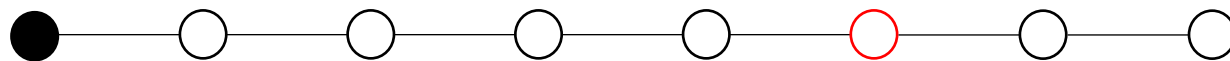


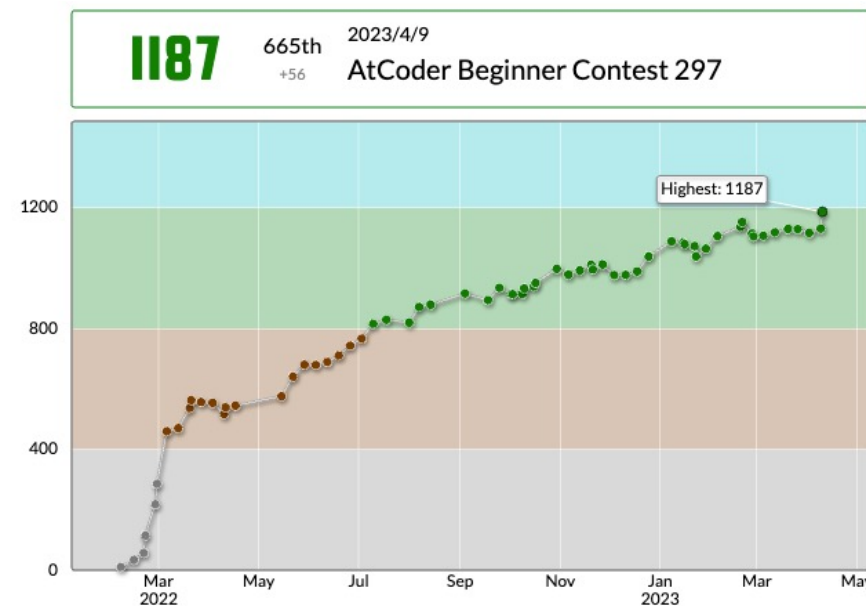
BFS(幅優先探索)

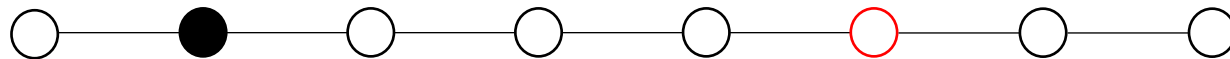
BFSを実装できるようになる



自己紹介

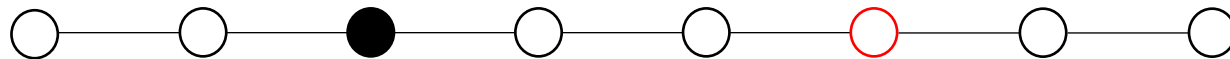
- 井桁 広翔(いげた ひろと)
- B3 情報科学科
- Twitter @igeeeeeee_
- 趣味：ゲーム





目次

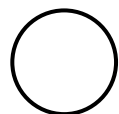
- 自己紹介
- グラフ理論の用語確認
- BFSとは
- BFSの実装
- **BFSの応用**
- 鉄則本の紹介
- 練習問題



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)

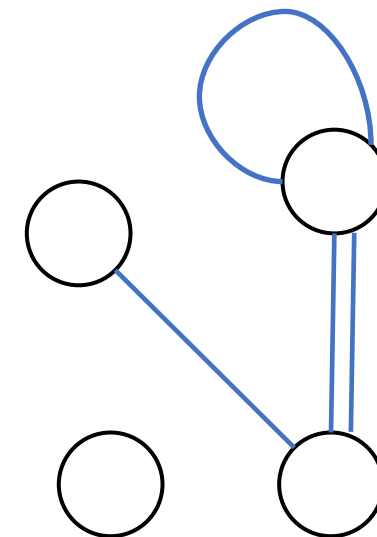


- 辺(arc, edge)

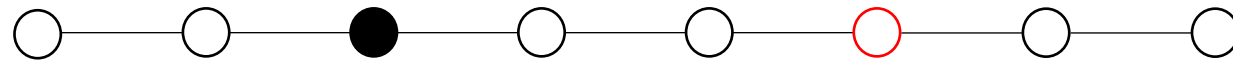
- 無向辺 : 向きのない辺



- 有向辺 : 向きのある辺



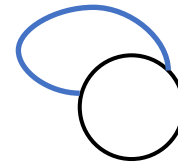
- 無向辺のグラフは無向グラフ、有向辺のグラフは有向グラフ



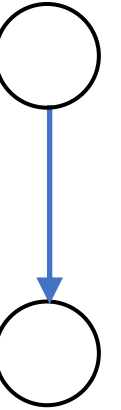
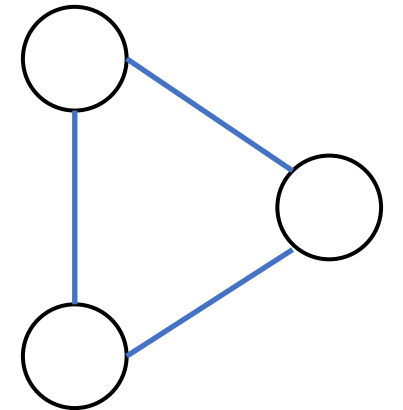
グラフの用語確認 2

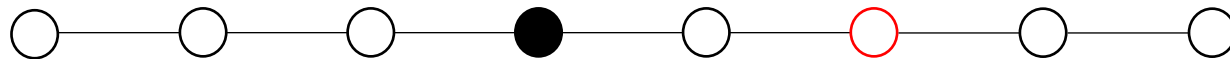
- 頂点の次数 : 頂点から出ている辺の本数
- 有向辺の時は入次数、出次数に分かれる

- 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)

