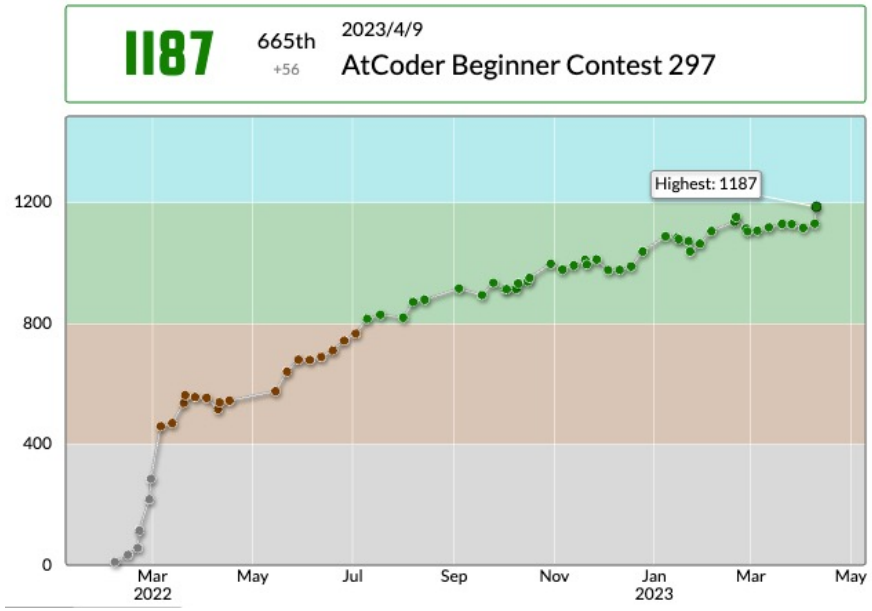
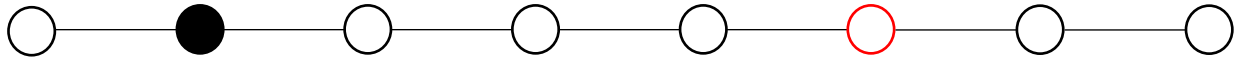
BFS(幅優先探索)

BFSを実装できるようになる

自己紹介

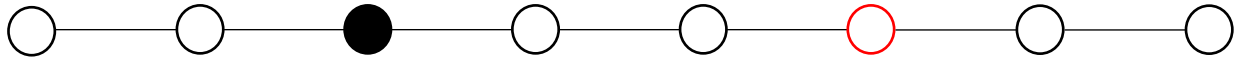
- 井桁　広翔(いげた　ひろと)
- B3　情報科学科
- Twitter @igeeeeeee_
- 趣味：ゲーム



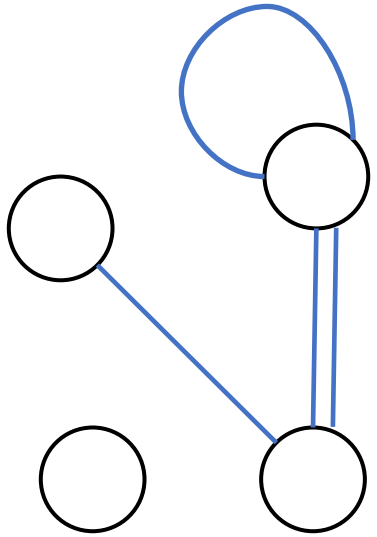


目次

- 自己紹介
- グラフ理論の用語確認
- BFSとは
- BFSの実装
- BFSの応用
- 鉄則本の紹介
- 練習問題

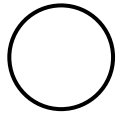


グラフの用語確認



- グラフ：頂点と辺の集合により構成されるもの

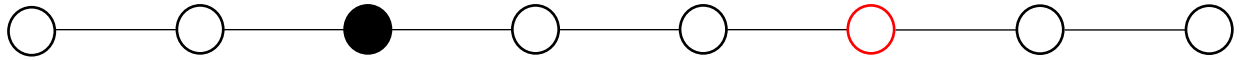
- 頂点(node, vertex)



- 辺(arc, edge)
  - 無向辺 :向きのない辺

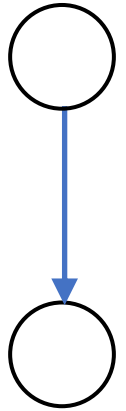
  - 有向辺 :向きのある辺

- 無向辺のグラフは無向グラフ、有向辺のグラフは有向グラフ

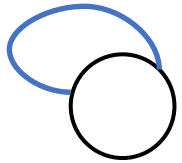


グラフの用語確認２

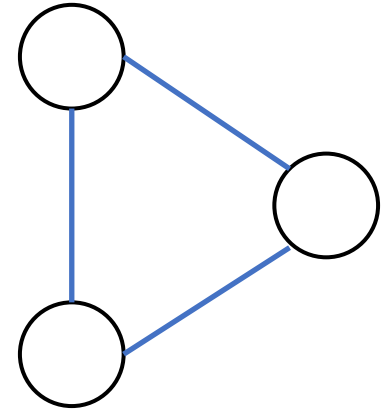
- 頂点の次数 :頂点から出ている辺の本数
-有向辺の時は入次数、出次数に分かれる

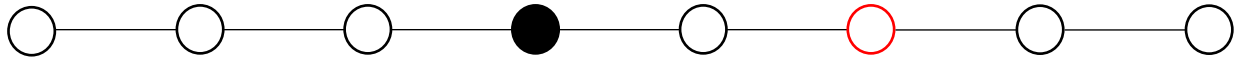


- ループ(自己ループ)



- 閉路(サイクル):隣接する頂点をたどったもので
  始点と終点が同じもの。(同じ辺は２回以上通れない)



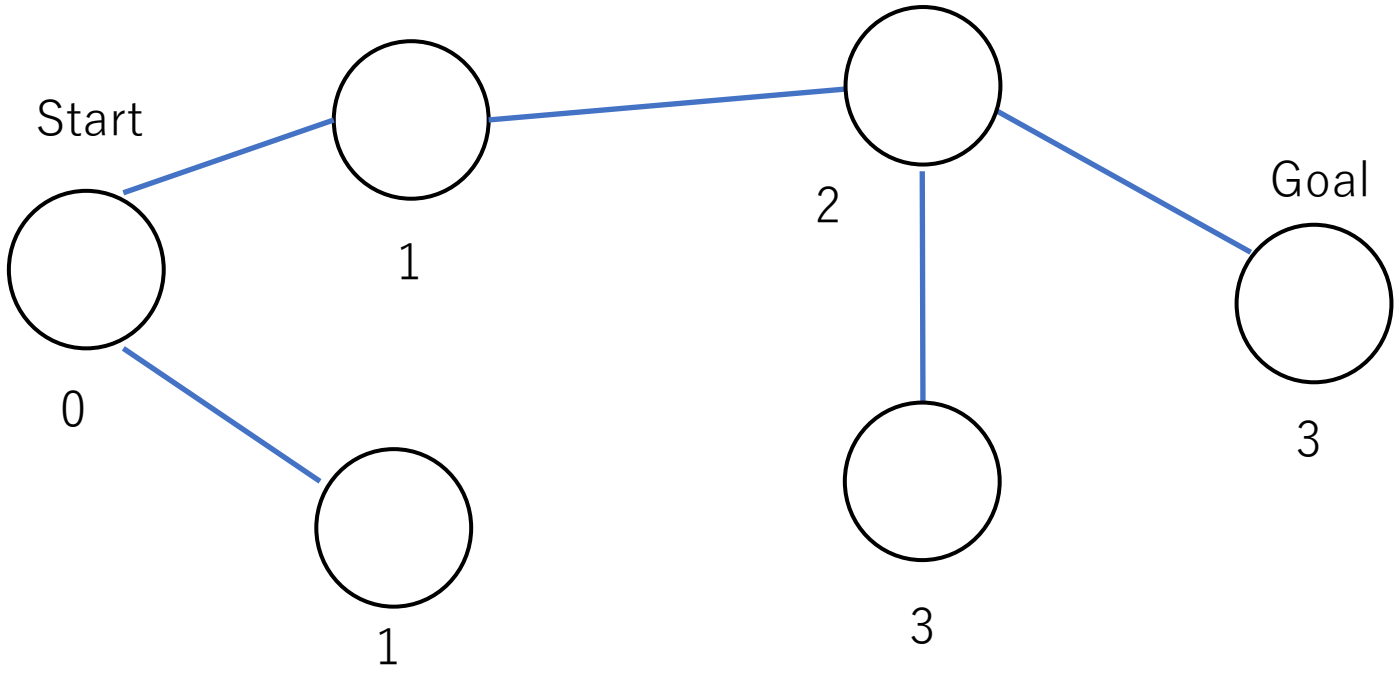


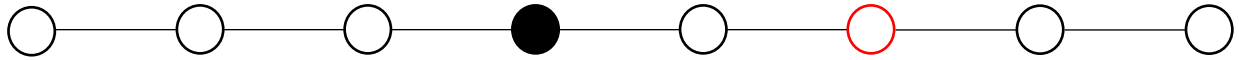
BFS(幅優先探索)

幅優先探索とは...

