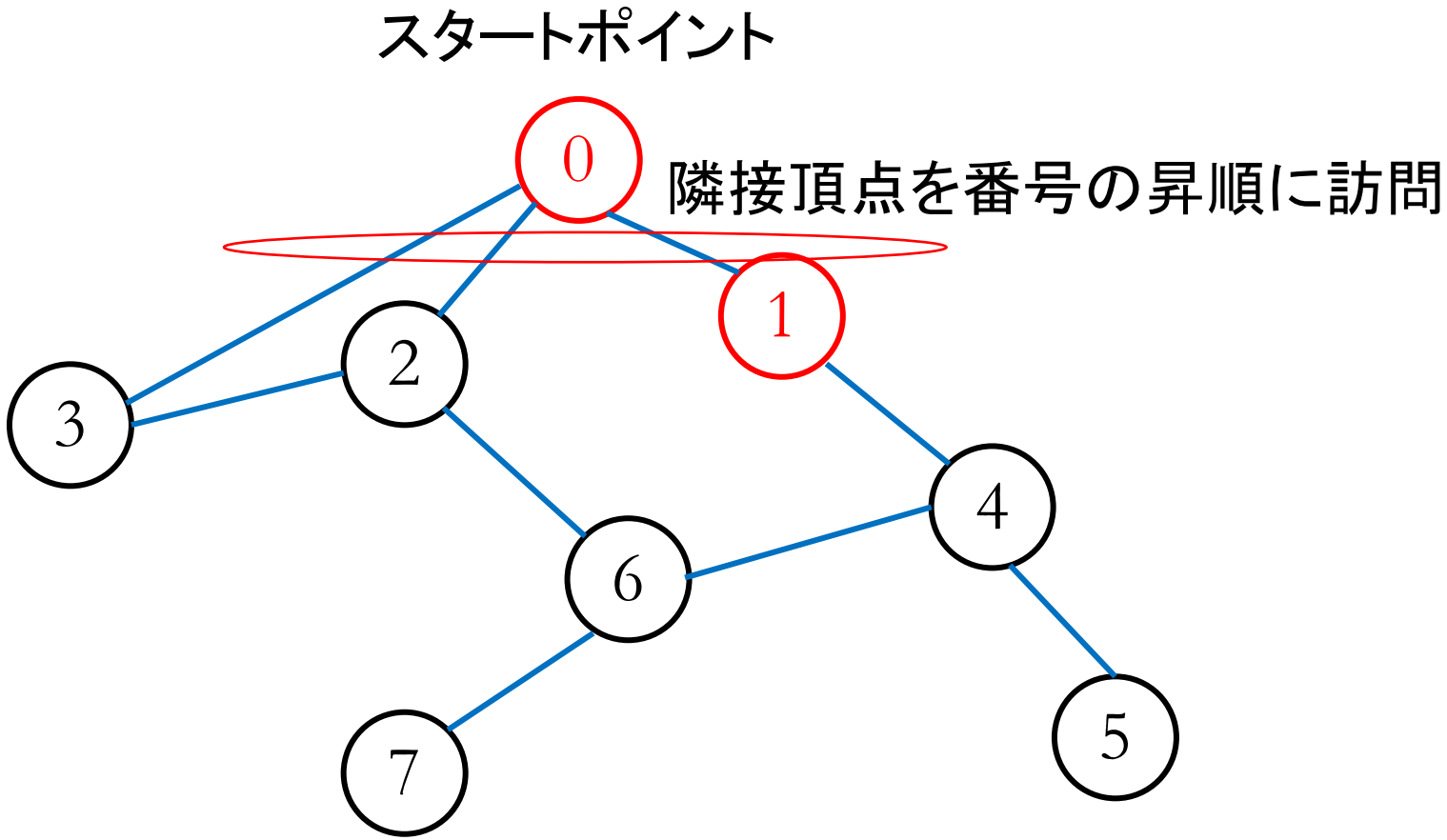
グラフ

グラフ探索の実装

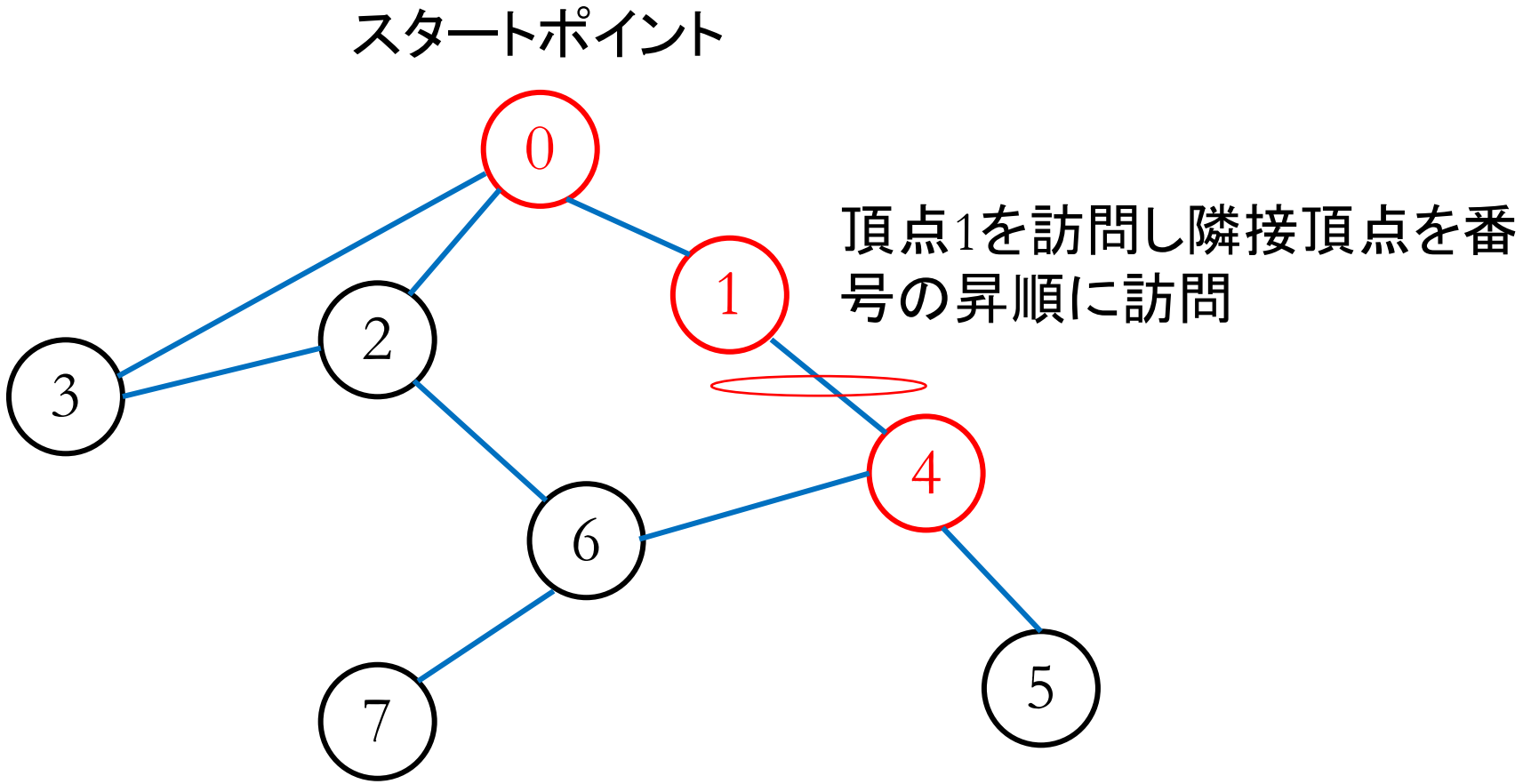
深さ優先探索の実装

- 再帰
- スタック利用

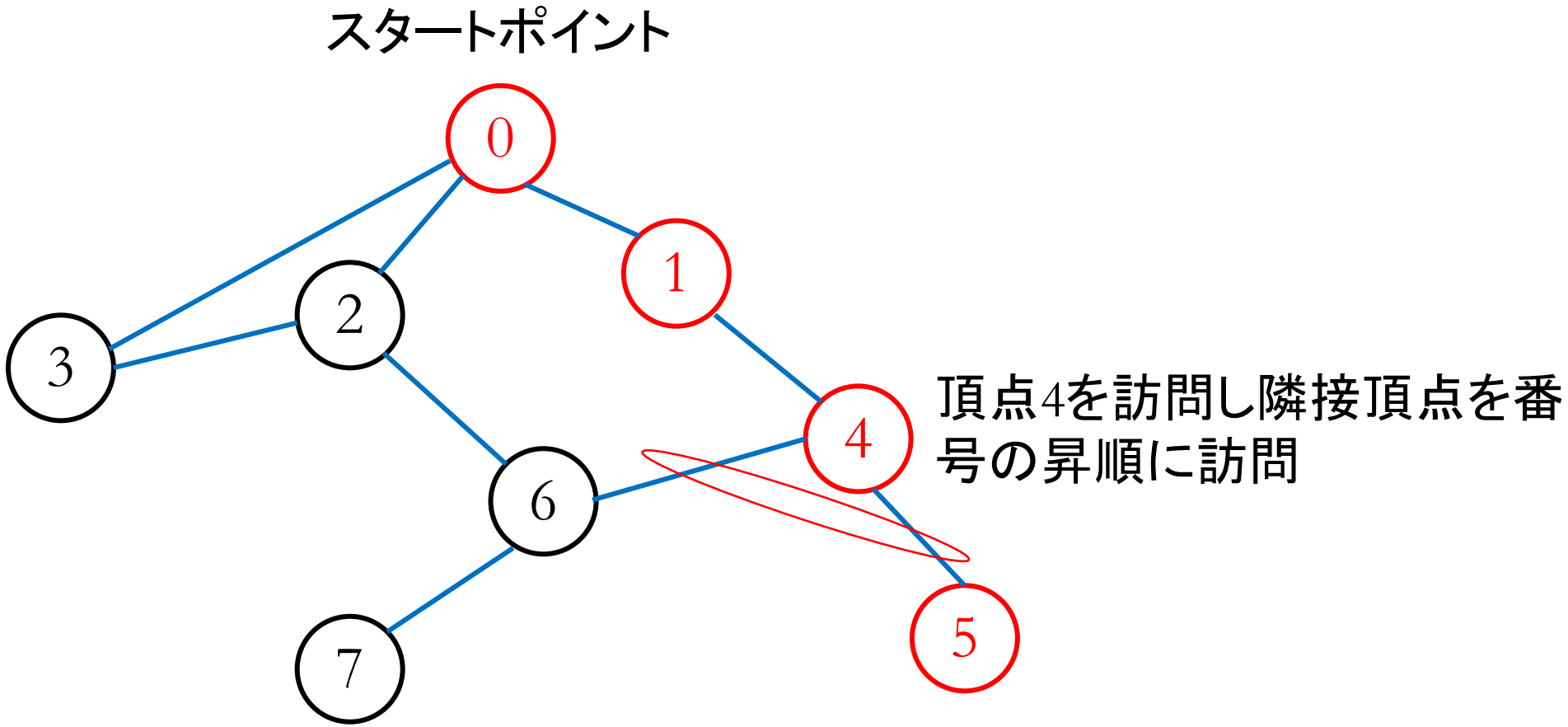
実装のイメージ



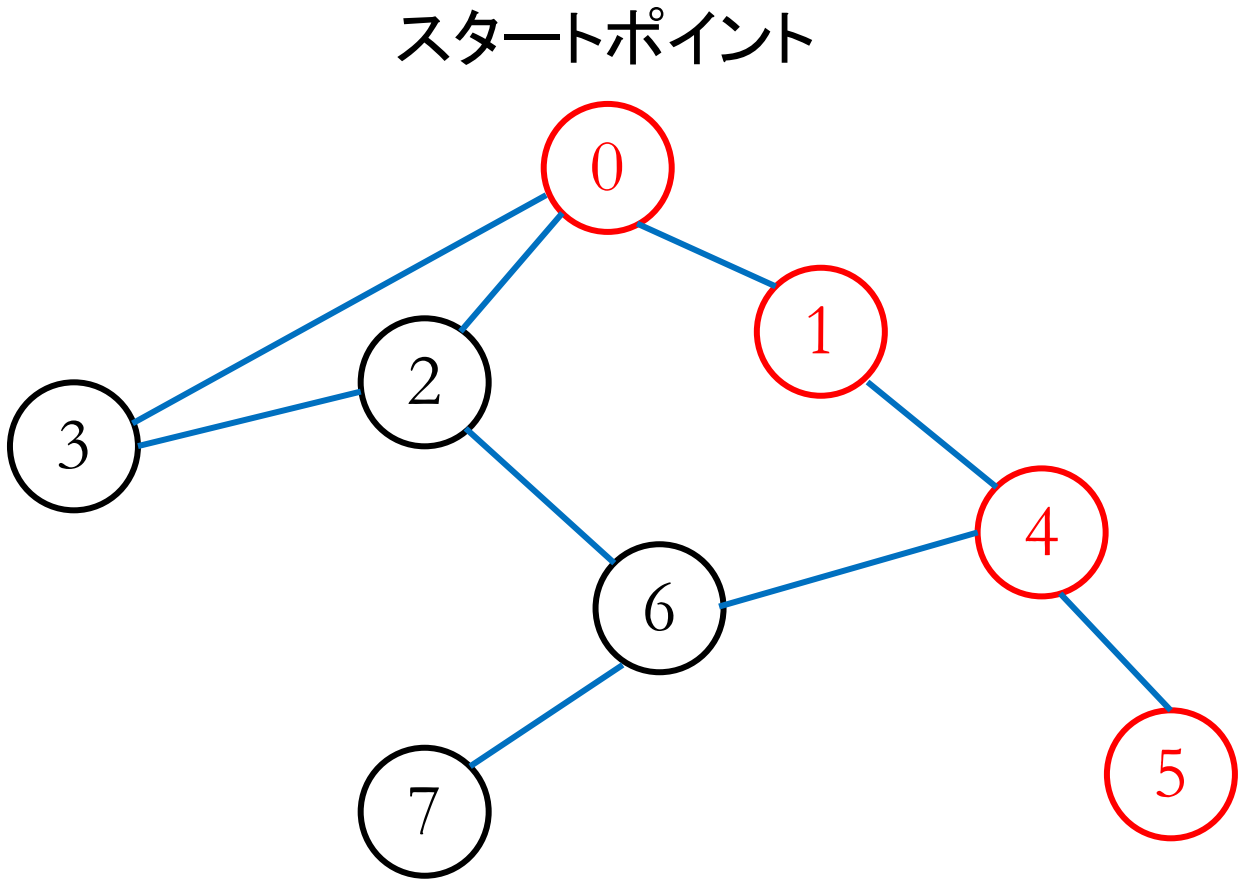
実装のイメージ



実装のイメージ

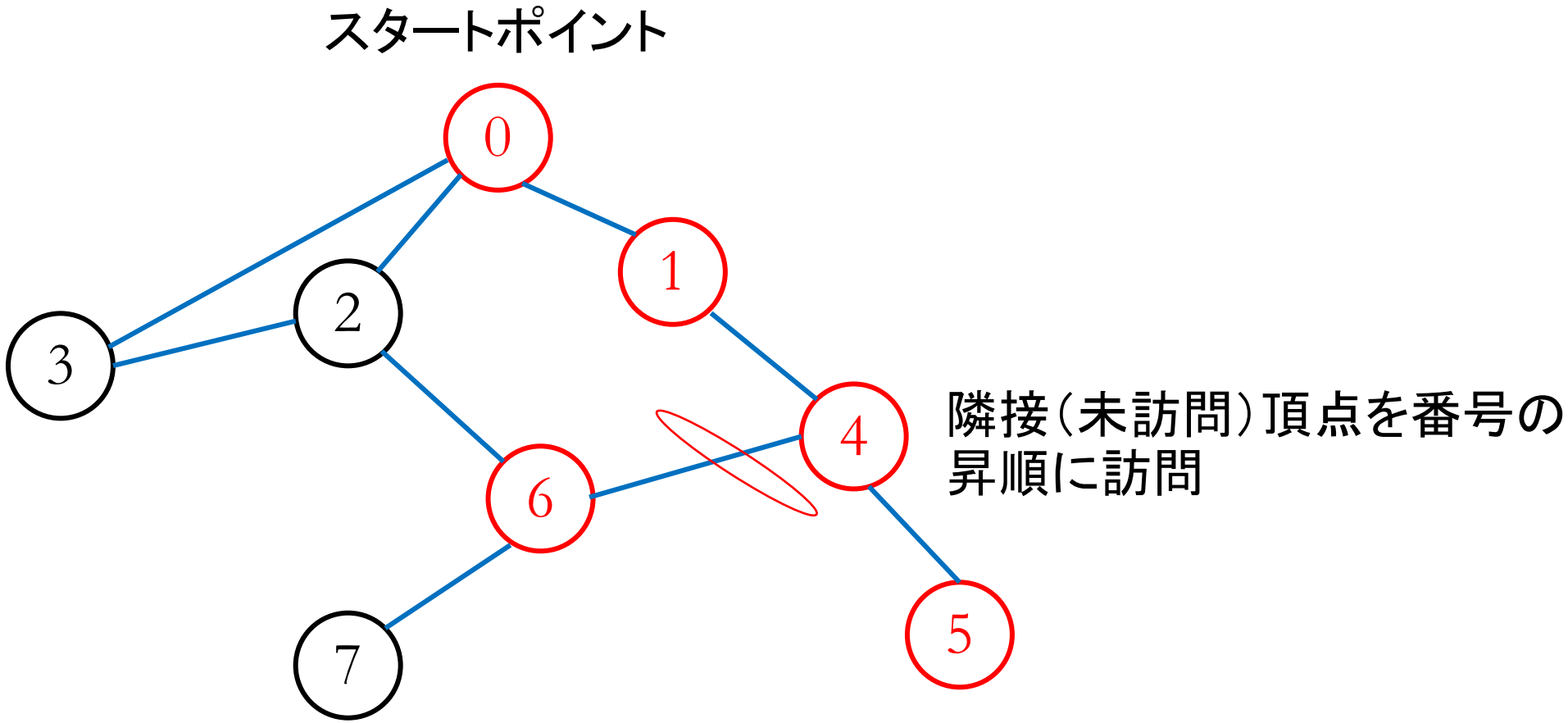


実装のイメージ

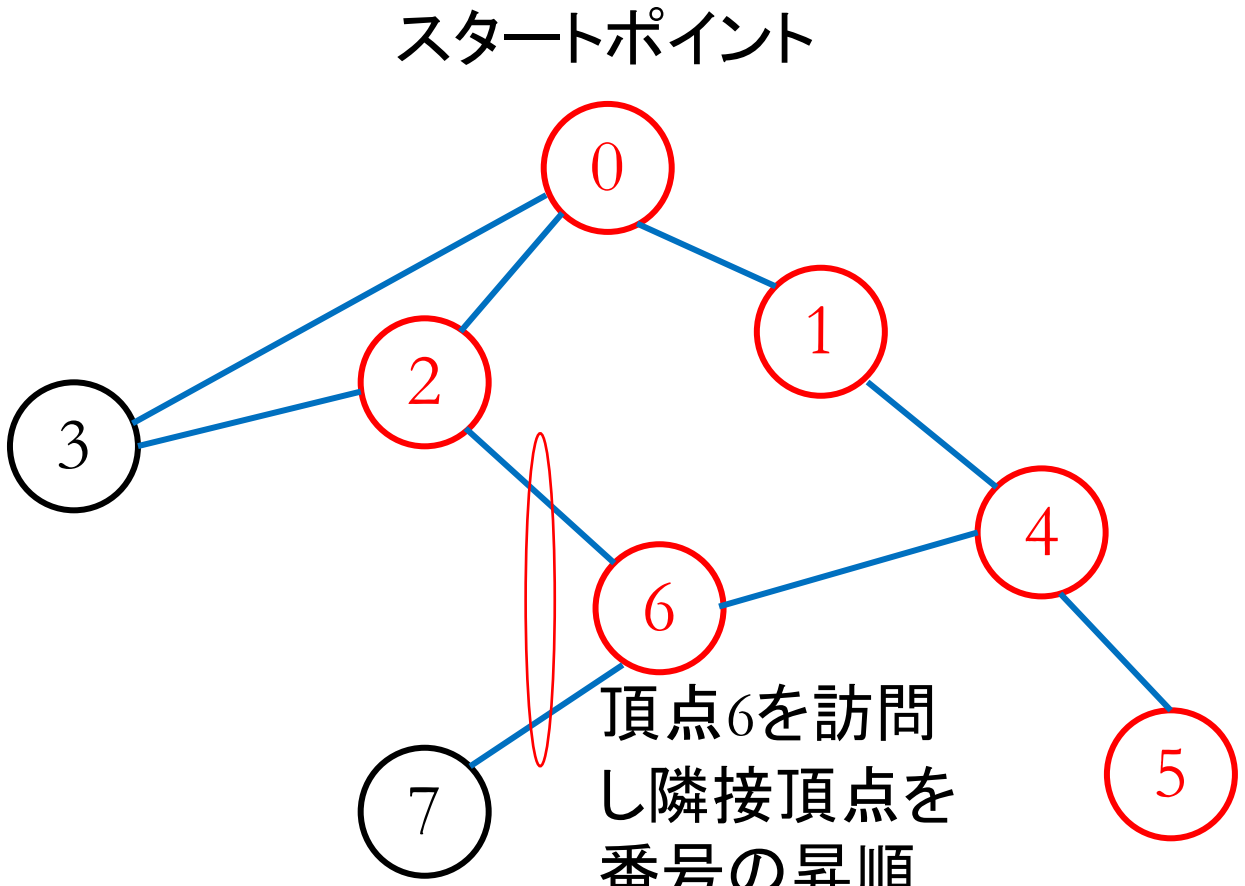


頂点5を訪問し隣接頂点がないため、4に戻って

実装のイメージ



実装のイメージ