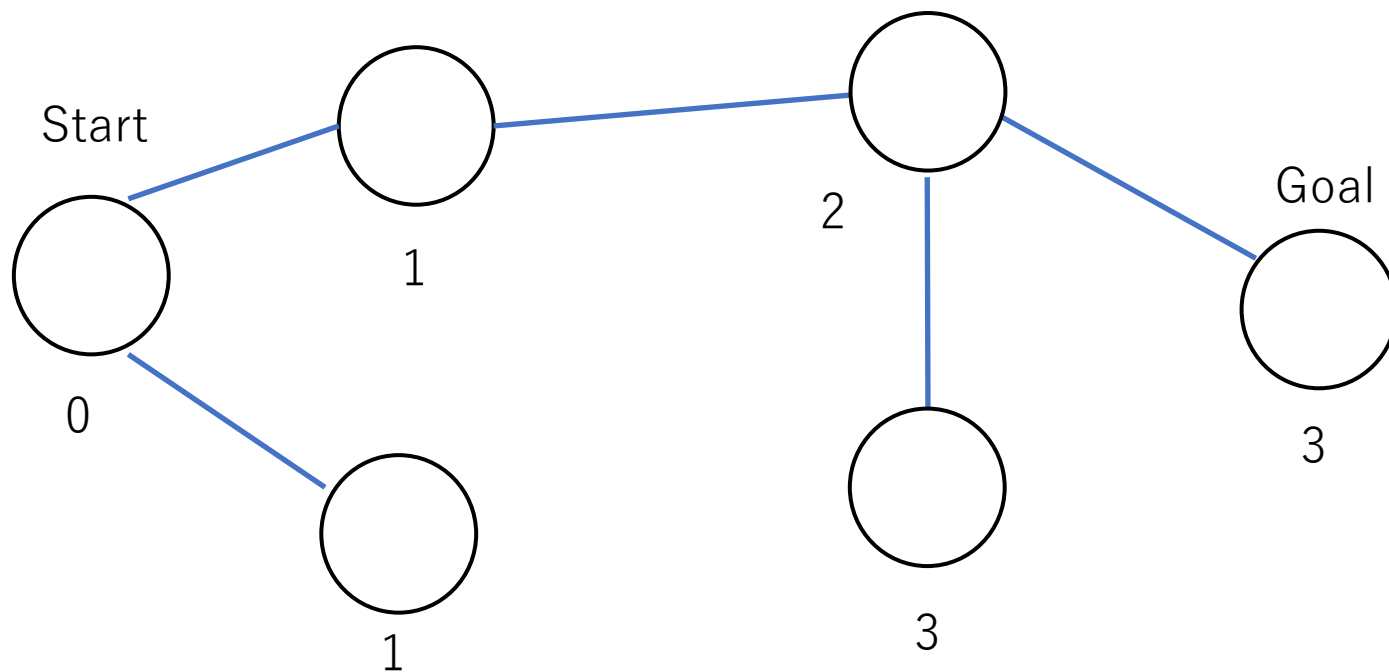


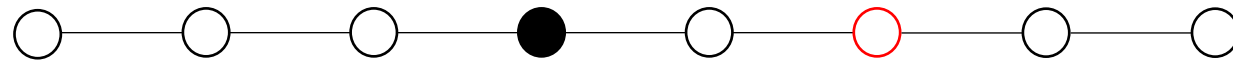


BFS(幅優先探索)

幅優先探索とは...

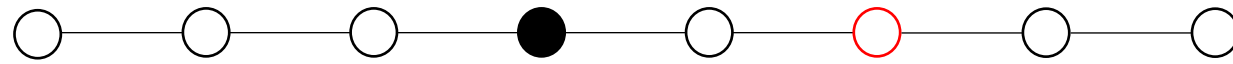
スタートに近い頂点から順番に探索していくアルゴリズム
単一起点から各頂点まで何辺でたどり着けるかが分かる





BFSの実装方法

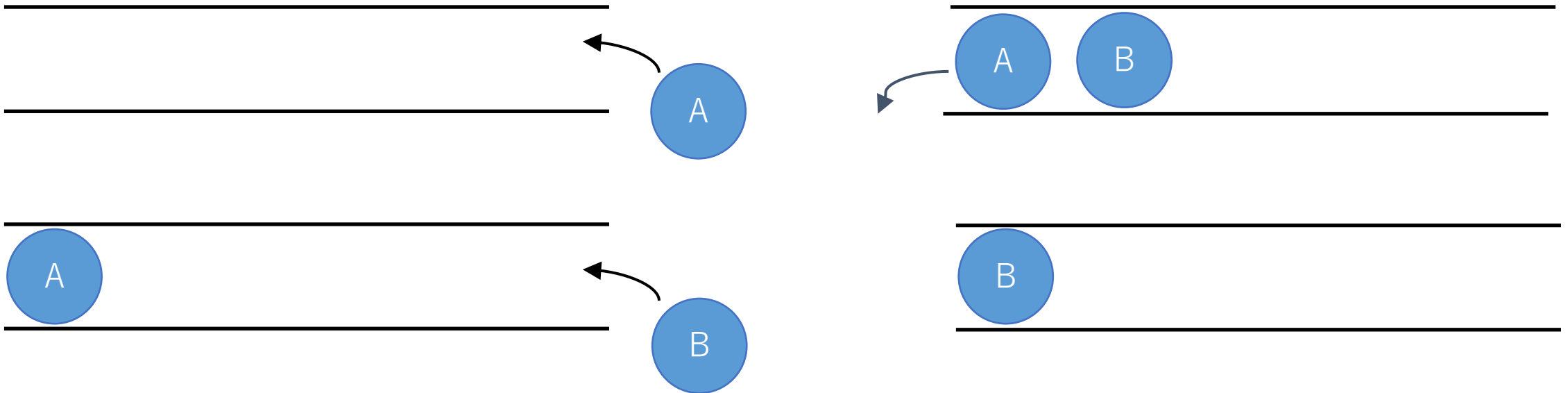
- 手順1 始点sから各頂点xへの最短経路長を持つ配列($\text{dist}[x]$)とキューを用意(配列の要素を-1で初期化)
- 手順2 キューに始点sを追加し, $\text{dist}[s] = 0$ にする。
- 手順3 キューが空になるまで次の操作を繰り返す。
 - キューの先頭要素posを取得し,キューから削除。
 - posと隣接する全ての未確定の頂点nextに対して,
 $\text{dist}[\text{next}] = \text{dist}[\text{pos}] + 1$ に変更した後,キューにnextを追加。

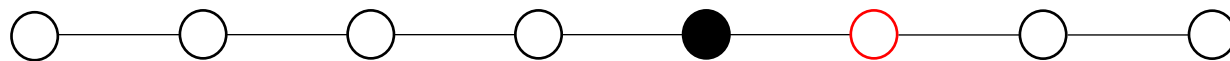


キューについて

- 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



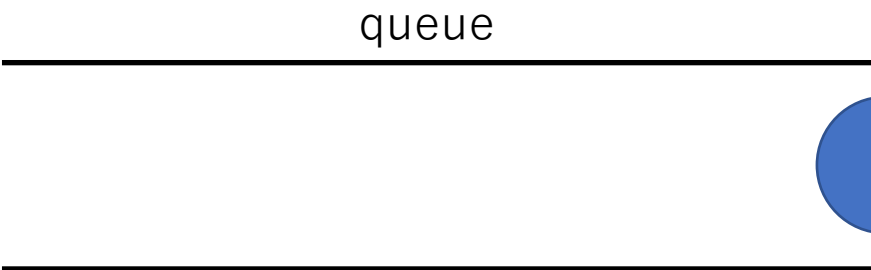
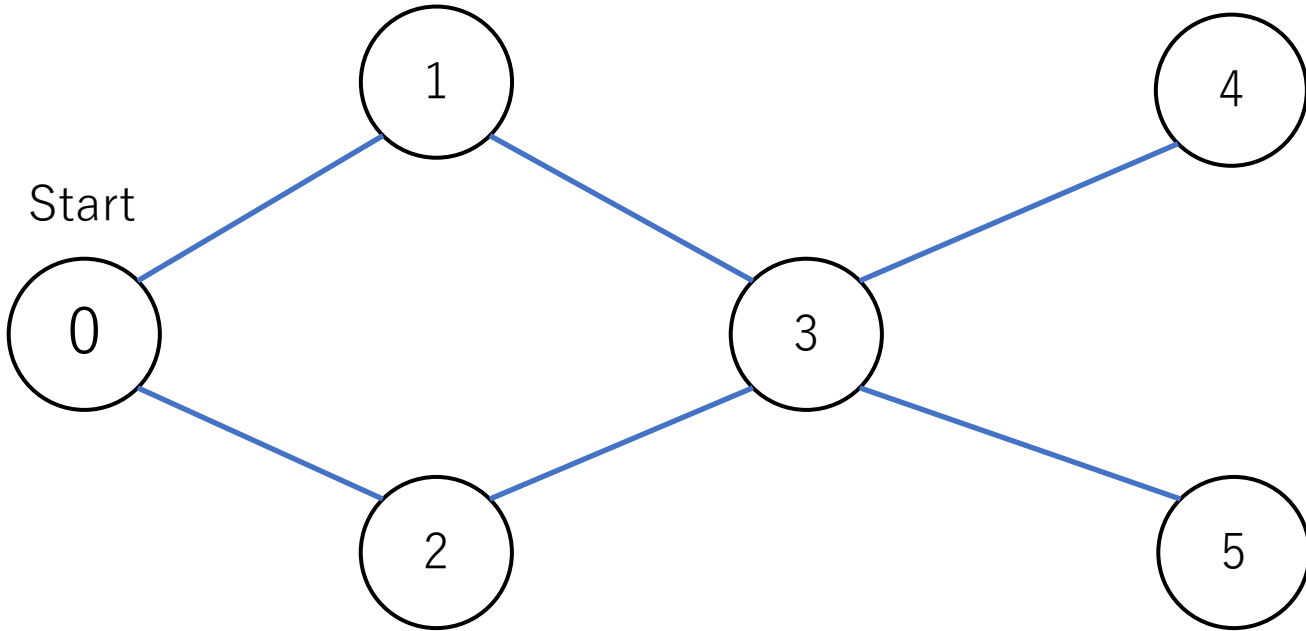