スタートに近い頂点から順番に探索していくアルゴリズム

単一始点から各頂点まで何辺でたどり着けるかが分かる





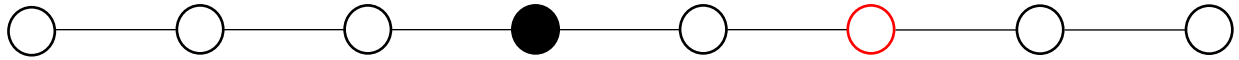
BFSの実装方法

- 手順1 始点sから各頂点xへの最短経路長を持つ配列(dist[x])とキューを用意(配列の要素を-1で初期化)

- 手順2 キューに始点sを追加し,dist[s] = 0にする。

- 手順3 キューが空になるまで次の操作を繰り返す。

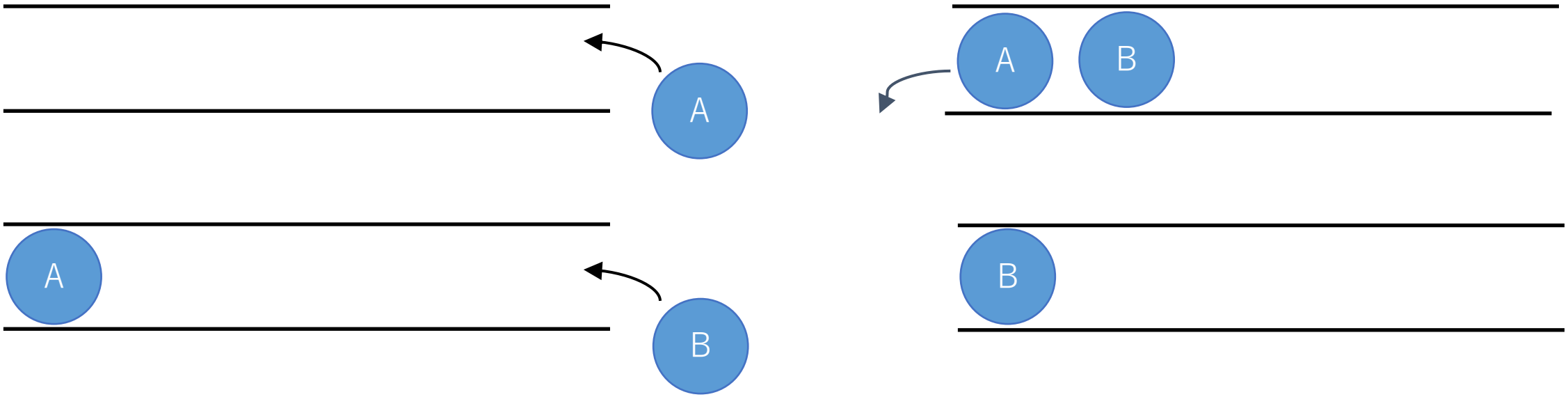
- キューの先頭要素posを取得し,キューから削除。
- posと隣接する全ての未確定の頂点nextに対して,
  dist[next] = dist[pos] +1に変更した後,キューにnextを追加。

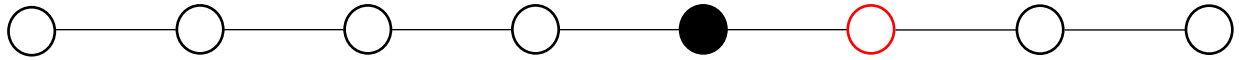


キューについて

- キュー(queue)とは

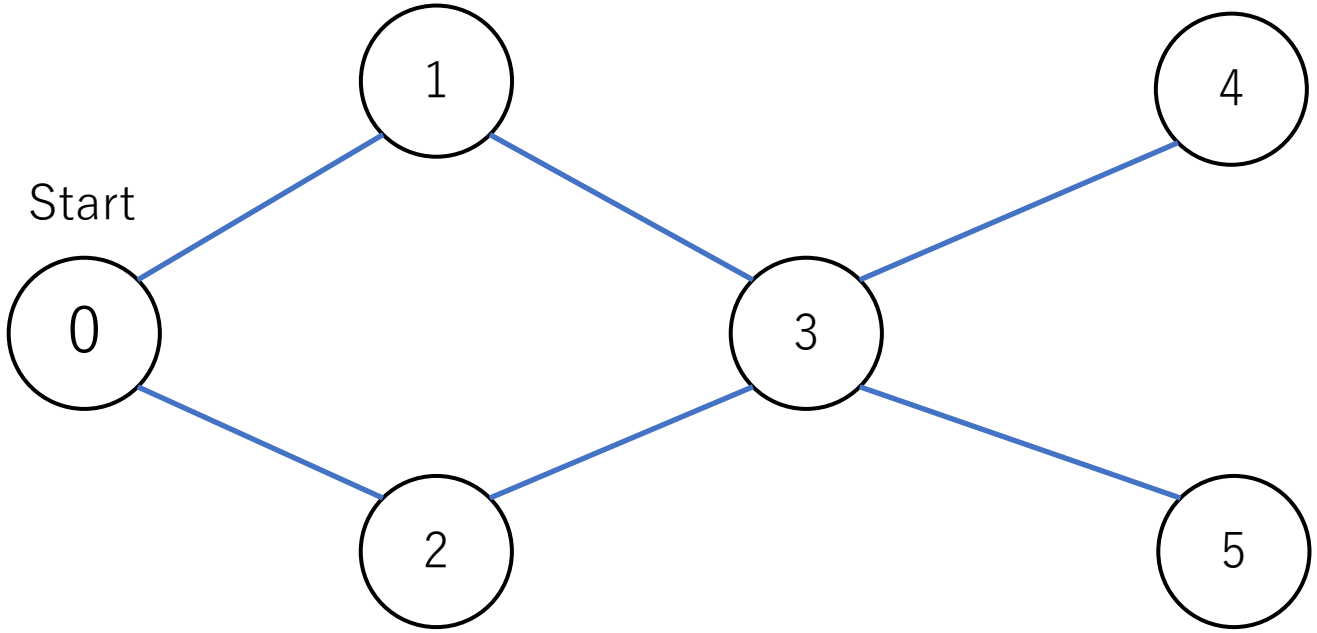
キュー :待ち行列の最後尾に要素を追加、先頭の要素取得、先頭の要素を削除がO(1)で行えるデータ構造





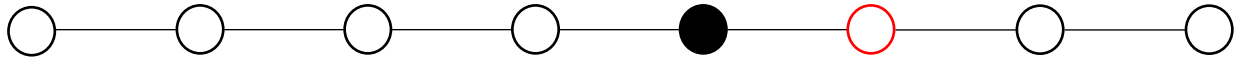
BFSの実装(1/7)

