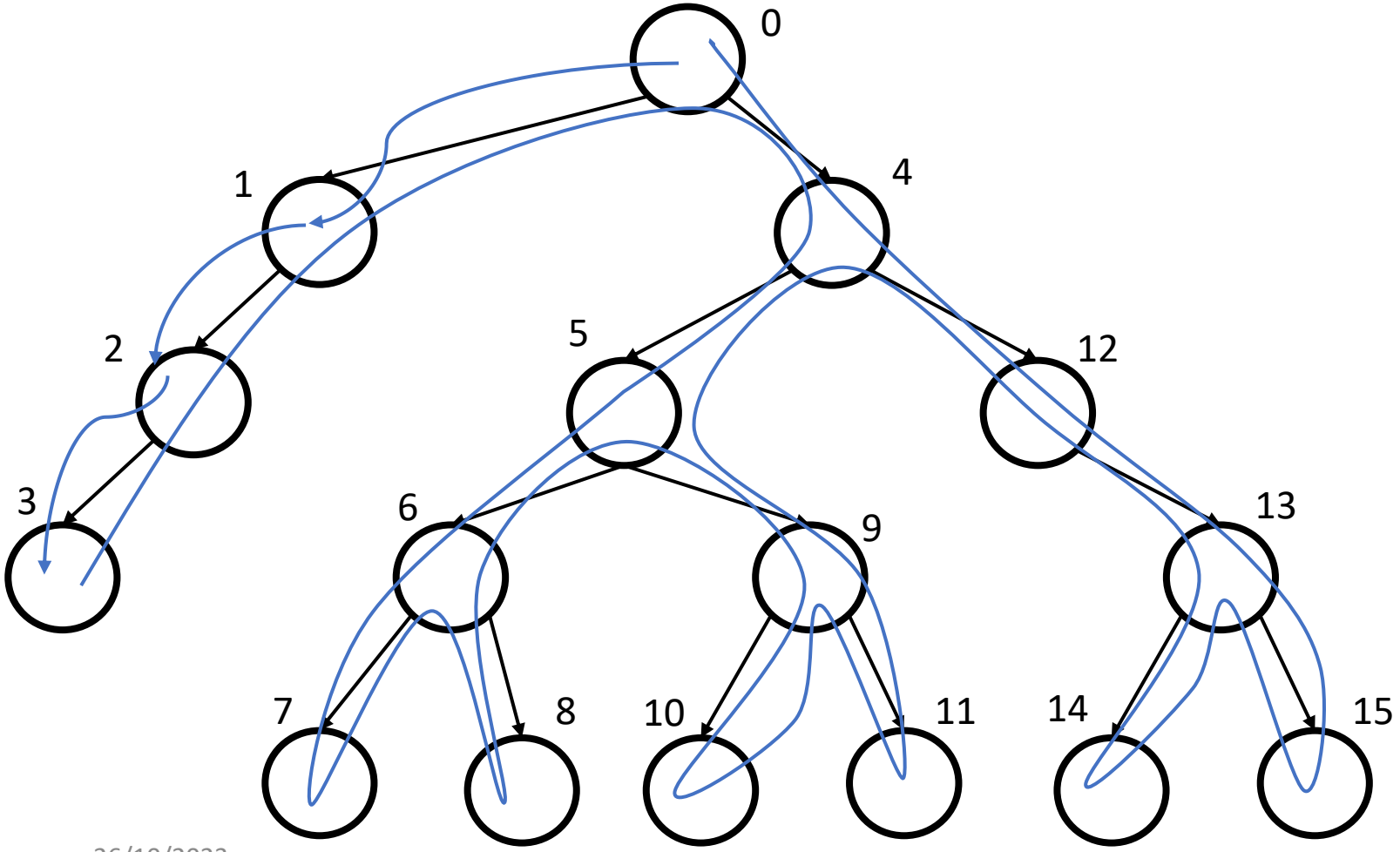
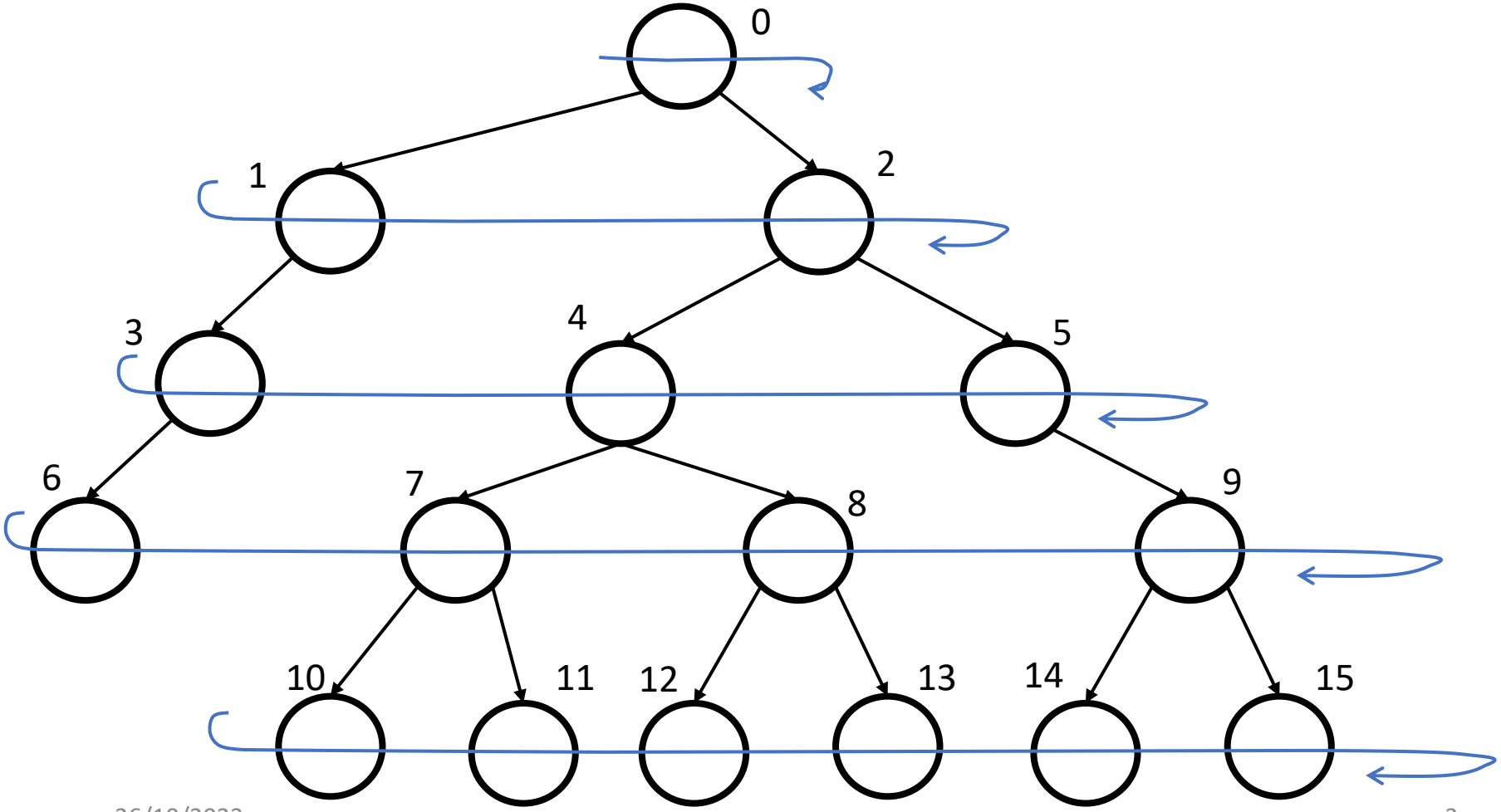
応用アルゴリズム演習

