
Islands
Editorial : blackyuki


おにが島
「ようこそ、おときの国へ。」
自然をきれいにしましょう。
鬼ヶ島
ようこそ
おときの国へ
遊歩道下り口

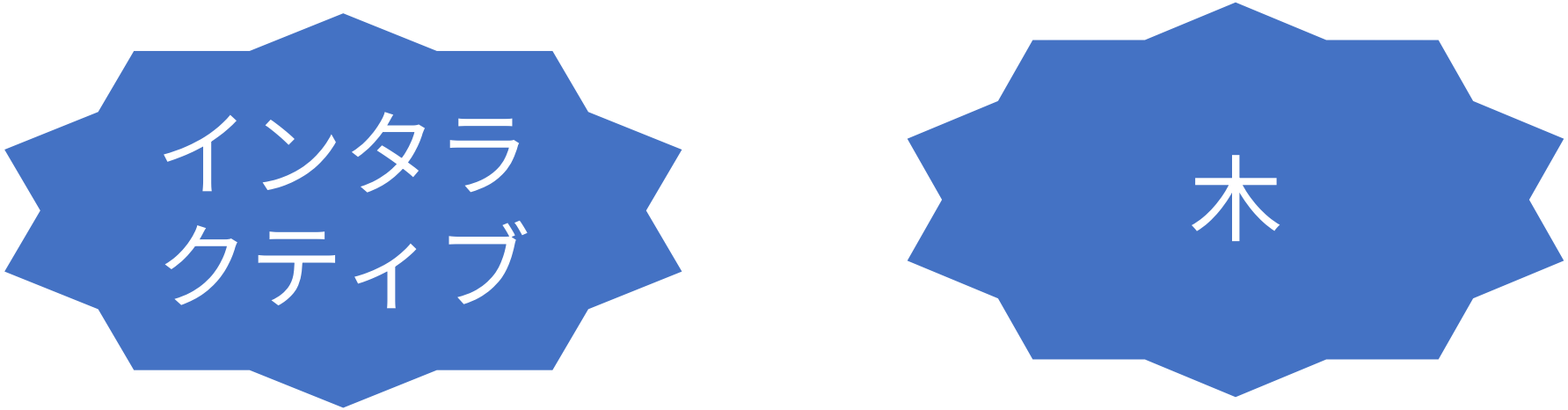
問題概要

問題概要

- N 頂点の木がある
- 木の情報は与えられていない
- 木の形を求めたい
- 「頂点 i から k 番目に近い頂点の番号は何か？」を質問できる

・・・同じ距離の頂点は番号の大小で決まる

問題の性質



インタラクティブにおける典型考察

- 色んな解法を試して上手くいくものを選ぶ
- 一つの解法に固執するのはもったいない
- 普通のBatch課題における典型アルゴリズムのシミュレーション

・・・BFS, DFS, Dijkstra, など

木を特定する問題における典型考察

A．頂点 1, 2, …, N の順に追加していく

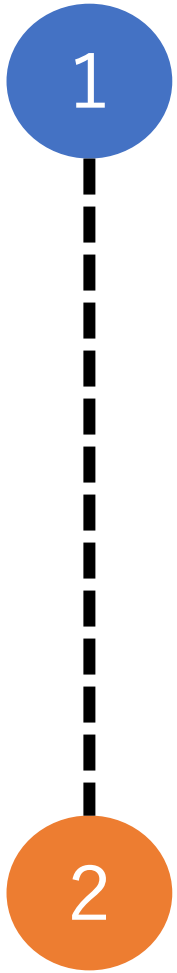
B．頂点 1 から近い順に探索していく

C．葉から削っていく

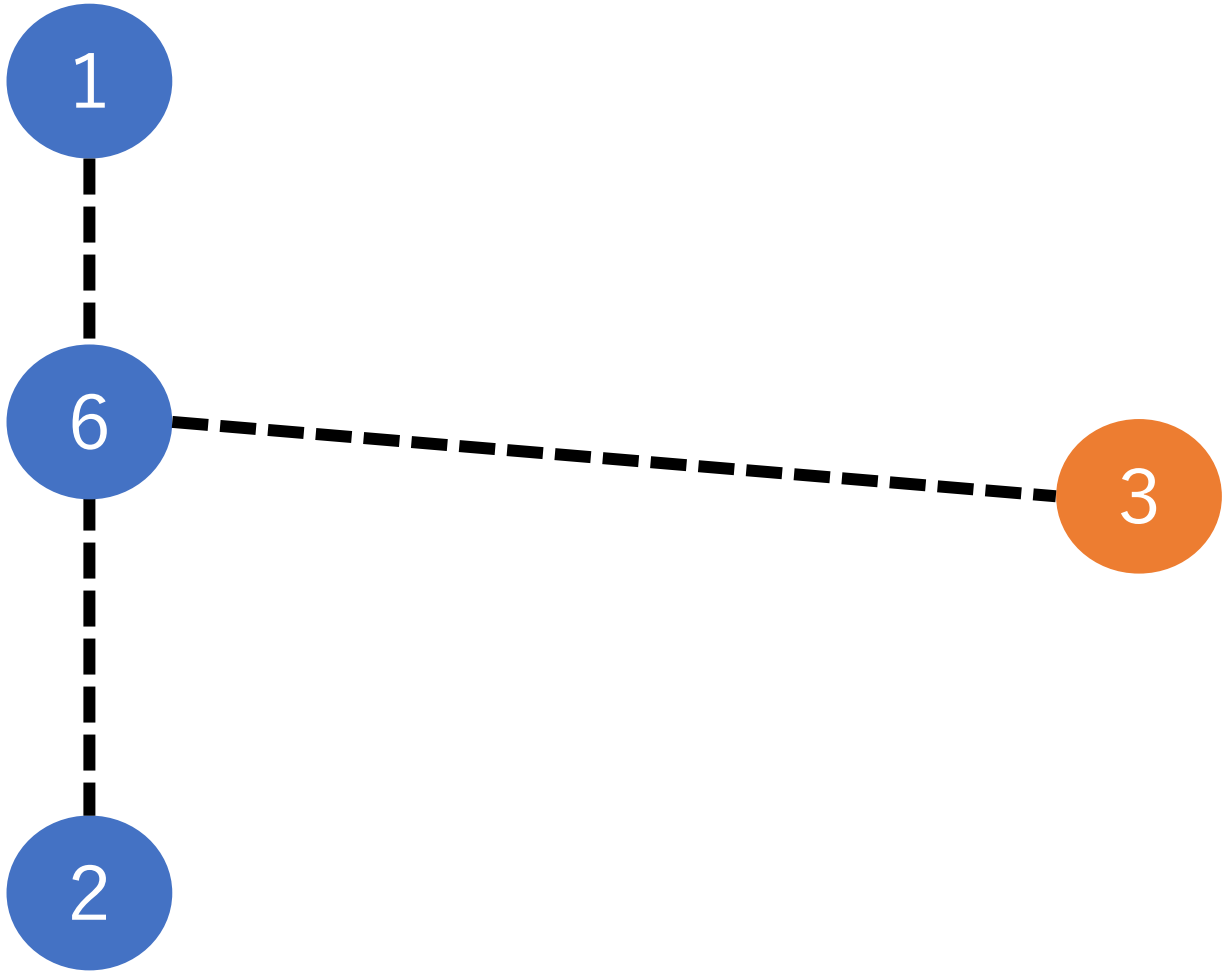
D．直径の利用

A．頂点 1, 2, …, N の順に追加していく

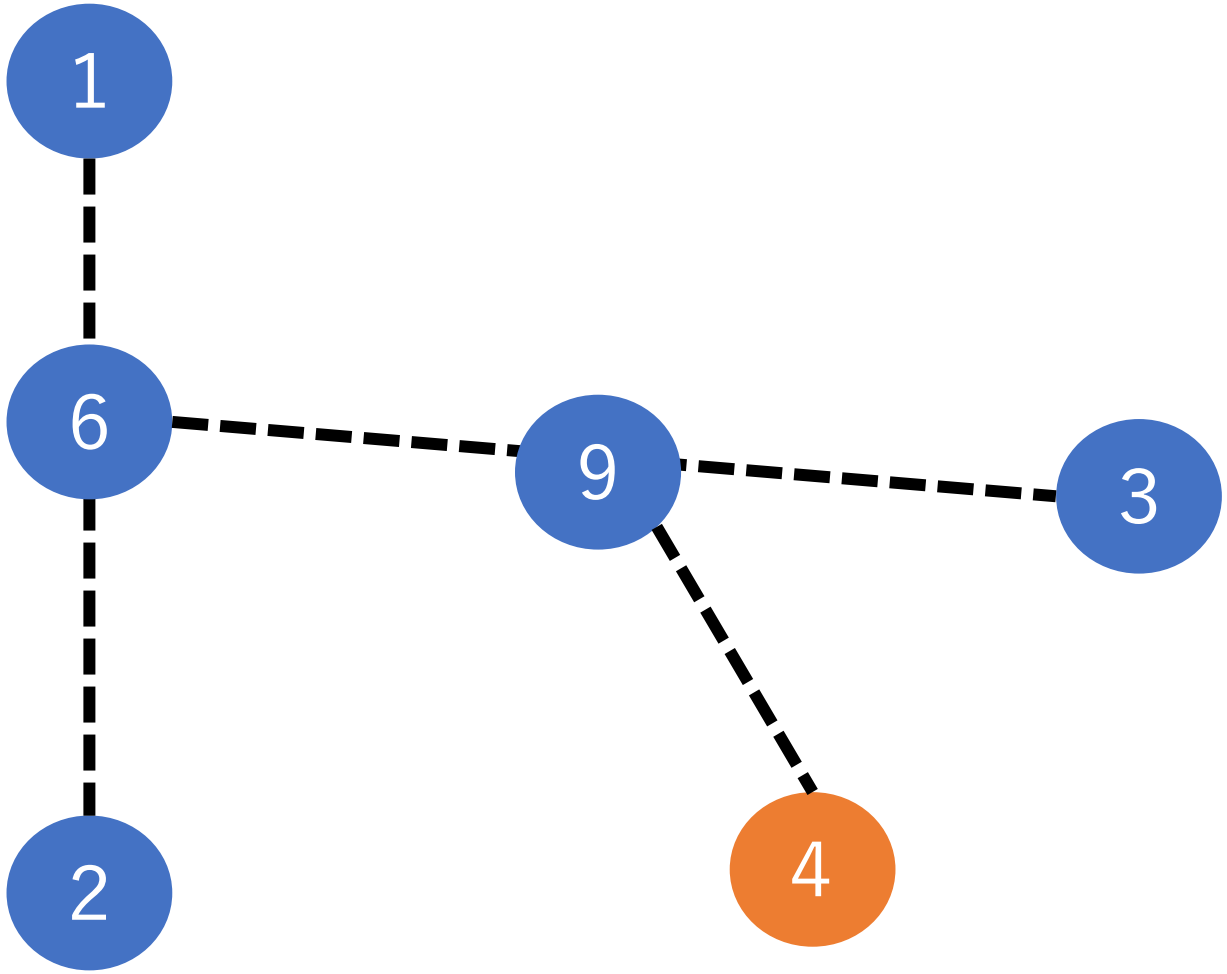
A. 頂点 1, 2, …, N の順に追加していく