0	-1	-1	-1	-1	-1	配列
0	1	2	3	4	5	



queue

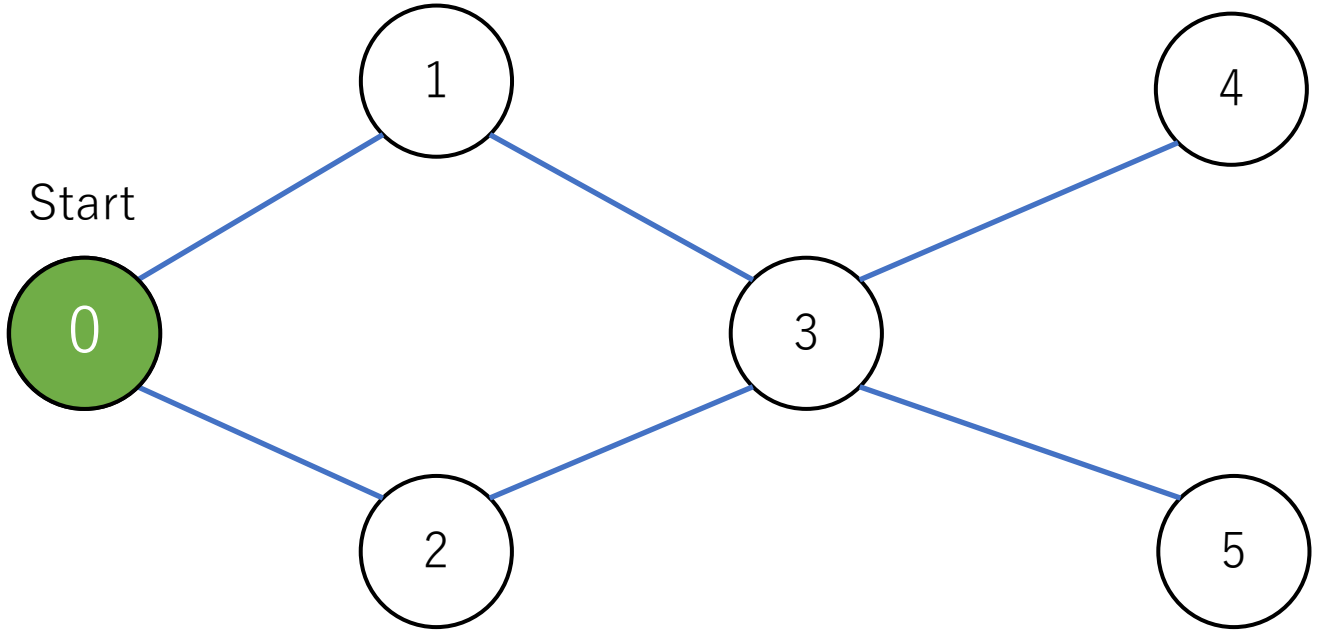
0



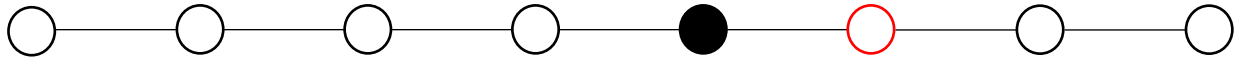
BFSの実装(2/7)

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

配列



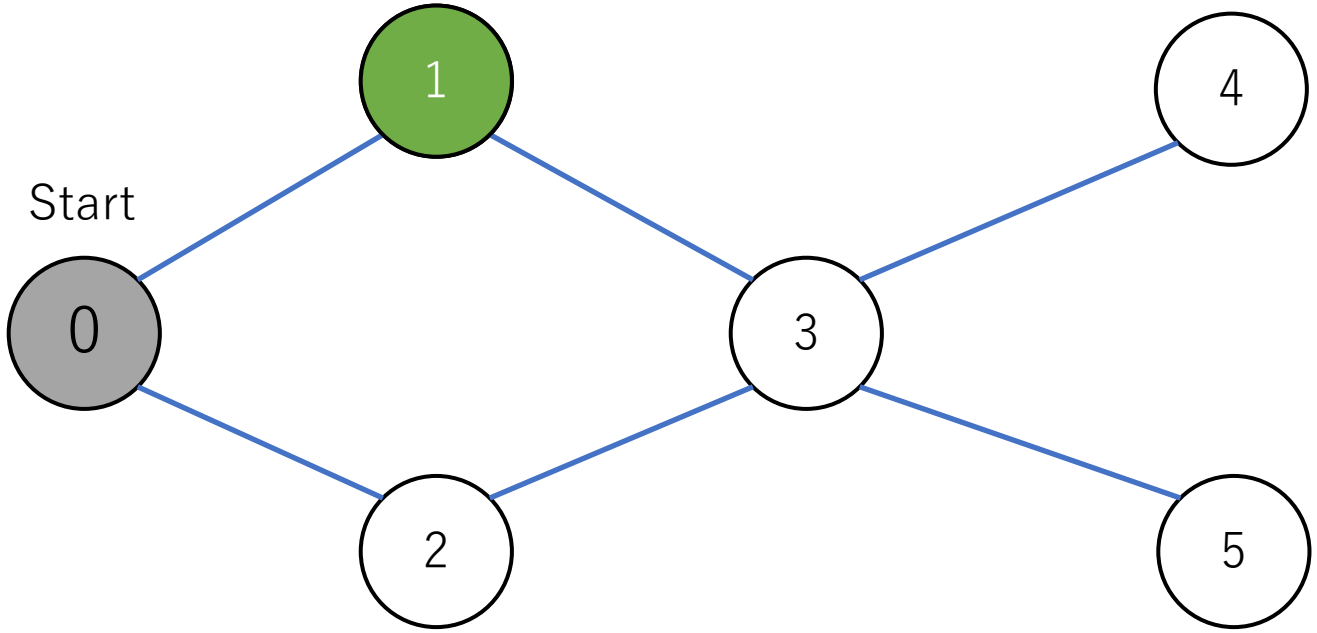
queue



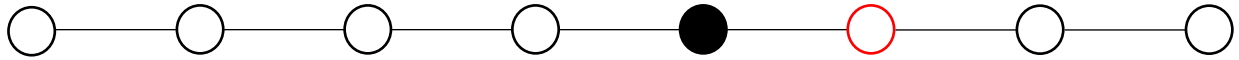
BFSの実装(3/7)

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

配列



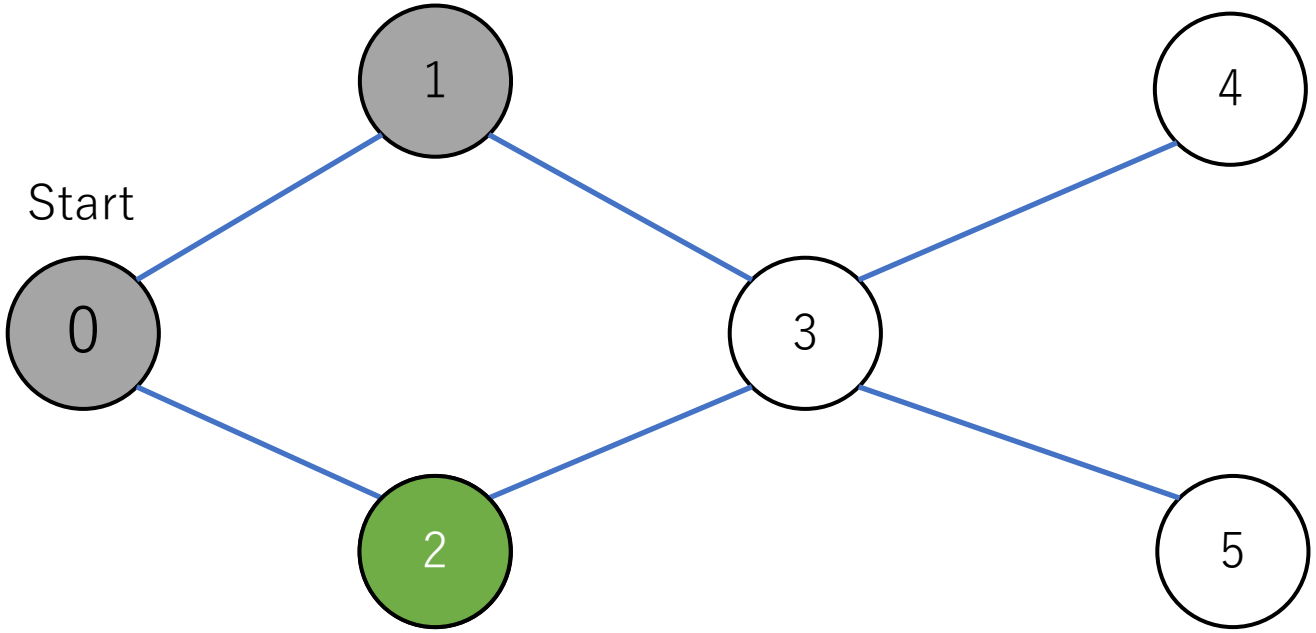
queue



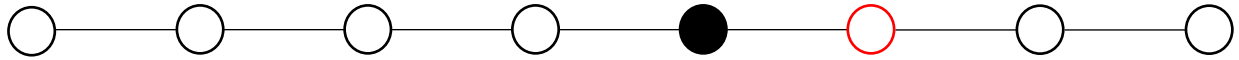
BFSの実装(4/7)