Breadth First Search　/　幅優先探索

深さ優先探索（リマインド）



幅優先探索

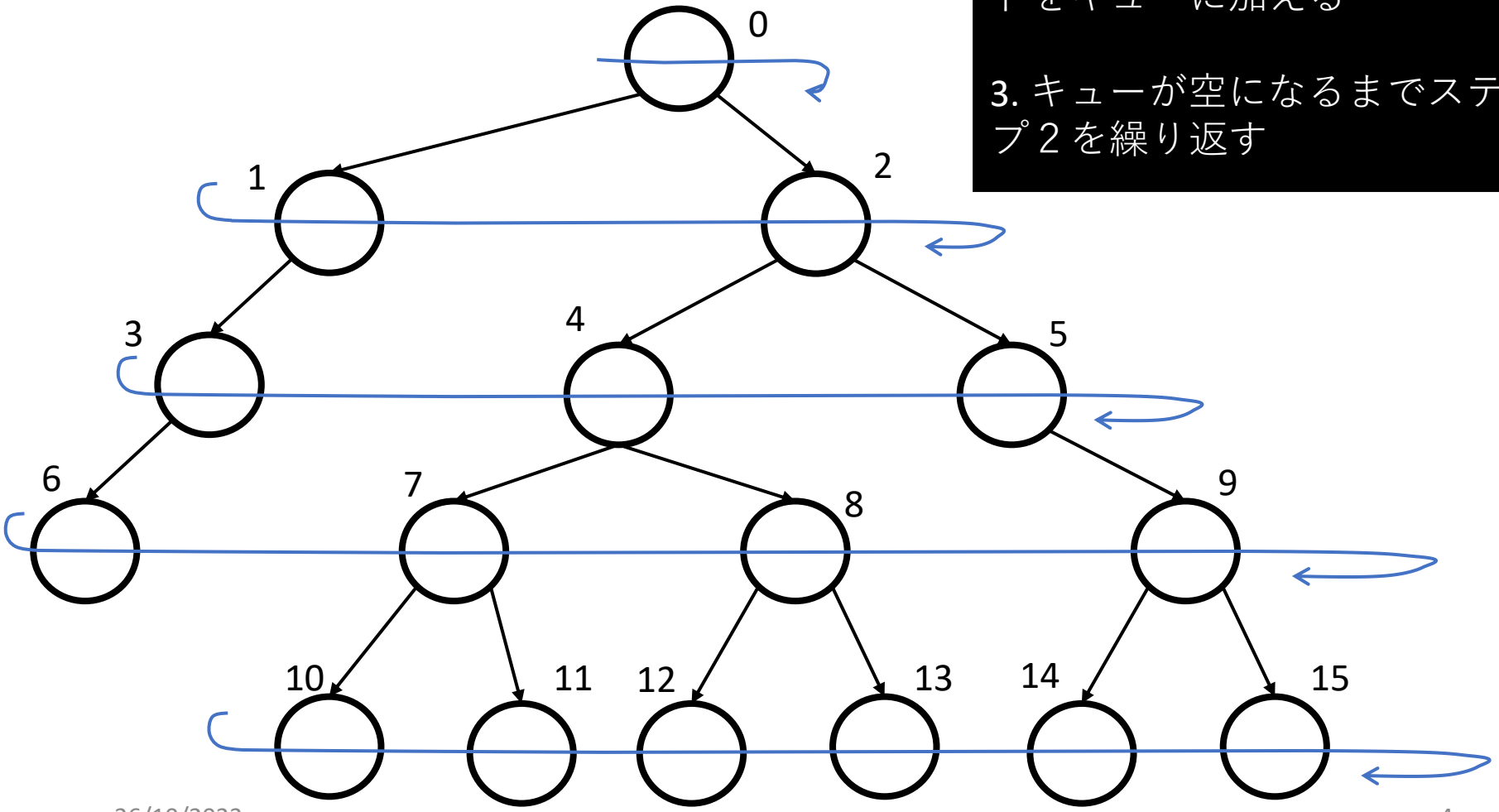


幅優先探索

アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す



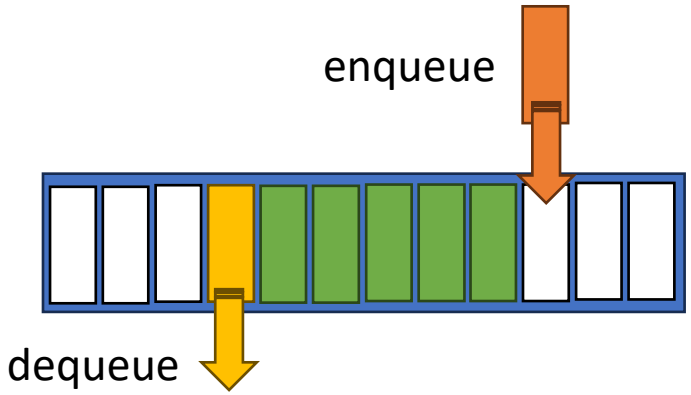
待ち行列・キュー (Queue)

- FIFO (First-In First-Out)
  先に入れた要素が先に出る

- リングバッファで実装：

```
typedef struct queue {
    int head, tail, size;
    ELEM buf[BUFSIZE];
} queue_t;
```

```
void enqueue(queue_t * q, ELEM data) {
    q->buf[q->tail++]=data;
    q->size++;
    if(q->tail==BUFSIZE) {q->tail = 0;}
}
```



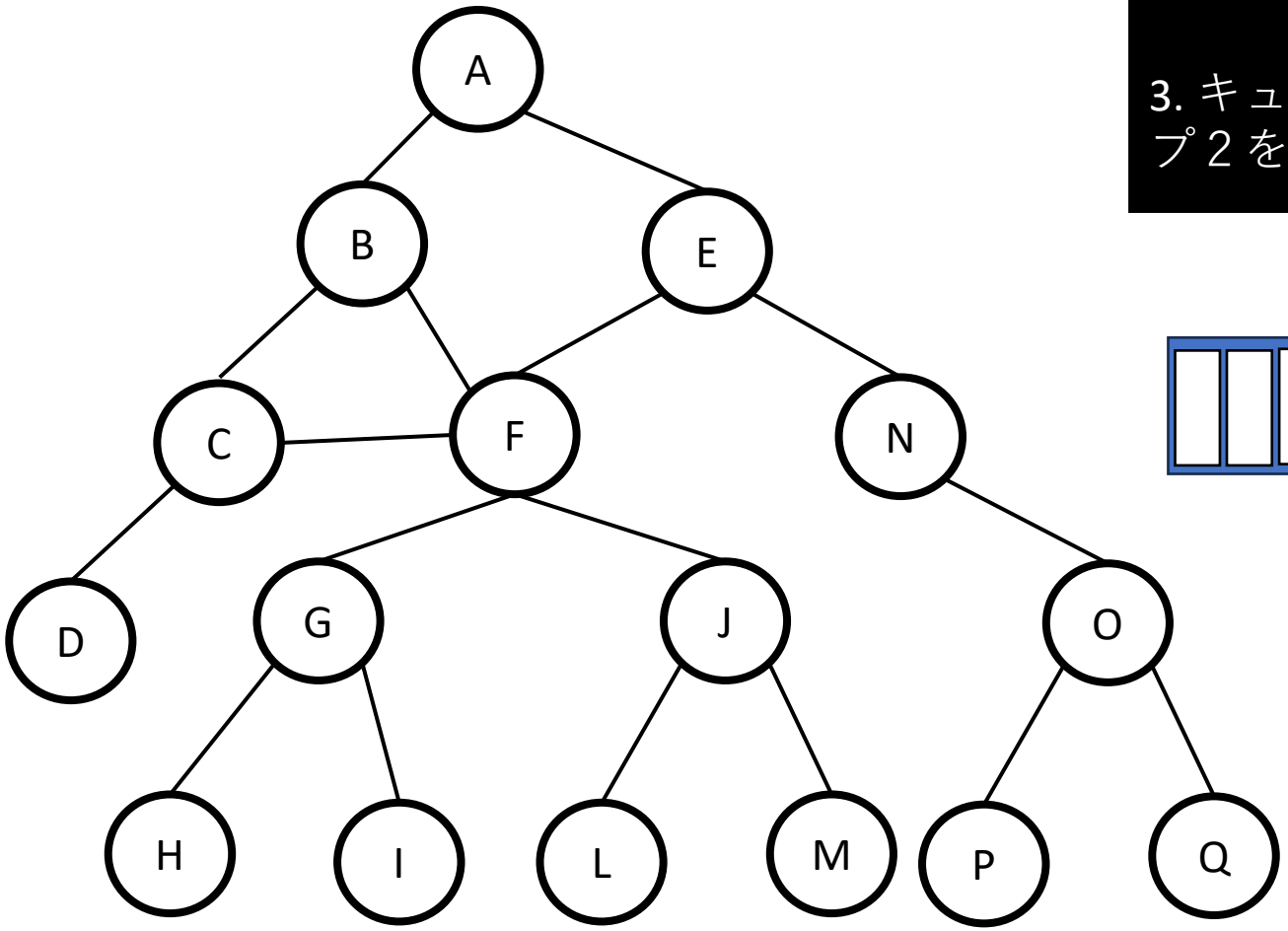

```
ELEM dequeue(queue_t * q) {
    ELEM result = q->buf[q->head++];
    q->size--;
    if(q->head==BUFSIZE) q->head = 0;
    return result;
}
```