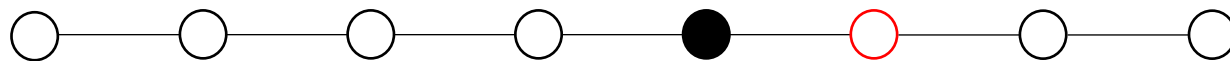
BFSの実装(1/7)

0	-1	-1	-1	-1	-1
---	----	----	----	----	----

配列 dist

0 1 2 3 4 5

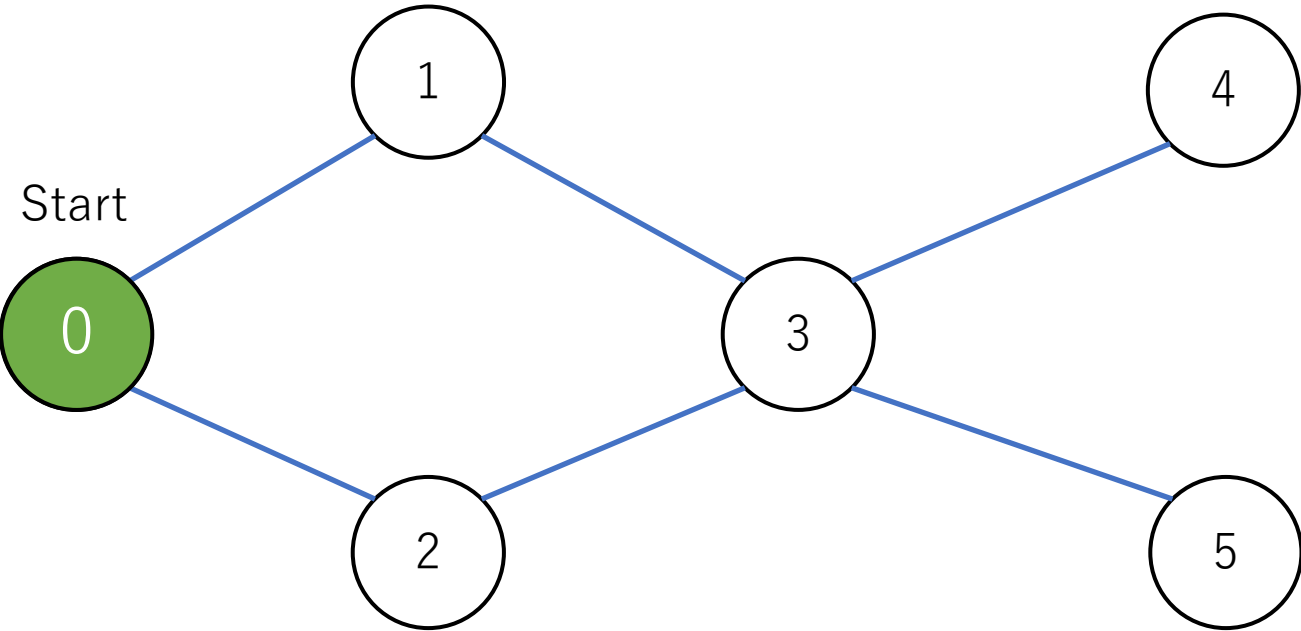


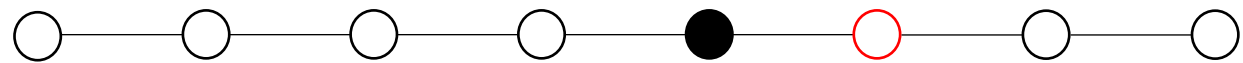


BFSの実装(2/7)

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

配列 dist

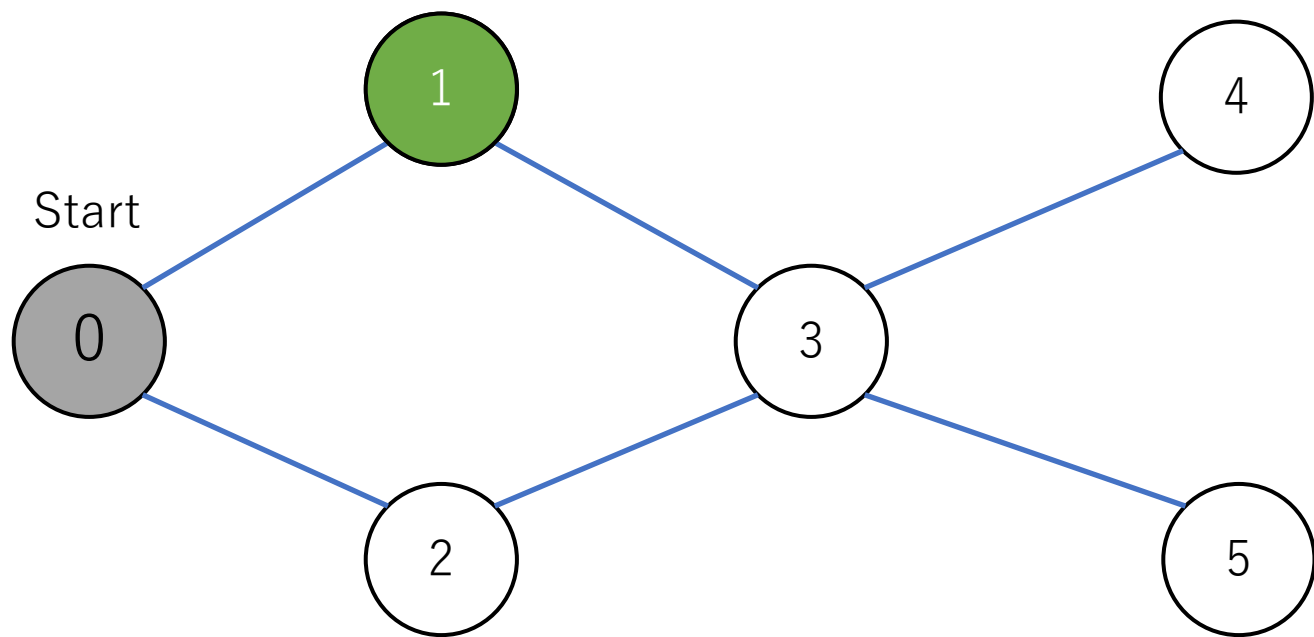


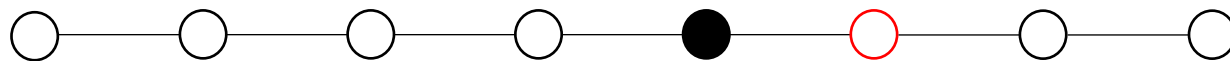


BFSの実装(3/7)