以降
2,3を訪問、2に
戻って6に戻って
7を訪問
（詳細省略）

実装のイメージ

- つまり、処理は「頂点iを訪問し隣接の頂点を番号の昇順に深さ優先的に訪問する」の繰り返し
- これを再帰プログラム（基本形）で表現すると、以下となる

```
i=0; // 頂点0を始点とする
visit(i) // iを訪問
{
   for(j)
      jがiに隣接ならvisit(j)
}
```

再帰のアルゴリズム

1. 頂点iを訪問

    1.1 iを訪問済みとする

    1.2 各頂点jについて、iと隣接しているかつ未訪問かをチェック

        1.2.1 Yesなら頂点jを訪問（再帰）

        1.2.2 Noなら、手順1.2に戻る（次のjをチェック）

実装

```
#define N 8

// 隣接行列
int a[][N] = {
            {0,1,1,1,0,0,0,0},
            {1,0,0,0,1,0,0,0},
            {1,0,0,1,0,0,1,0},
            {1,0,1,0,0,0,0,0},
            {0,1,0,0,0,1,1,0},
            {0,0,0,0,1,0,0,0},
            {0,0,1,0,1,0,0,1},
            {0,0,0,0,0,0,1,0}
};
```

```
// 頂点チェック用（訪問済み）配列
// 0: 未訪問   1: 訪問済み

int v[N]; // v[N]:{0, 0, 0, 0, 0, 0, 0, 0};
```