A. 頂点 1, 2, …, N の順に追加していく



A. 頂点 1, 2, …, N の順に追加していく

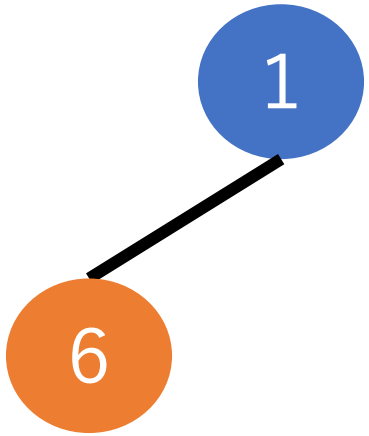


B．頂点１から近い順に探索していく

BFS（幅優先探索）

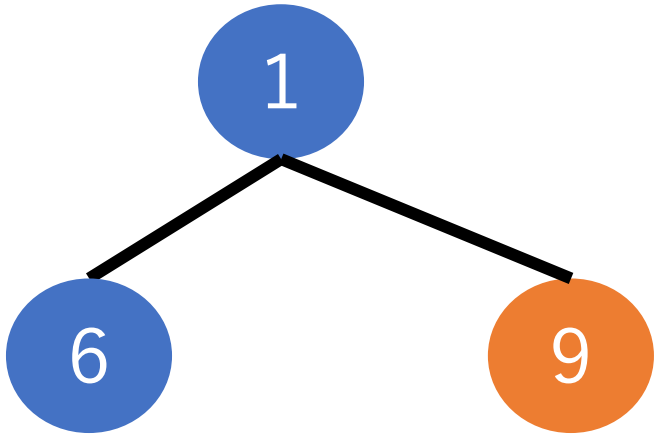
B．頂点１から近い順に探索していく

BFS（幅優先探索）



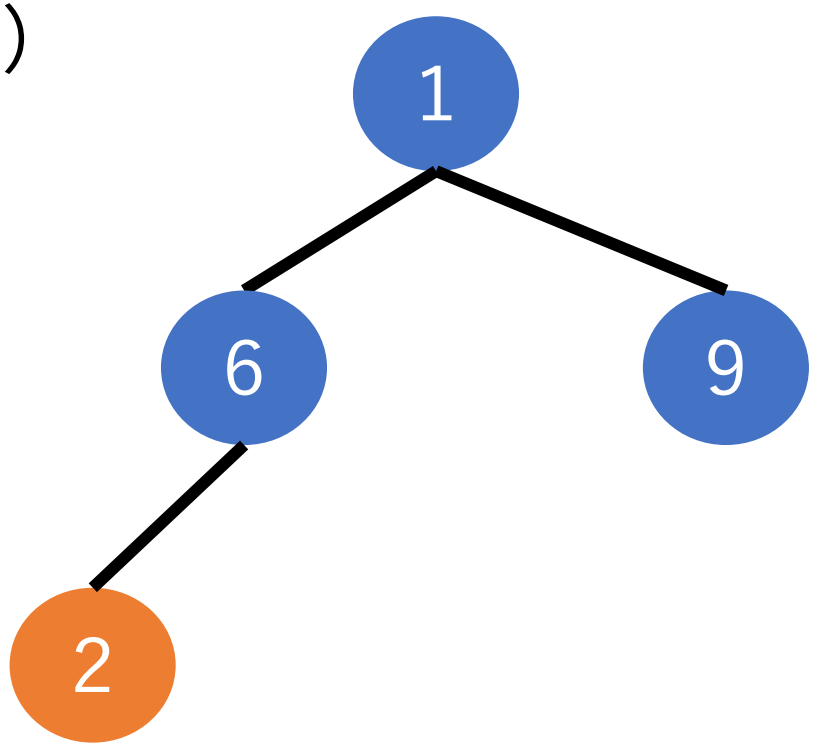
B．頂点１から近い順に探索していく

BFS（幅優先探索）



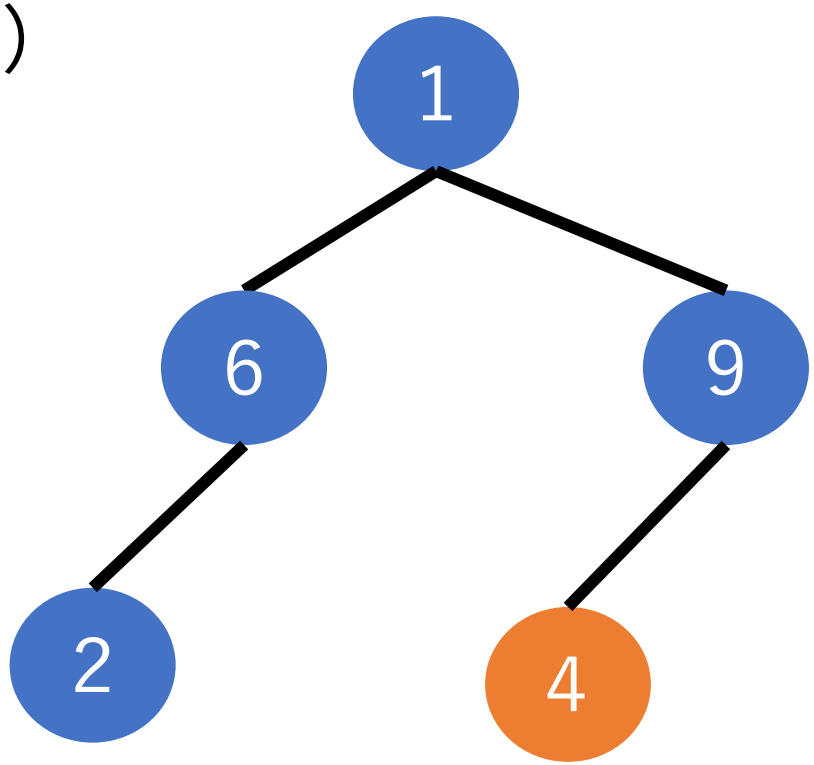
B．頂点１から近い順に探索していく

BFS（幅優先探索）



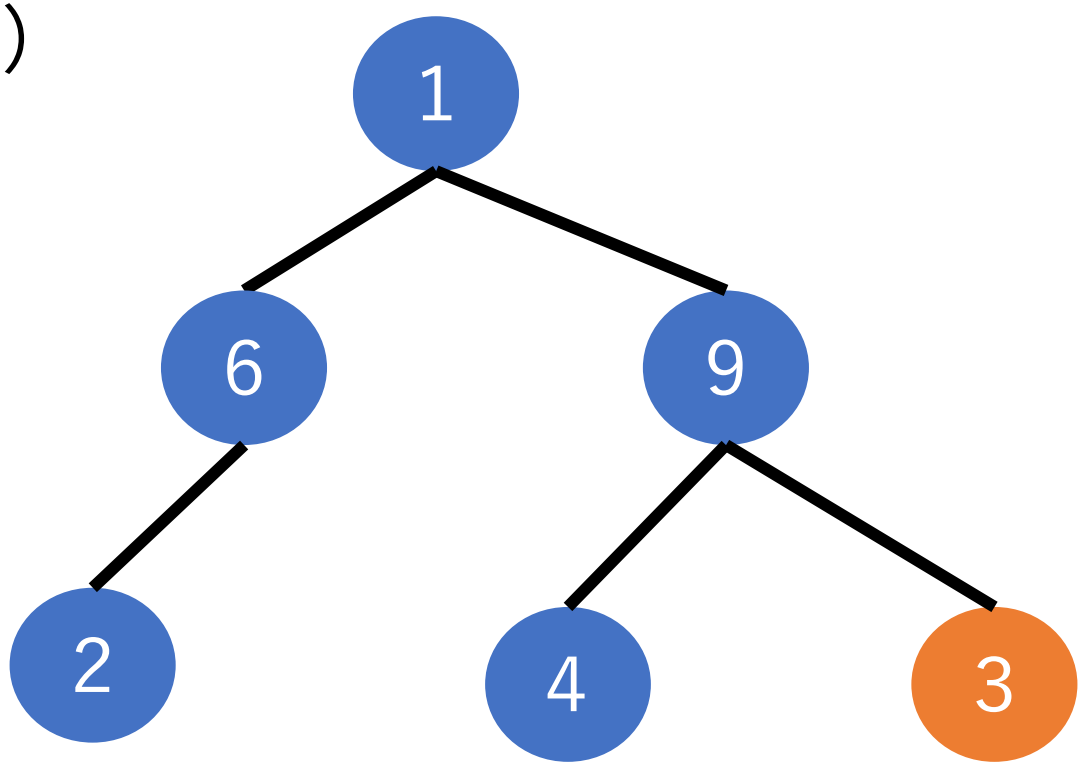
B．頂点１から近い順に探索していく

BFS（幅優先探索）



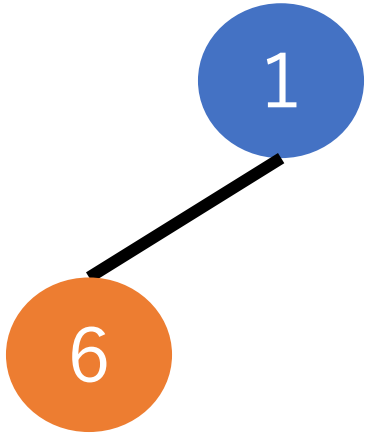
B．頂点１から近い順に探索していく

BFS（幅優先探索）



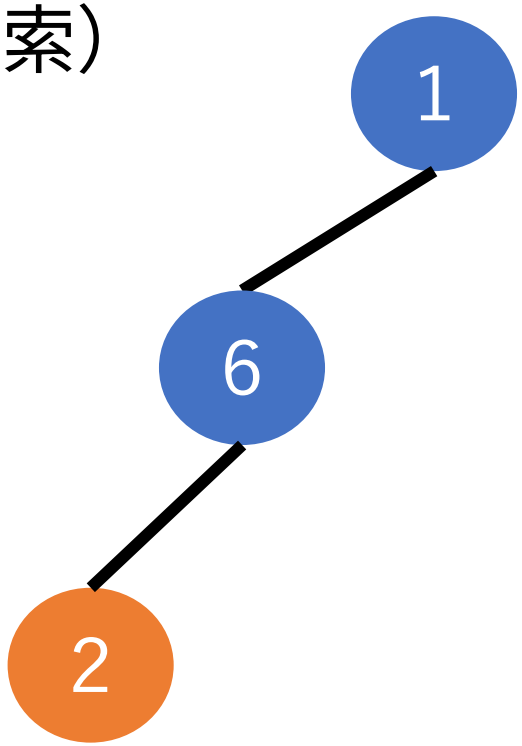
B．頂点１から近い順に探索していく

DFS（深さ優先探索）



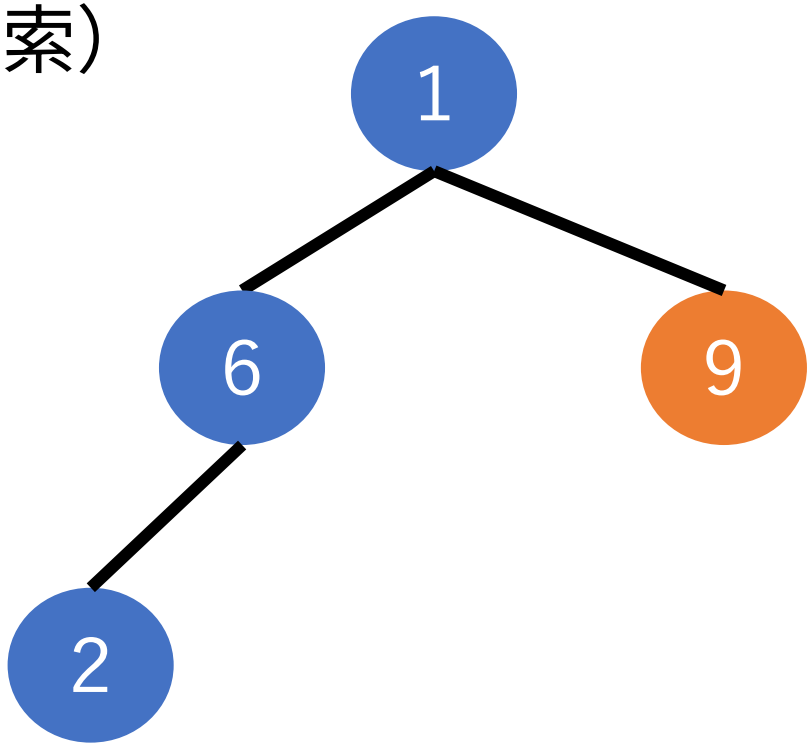
