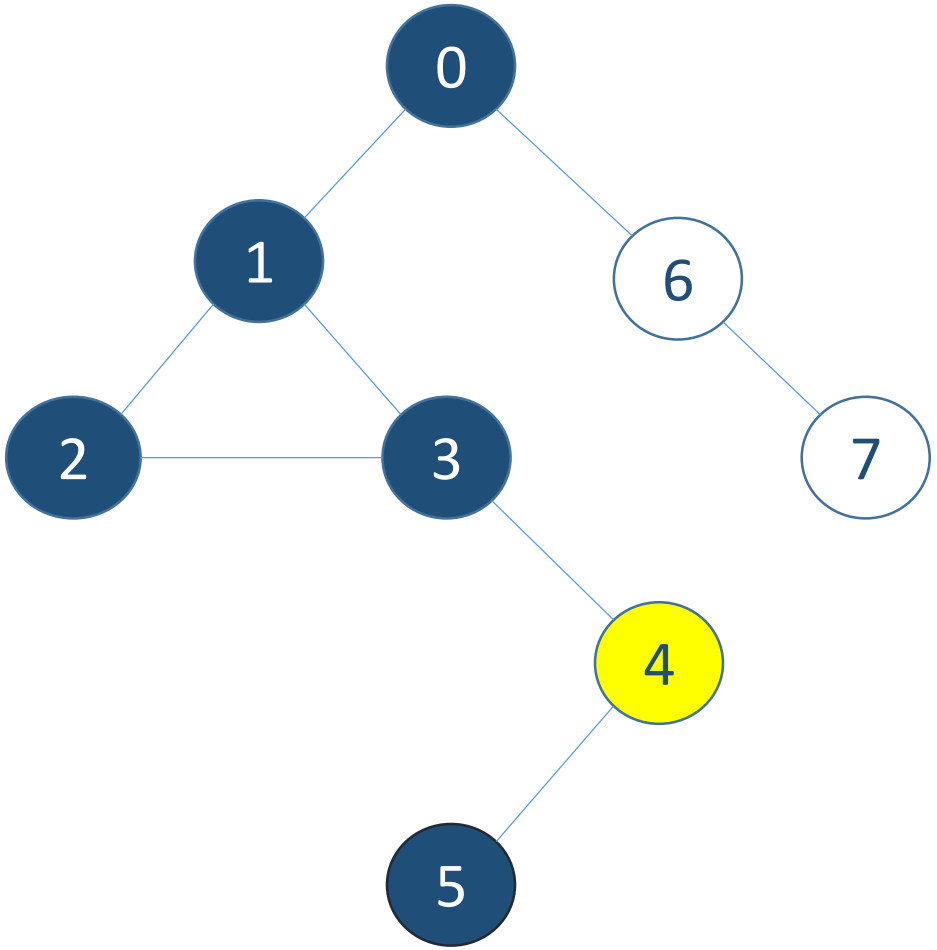
初めて頂点5を訪れたので5を出力



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5

preorder

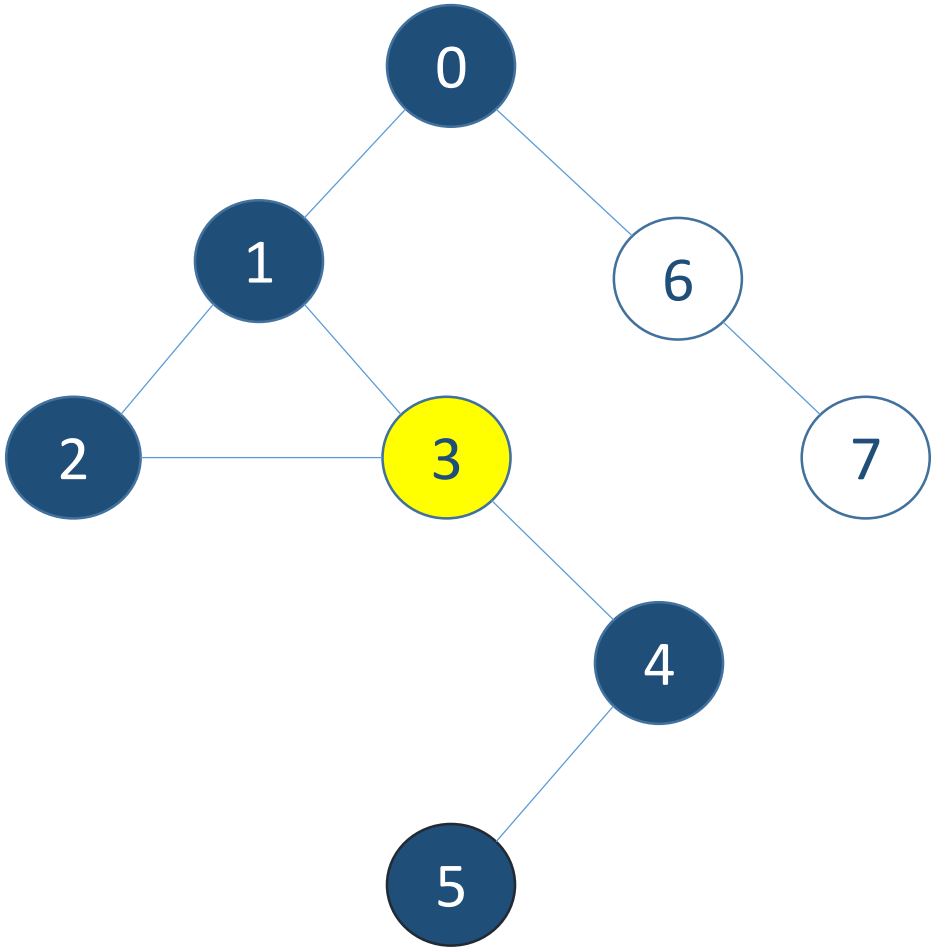
頂点4に戻る



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5

preorder

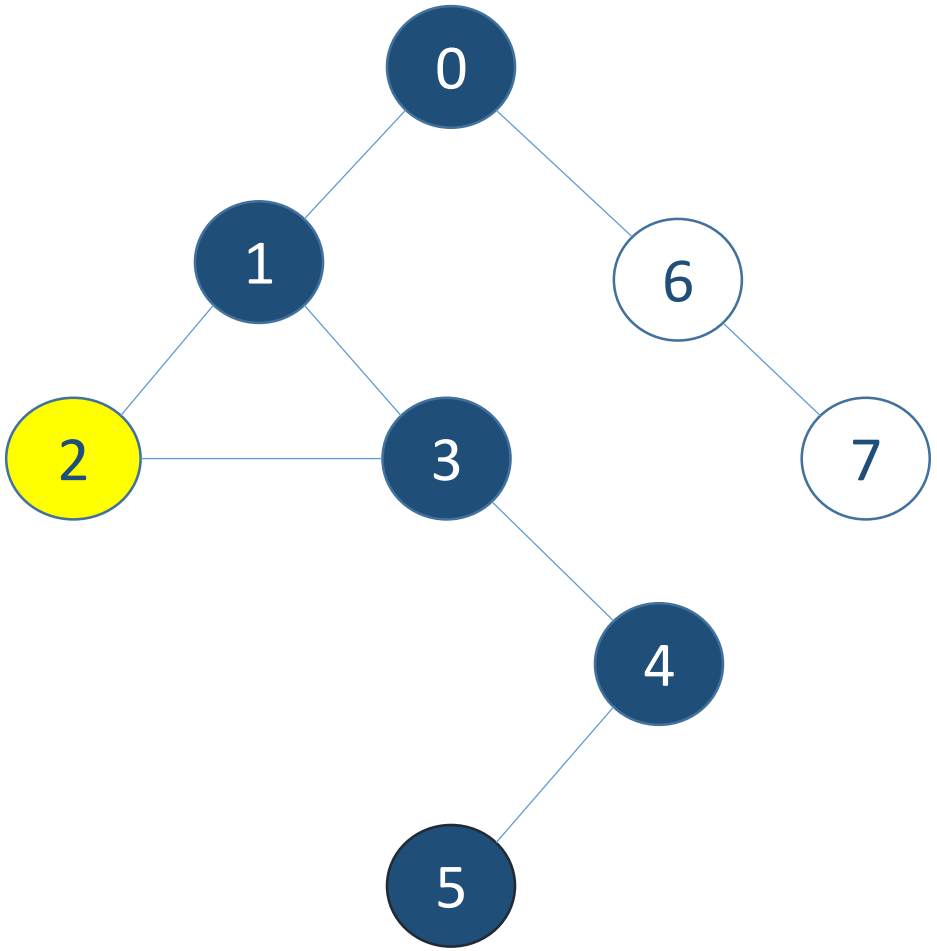
頂点3に戻る



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5

preorder

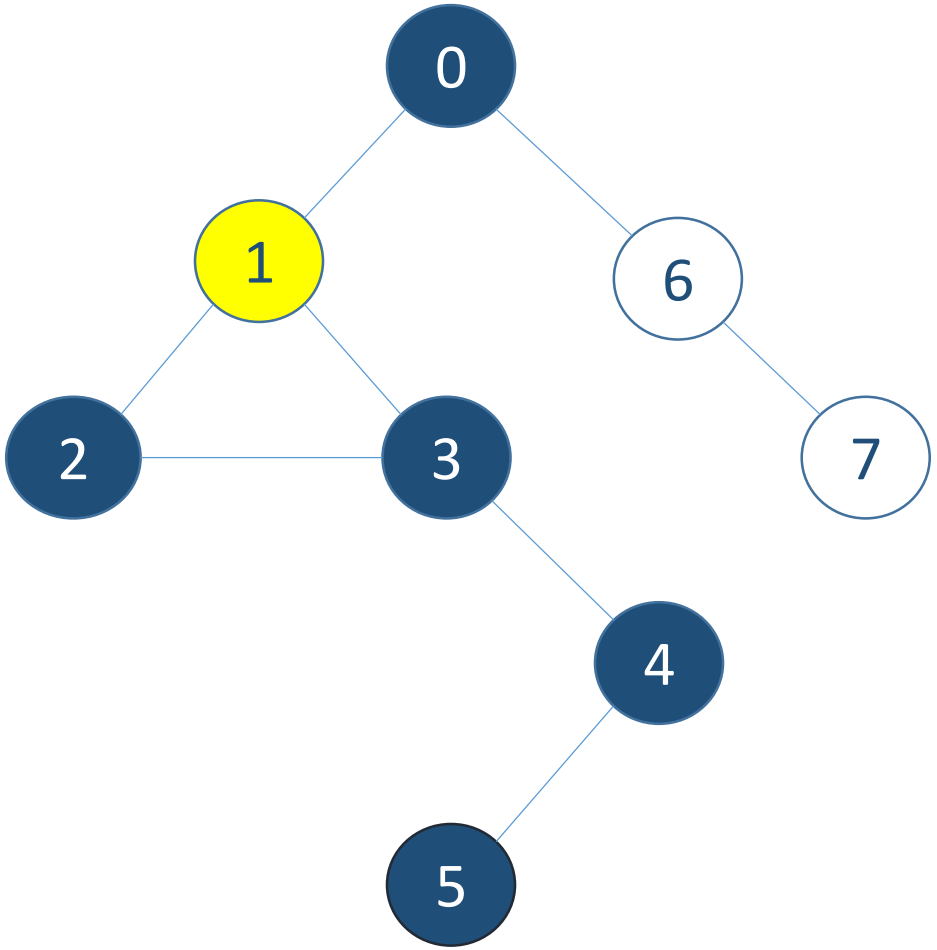
頂点2に戻る



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5

preorder

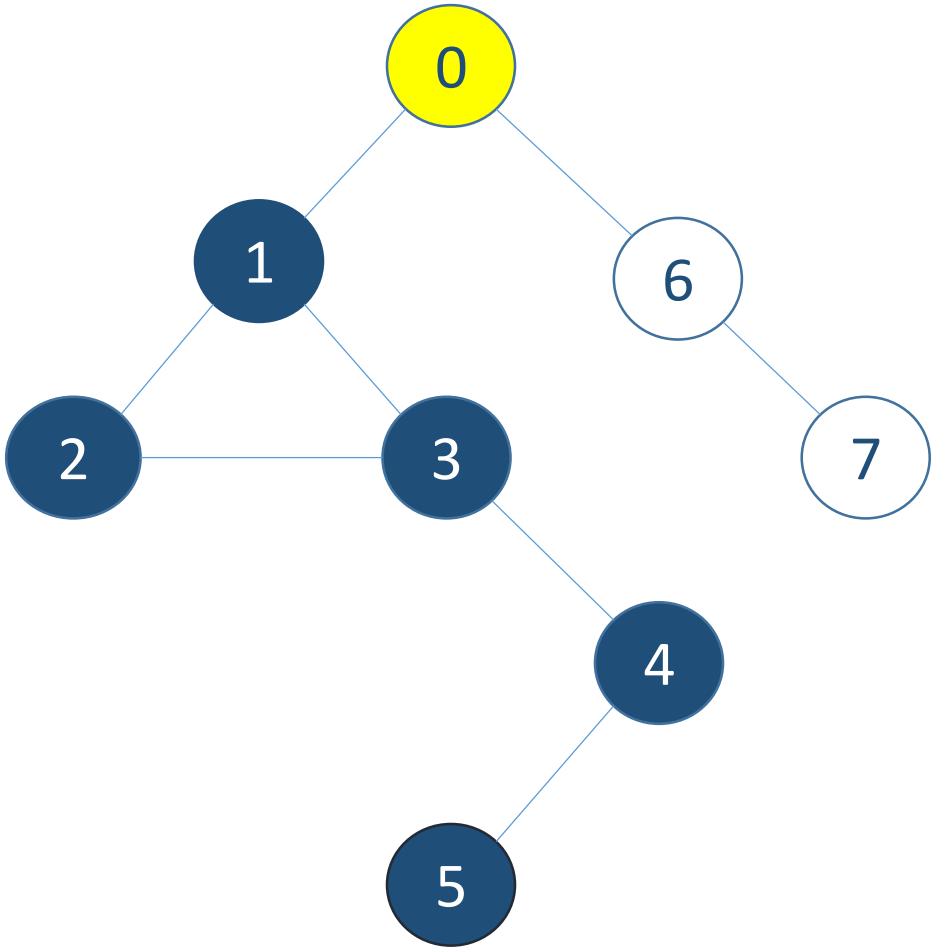
頂点1に戻る



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5

preorder

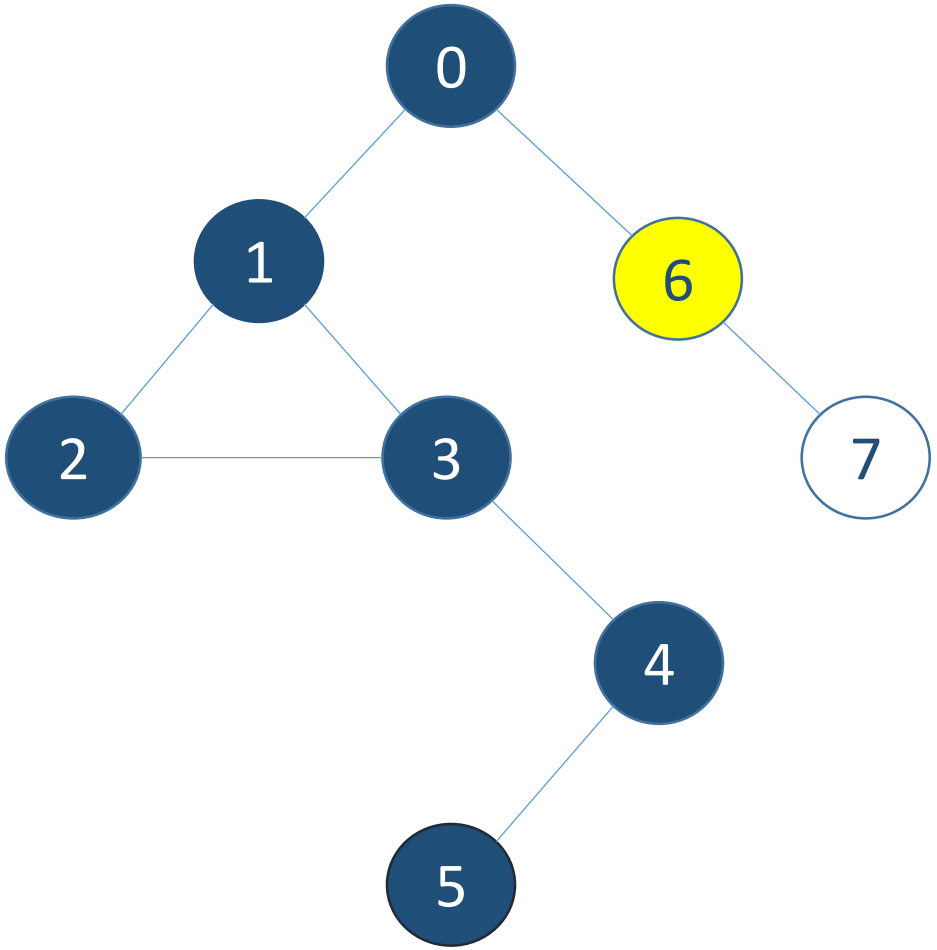
頂点0に戻る



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5

preorder

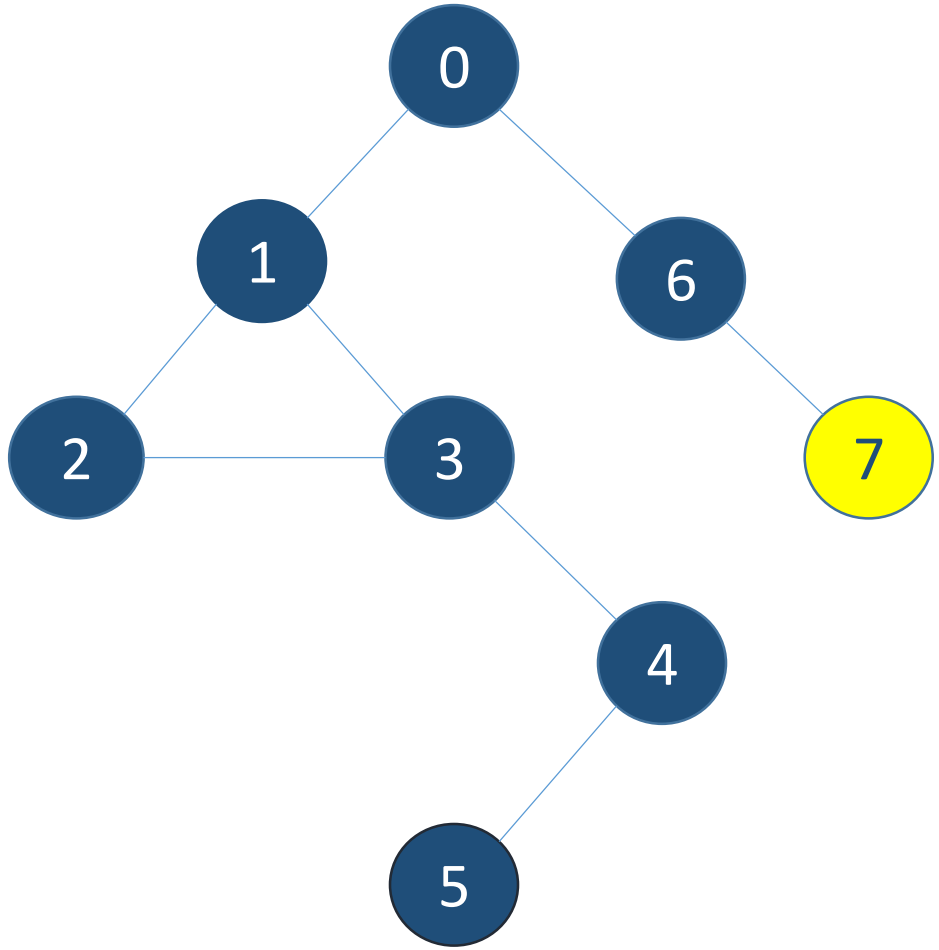
初めて頂点6を訪れたので6を出力



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5, 6

preorder

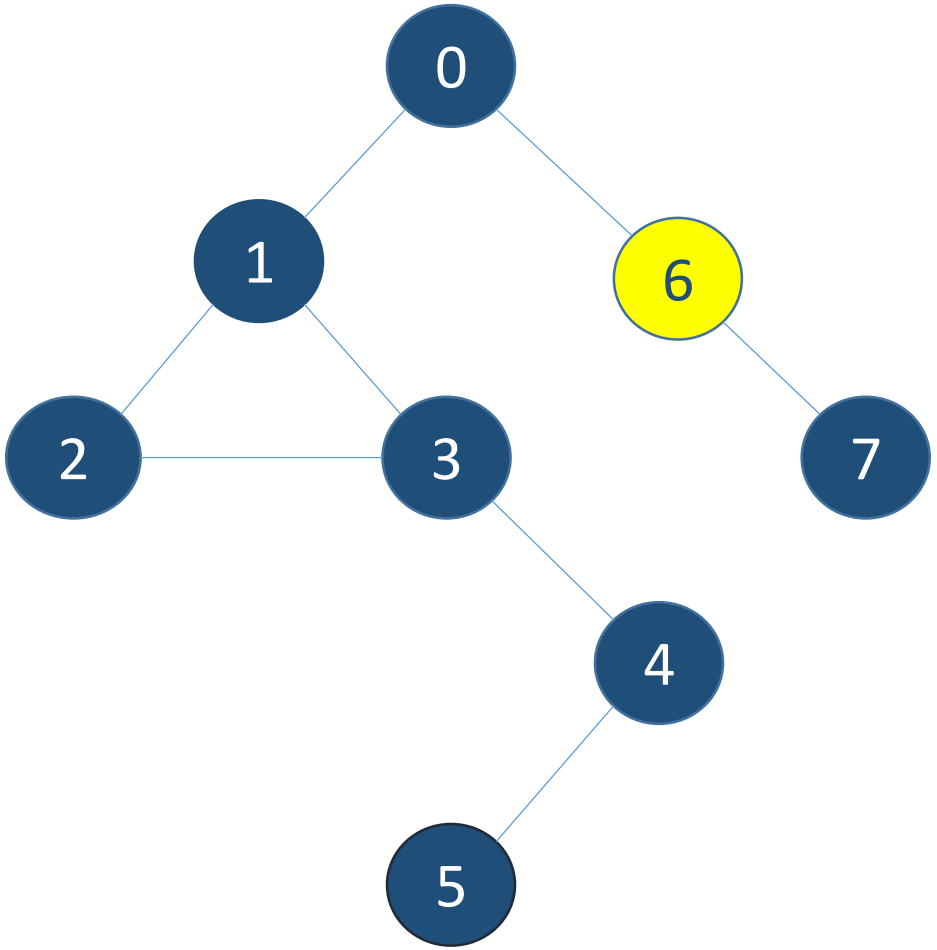
初めて頂点7を訪れたので7を出力



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5, 6, 7

preorder

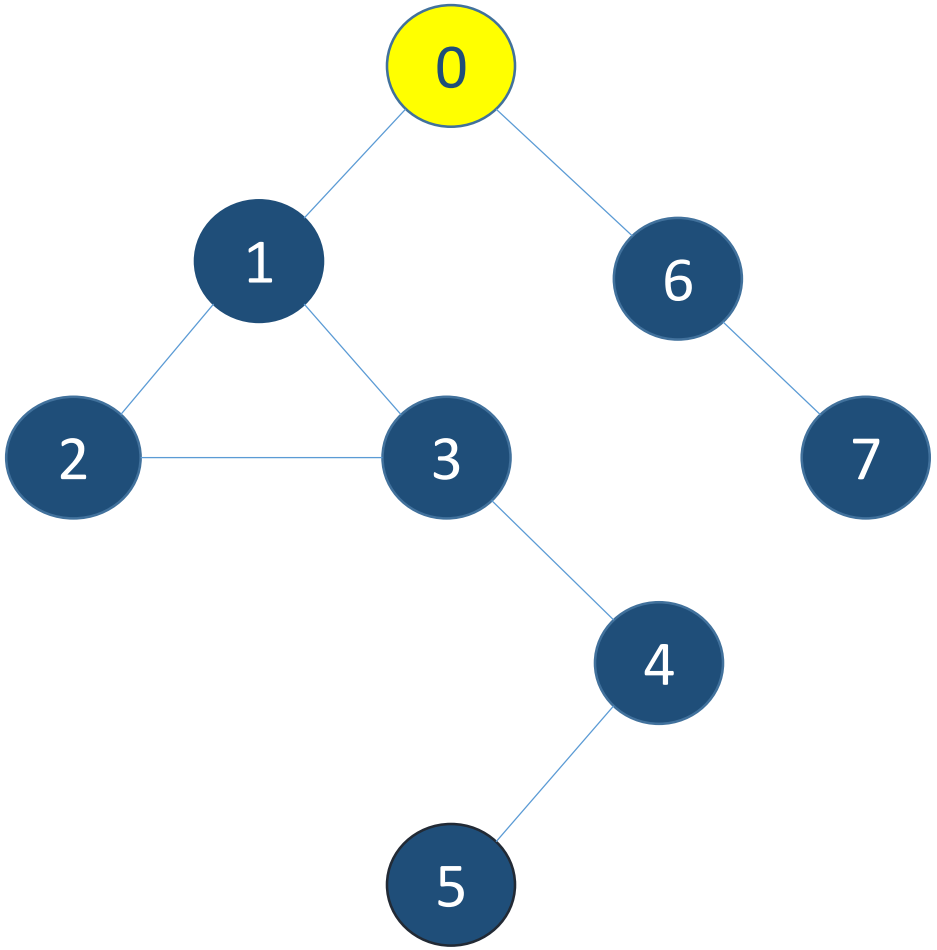
頂点6に戻る



- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5, 6, 7

preorder

頂点0に戻る



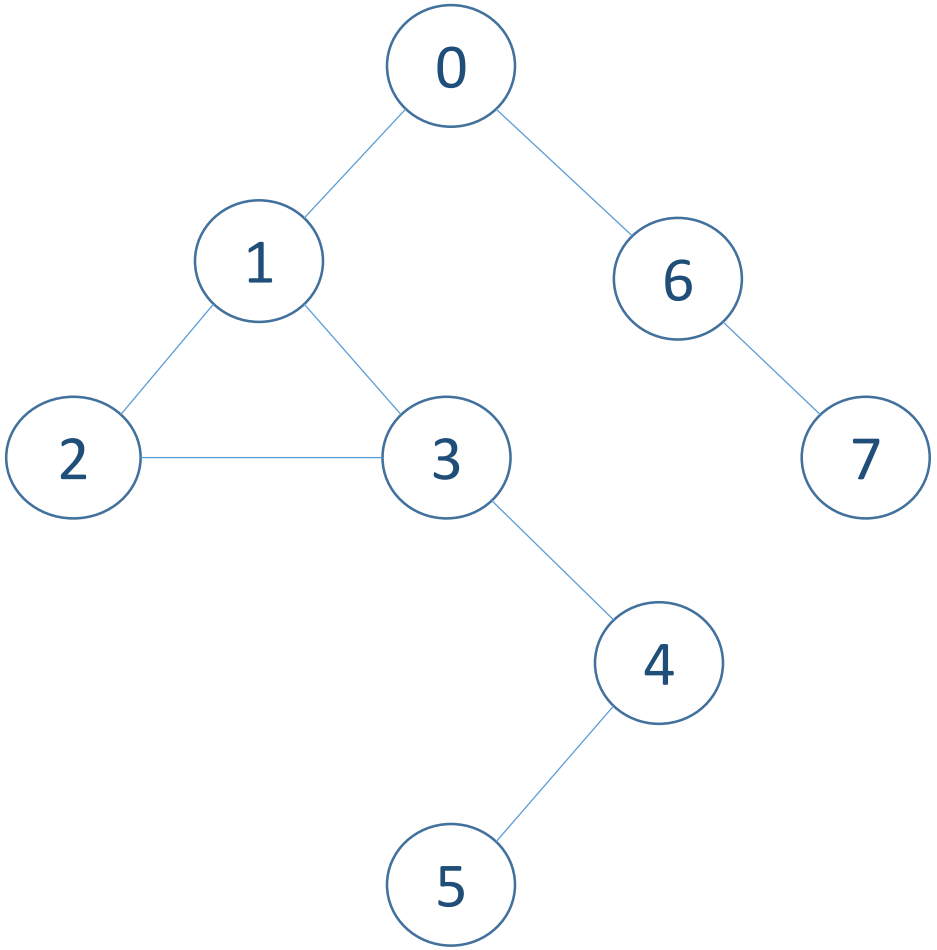
- 行きがけ順(preorder)
- 0, 1, 2, 3, 4, 5, 6, 7

preorder

```cpp
#include<iostream>
#include<vector>
using namespace std;

//now := 現在の頂点, G := 隣接リスト, visited 訪問済みかどうかを管理
void dfs(int now, vector<vector<int> > &G, vector<bool> &visited){

    int V = (int)G.size();

    //pre_orderなのでここで出力
    cout << now << endl;

    //出ている辺をすべてなめる
    for(int i = 0; i < (int)G[now].size(); i++){

        int next = G[now][i];
        //まだ未訪問なら
        if(!visited[next]){

            //訪問済みにして
            visited[next] = true;

            //再帰呼び出し
            dfs(next, G, visited);
        }
    }

    return;
}
```