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

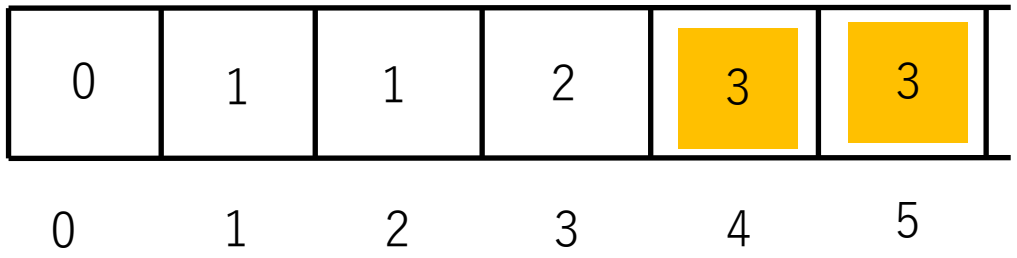
配列

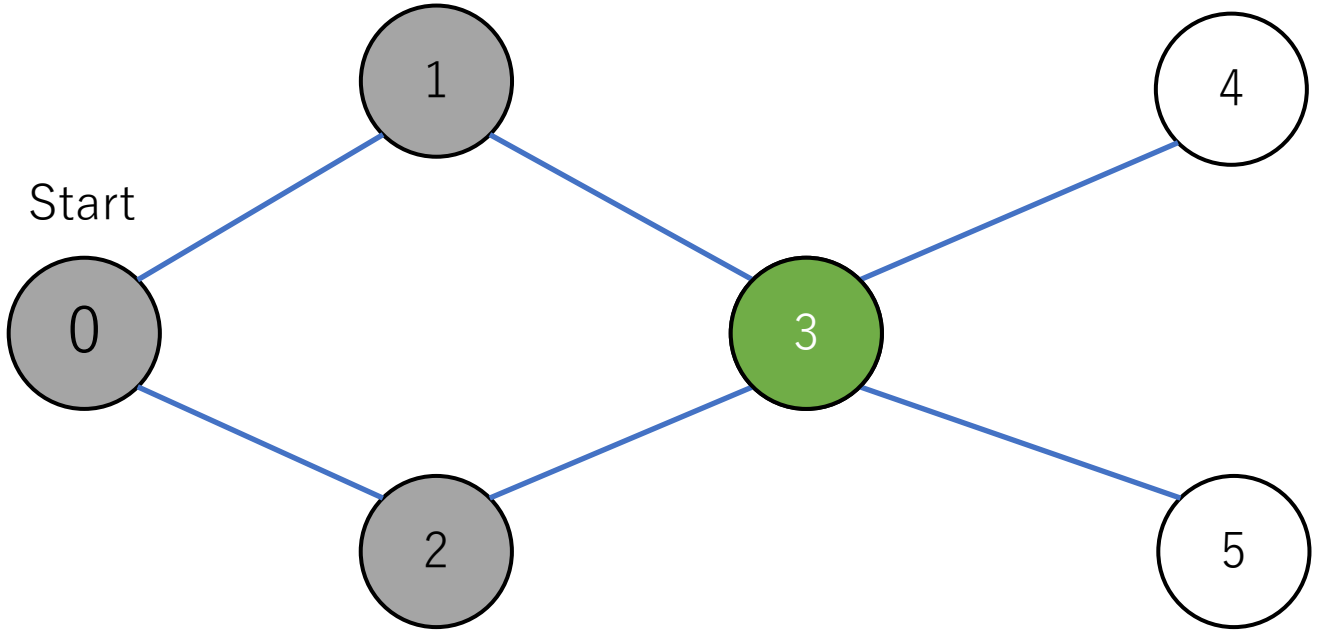


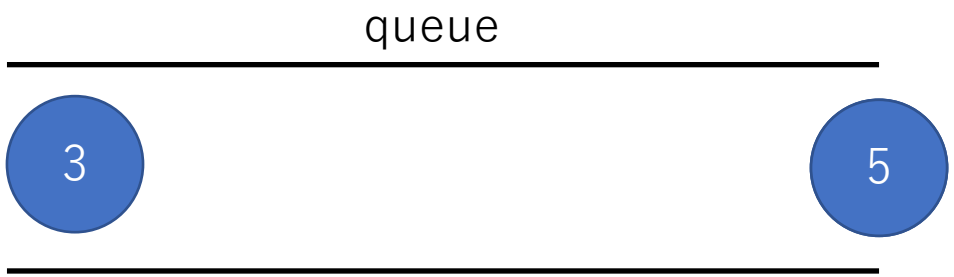
queue

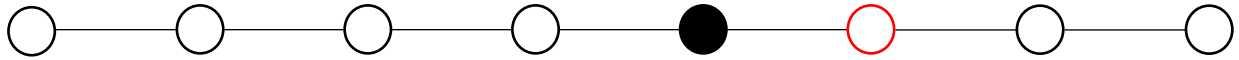


BFSの実装(5/7)





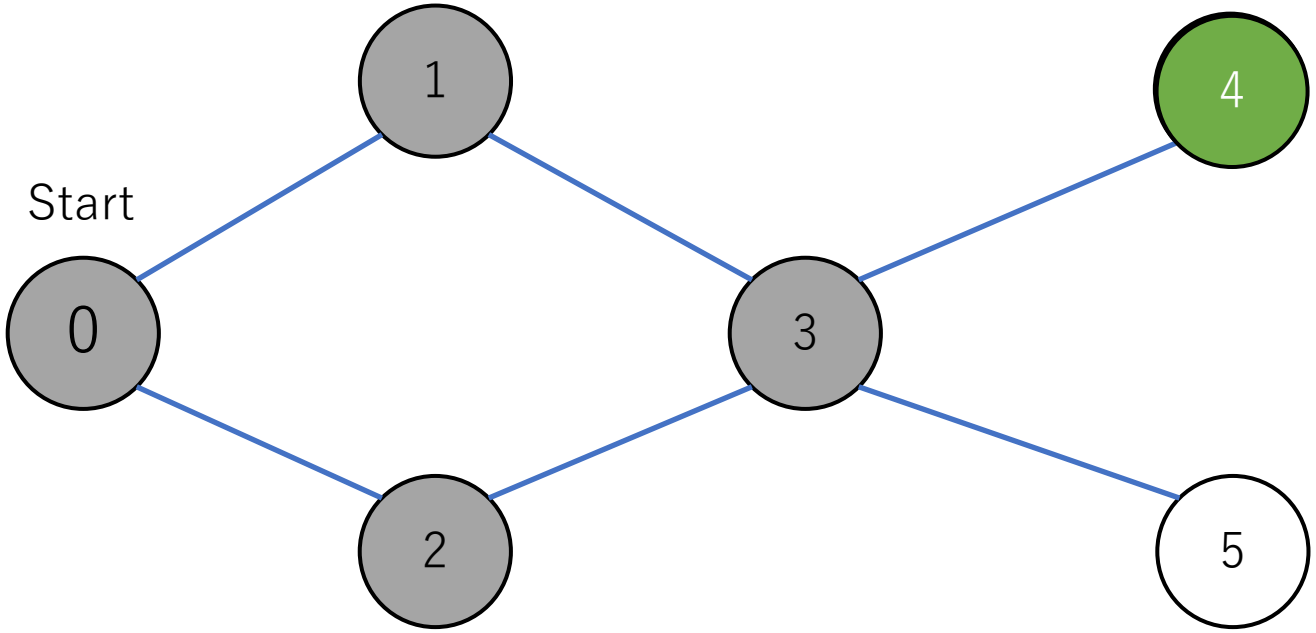




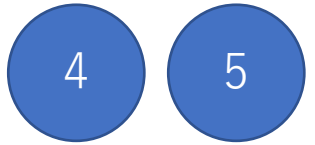
BFSの実装(6/7)