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

配列 dist

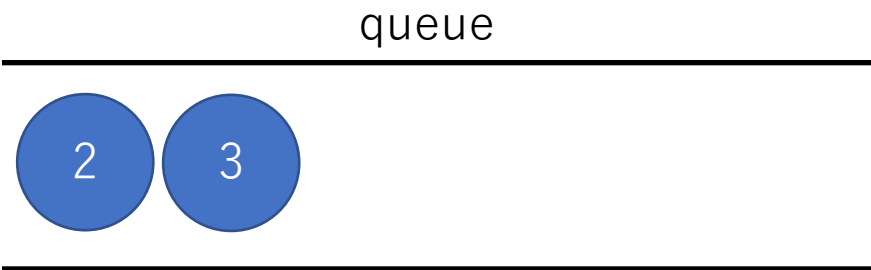
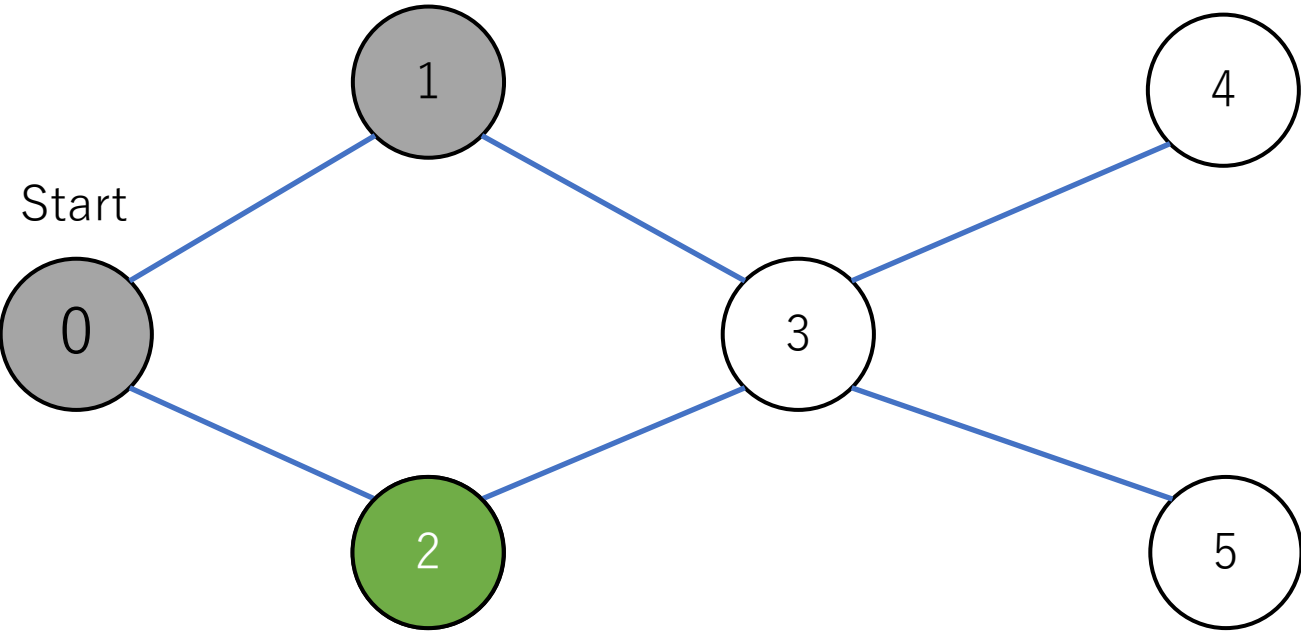


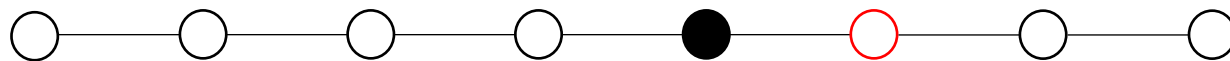


BFSの実装(4/7)

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

配列 dist

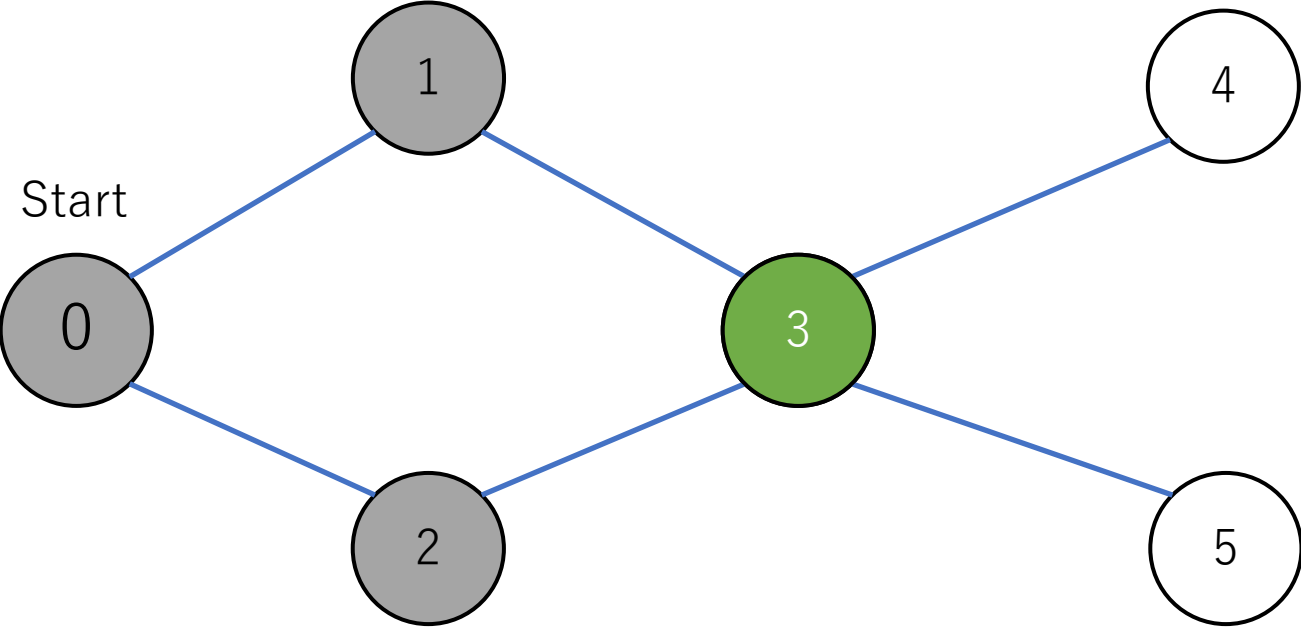


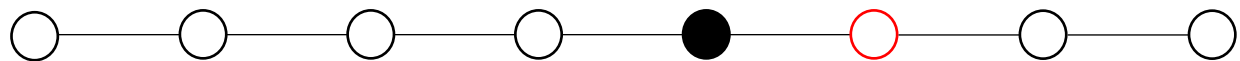


BFSの実装(5/7)

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

配列 dist

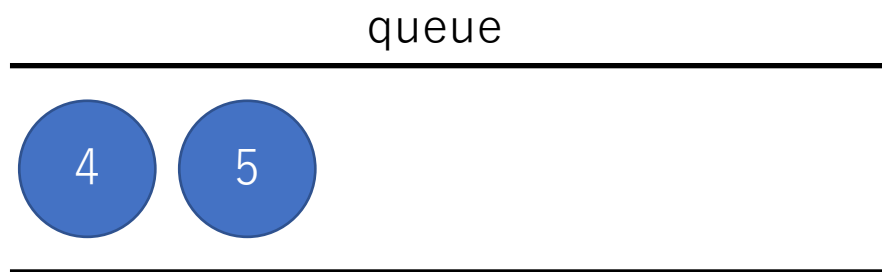
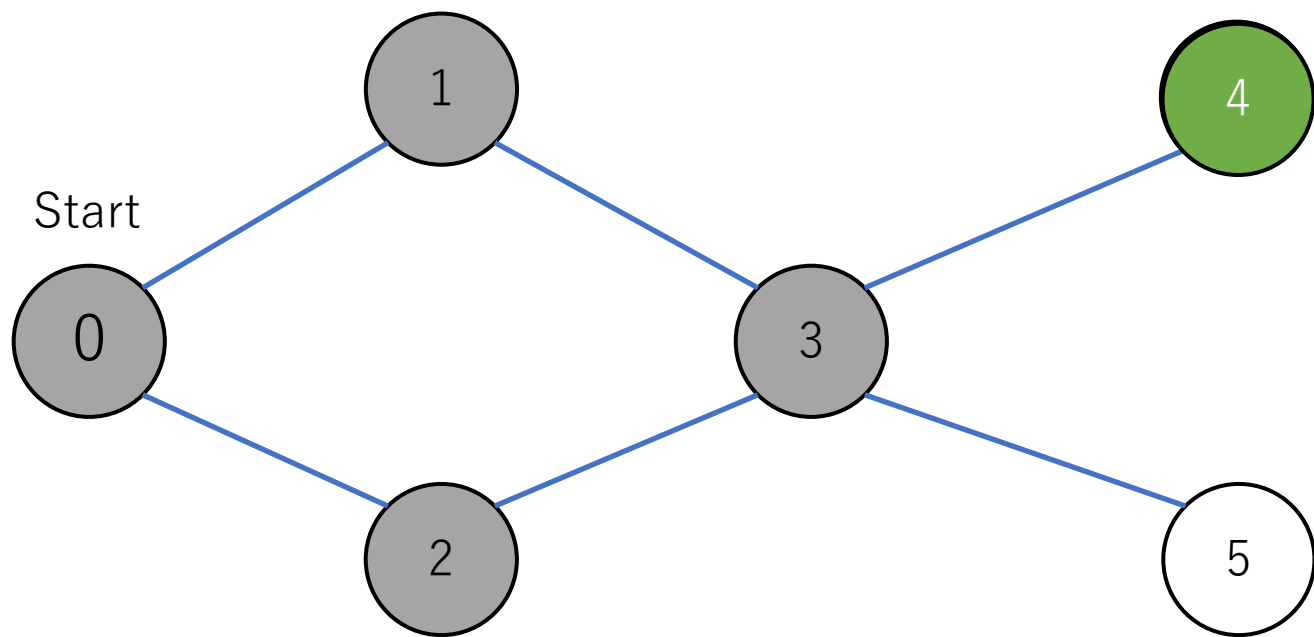