postorder

頂点0からDFSを始めることを考える

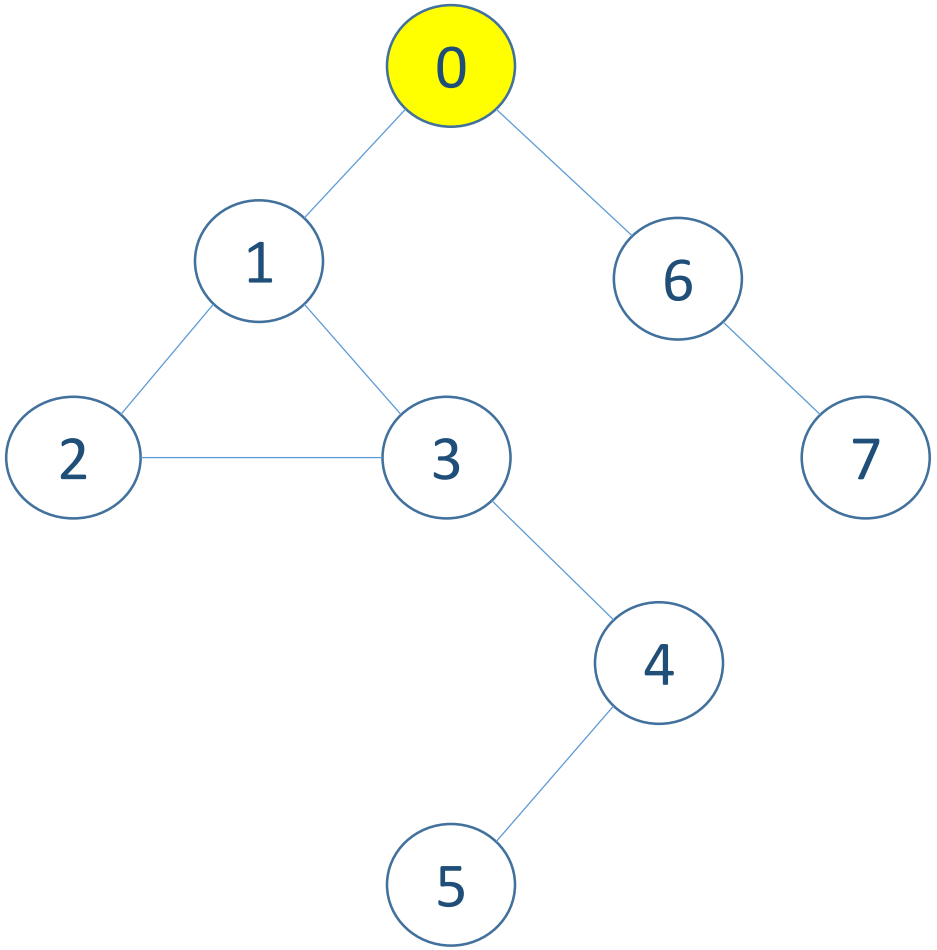


- 行きがけ順(preorder)
  - 最初の訪問時に出力
- 帰りがけ順(postorder)
  - 最後の訪問時に出力

それでは
ノーカットで
ご覧ください

postorder

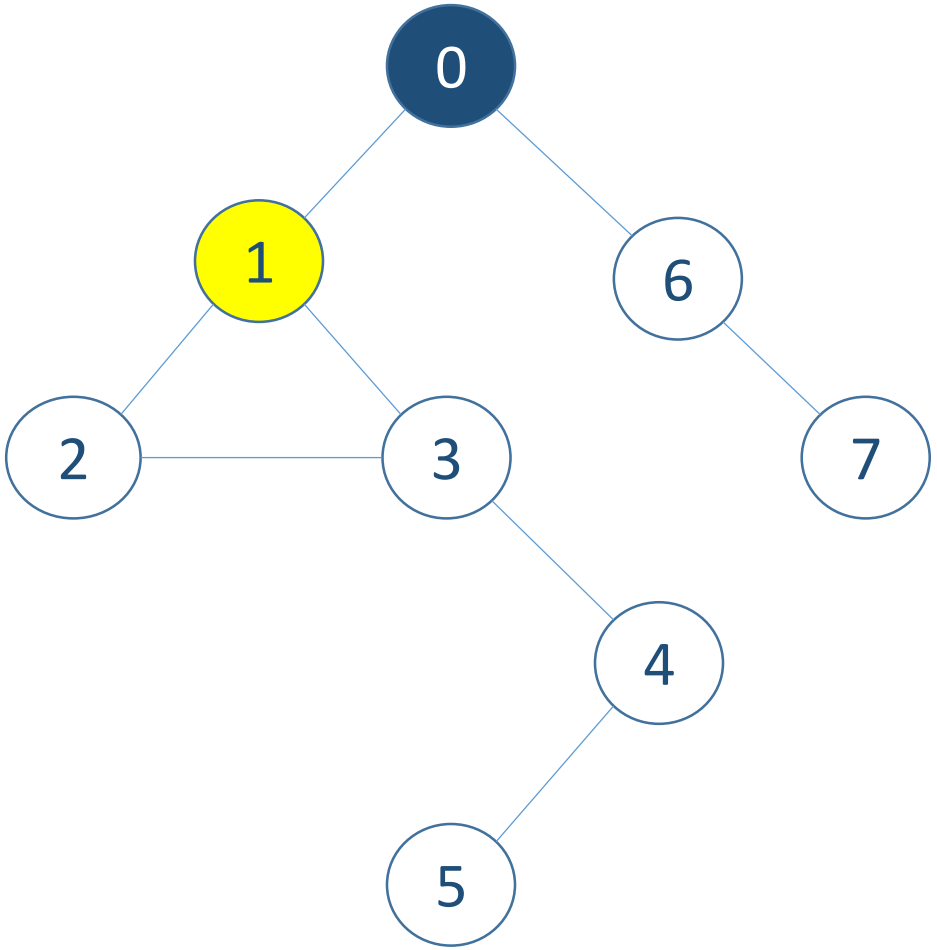
頂点0を訪れる



- 帰りがけ順(preorder)
-

postorder

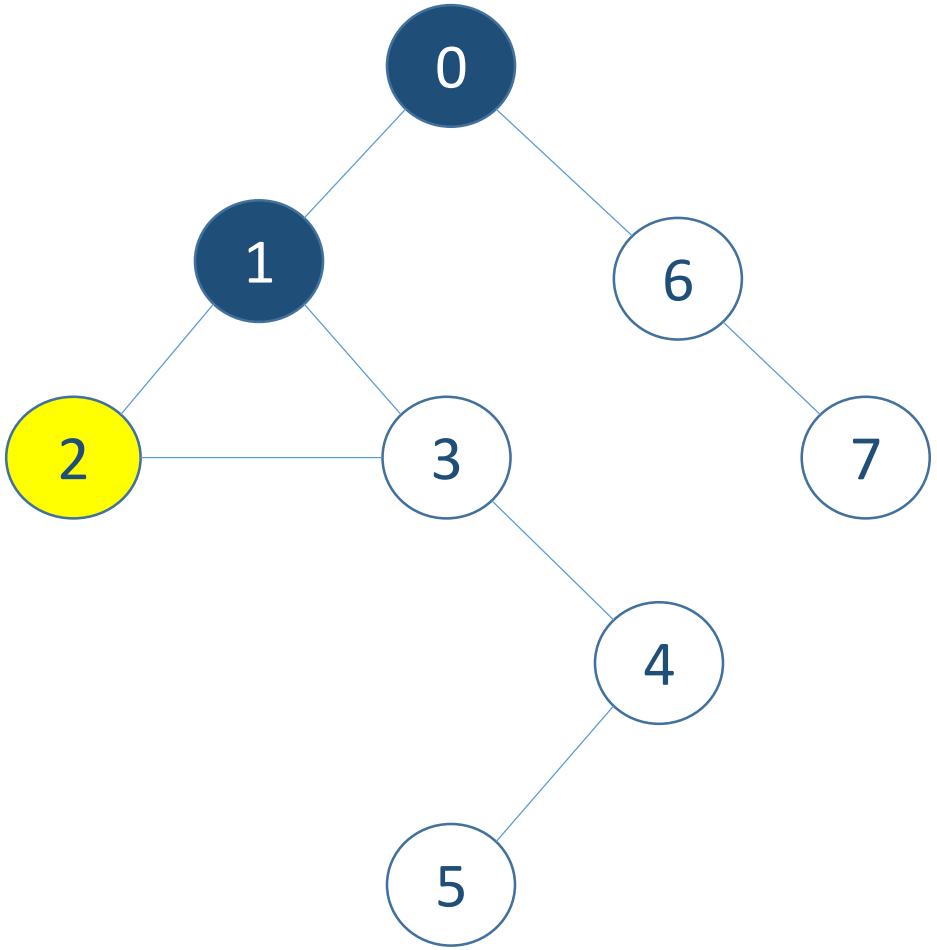
頂点1を訪れる



- 帰りがけ順(preorder)
-

postorder

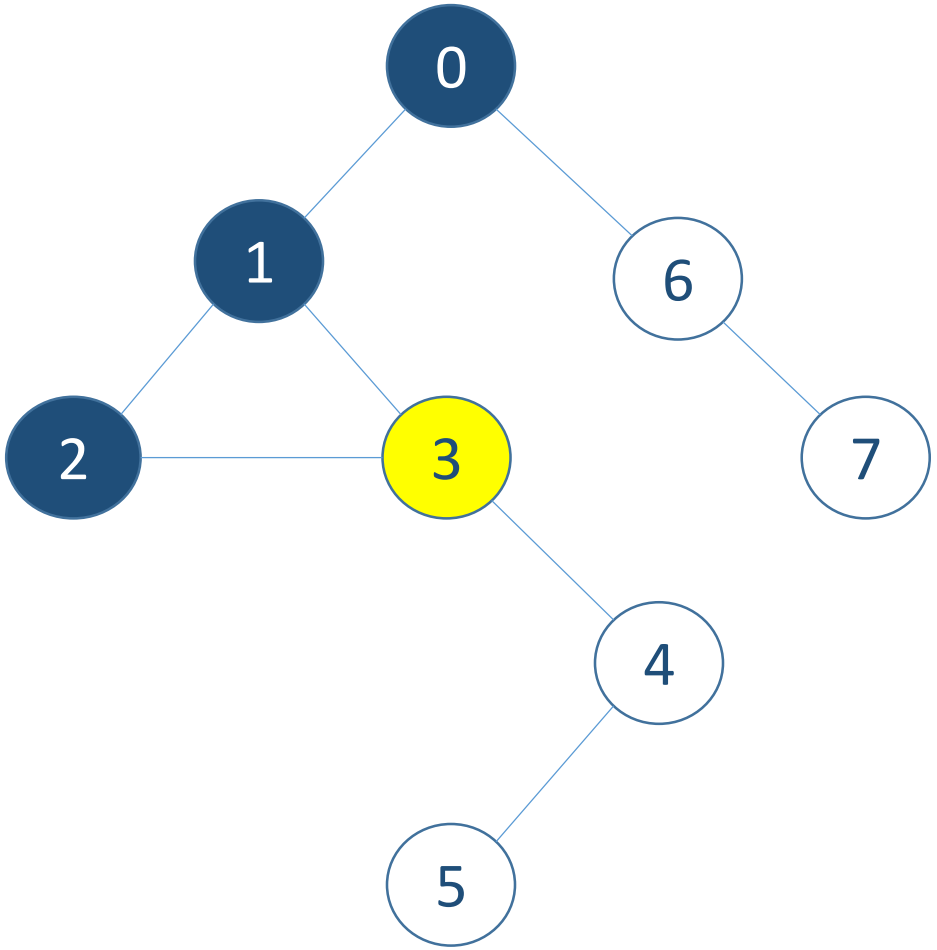
頂点2を訪れる



- 帰りがけ順(preorder)
-

postorder

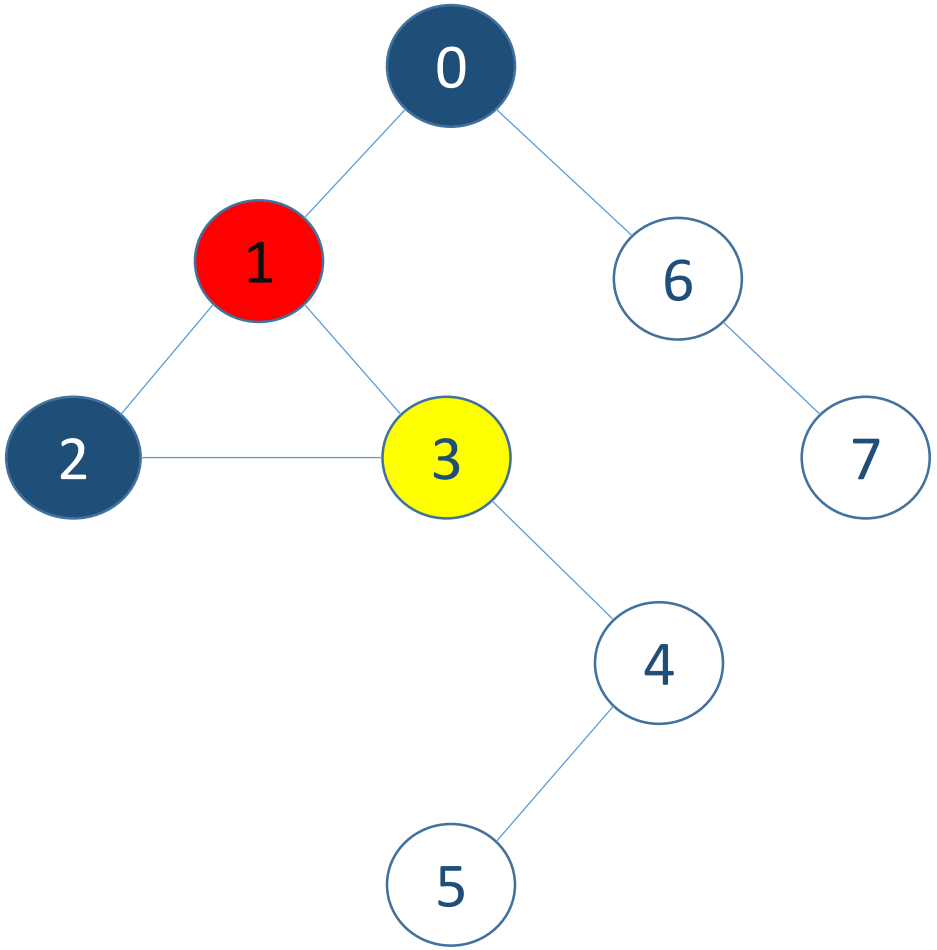
頂点3を訪れる



- 帰りがけ順(preorder)
-

postorder

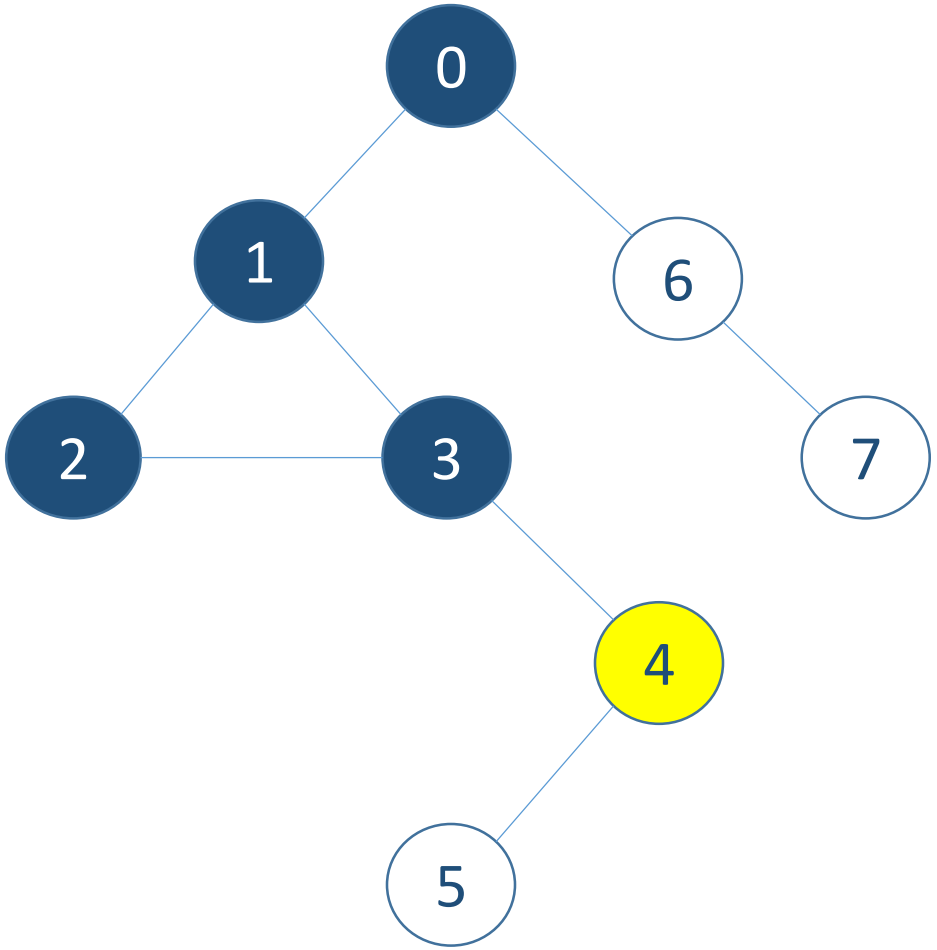
頂点1は訪問済みなので訪れない



- 帰りがけ順(preorder)
-

postorder

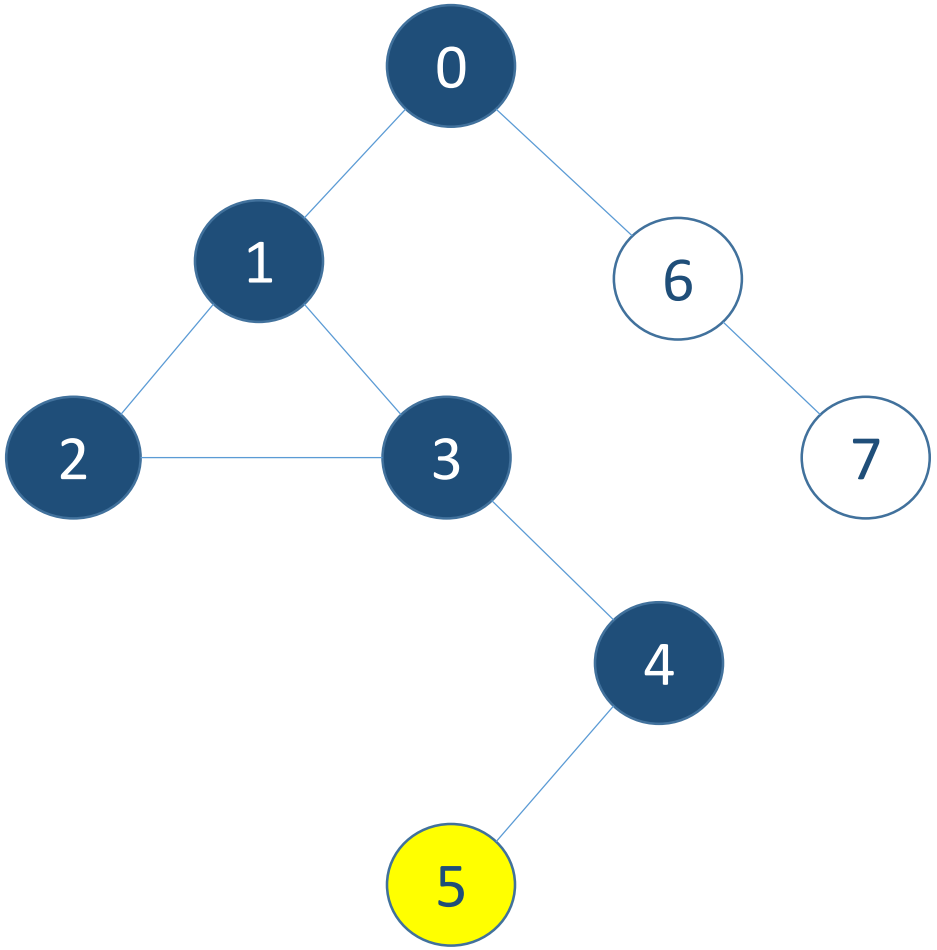
頂点4を訪れる



- 帰りがけ順(preorder)
-

postorder

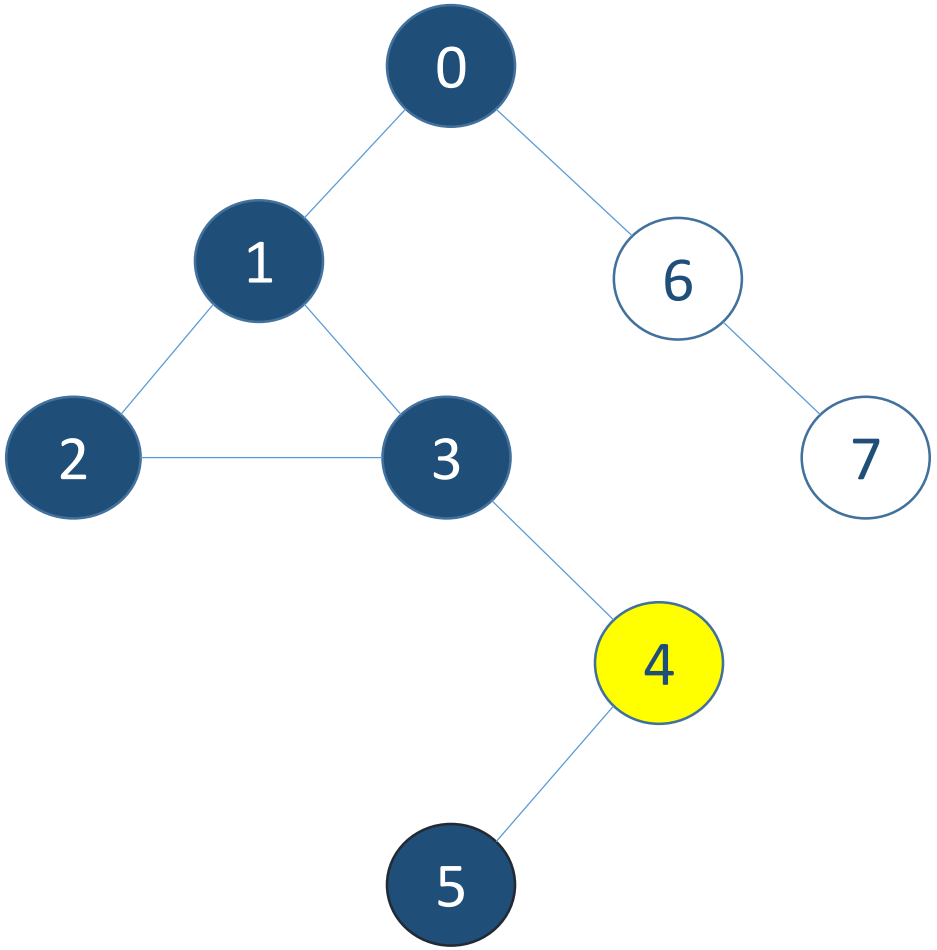
頂点5を訪れる
これが最後の訪問なので5を出力



- 帰りがけ順(preorder)
- 5

postorder

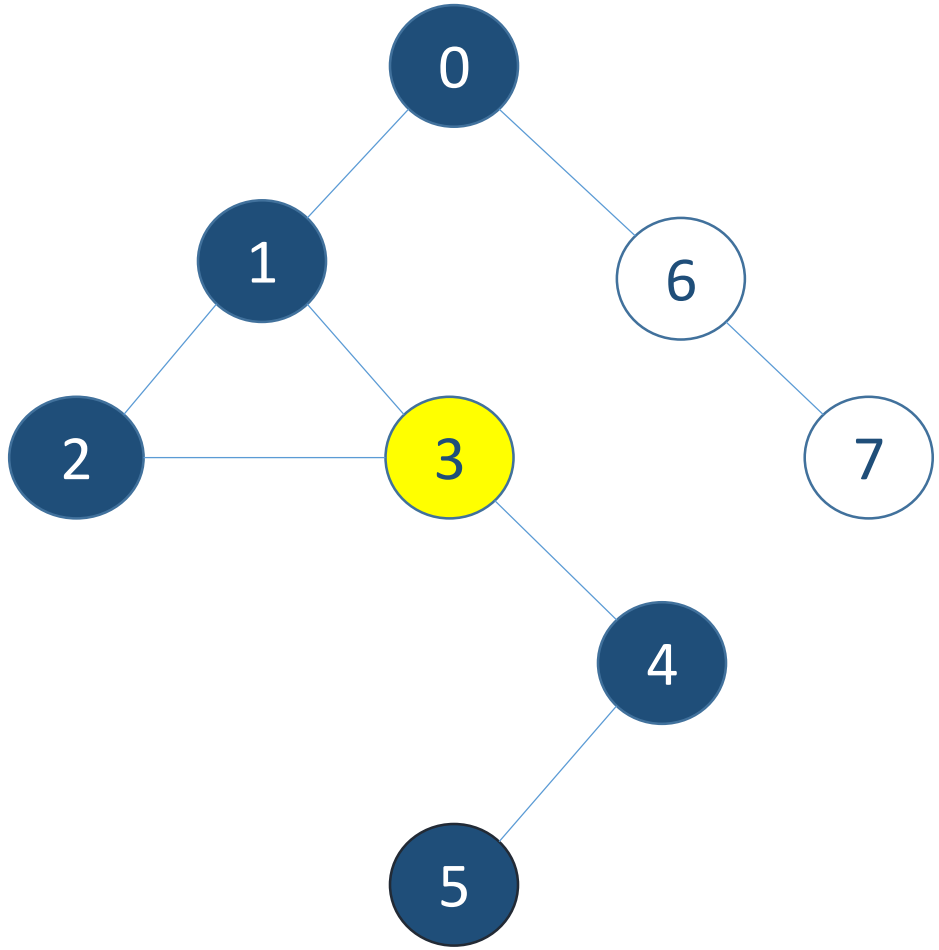
頂点4に戻る
最後なので4を出力



- 帰りがけ順(preorder)
- 5, 4

postorder

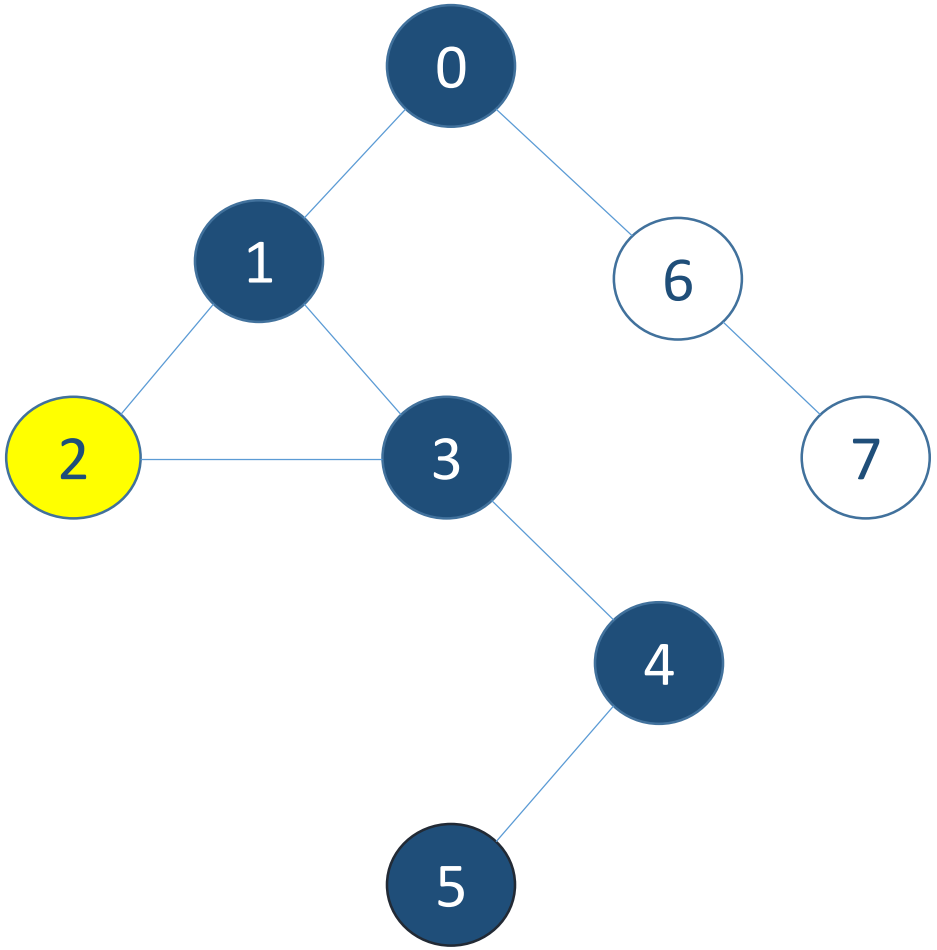
頂点3に戻る
3を出力



- 帰りがけ順(preorder)
- 5, 4, 3

postorder

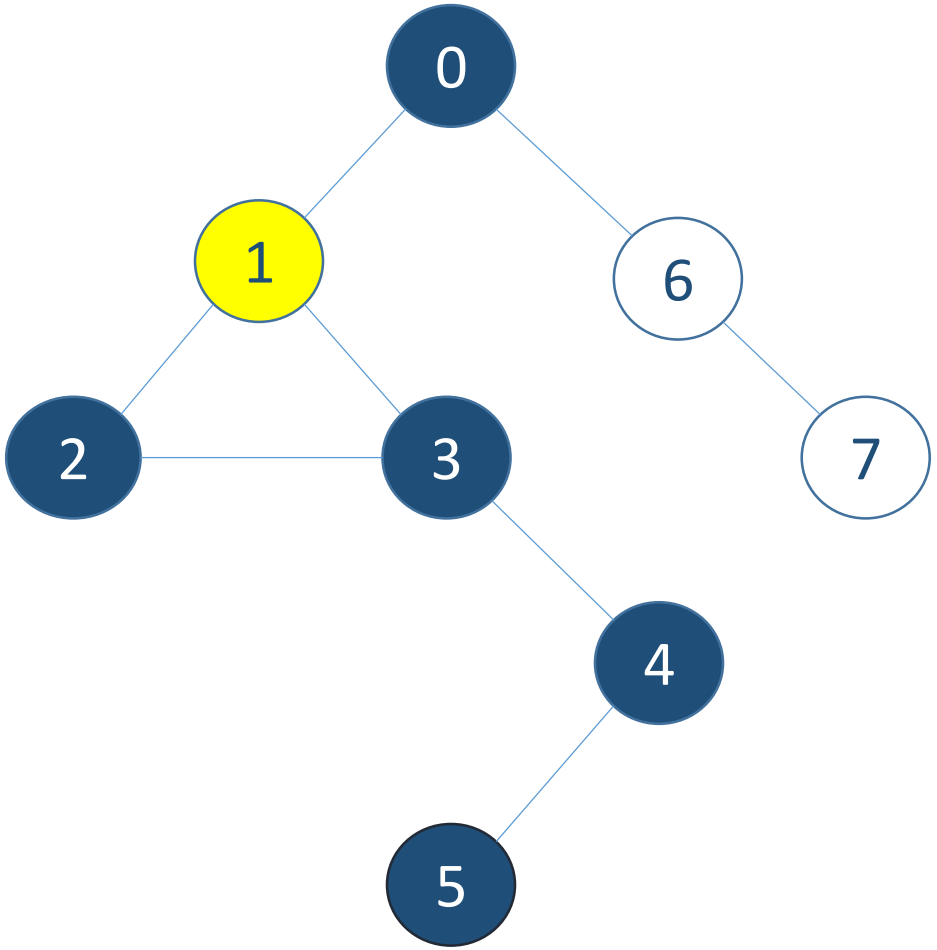
頂点2に戻る
2を出力



- 帰りがけ順(preorder)
- 5, 4, 3, 2

postorder

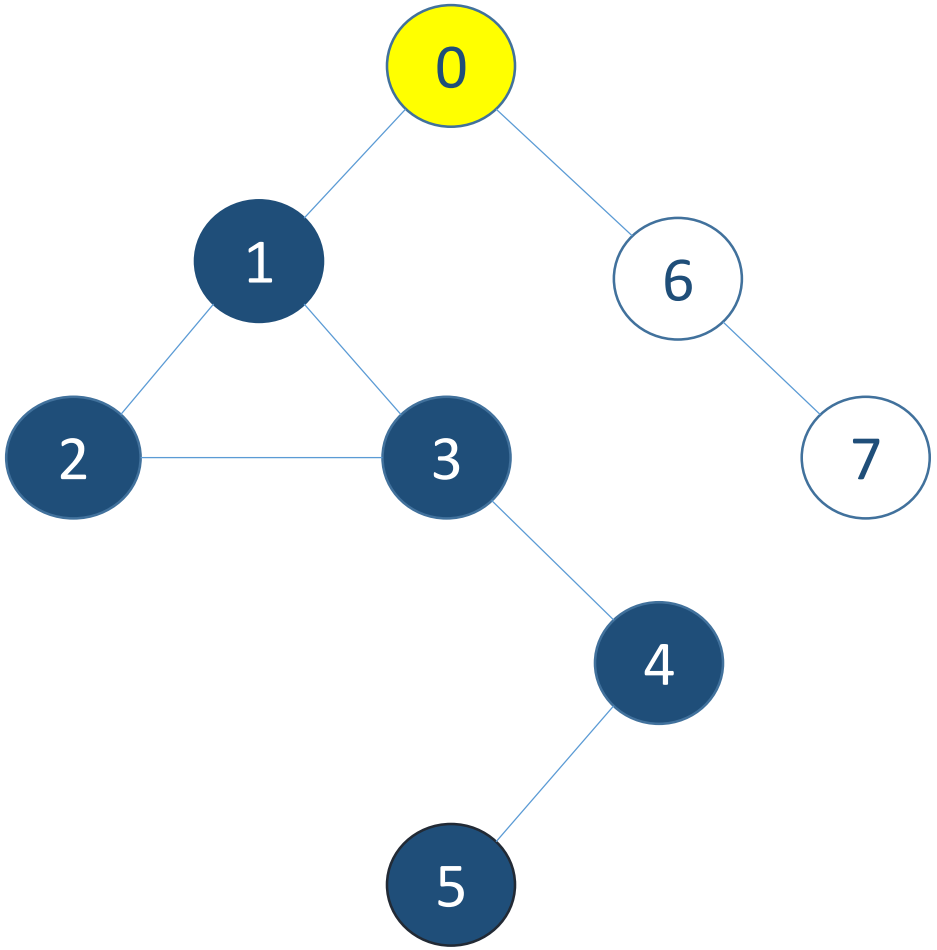
頂点1に戻る



- 帰りがけ順(preorder)
- 5, 4, 3, 2, 1

postorder

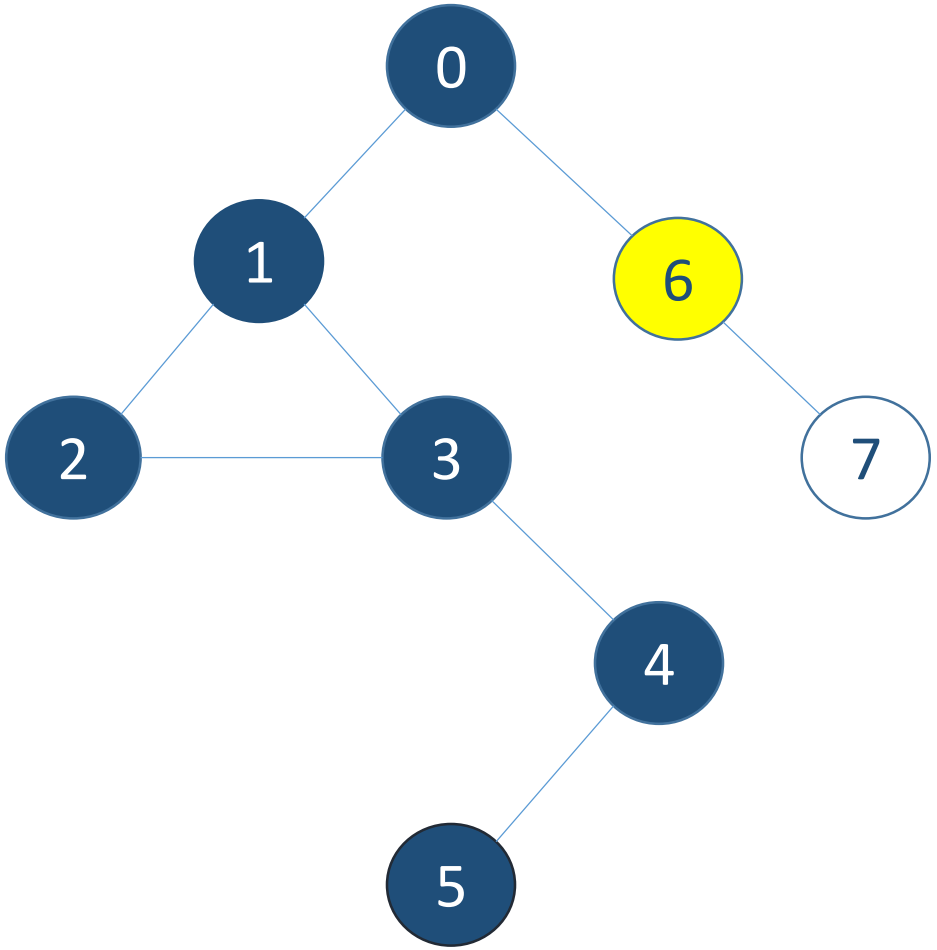
頂点0に戻る



- 帰りがけ順(preorder)
- 5, 4, 3, 2, 1

postorder

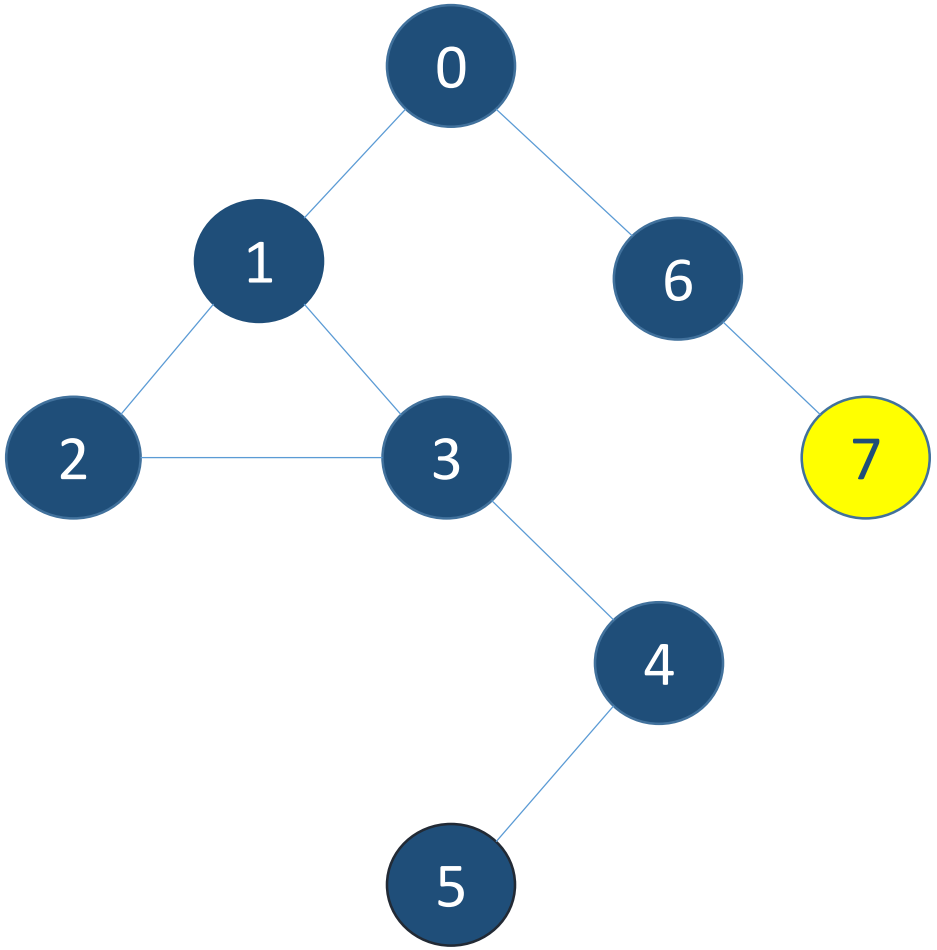
頂点6を訪れる



- 帰りがけ順(preorder)
- 5, 4, 3, 2, 1

postorder

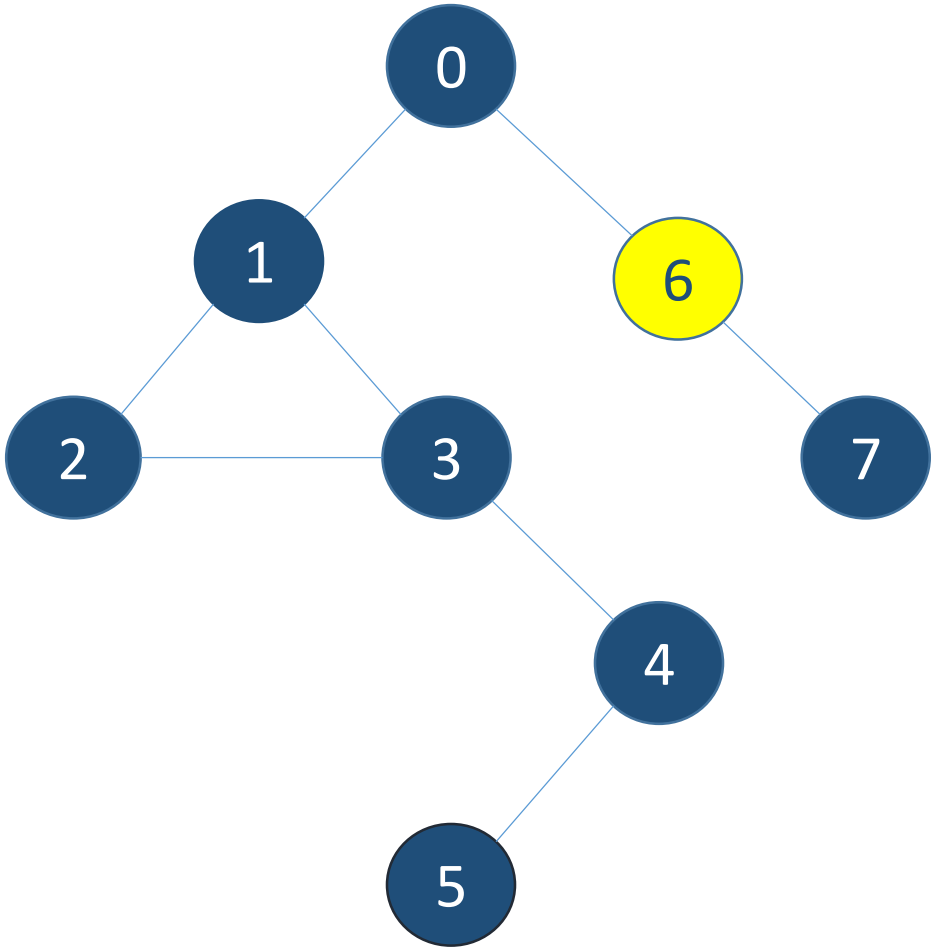
頂点7を訪れる
7を出力



- 帰りがけ順(preorder)
- 5, 4, 3, 2, 1, 7

postorder

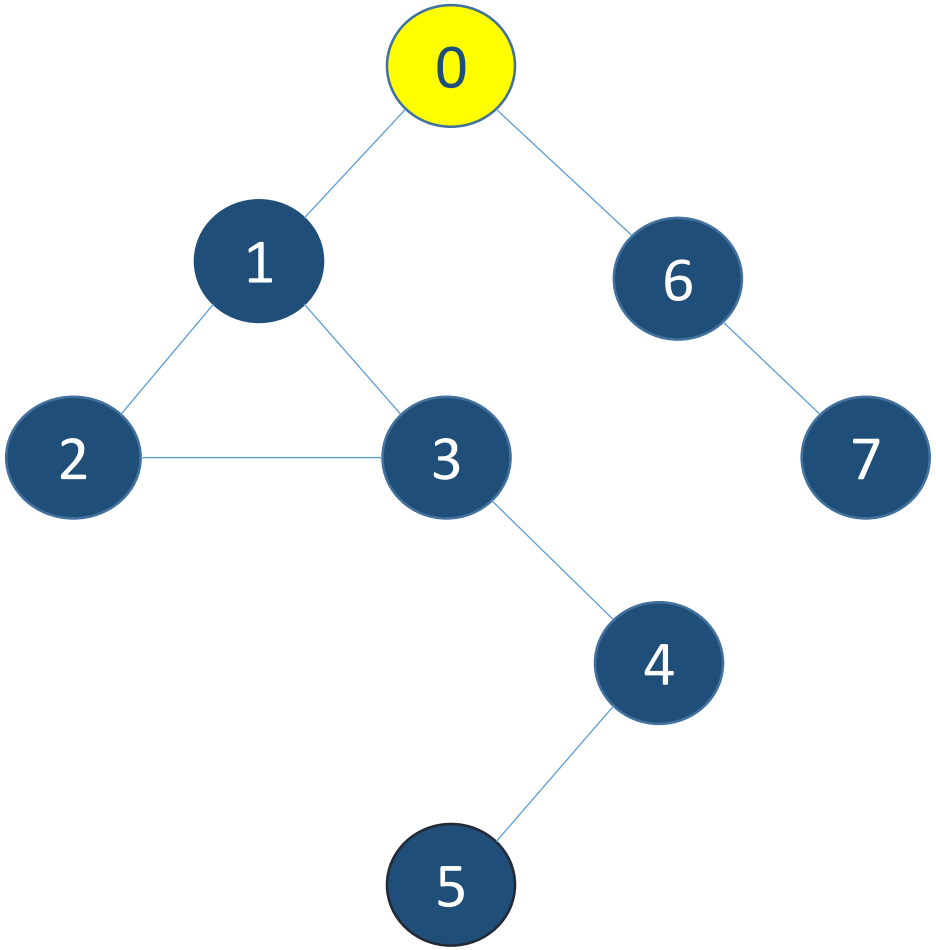
頂点6に戻る
6を出力



- 帰りがけ順(preorder)
- 5, 4, 3, 2, 1, 7, 6

postorder

頂点0に戻る
0を出力



- 帰りがけ順(preorder)
- 5, 4, 3, 2, 1, 7, 6

postorder

```cpp
#include<iostream>
#include<vector>
using namespace std;


void dfs(int now, vector<vector<int> > &G, vector<bool> &visited){

    int V = (int)G.size();

    //出ている辺をすべてなめる
    for(int i = 0; i < (int)G[now].size(); i++){

        int next = G[now][i];
        //まだ未訪問なら
        if(!visited[next]){

            //訪問済みにして
            visited[next] = true;

            //再帰呼び出し
            dfs(next, G, visited);
        }
    }

    //post_orderはここで出力
    cout << now << endl;

    return;
}
```

目次

・グラフの話

-基礎

-dfs

・トポロジカルソート

・強連結成分分解

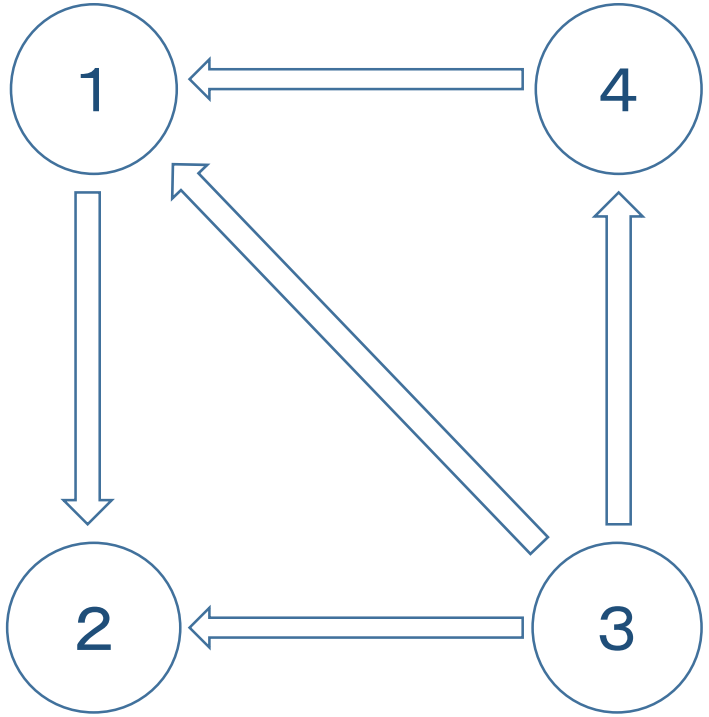
・問題への応用

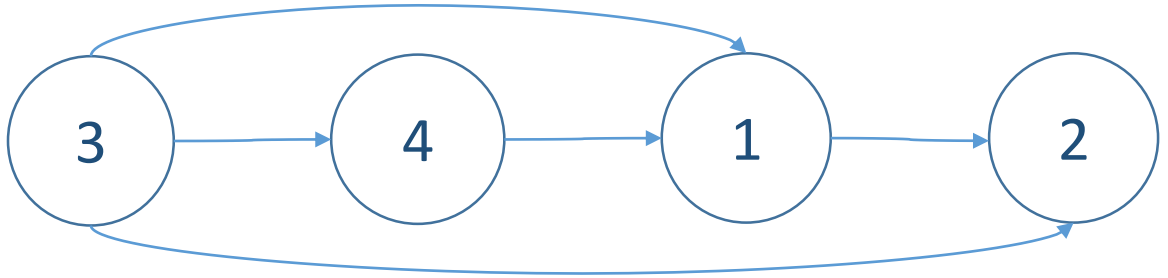
トポロジカルソート

トポロジカルソートってなに？

-各頂点に対して順序付けをし、i番目の頂点を v_i とする

-v_i から v_j に辺があるとき $i < j$ が成り立つような順序付け





トポロジカルソート

何がうれしいか

-DPの更新の順序がわかる！

-そもそも、いつもやってるDPの更新順序はトポロジカルソートなんです。

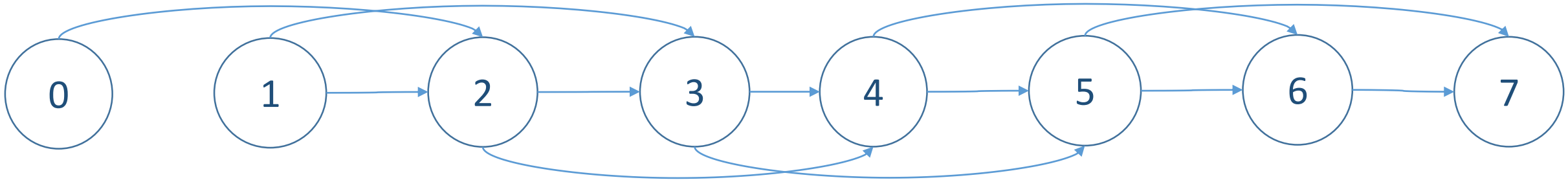
-簡単なDPは更新順序が自明(フィボナッチとかナップサックとか)

-

-ほかにも嬉しいことあるかも(強い人に聞いてください)

フィボナッチ数列

フィボナッチ数列 $(f[n] = f[n-1] + f[n-2])$
更新順序は $v0, v1, v2, \ldots,$

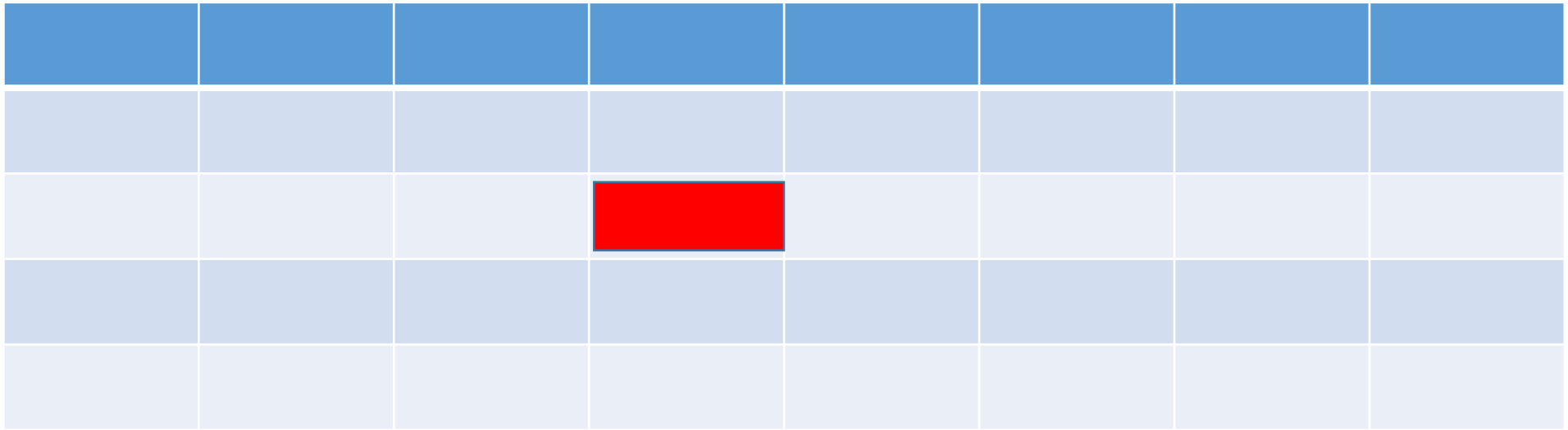


ナップサック問題

ナップサック問題
よくあるdpテーブルの取り方

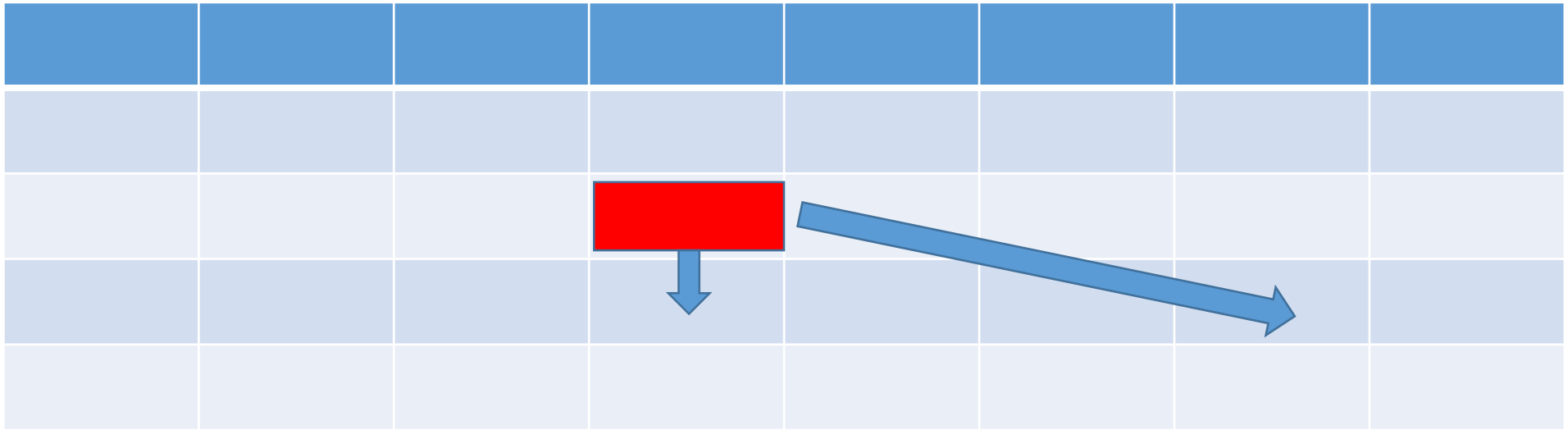
ナップサック問題

ナップサック問題
例えばこのマスから更新できるマス



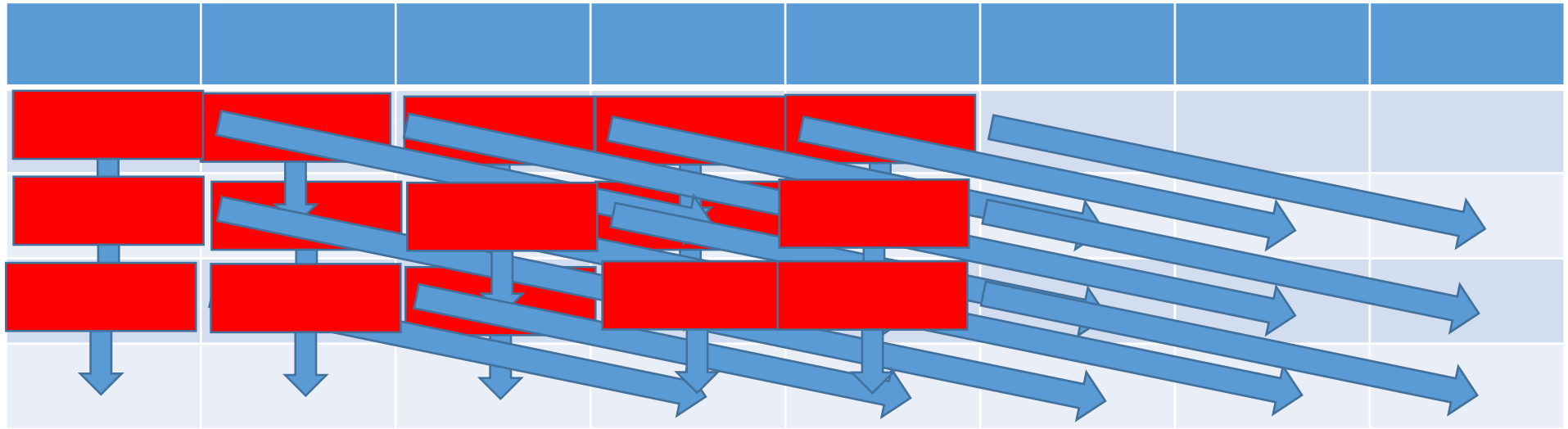
ナップサック問題

ナップサック問題
この二種類



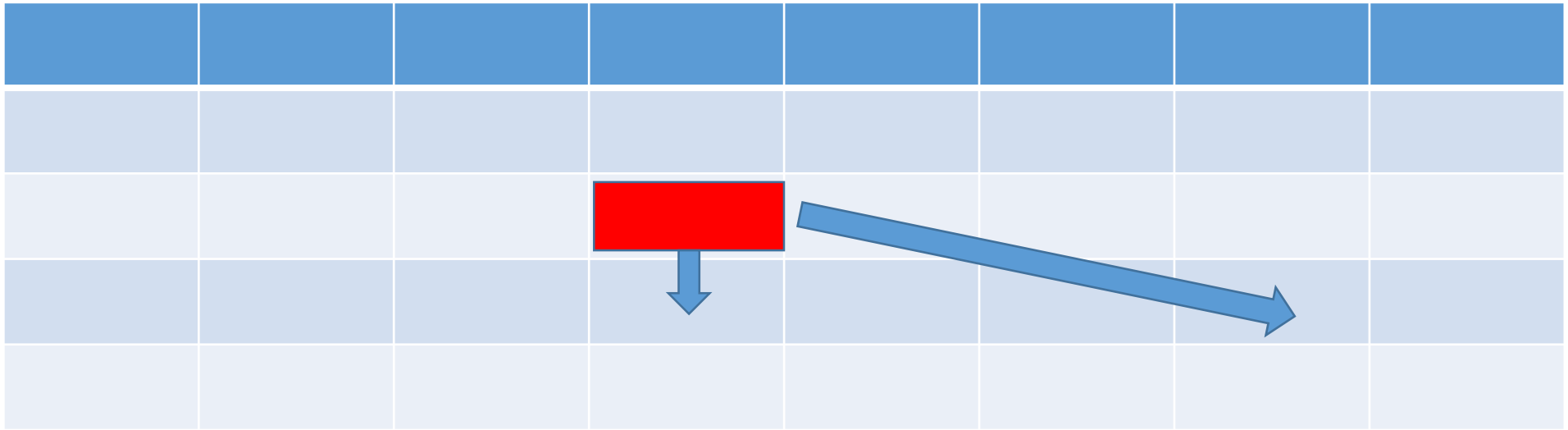
ナップサック問題

ナップサック問題
同じことがすべての頂点に言える



ナップサック問題

dp[i][j]と定義した時に、
i段目からはi+1段目にしか更新が起こらない



ナップサック問題

dpテーブルのマスをグラフの頂点とみると、図の番号付けはトポロジカル順序である！

(自分より若い番号の頂点に更新が起こらない)

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32

DAGとの関係

有向グラフGにおいて
「GはDAGである」⇔「Gのトポロジカルソートが可能」

DAGとの関係

有向グラフGにおいて

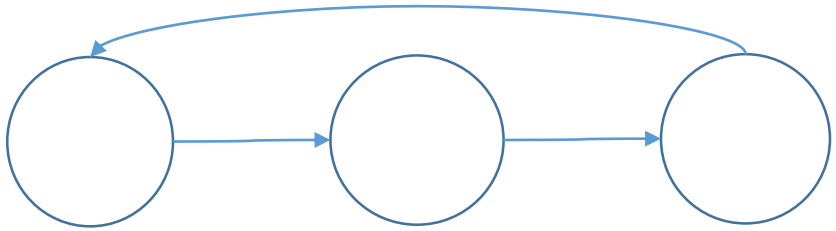
「GはDAGである」⇔「Gのトポロジカルソートが存在」

証明(⇐)

対偶「GはDAGじゃない」⇒「Gのトポロジカルソートは存在しない」を示す

閉路ができていたらトポソが存在しないことは自明

証明終わり



DAGとの関係

有向グラフGにおいて

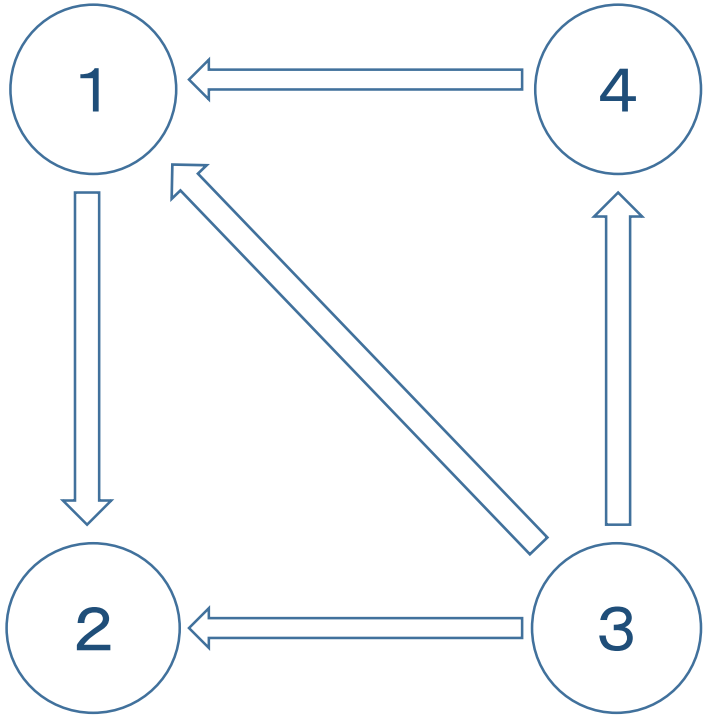
「GはDAGである」⇔「Gのトポロジカルソートが存在」

証明(⇒)

後述のトポロジカルソートを求めるアルゴリズムが構築できる

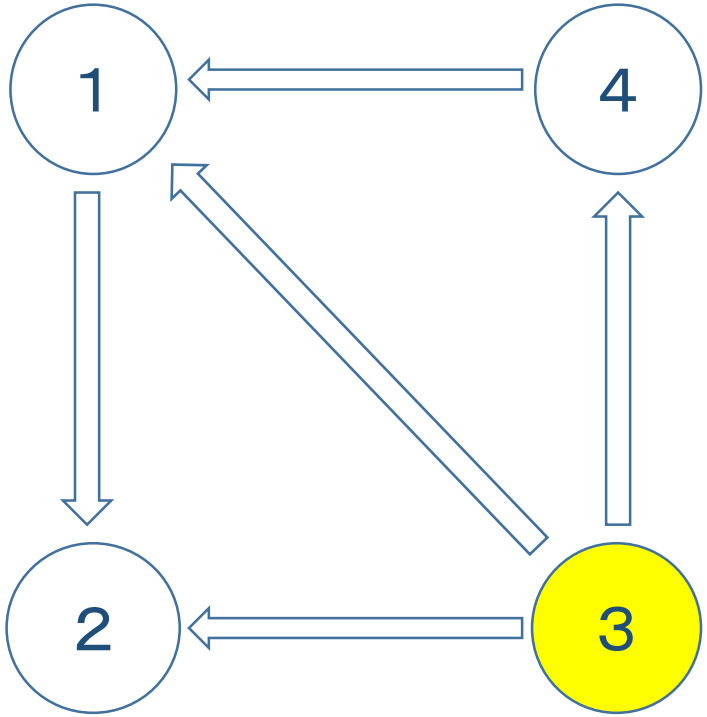
証明終わり

アルゴリズム



アルゴリズム

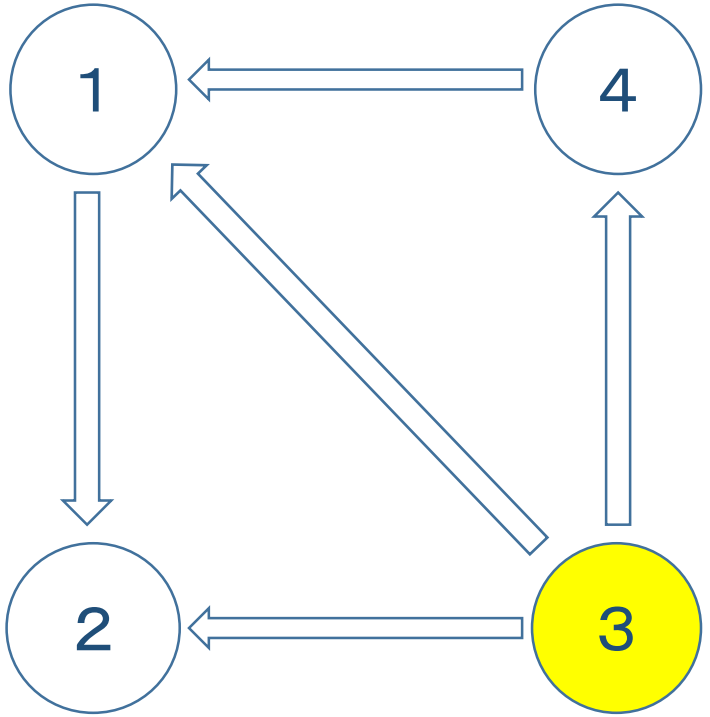
.DAGには入次数が0の頂点が少なくとも一つ存在する



アルゴリズム

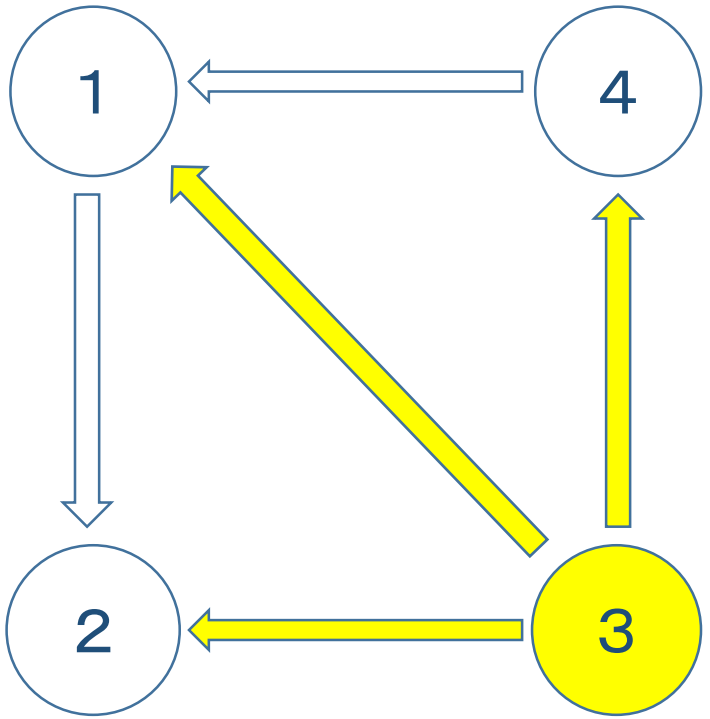
.DAGには入次数が0の頂点が少なくとも一つ存在する

.その頂点はトポロジカルソートの最初の頂点になれる



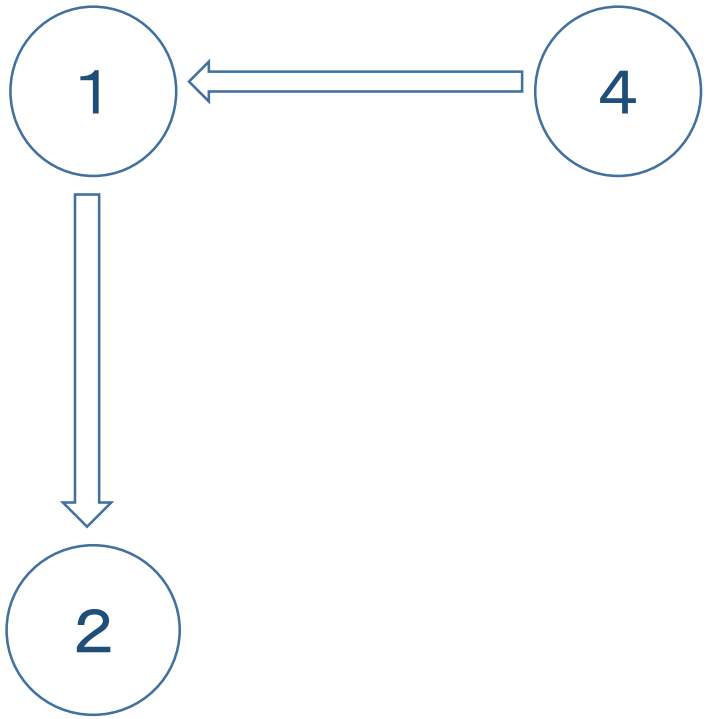
アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く



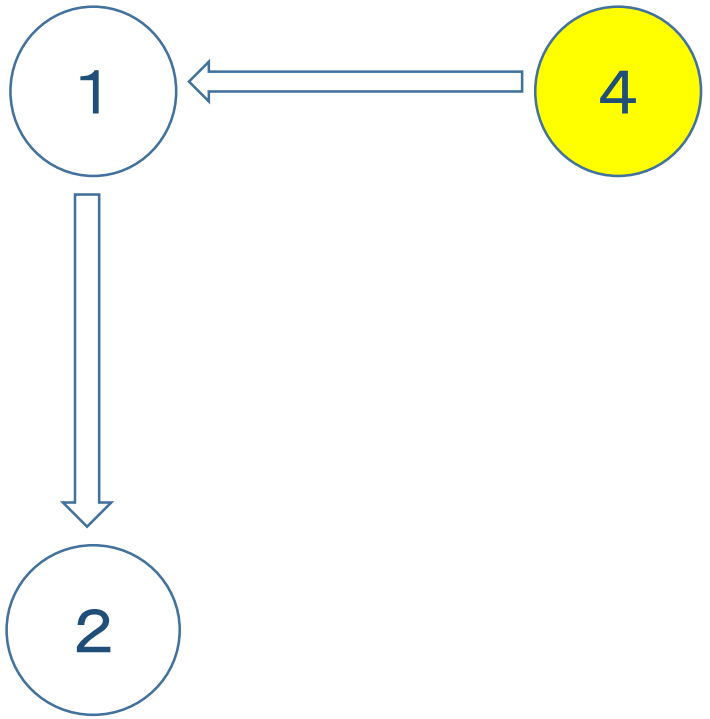
アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである



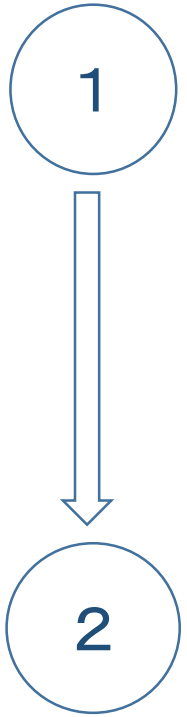
アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである
.以下繰り返し



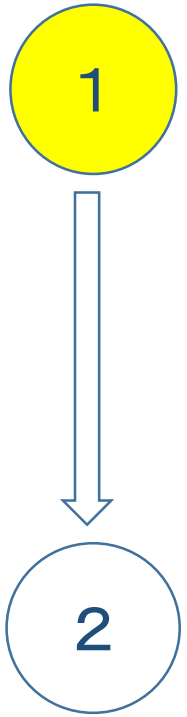
アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである
.以下繰り返し



アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである
.以下繰り返し



アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである
.以下繰り返し

アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである
.以下繰り返し
.トポロジカルソートが複数存在する場合もあるが、その中の一つが構成できる

アルゴリズム

.DAGには入次数が0の頂点が少なくとも一つ存在する
.その頂点はトポロジカルソートの最初の頂点になれる
.その頂点と、その頂点から出る辺を取り除く
.取り除いたグラフもまた、DAGである
.以下繰り返し
.トポロジカルソートが複数存在する場合もある