B．頂点１から近い順に探索していく

DFS（深さ優先探索）



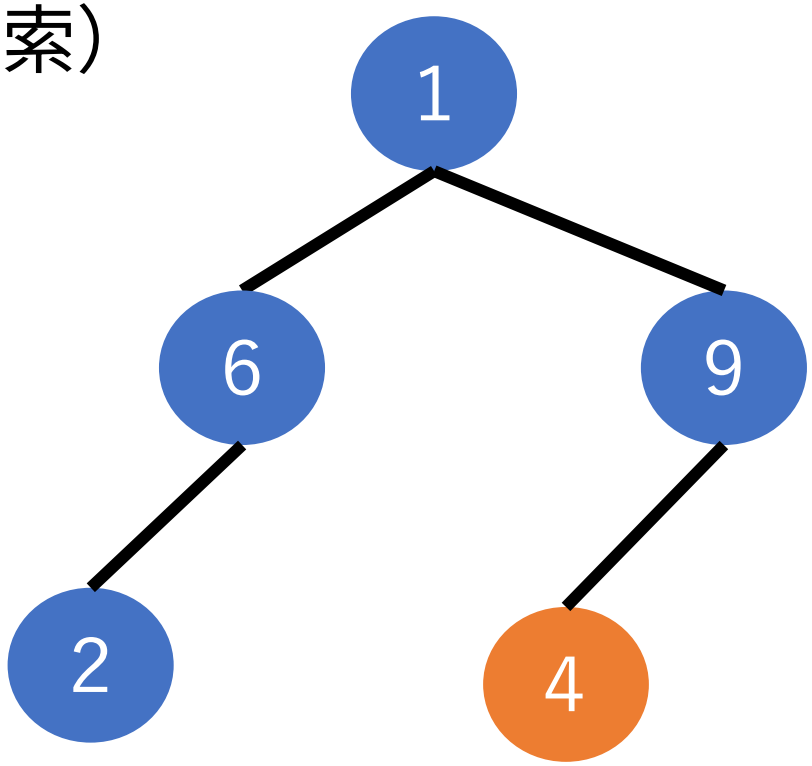
B．頂点１から近い順に探索していく

DFS（深さ優先探索）



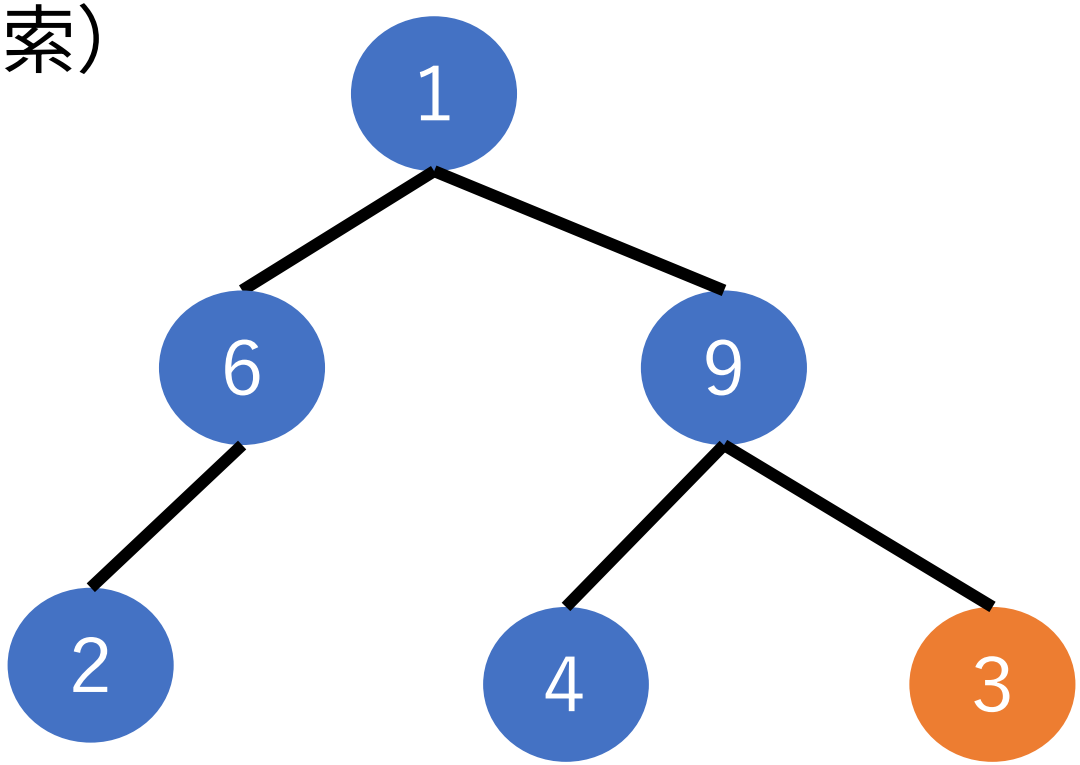
B．頂点１から近い順に探索していく

DFS（深さ優先探索）



B．頂点１から近い順に探索していく

DFS（深さ優先探索）

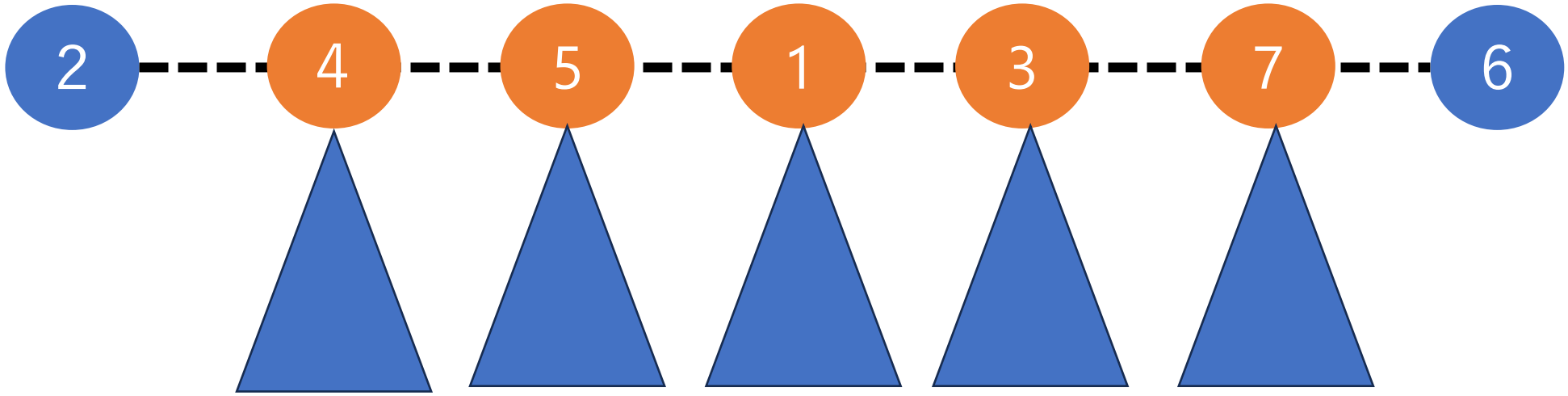


C．葉から削っていく

まず何らかの方法で1つの葉とその接続先を見つける

→その頂点を取り除いて、頂点数が1つ少ない問題に帰着させる

D．直径の利用

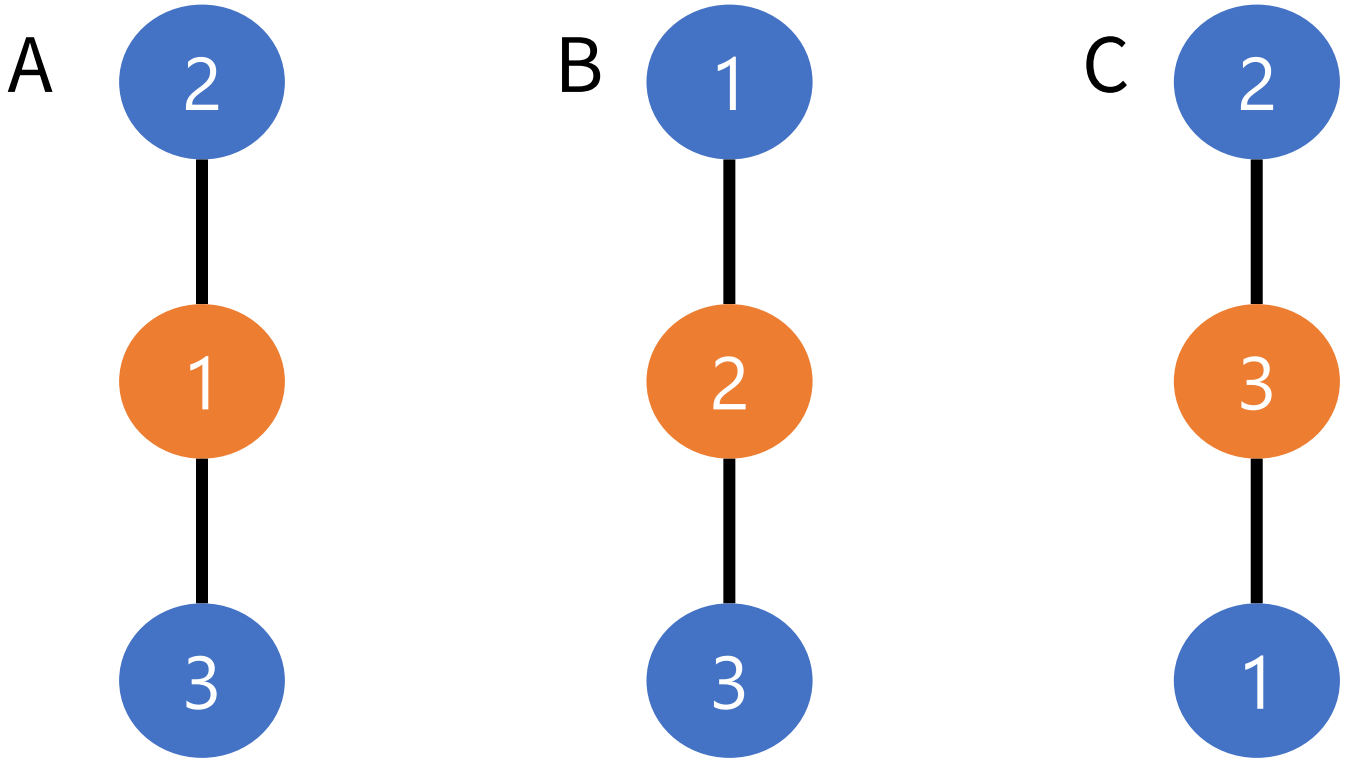


小課題１

N=3

気づき

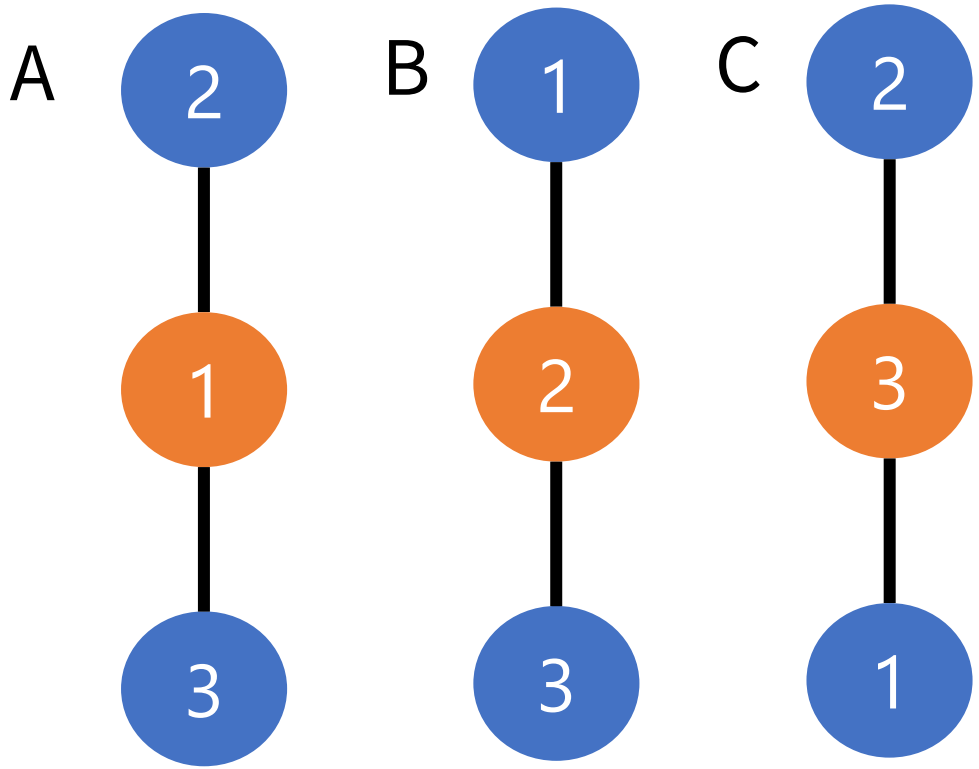
3頂点の木は種類が少なく、簡単に列挙できる



気づき

頂点 i から最も近い距離の頂点を聞く

	i=1	i=2	i=3
A	2	1	1
B	2	1	2
C	3	3	1



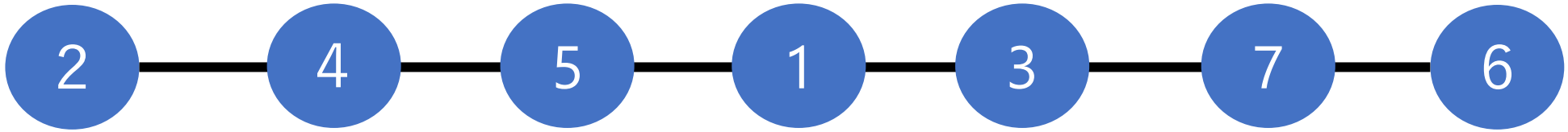
```
void solve(int N, int L) {
    // N = 3, L = N^2
    if (query(1, 1) == 3) {