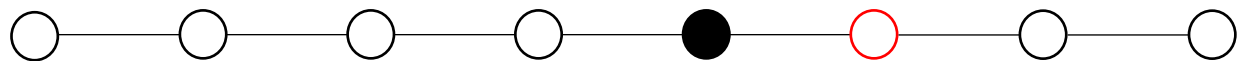


BFSの実装(6/7)

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

配列 dist

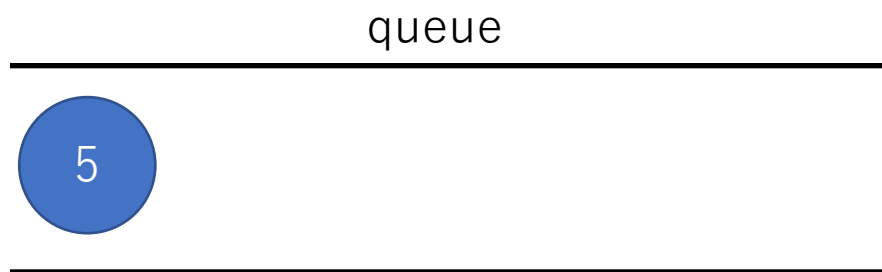
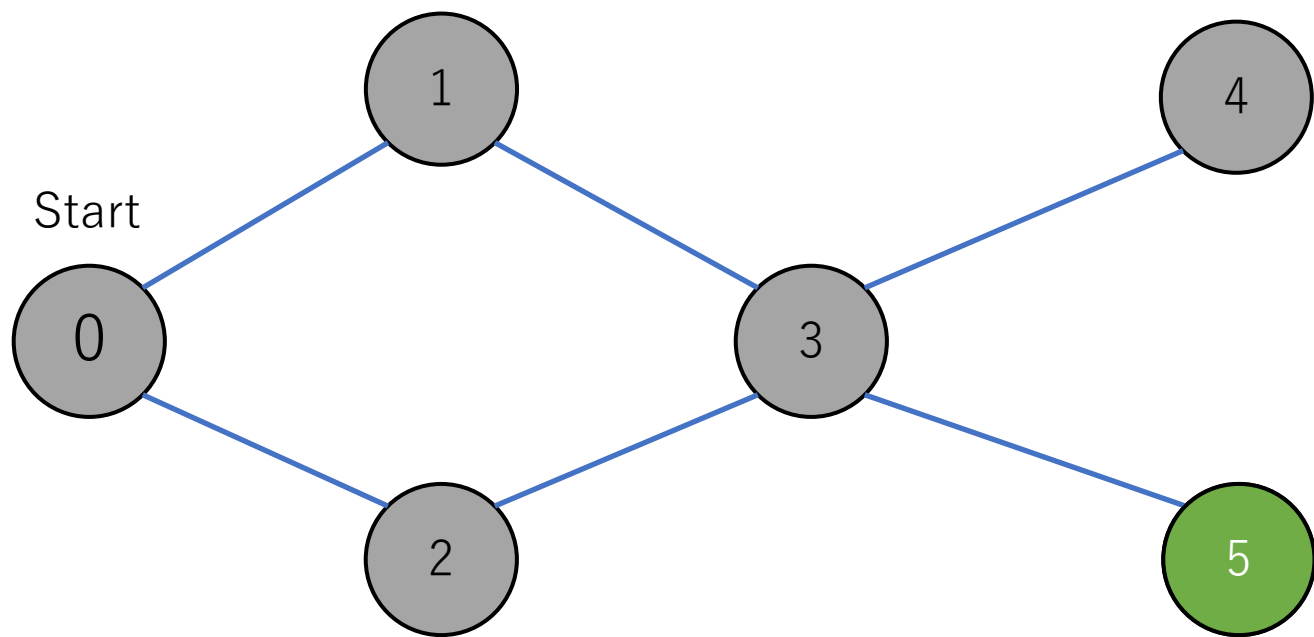


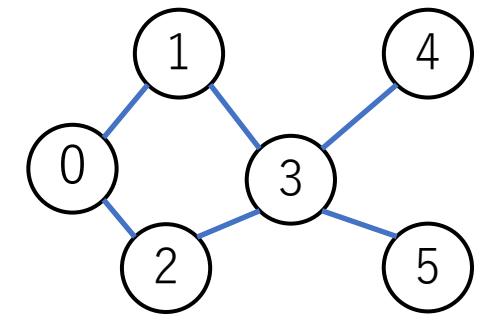
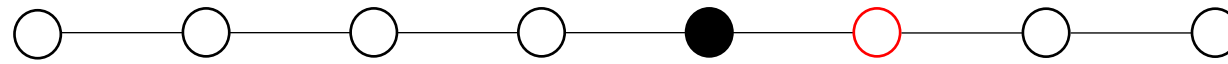


BFSの実装(7/7)

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

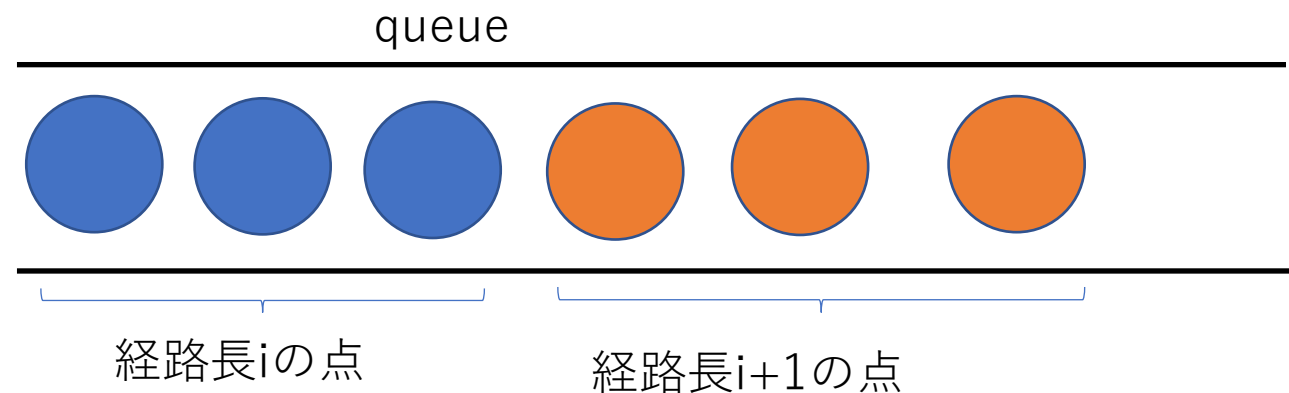
配列 dist

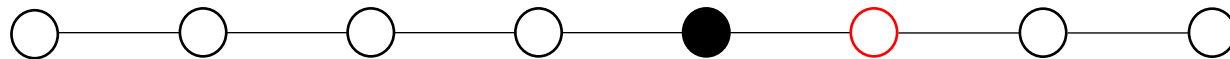




BFSの正当性

- 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
 \because もし V_j までの最短経路が $0 \rightarrow V_i \rightarrow V_j$ となっている時に $0 \rightarrow V_k \rightarrow V_j$ なる V_k が存在したとすると (ただし V_i 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