ほんまか？？？？

DAG⇒入次数0の頂点が少なくとも一つ存在

monkukui「うーんわからん」
monkukui「助けてTAB~~]
TAB　「DAGであり、すべての頂点の入次数が1以上と仮定する。
ここである頂点から、逆辺に移動することを繰り返すことを考えるが、
DAGであるので、同じ頂点は踏まない。
全ての頂点において入次数が1以上であるので、この操作は無限回行えるが、
頂点は有限個を想定しているので矛盾。
(証明終わり)」
monkukui「はいプロ」

```cpp
//トポソ
//

#include<iostream>
#include<vector>
#include<queue>
#include<cstring>
using namespace std;

int main(){

    int n, m; cin >> n >> m;
    vector<vector<int> > adj(n);
    vector<int> d(n, 0);
    for(int i = 0; i < m; i++){
        int a, b; cin >> a >> b;
        d[b]++;
        adj[a].push_back(b);
    }

    queue<int> q;

    //入次数が0の頂点queueに入れる
    for(int i = 0; i < n; i++){
        if(d[i] == 0) q.push(i);
    }

    //トポロジカル順に頂点を格納
    vector<int> ans;

    while(!q.empty()){

        int v = q.front();
        q.pop();
        ans.push_back(v);

        //vから伸びている有向辺をなめ,到達できる頂点の入次数を1減らす
        for(int i = 0; i < (int)adj[v].size(); i++){
            d[adj[v][i]]--;

            //辺を取り除いた後に入次数が0になったらqueueにpushする
            if(d[adj[v][i]] == 0) q.push(adj[v][i]);
        }

    }

    for(int i = 0; i < (int)ans.size(); i++) cout << ans[i] << endl;
    return 0;
}
```

$$O(V + E)$$

アルゴリズム

別解

すべての頂点からdfsをはじめ、その帰りがけ順がトポロジカルソートである

帰りがけ順 = 最後にその頂点を訪れるタイミング

トポロジカルソートの最後に来れる $O(V+E)$

目次

・グラフの話

-基礎

-dfs

・トポロジカルソート

・強連結成分分解

・問題への応用

強連結成分分解

有向グラフGの頂点の部分集合をSとする

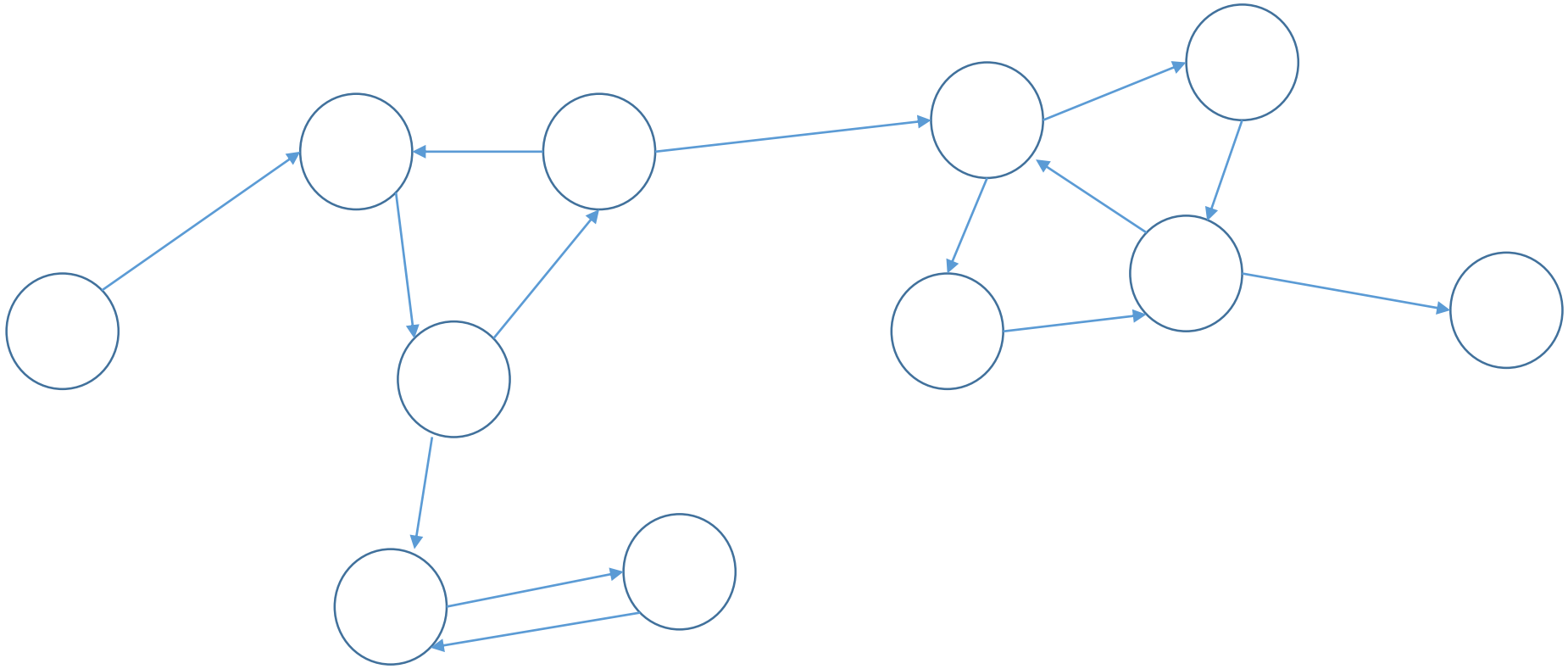
・Sが強連結であるとは

- 任意2頂点u, v(∈ S)を取ったときuからvに到達できる

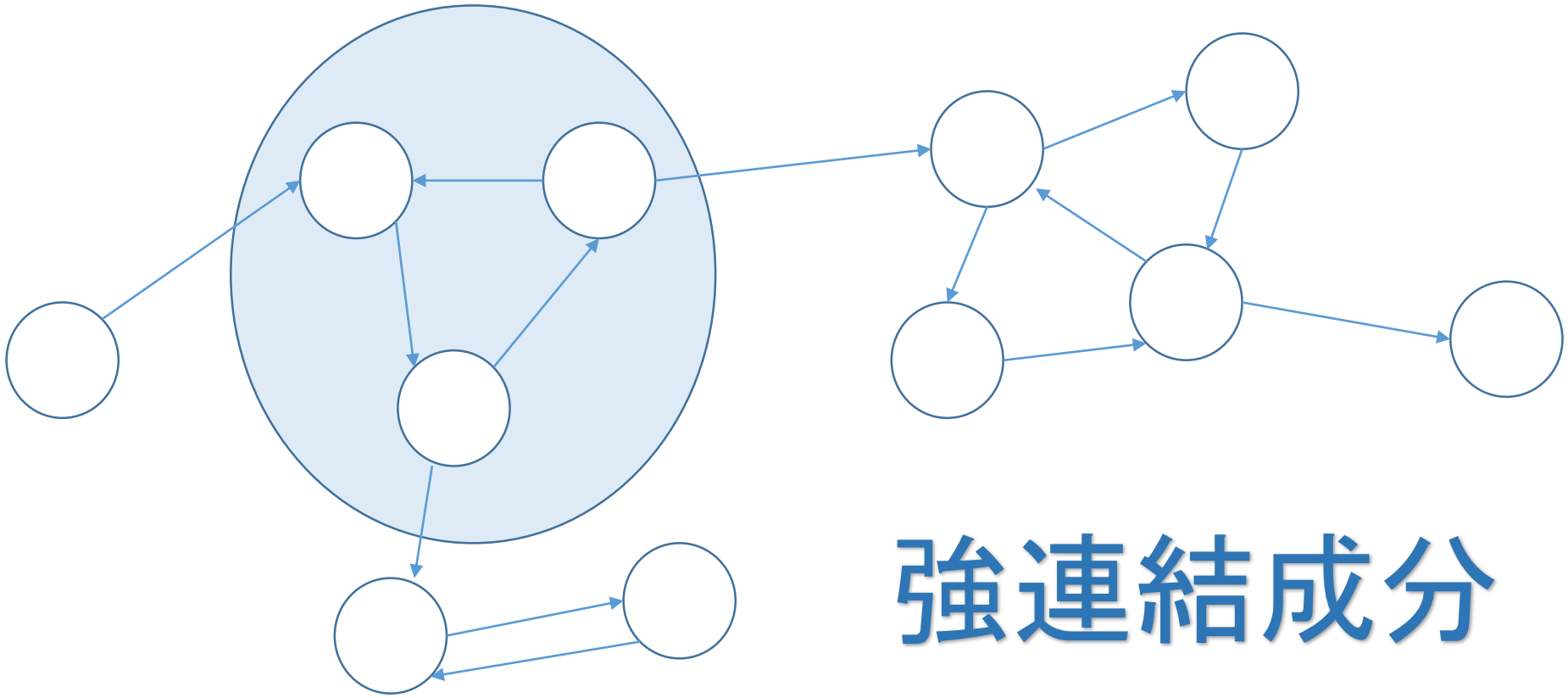
・Sが強連結成分である

- 強連結な集合Sに、他のどの頂点集合を付け加えても強連結にならない

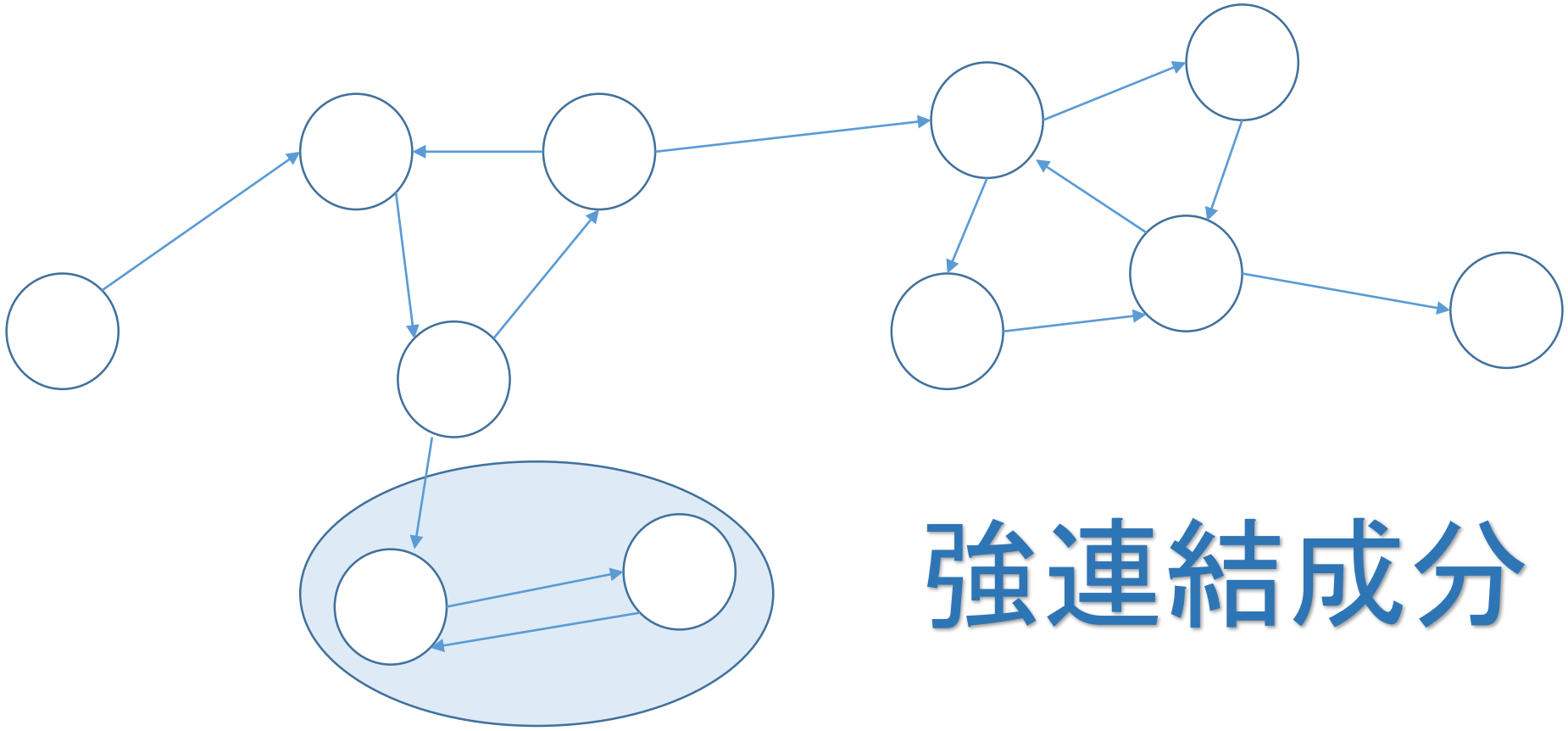
強連結成分分解



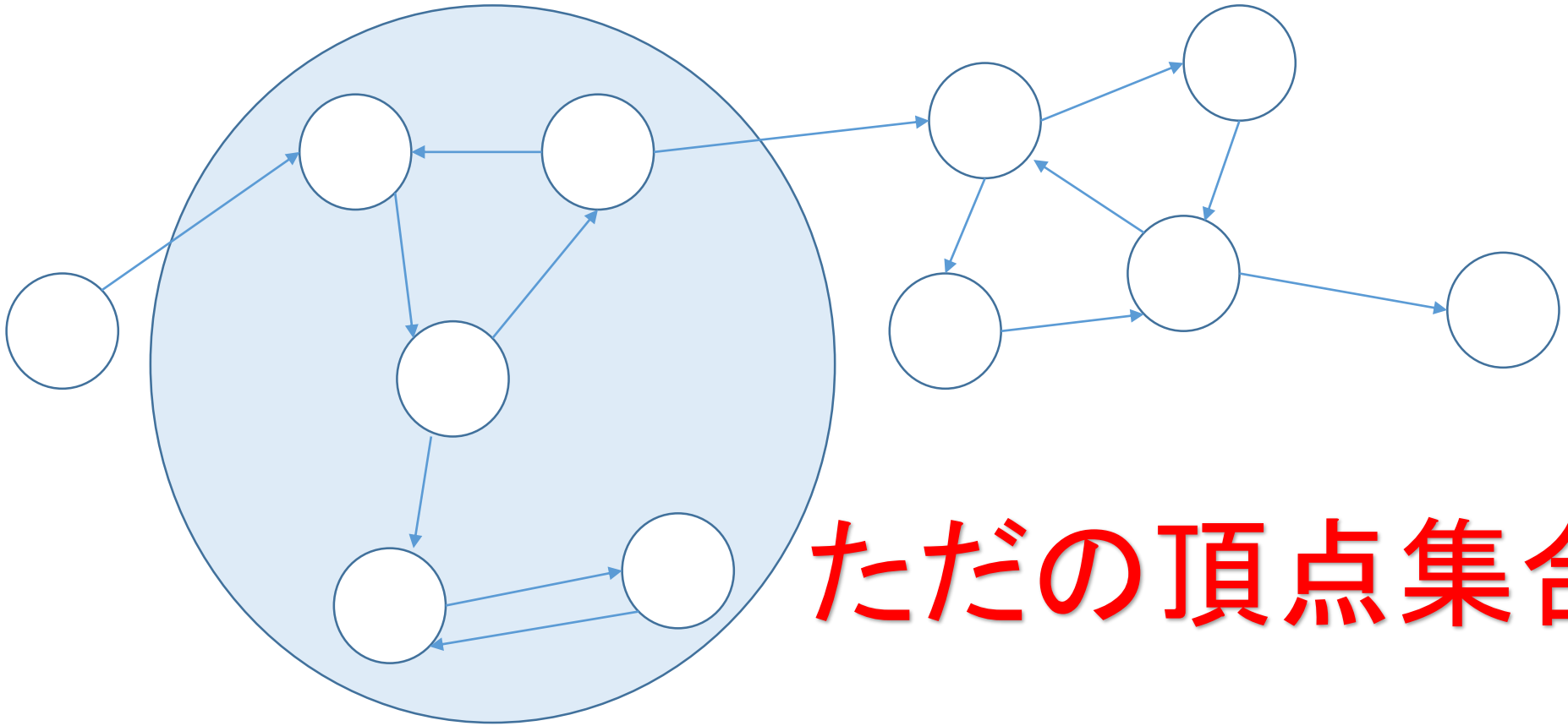
強連結成分分解



強連結成分分解

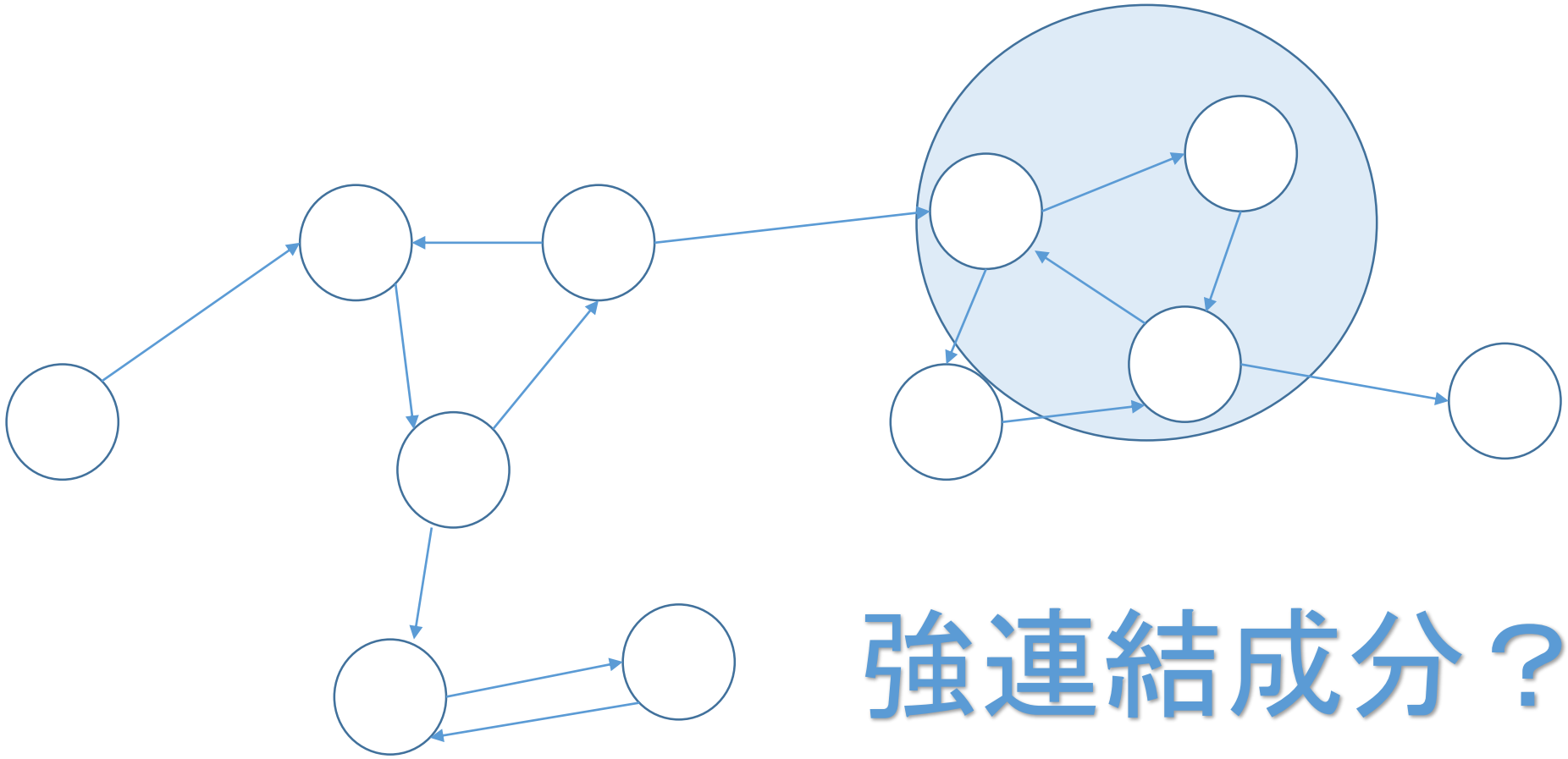


強連結成分分解



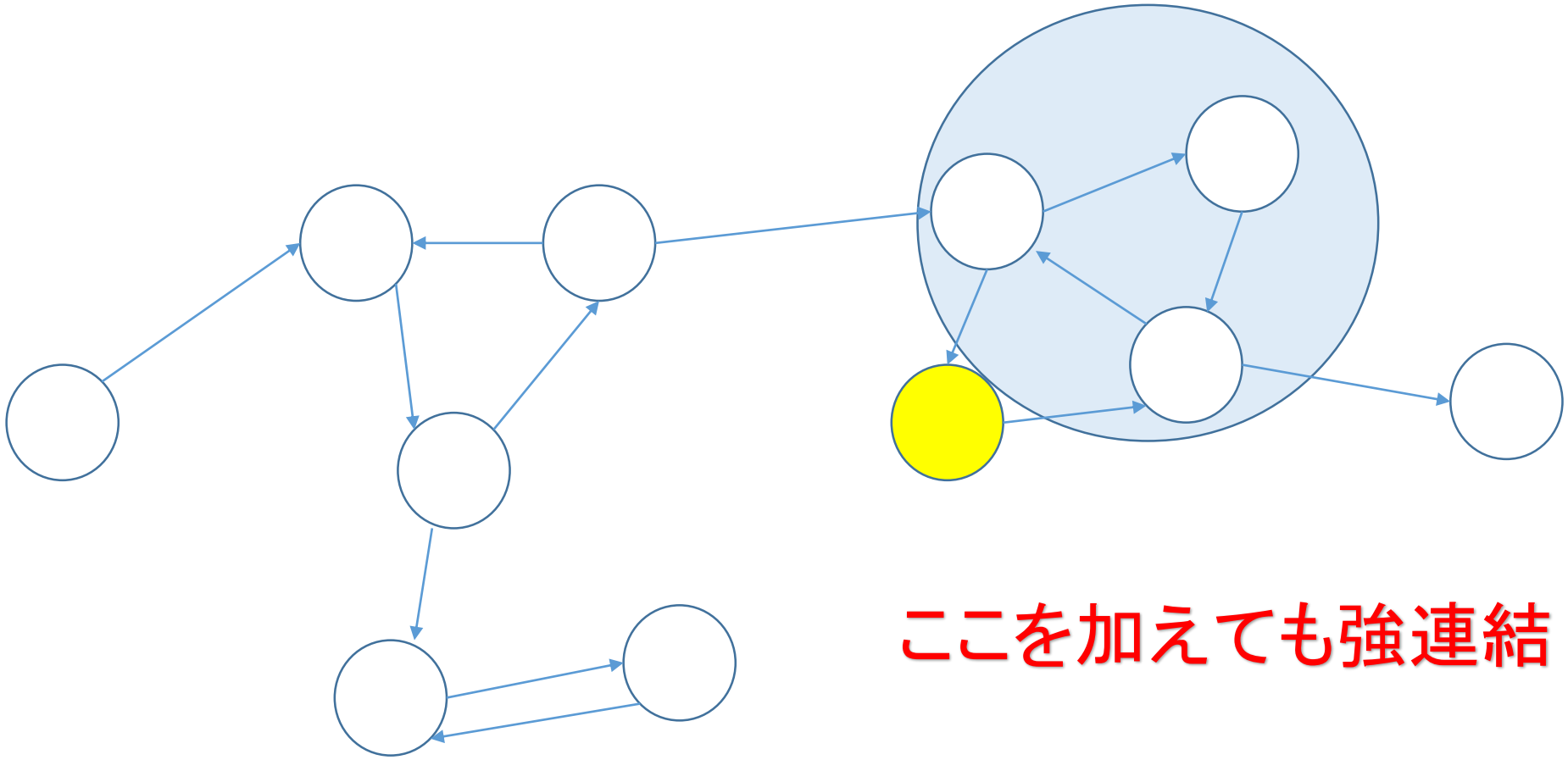
ただの頂点集合

強連結成分分解



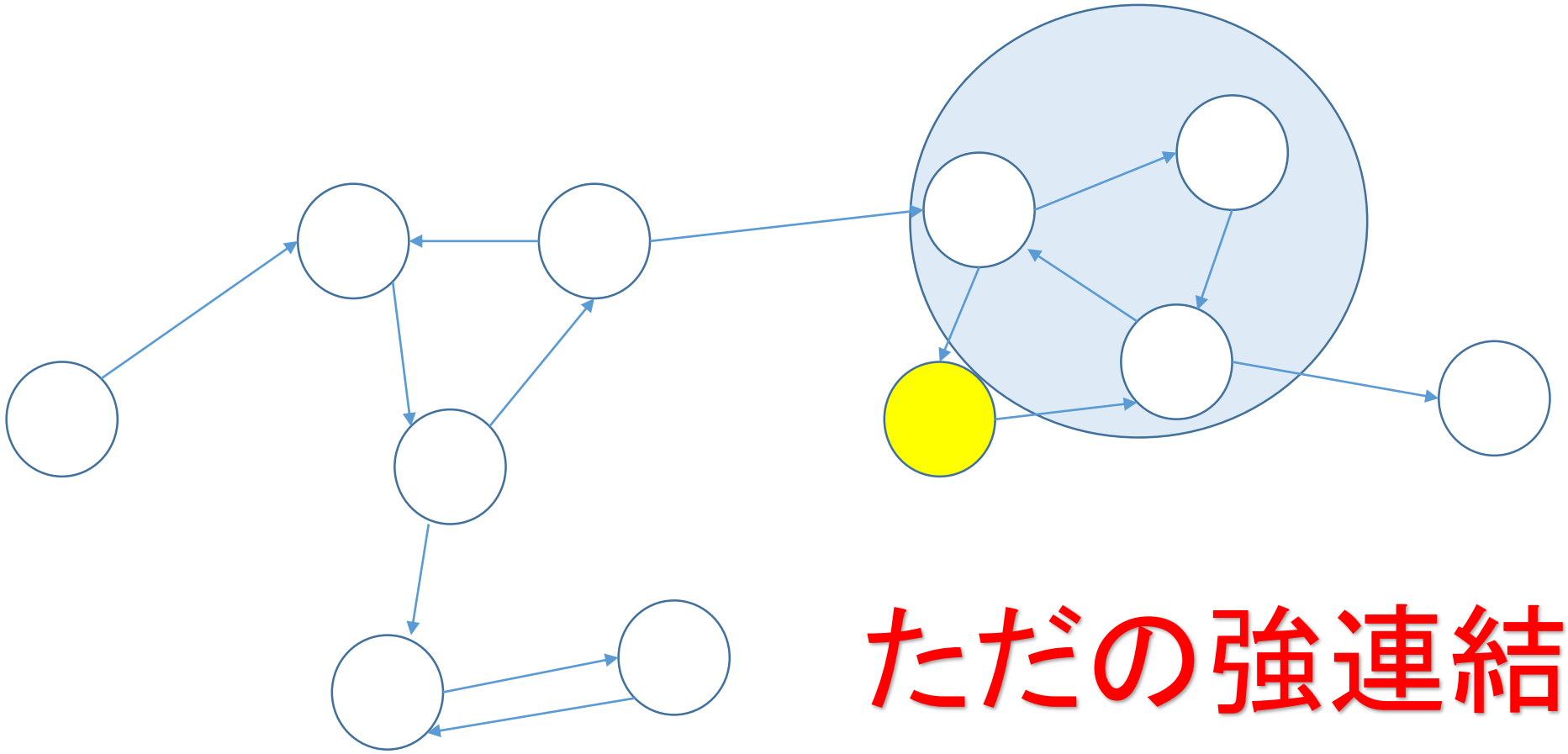
強連結成分？

強連結成分分解



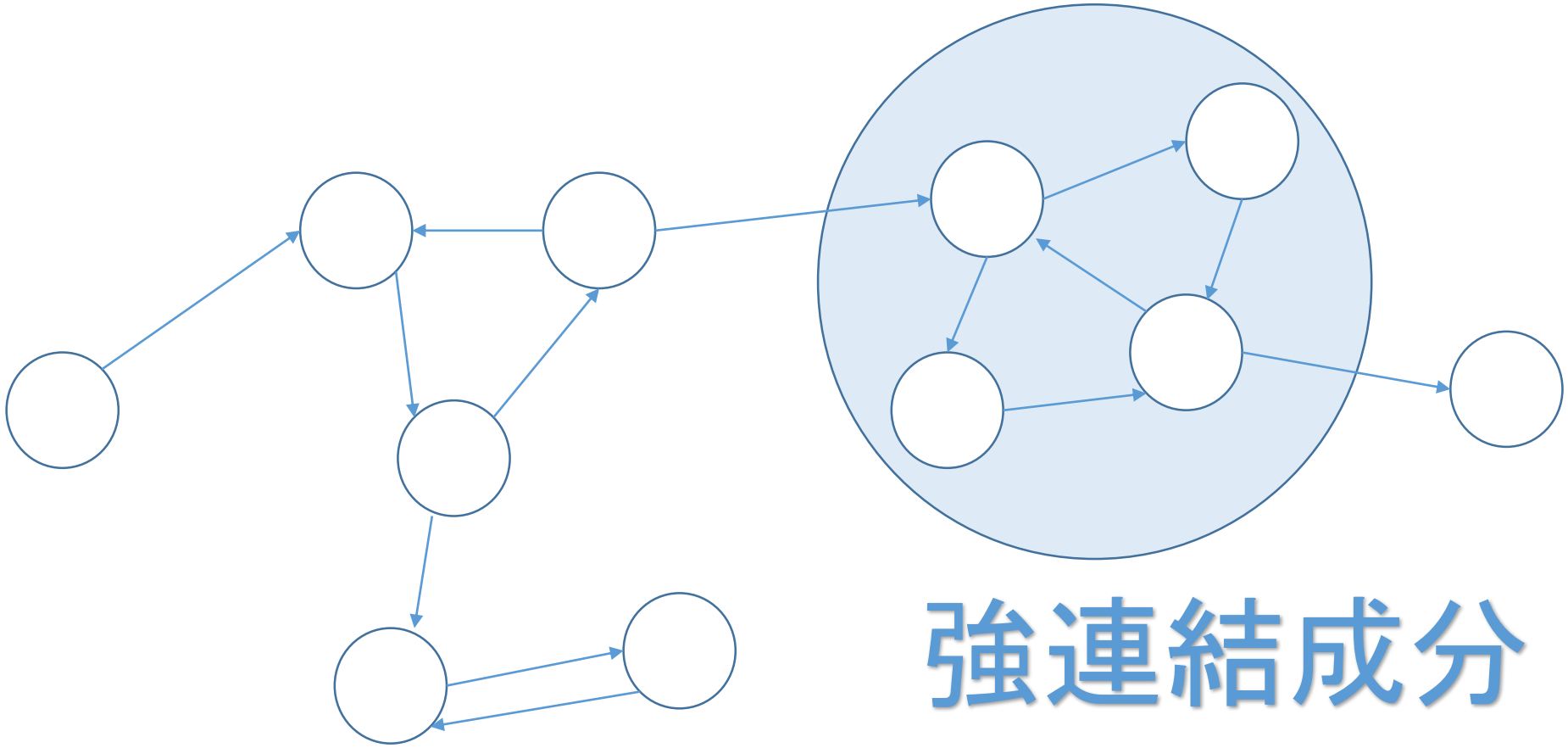
ここを加えても強連結

強連結成分分解



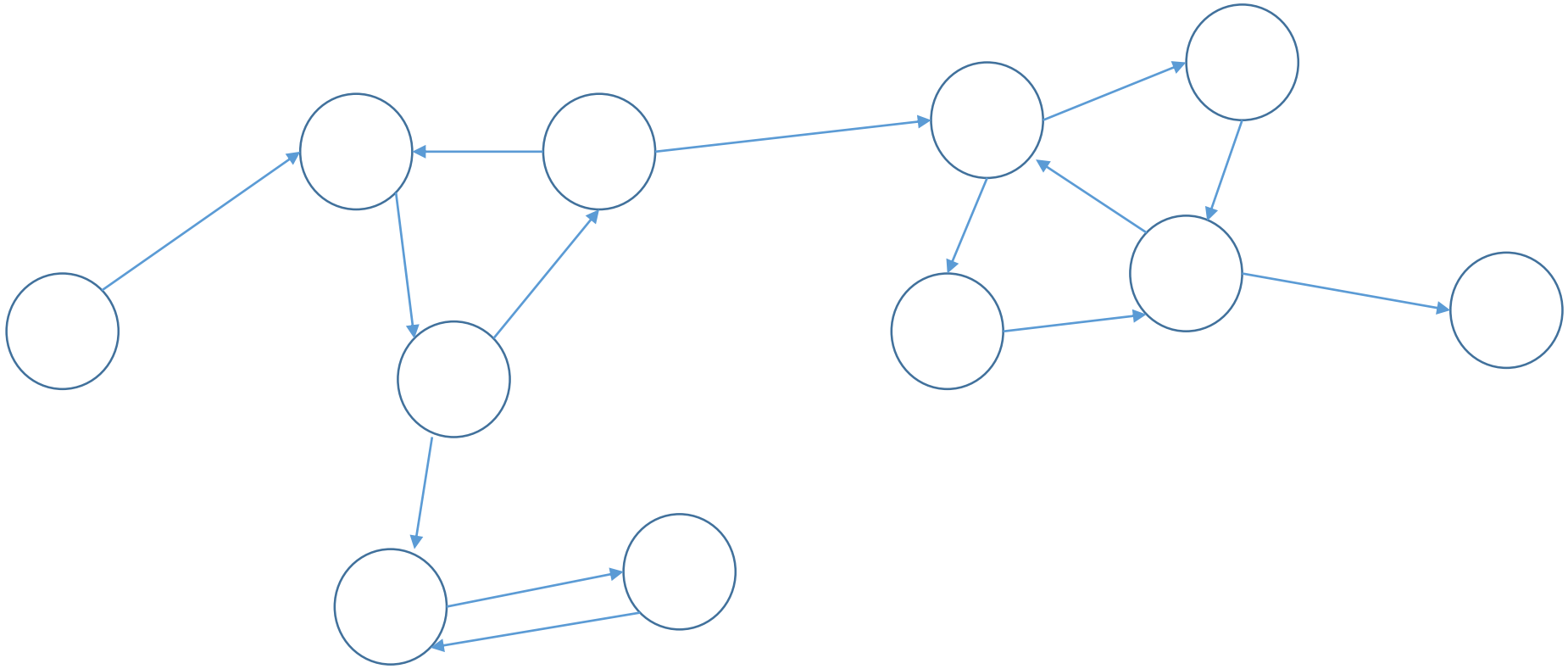
ただの強連結

強連結成分分解



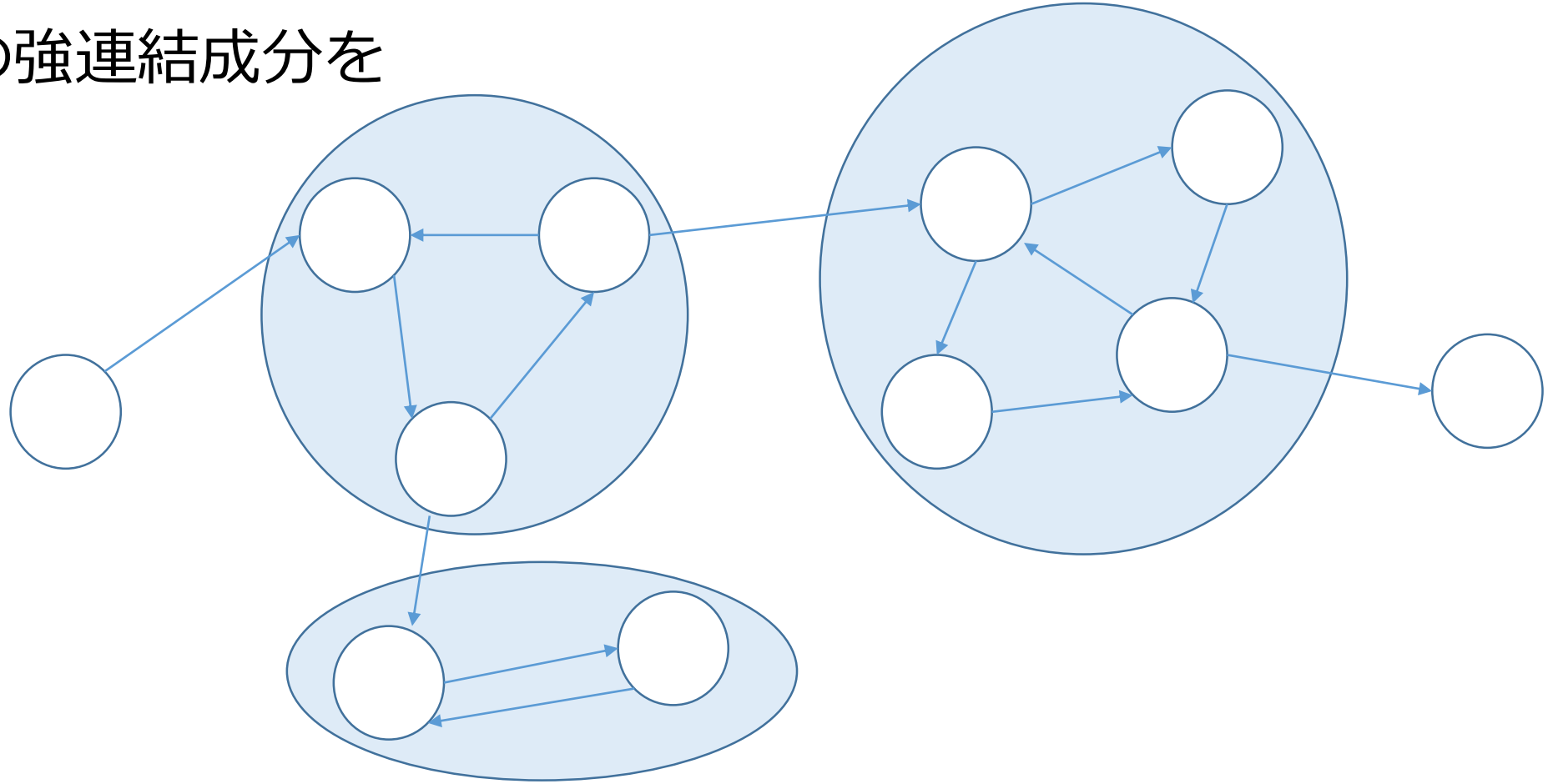
強連結成分分解

強連結成分分解とは



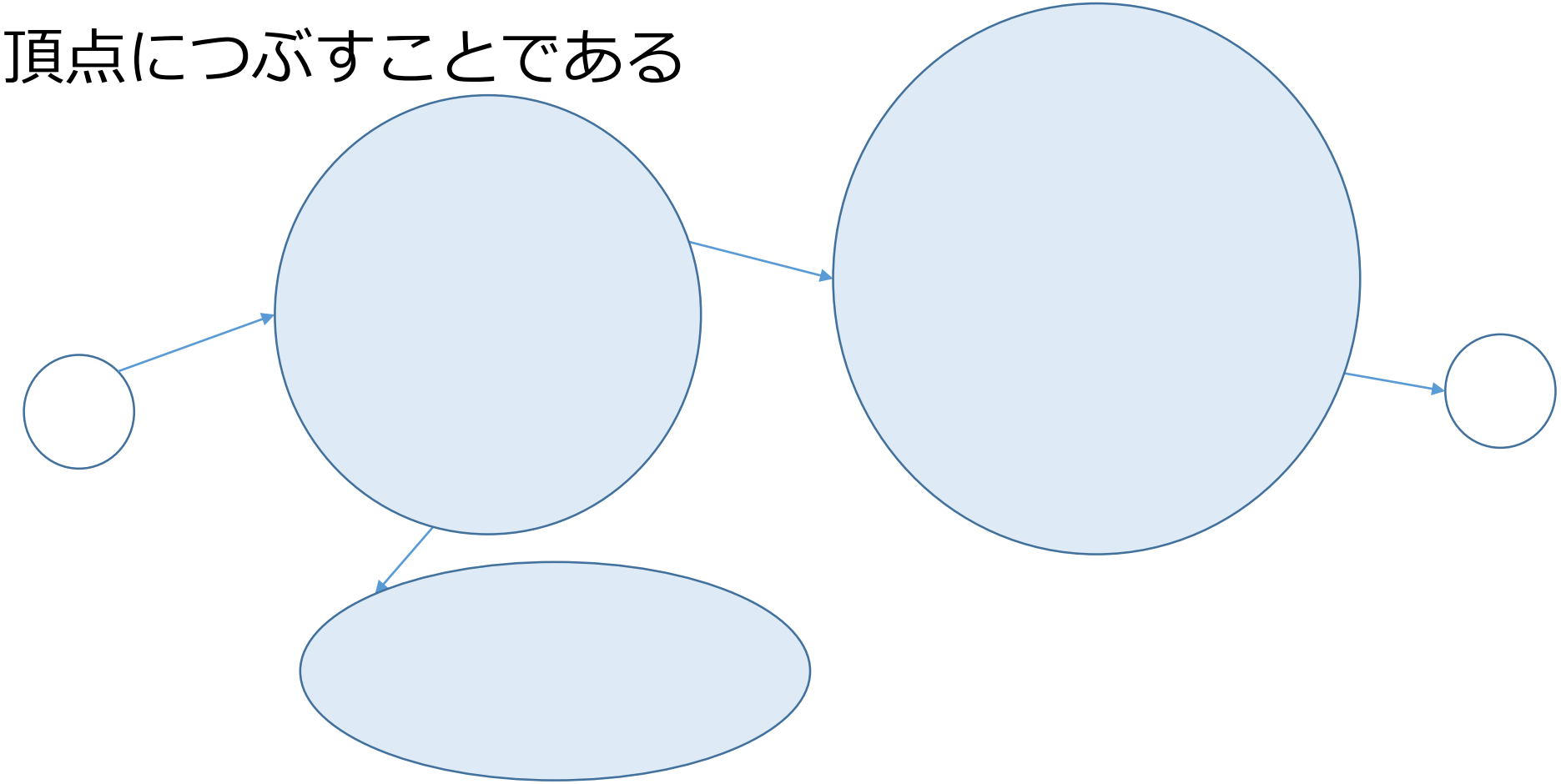
強連結成分分解

全ての強連結成分を



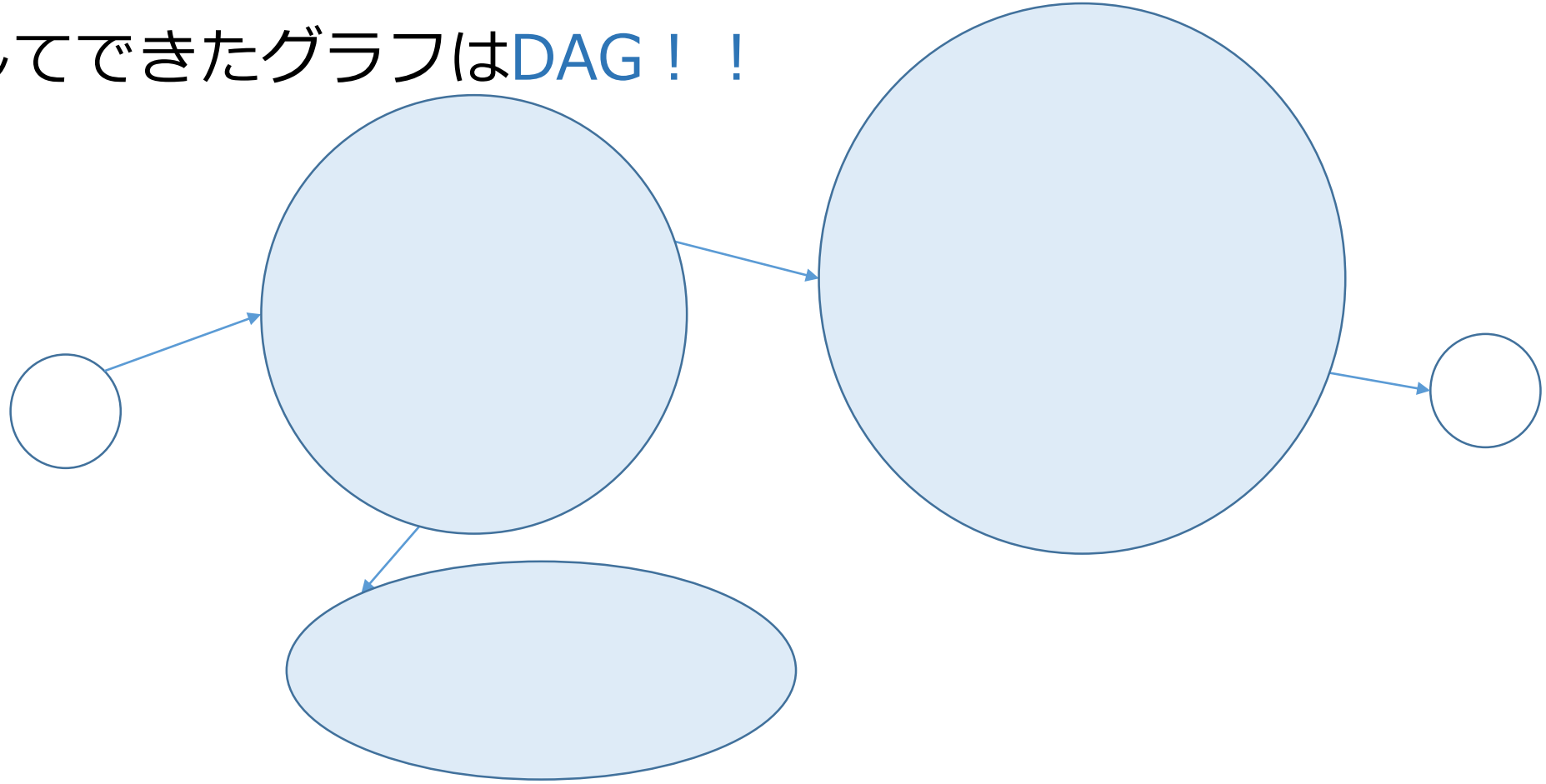
強連結成分分解

1つの頂点につぶすことである



強連結成分分解

そうしてできたグラフはDAG！！



強連結成分分解

・何がうれしいか
- 非DAGをDAGにして、DPなり、なんなりできる！
- 2-SATが解ける!(らしい)
- アドホックでも大活躍！

アルゴリズム

・2回DFSをするだけ

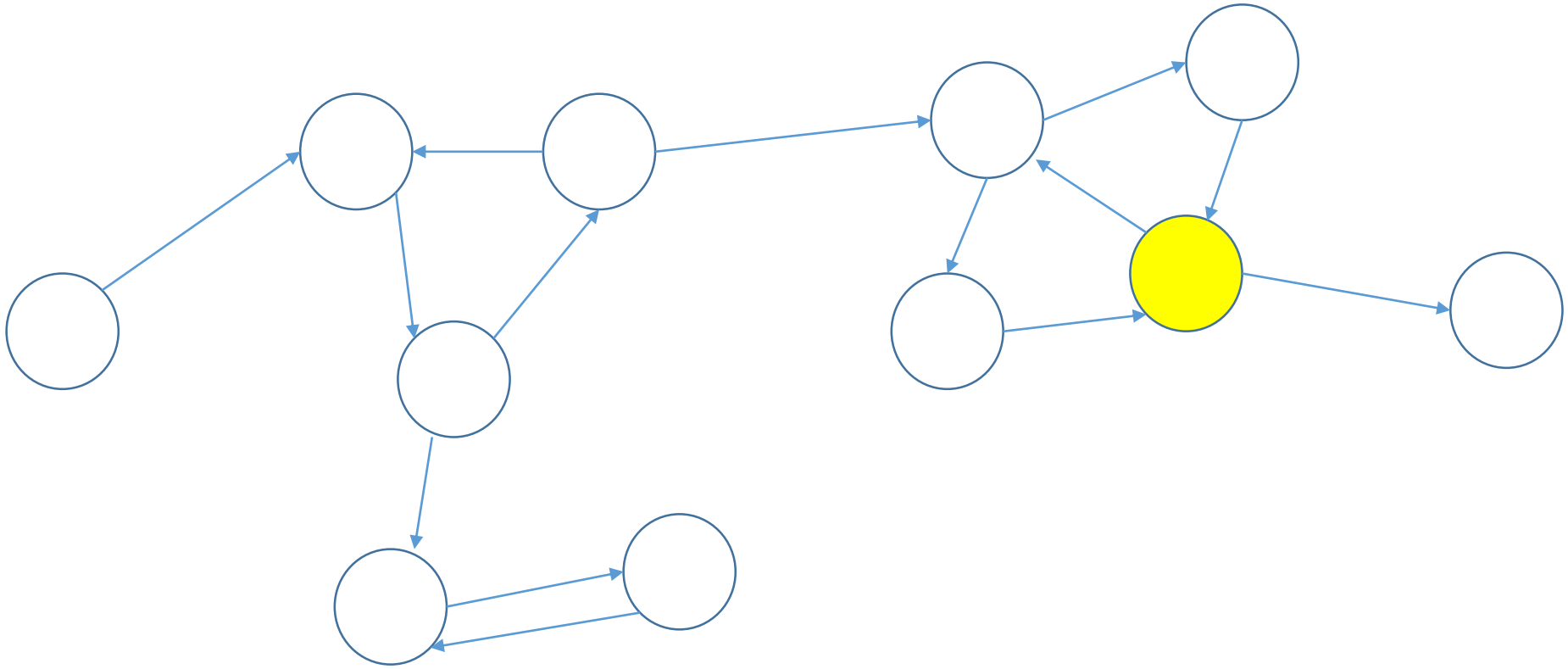
・適当な頂点から一回目のDFSをし、帰りがけ順に番号を振る
・まだ通ってない頂点があるなら再びそこから始めるということを繰り返す

・辺をすべて逆向にし,番号が一番大きい頂点から二回目のDFSを始める
・通ることができた頂点が1つの強連結成分(もしくはただ一つの頂点)となる
・まだ通っていない頂点があれば、その中で番号が一番大きなところから再びDFSをする

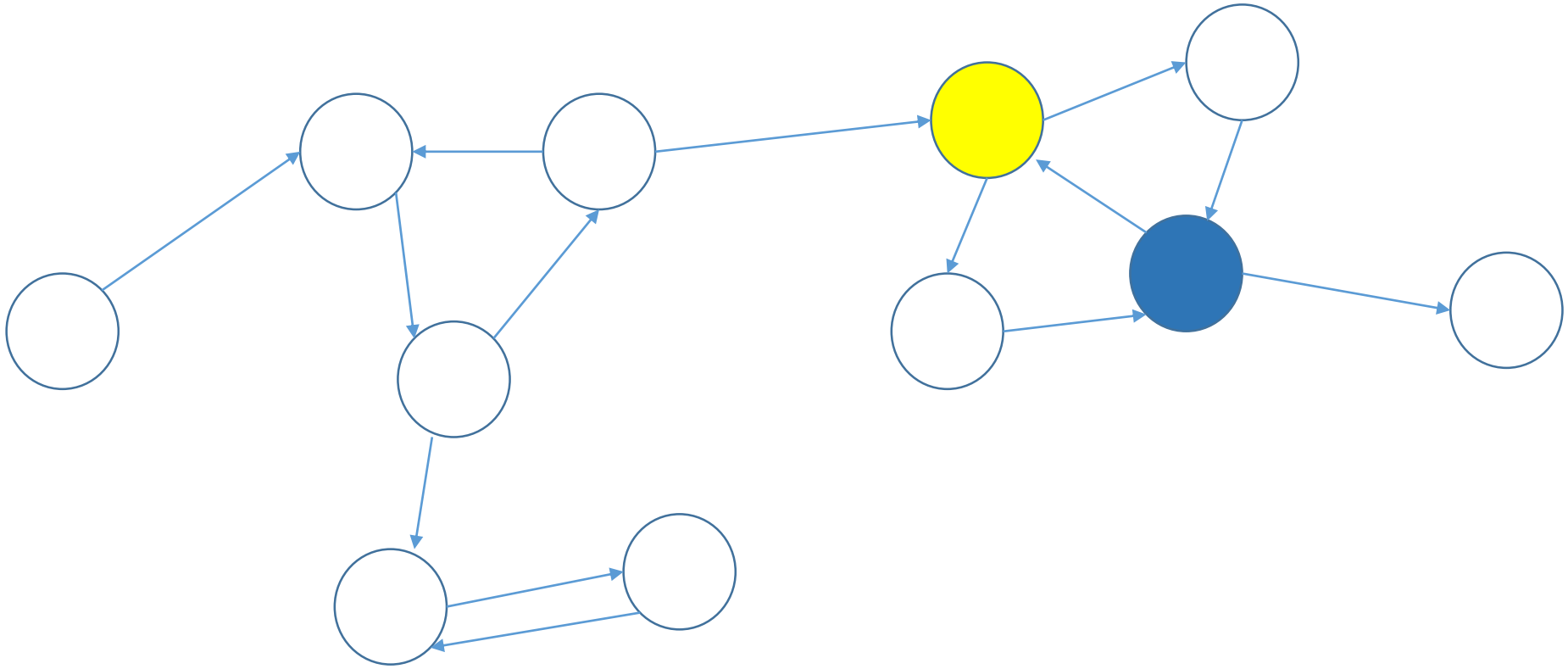
それでは
ノーカットで
ご覧ください

アルゴリズム(１回目のDFS)

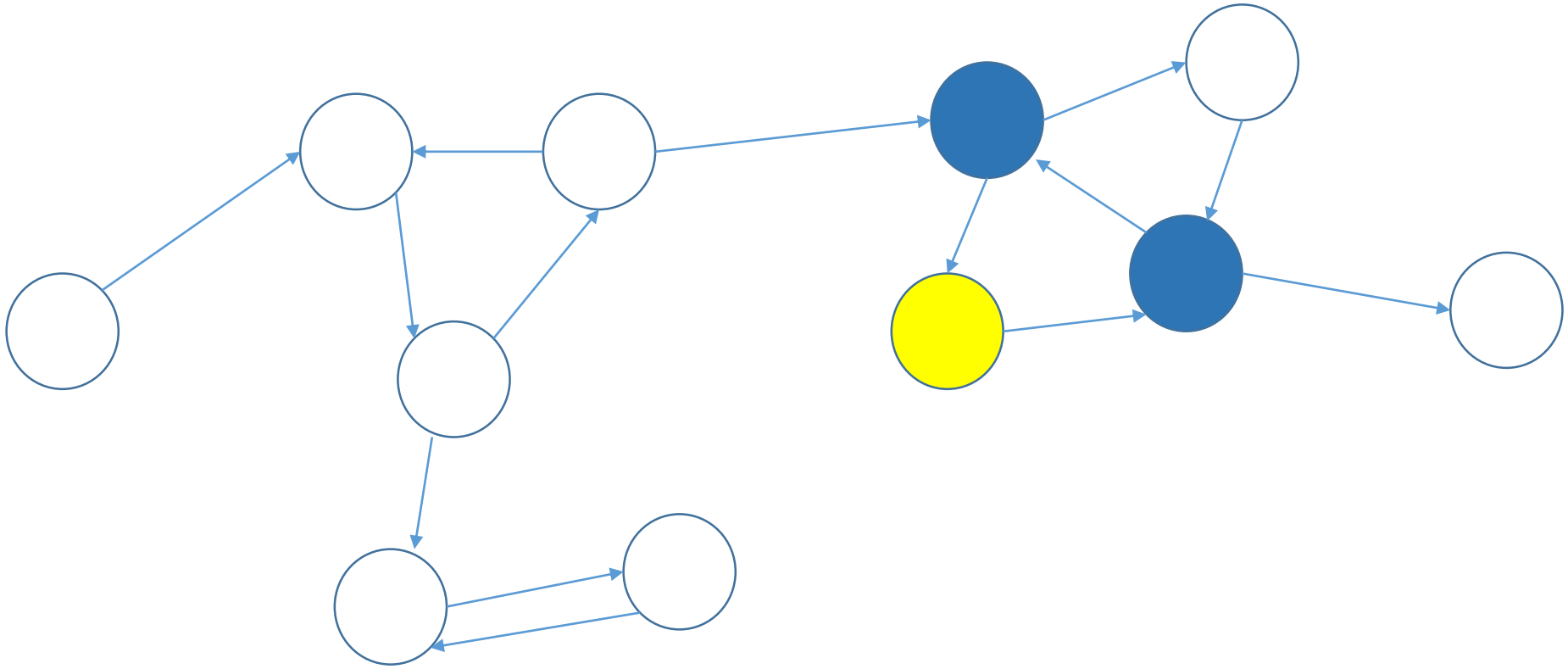
適当な頂点からDFS開始



アルゴリズム(１回目のDFS)