再帰のプログラム

```
#include <stdio.h>

void visit(int i){
  int j;

  ?1 = 1;
  printf(“%d ”, i);

  for(j = 0; j < N; j++)
    if(隣接チェック && 未訪問チェック)
      visit(j);
}
```

```
int main(void)
{
  visit(0); //任意の頂点番号
  printf("¥n");
  return 0;
}

./a.out
0 1 4 5 6 2 3 7
```

manaba小テスト：04-1

- 6分
- 6点

再帰の代わりにスタックを用いる

- 前述の通り、処理は「頂点iを訪問し隣接の頂点を番号の昇順に深さ優先的に訪問する」の繰り返し
- これをスタックを用いて表現すると、アルゴリズムの基本形は以下となる

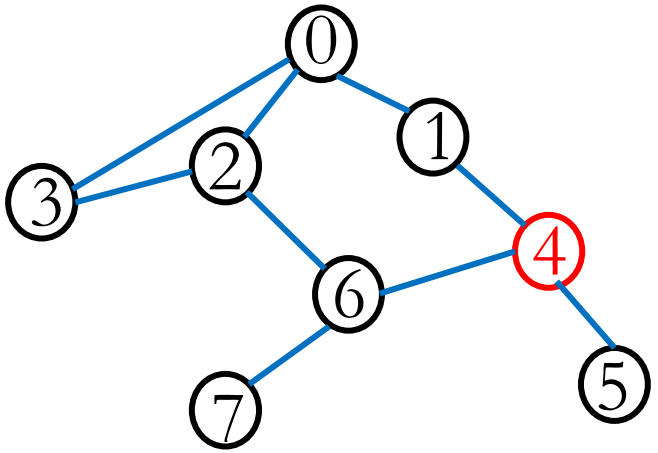
```
起点とする頂点をスタックに格納
スタックが空となるまで以下を繰り返す
    スタックから頂点（頂点番号）1つ取り出し、iとする   // iを訪問
    各頂点jについて以下を繰り返す
        iに隣接ならjをスタックに格納
}
```

スタック利用のアルゴリズム

1. 起点となる頂点をスタックに入れる
2. スタックが空になるまで以下を繰り返す
   2.1 スタックから頂点1つを取り出し、iとする
   2.2 iが訪問済みであれば手順2に戻る
   2.3 iを訪問済みとする
   2.4 すべての頂点j（ここでは逆順：j=N-1, ..., 0）について、iと隣接しているかつ未訪問かをチェック
      2.3.1 Yesならjをスタックにいれる

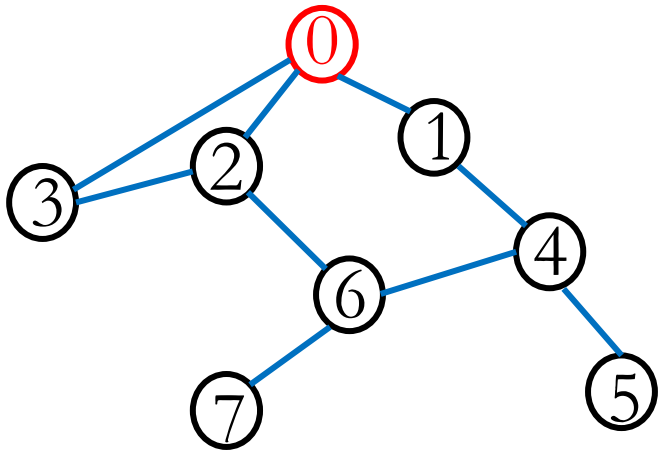
補足

- グラフの場合、複数の頂点が1つの頂点に隣接するので、上記アルゴリズムだと、同じ頂点が複数回スタックに格納される場合がある
- 右のグラフの場合、4を始点とした場合、4,1,0,2,3の順で探索していくが、0, 2を探索するとき、3が未訪問なので、2回スタックに格納される（6も同様）
- その結果、手順2.2がなければ、
  - 4, 1, 0, 2, 3, 6, 7, 3, 5, 6と、同じ頂点が複数回訪問され、訪問頂点数が頂点総数を超えてしまう
  - 一方、訪問頂点数が頂点総数と等しいとなった場合に訪問を打ち切る、というような方法を取った場合は頂点3の2回目の訪問が頂点5よりも前になるので、探索結果が間違ってしまう



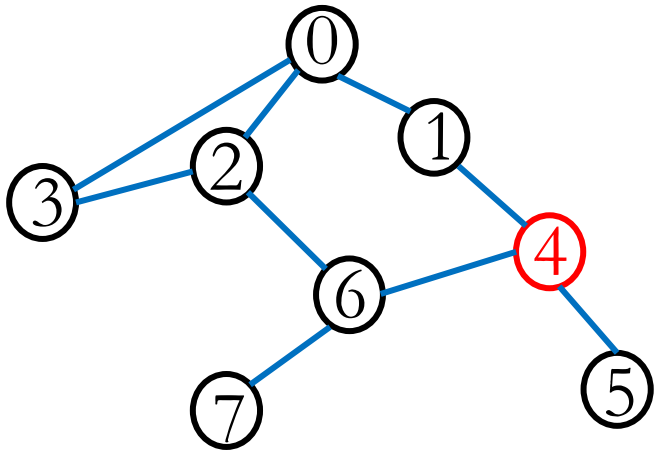
演習

- 下記グラフの頂点0を起点とし、上記アルゴリズムを適用した場合、スタックが空になるまでのスタックの中身と、訪問する頂点番号（取り出して訪問済みとする頂点番号）を時間順に示しなさい



演習

- 下記グラフの頂点4を起点とし、上記アルゴリズムを適用した場合、スタックが空になるまでのスタックの中身と、訪問する頂点番号（取り出して訪問済みとする頂点番号）を時間順に示しなさい



manaba小テスト：04-2

- 15分
- 10点

スタック利用のプログラム

```
#include <stdio.h>
#include “stack.h”
//「スタックの講義資料」(ダウロード可能)を参照
// a, vの宣言はP11
void visit(int i)
{
  int j;

  InitStack();
  ?1;
  while(!StackEmpty()){
    i=?2;
    if(v[i]==1) continue;
    v[i]=?3;
    printf("%d ", i);
    for(j=N-1; j>=0; j--) // 逆順！！！
      if(a[i][j] == 1 && (v[j] == 0)) ?4;
  }
  printf("¥n");
}

int main(void)
{
  visit(0);
  return 0;
}
```