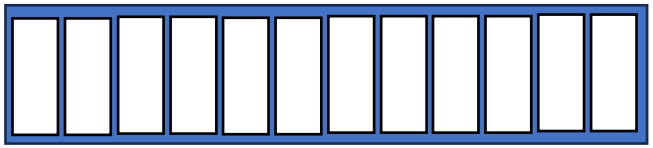
幅優先探索

アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す





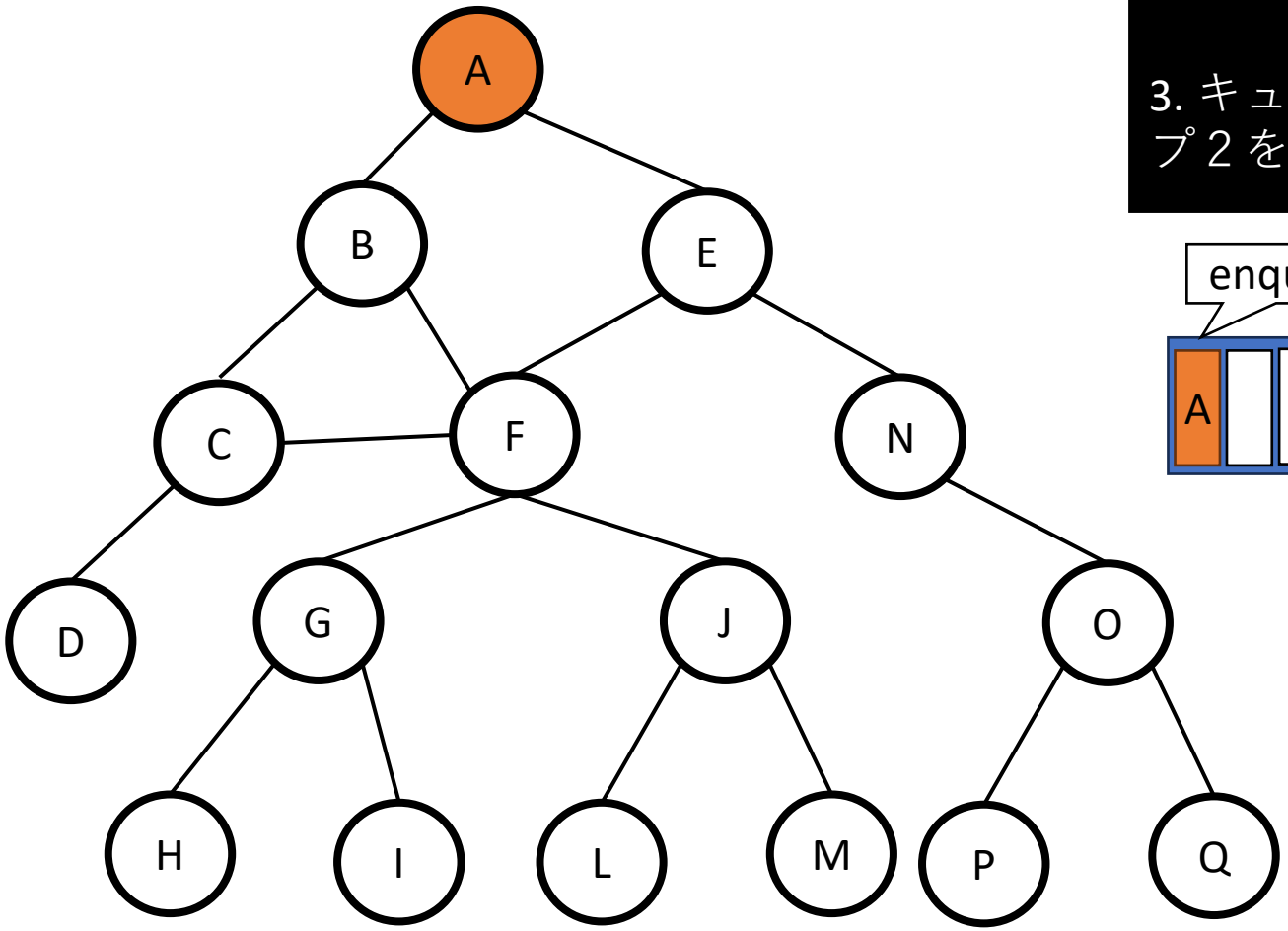
幅優先探索

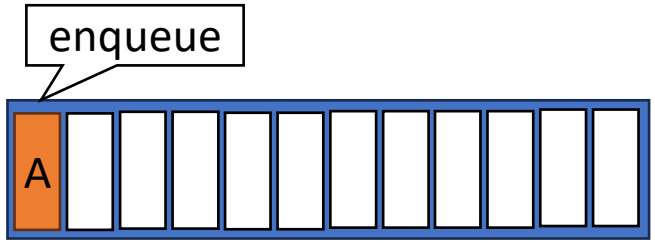
初期設定

アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す





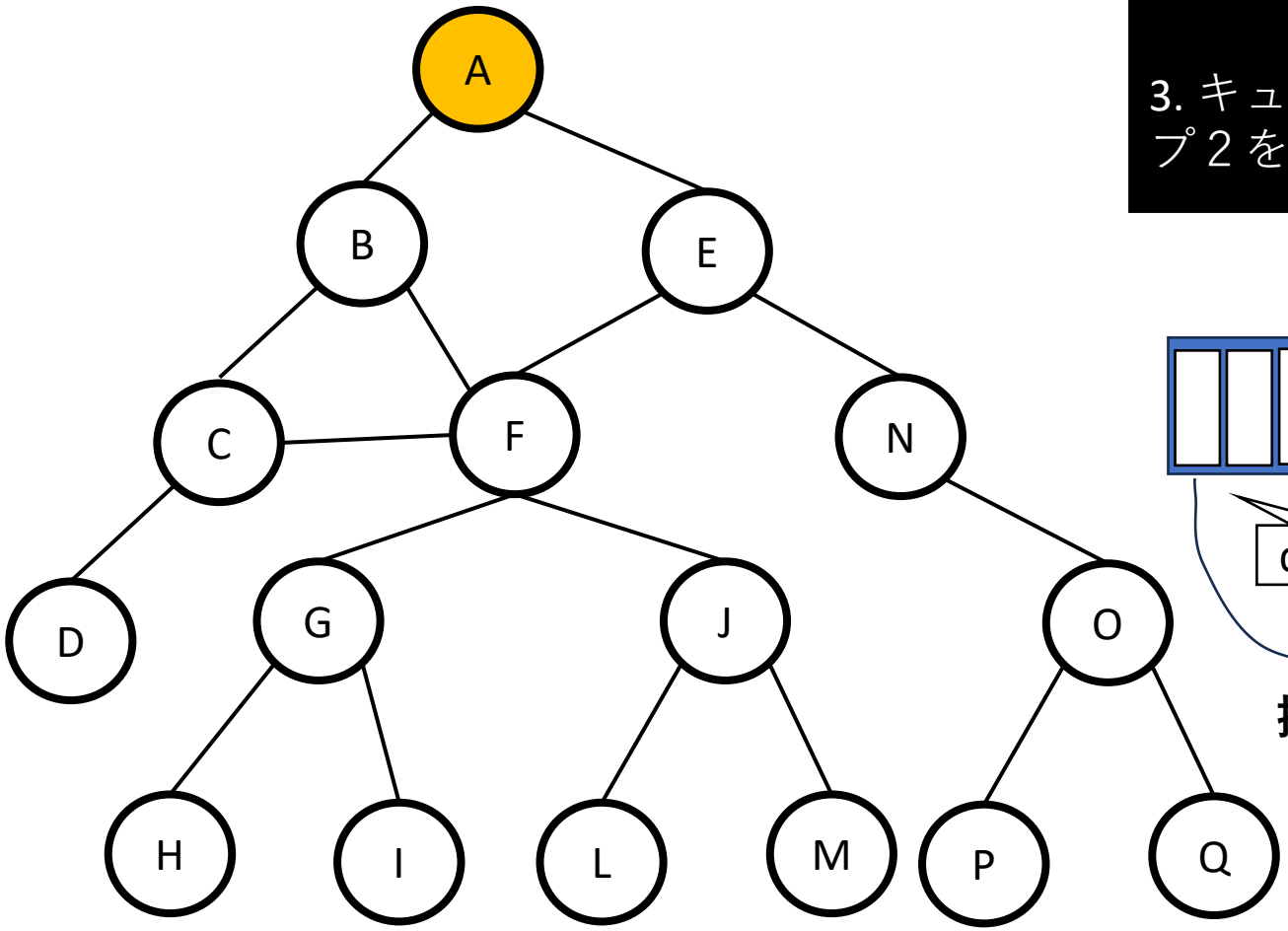
幅優先探索

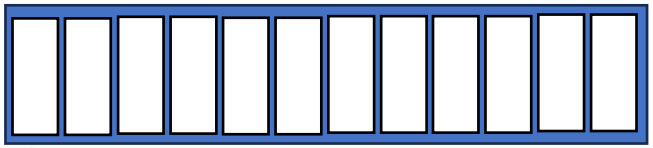
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





dequeue

探索中：

A

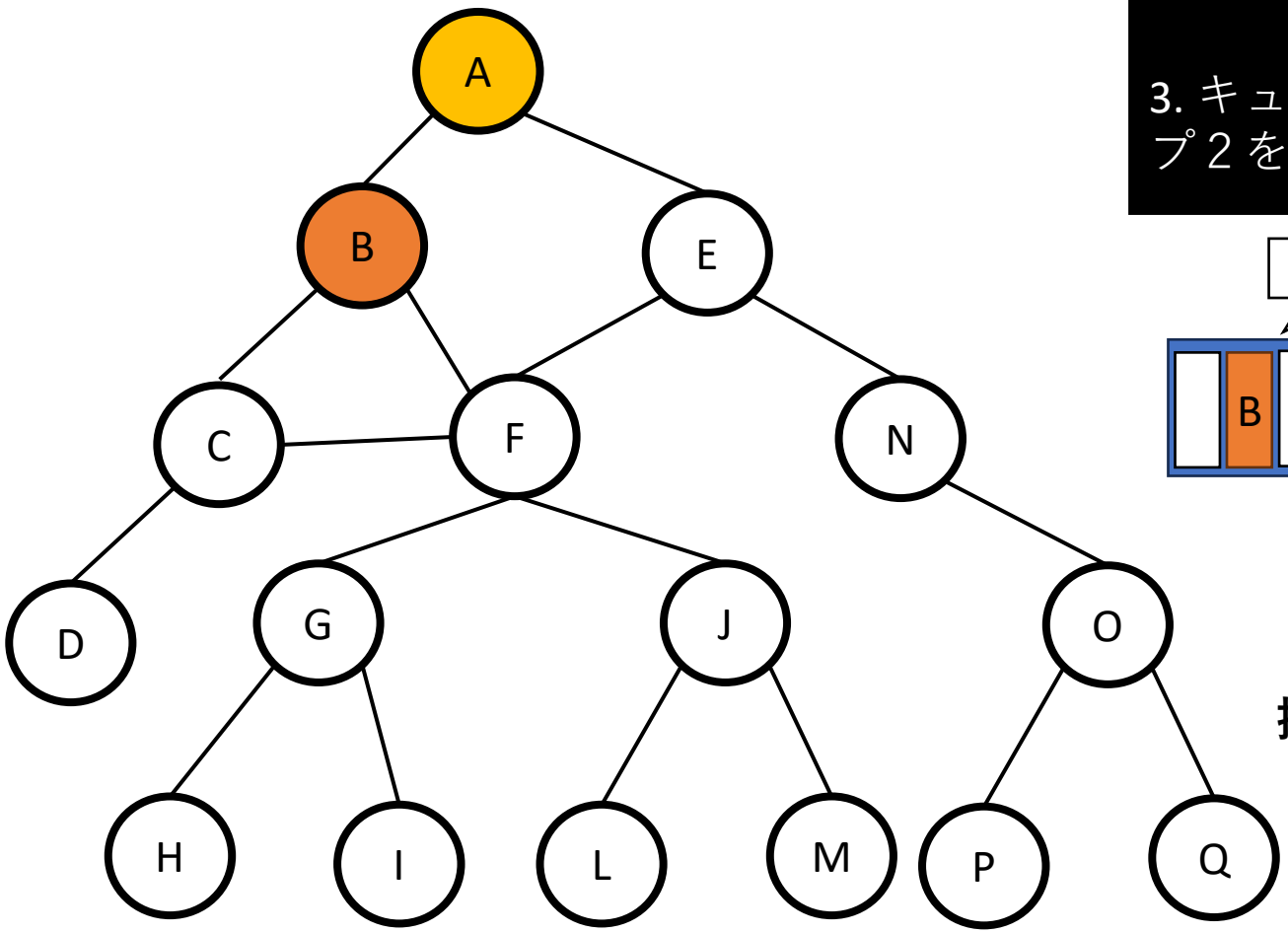
幅優先探索

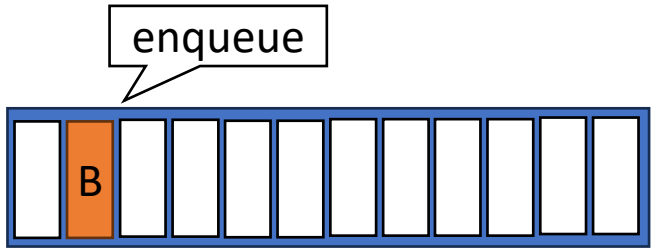
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