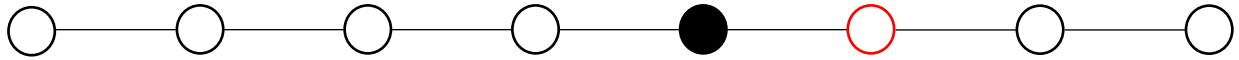
0	1	1	2	3	3
0	1	2	3	4	5

配列



queue

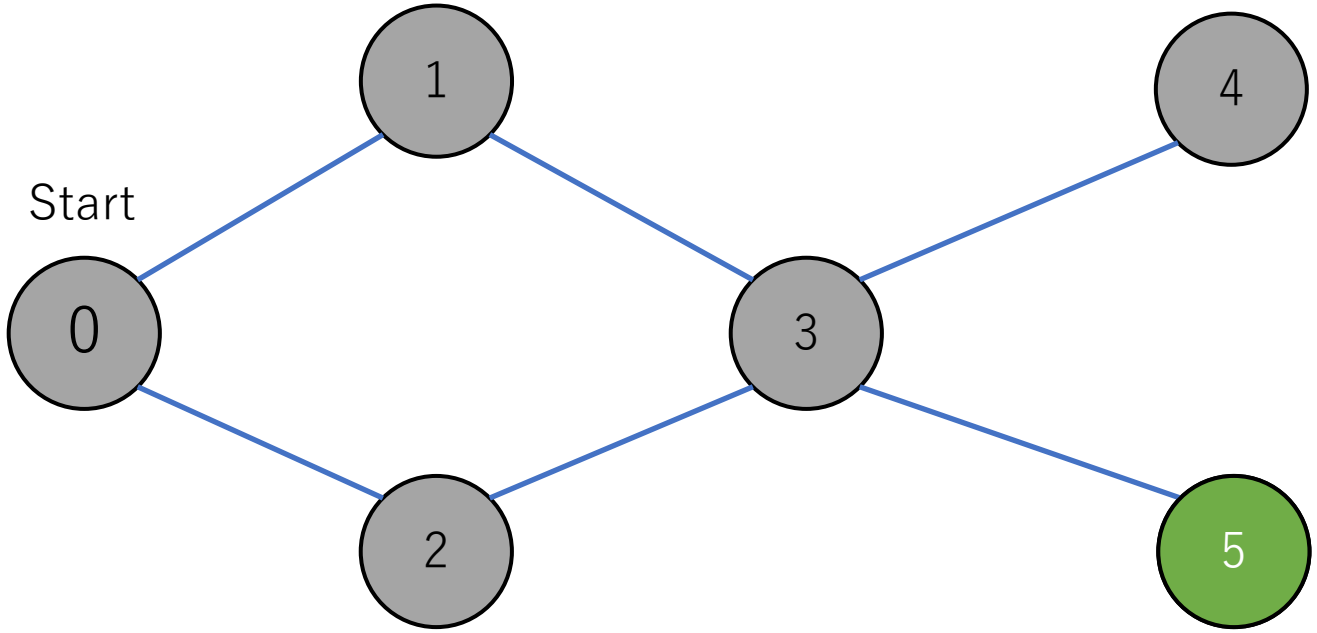




BFSの実装(7/7)

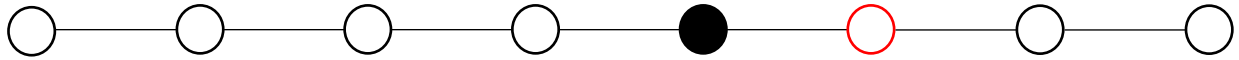
0	1	1	2	3	3
0	1	2	3	4	5

配列

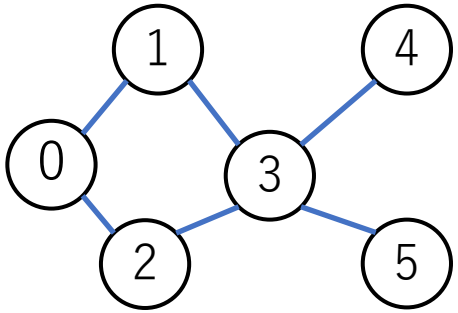


queue

5



BFSの正当性

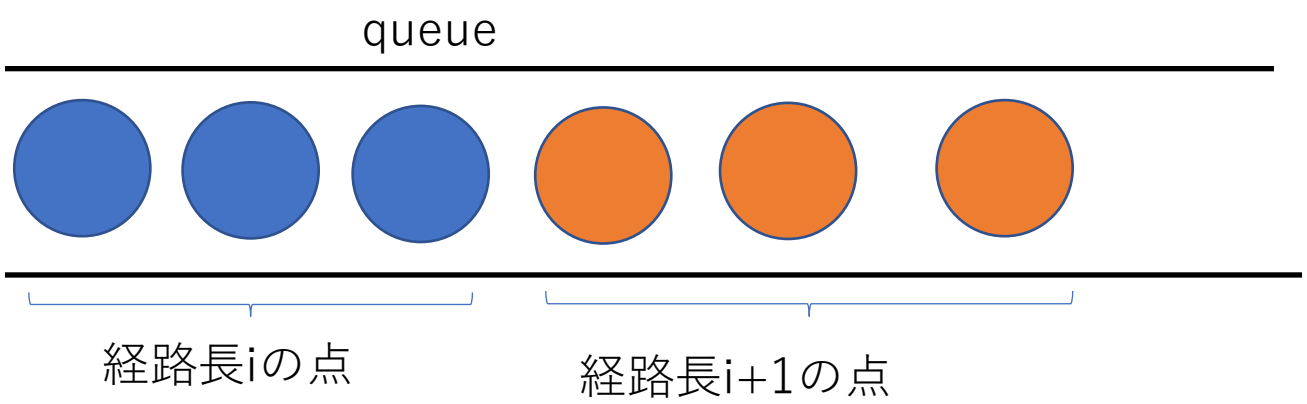


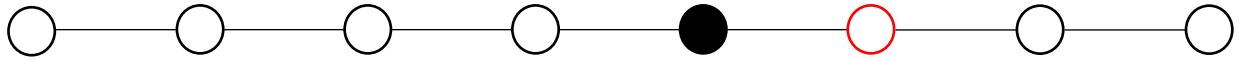
- 始点から近い点から見ていくだけで各頂点までの最短経路長が求まる

∵もしV_jまでの最短経路が0⇢V_i→V_jとなっている時に0 ⇢ V_k→V_jなるV_kが存在したとすると(ただしViとV_j, V_kとV_jは隣接),V_iとV_kの探索順序が矛盾

- キューを使ってうまくいく理由

  キューに最短経路長が小さい頂点から順に追加されるから



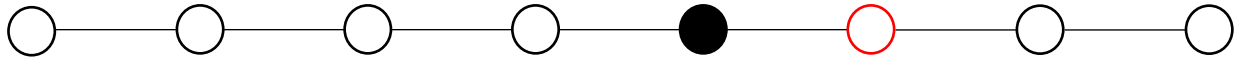


例題

A63 - Shortest Path 1: https://atcoder.jp/contests/tessoku-book/tasks/math_and_algorithm_an