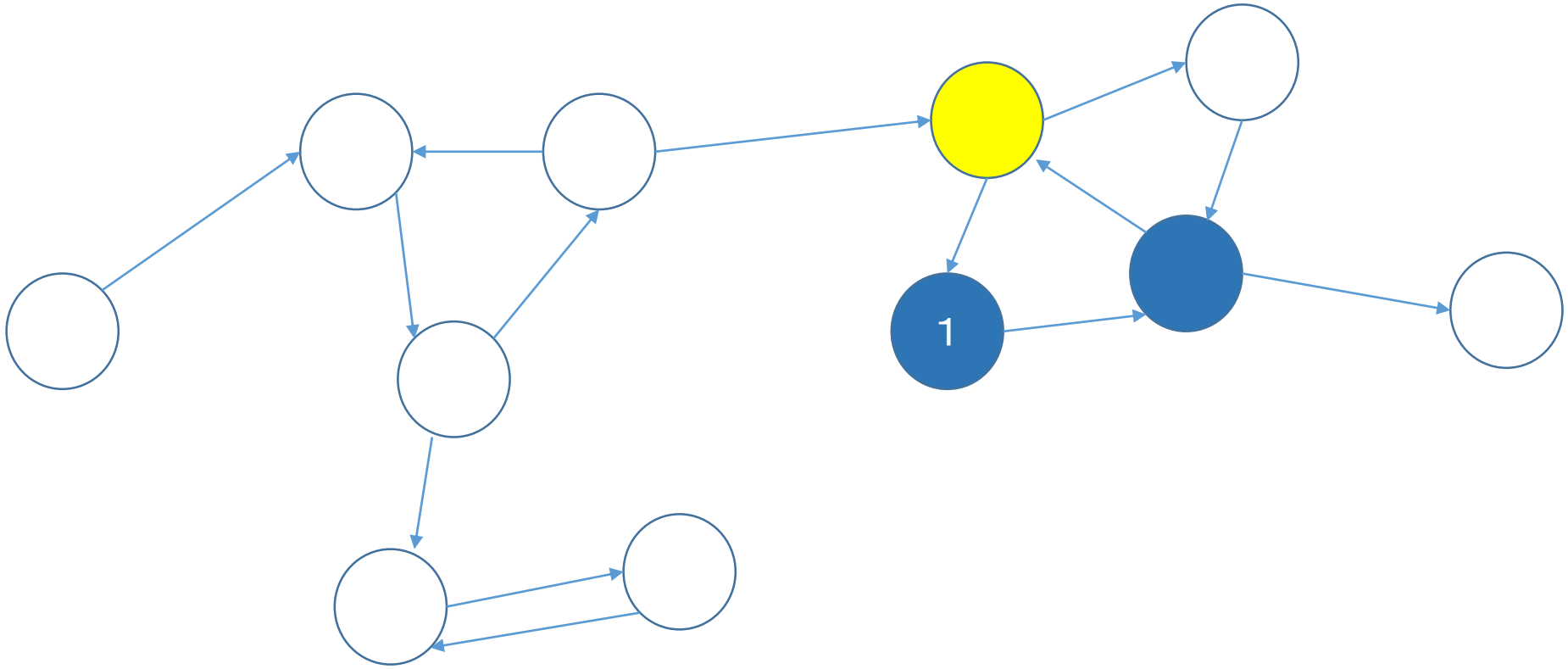


アルゴリズム(１回目のDFS)

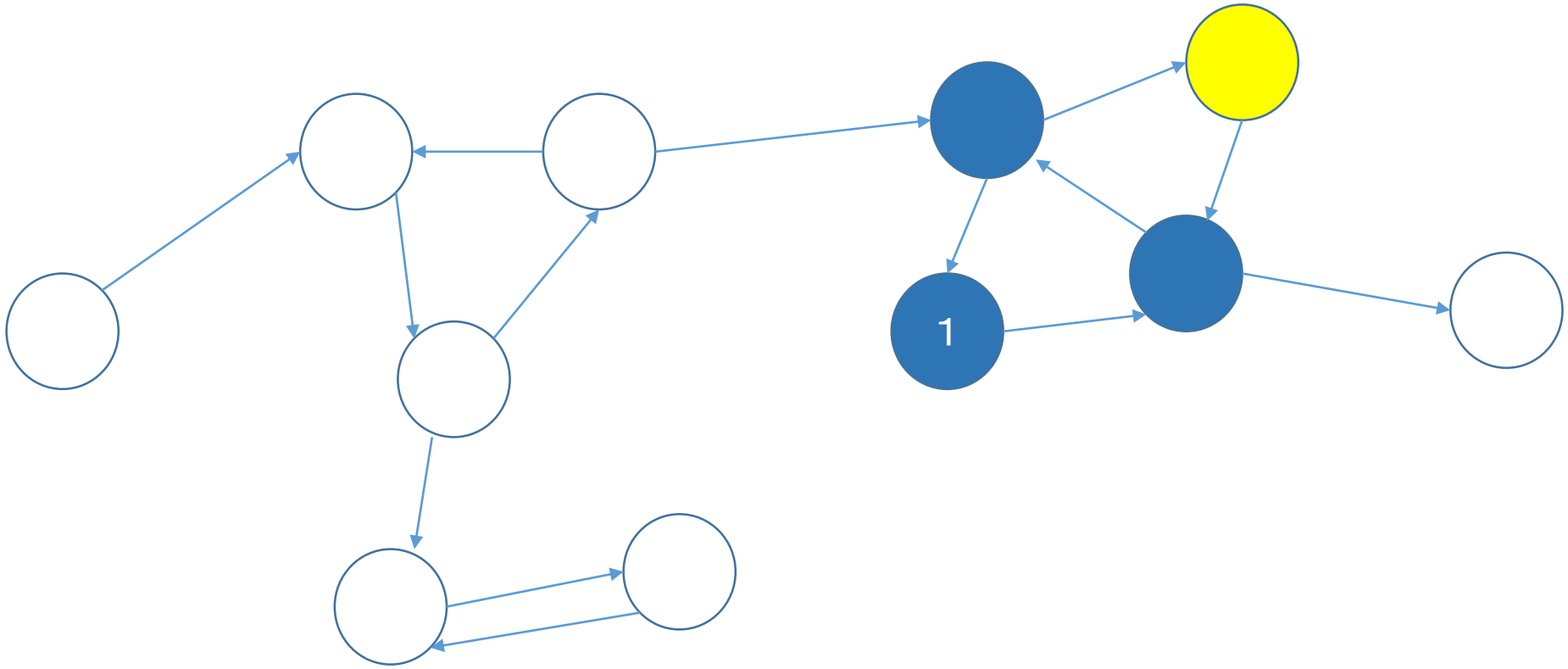


アルゴリズム(１回目のDFS)

帰りがけに番号1を振る

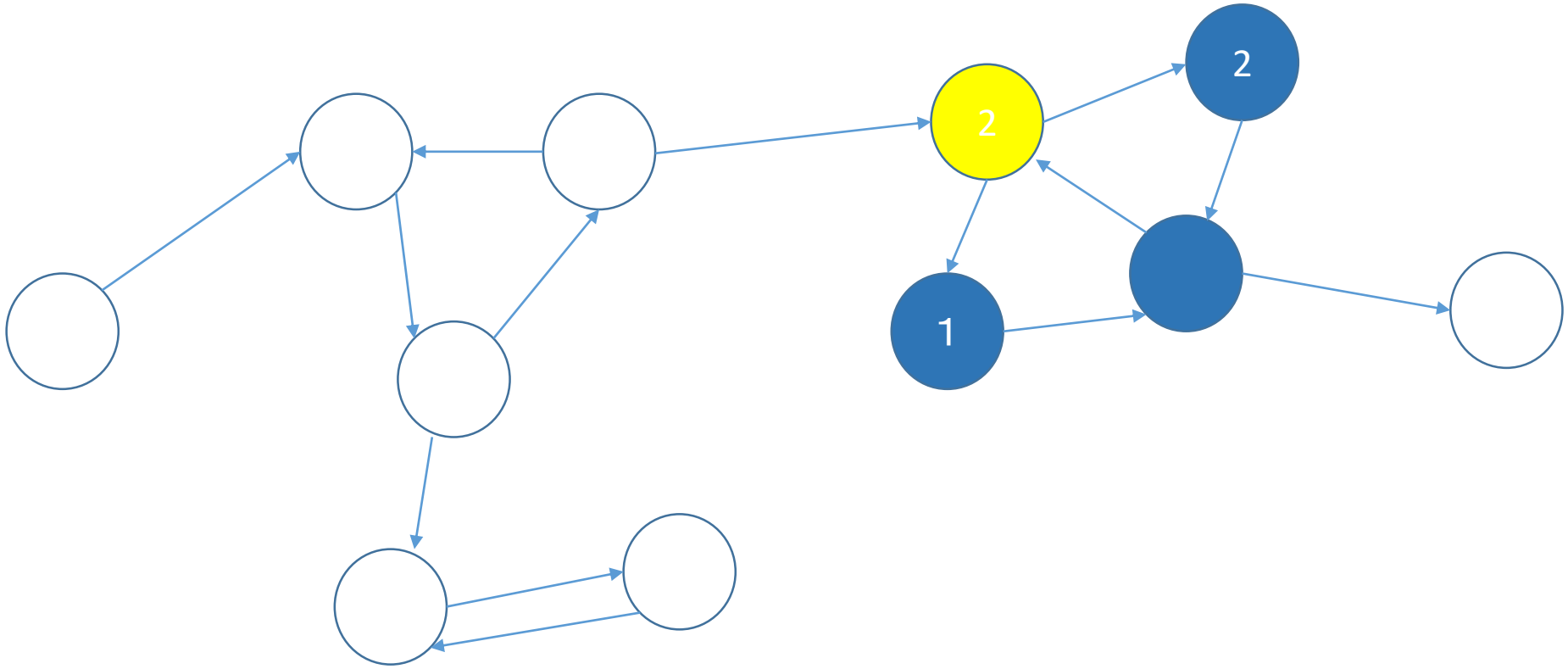


アルゴリズム(１回目のDFS)



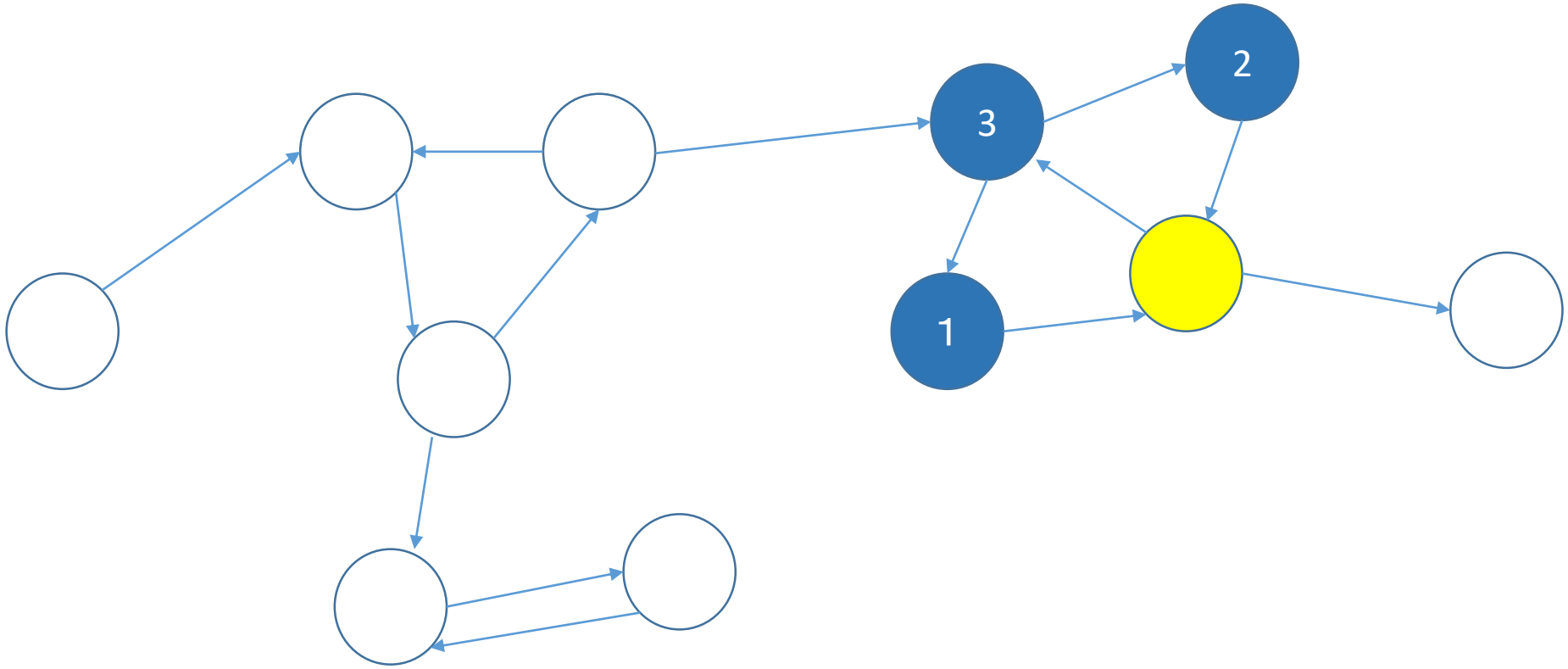
アルゴリズム(１回目のDFS)

帰りがけに番号3を振る

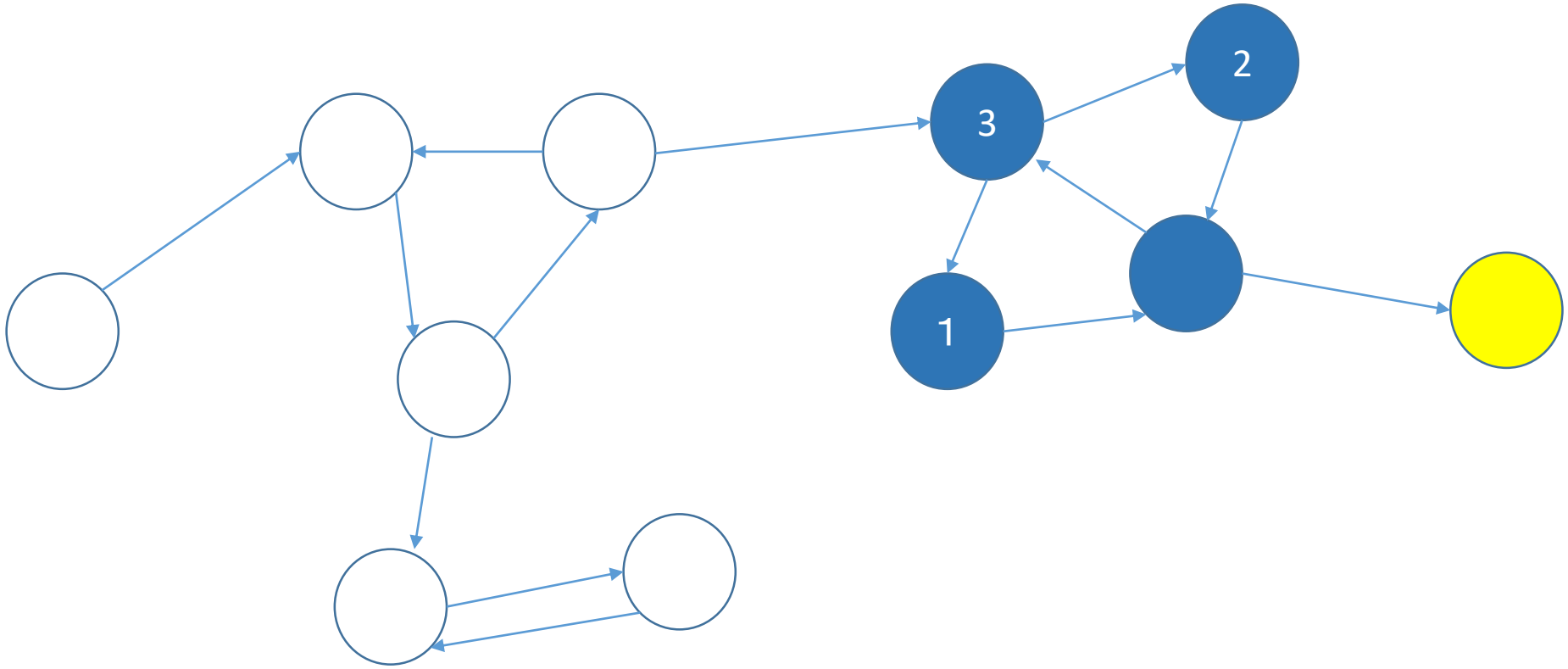


アルゴリズム(１回目のDFS)

帰りがけに番号2を振る

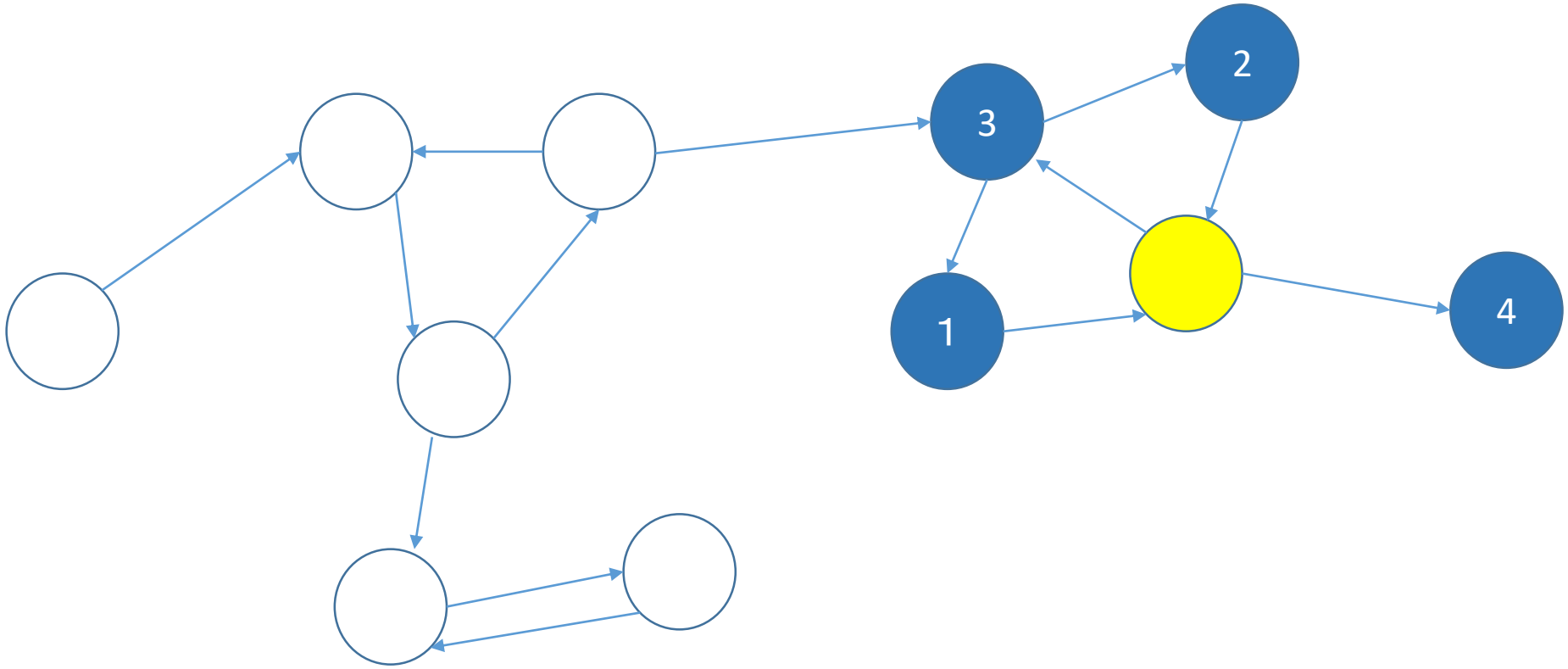


アルゴリズム(１回目のDFS)



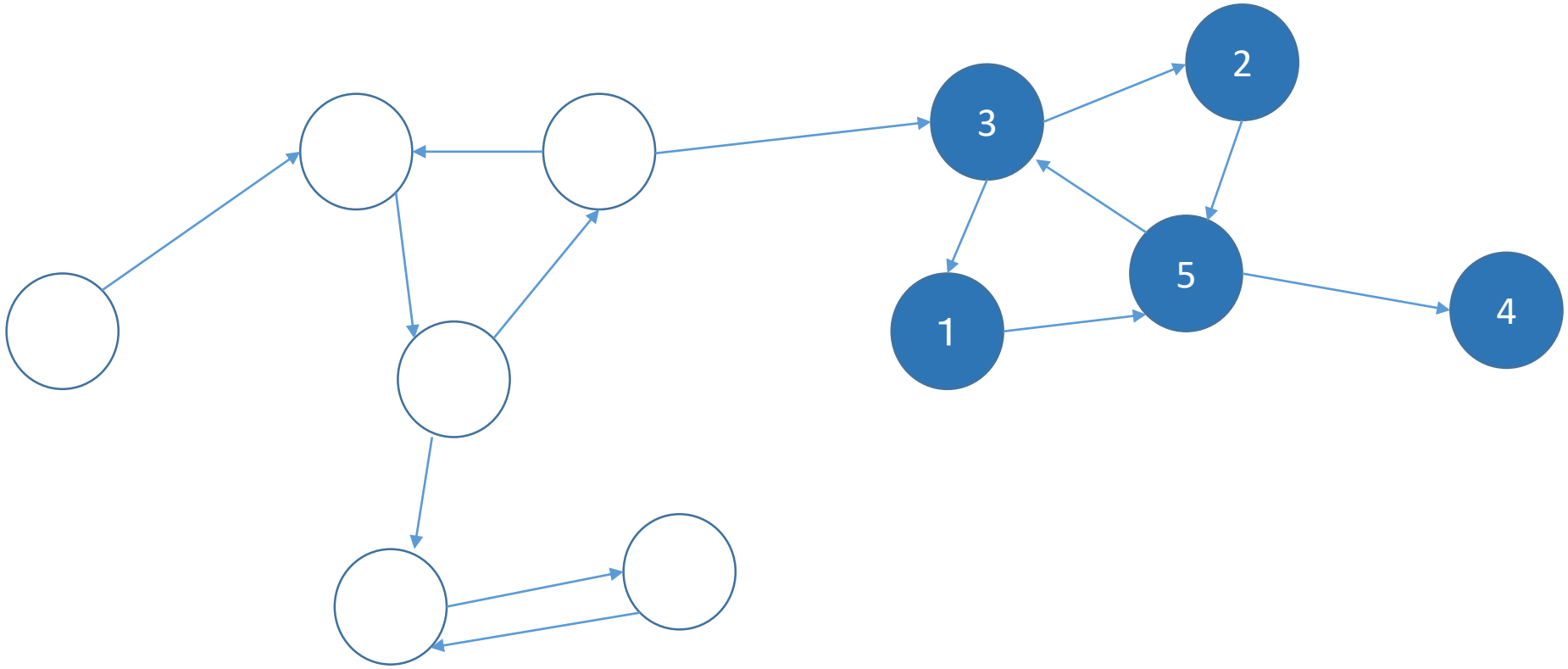
アルゴリズム(１回目のDFS)

帰りがけに番号4を振る



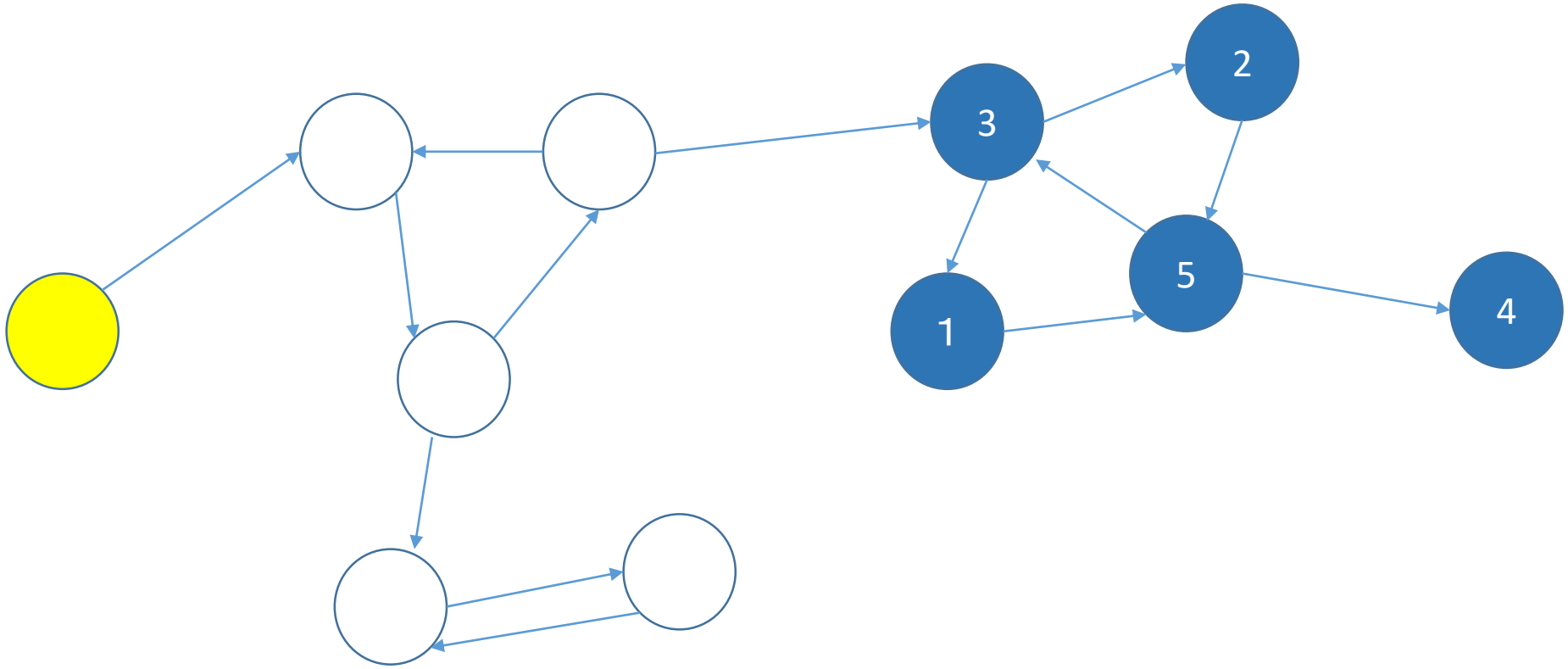
アルゴリズム(１回目のDFS)

帰りがけに番号5を振る

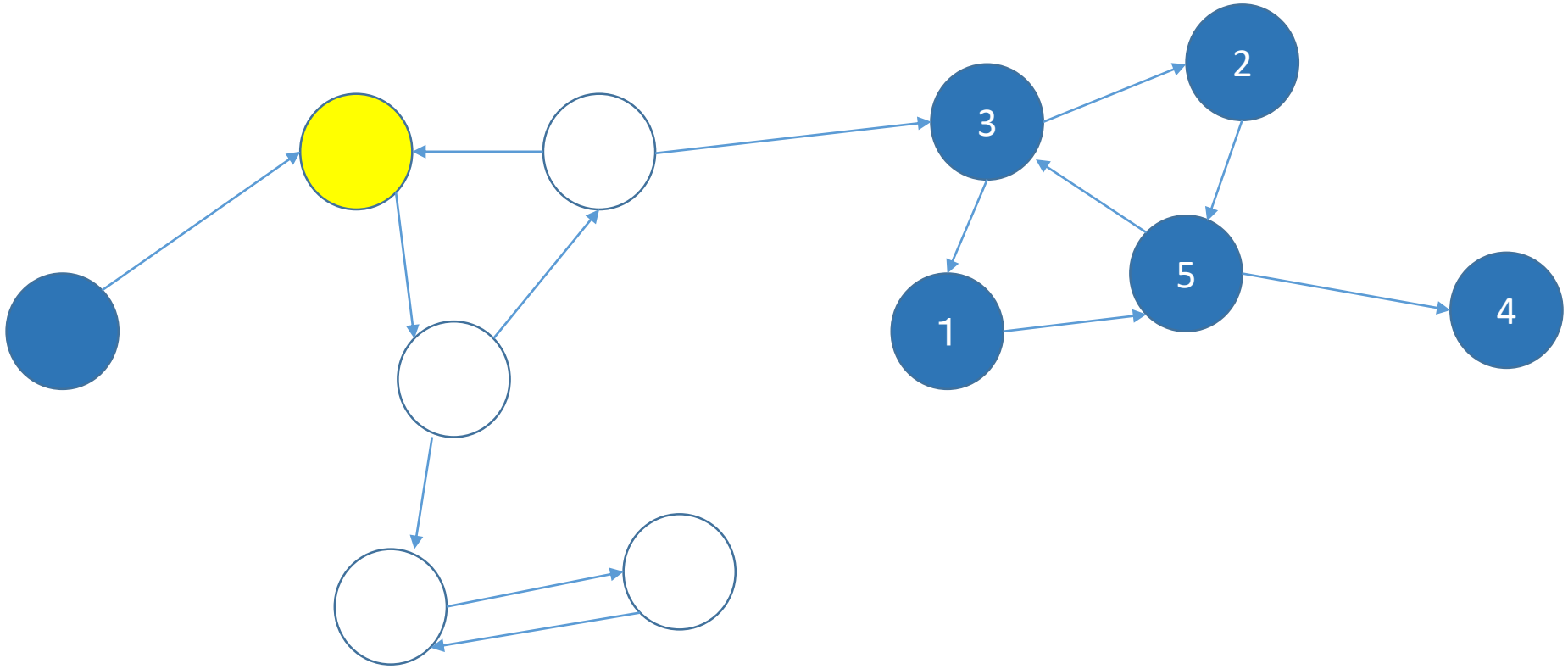


アルゴリズム(１回目のDFS)

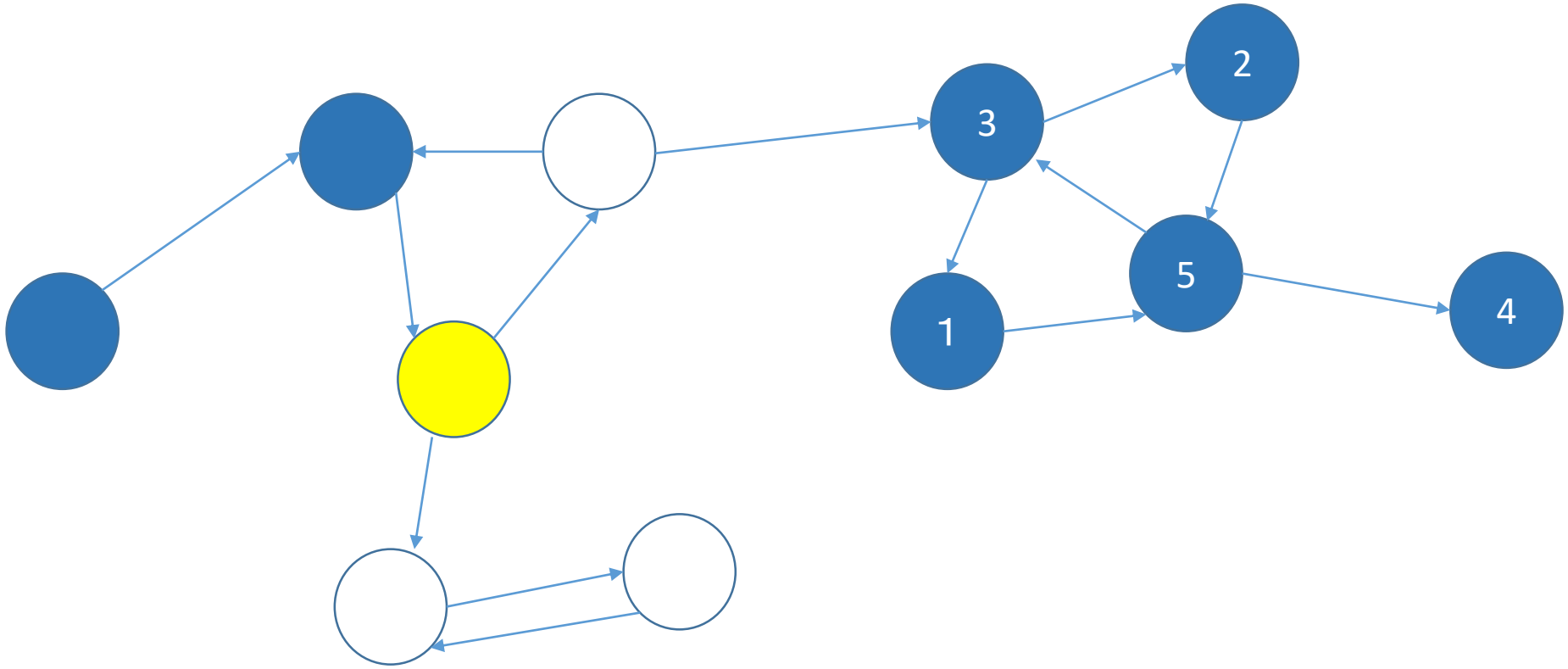
新しい頂点から再び



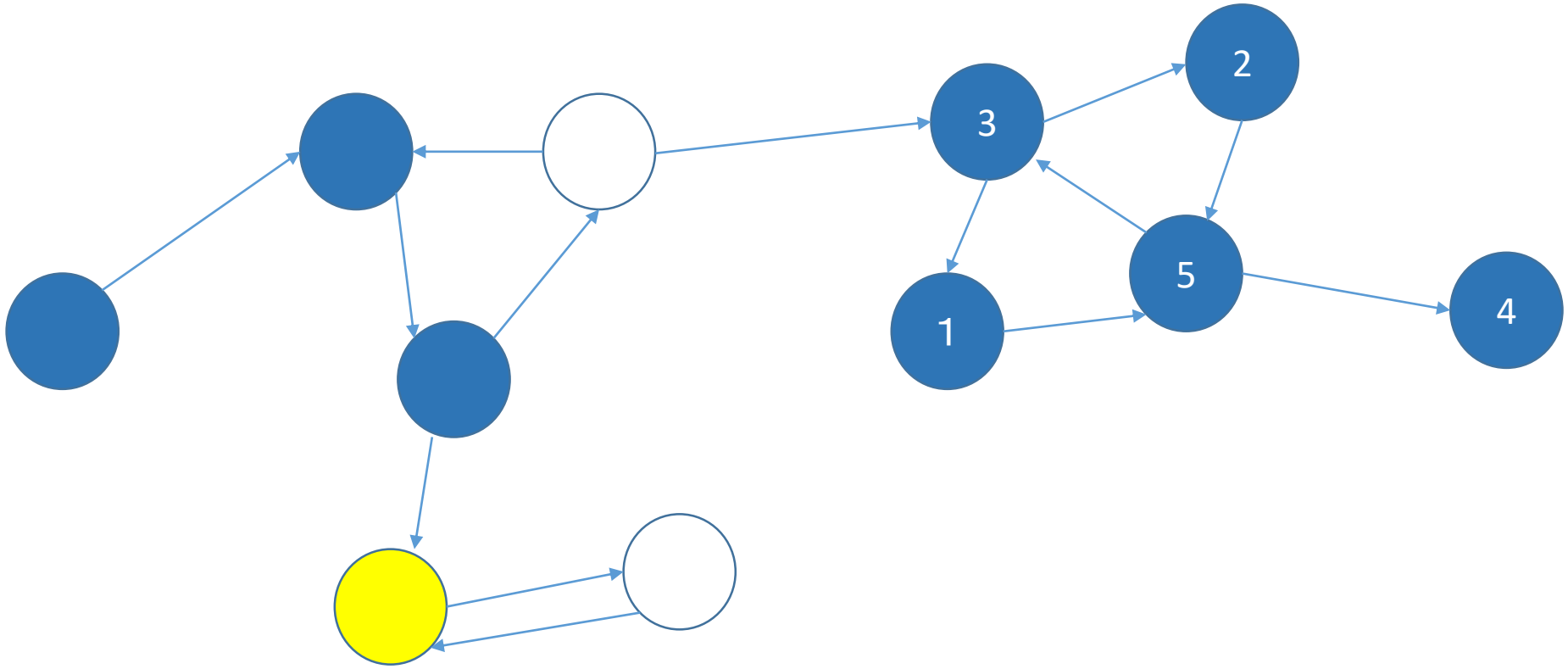
アルゴリズム(１回目のDFS)



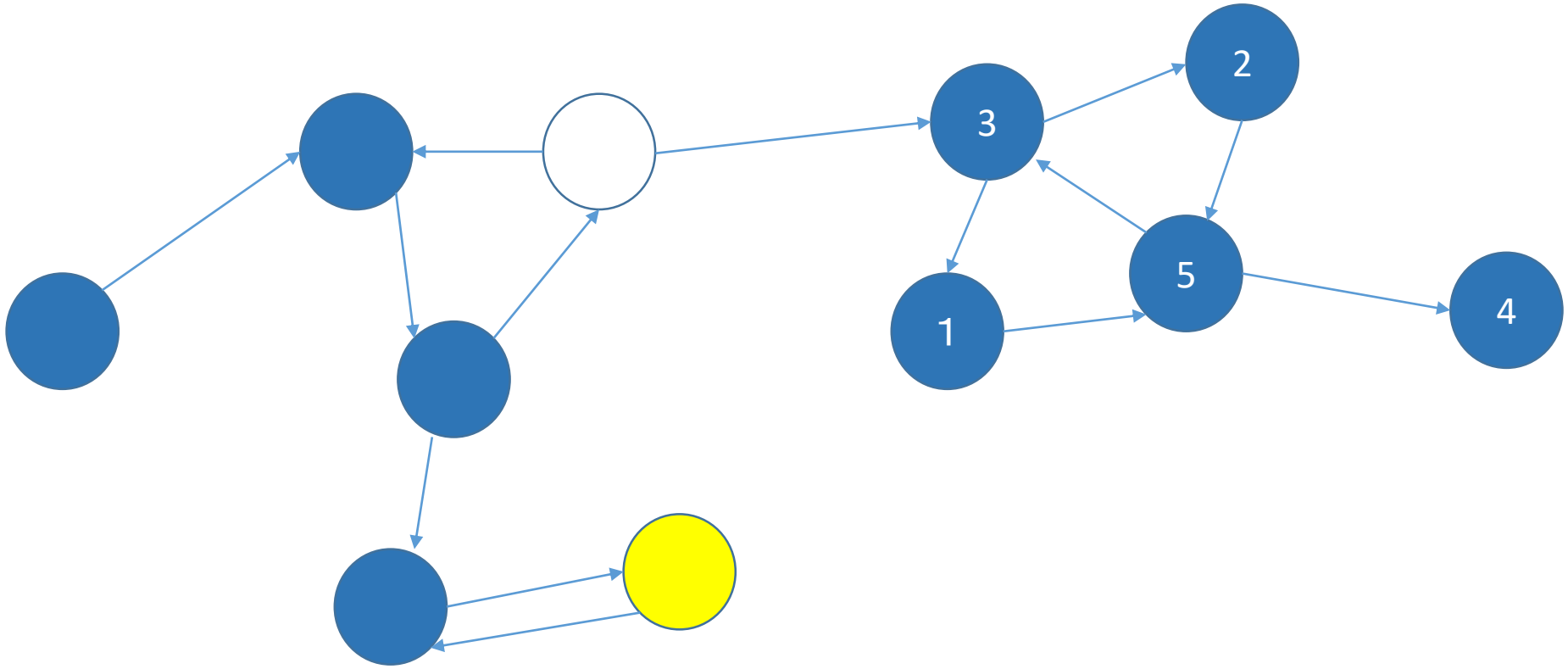
アルゴリズム(１回目のDFS)



アルゴリズム(１回目のDFS)

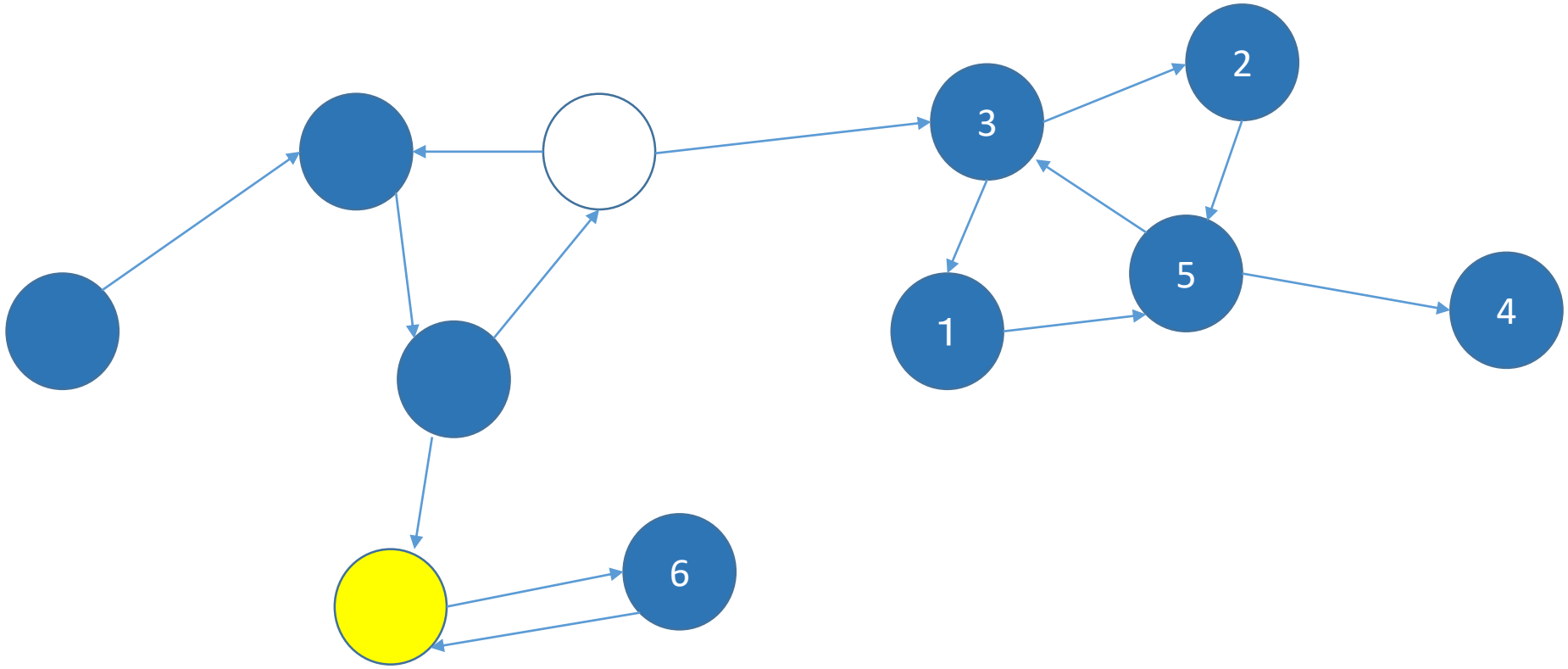


アルゴリズム(１回目のDFS)



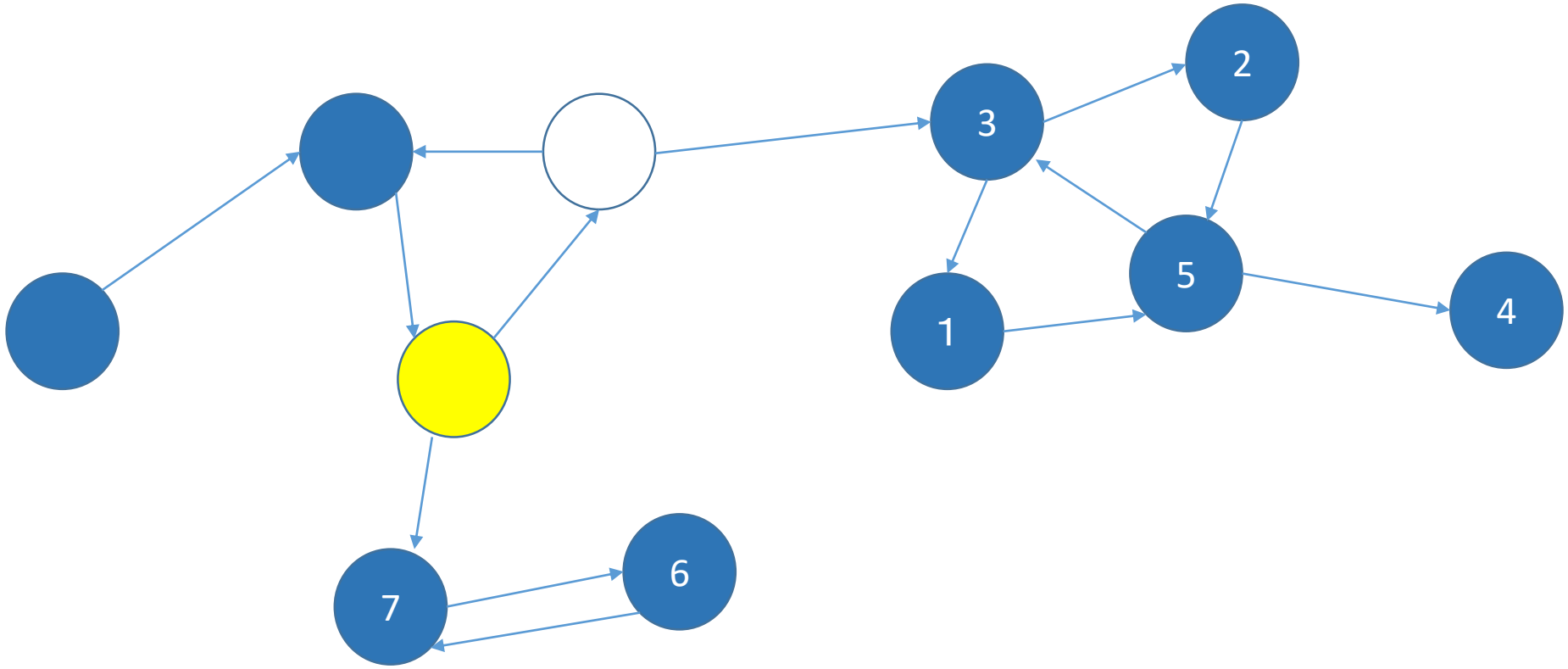
アルゴリズム(１回目のDFS)

帰りがけに番号6を振る

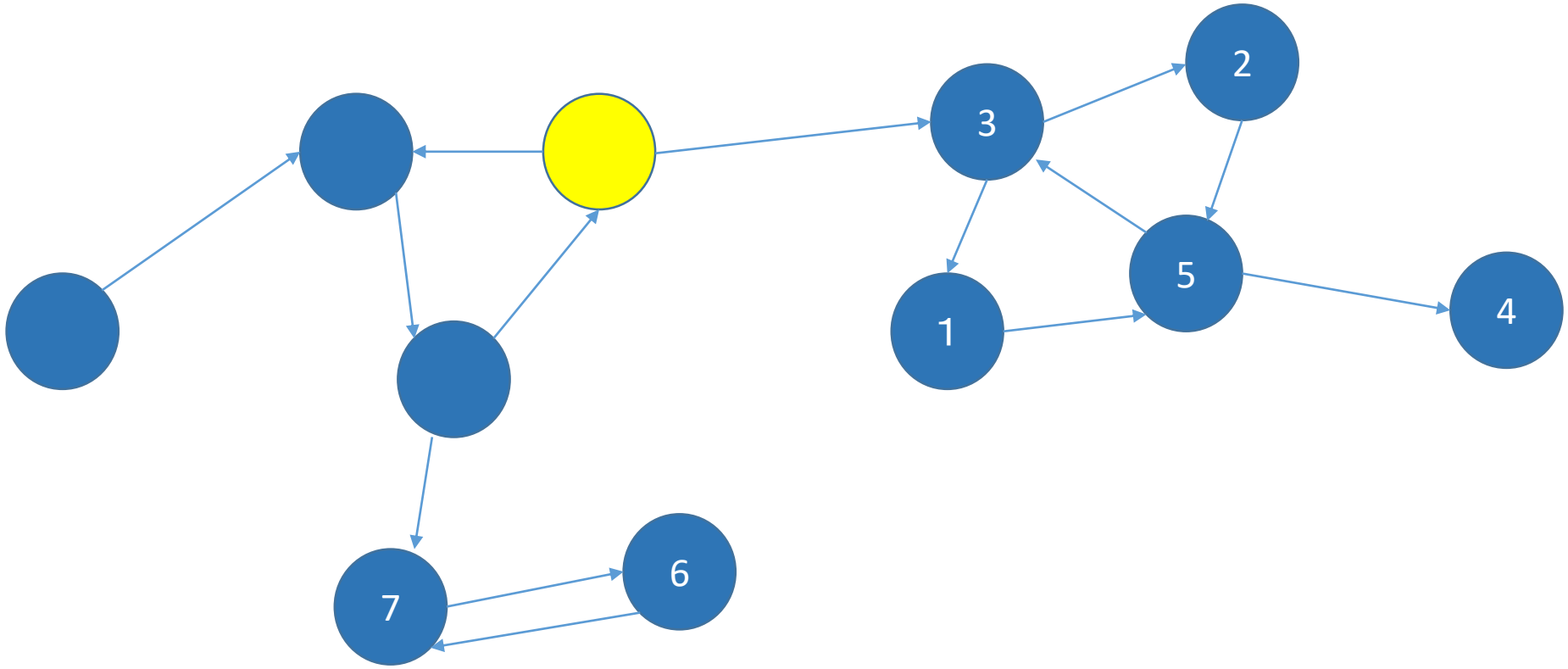


アルゴリズム(１回目のDFS)

帰りがけに番号7を振る

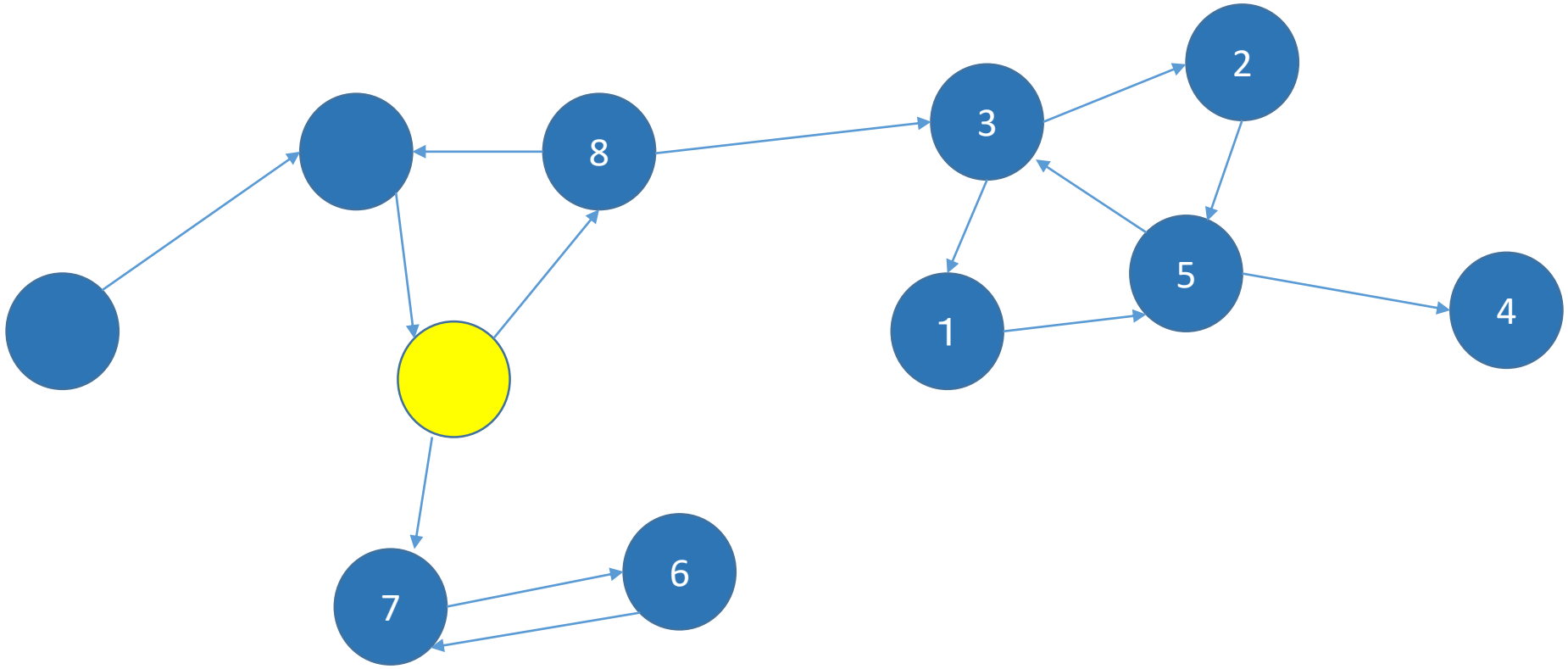


アルゴリズム(１回目のDFS)