        answer(1, 3);
        answer(2, 3);
    }
    else {
        answer(1, 2);
        answer(query(3, 1), 3);
    }
}
```

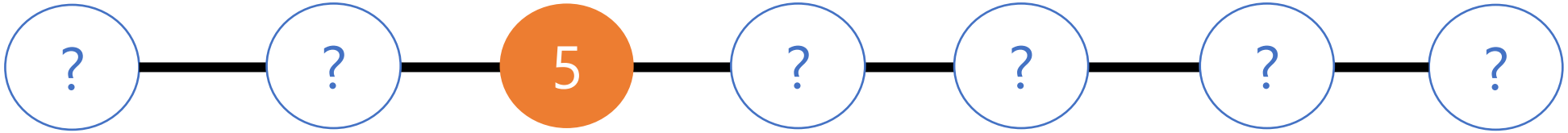
→小課題1 AC!

小課題2,3

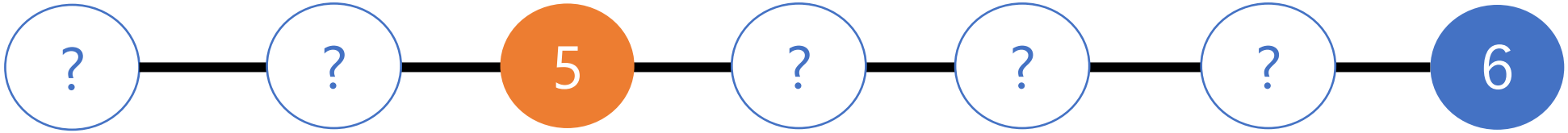
パスグラフ

パスグラフ

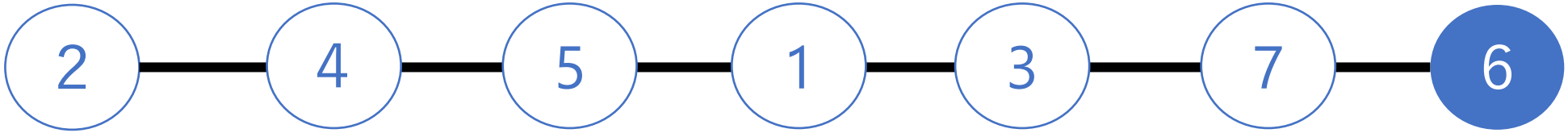




適当な頂点から最も遠い頂点を聞く



適当な頂点から最も遠い頂点を聞く



その頂点から近い頂点を順に聞く

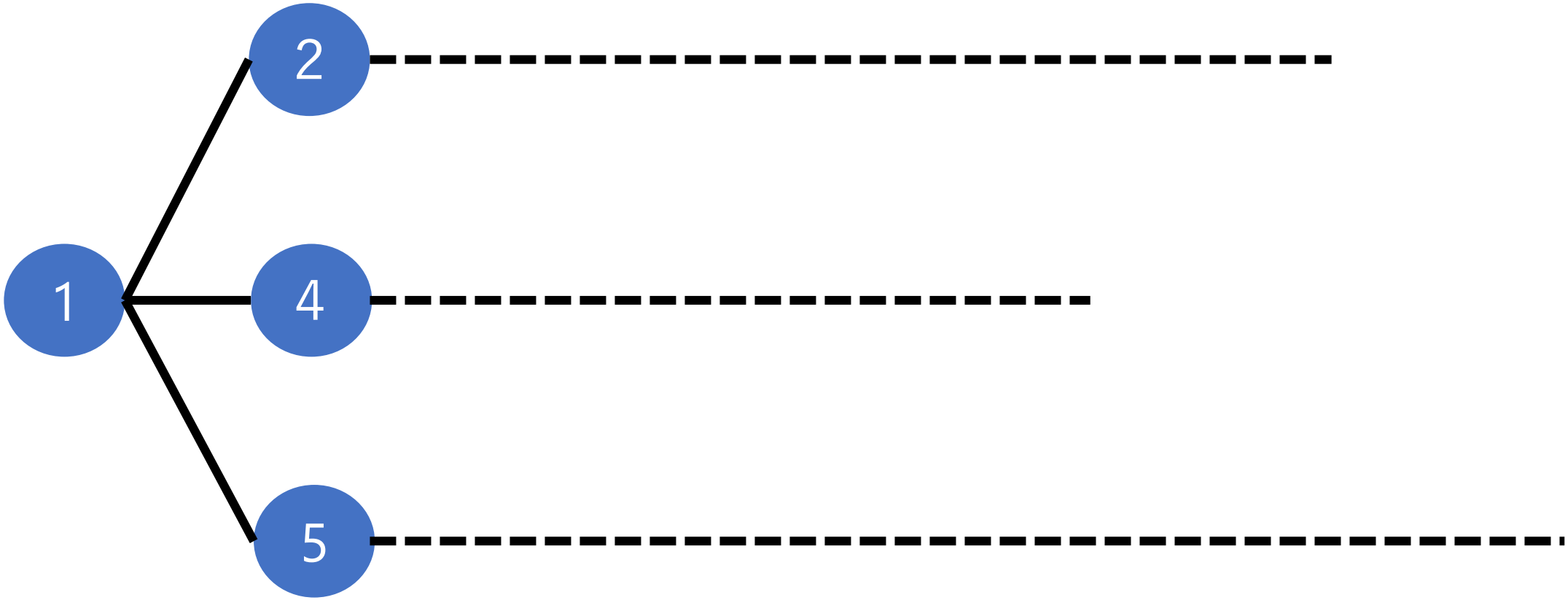
```
void solve(int N, int L){
    // path, L = 2N
    int a = query(1, N - 1), b;
    b = a;
    for(int i = 0; i < N - 1; i++){
        int c = query(a, i + 1);
        answer(min(b, c), max(b, c));
        b = c;
    }
}
```

→小課題2,3 AC!