探索中： A

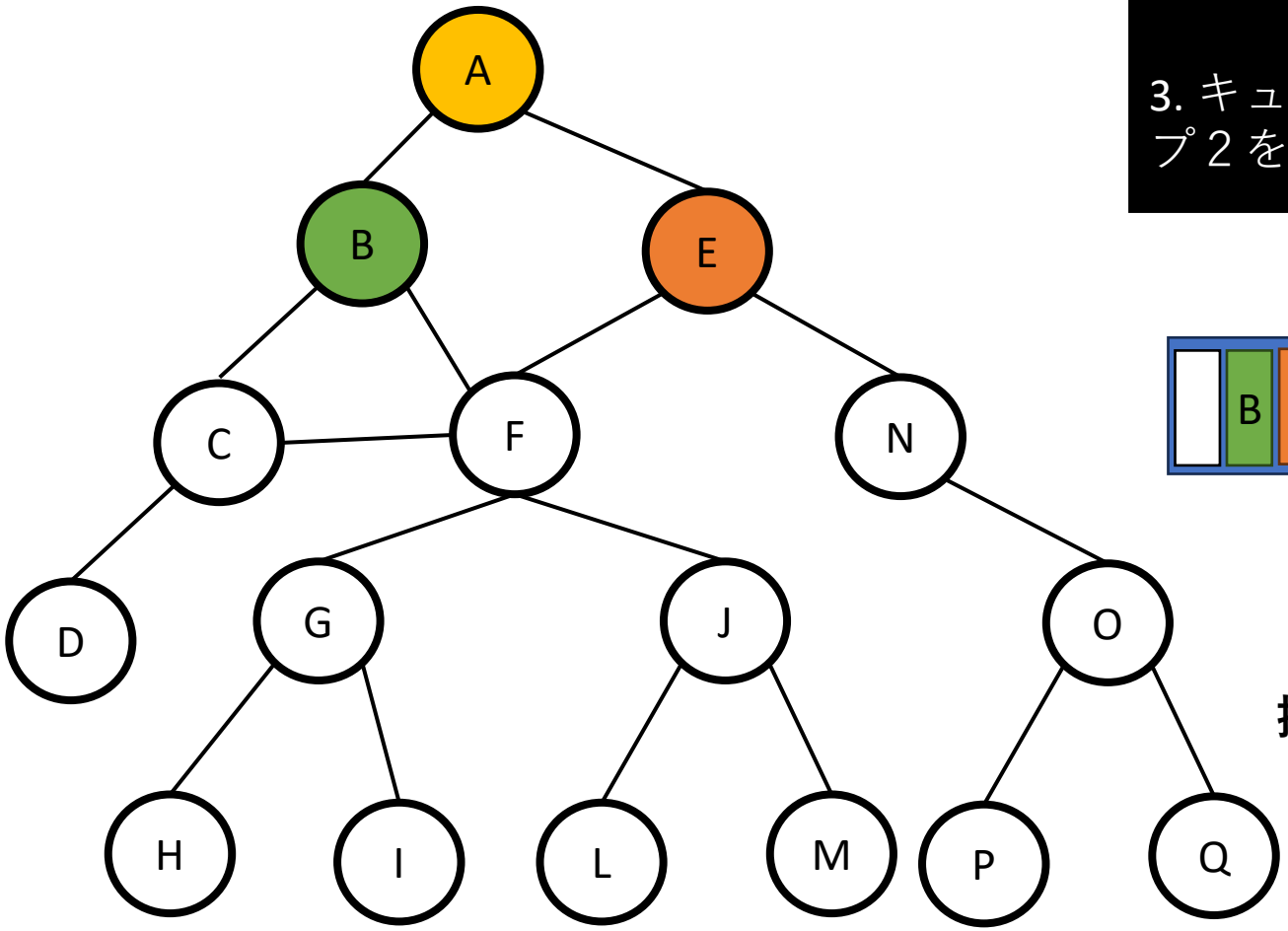
幅優先探索

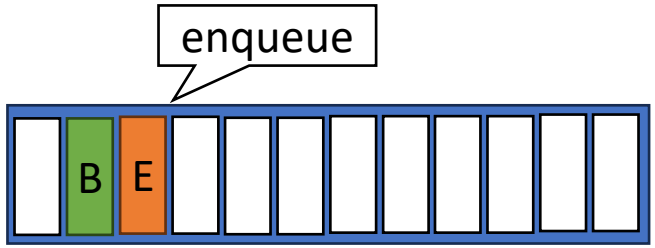
ループ

アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す





探索中： A

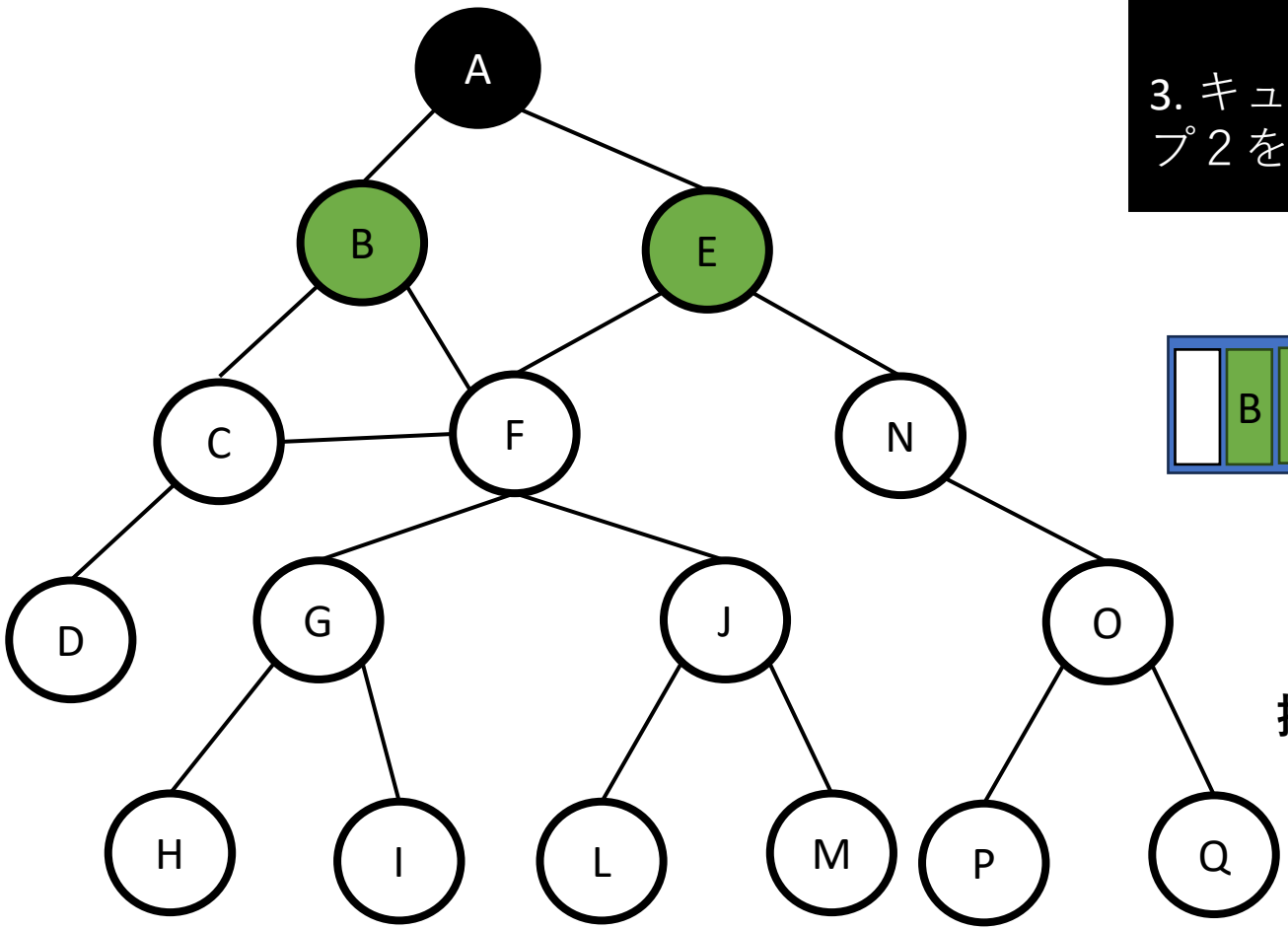
幅優先探索

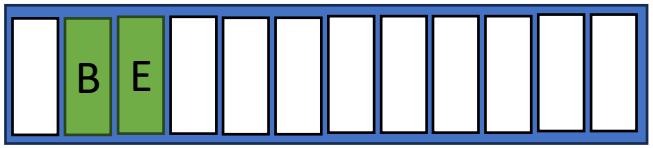
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





探索中： A

ノードAからの探索が終わりました

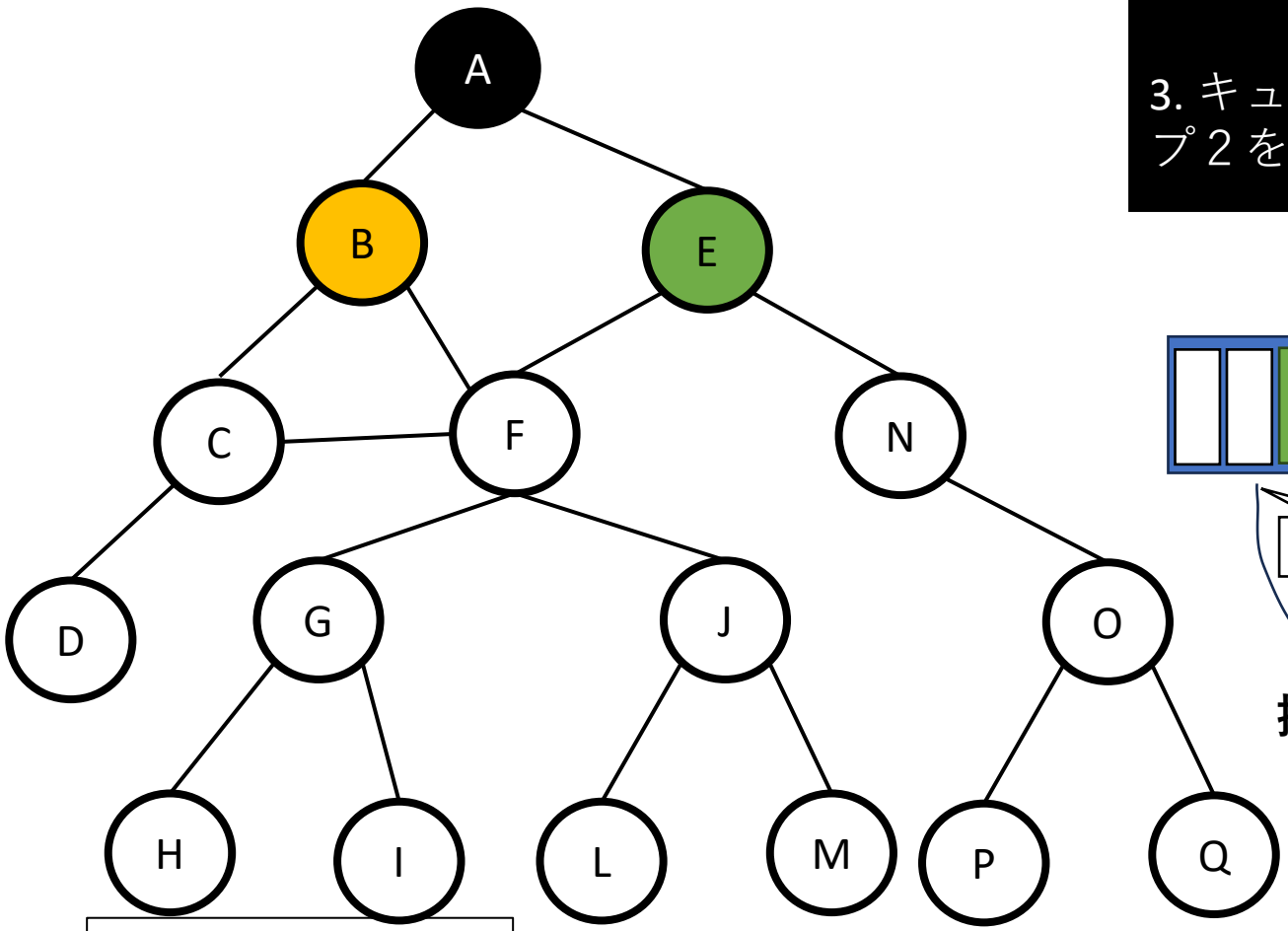
幅優先探索

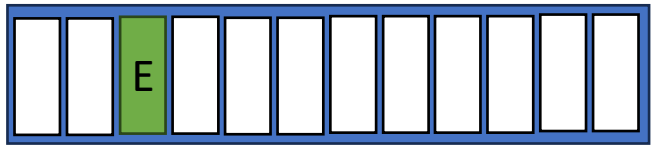
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