manaba小テスト：04-3

- 6分
- 8点

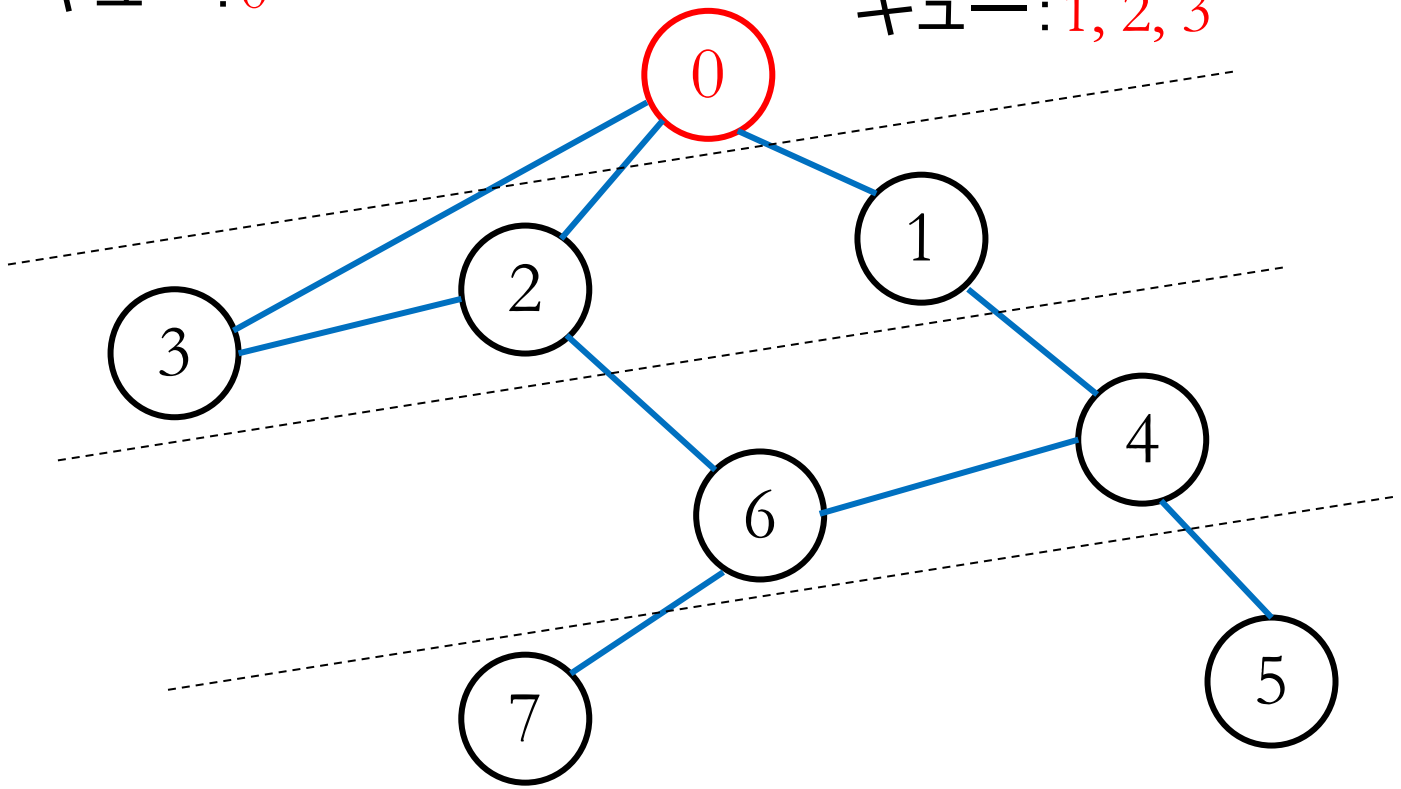
幅優先探索の実装

- キューを利用する

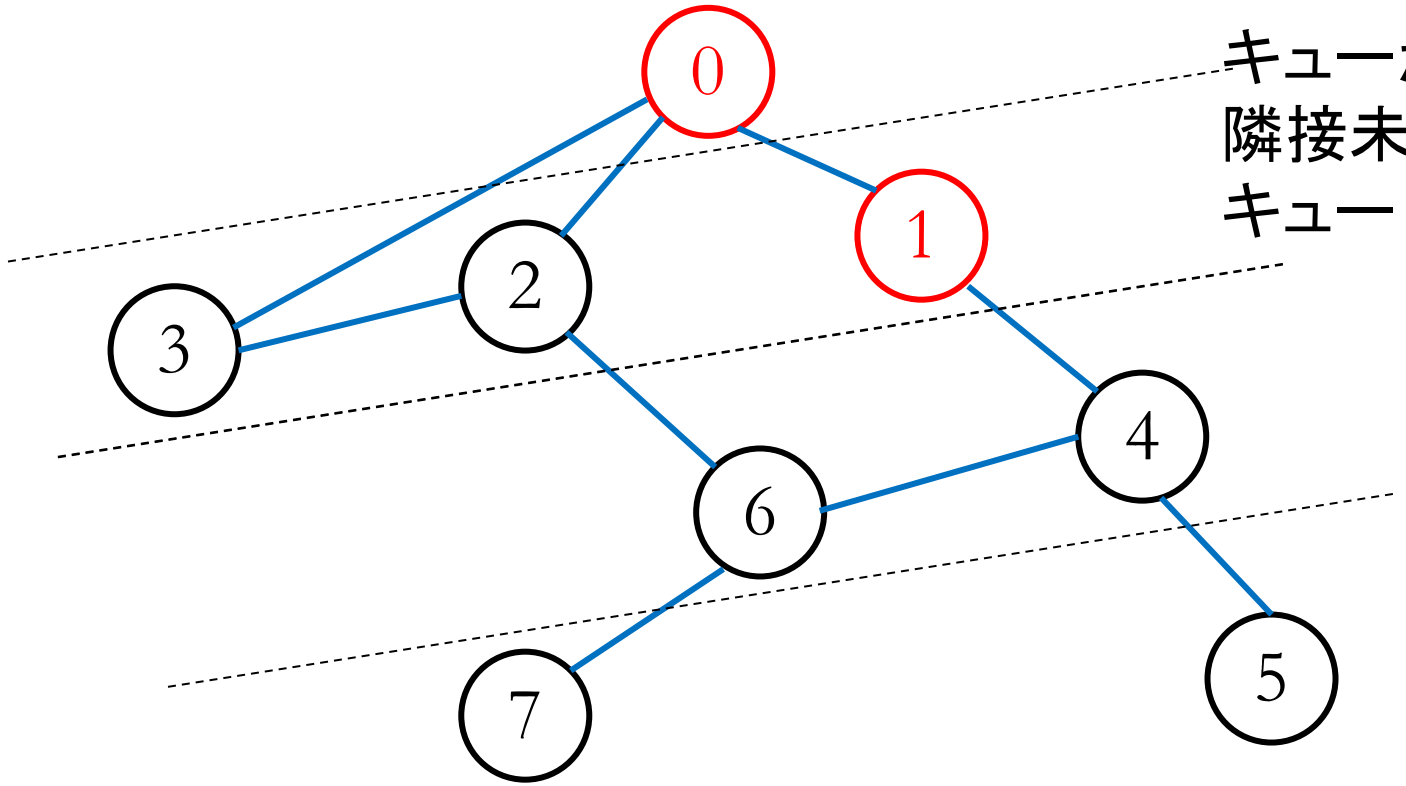
実装のイメージ

初期状態
キュー：0

キューから先頭（0）を取り出し（0を訪問）、
隣接未格納頂点を番号の昇順にキューに
キュー：1, 2, 3

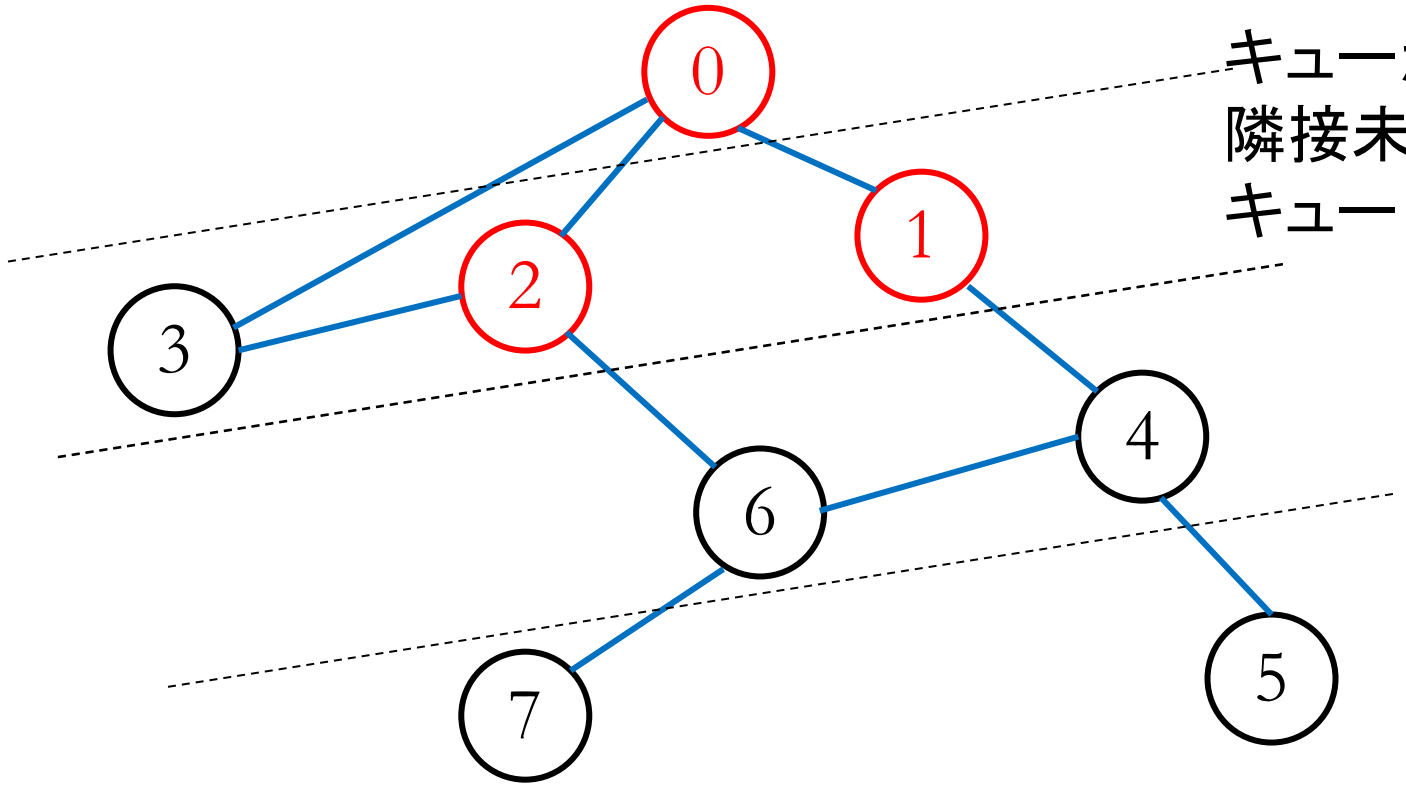


実装のイメージ