小課題4,5

特殊な形のグラフ

特殊な形のグラフ



木を特定する問題における典型考察

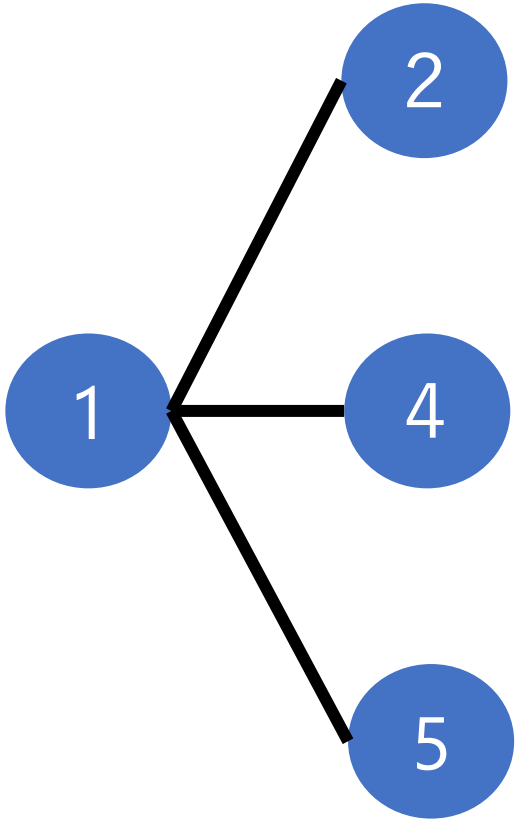
A．頂点 1, 2, …, N の順に追加していく

B．頂点 1 から近い順に探索していく

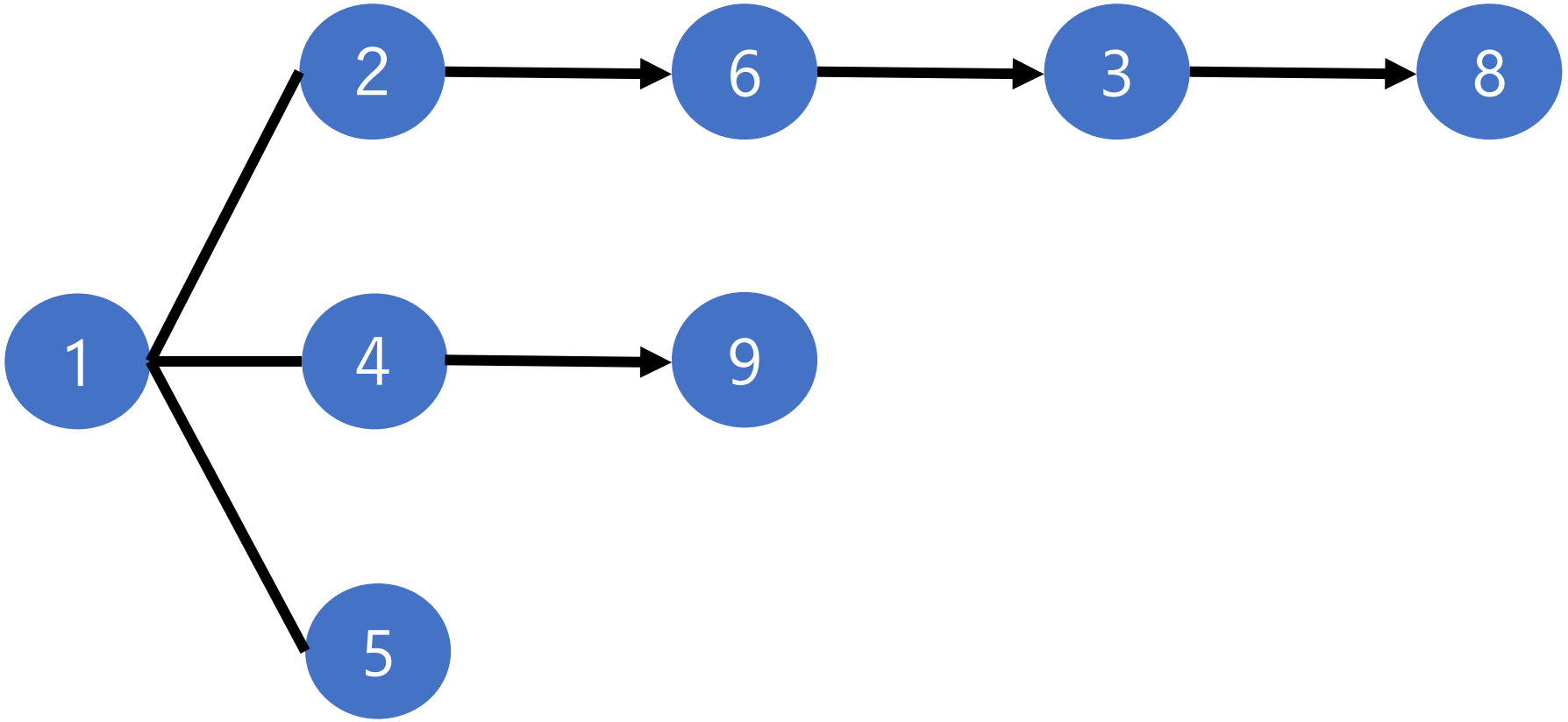
C．葉から削っていく

D．直径の利用

最初に頂点1に接続する3頂点の番号を求める



DFSする（今いる頂点から近い2頂点を聞く）

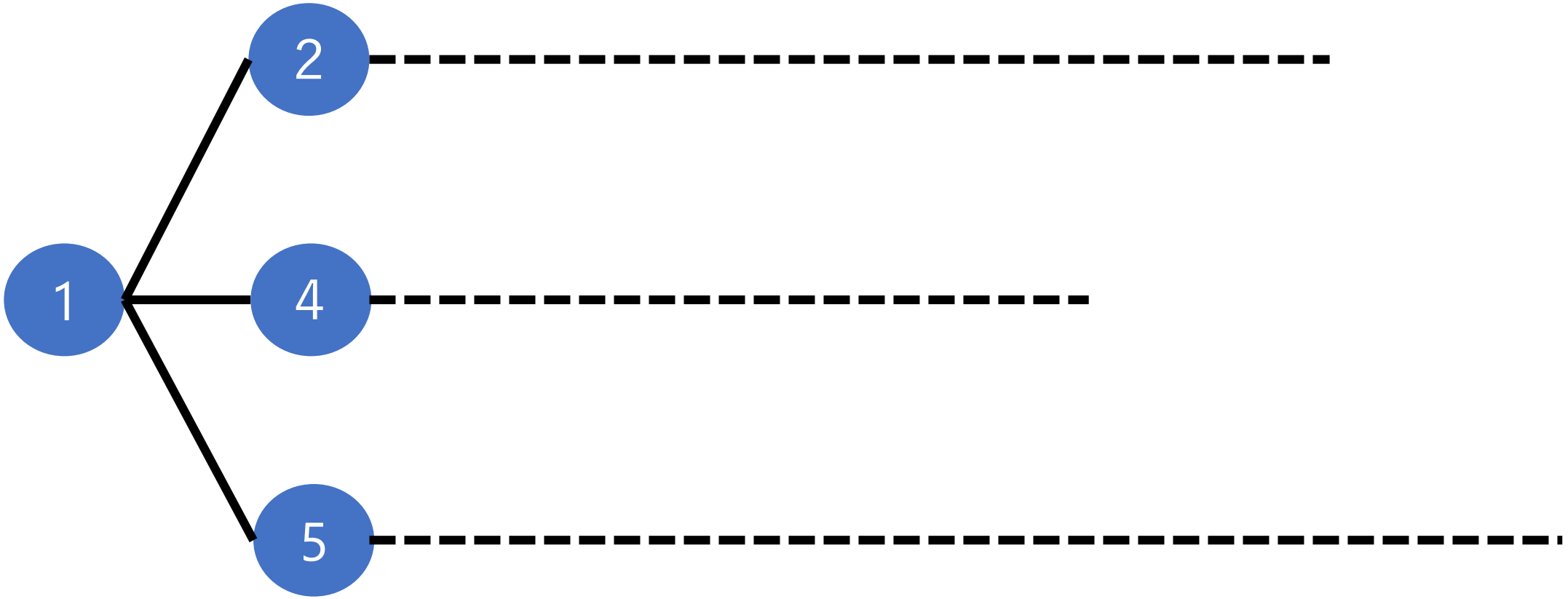


```
void solve(int N, int L){
    vector<int> ok(N + 1, 0);
    vector<int> v = {query(1, 1), query(1, 2), query(1, 3)};
    for(int i = 0; i < 3; i++){
        answer(1, v[i]);
    }
    ok[1] = 1;

    for(int i = 0; i < 3; i++){
        while(true){
            int b = query(v[i], 1);
            int c = query(v[i], 2);
            if(ok[b] == 0){
                ok[v[i]] = 1;
                answer(v[i], b);
                v[i] = b;
            }
            else if(ok[c] == 0){
                ok[v[i]] = 1;
                answer(v[i], c);
                v[i] = c;
            }
            else{
                break;
            }
        }
    }
}
```

→小課題4,5 AC!

(別解)