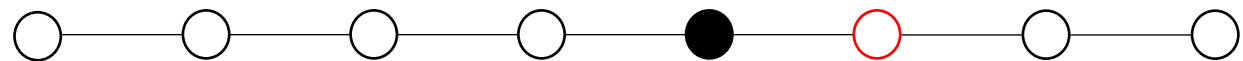
N M

A_1 B_1

A_2 B_2

\vdots

A_M B_M

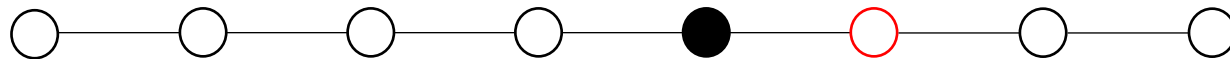


例題の実装

- 実装は2ステップ

① グラフを受け取る

② BFS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

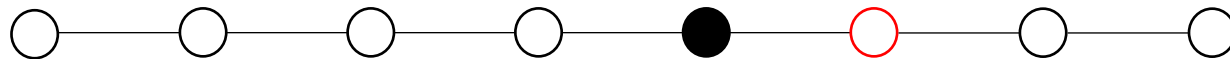
実装:C++

```
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--; // 0Indexにする
    G[A].push_back(B);
    G[B].push_back(A);
}
```

実装: 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++

```
//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

```
#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)
```

```

1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 using namespace std;
5
6 int main(){
7     int N,M;cin >> N >> M;
8     vector<vector<int>> G(N); //隣接リスト
9     for(int i = 0;i < M;i++){
10         int A,B;cin >> A >> B;
11         A--;B--; // 0Indexにする
12         G[A].push_back(B);
13         G[B].push_back(A);
14     }
15
16     //BFS
17     vector<int> dist(N,-1); //訪れていない頂点は-1で初期化
18     queue<int> Q;
19     dist[0] = 0; //始点の始点からの距離は0
20     Q.push(0);
21
22     while(!Q.empty()){
23         int pos = Q.front(); //キューの先頭の要素を取得
24         Q.pop(); //キューの先頭の要素を削除
25         for(int i = 0;i < (int)G[pos].size();i++){
26             int next = G[pos][i];
27             if(dist[next] != -1)continue;//すでに発見されていた
28
29             dist[next] = dist[pos]+1;
30             Q.push(next); //キューの最後尾に要素を追加
31         }
32     }
33
34     for(int i = 0;i < N;i++){
35         if(dist[i] == -1){ //たどり着けなかったということ
36             cout << -1 << endl;
37         }else{
38             cout << dist[i] << endl;
39         }
40     }
41 }

```

コード全体

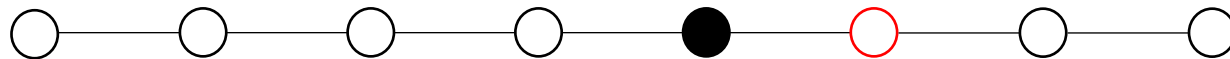
←C++

Python→