- グラフの頂点数はN,辺の数はMである。
- i番目の辺は頂点A_iと頂点B_iを結ぶ。
- 頂点1から各頂点への最短経路長を求めたい。
- 入力形式

  N M
  A_1 B_1
  A_2 B_2
  ⋮
  A_M B_M



例題の実装

- 実装は2ステップ

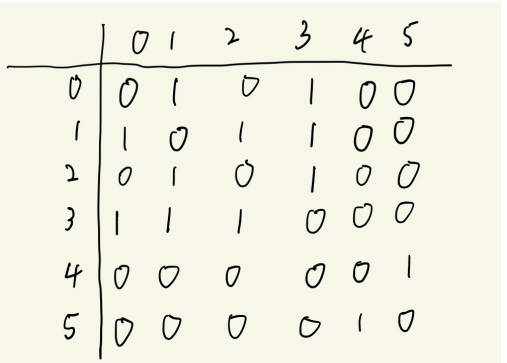
①グラフを受け取る

②BFSをする

①グラフを受け取る

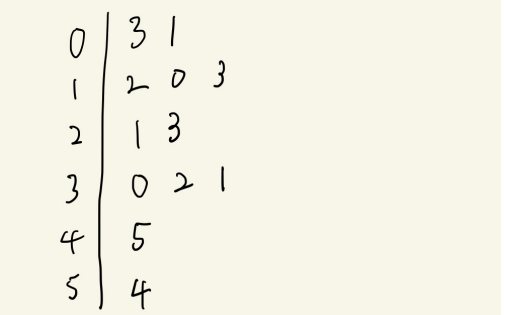
- グラフは隣接リストで受け取る！
- 0 originにしておくと扱いやすい

隣接行列→



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

隣接リスト→



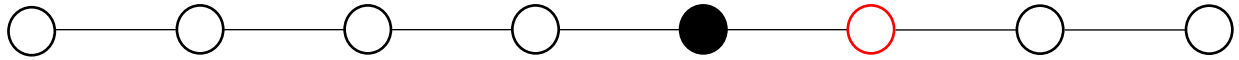
0	3 1
1	2 0 3
2	1 3
3	0 2 1
4	5
5	4

実装:C++

```cpp
int N,M;cin >> N >> M;
vector<vector<int>> G(N);      //隣接リスト
for(int i = 0;i < M;i++){
  int A,B;cin >> A >> B;
  A--;B--;                     // 0Indentにする
  G[A].push_back(B);
  G[B].push_back(A);
}
```

実装: Python

```python
N, M = map(int, input().split()) #入力
edges = [ list(map(int, input().split())) for i in range(M) ]

G = [ list() for i in range(N) ] # 隣接リストの作成
for A, B in edges:
  G[A-1].append(B-1)
  G[B-1].append(A-1)
```



②BFSをする

実装: C++

```cpp
//BFS
vector<int> dist(N,-1);          //訪れていない頂点は-1で初期化
queue<int> Q;
dist[0] = 0;                     //始点の始点からの距離は0
Q.push(0);

while(!Q.empty()){
  int pos = Q.front();           //キューの先頭の要素を取得
  Q.pop();                       //キューの先頭の要素を削除
  for(int i = 0;i < (int)G[pos].size();i++){
    int next = G[pos][i];
    if(dist[next] != -1)continue;//すでに発見されていた

    dist[next] = dist[pos]+1;
    Q.push(next);                //キューの最後尾に要素を追加
  }
}
```

実装: Python

```python
#BFS
dist = [ -1 ] * (N)
dist[0] = 0
Q = deque()
Q.append(0)

while len(Q) >= 1:
  # キュー Q の先頭要素を取り除き、その値を pos に代入する
  pos = Q.popleft()
  for next in G[pos]:
    if dist[next] == -1:
      dist[next] = dist[pos] + 1
      Q.append(next)
```

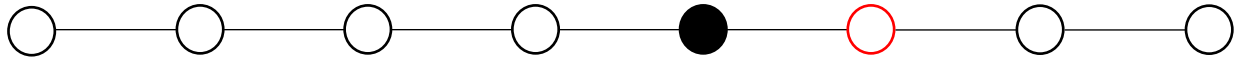
コード全体

←C++

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

int main(){
  int N,M;cin >> N >> M;
  vector<vector<int>> G(N);      //隣接リスト
  for(int i = 0;i < M;i++){
    int A,B;cin >> A >> B;
    A--;B--;                     // 0Indentにする
    G[A].push_back(B);
    G[B].push_back(A);
  }

  //BFS
  vector<int> dist(N,-1);        //訪れていない頂点は-1で初期化
  queue<int> Q;
  dist[0] = 0;                   //始点の始点からの距離は0
  Q.push(0);

  while(!Q.empty()){
    int pos = Q.front();         //キューの先頭の要素を取得
    Q.pop();                     //キューの先頭の要素を削除
    for(int i = 0;i < (int)G[pos].size();i++){
      int next = G[pos][i];
      if(dist[next] != -1)continue;//すでに発見されていた

      dist[next] = dist[pos]+1;
      Q.push(next);              //キューの最後尾に要素を追加
    }
  }

  for(int i = 0;i < N;i++){
    if(dist[i] == -1){           //たどり着けなかったということ
      cout << -1 << endl;
    }else{
      cout << dist[i] << endl;
    }
  }
}
```

Python→

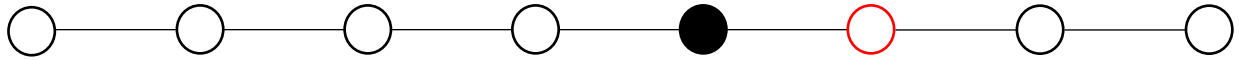
```python
from collections import deque

N, M = map(int, input().split()) #入力
edges = [ list(map(int, input().split())) for i in range(M) ]

G = [ list() for i in range(N) ] # 隣接リストの作成
for A, B in edges:
  G[A-1].append(B-1)
  G[B-1].append(A-1)

#BFS
dist = [ -1 ] * (N)
dist[0] = 0
Q = deque()
Q.append(0)

while len(Q) >= 1:
  # キュー Q の先頭要素を取り除き、その値を pos に代入する
  pos = Q.popleft()
  for next in G[pos]:
    if dist[next] == -1:
      dist[next] = dist[pos] + 1
      Q.append(next)