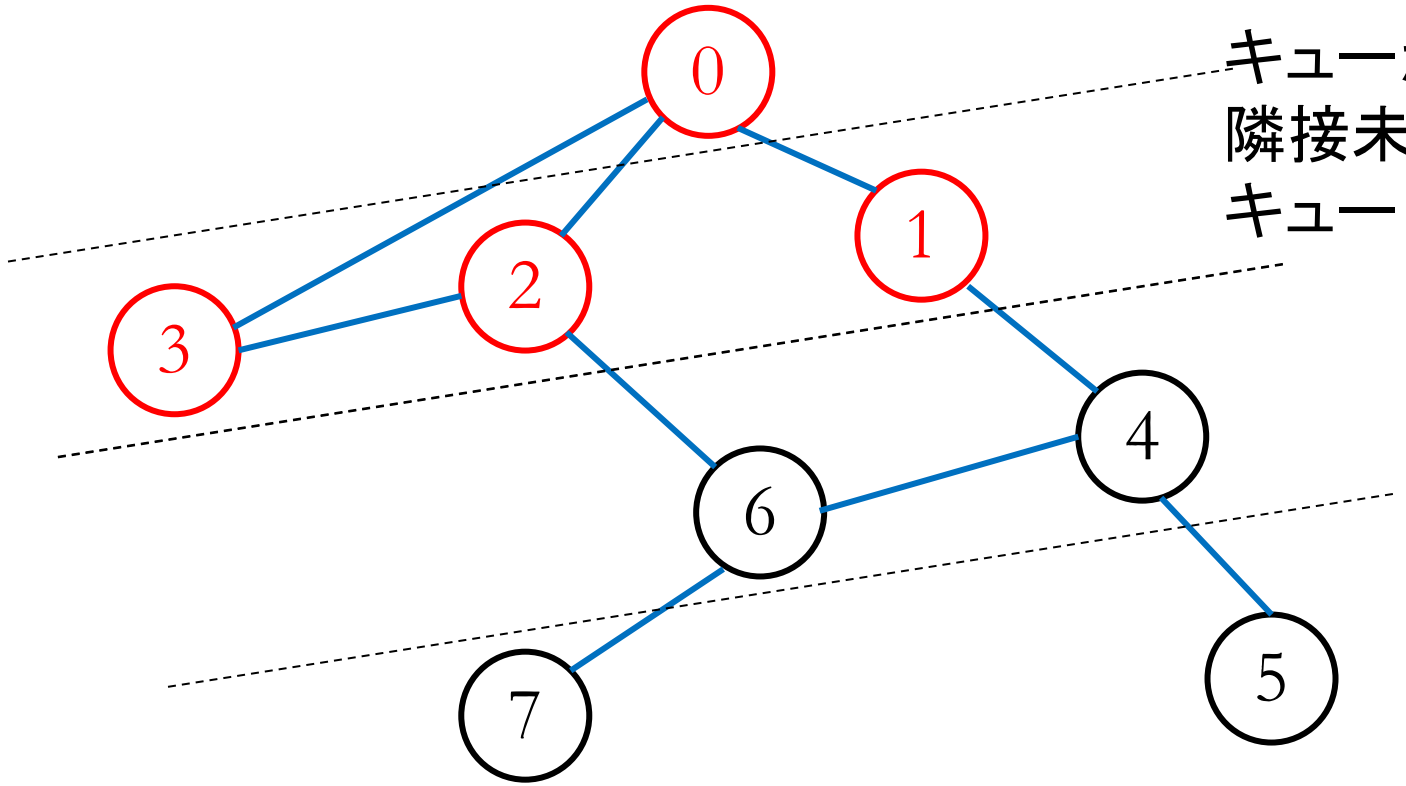
キューから先頭(1)を取り出し(1を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー:2, 3, 4

実装のイメージ



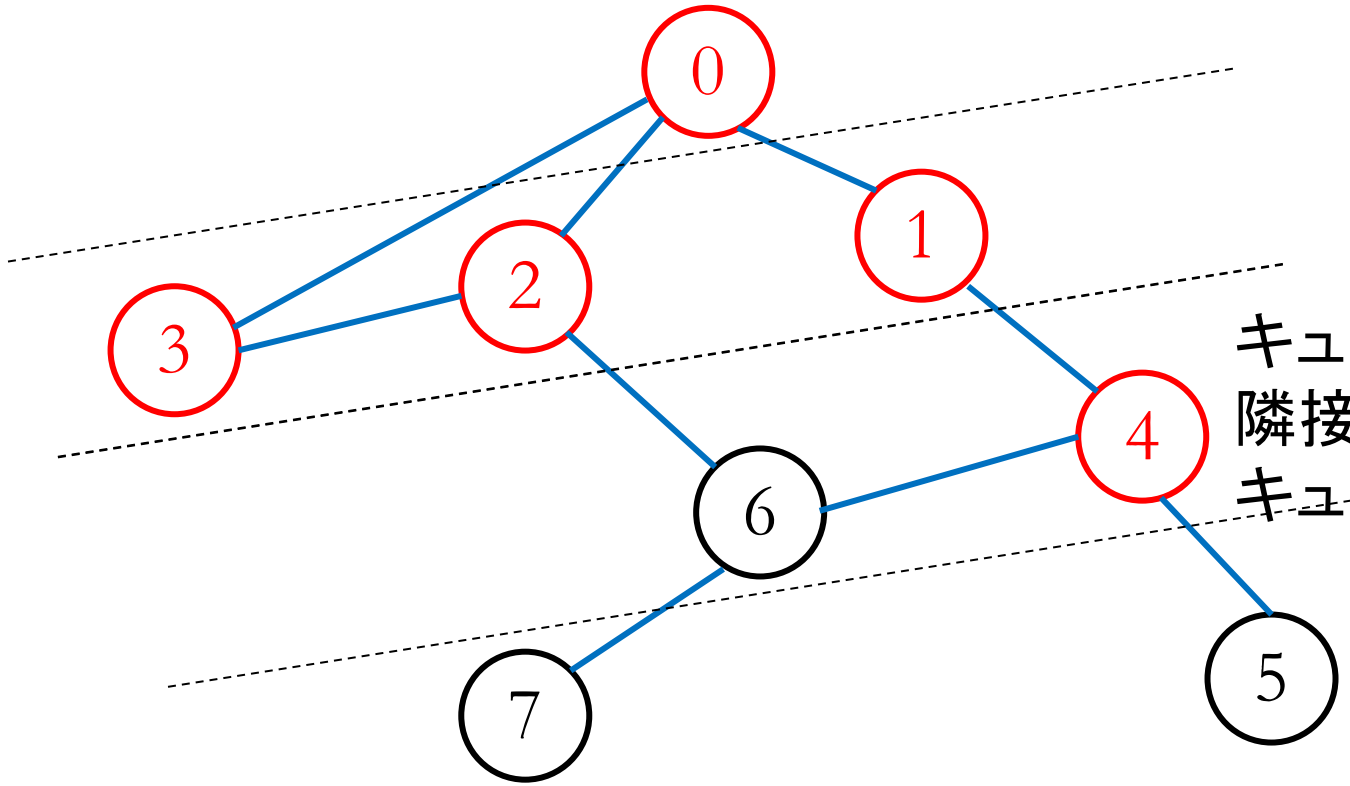
キューから先頭(2)を取り出し(2を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー: 3, 4, 6