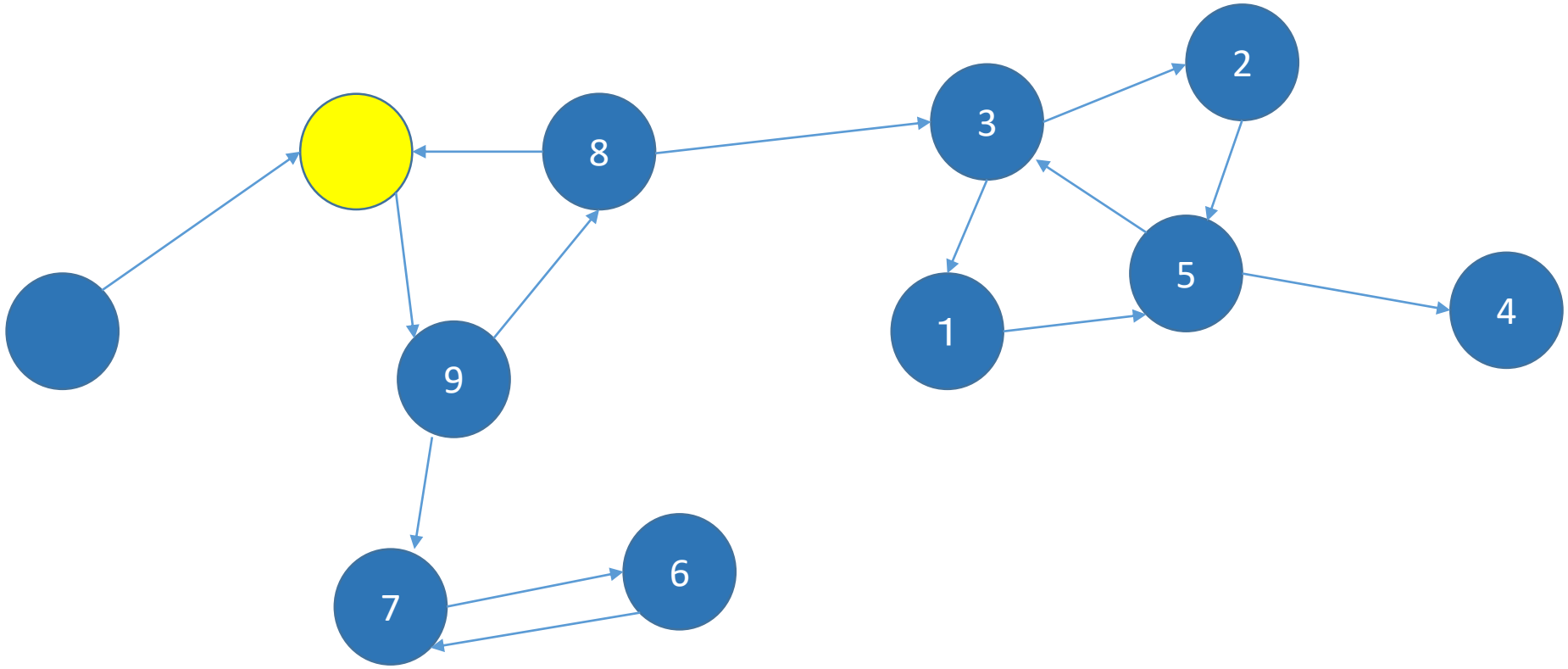
アルゴリズム(１回目のDFS)

帰りがけに番号8を振る



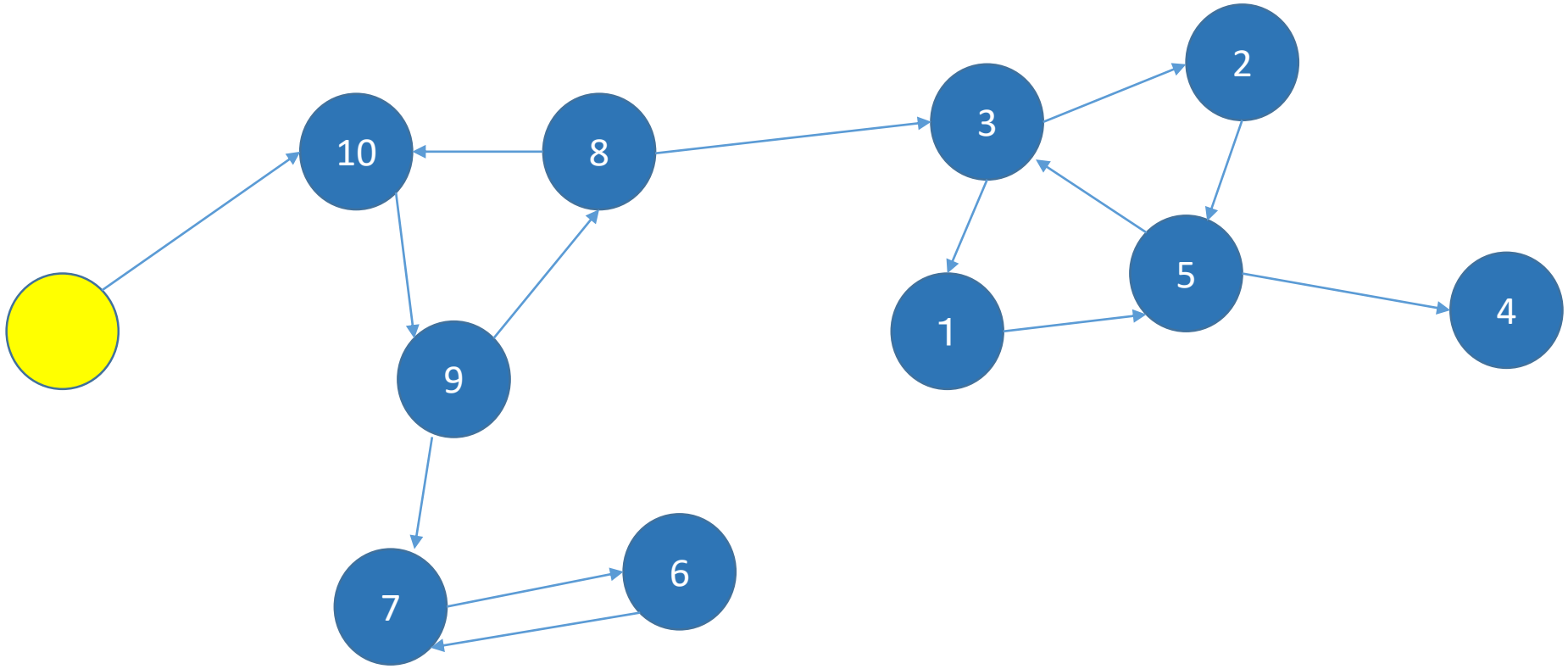
アルゴリズム(１回目のDFS)

帰りがけに番号9を振る



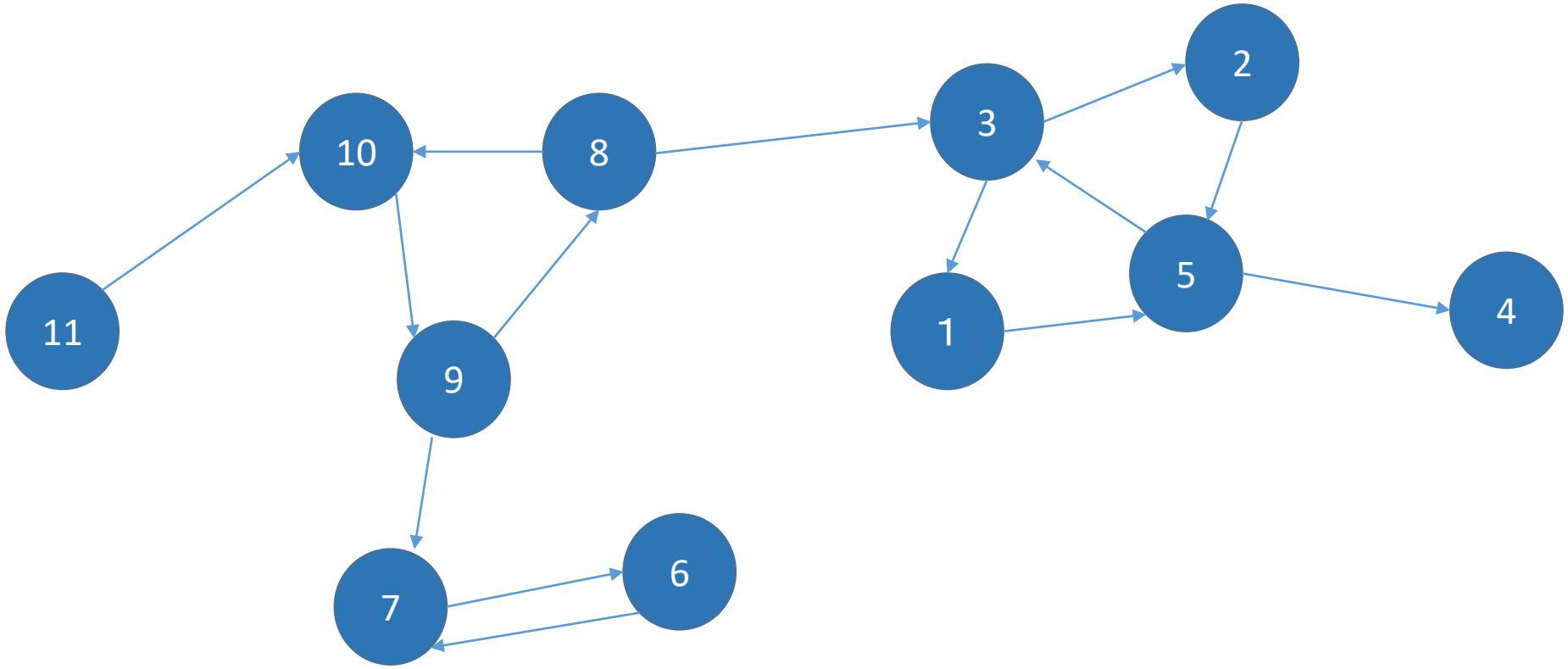
アルゴリズム(１回目のDFS)

帰りがけに番号10を振る



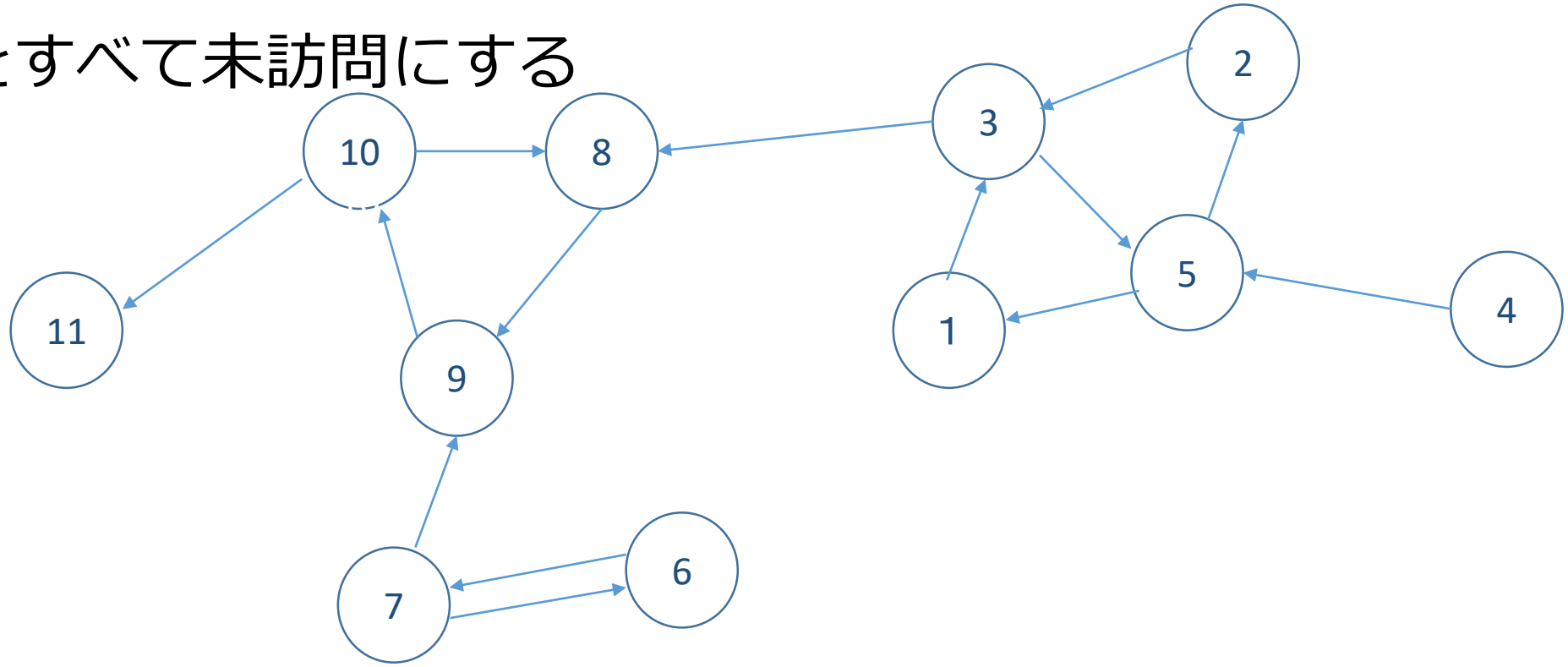
アルゴリズム(１回目のDFS)

帰りがけに番号11を振る



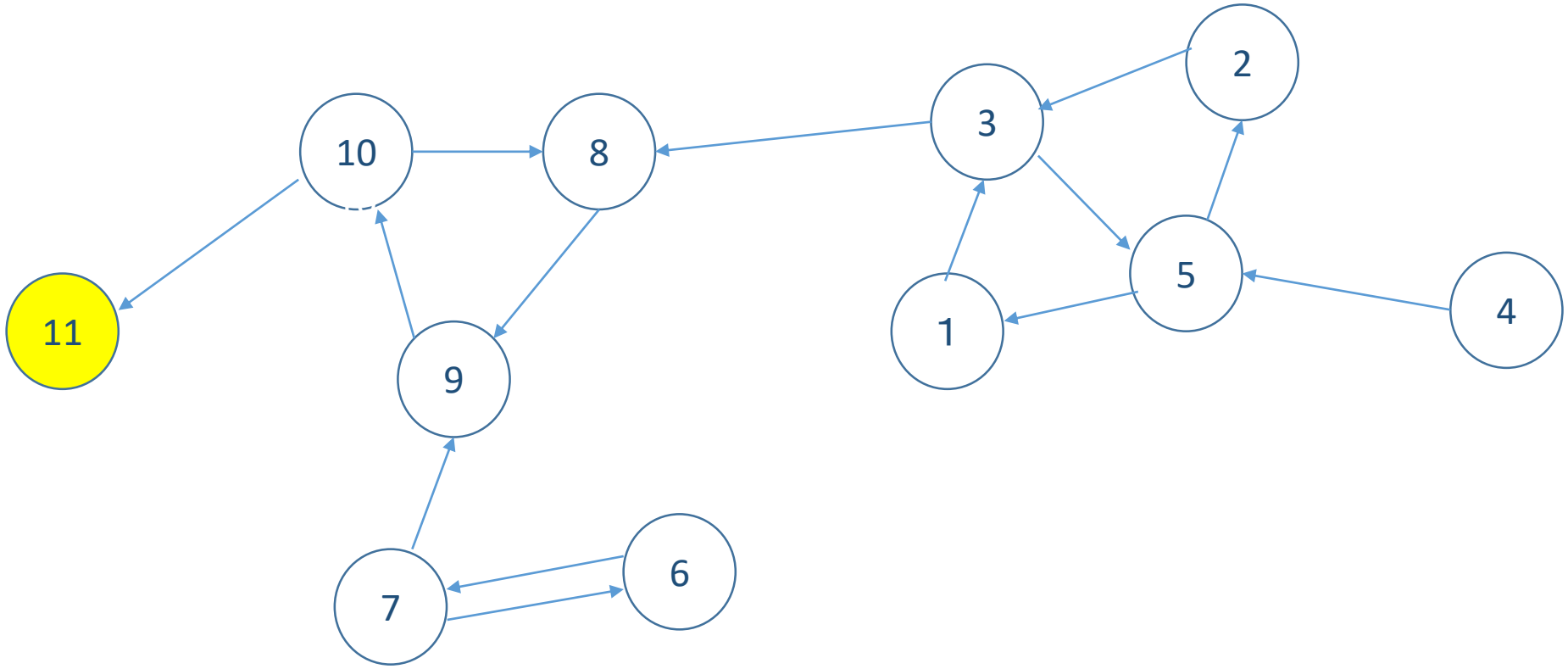
アルゴリズム

辺をすべて逆にする
頂点をすべて未訪問にする



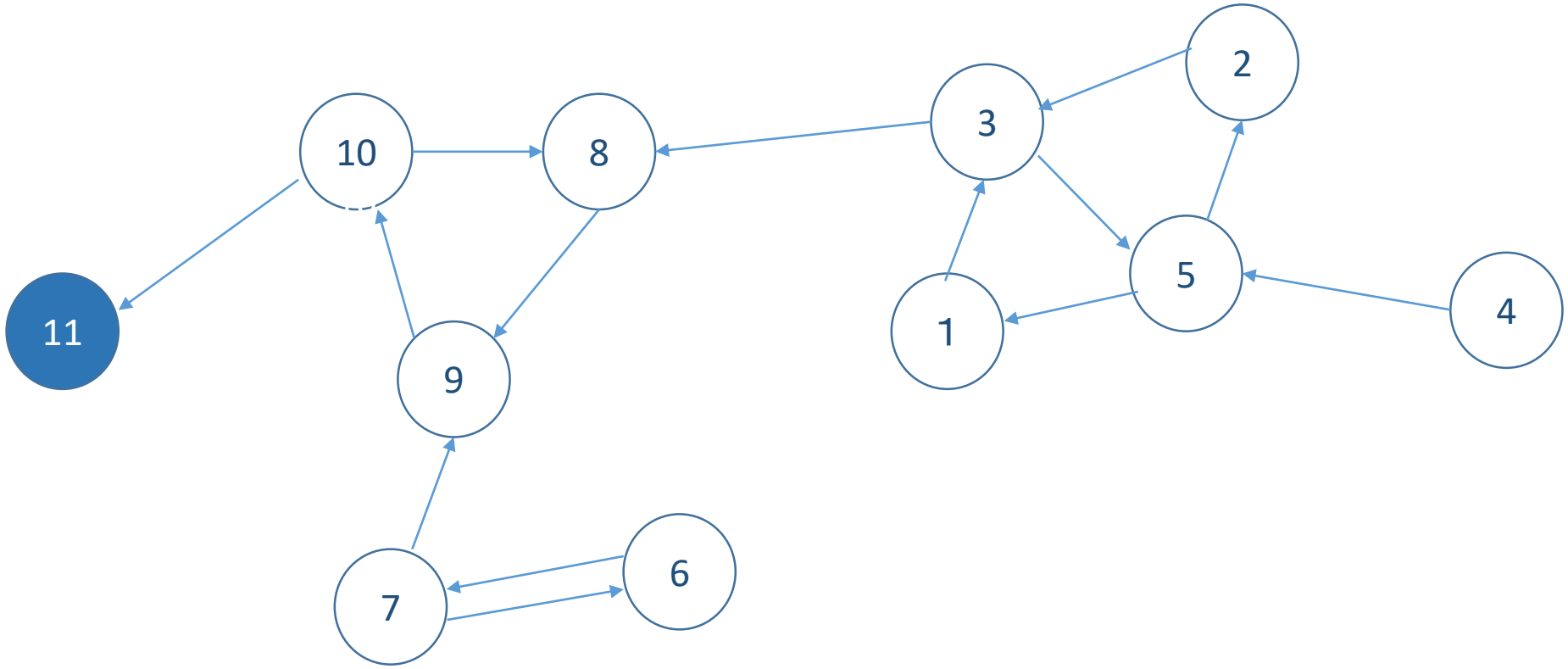
アルゴリズム(2回目のDFS)

一番番号が大きいところからDFS



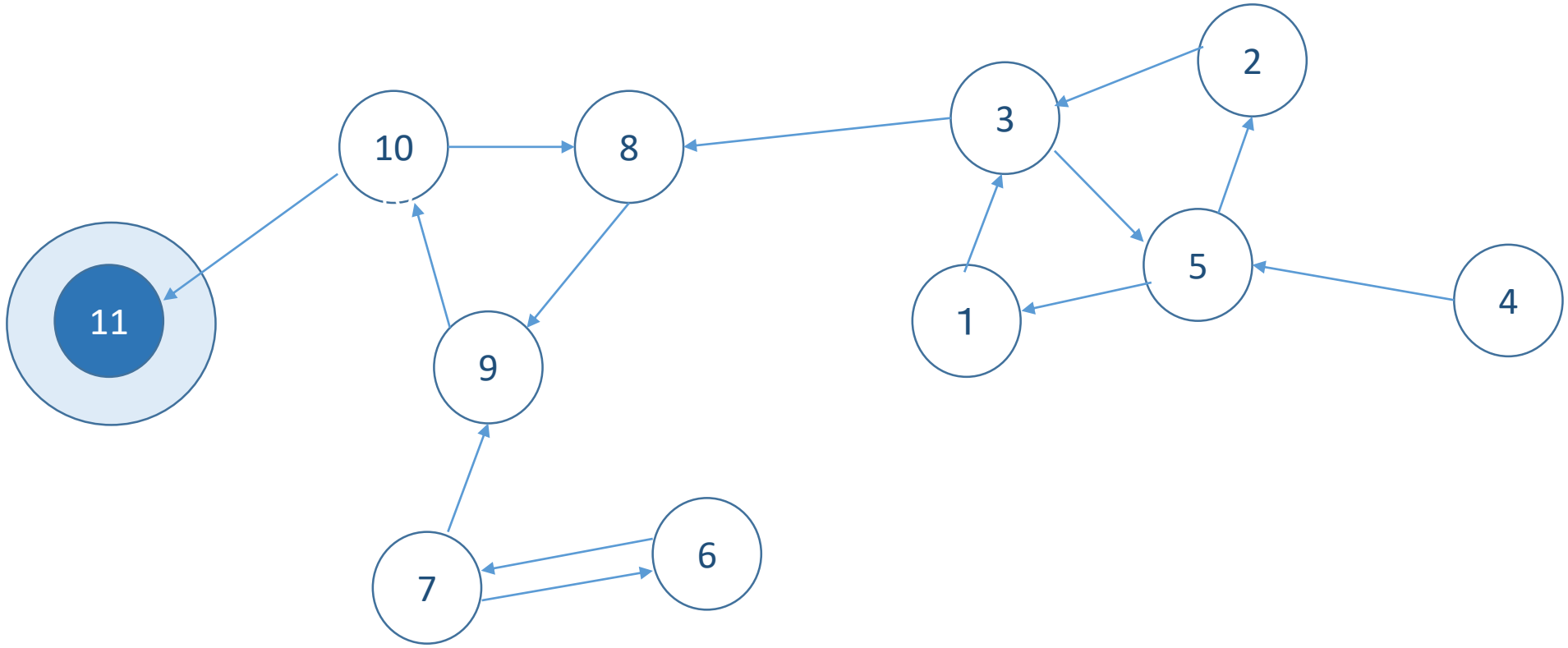
アルゴリズム(2回目のDFS)

DFS終わり



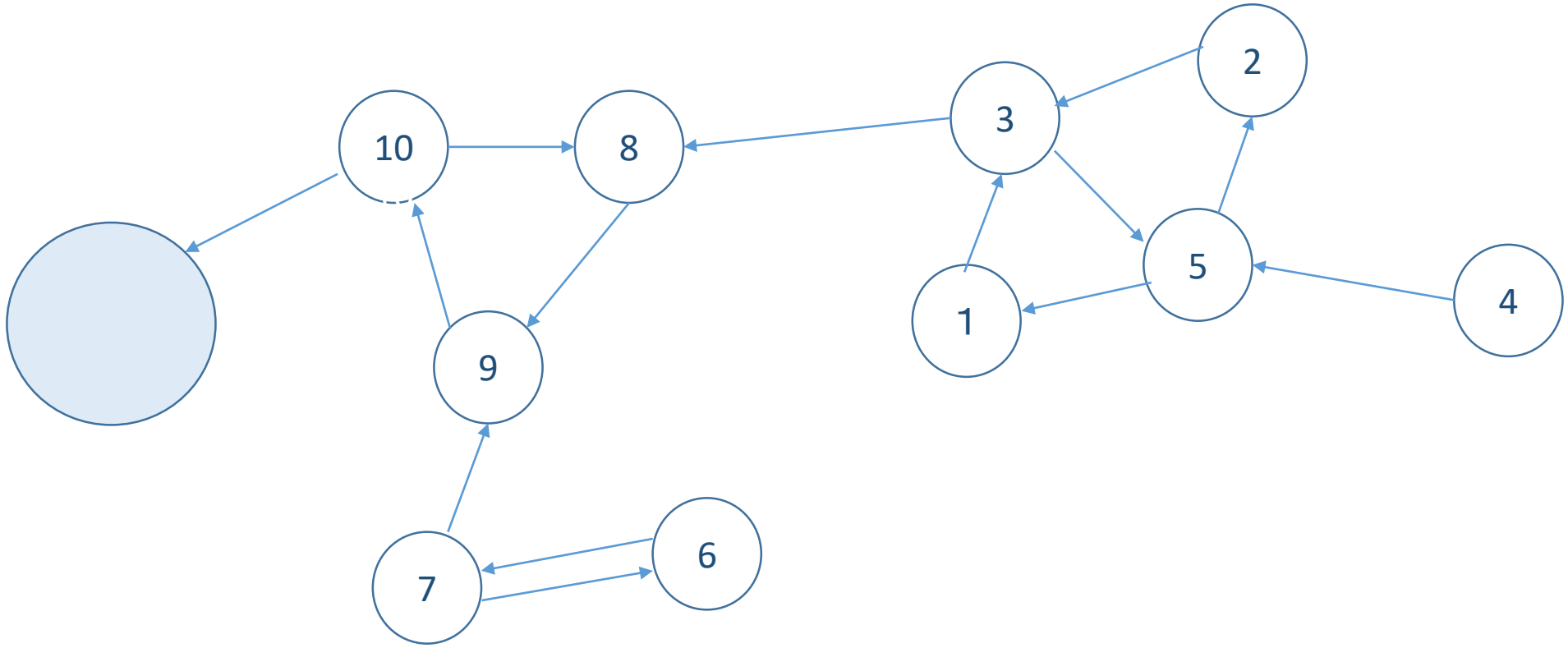
アルゴリズム(2回目のDFS)

通ることのできた頂点を



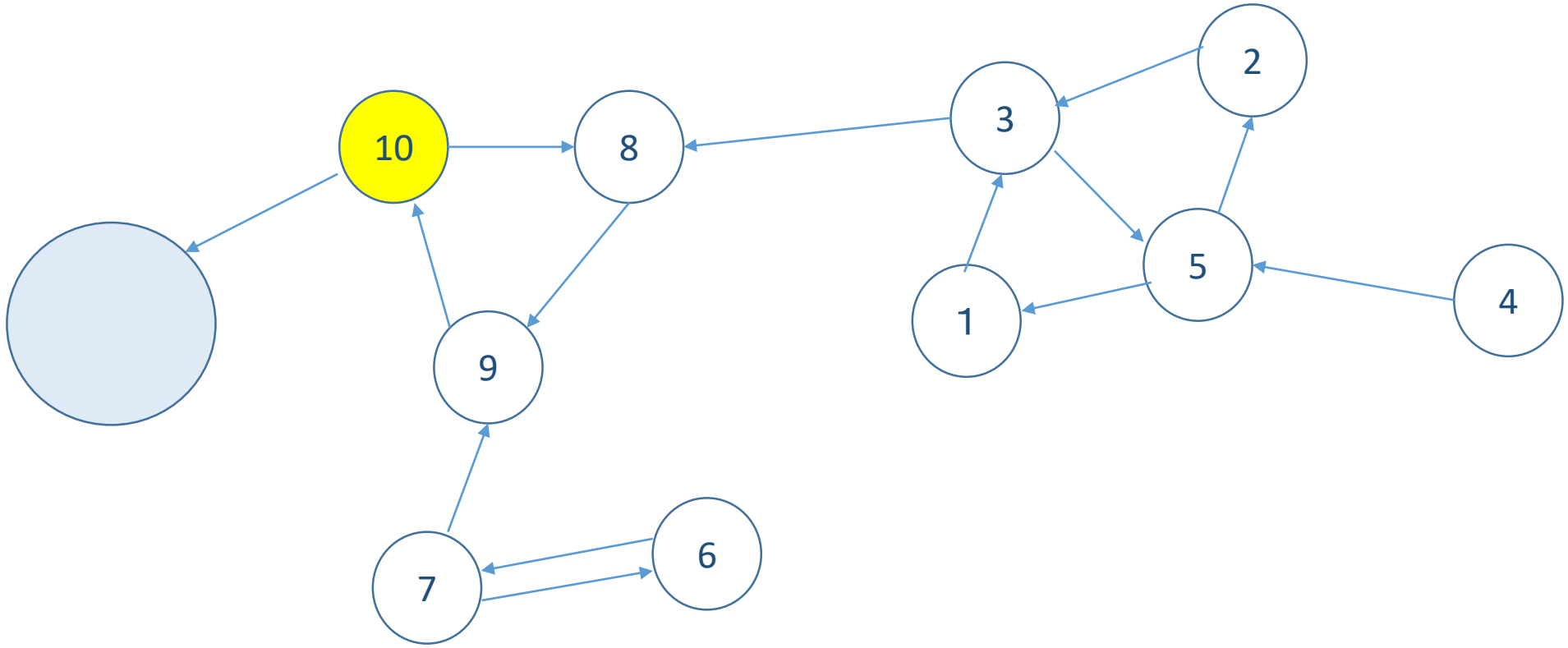
アルゴリズム(2回目のDFS)

通ることのできた頂点をまとめる

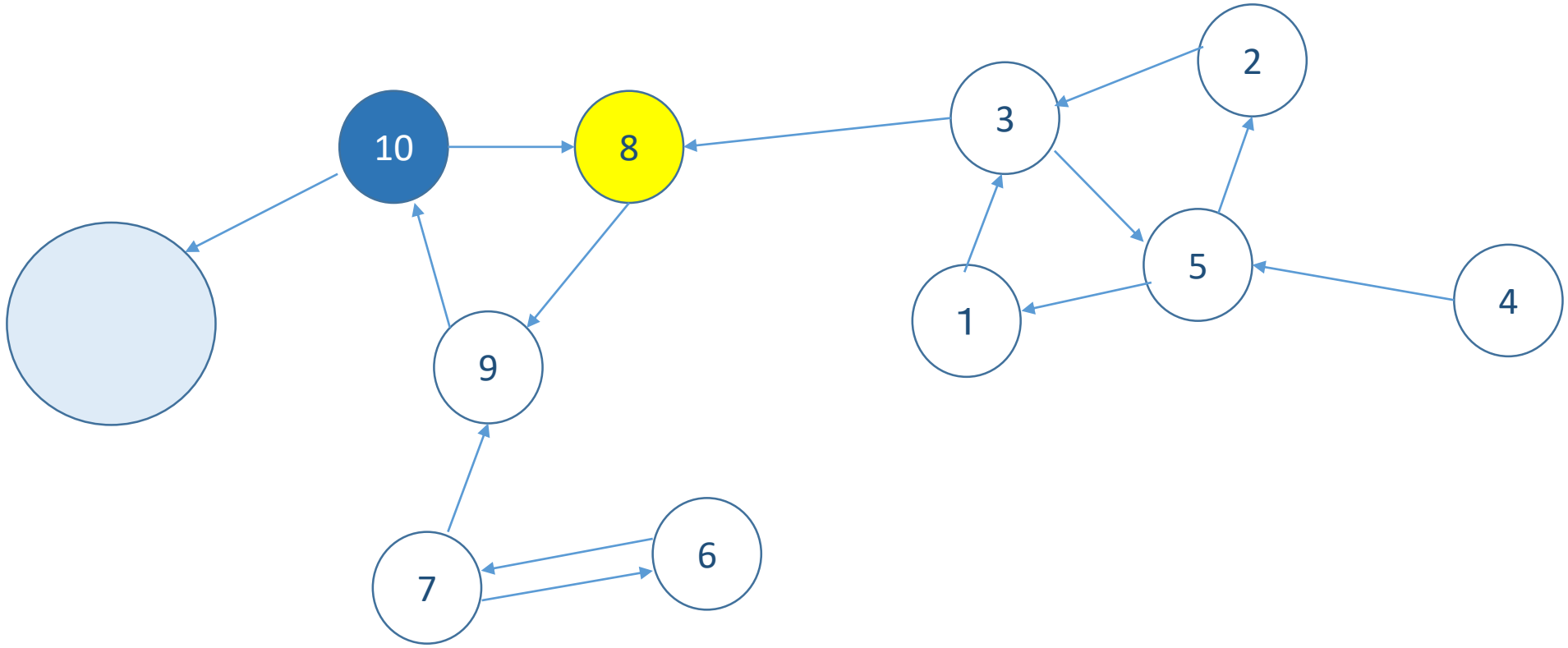


アルゴリズム(2回目のDFS)

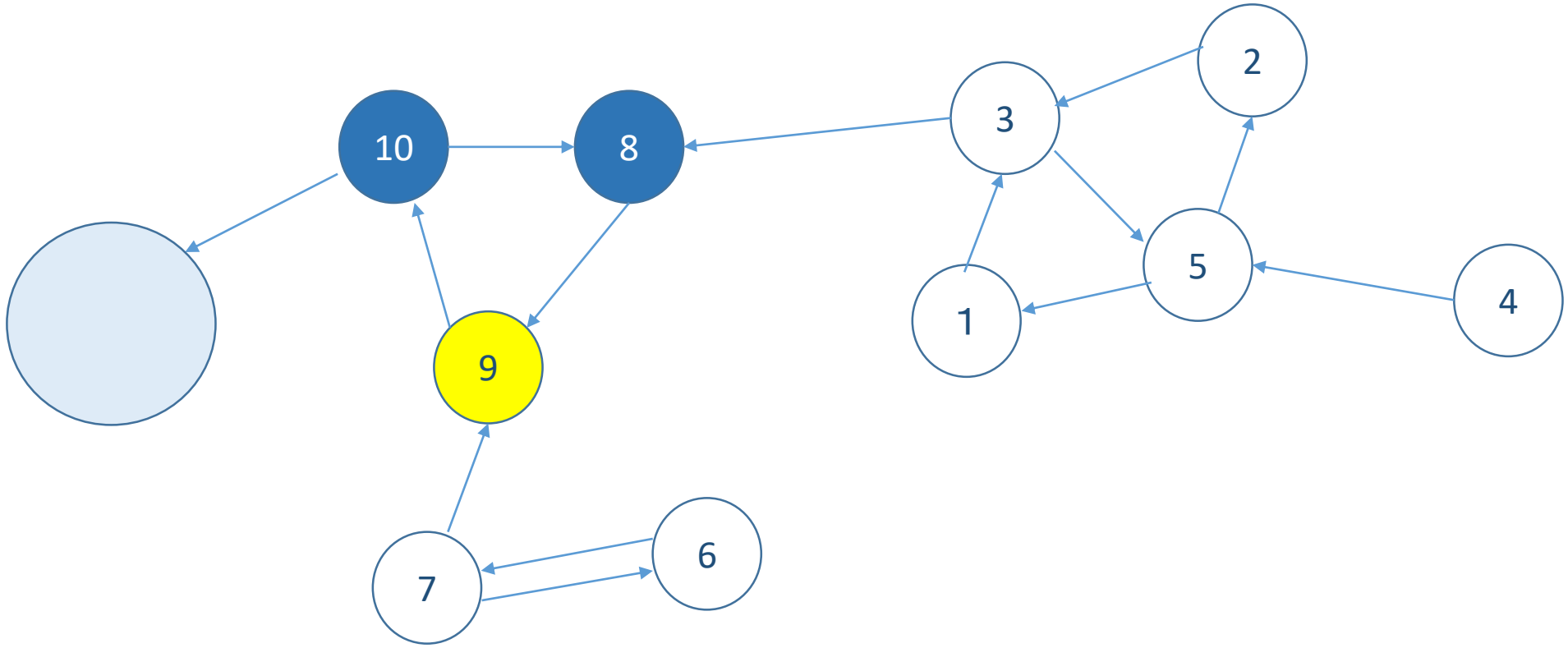
一番大きいところからDFS



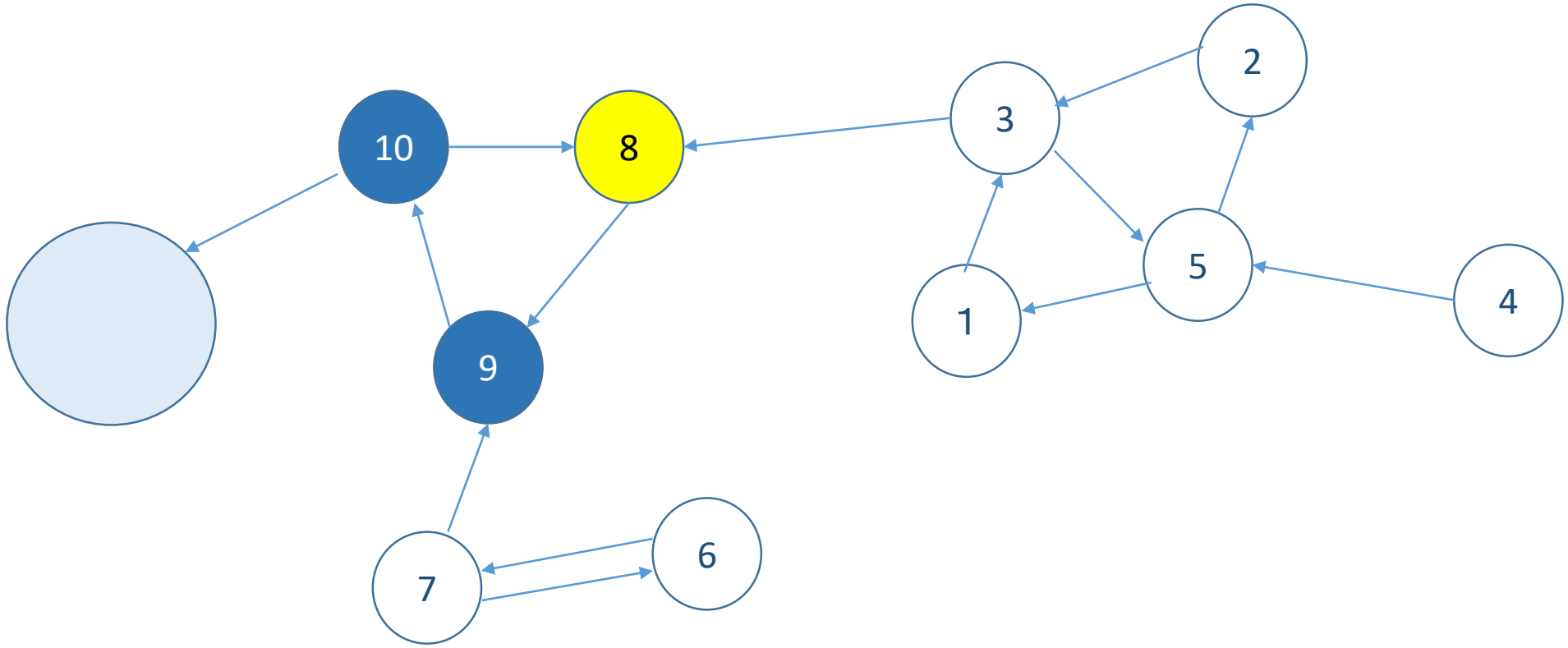
アルゴリズム(2回目のDFS)



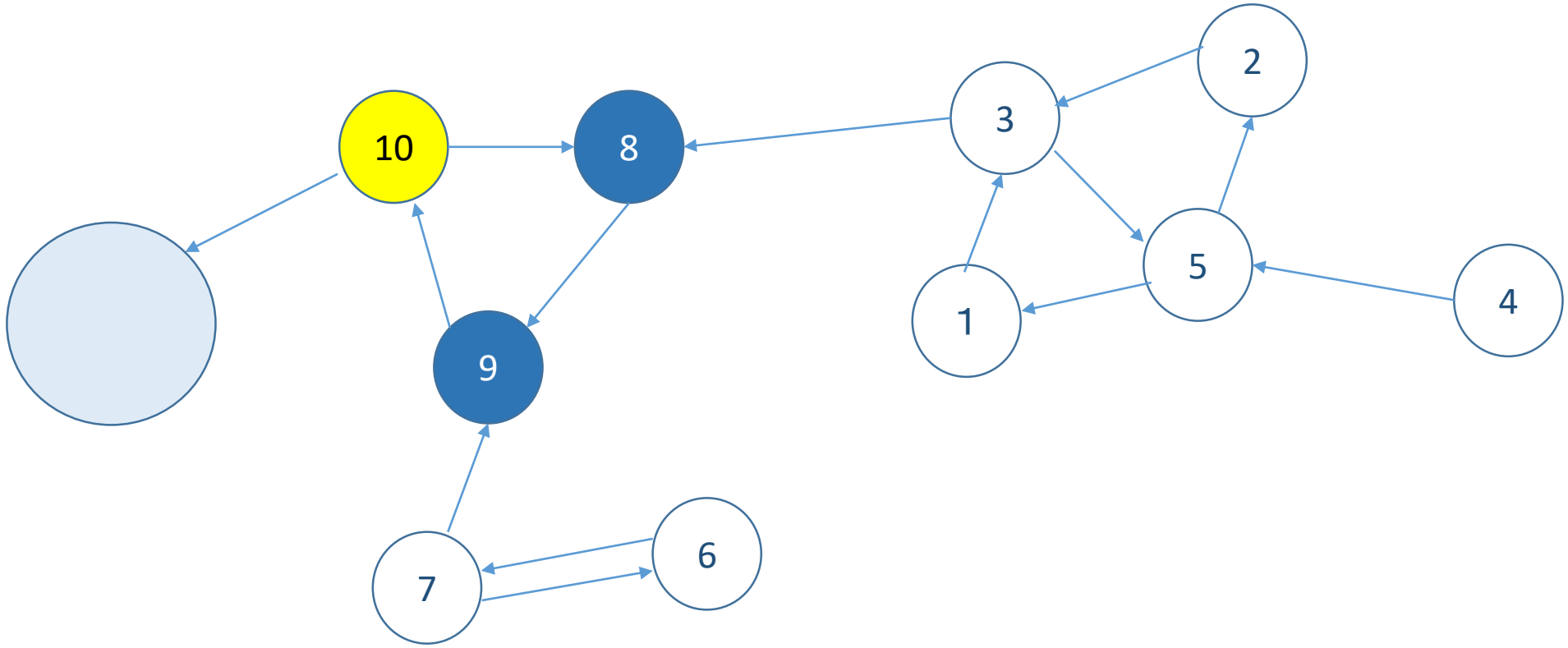
アルゴリズム(2回目のDFS)



アルゴリズム(2回目のDFS)

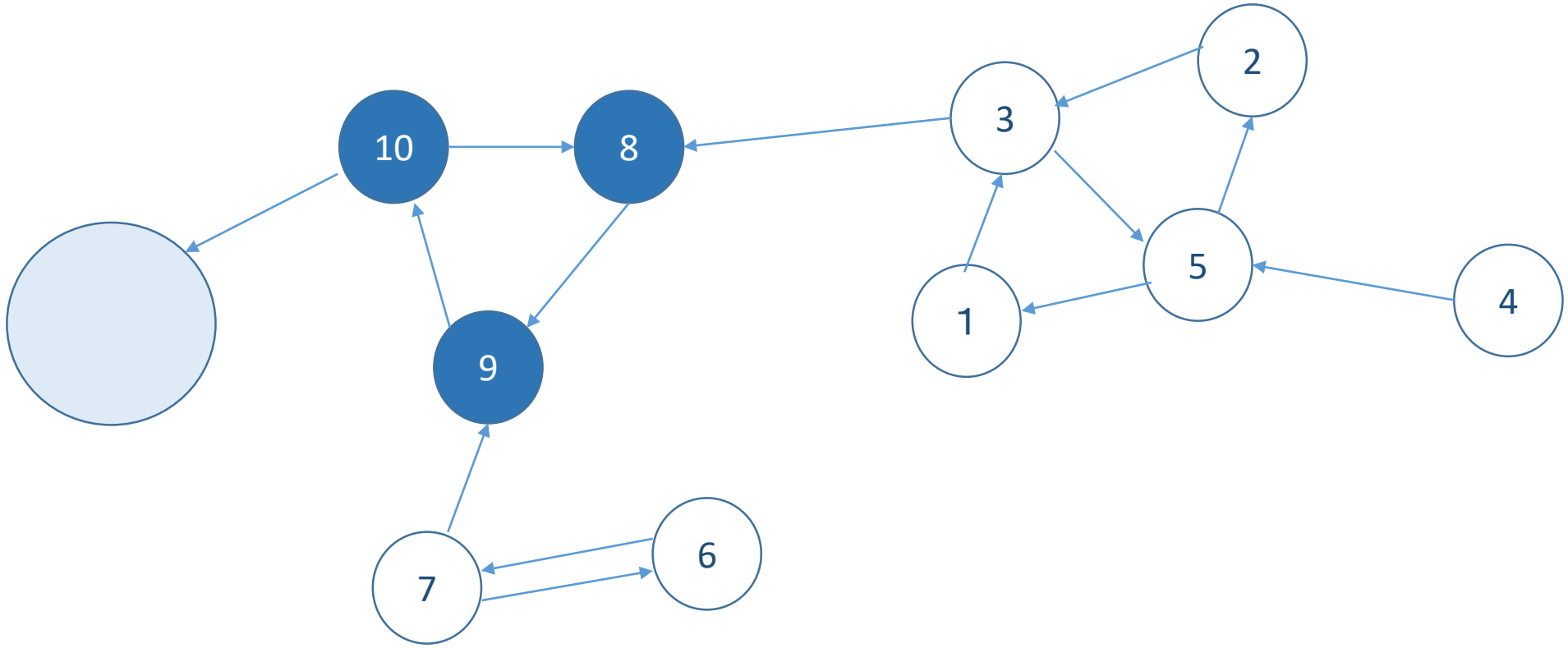


アルゴリズム(2回目のDFS)



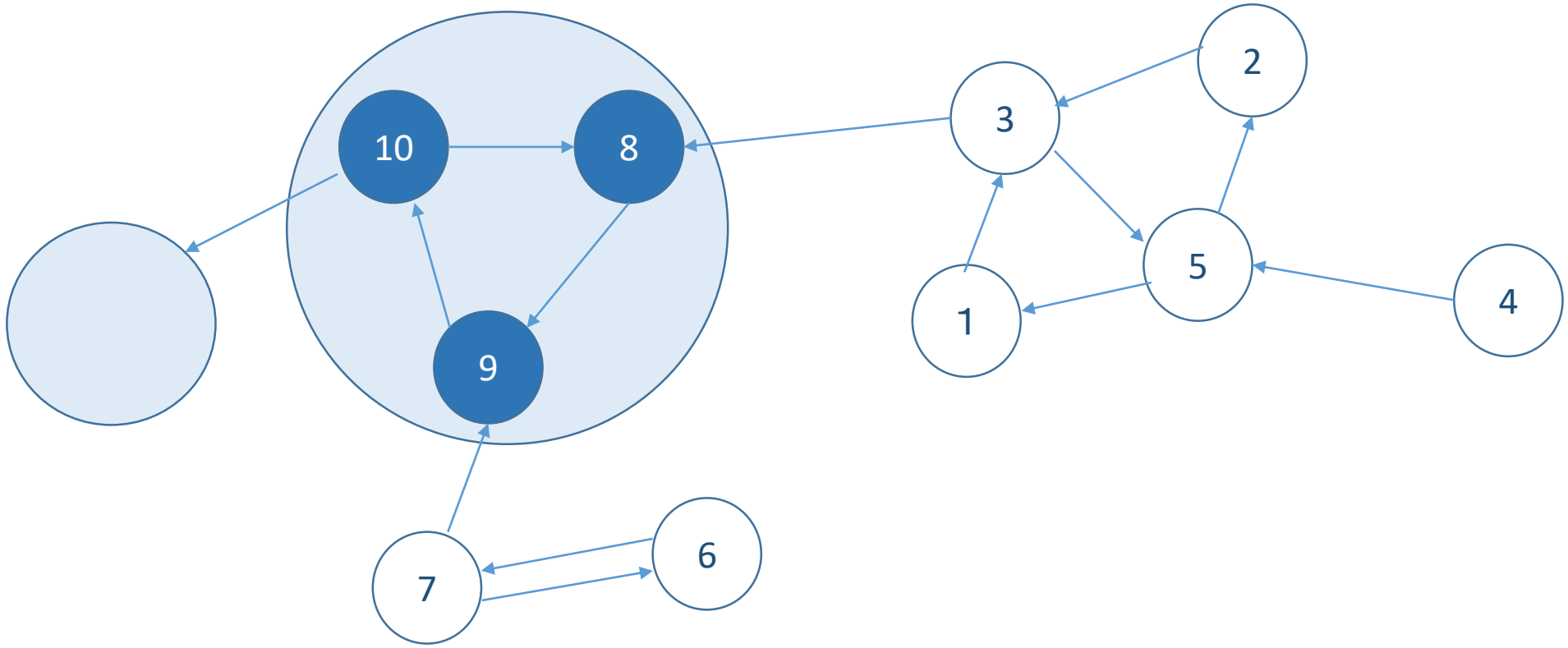
アルゴリズム(2回目のDFS)

DFS終わり



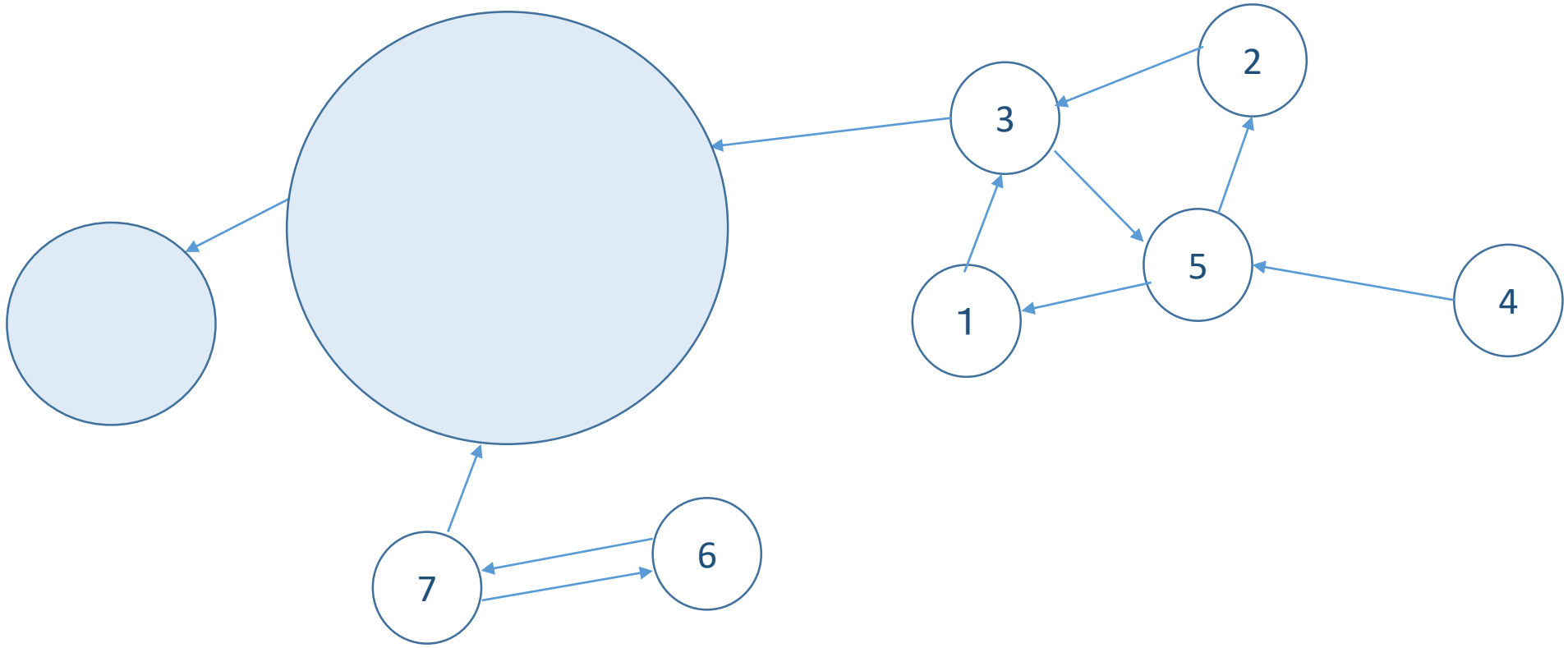
アルゴリズム(2回目のDFS)