# 頂点 1 から各頂点までの最短距離を出力
for i in range(N):
  print(dist[i])
```



計算量について

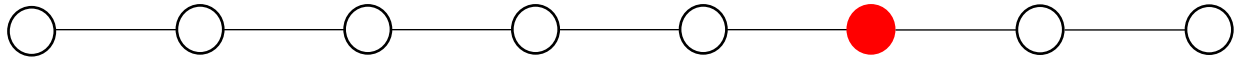
- 計算量は頂点数N,辺の数Mのとき$O(N+M)$

∵キューには各頂点が高々1回のみ格納される(N)。
　各辺は高々一回走査される(M)。



BFSまとめ

- 単一始点からグラフ上の各頂点への最短経路が求められる。
- 計算量はO(N+M)
- 辺の重みが異なるグラフではできない！



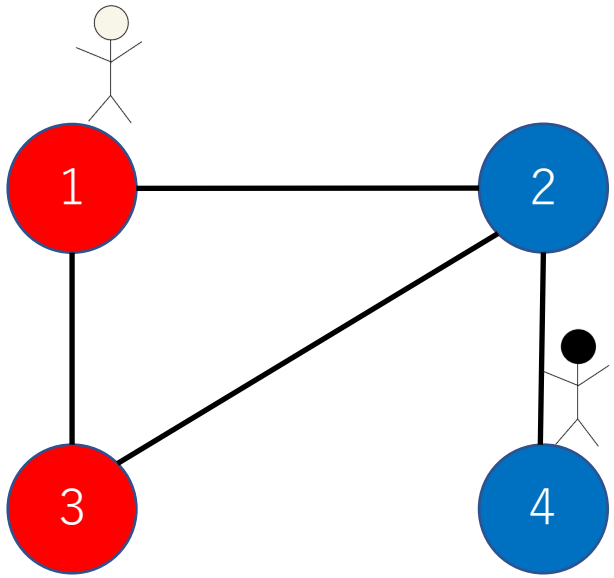
BFSの応用(頂点の持ち方を工夫する)

- E - Swap Places https://atcoder.jp/contests/abc289/tasks/abc289_e

- N頂点M辺の単純無向グラフが与えられる。
- 各頂点は赤か青のいずれかで塗られている。
- 頂点1と頂点Nにいる2人が, 隣接する頂点に移動を繰り返すことでそれぞれ頂点N,頂点１にいる状態にできるか。ただし２人の移動先の頂点の色は異なる必要がある。
- Nの総和,Mの総和はともに2000を超えない。

考察

- 簡単なケースで試してみる。

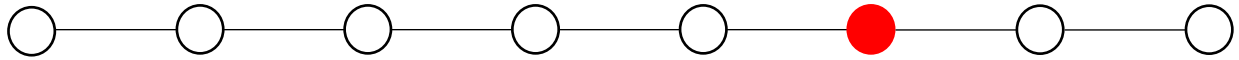


- 最短経路長では行けると限らない
- 同時にゴールしなくては行けないから２人の

移動回数は同じ

- １人の動き決めたらもう一方の動きは制限がかかる

- 制約的にO(N^2)とかO(M^2)もOK



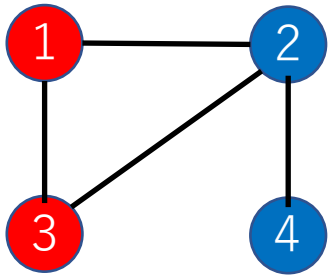
解き方

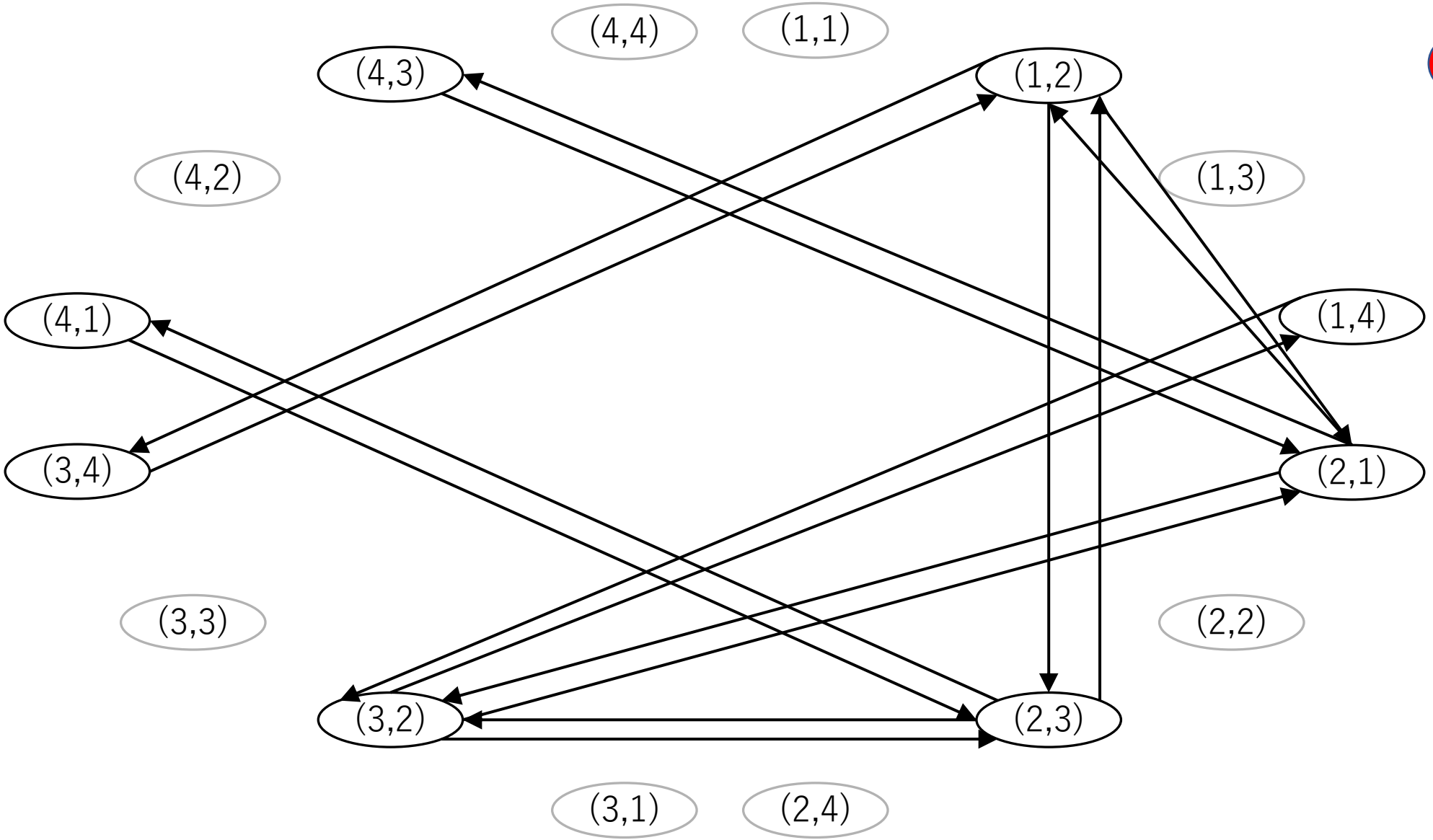
- 操作を繰り返して発生する状態の個数は
「Aさんが頂点1~頂点Nのどの頂点にいるか」×「Bさんが頂点1~頂点Nのどの頂点にいるか」
のN^2通りで抑えられるので

Aさんがいる頂点とBさんがいる頂点の組を頂点とするグラフを考える！

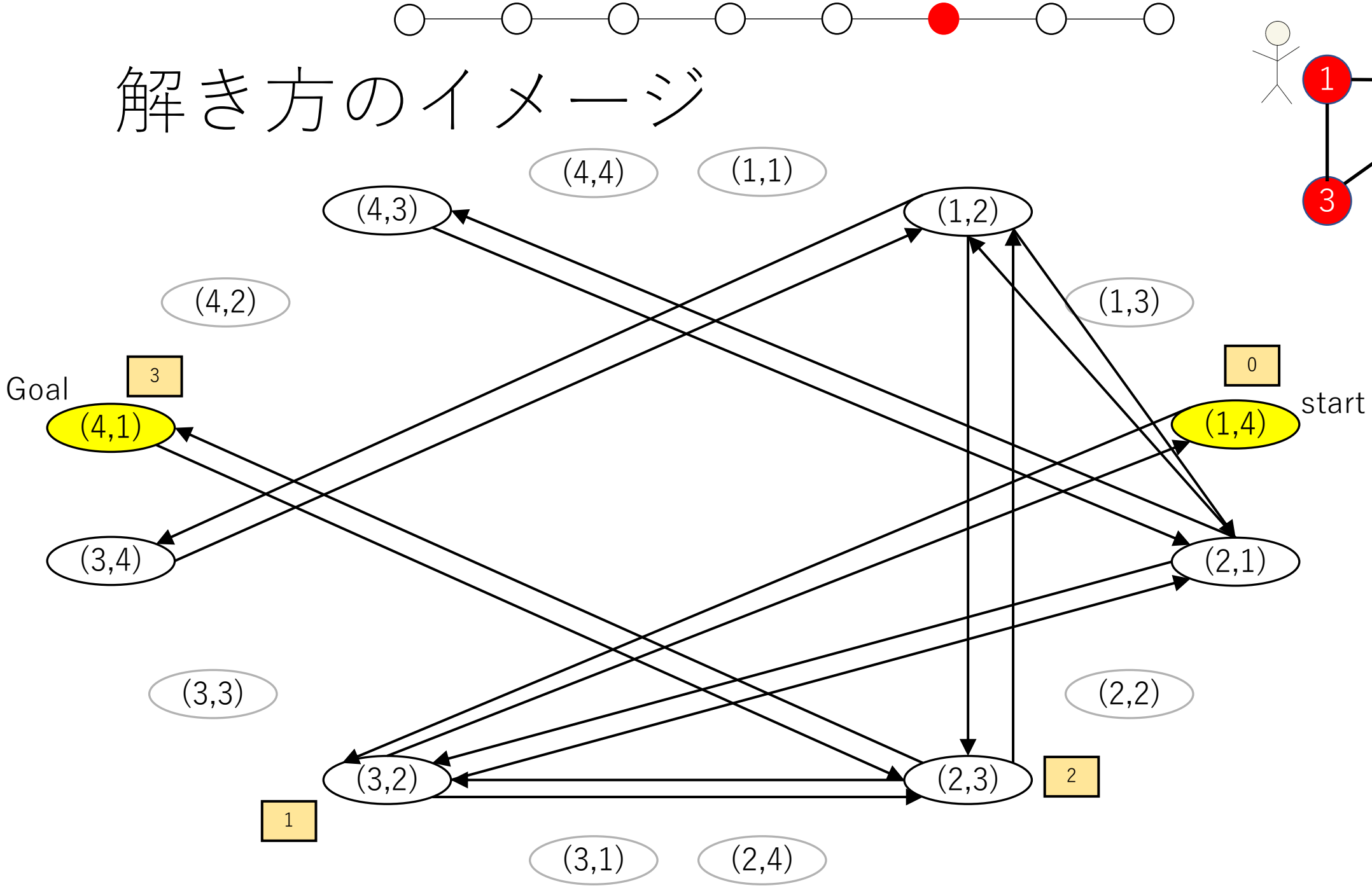
頂点(i,j)を頂点iにAさんがいて,頂点jにBさんがいる状態に対応する点とすると,求める答えは
頂点(1,N)から頂点(N,1)の最短経路長。

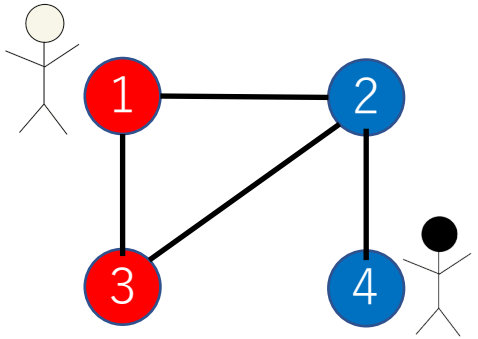
解き方のイメージ





解き方のイメージ

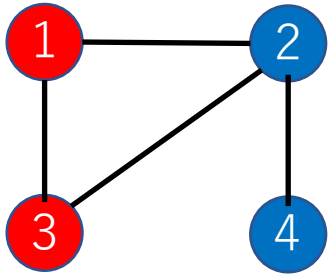




計算量の見積もり