実装のイメージ



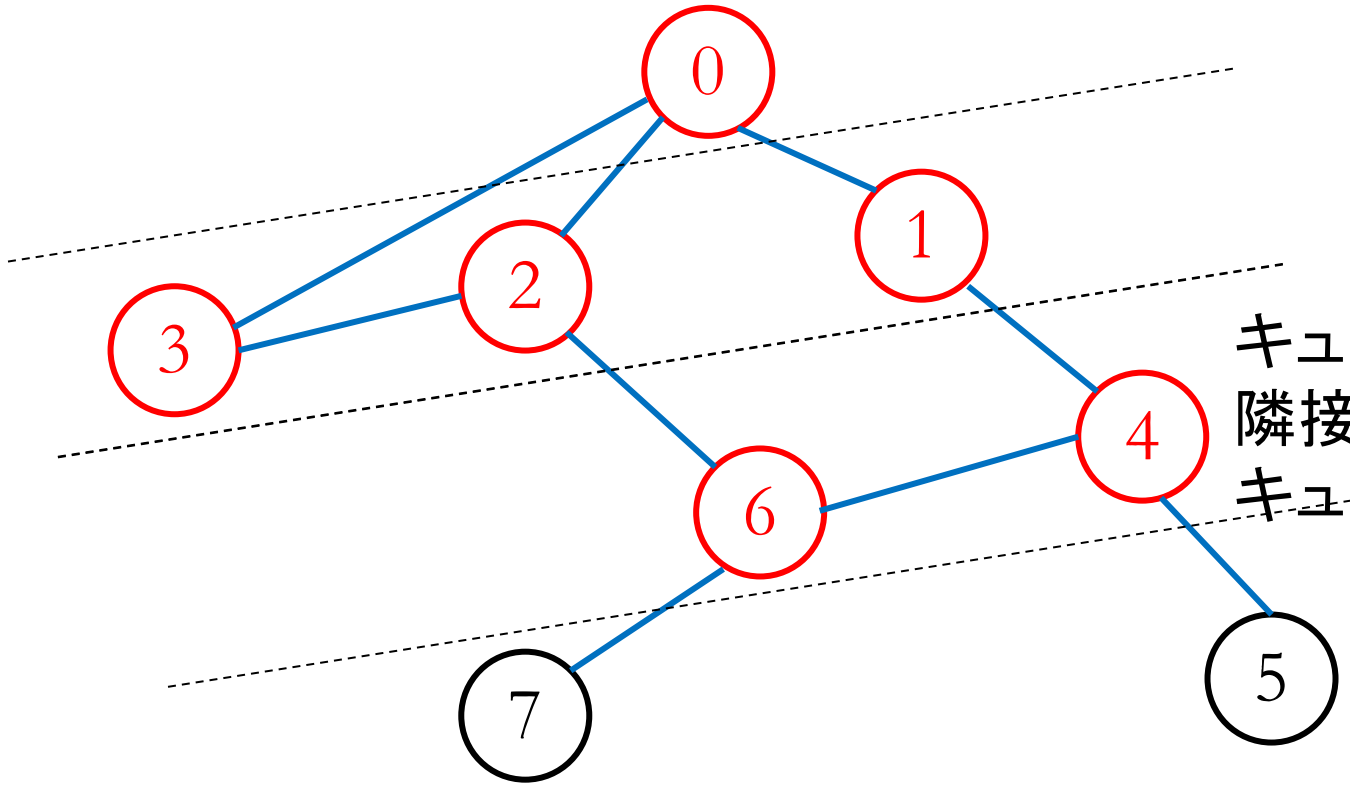
キューから先頭(3)を取り出し(3を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー: 4, 6

実装のイメージ



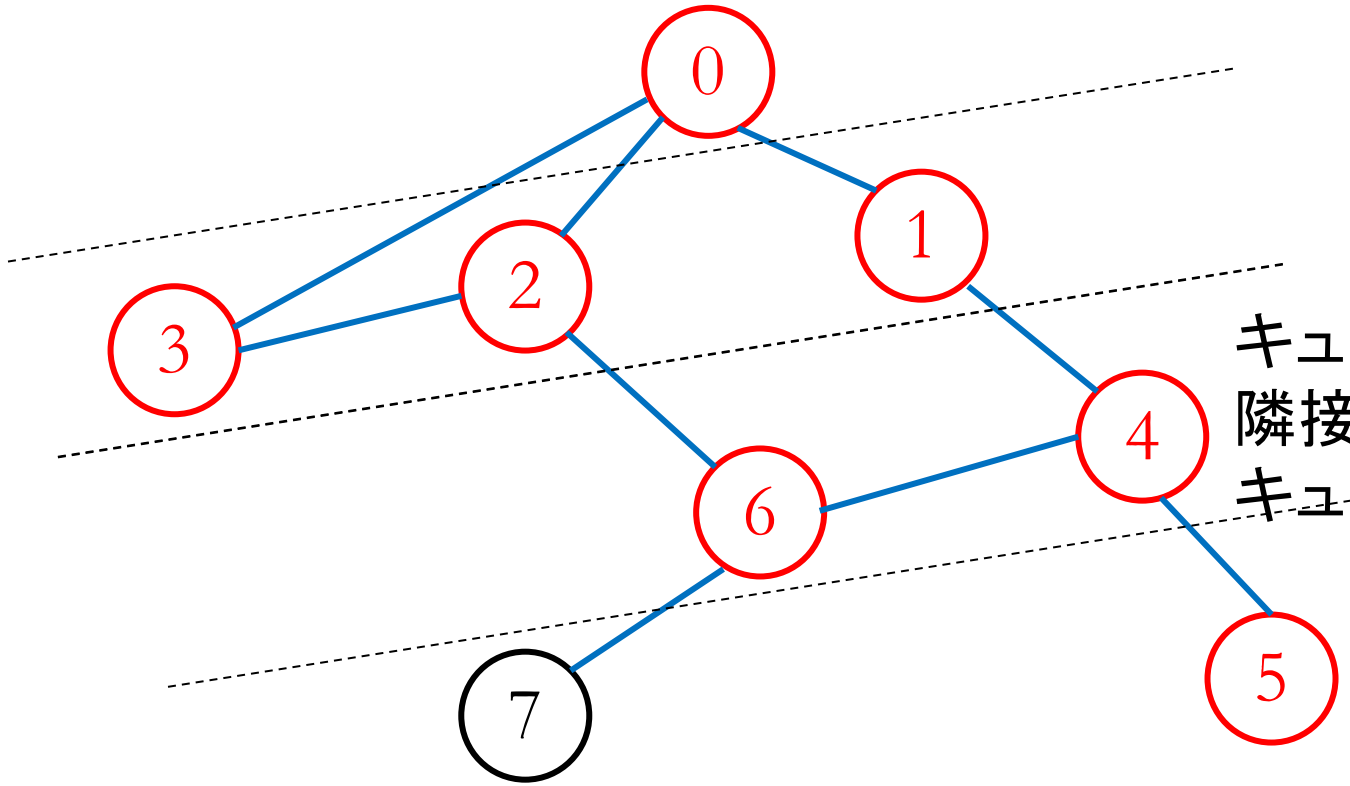
キューから先頭(4)を取り出し(4を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー: 6, 5