dequeue

探索中：

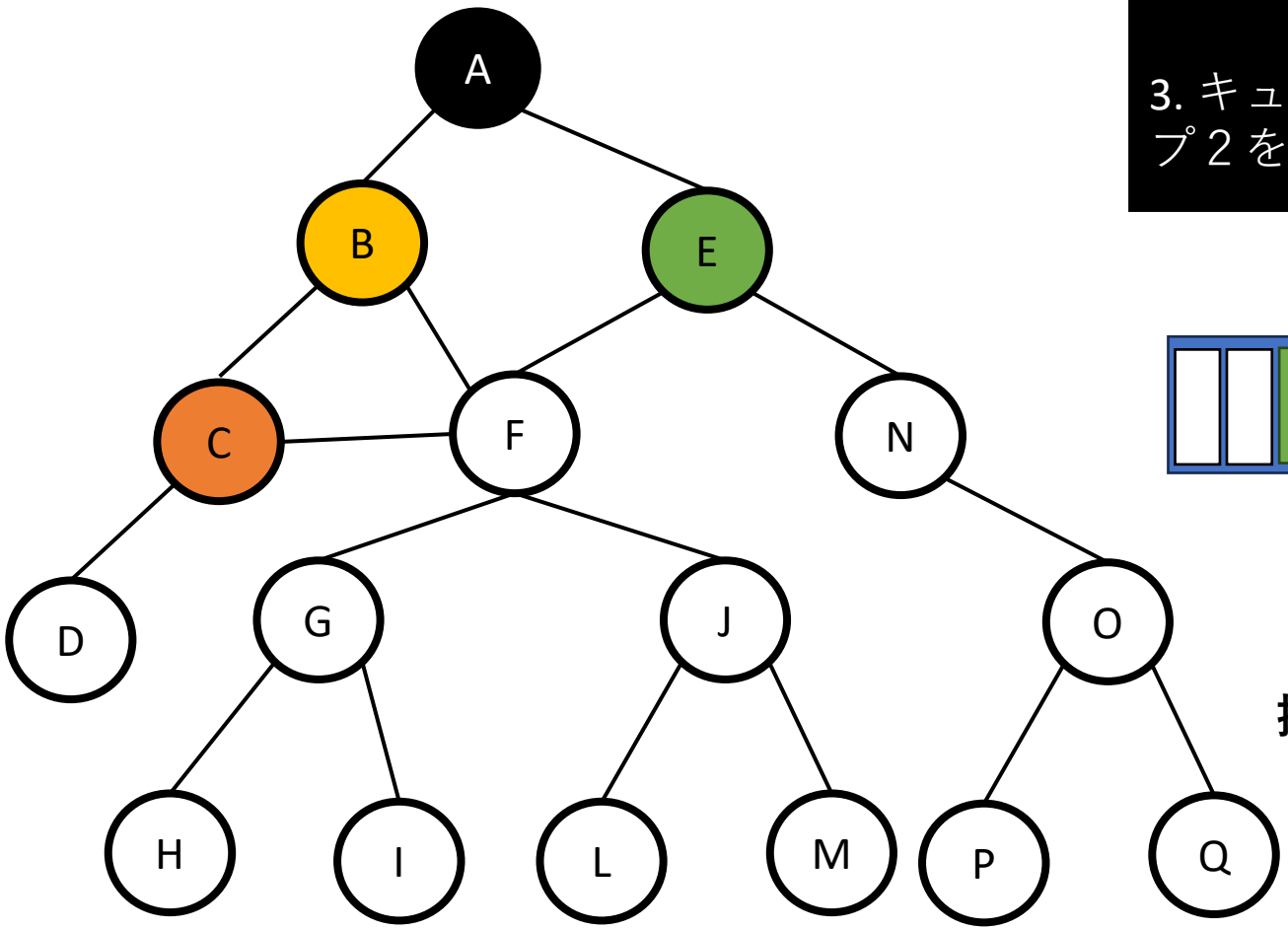
幅優先探索

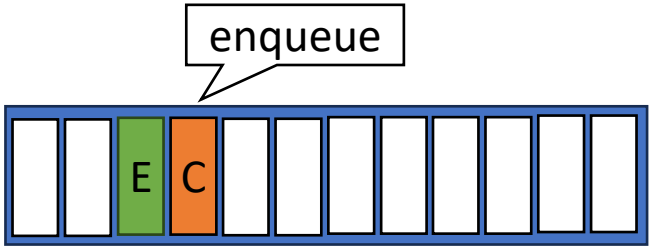
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





探索中： B

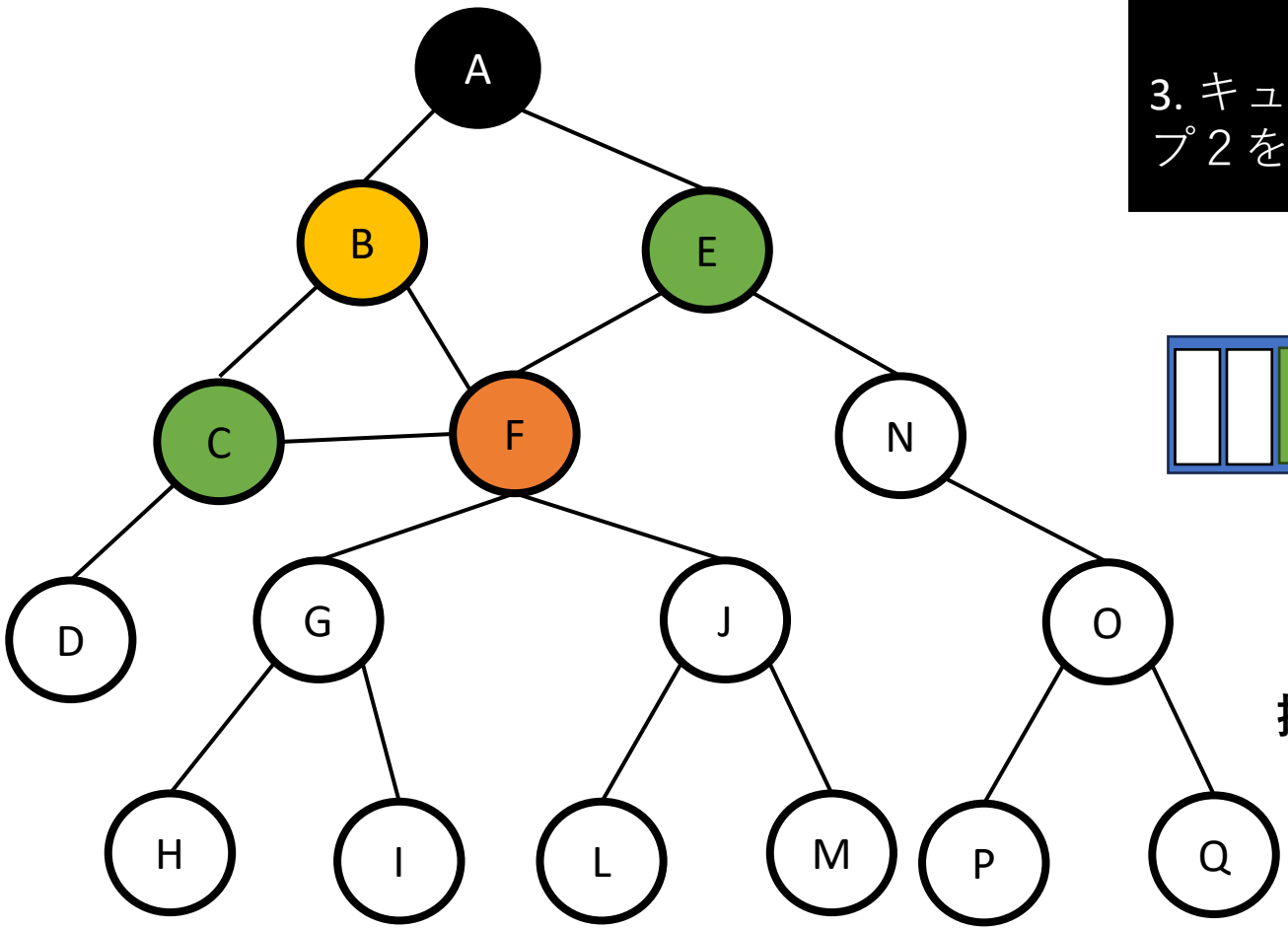
幅優先探索

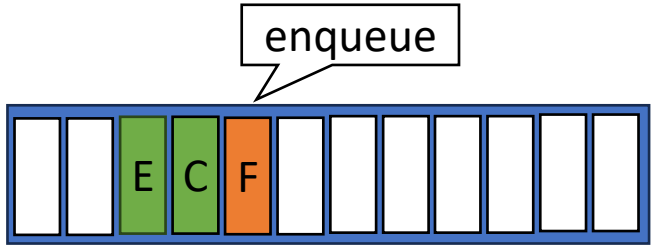
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