木を特定する問題における典型考察

A．頂点 1, 2, …, N の順に追加していく

B．頂点 1 から近い順に探索していく

C．葉から削っていく

D．直径の利用

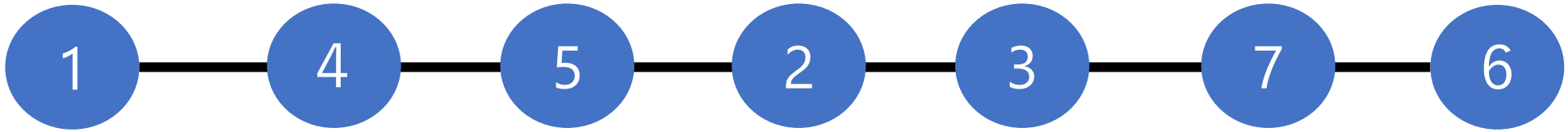
頂点1から最も距離の遠い頂点を聞く

→その頂点から近い順に決めていく

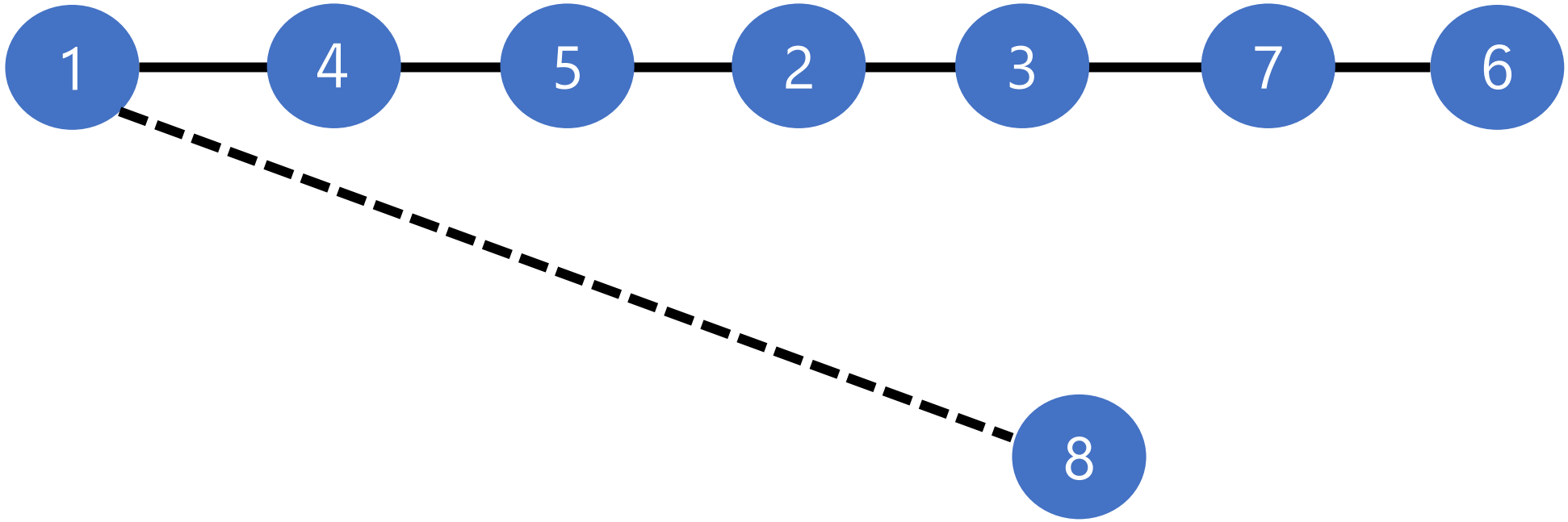
頂点1から最も距離の遠い頂点を聞く

→その頂点から近い順に決めていく

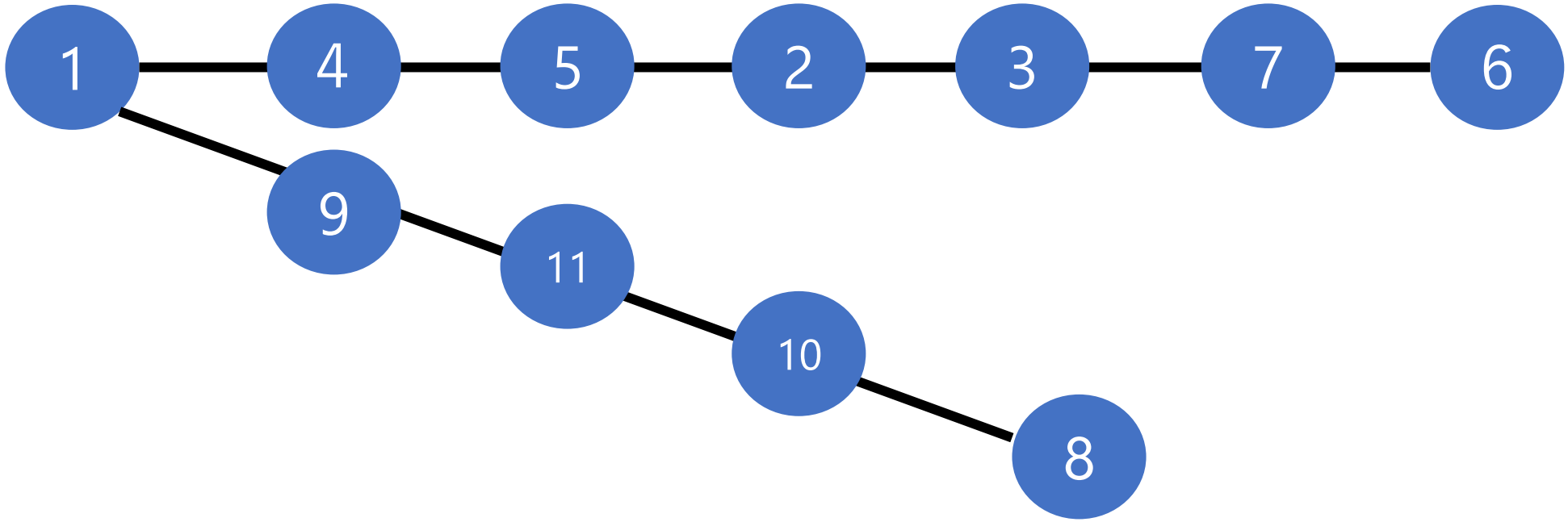
→頂点1が来たら打ち切り



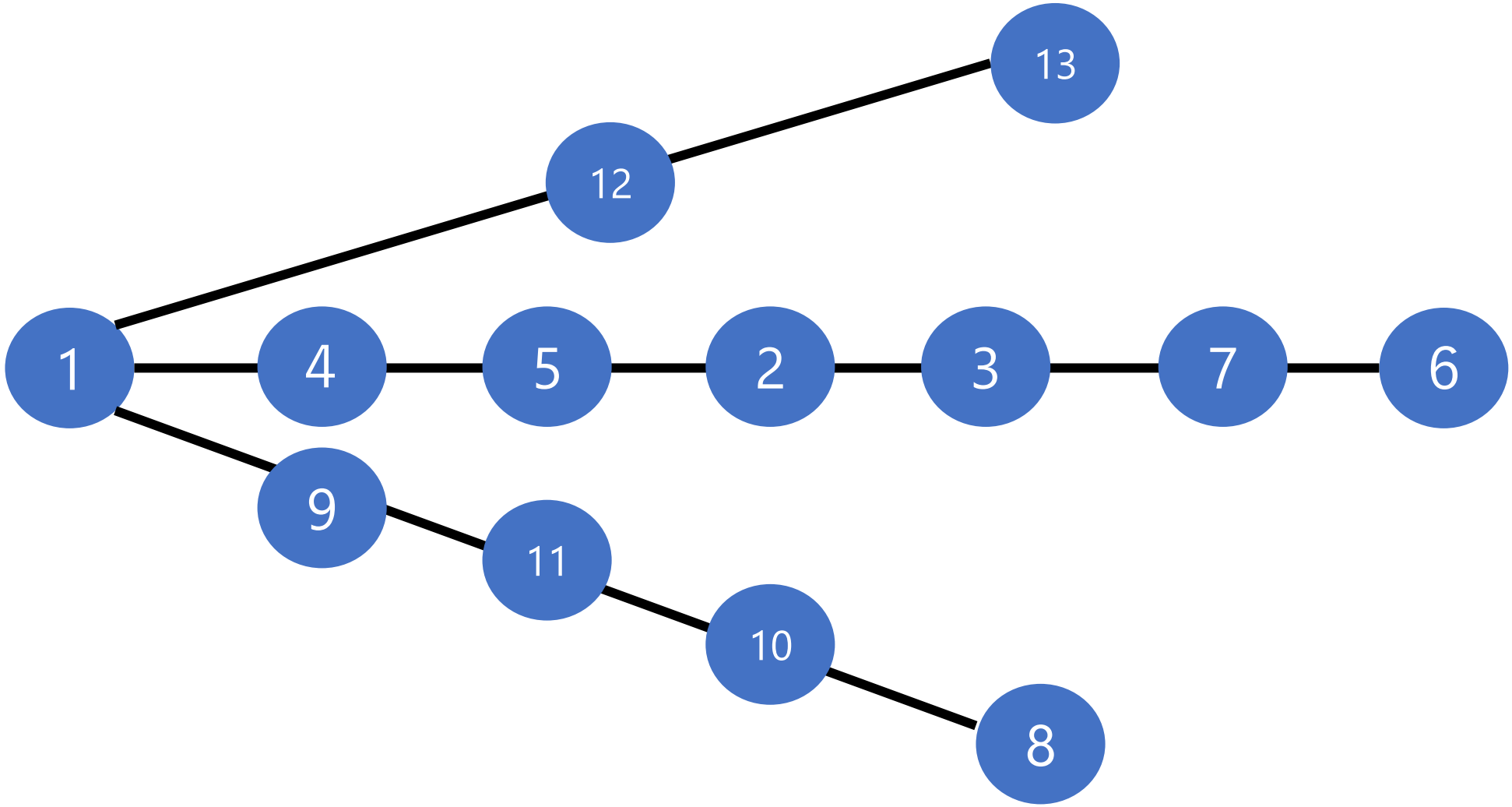
頂点6から最も遠い頂点を聞く



頂点8からも同様に聞いていく(頂点1で打ち切り)



残りの頂点は１からの近い順につなげる



小課題6

L=N^2

木を特定する問題における典型考察

A．頂点 1, 2, …, N の順に追加していく

B．頂点 1 から近い順に探索していく

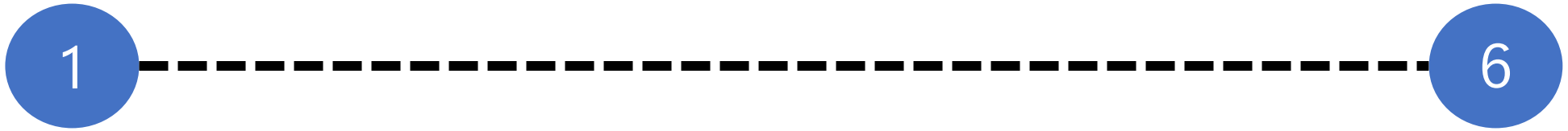
C．葉から削っていく

D．直径の利用

頂点1から最も距離の遠い頂点を聞く

→その頂点は葉であり、その接続先も分かる

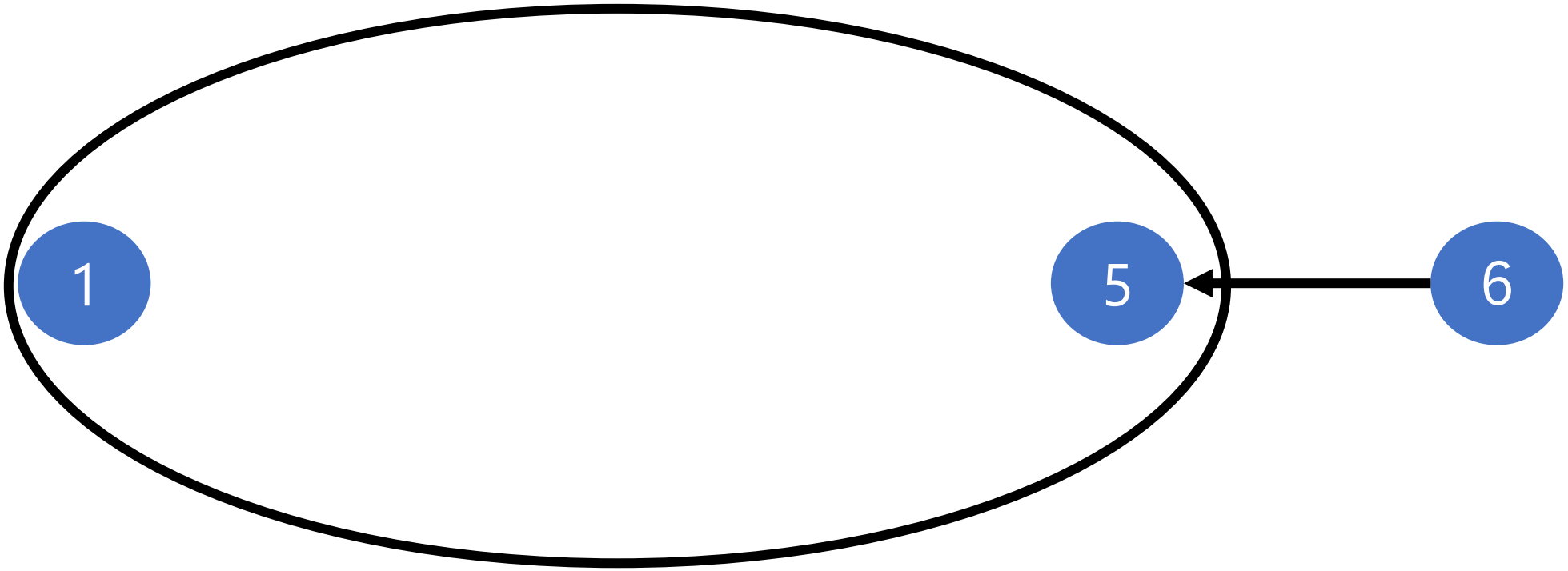
→その葉を無視して残りの木を再帰的に求めればよい



頂点1から最も距離の遠い頂点を聞く

→その頂点は葉であり、その接続先も分かる

→その葉を無視して残りの木を再帰的に求めればよい



```
vector<int> used(N, 0);
int a = Q[0][N - 1];

for(int i = 0; i < N - 1; i++){
    for(int j = 1; j < N; j++){
        int b = Q[a][j];
        if(!used[b]){
            used[a] = 1;
            answer(a + 1, b + 1);
            for(int k = N - 1; k >= 0; k--){
                int c = Q[b][k];
                if(!used[c]){
                    a = c;
                    break;
                }
            }
            break;
        }
    }
}
```

→小課題6 AC!