```

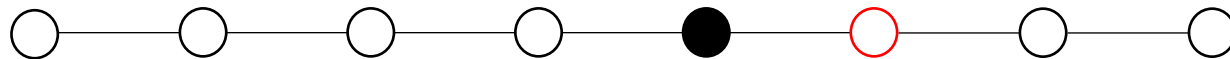
1 from collections import deque
2
3 N, M = map(int, input().split()) #入力
4 edges = [ list(map(int, input().split())) for i in range(M) ]
5
6 G = [ list() for i in range(N) ] # 隣接リストの作成
7 for A, B in edges:
8     G[A-1].append(B-1)
9     G[B-1].append(A-1)
10
11 #BFS
12 dist = [ -1 ] * (N)
13 dist[0] = 0
14 Q = deque()
15 Q.append(0)
16
17 while len(Q) >= 1:
18     # キュー Q の先頭要素を取り除き、その値を pos に代入する
19     pos = Q.popleft()
20     for next in G[pos]:
21         if dist[next] == -1:
22             dist[next] = dist[pos] + 1
23             Q.append(next)
24
25 # 頂点 1 から各頂点までの最短距離を出力
26 for i in range(N):
27     print(dist[i])

```



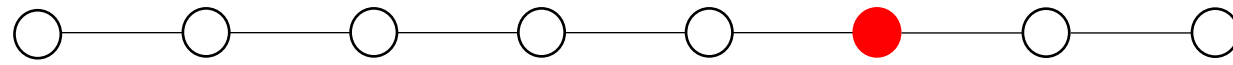
計算量について

- 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)。



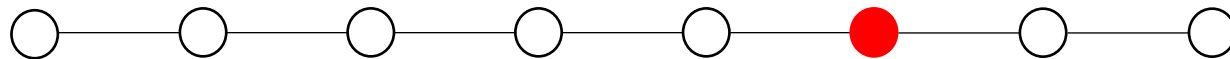
BFSまとめ

- 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！



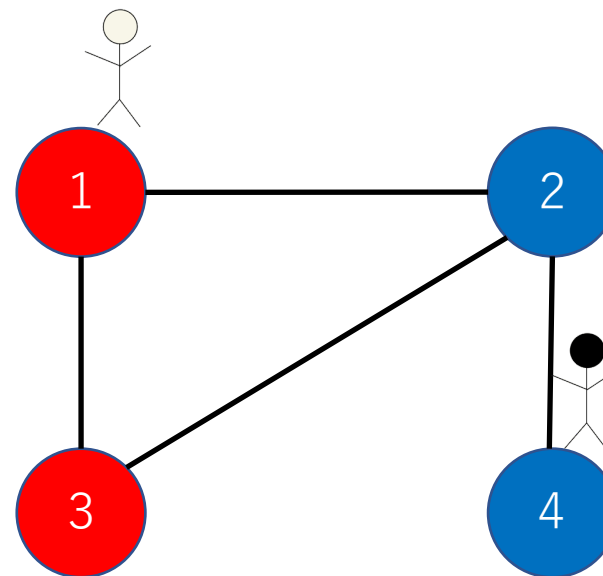
BFSの応用(頂点の持ち方を工夫する)

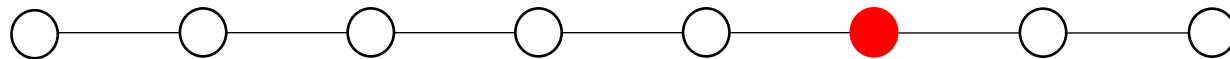
- E - Swap Places https://atcoder.jp/contests/abc289/tasks/abc289_e
- N頂点M辺の単純無向グラフが与えられる。
- 各頂点は赤か青のいずれかで塗られている。
- 頂点1と頂点Nにいる2人が、隣接する頂点に移動を繰り返すことでそれぞれ頂点N,頂点1にいる状態にできるか。ただし2人の移動先の頂点の色は異なる必要がある。
- Nの総和,Mの総和はともに2000を超えない。



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





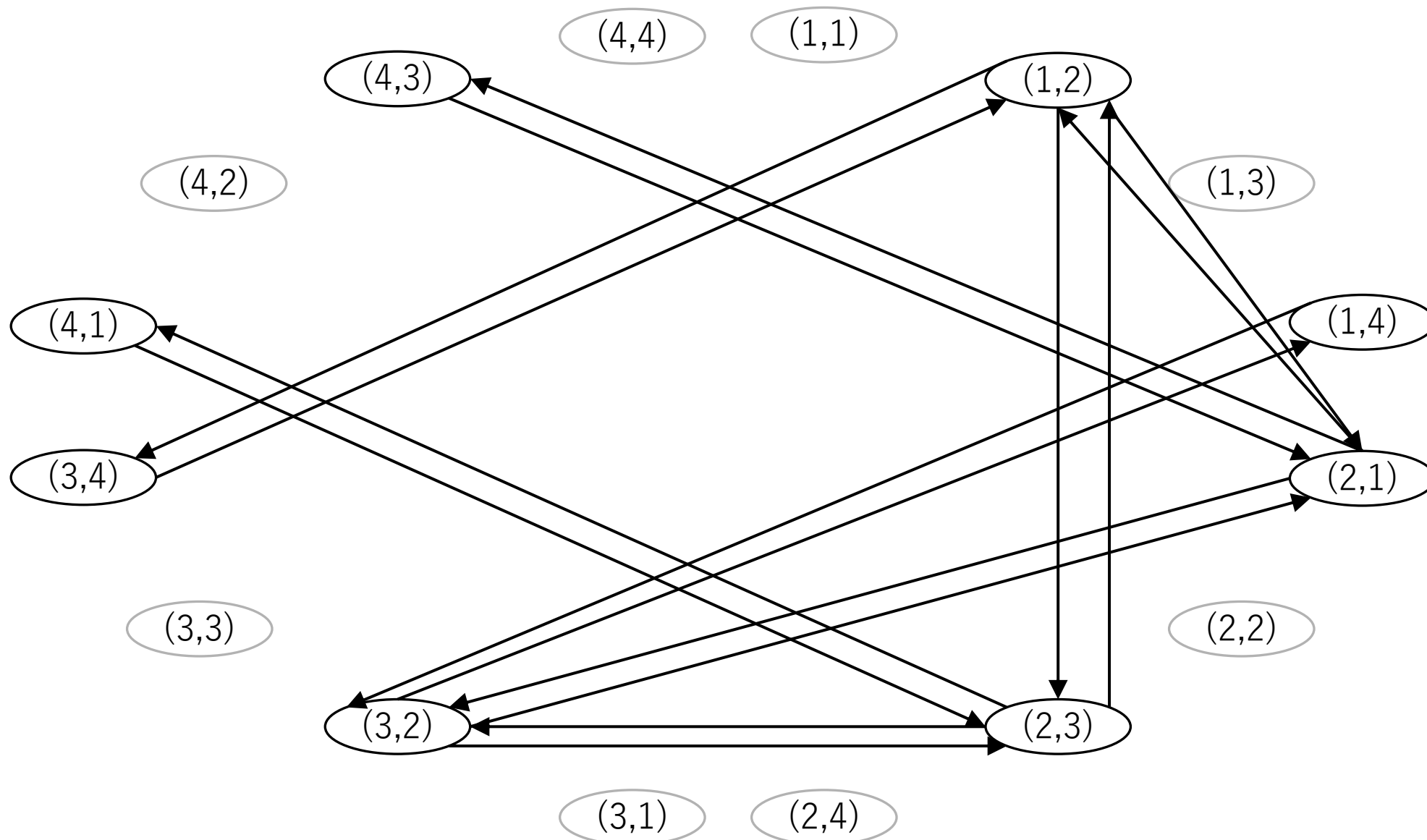
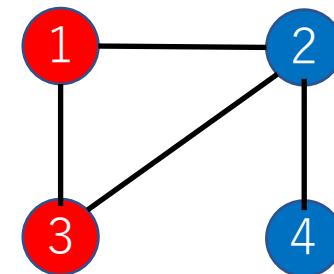
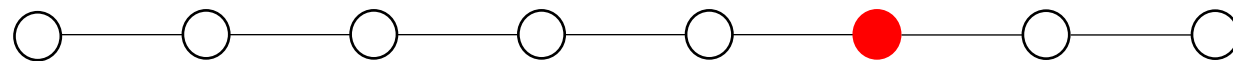
解き方