実装のイメージ



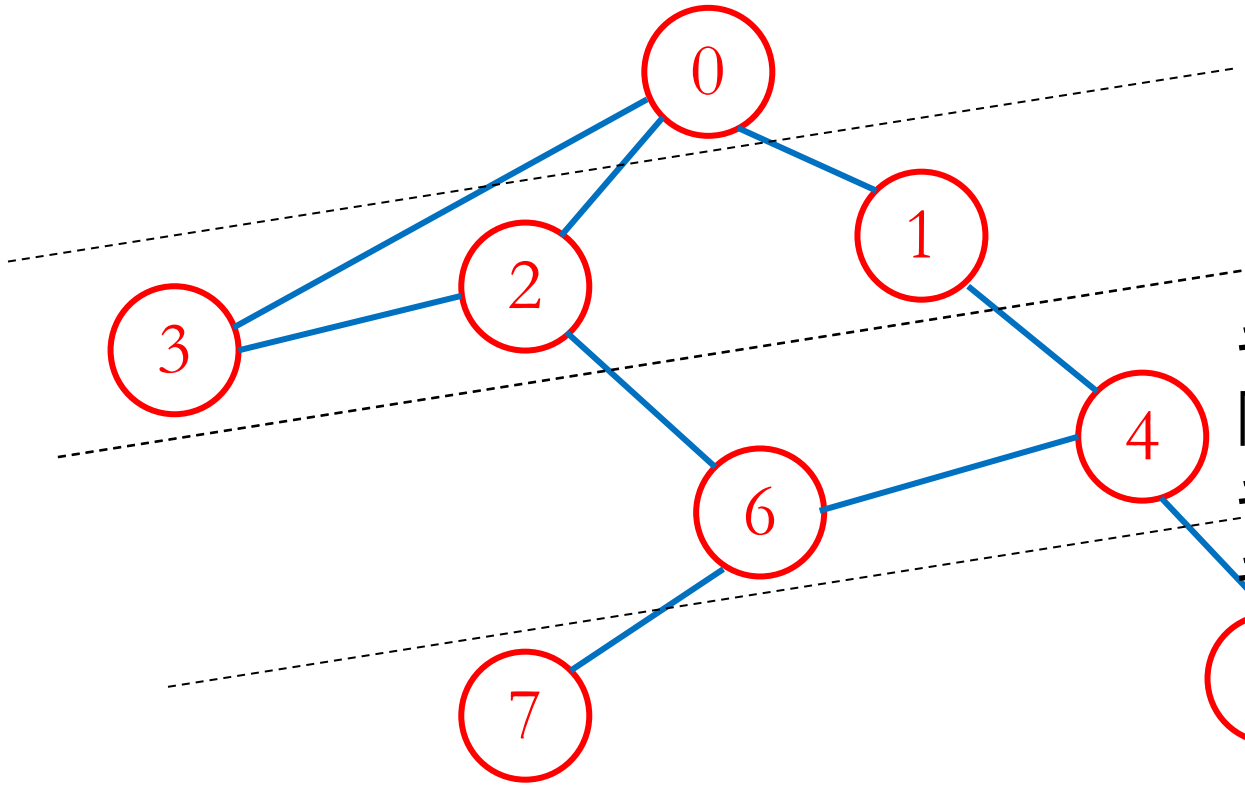
キューから先頭(6)を取り出し(6を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー: 5, 7

実装のイメージ



キューから先頭(5)を取り出し(5を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー:7

実装のイメージ



キューから先頭(7)を取り出し(7を訪問)、
隣接未格納頂点を番号の昇順にキューに
キュー:
キューが空となったため、終了

実装のイメージ

- 前述の通り、処理は「キューから先頭(i)を取り出し(iを訪問)、隣接未格納頂点jを番号順にキューに」の繰り返し
- アルゴリズムの基本形は以下となる

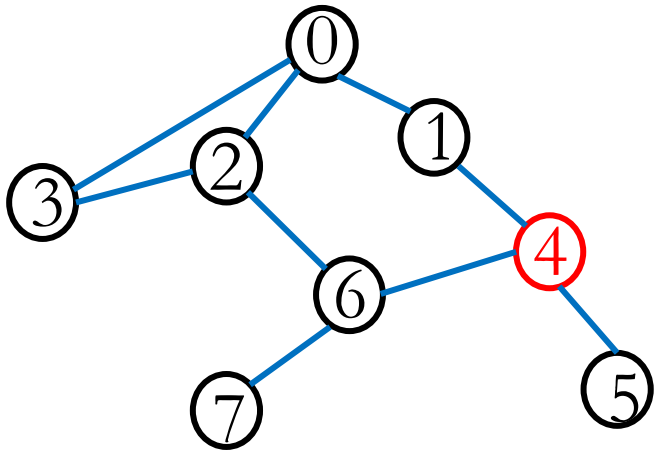
起点とする頂点をキューに格納
キューが空となるまで以下を繰り返す
　キューからデータを取り出し、i とする
　各頂点jについて以下を繰り返す
　　iに隣接ならjをキューに格納

アルゴリズム

1. 始点となる頂点をキューに入れ、格納済みとする
2. キューが空になるまで以下を繰り返す
   2.1 キューから頂点1つを取り出す。これをiとする
   2.2 すべての頂点j（j=0,...,N-1）について、iと隣接しているかつ未格納かをチェック
      2.2.1 Yesならjをキューにいれ、格納済みとする

演習

- 下記グラフの頂点4を起点とし、上記アルゴリズムを適用した場合、キューが空になるまでのキューの中身と取り出した頂点番号を時間順に示しなさい



manaba小テスト：04-4

- 10分
- 8点

実装

```
#define N 8

// 隣接行列
int a[][N] = {
            {0,1,1,1,0,0,0,0},
            {1,0,0,0,1,0,0,0},
            {1,0,0,1,0,0,1,0},
            {1,0,1,0,0,0,0,0},
            {0,1,0,0,0,1,1,0},
            {0,0,0,0,1,0,0,0},
            {0,0,1,0,1,0,0,1},
            {0,0,0,0,0,0,1,0}
};
```

```
// 頂点チェック用（格納済みかの）配列
// 0: 未格納　1: 格納済み

int v[N]; // v[N]={0, 0, 0, 0, 0, 0, 0, 0};
```

プログラム

```
#include <stdio.h>
#include “queue.h”
//「キューの講義資料」(ダウロード可能)を参照

visit(int i)
{
  int j;
  InitQueue();
  ?1;
  v[i]=?2;
  while(!QueueEmpty()){{
    i = ?3;
    printf("%d ", i);
  for(j=0; j<N; j++)
      if(a[i][j] == 1 && v[j] == 0 ){
        ?4;
        v[j]=?5;
      }
  }
  printf("¥n");
}

int main(void){
  visit(0); //任意の頂点
  return 0;
}
```

manaba小テスト：04-5

- 10分
- 10点