小課題7,8

L=3N,2N

木を特定する問題における典型考察

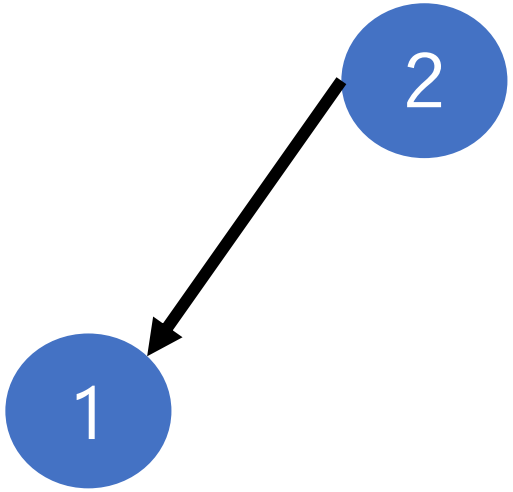
A．頂点 1, 2, …, N の順に追加していく

B．頂点 1 から近い順に探索していく

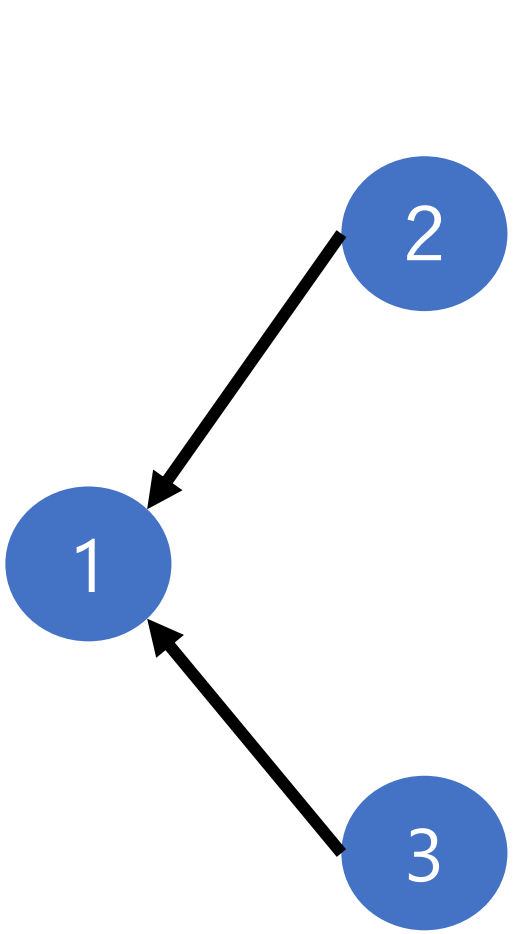
C．葉から削っていく

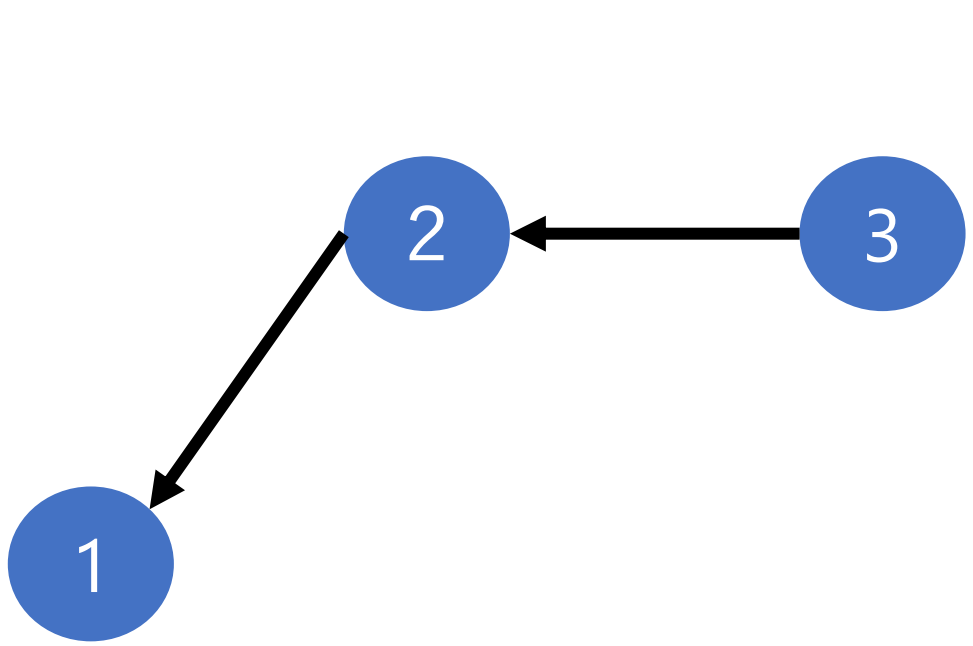
D．直径の利用

頂点1から最も近い頂点をつなげる

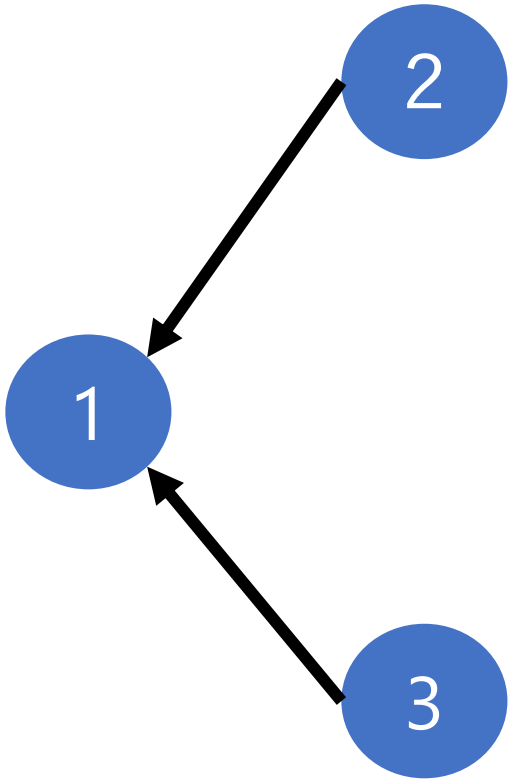


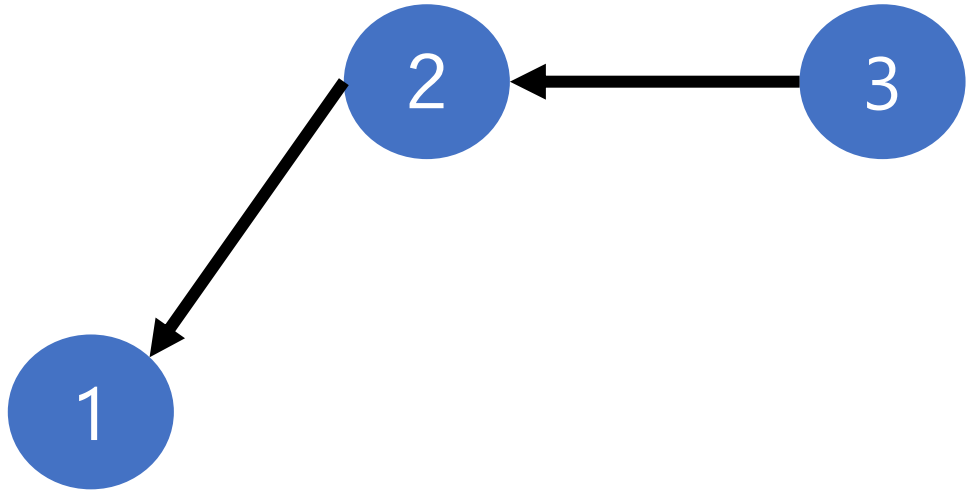
頂点1から次に近い頂点をつなげる





(どっちか分からない)





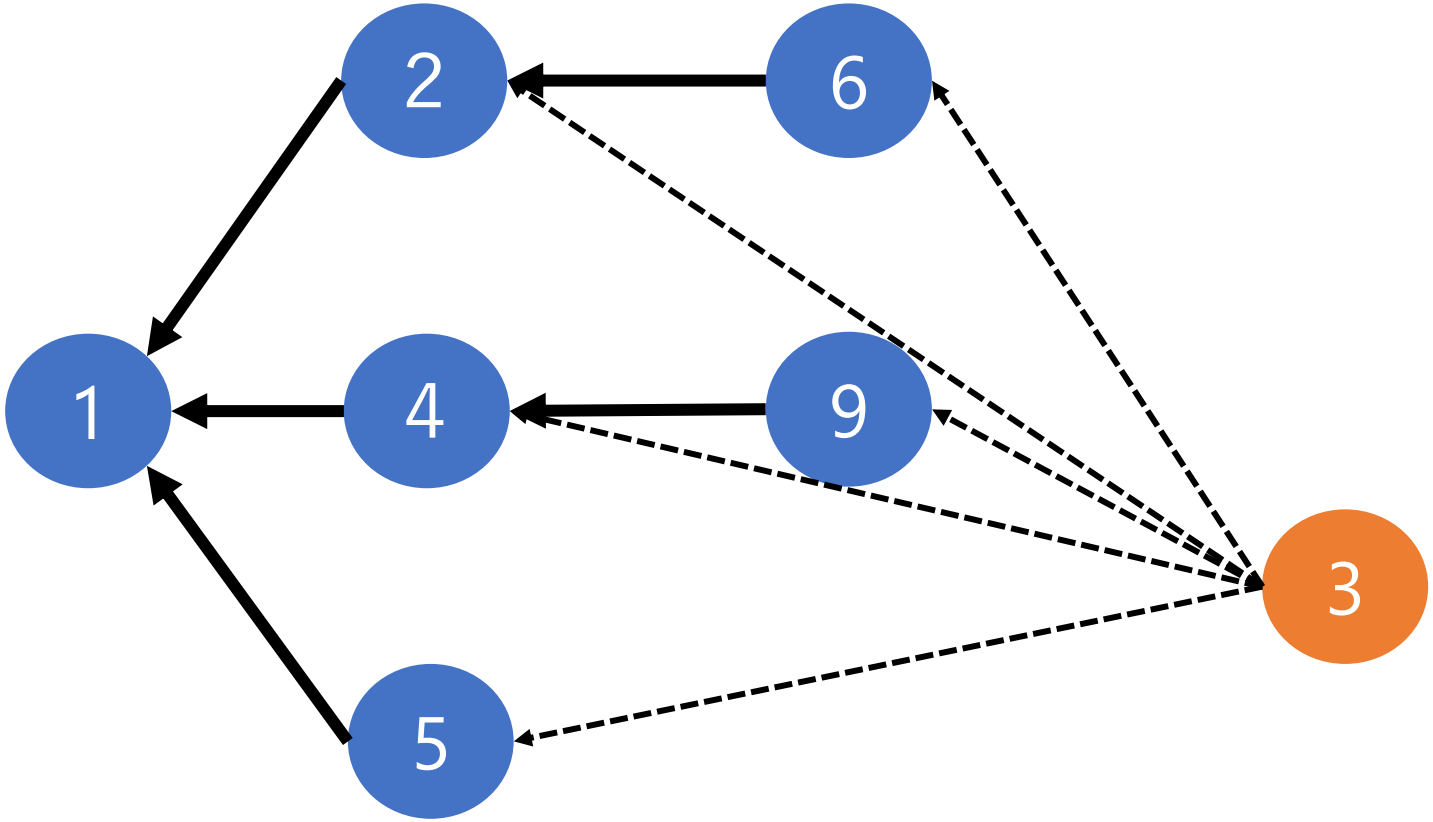
頂点3から近い頂点を順に聞いていき、
頂点1と頂点2のどちらが先に現れるか
で判定