通ることのできた頂点を



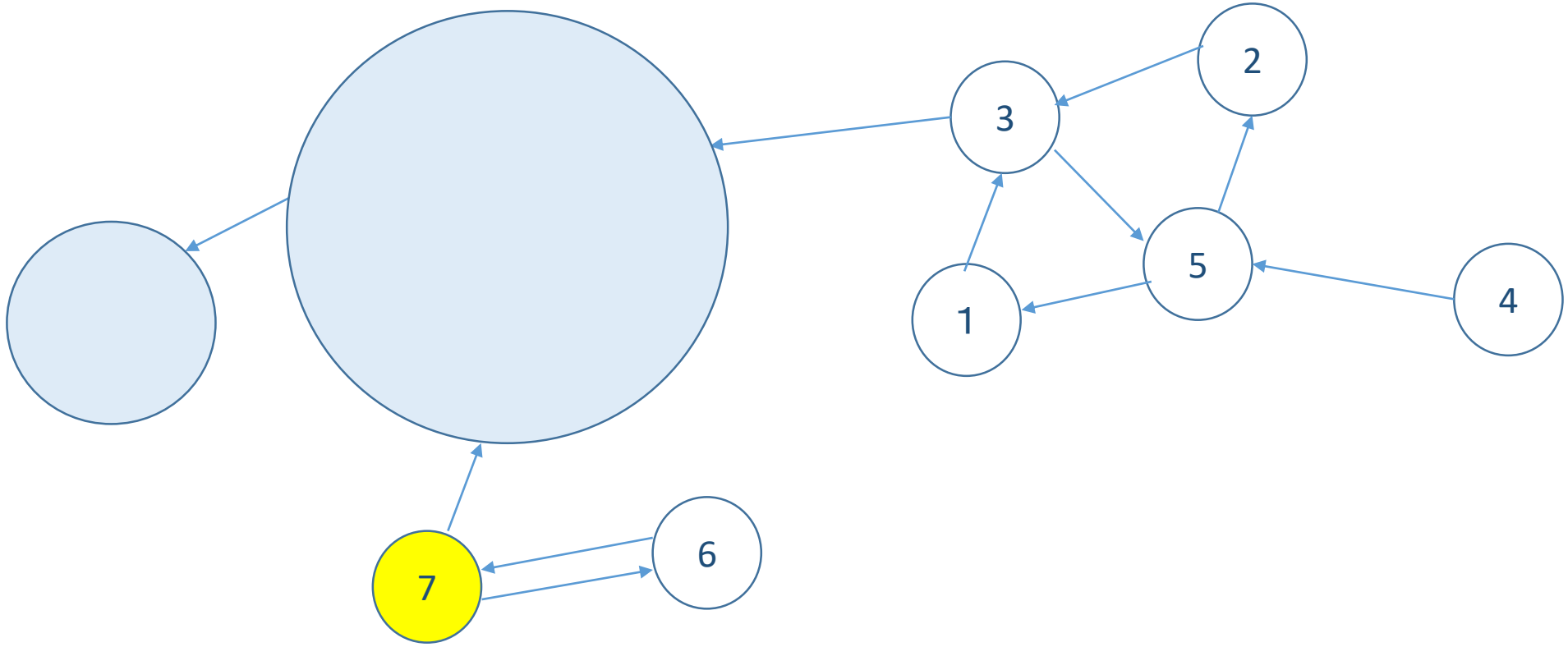
アルゴリズム(2回目のDFS)

通ることのできた頂点をまとめる

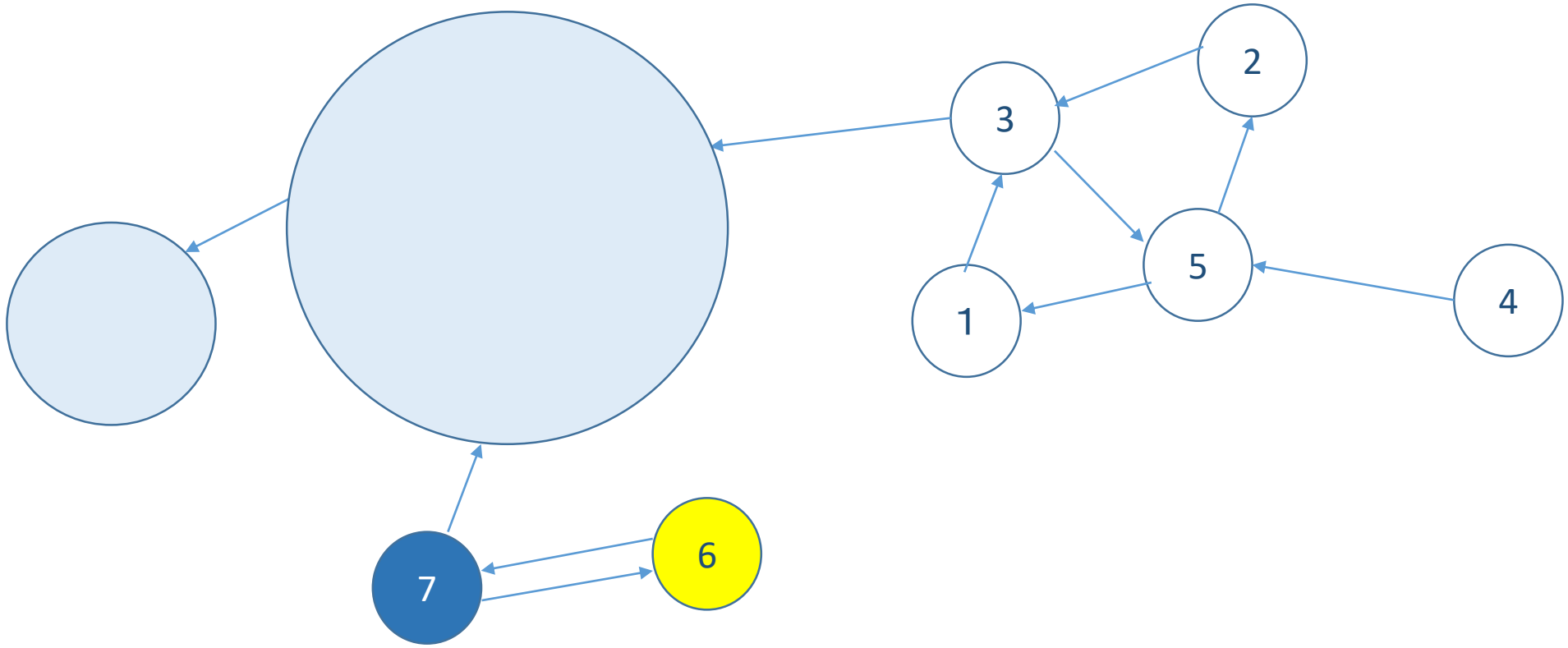


アルゴリズム(2回目のDFS)

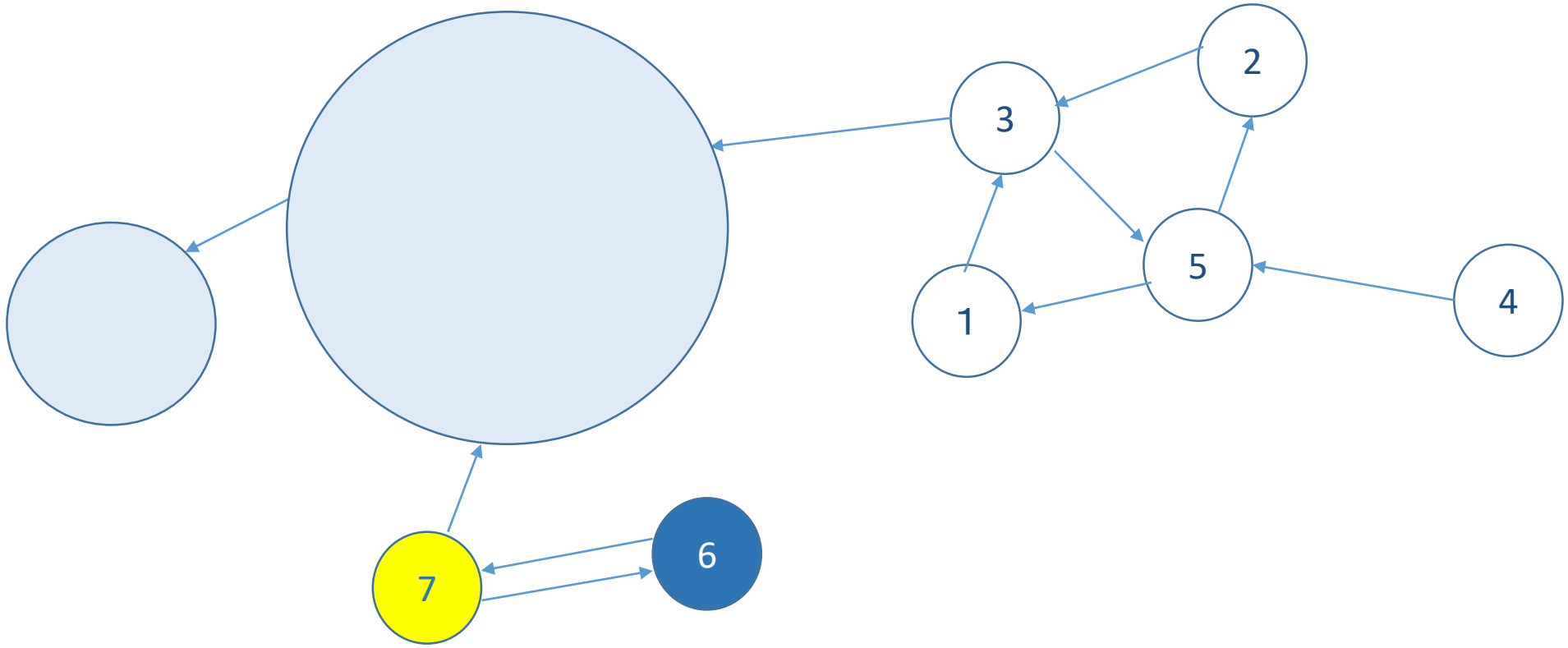
一番大きいところからDFS



アルゴリズム(2回目のDFS)

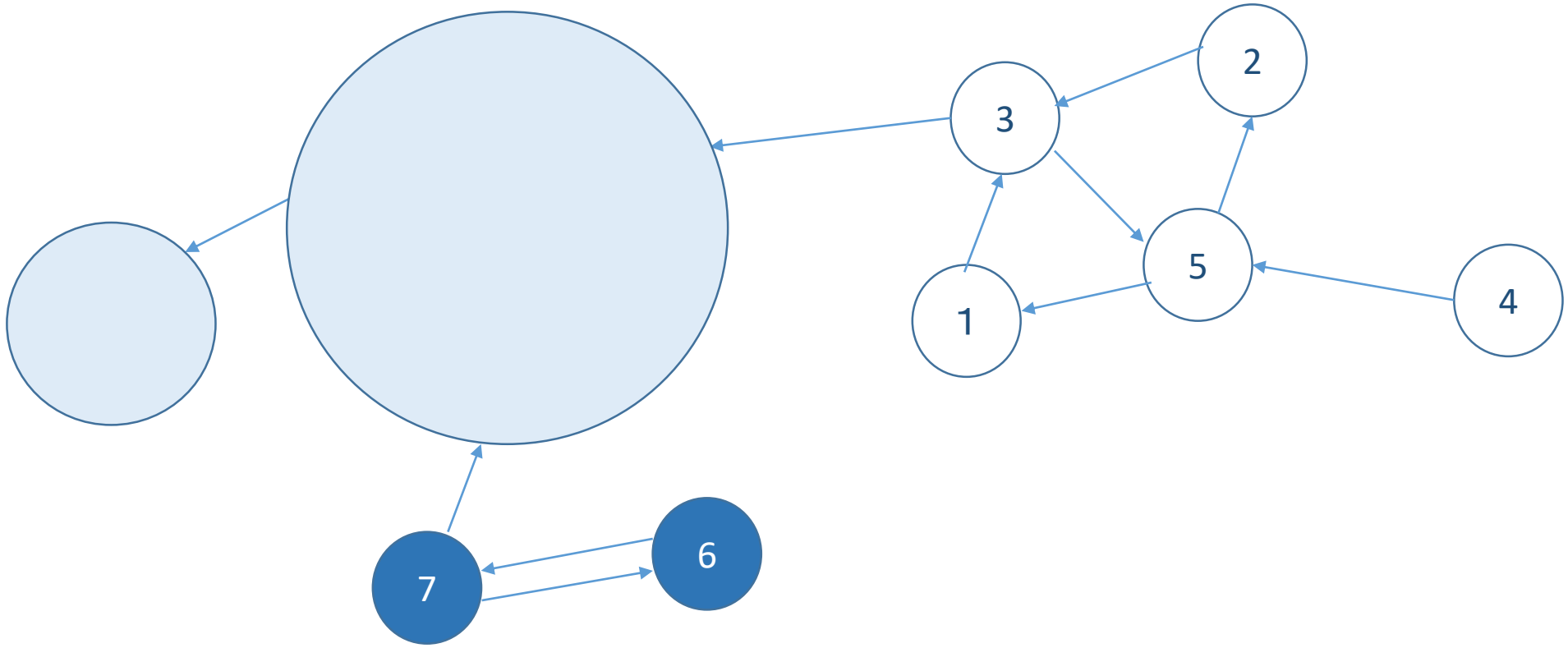


アルゴリズム(2回目のDFS)



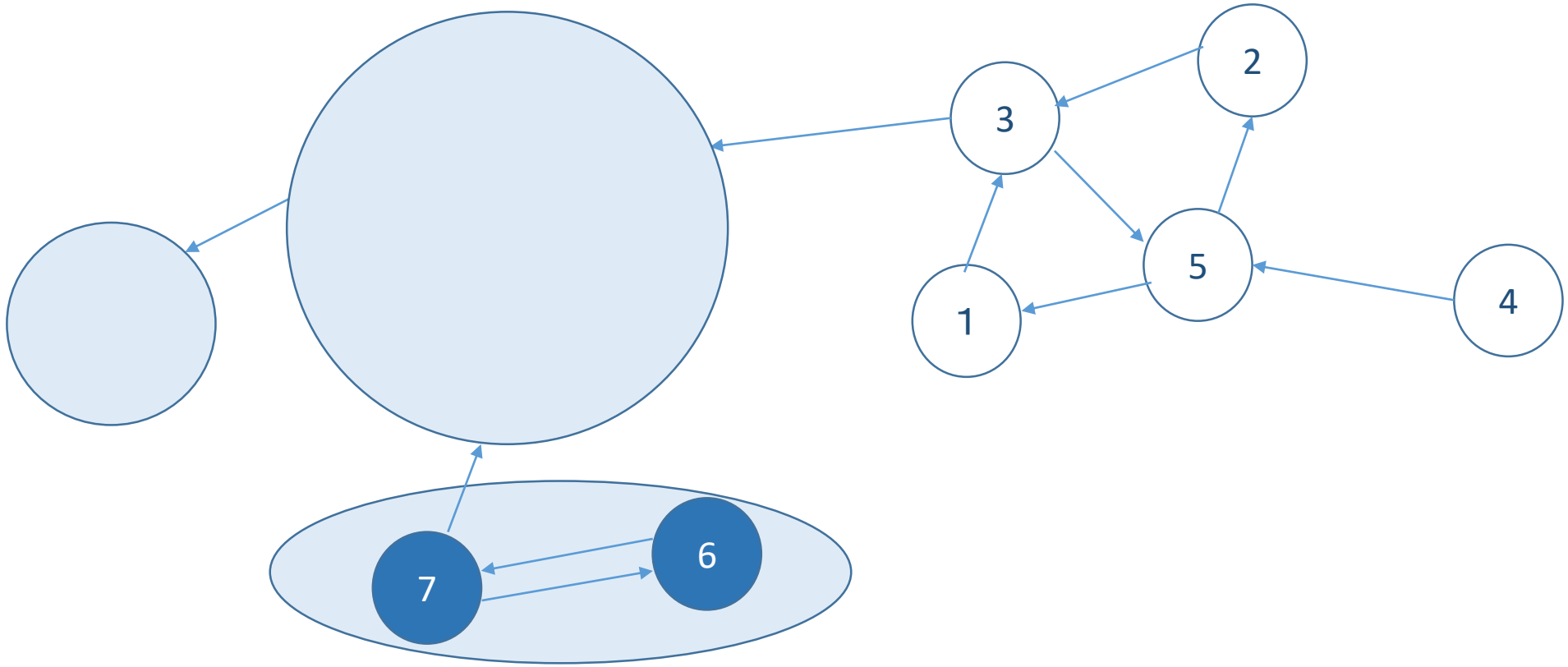
アルゴリズム(2回目のDFS)

DFS終わり



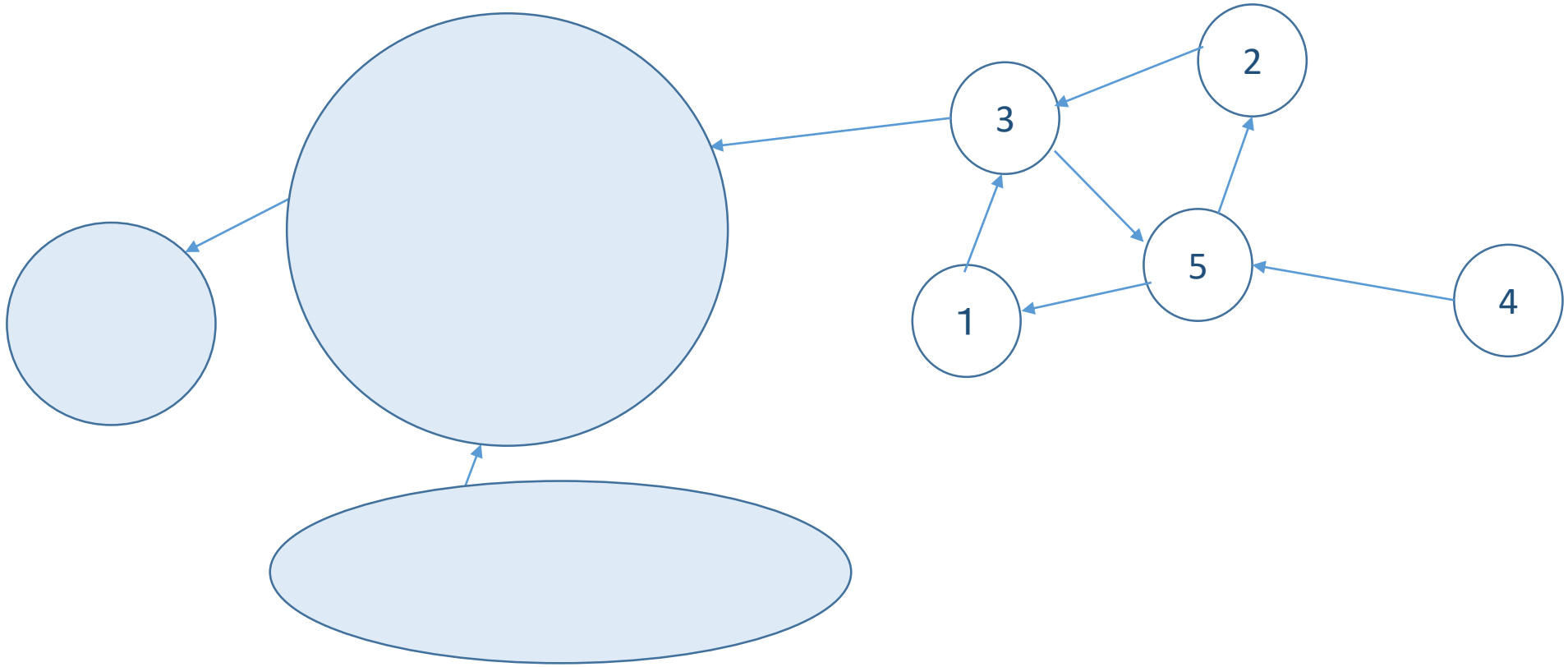
アルゴリズム(2回目のDFS)

通ることのできた頂点を



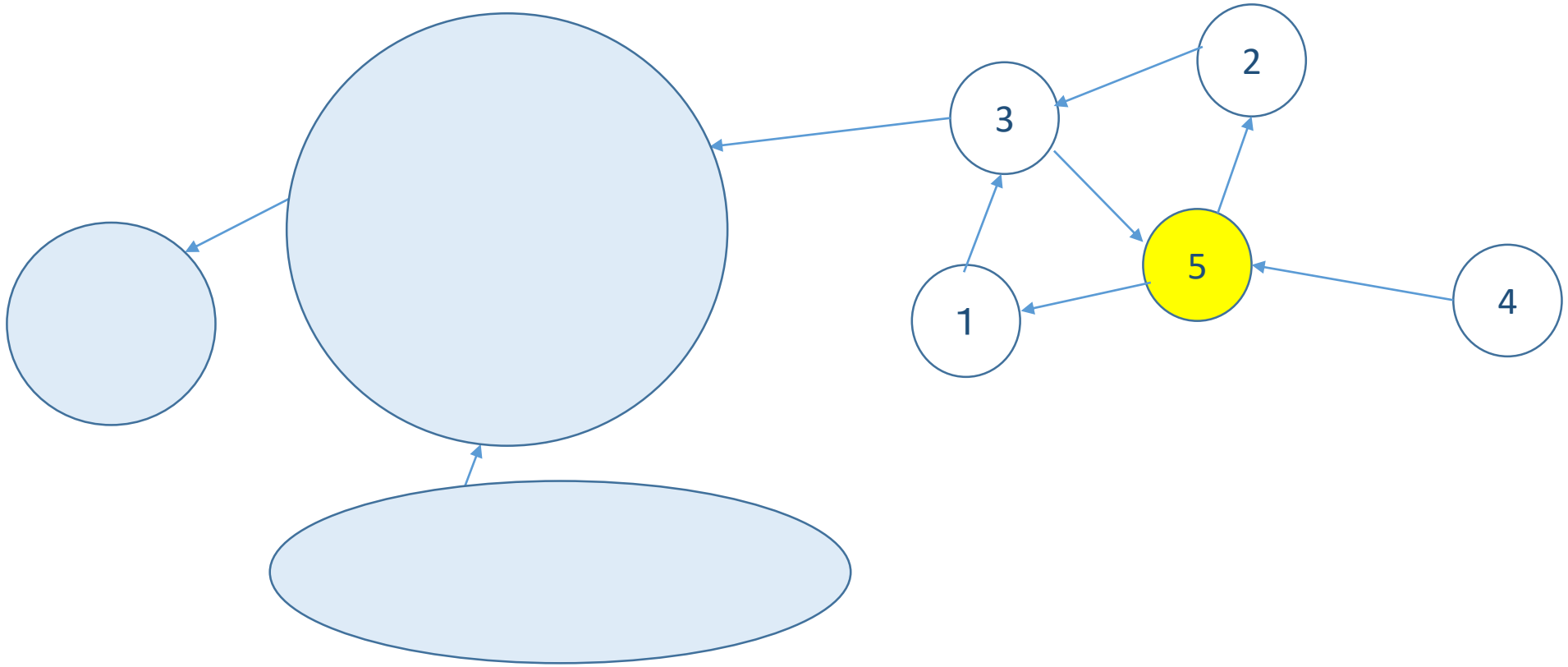
アルゴリズム(2回目のDFS)

通ることのできた頂点をまとめる

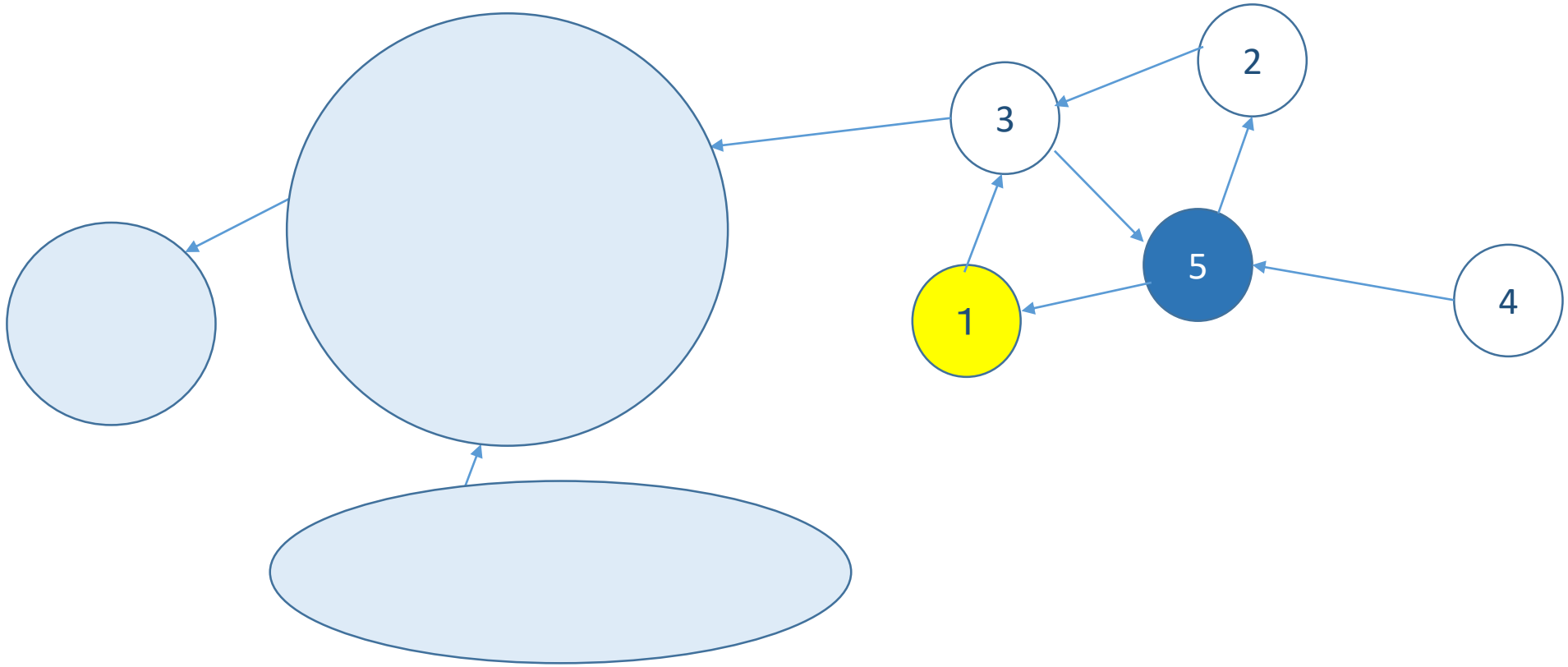


アルゴリズム(2回目のDFS)

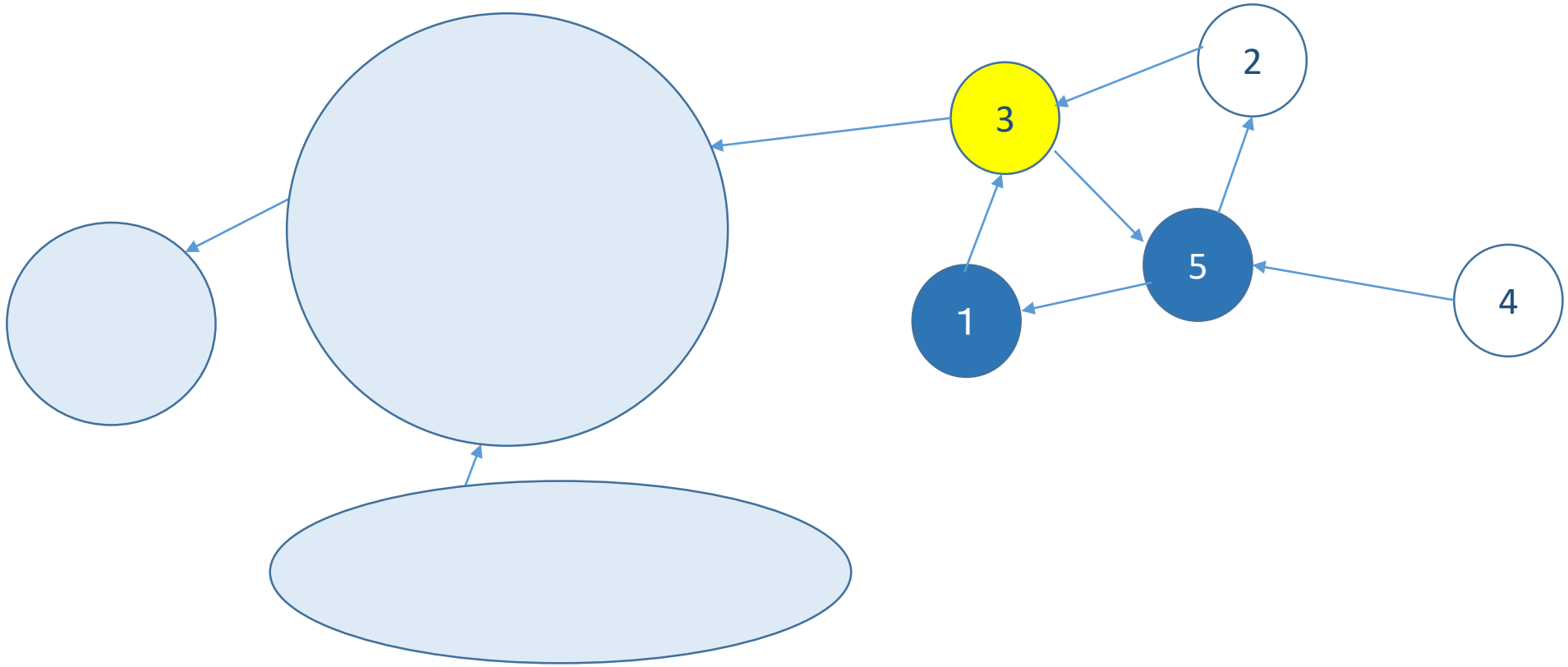
一番大きいところからDFS



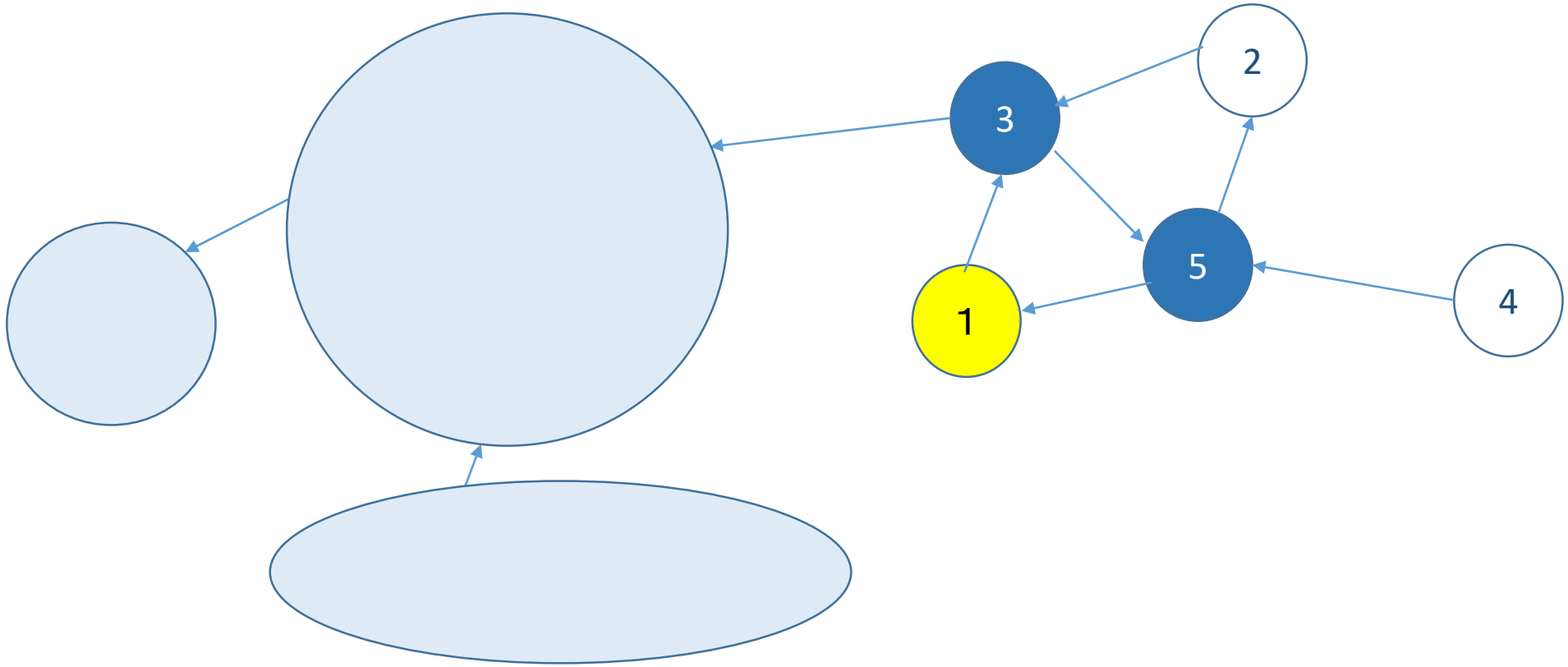
アルゴリズム(2回目のDFS)



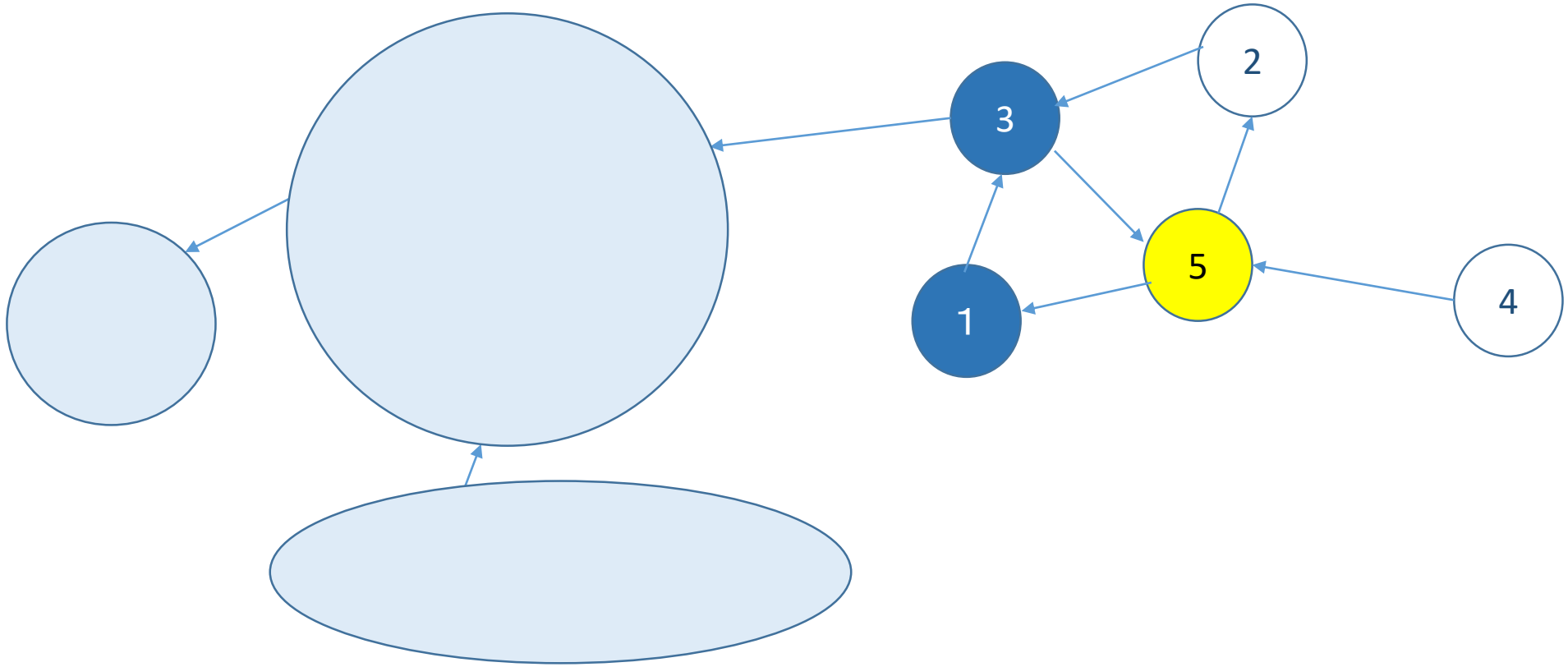
アルゴリズム(2回目のDFS)



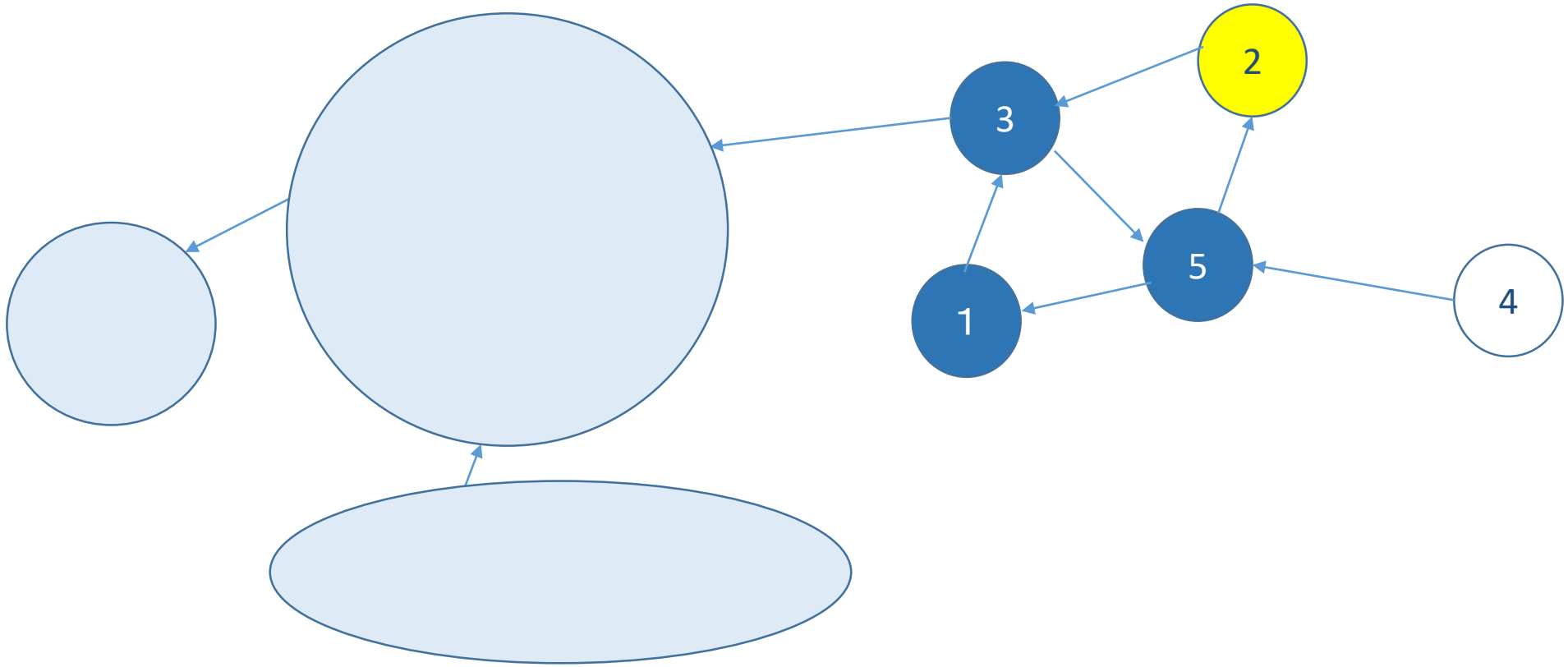
アルゴリズム(2回目のDFS)



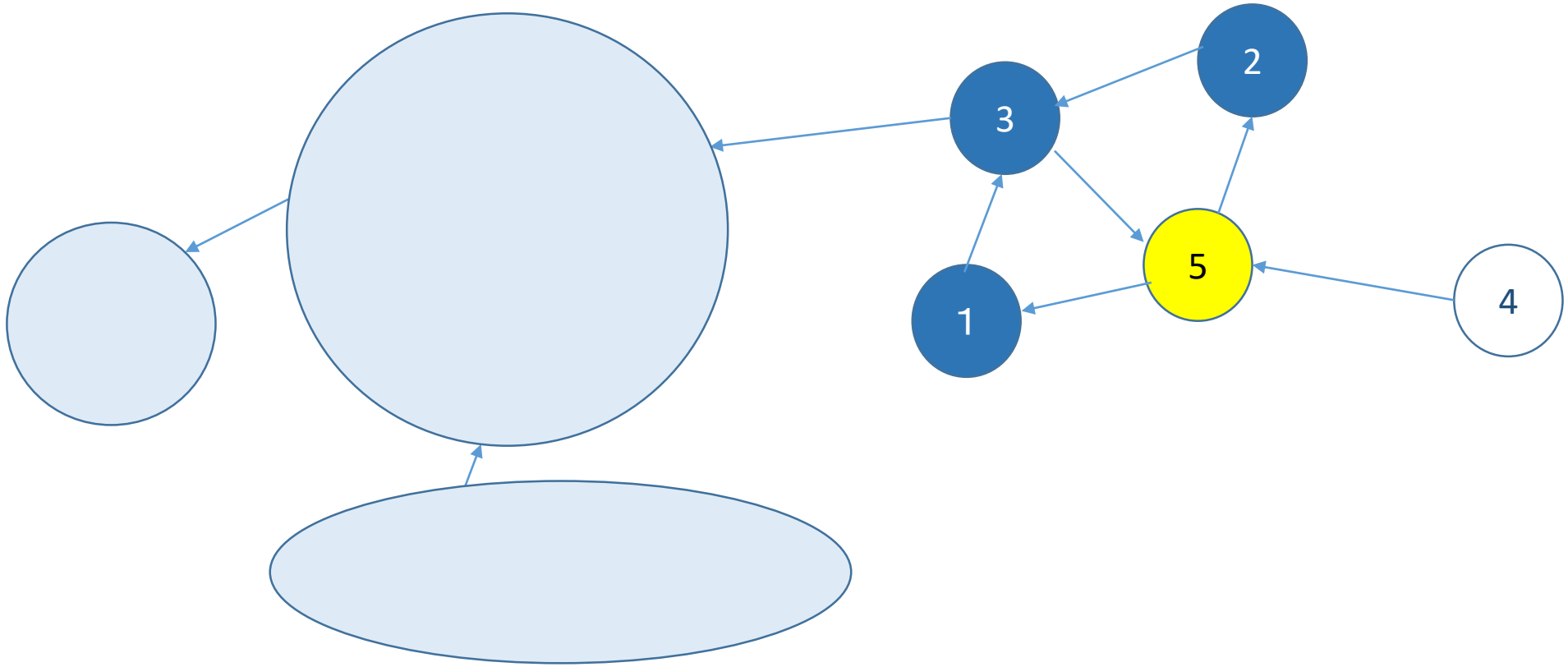
アルゴリズム(2回目のDFS)



アルゴリズム(2回目のDFS)

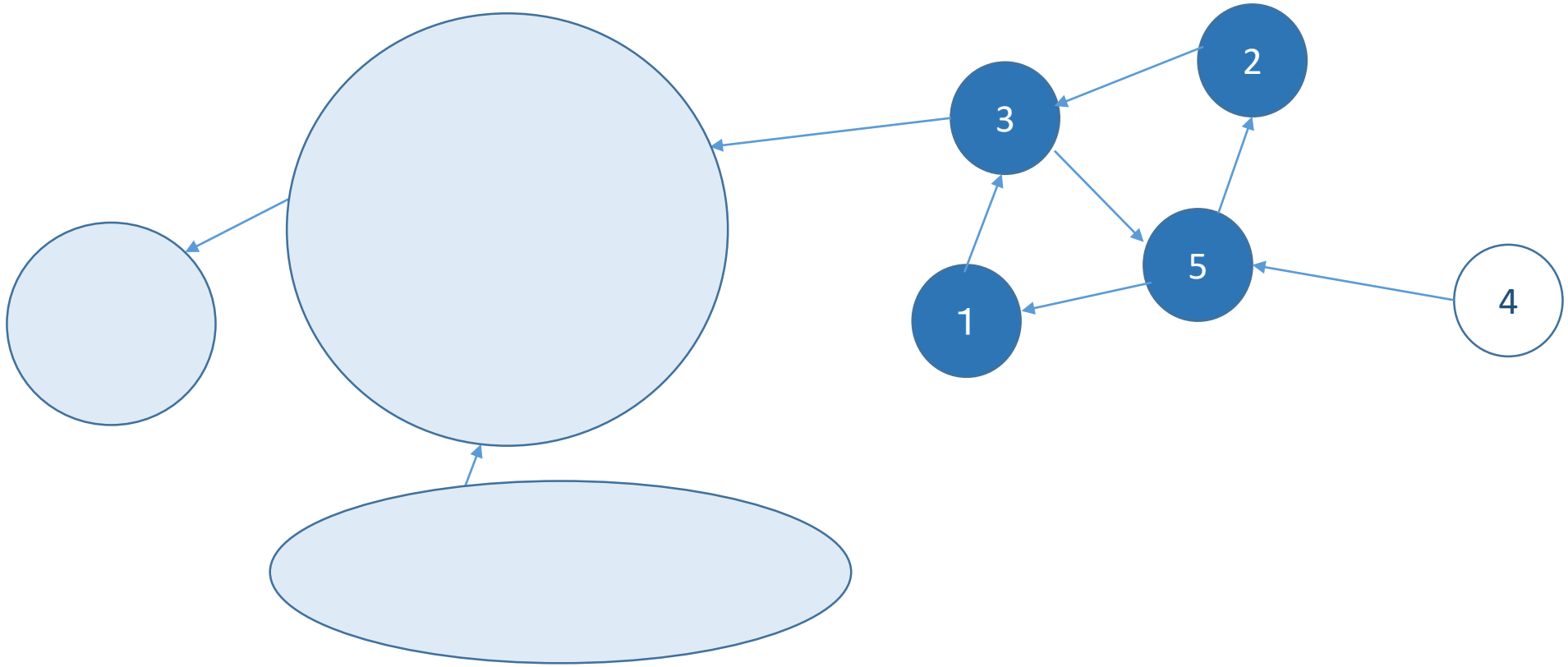


アルゴリズム(2回目のDFS)



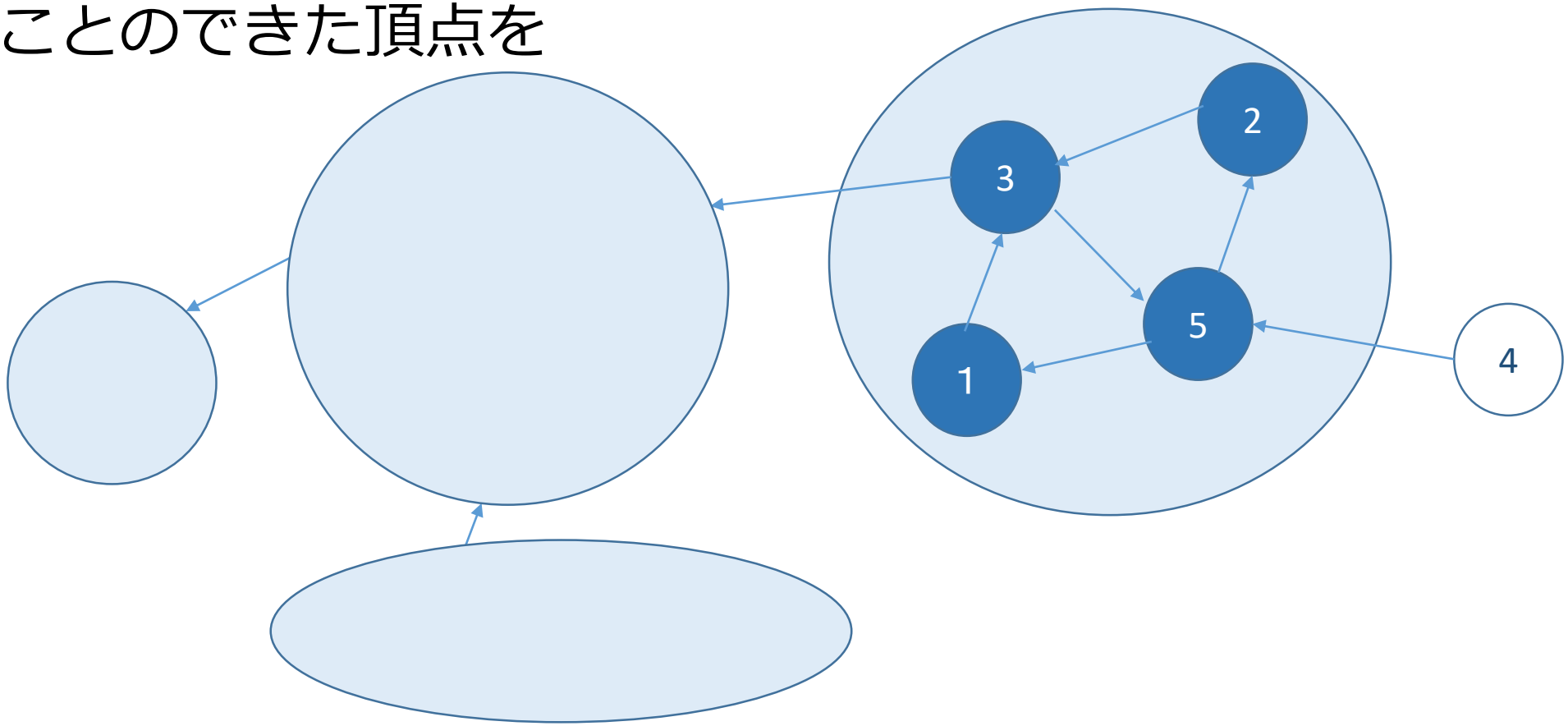
アルゴリズム(2回目のDFS)