探索中： B

幅優先探索

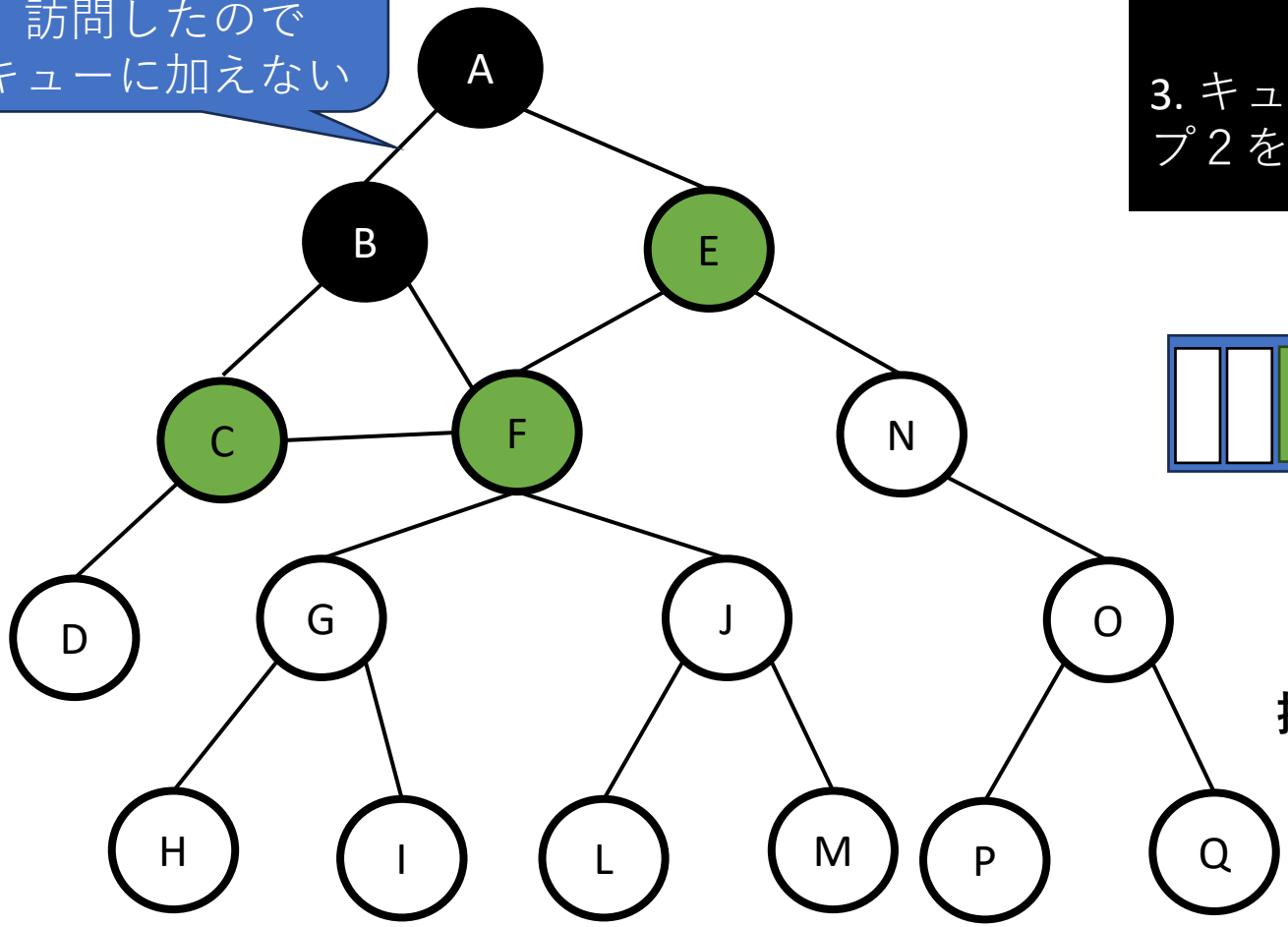
アルゴリズムの流れ
1. 最初のノードをキューに入れる

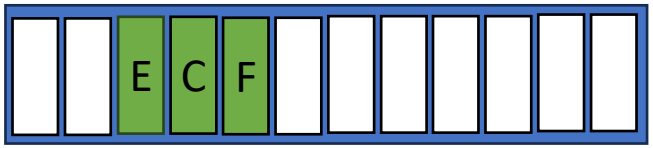
2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ

ノードAもBに隣接するけど、すでに訪問したのでキューに加えない





探索中：

ノードBからの探索が終わりました

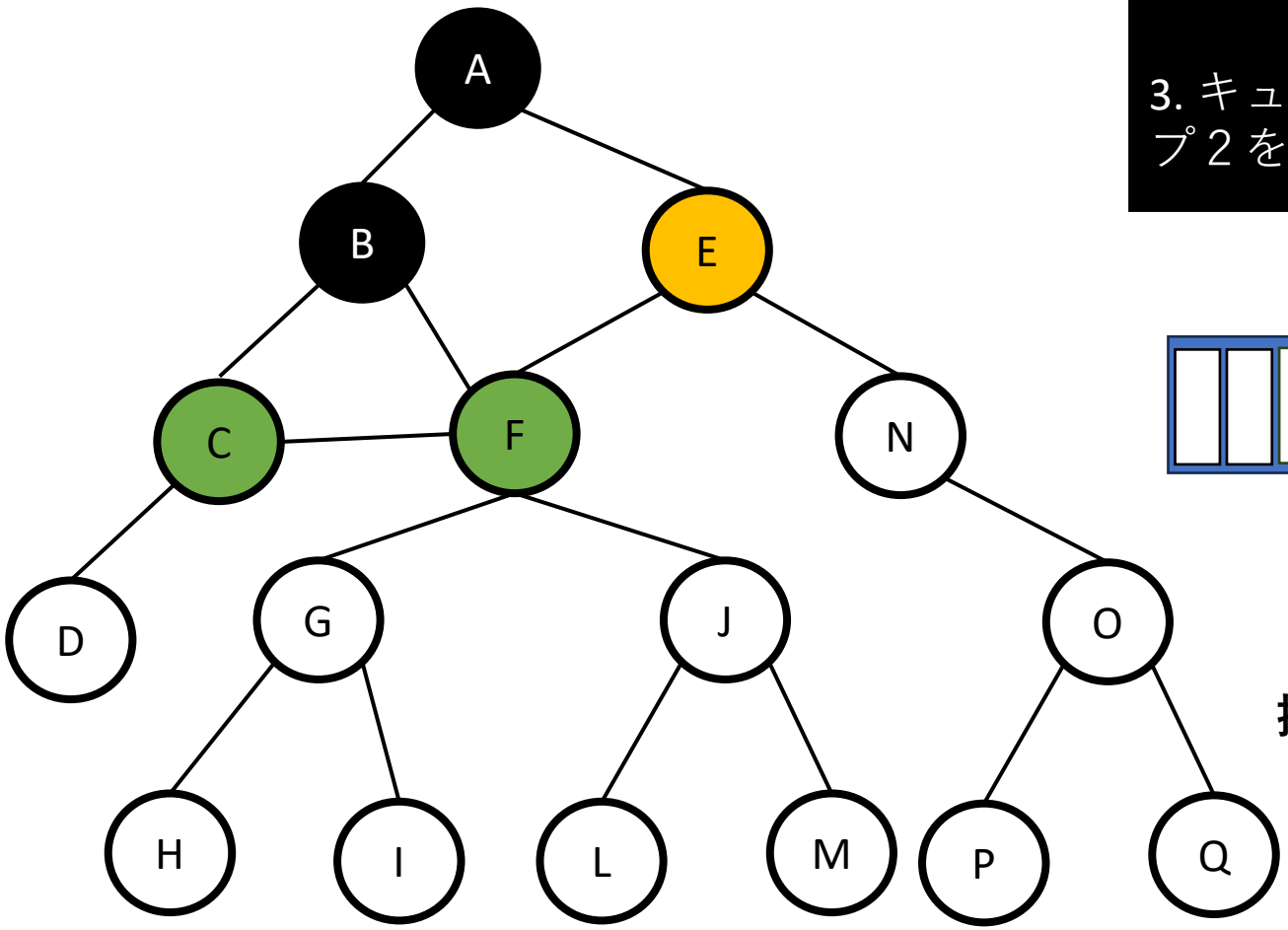
幅優先探索

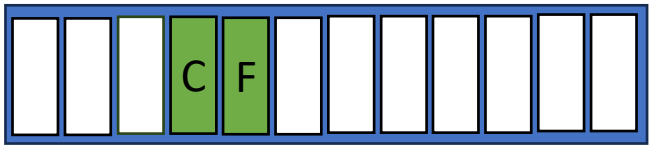
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





dequeue

探索中：

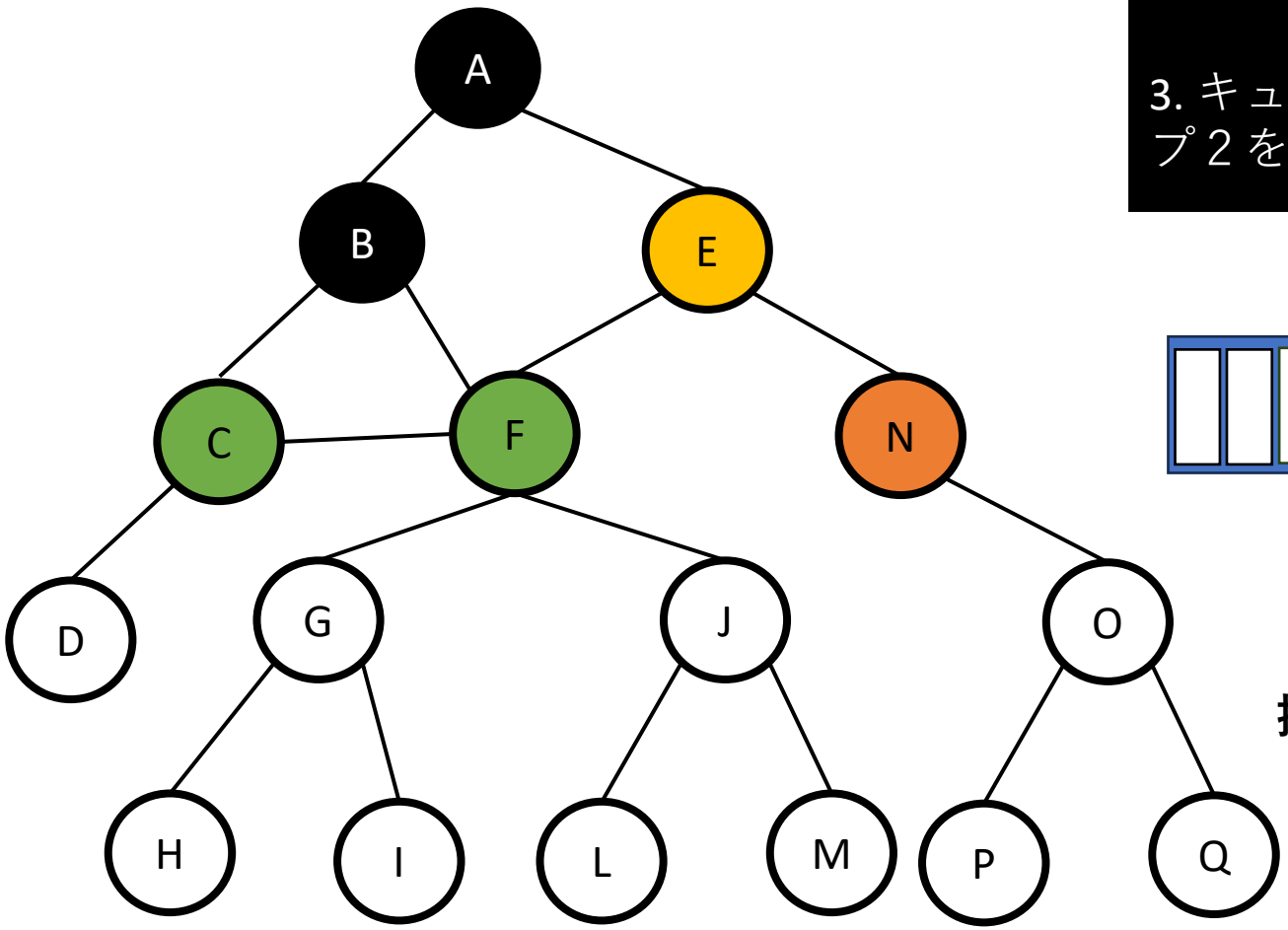
幅優先探索

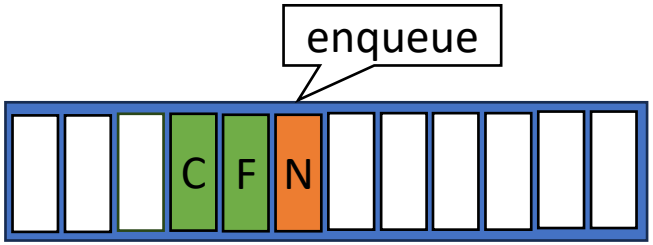
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