- 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

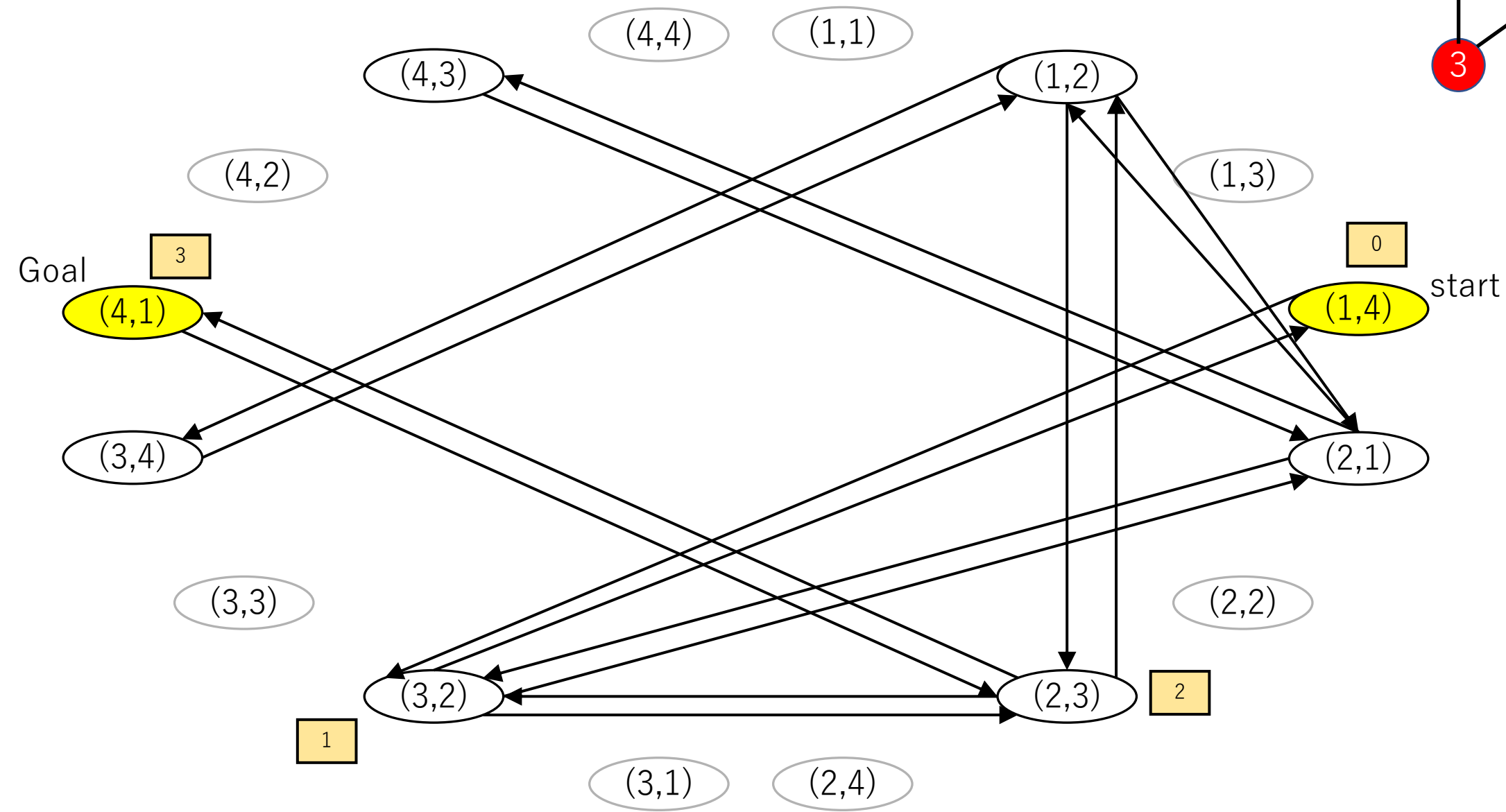
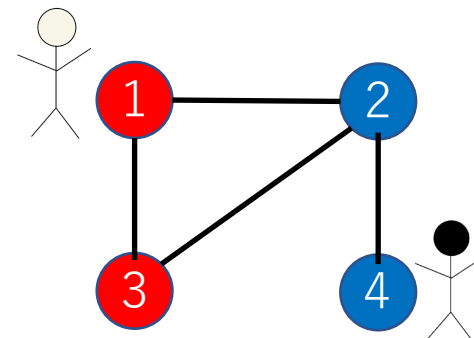
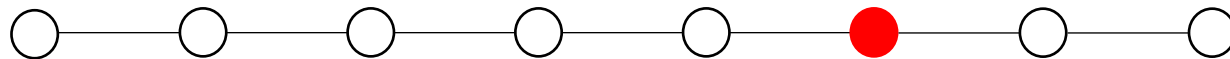
Aさんがいる頂点と**Bさんがいる頂点**の組を頂点とするグラフを考える！

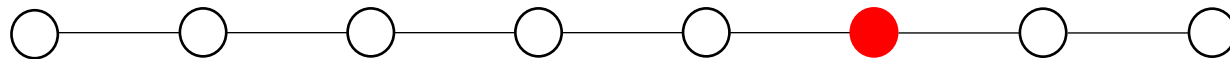
頂点 (i,j) を頂点 i にAさんがいて,頂点 j にBさんがいる状態に対応する点とすると,求める答えは頂点 $(1,N)$ から頂点 $(N,1)$ の最短経路長。

解き方のイメージ



解き方のイメージ

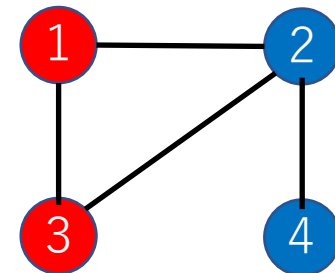




計算量の見積もり

- 計算量は $O(N^2 + M^2)$

∵ 頂点数は N^2 , 辺数は ${}_M C_2 \div 2 = M^2$



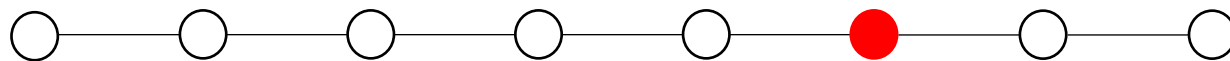
2つの頂点の組を結ぶ辺は元の辺を2本使う。

例: $(1,2) \rightarrow (3,4)$ 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。

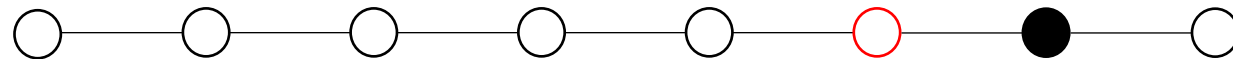
辺13は(1を含む頂点組) \rightarrow (3を含む頂点組) と (3を含む頂点組) \rightarrow (1を含む頂点組) で使われる。

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



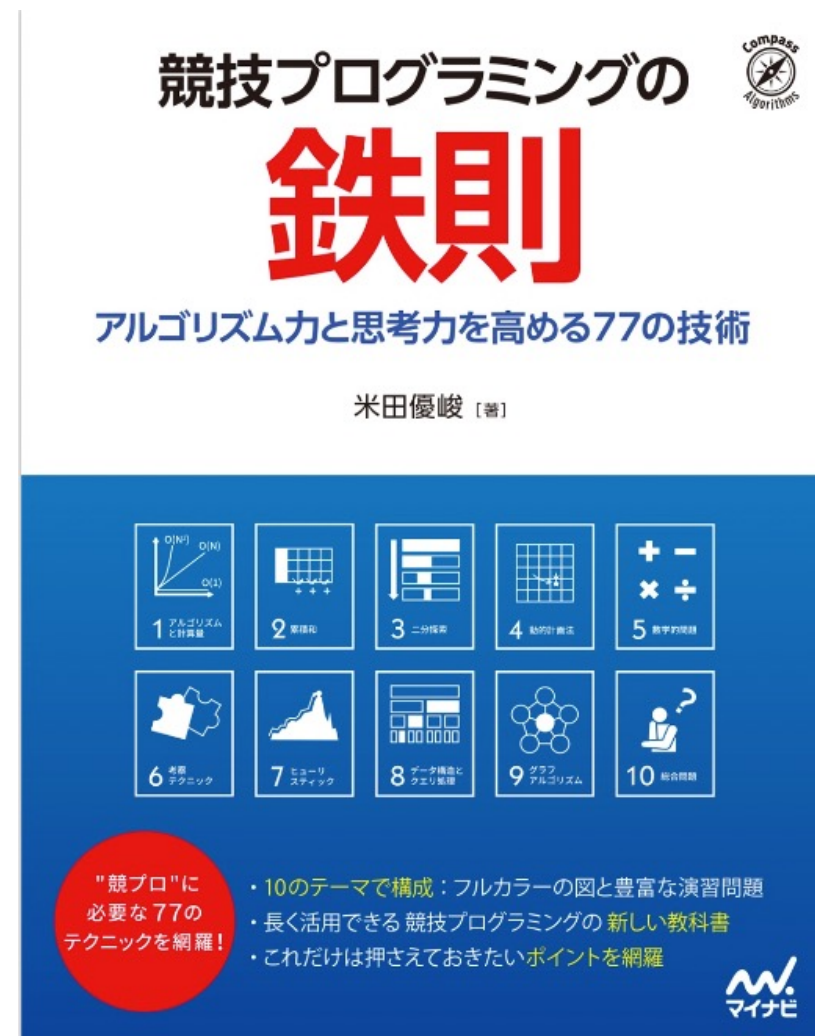
この問題のポイント(まとめ)

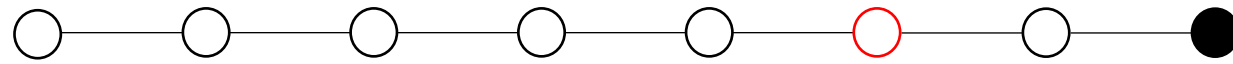
- ある状態を点にして考える。
(Bit使うものが多い)
- 制約が小さいのがポイントだった。
($N^2 \leq 10^6$)



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



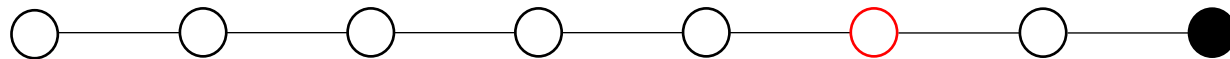


BFSの問題1

- 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

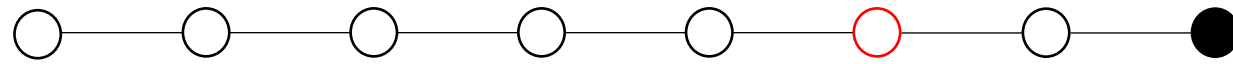
- A - 幅優先探索 https://atcoder.jp/contests/atc002/tasks/abc007_3

グリッド上のBFS。

この形式も頻出なので慣れておきたい。

4方向の探索は

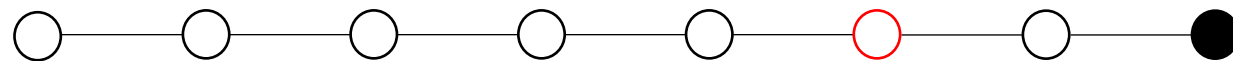
$vx = \{0, 1, 0, -1\}, vy = \{1, 0, -1, 0\}$ と2つの配列を用意しておくと便利



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！