まとめると・・・

１．頂点1から近い順に頂点を追加していく

２．追加したい頂点から近い頂点を順に質問していく

３．既に見た頂点があればそれが親に確定する


2
6
1
4
9
3
5

まとめると・・・

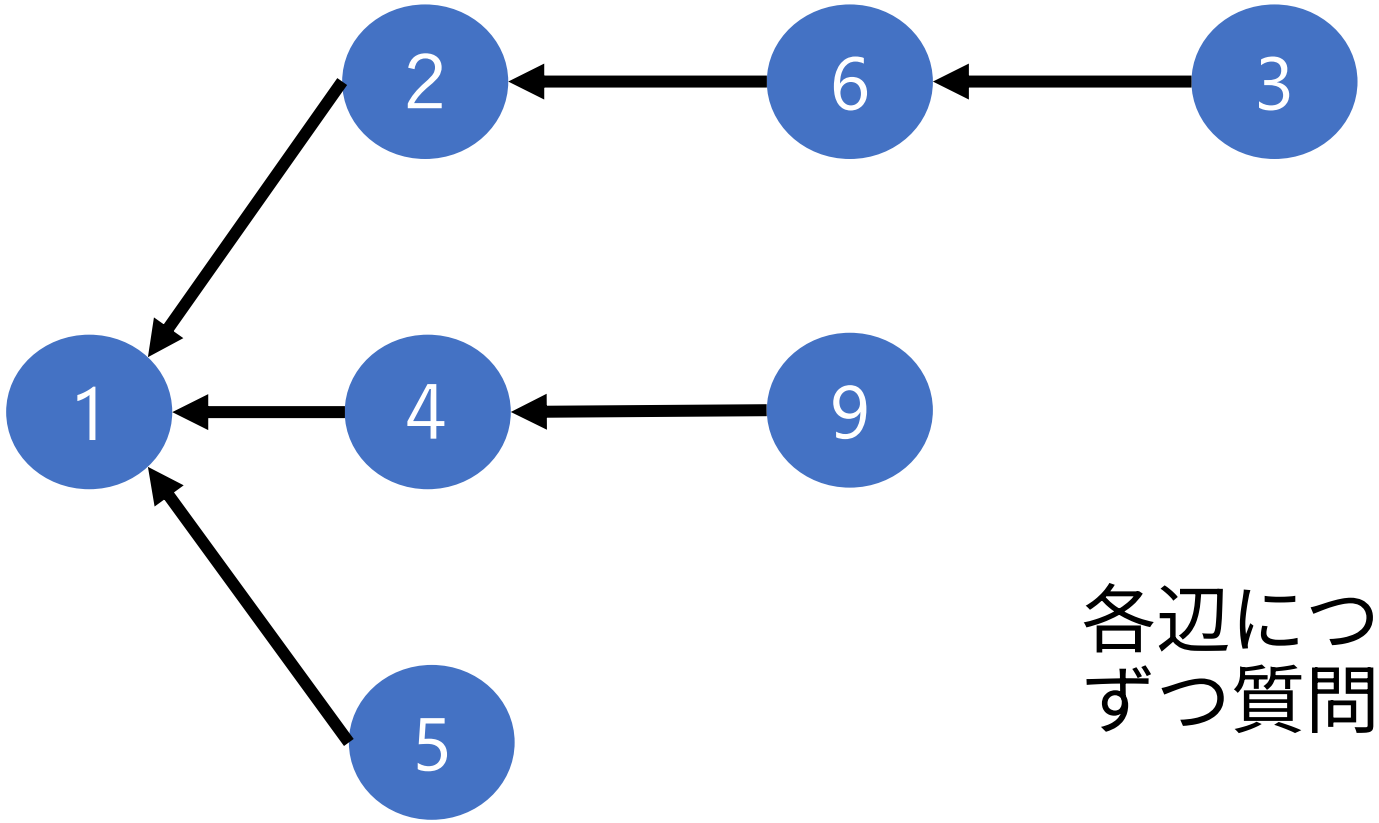
１．頂点1から近い順に頂点を見ていく

２．追加したい頂点から近い頂点を順に質問していく

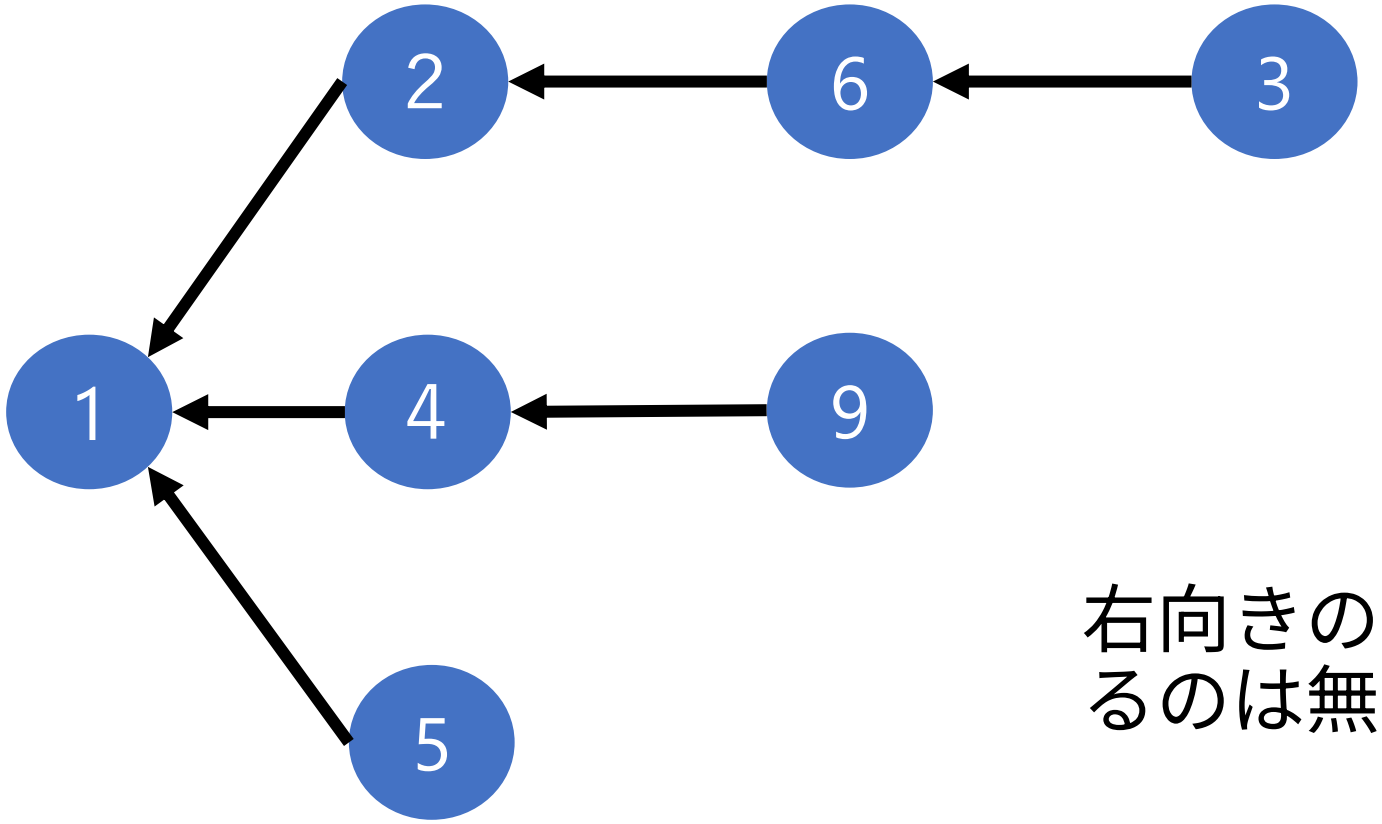
３．既に見た頂点があればそれが親に確定する

最初の N 回 + 2N 回 → 合計 3N 回

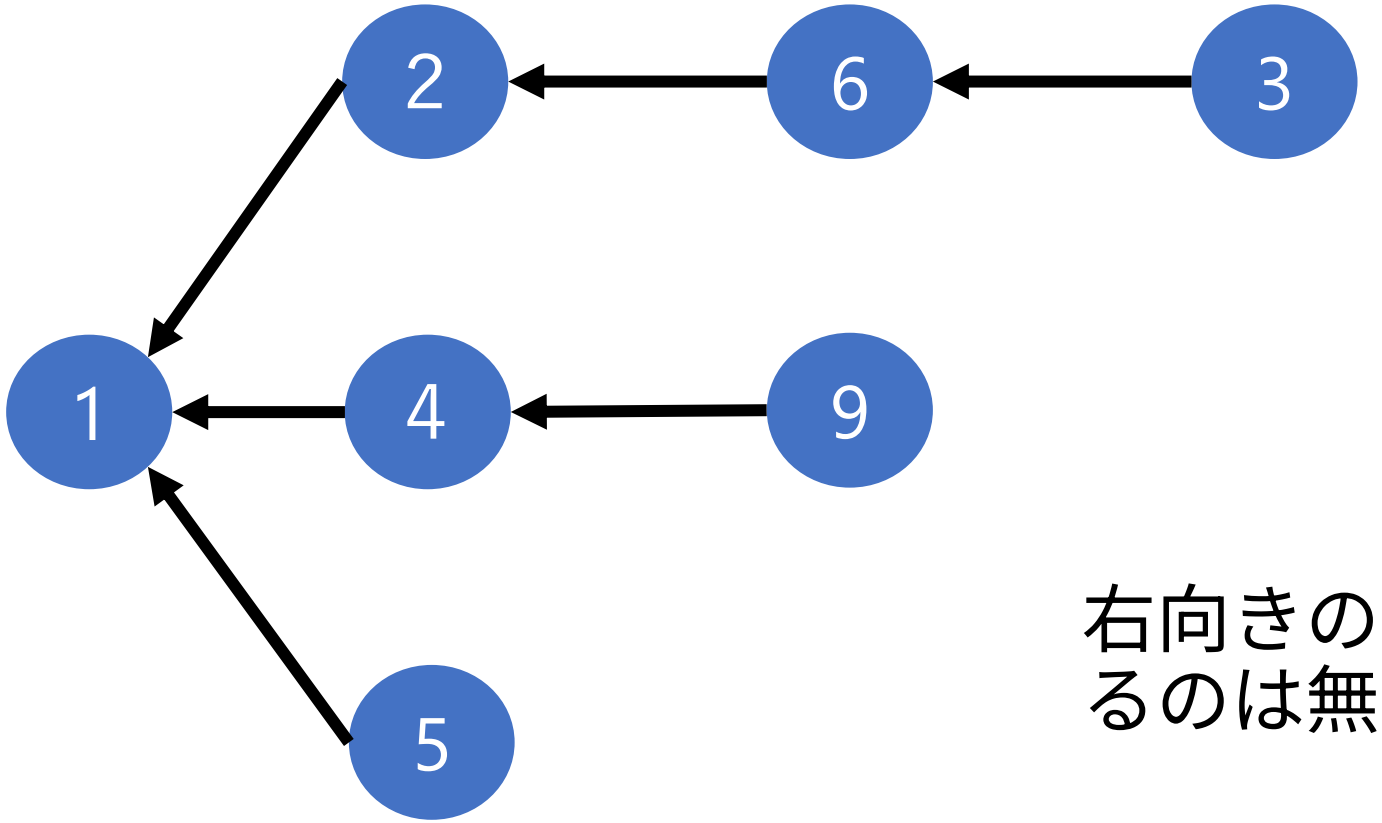
なぜか？



各辺について左向きと右向きの2回
ずつ質問することになる



右向きの辺を聞いた時の情報を捨てるのは無駄



右向きの辺を聞いた時の情報を捨てるのは無駄

→再利用すれば合計 2N で満点が得られる

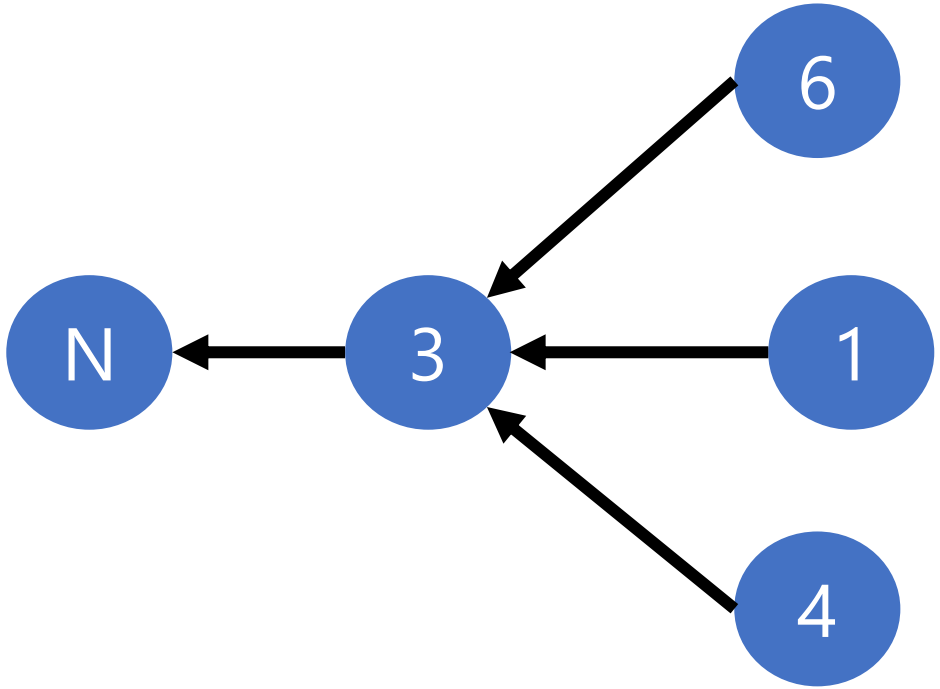
```
for(int i = 1; i < N; i++){
    int a = query(1, i) - 1;
    dist[a] = i;
    near[i] = a;
}

for(int t = 1; t < N; t++){
    int i = near[t];
    if(ok[i]){
        continue;
    }
    for(int j = 1; j < N; j++){
        int a = query(i + 1, j) - 1;
        if(dist[a] < dist[i]){
            answer(i + 1, a + 1);
            break;
        }
        else{
            answer(i + 1, a + 1);
            ok[a] = 1;
        }
    }
}
```

→100点!

頂点番号の大小をうまく利用する解法

・・・頂点 N の隣の頂点を聞く



→L=2.5Nで満点は厳しい

得点分布

JOI

100点：5人	26点：7人
72点：9人	4点：4人
50点：3人	0点：0人

JOIG

66点：1人

20点：1人

52点：1人

4点：4人

41点：1人

0点：1人