DFS終わり



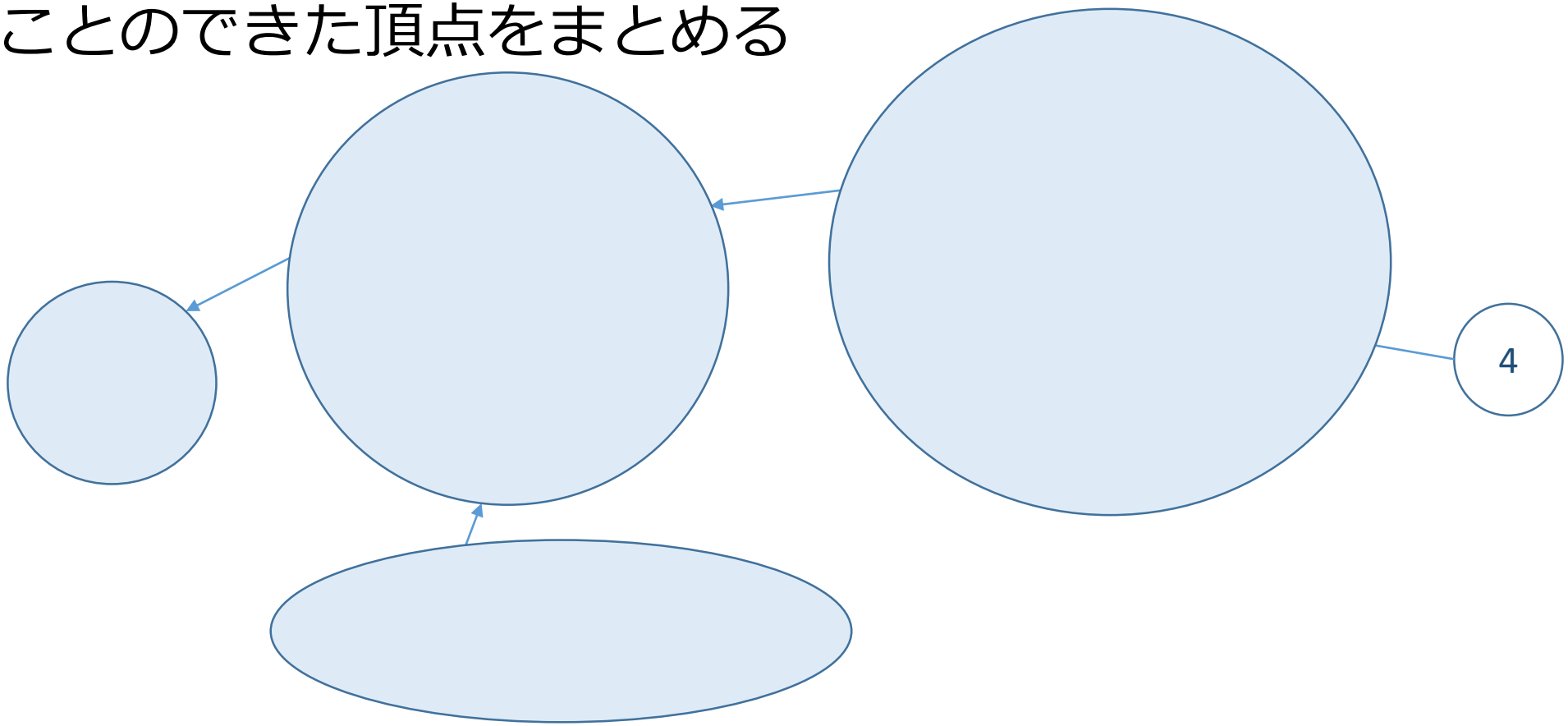
アルゴリズム(2回目のDFS)

通ることのできた頂点を



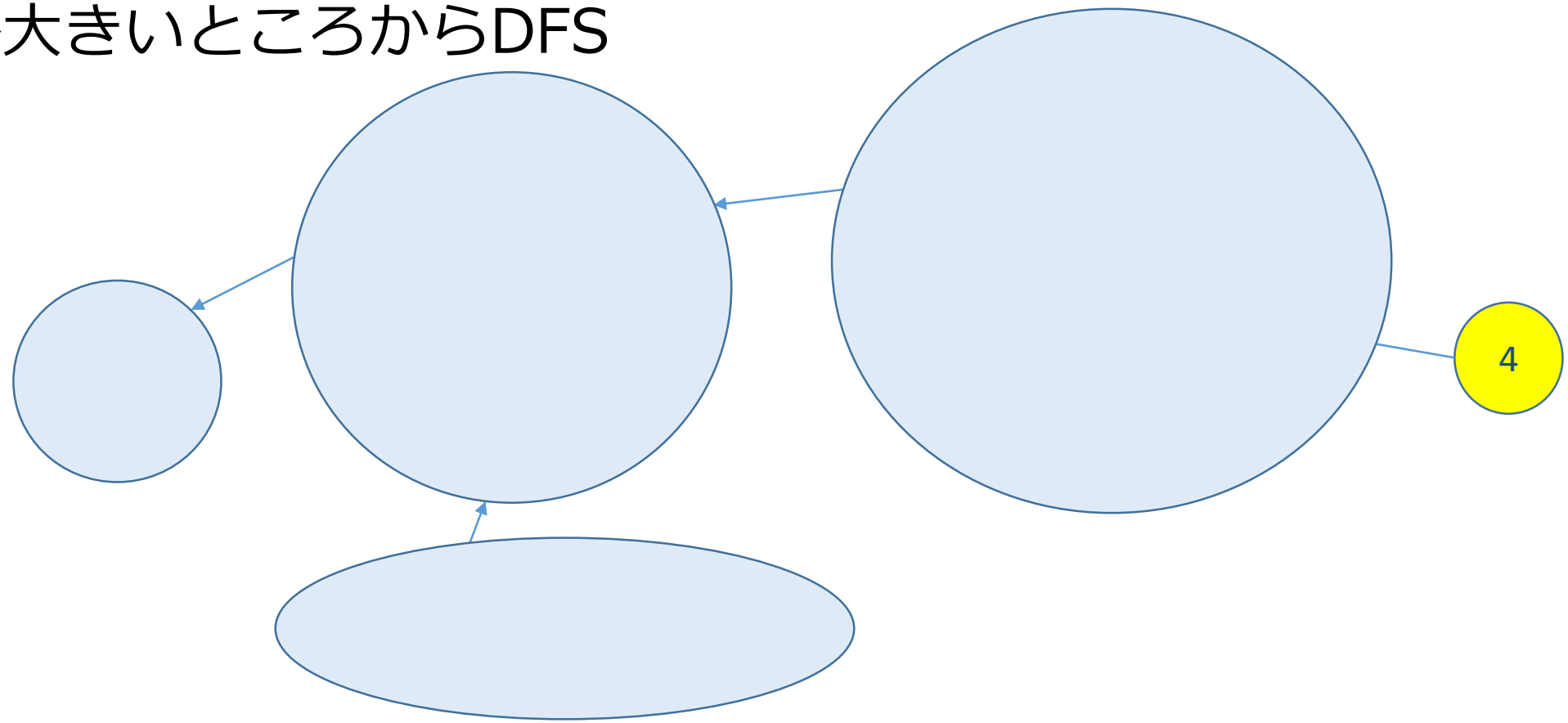
アルゴリズム(2回目のDFS)

通ることのできた頂点をまとめる



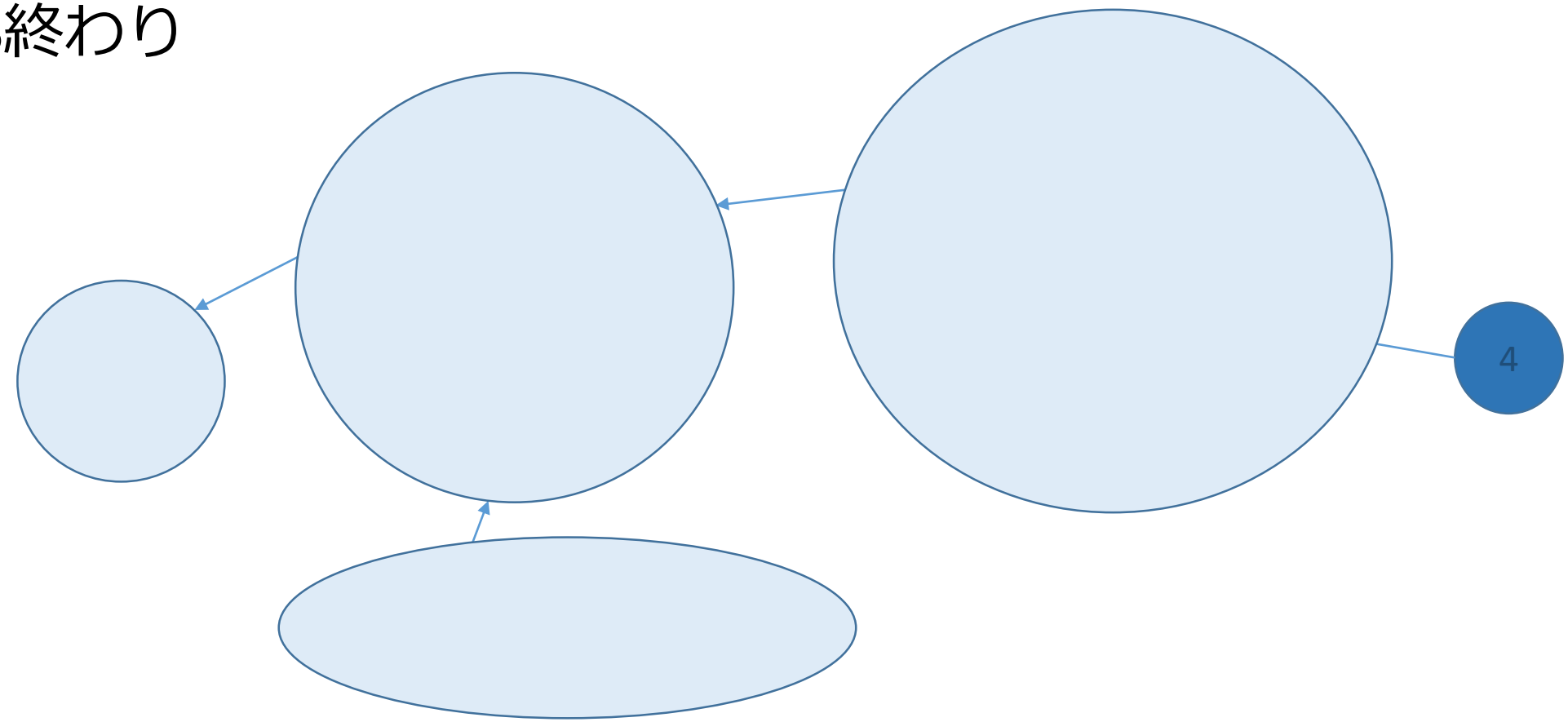
アルゴリズム(2回目のDFS)

一番大きいところからDFS



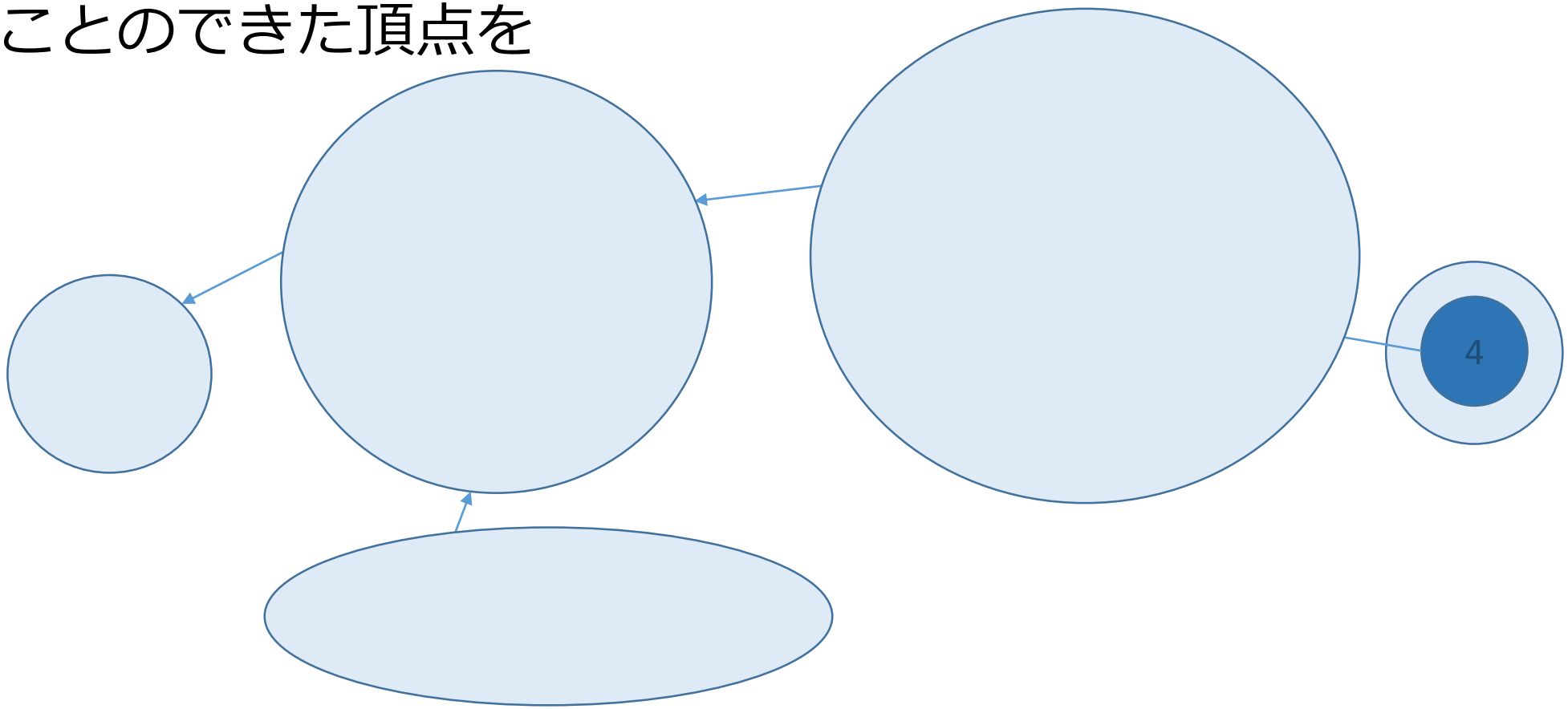
アルゴリズム(2回目のDFS)

DFS終わり



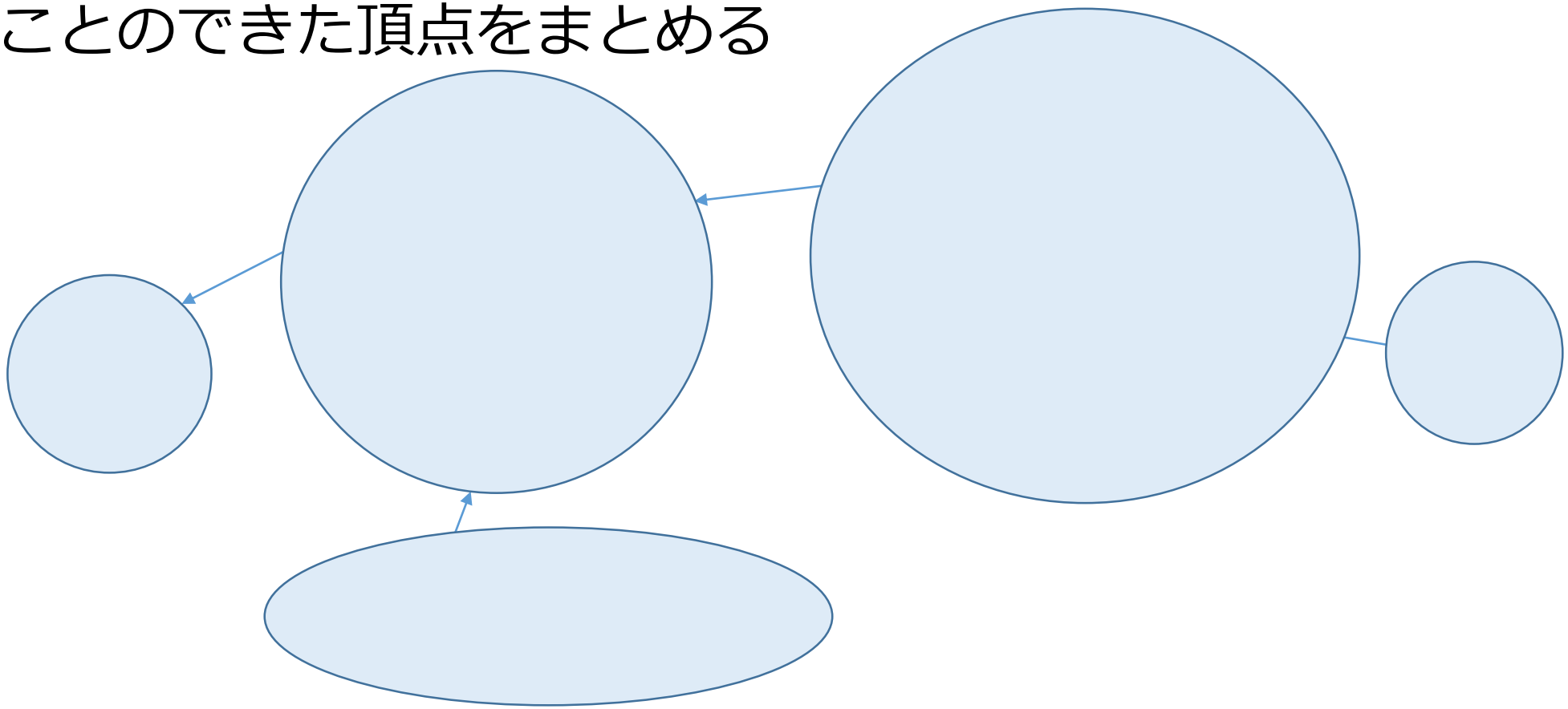
アルゴリズム(2回目のDFS)

通ることのできた頂点を



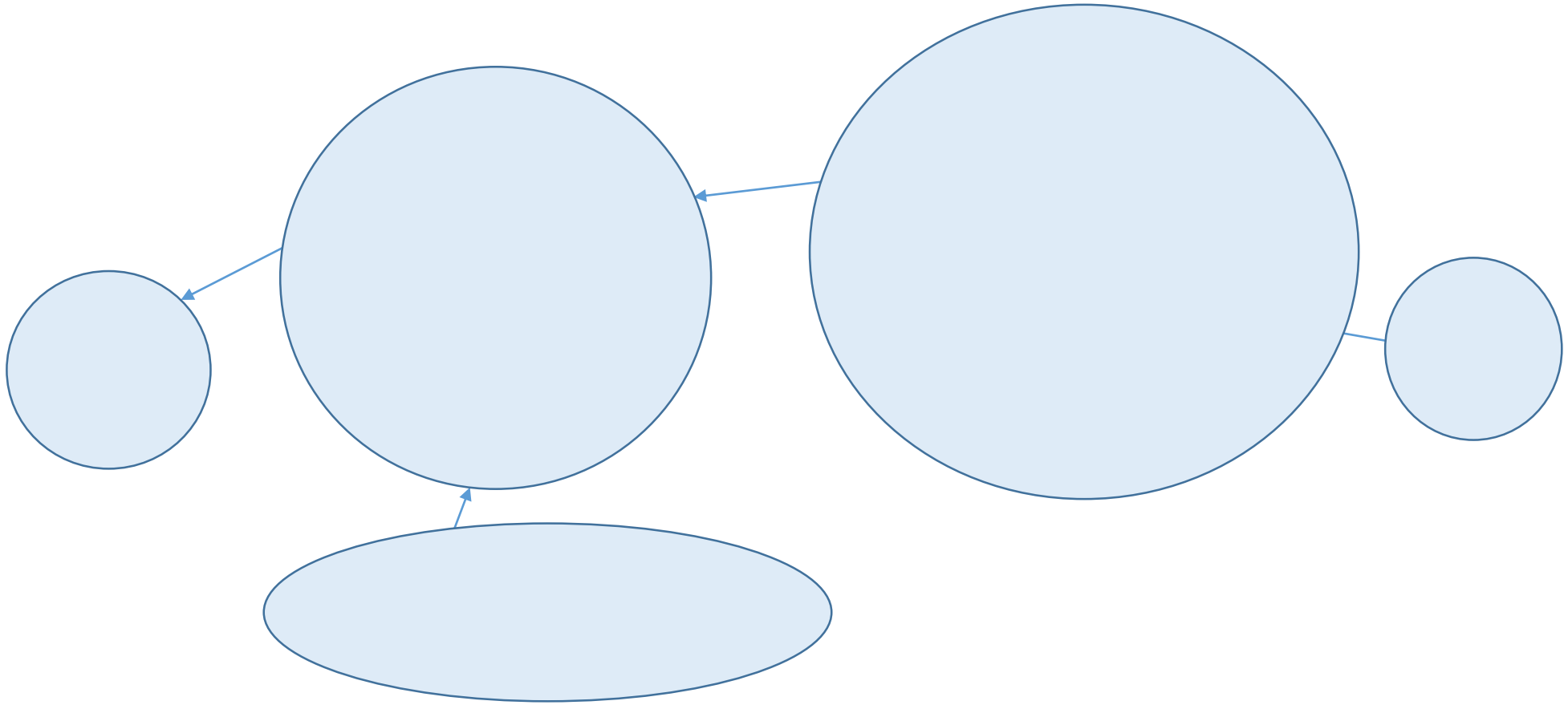
アルゴリズム(2回目のDFS)

通ることのできた頂点をまとめる



アルゴリズム(2回目のDFS)

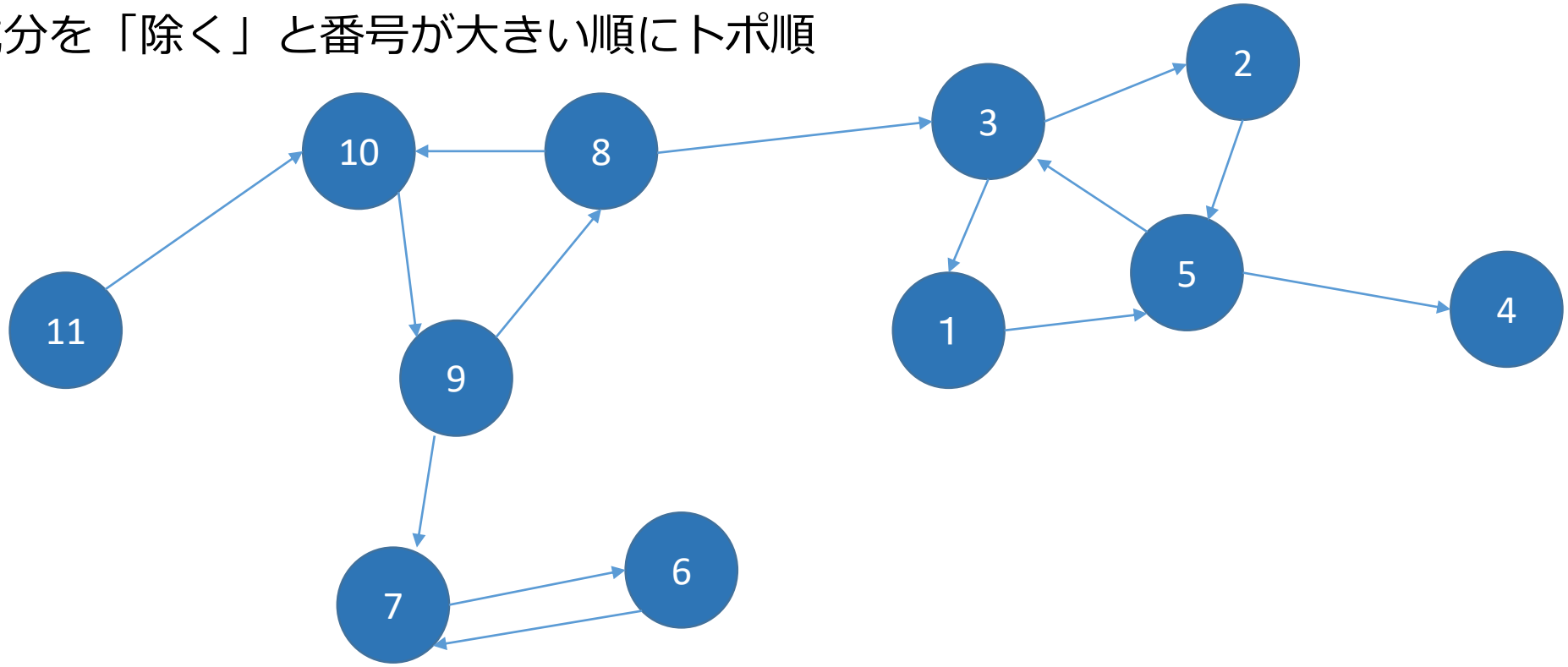
完成



アルゴリズムの正当性

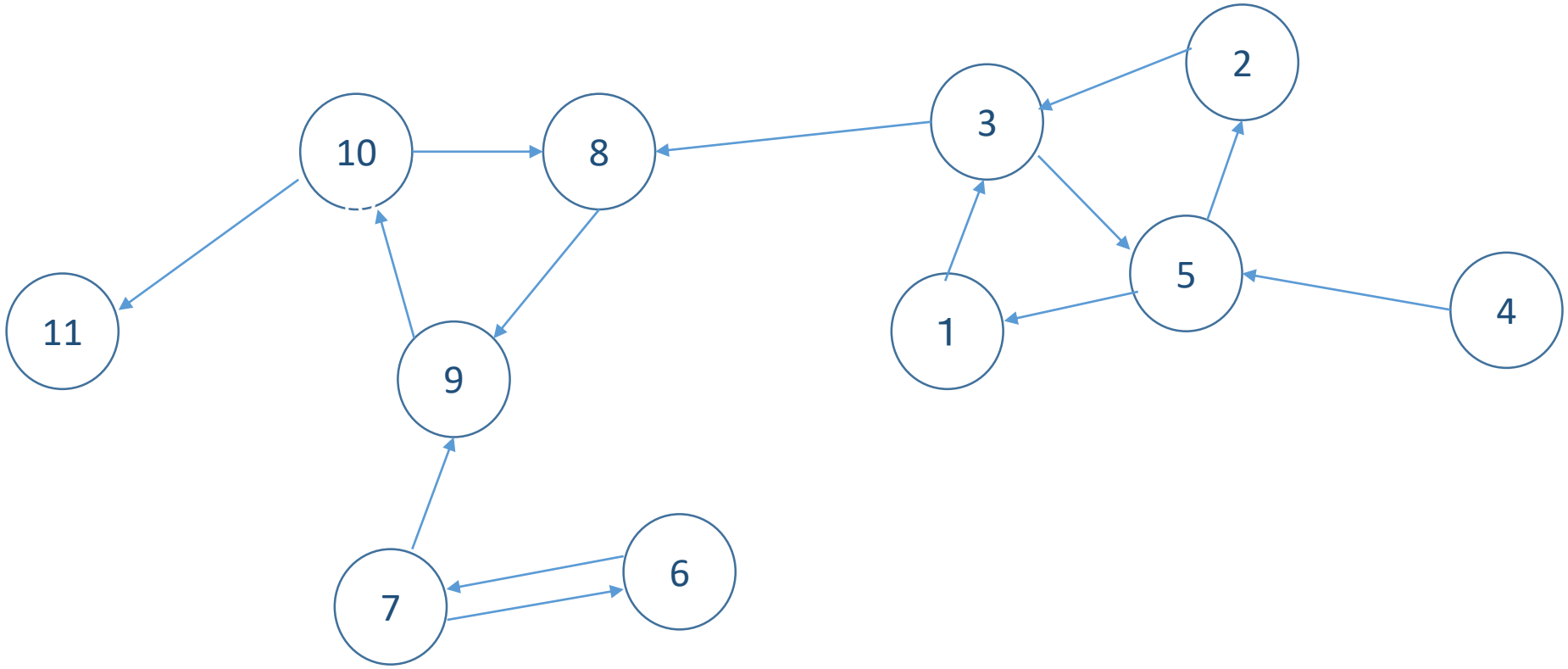
1回目のdfs後

強連結成分を「除く」と番号が大きい順にトポ順



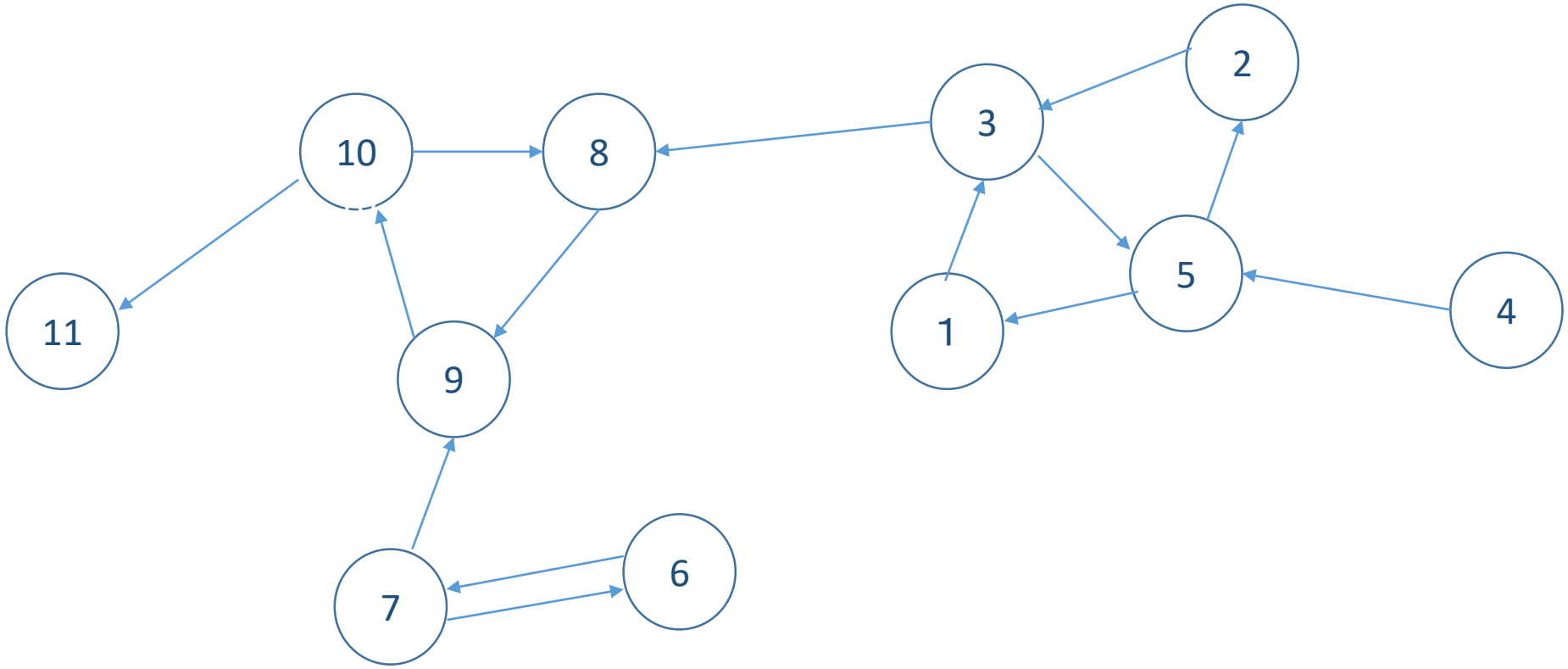
アルゴリズの正当性

逆辺にすることで、自分より若い頂点に行くことができない



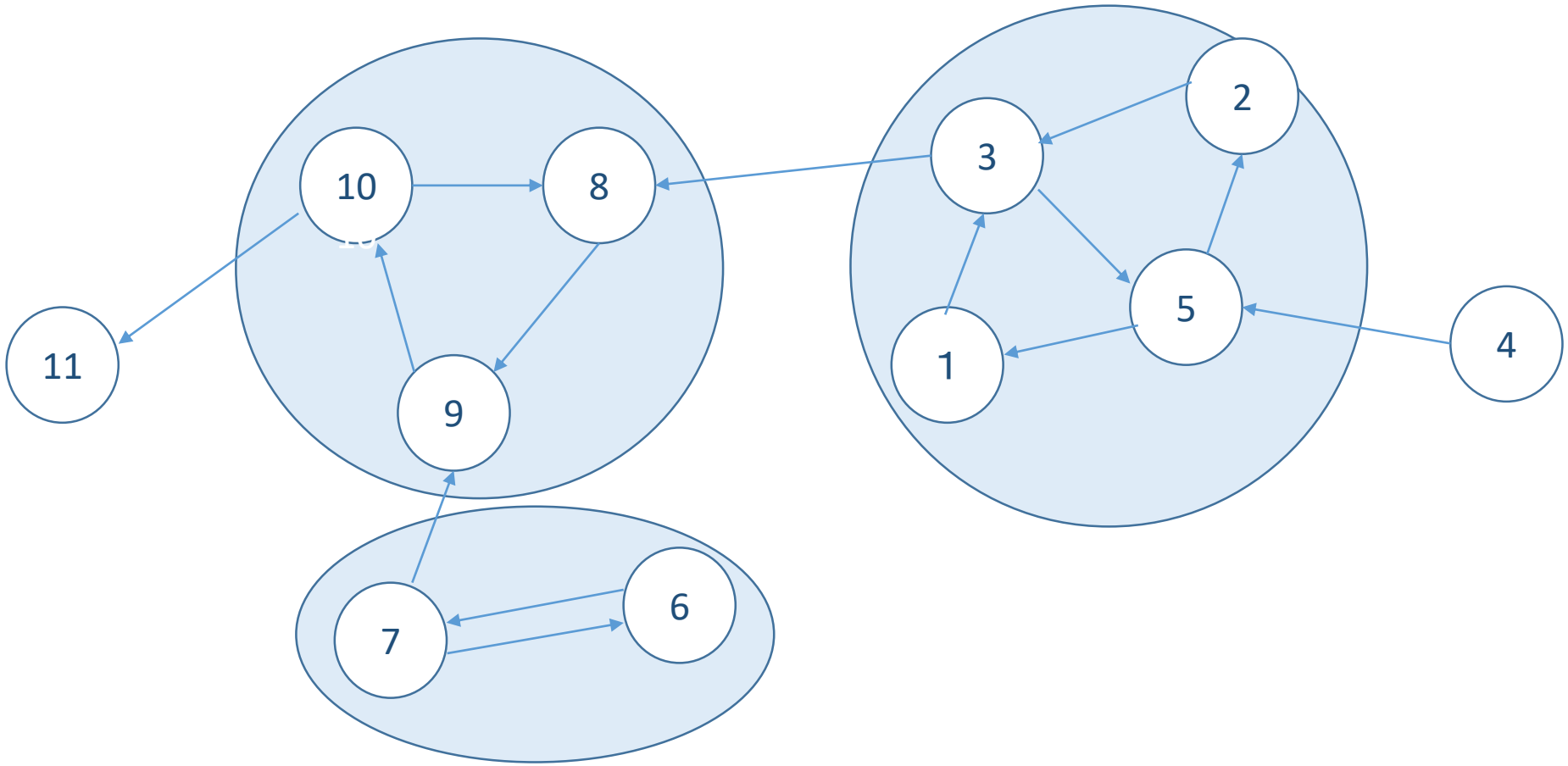
アルゴリズの正当性

ただし、強連結成分は辺を逆にしても強連結成分



アルゴリズの正当性

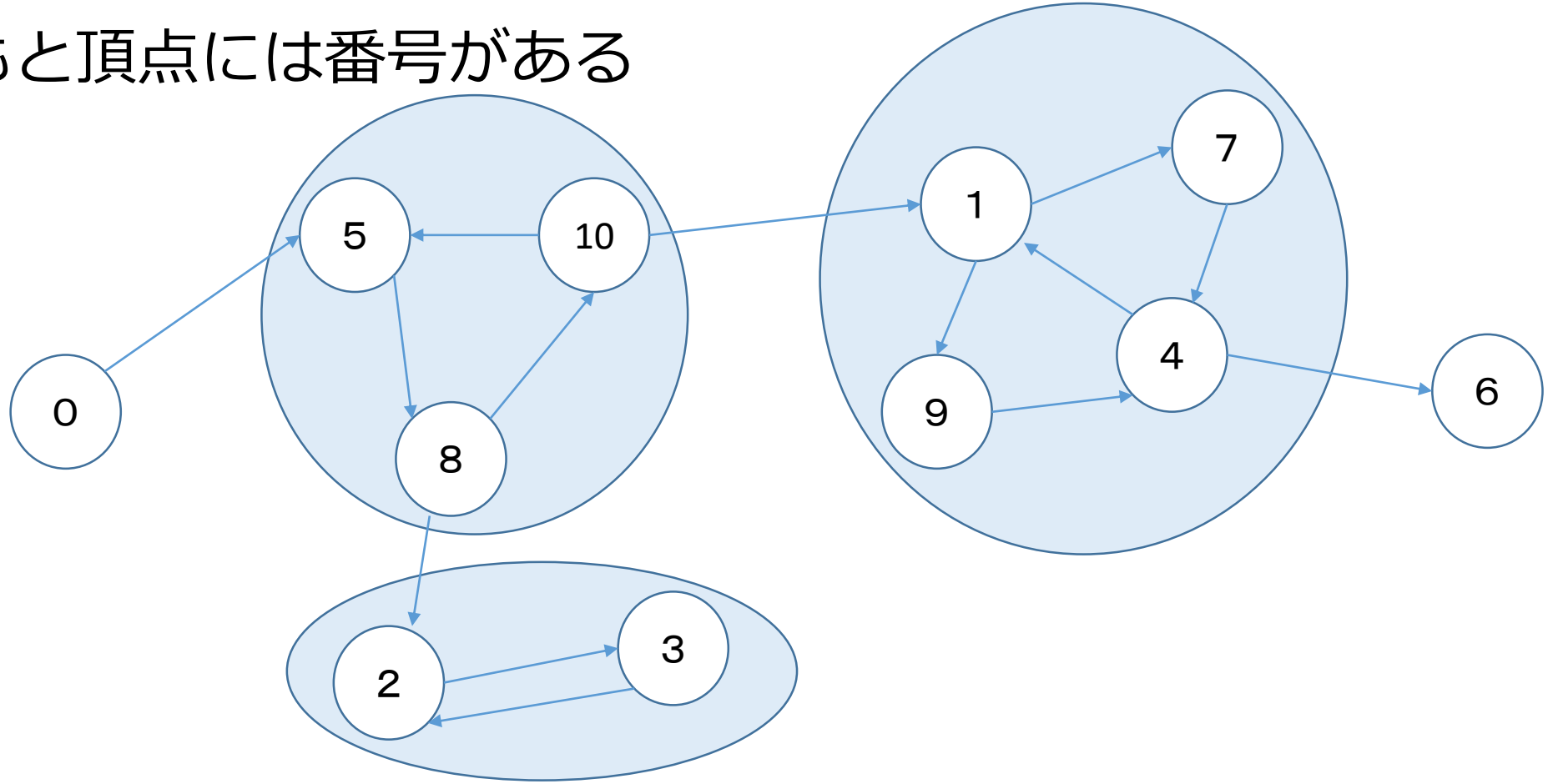
強連結成分内では全ての頂点を訪れることができる



実装上の注意点

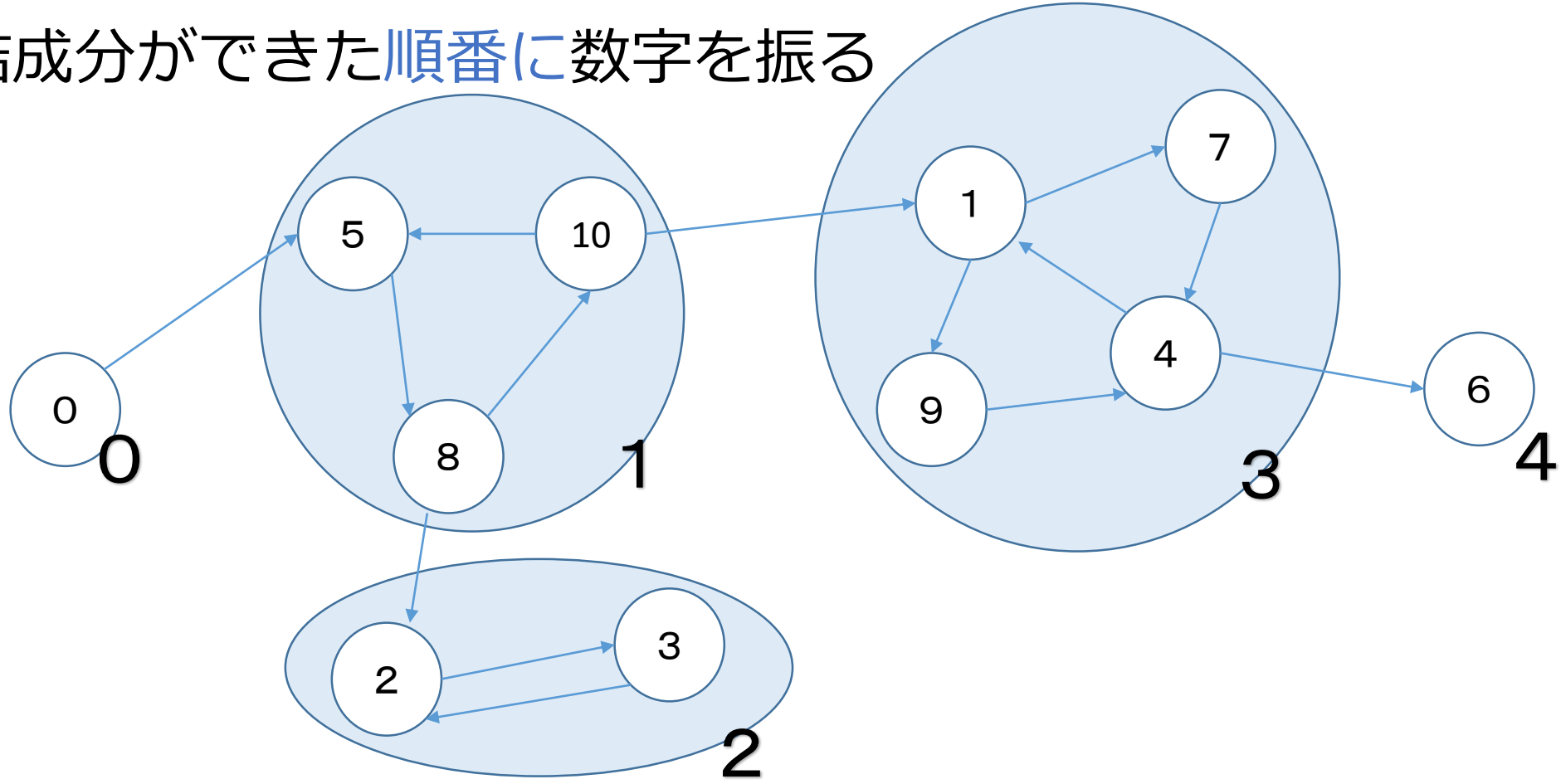
実装上の注意点

もともと頂点には番号がある



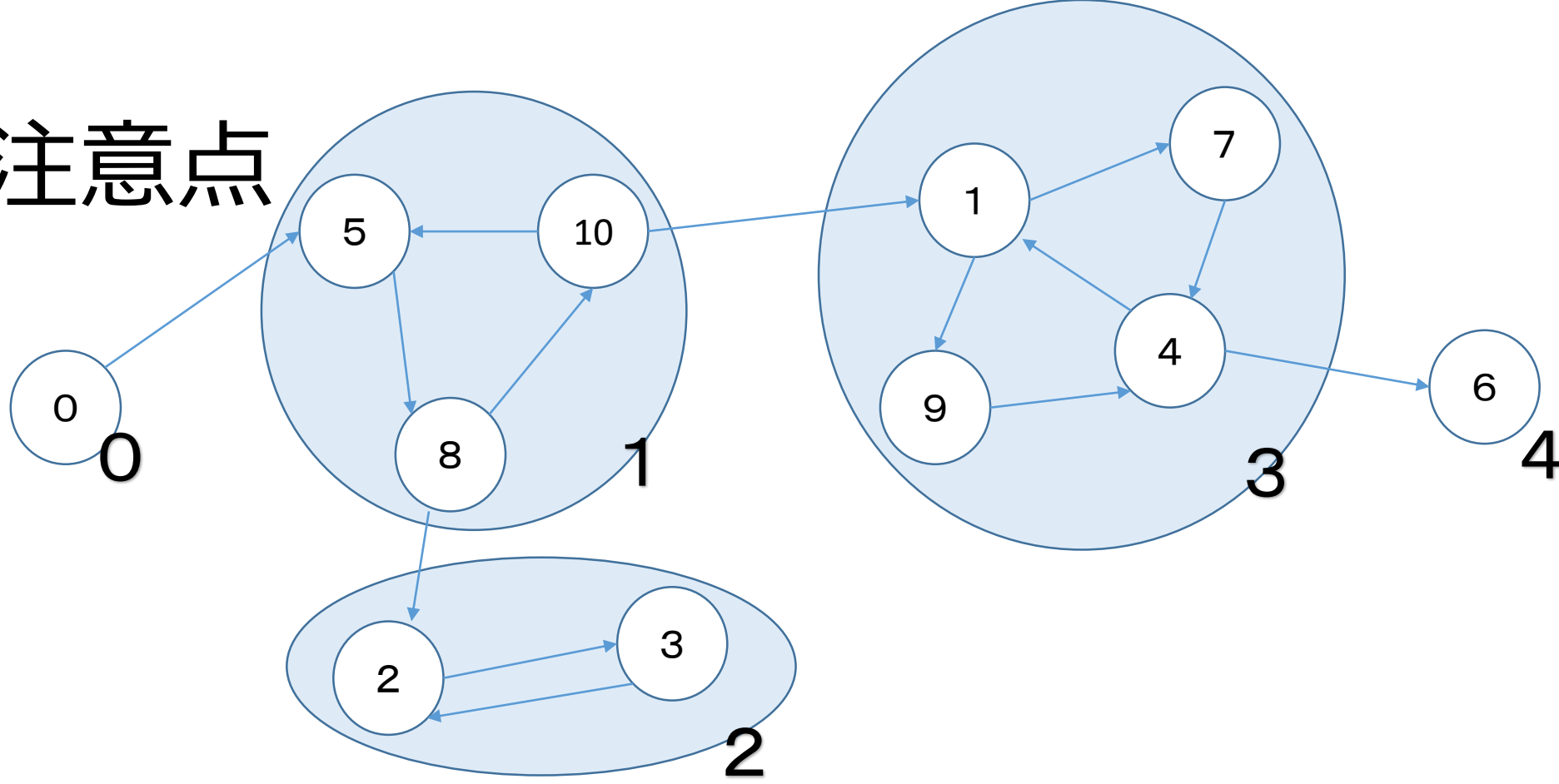
実装上の注意点

強連結成分ができた順番に数字を振る



実装上の注意点

その情報を
配列に



i	0	1	2	3	4	5	6	7	8	9	10
cmp	0	3	2	2	3	1	4	3	1	3	1

```cpp
int V, E;                                   //頂点数　辺数
vector<int> G[MAX_V];                       //グラフの隣接リスト表現(0-indexed)
vector<int> rG[MAX_V];                      //辺の向きを逆向きにしたグラフ
vector<int> vs;                             //帰りがけ順での頂点の並び
bool used[MAX_V];                           //dsfのために、すでに訪れたかどうかを管理する
int cmp[MAX_V];                             //属する強連結成分のトポロジカル順序

void dfs(int v){

    used[v] = true;
    for(int i = 0; i < (int)G[v].size(); i++){
        if(!used[G[v][i]]) dfs(G[v][i]);
    }

    //帰りがけ順に格納
    vs.push_back(v);
}

void rdfs(int v, int k){
    used[v] = true;
    //通ることのできた頂点を
    //kと番号を振る
    cmp[v] = k;
    for(int i = 0; i < (int)rG[v].size(); i++){
        if(!used[rG[v][i]]) rdfs(rG[v][i], k);
    }
}
```

```cpp
//返り値は強連結成分成分の個数
int scc(){
    memset(used, false, sizeof(used));
    vs.clear();

    //dfs1回目
    for(int v = 0; v < V; v++){
        if(!used[v]) dfs(v);

    }
    memset(used, false, sizeof(used));

    //dfs2回目
    int k = 0;
    for(int i = V - 1; i >= 0; i--){
        if(!used[vs[i]]){
            rdfs(vs[i], k);
            k++;
        }
    }
    return k;
}
```

```cpp
int main(){

    //入力
    cin >> V >> E;                          //頂点数  辺数

    for(int i = 0; i < E; i++){
        int from, to; cin >> from >> to;        //from -> toへの有向辺
        G[from].push_back(to);
        rG[to].push_back(from);
    }

    memset(used, false, sizeof(used));
    cout << scc() << endl;

    int Q; cin >> Q;
    for(int i = 0; i < Q; i++){
        int u, v; cin >> u >> v;
        cout << (cmp[u] == cmp[v]) << endl;
    }

    return 0;
}
```

目次

・グラフの話

-基礎

-dfs

・トポロジカルソート

・強連結成分分解

・問題への応用

山田山本問題(atcoder 600点)

https://tenka1-2016-quala.contest.atcoder.jp/tasks/tenka1_2016_qualA_c

問題

文字列A_i を文字列 B_i より辞書順で小さくしたいという要求が N 個与えられる。 与えられた要求をすべて満たすような、アルファベットの順番を求める。 条件を満たすようなアルファベットの順番が複数存在する場合は、並べ替えられる前のアルファベットの順番での辞書順最小の順番を求める。

制約

- $1 \leq N \leq 1000$
- $A_i \neq B_i$
- $1 \leq |A_i|, |B_i| \leq 1000$
- A_i, B_iは英小文字で構成される

山田山本問題(atcoder 600点)

例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓
yamamoto
yamada

abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto

yamada

同じ

abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto
yamada

同じ

abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto
yamada

同じ

abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto
yamada

同じ

abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto
yamada

違う

abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

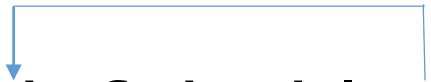
例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto

yamada

つまり、m -> dの順番でアルファベットを構成しなくてはいけない



abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

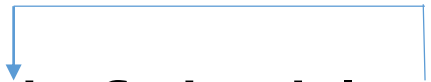
例えば、“yamamoto”が“yamada”より辞書順小さくしたいという要求を満たすことを考える

↓

yamamoto

yamada

要求ごとに辺を張ってトポソをすればよい



abcdefghijklmnopqrstuvwxyz

山田山本問題(atcoder 600点)

- 細かい問題1

トポソが構成できない場合は？ (-1を出力したい)

そもそもDAGじゃなくなる

先のアルゴリズムを少し変えれば簡単に判定ができる

トポロジカルソート閉路検出

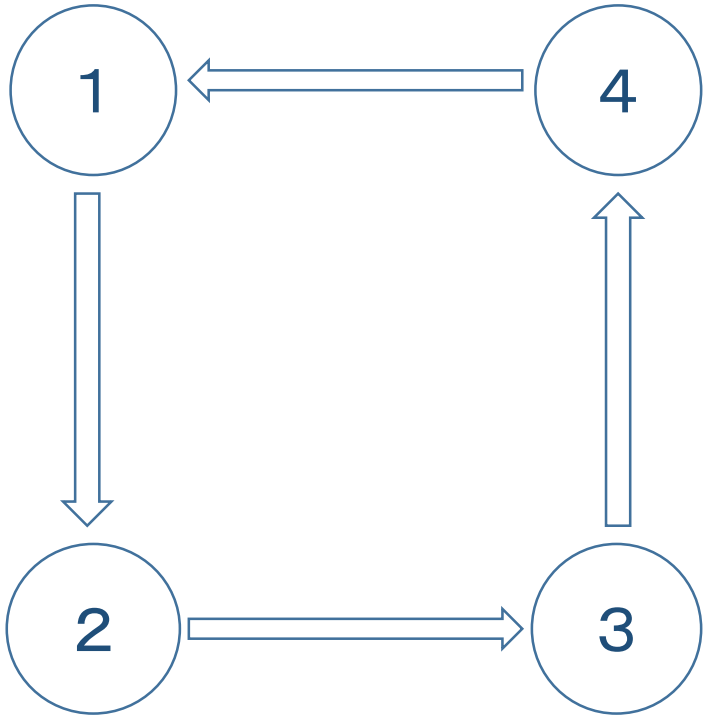
- 細かい問題1

DAGじゃないということは右のような閉路が存在

閉路を構成する頂点は(取り除かれるまでは)全ての入次数が1以上

入次数が0になることはないので、閉路が取り除かれないままプログラムが終了する

取り除かれた頂点数と、元の頂点数を比較するだけで判定可能

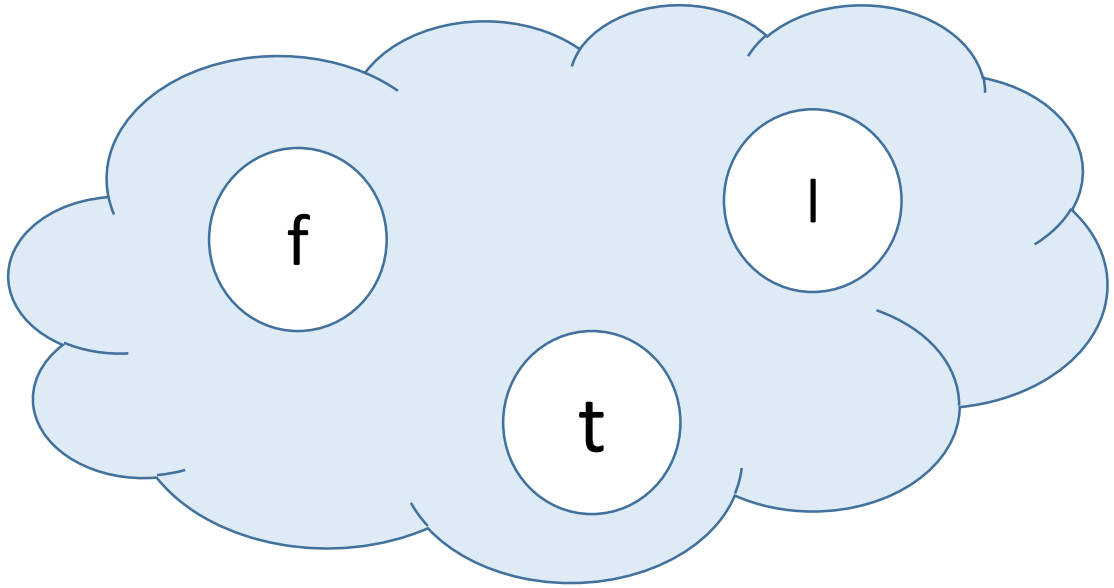


山田山本問題(atcoder 600点)