- 計算量は$O(N^2 + M^2)$

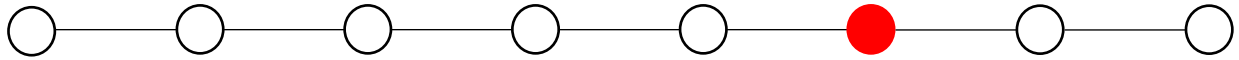
∵頂点数はN^2 ,辺数は${}_{2M}C_2 \fallingdotseq 2M^2$



２つの頂点の組を結ぶ辺は元の辺を２本使う。
例: (1,2) →(3,4)は 辺13と辺24を使用。

元の辺は無向辺なので2方向から使われる。
辺13は(1を含む頂点組)→(3を含む頂点組) と (3を含む頂点組)→(1を含む頂点組) で使われる。

詳しい話は公式の解説で.. https://atcoder.jp/contests/abc289/editorial/5726



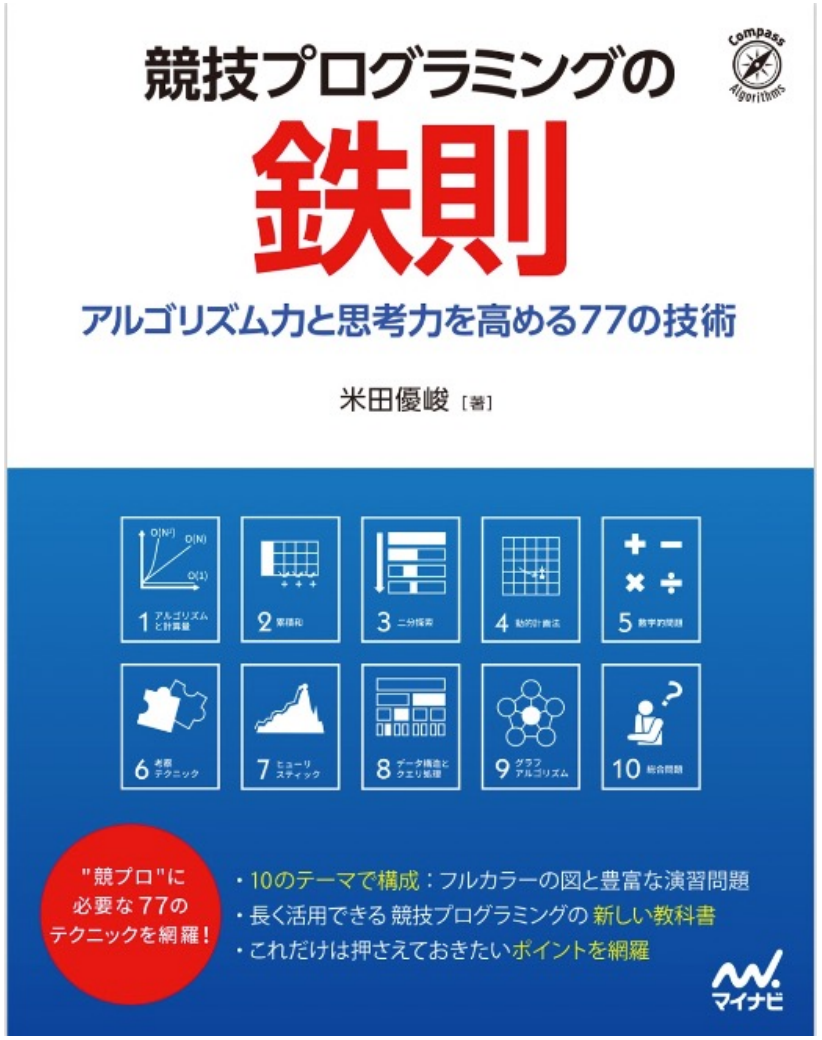
この問題のポイント(まとめ)

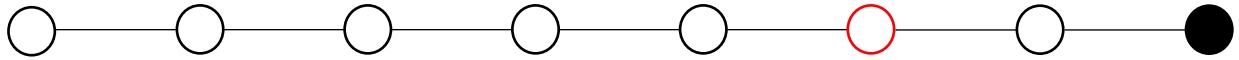
- ある状態を点にして考える。
(Bit使うものが多い)

- 制約が小さいのがポイントだった。
($N^2 \leqq 10^6$)

鉄則本の紹介

- 有名なレッドコーダーの人が執筆(E869120)
- レート0~1500の人におすすめ
- カラーでとてもわかり易い



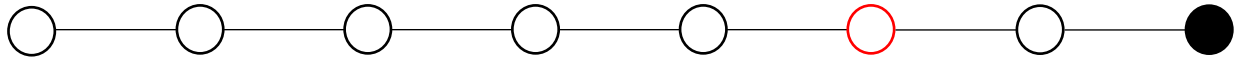


BFSの問題1

- Breadth First Search

https://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ALDS1_11_C&lang=ja

入出力の形式が違うだけ。確認用。



BFSの問題2

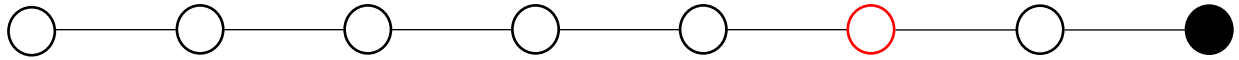
- A - 幅優先探索　https://atcoder.jp/contests/atc002/tasks/abc007_3

グリット上のBFS。

この形式も頻出なのでなれておきたい。

４方向の探索は

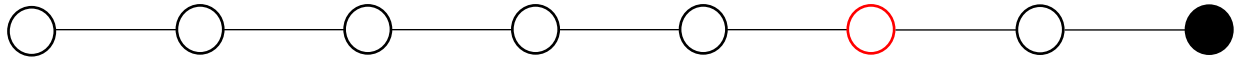
vx = {0,1,0,-1},vy = {1,0,-1,0}　と２つの配列を用意しておくと便利



BFSの問題３

- D - Grid Repainting　https://atcoder.jp/contests/abc088/tasks/abc088_d

応用問題。



BFS＋点の持ち方(bit)の問題

- A70 – Lanterns https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_br

制約に注目！