- 細かい問題2

-トポソが複数ある場合は？(元の辞書順最小を出力したい)

-例えば、入次数0の頂点が右のように複数入っている



山田山本問題(atcoder 600点)

- 細かい問題2

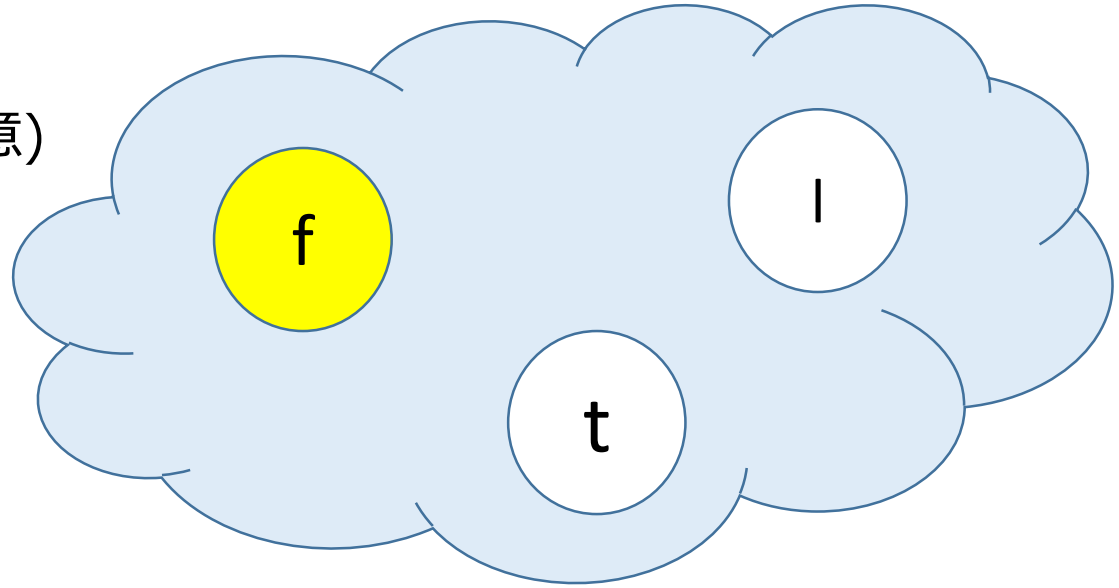
-トポソが複数ある場合は？(元の辞書順最小を出力したい)

-例えば、入次数0の頂点が右のように複数入っている

-この中から取り出す際に、元の辞書順最小を取り出せばよさそう

priority_queueを使えばよい！

(デフォルトでは大きい順に出てくるので注意)



```cpp
//山田山本問題
#include<iostream>
#include<vector>
#include<string>
#include<queue>
using namespace std;

int main(){


    vector<vector<int> > adj(26);       //隣接リスト
    vector<int> d(26, 0);               //入次数
    int n; cin >> n;
    bool dame = false;
    //入力受け取り
    for(int i = 0; i < n; i++){
        string A, B; cin >> A >> B;
        int N = A.length(), M = B.length();
        bool ok = false;
        for(int j = 0; j < min(N, M); j++){
            if(A[j] != B[j]){
                //cout << j << endl;
                int from = (int)(A[j] - 'a');
                int to = (int)(B[j] - 'a');
                d[to]++;
                adj[from].push_back(to);
                ok = true;
                break;
            }
        }
        if(dame){
            cout << -1 << endl;
            return 0;
        }
    }

    //トポロジカルソート
    priority_queue<int> pq;
    for(int i = 0; i < 26; i++){
        if(d[i] == 0) pq.push(-i);
    }

    string ans = "";
    while(!pq.empty()){
        int v = -pq.top();
        pq.pop();
        ans.push_back((char)(v + 'a'));

        for(int i = 0; i < (int)adj[v].size(); i++){
            //vから行きつく頂点の入次数を1減らし
            d[adj[v][i]]--;
            //0になったらpqに突っ込む
            if(d[adj[v][i]] == 0) pq.push(-adj[v][i]);
        }
    }

    //ansがアルファベットの個数分なかったら-1
    if(ans.size() != 26) cout << -1 << endl;
    else cout << ans << endl;

    return 0;
}
```

余談

トポロジカルソートがただ一つ存在するのか、それとも複数あるのか
どうやって判定すればよいか

ただ一つの場合、queueから取り出すときに候補が一つしかないはず
つまり、queueの要素数が常に1
毎回それを判定すればよい！

```cpp
    bool flag = false;                                  //トポソがただ一つ存在するか
    vector<int> ans;

    while(!q.empty()){

        flag |= (q.size() > 1);
        int v = q.front();
        q.pop();
        ans.push_back(v);

        //vから伸びている有向辺をなめる
        for(int i = 0; i < (int)adj[v].size(); i++){
            d[adj[v][i]]--;
            //辺を取り除いた後に入次数が0になったら
            if(d[adj[v][i]] == 0) q.push(adj[v][i]);
        }

    }

    for(int i = 0; i < (int)ans.size(); i++) cout << ans[i] + 1 << endl;
    cout << flag << endl;
    return 0;
}
```

夏合宿の朝は早い(aoj icpc 450)

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2748

問題

- N人の人がいる、i番目の人は確率 pi で寝坊する
- 起きられた人は、連絡先を知っている人にモーニング コールし、確実に起こすことができる
- モーニングコールで起きた人もモーニングコールする
- 全員がちゃんと起きられる確率は？

制約

$2 \leq N \leq 100$

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

0.6	0.6	確率
起きる	起きる	
起きる	寝つぼる	
寝つぼる	起きる	
寝つぼる	寝つぼる	

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

0.6	0.6	確率
起きる	起きる	0.16
起きる	寝つぼる	
寝つぼる	起きる	
寝つぼる	寝つぼる	

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

0.6	0.6	確率
起きる	起きる	0.16
起きる	寝つぼる	0.24
寝つぼる	起きる	
寝つぼる	寝つぼる	

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

0.6	0.6	確率
起きる	起きる	0.16
起きる	寝つぼる	0.24
寝つぼる	起きる	0.24
寝つぼる	寝つぼる	

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

0.6	0.6	確率
起きる	起きる	0.16
起きる	寝つぼる	0.24
寝つぼる	起きる	0.24
寝つぼる	寝つぼる	0.36

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

	0.6	0.6	確率
ok	起きる	起きる	0.16
ok	起きる	寝つぼる	0.24
ng	寝つぼる	起きる	0.24
ng	寝つぼる	寝つぼる	0.36

夏合宿の朝は早い(aoj icpc 450)

入力例1

2

0.60 1 2

0.60 0

	0.6	0.6	確率
ok	起きる	起きる	0.16
ok	起きる	寝つぼる	0.24
ng	寝つぼる	起きる	0.24
ng	寝つぼる	寝つぼる	0.36

全てはこいつにかかってる

夏合宿の朝は早い(aoj icpc 450)

入力例2

2

0.60 1 2

0.60 1 1

0.6	0.6	確率
起きる	起きる	0.16
起きる	寝つぼる	0.24
寝つぼる	起きる	0.24
寝つぼる	寝つぼる	0.36

夏合宿の朝は早い(aoj icpc 450)

入力例2

2

0.60 1 2

0.60 1 1

	0.6	0.6	確率
ok	起きる	起きる	0.16
ok	起きる	寝つぼる	0.24
ok	寝つぼる	起きる	0.24
ng	寝つぼる	寝つぼる	0.36

全員寝つぼるときがまずい！

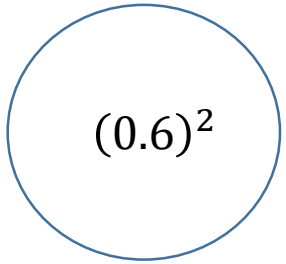
夏合宿の朝は早い(aoj icpc 450)

入力例2

2

0.60 1 2

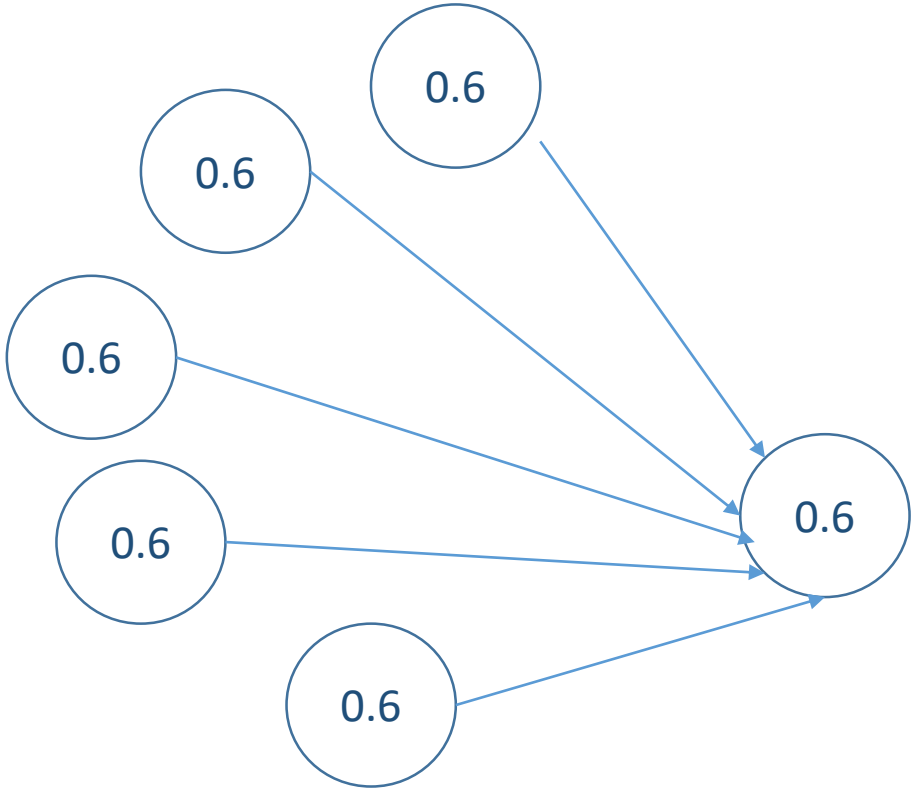
0.60 1 1



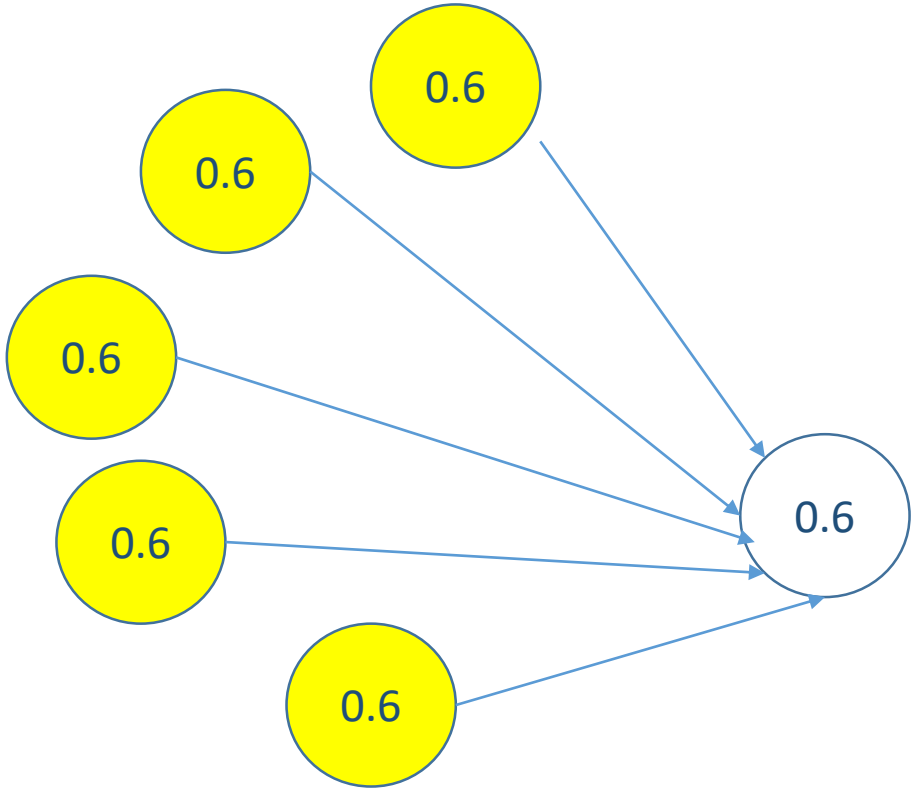
	0.6	0.6	確率
ok	起きる	起きる	0.16
ok	起きる	寝つぼる	0.24
ok	寝つぼる	起きる	0.24
ng	寝つぼる	寝つぼる	0.36

一つにまとめられるんじゃね？

夏合宿の朝は早い(aoj icpc 450)

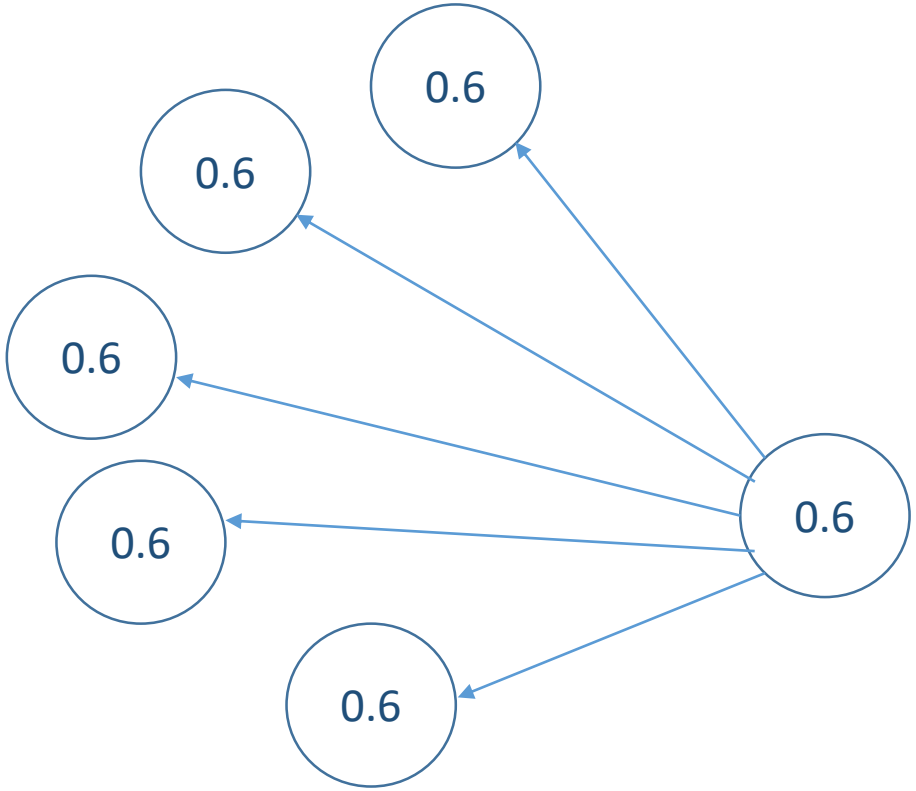


夏合宿の朝は早い(aoj icpc 450)

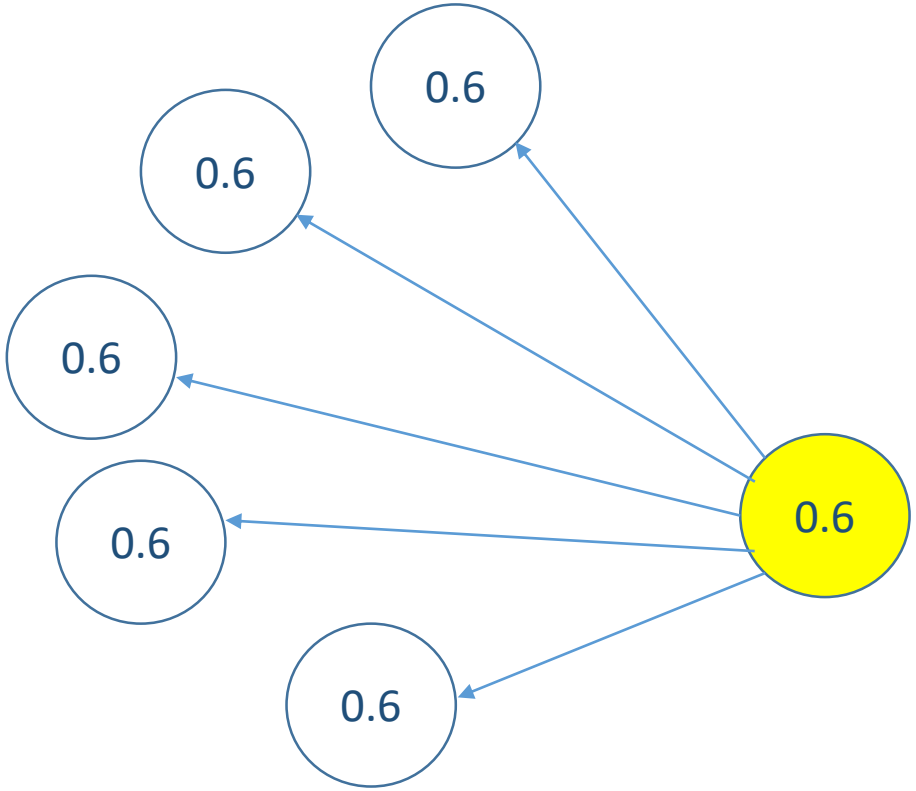


全てはこいつらにかかってる

夏合宿の朝は早い(aoj icpc 450)

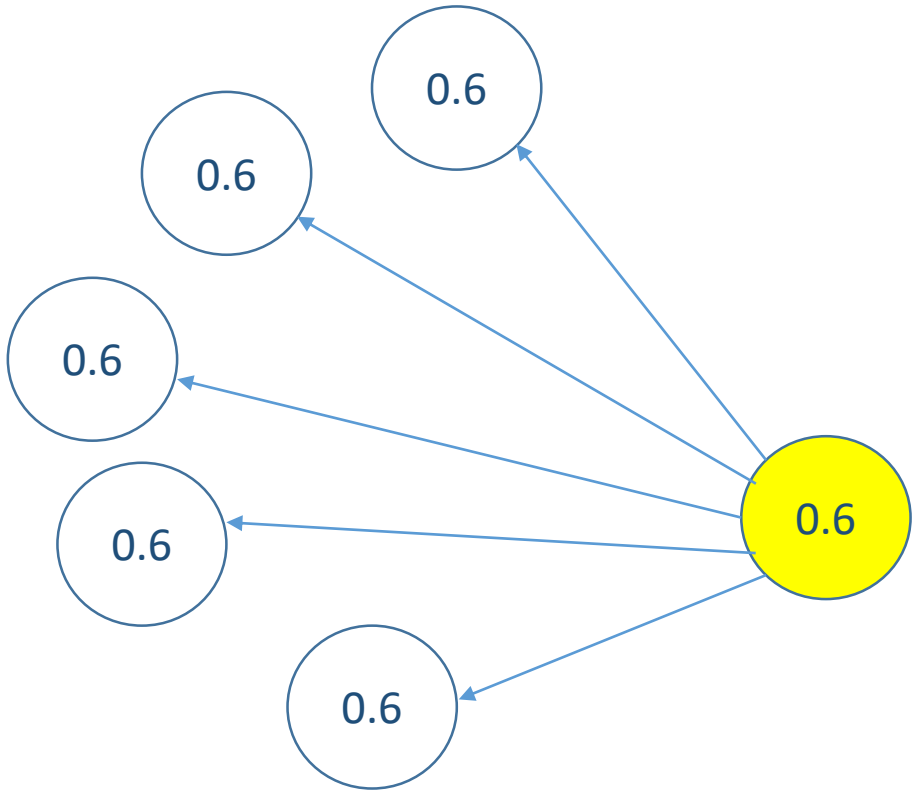


夏合宿の朝は早い(aoj icpc 450)



全てはこいつにかかってる

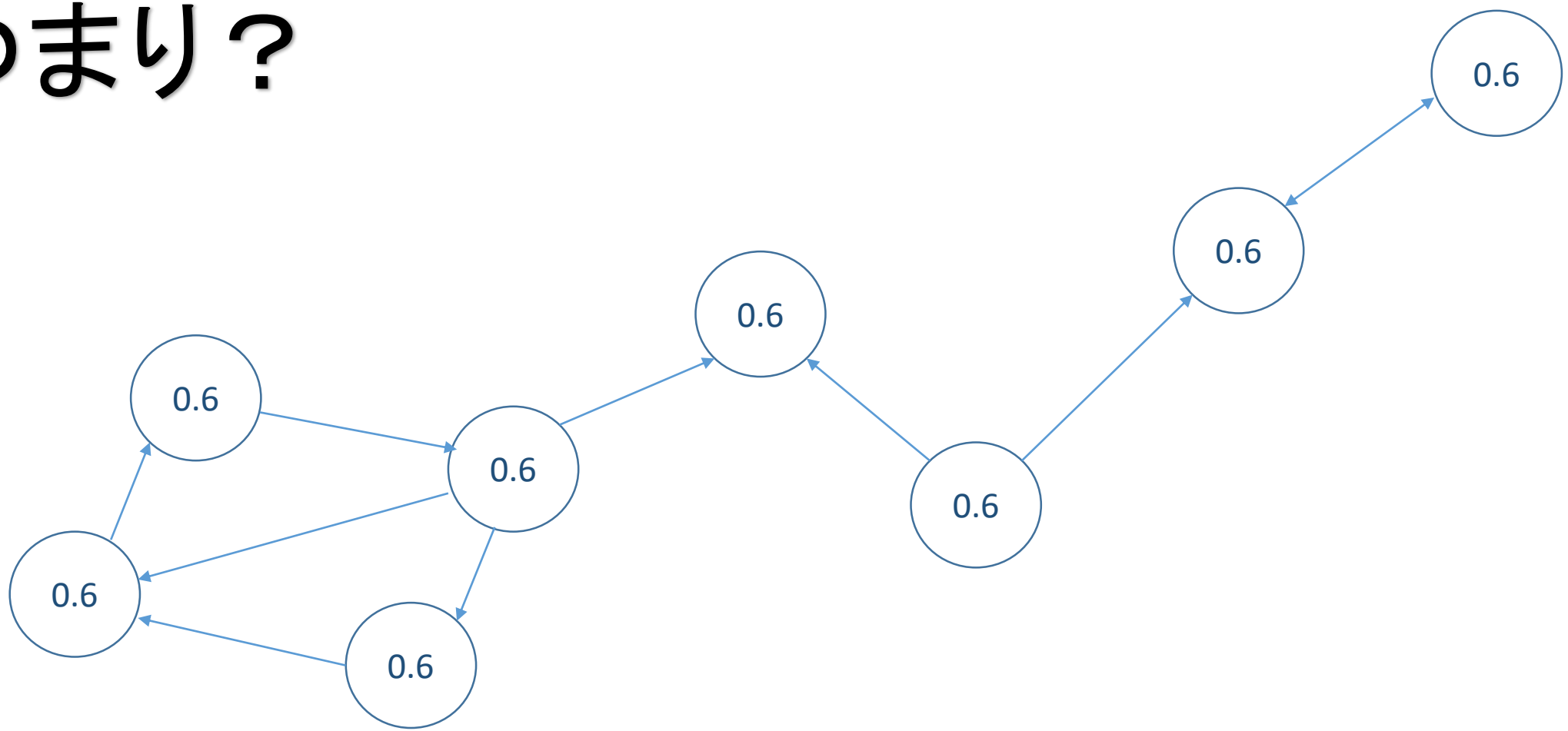
夏合宿の朝は早い(aoj icpc 450)



入次数0のやつが全員おきればいいんじゃね？

夏合宿の朝は早い(aoj icpc 450)

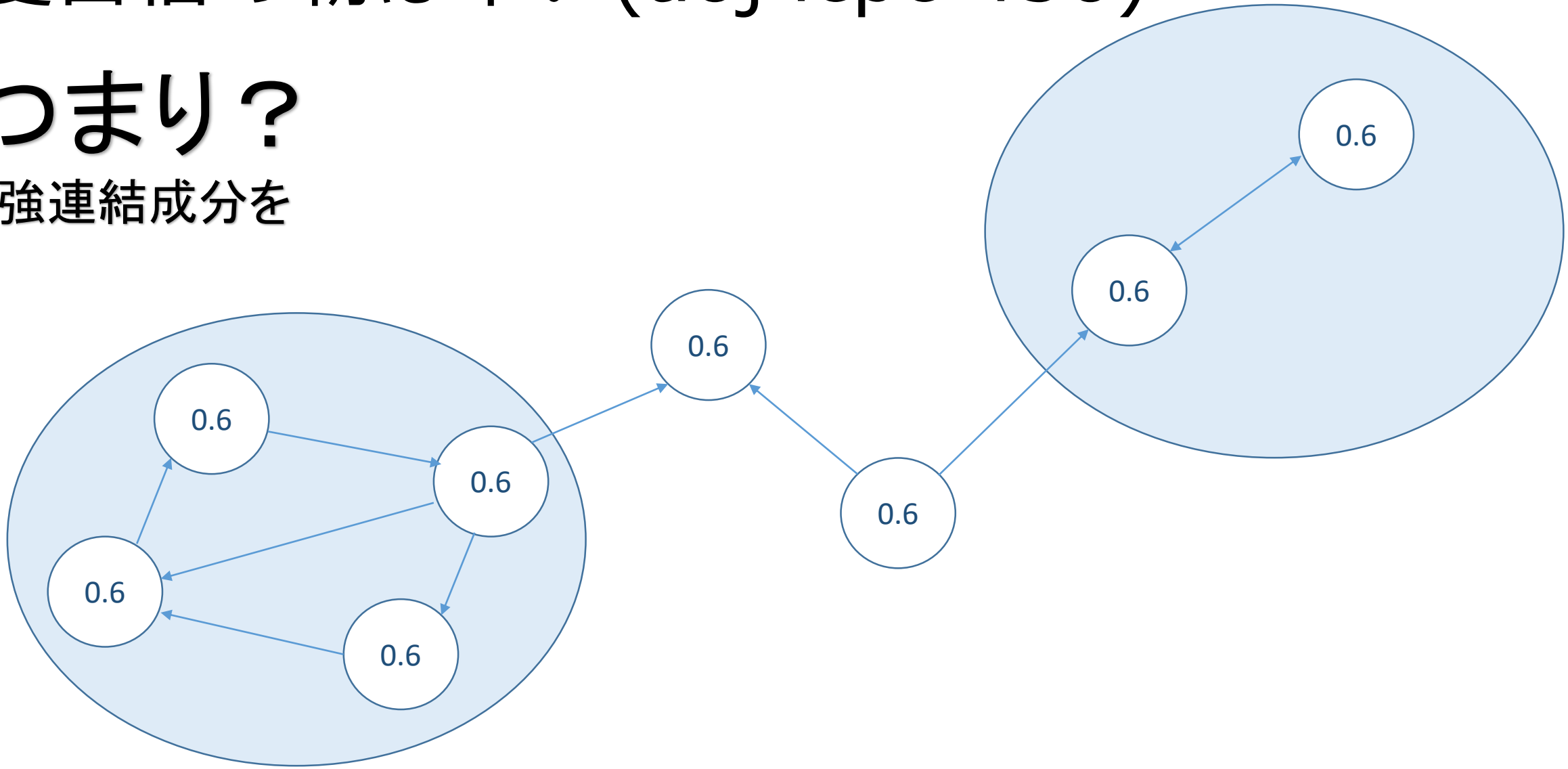
つまり？



夏合宿の朝は早い(aoj icpc 450)

つまり？

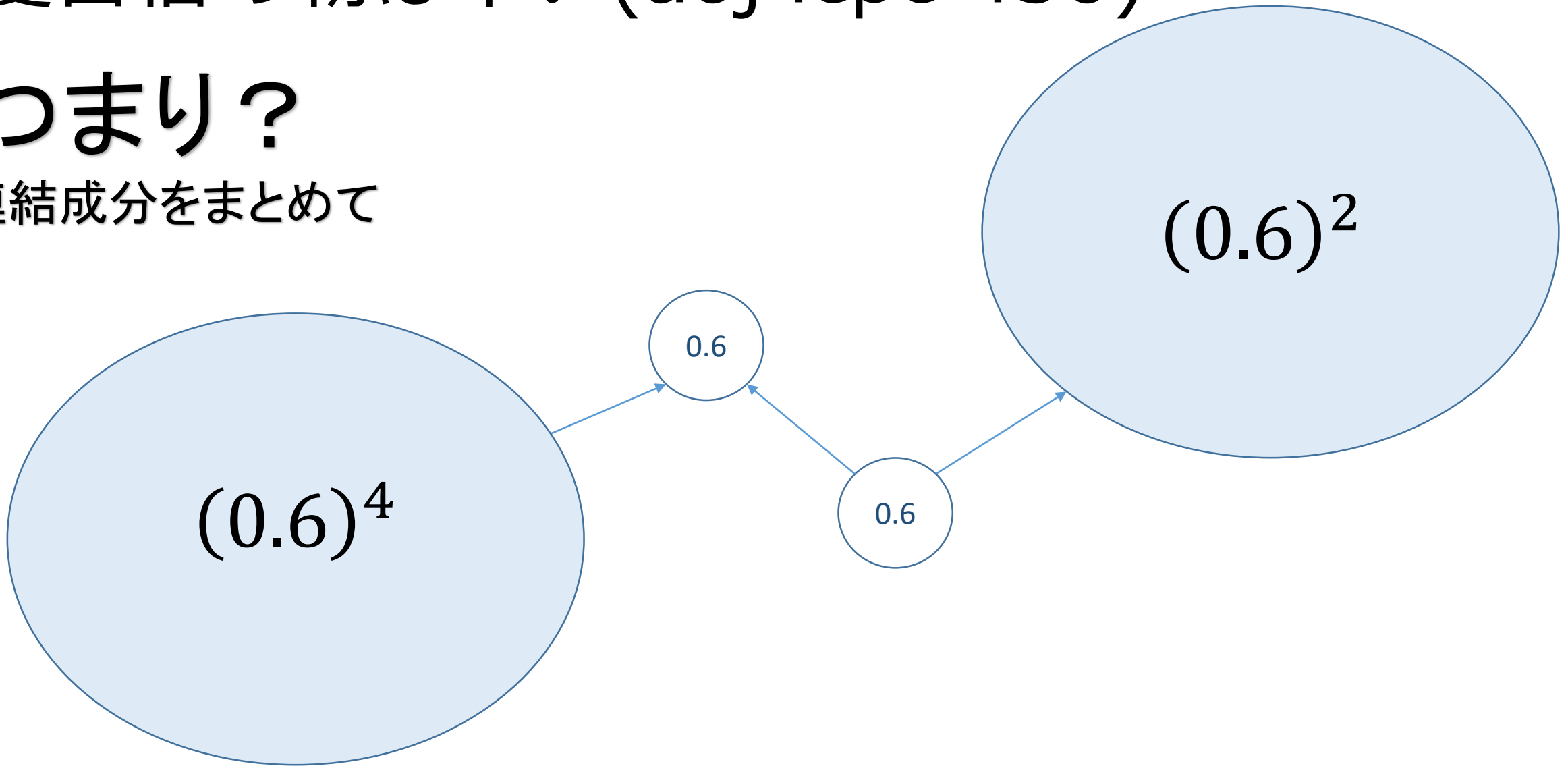
強連結成分を



夏合宿の朝は早い(aoj icpc 450)

つまり？

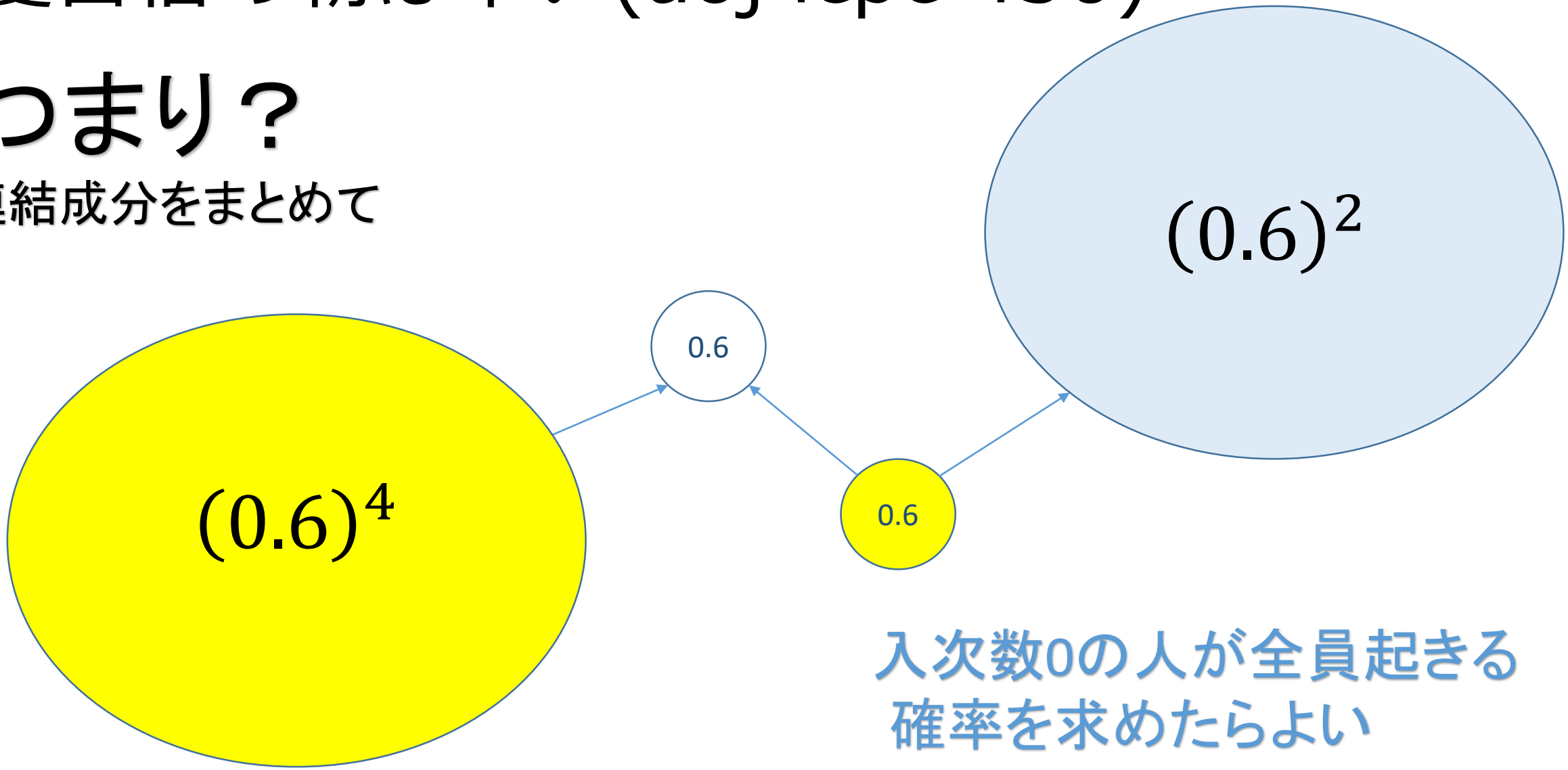
強連結成分をまとめて



夏合宿の朝は早い(aoj icpc 450)

つまり？

強連結成分をまとめて

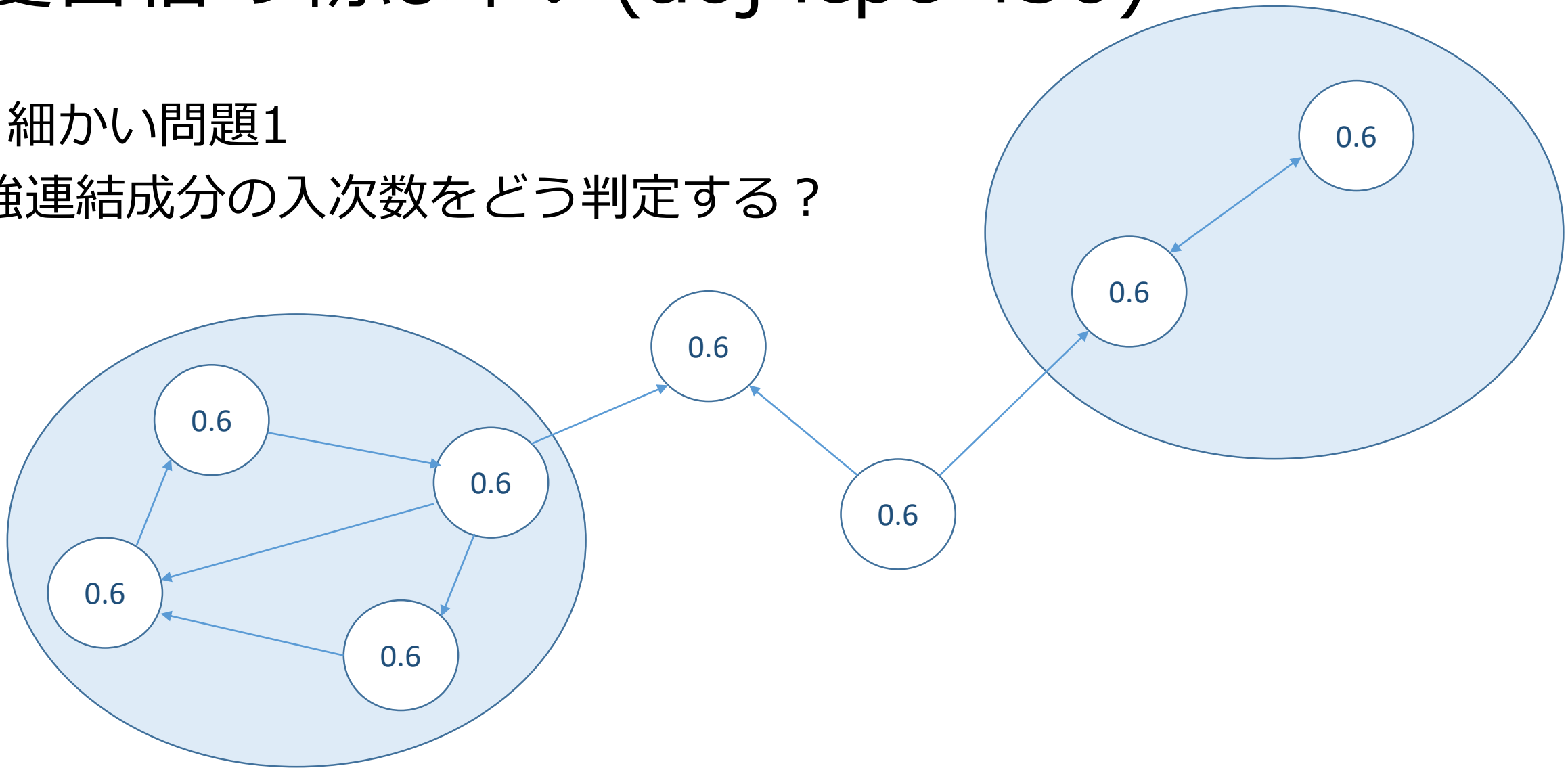


入次数0の人が全員起きる
確率を求めたらよい

夏合宿の朝は早い(aoj icpc 450)

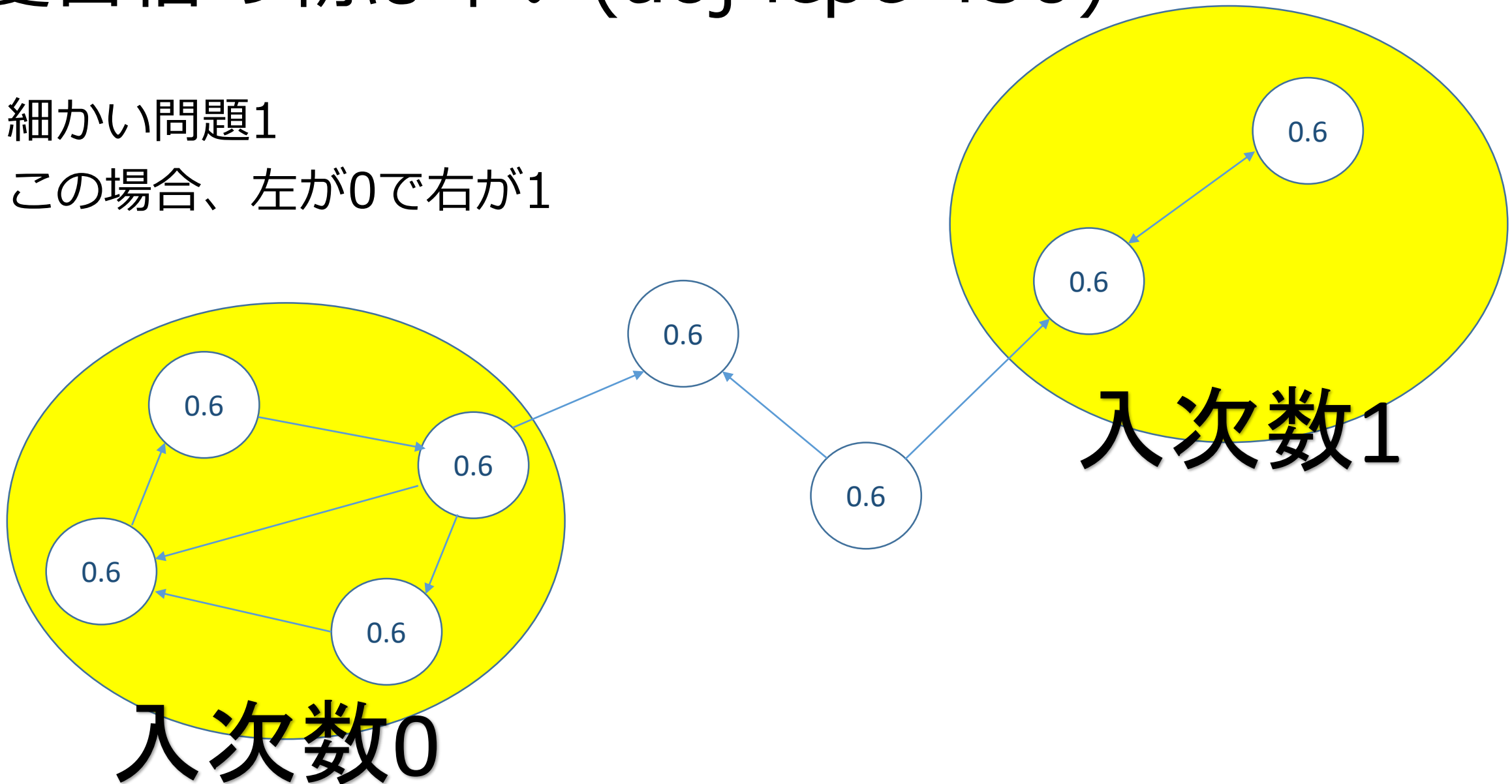
- 細かい問題1

強連結成分の入次数をどう判定する？



夏合宿の朝は早い(aoj icpc 450)

- 細かい問題1
- この場合、左が0で右が1

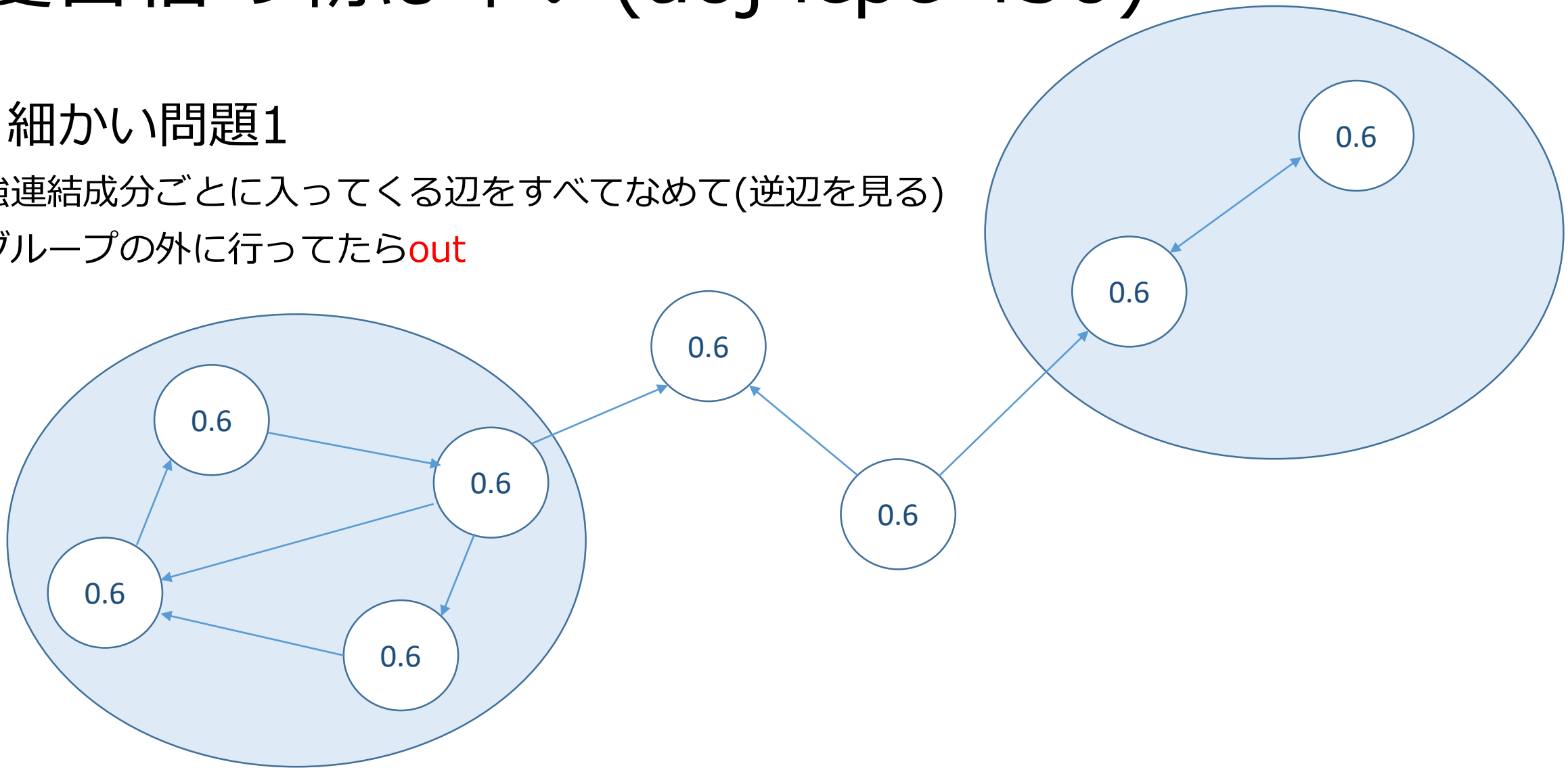


夏合宿の朝は早い(aoj icpc 450)

- 細かい問題1

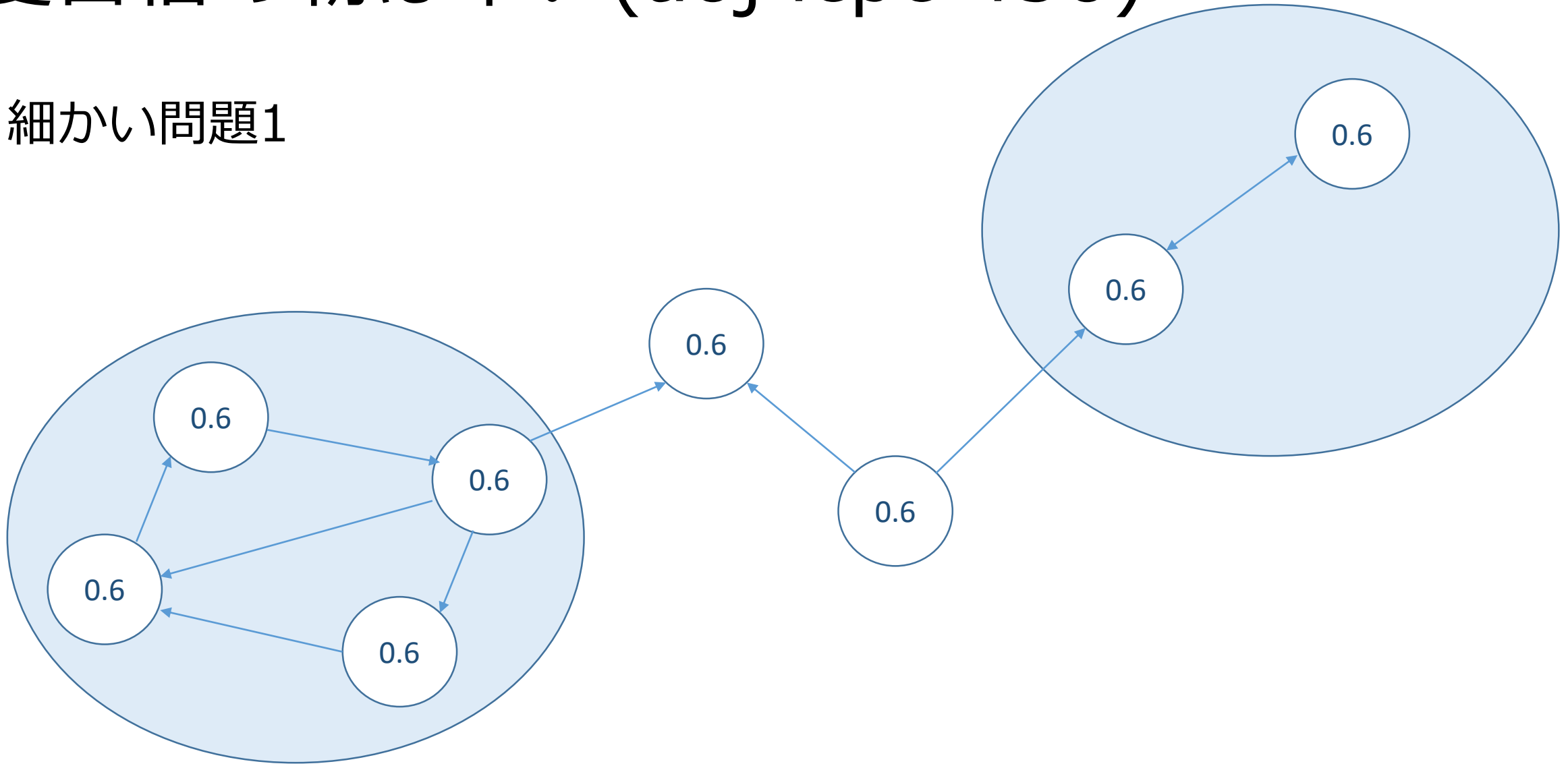
強連結成分ごとに入ってくる辺をすべてなめて(逆辺を見る)

グループの外に行ってたらout



夏合宿の朝は早い(aoj icpc 450)

- 細かい問題1



```
28    void rdfs(int v, int k){
29        used[v] = true;
30        cmp[v] = k;
31        vG[k].push_back(v);
32        for(int i = 0; i < (int)rG[v].size(); i++){
33            if(!used[rG[v][i]]) rdfs(rG[v][i], k);
34        }
35    }
```

```cpp
int main(){

    while(cin >> V, V){

        //初期化
        for(int i = 0; i < MAX_V; i++){
            G[i].clear();
            rG[i].clear();
            vG[i].clear();
        }

        //入力受け取り
        vector<double> p(V);
        for(int i = 0; i < V; i++){
            int m;
            cin >> p[i] >> m;
            for(int j = 0; j < m; j++){
                int to; cin >> to; to--;
                add_edge(i, to);
            }
        }

        int k = scc();

        double ans = 1.0;
        for(int i = 0; i < k; i++){
            bool flag = true;                //入次数が0であるか
            double res = 1.0;

            //強連結成分をすべてなめる
            for(int j = 0; j < (int)vG[i].size(); j++){
                res *= p[vG[i][j]];

                //その頂点は入り次数0ですか？(逆辺をすべてなめる)
                for(int l = 0; l < (int)rG[vG[i][j]].size(); l++){
                    flag &= (cmp[vG[i][j]] == cmp[rG[vG[i][j]][l]]);
                }
            }
            if(flag){
                ans *= (1.0 - res);
            }
        }
        printf("%.10lf\n", ans);
    }
    return 0;
}
```

問題集

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_4_B
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_3_C

https://tenka1-2016-quala.contest.atcoder.jp/tasks/tenka1_2016_qualA_c
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=0519
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2748
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2222
https://tdpc.contest.atcoder.jp/tasks/tdpc_graph