探索中： E

幅優先探索

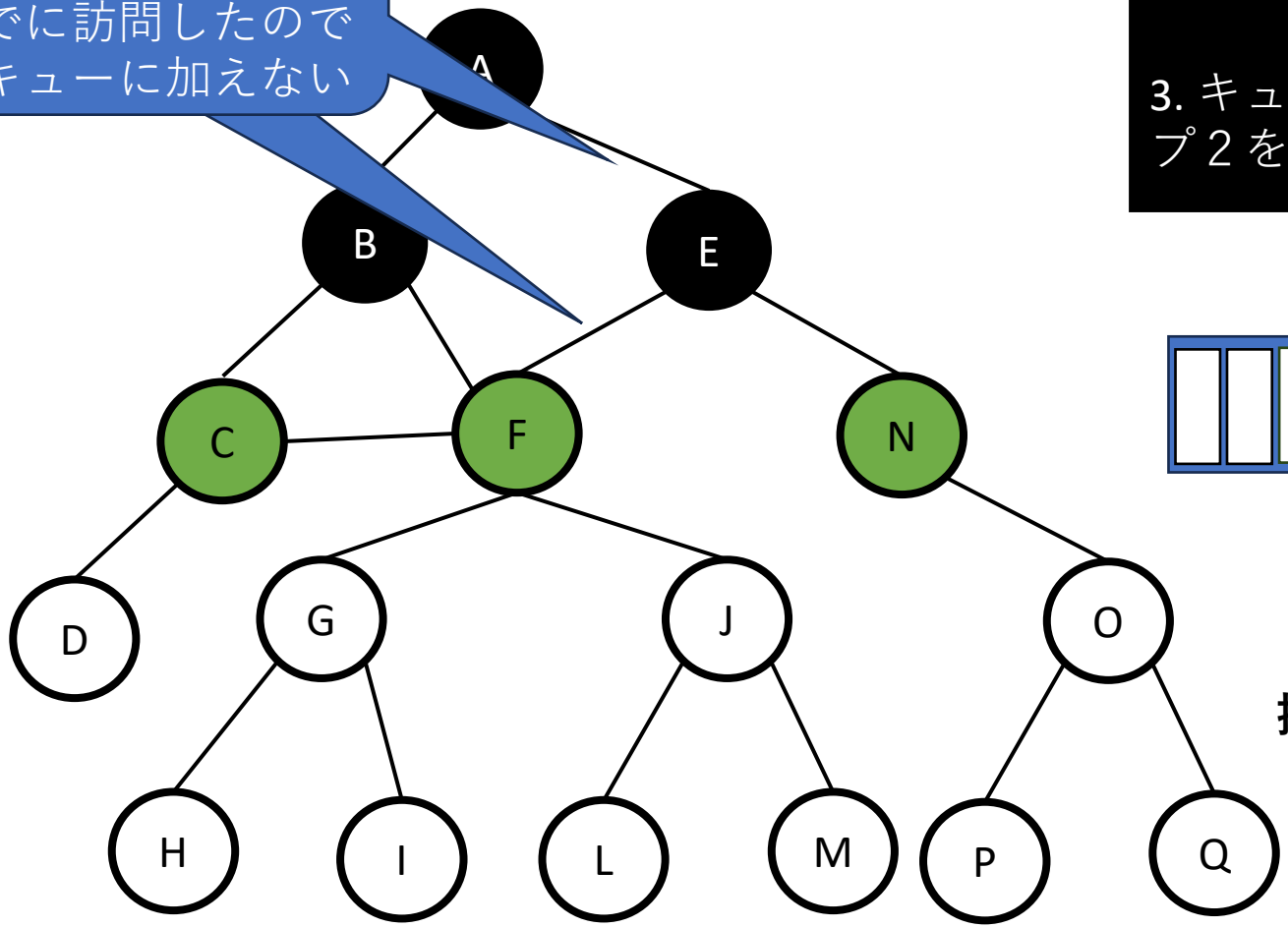
アルゴリズムの流れ
1. 最初のノードをキューに入れる

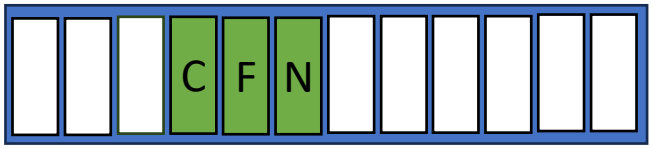
2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ

ノードAとFもEに隣接するけど、すでに訪問したのでキューに加えない





探索中： E

ノードEからの探索が終わりました

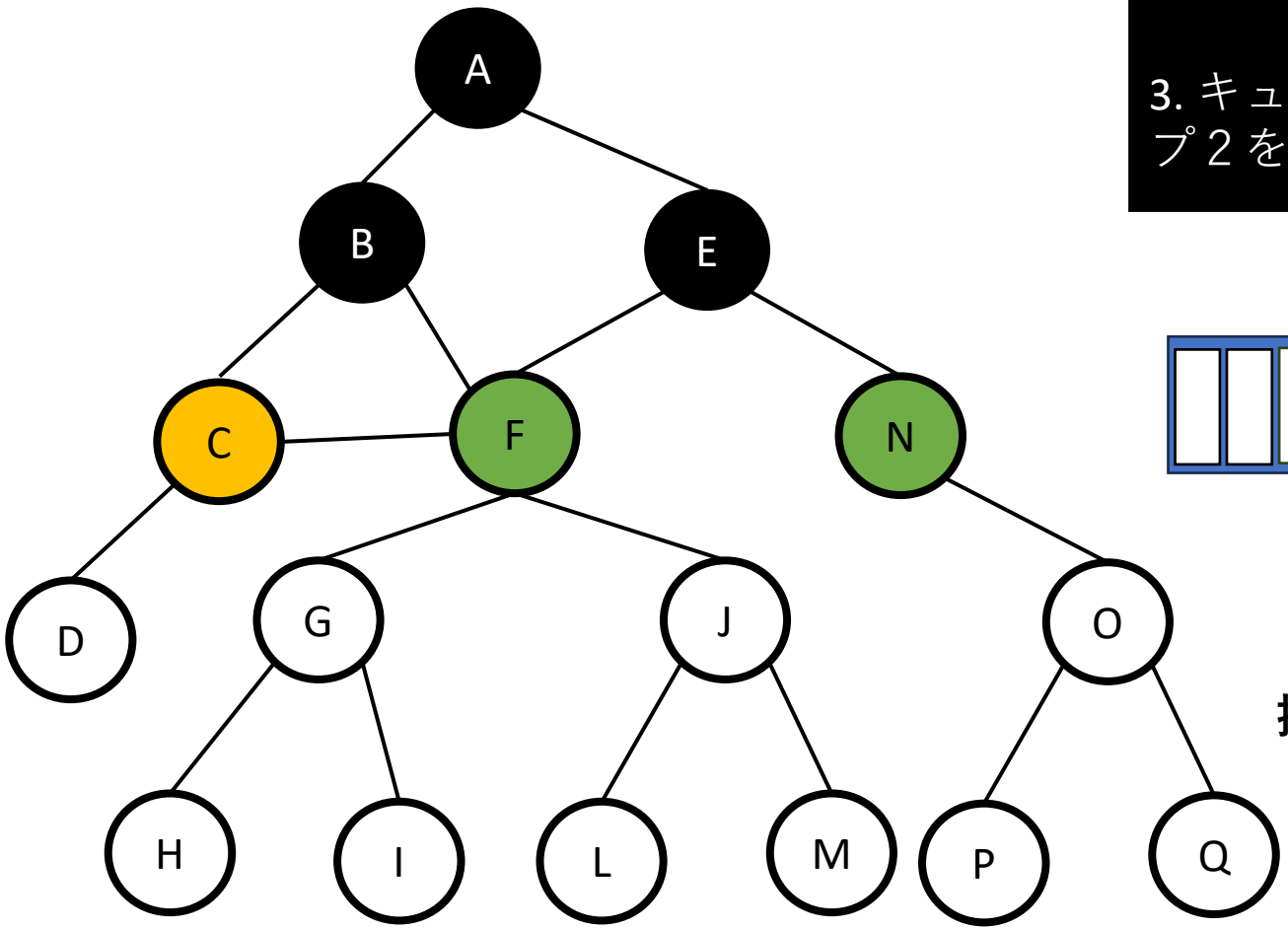
幅優先探索

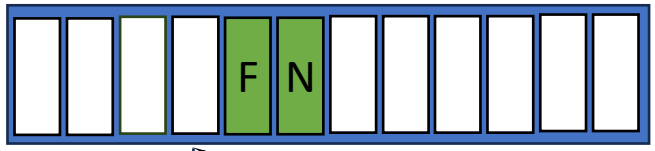
ループ

アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す





dequeue

探索中：

C

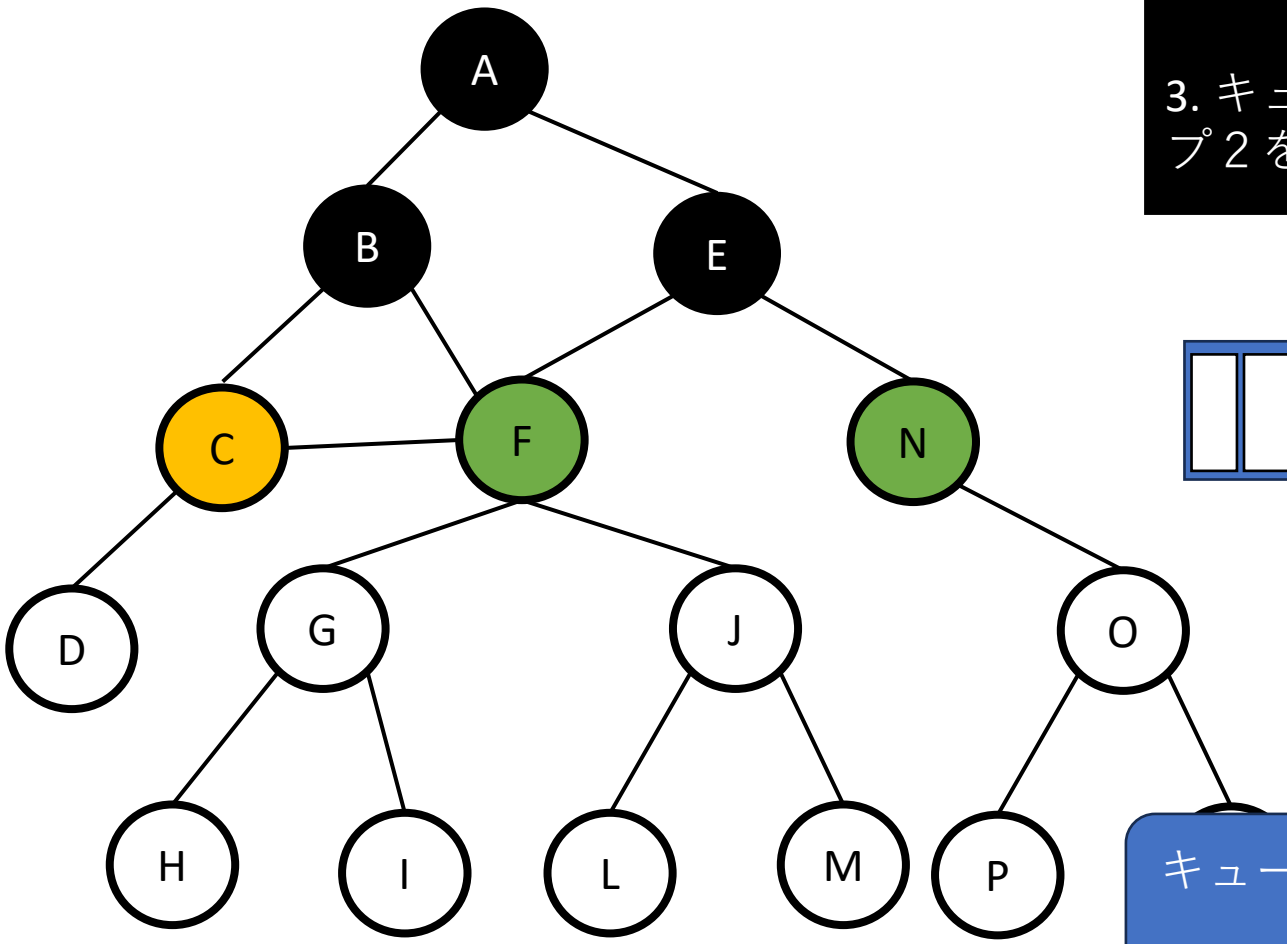
幅優先探索

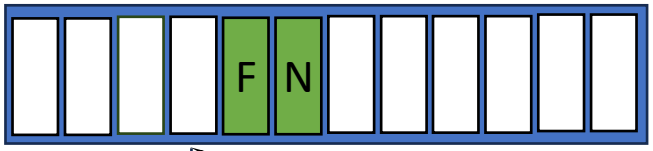
アルゴリズムの流れ
1. 最初のノードをキューに入れる

2. キューからノードを取り出して、隣接するかつ未探索のノードをキューに加える

3. キューが空になるまでステップ２を繰り返す

ループ





dequeue

探索中：

C

キューが空になるまで続く…

幅優先探索

このアルゴリズムを使って、何が出来る？

- **特定のノードからの距離　←（課題対象！）**

- **グラフの中にどのノードが連結しているか（つまり、どのノードからどこへ行けるか）（実は、深さ優先探索でも出来る）**

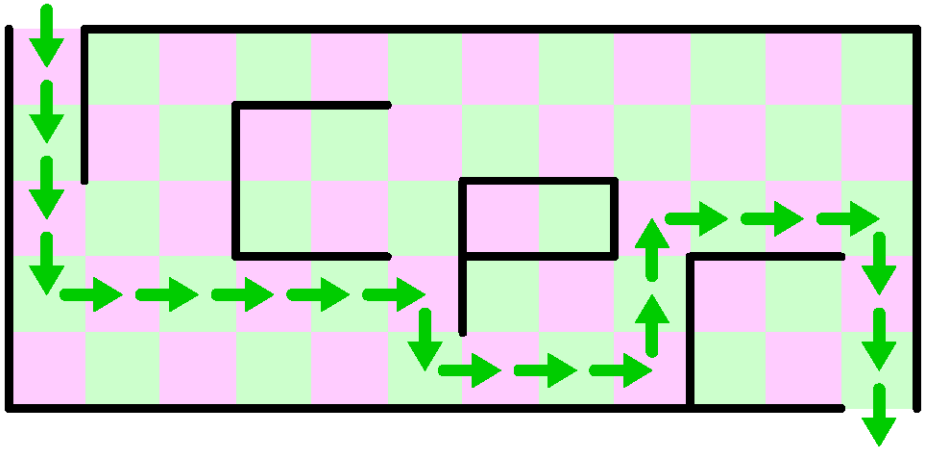
- **他に色々…**

課題２の説明

- **幅優先探索で問題を解けましょう**

問題：ACM ICPC のプログラミングコンテストの2010年国内予選問題

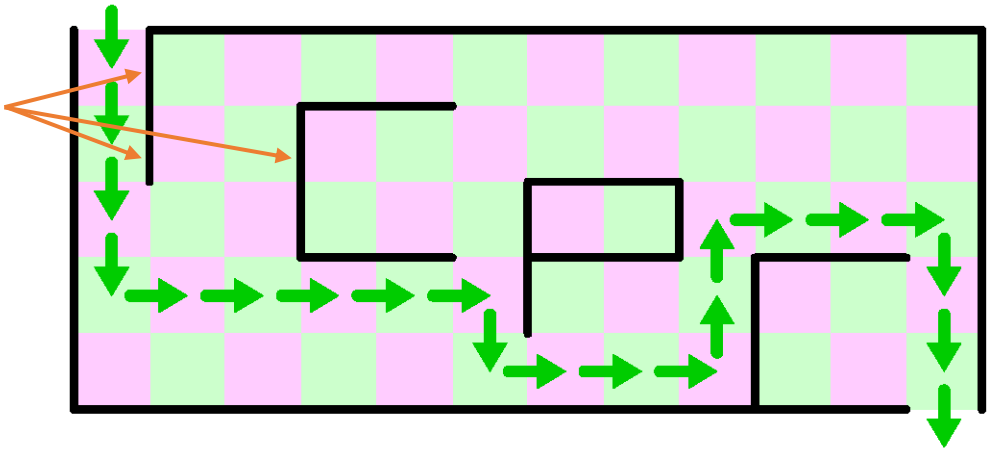
参考：https://icpc.iisf.or.jp/past-icpc/domestic2010/#section_B



課題２の説明

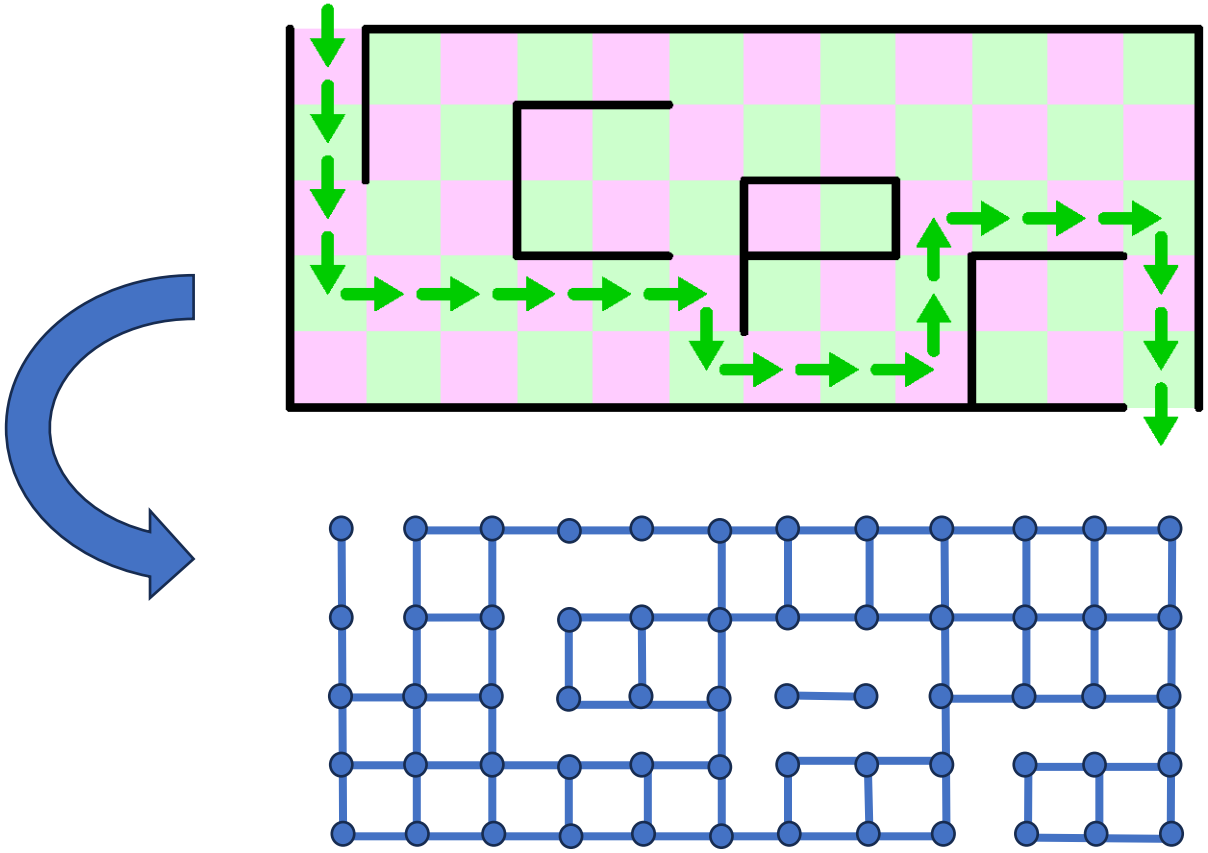
上左からスタートして、何ステップで下左に着けるか？

- **入力は：壁の場所**
- **迷路から出れない場合もあります**



課題２の説明

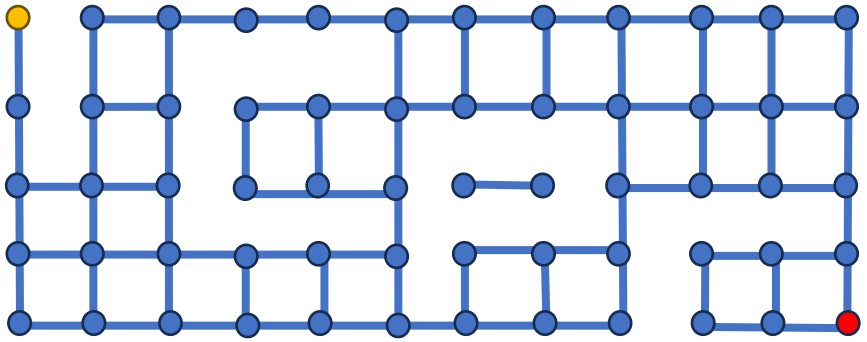
- **よく考えると、グラフの探索ですよね！**



課題２の説明

- **よく考えると、グラフの探索ですよね！**

```
int solve( ) {
  enqueue(queue, start);
  while(queue not empty) {
    node_t here = dequeue(queue);
    if(北に行けるか？ && 未訪問) {
      訪問記録
      enqueue(queue, 北へのノード);
    }
    /* 南・西・東同様
  }
  辿りつかなかった場合の処理
  return 答え;
}
```



スタート拠点からの距離を記録しながら幅優先探索すると問題解決！

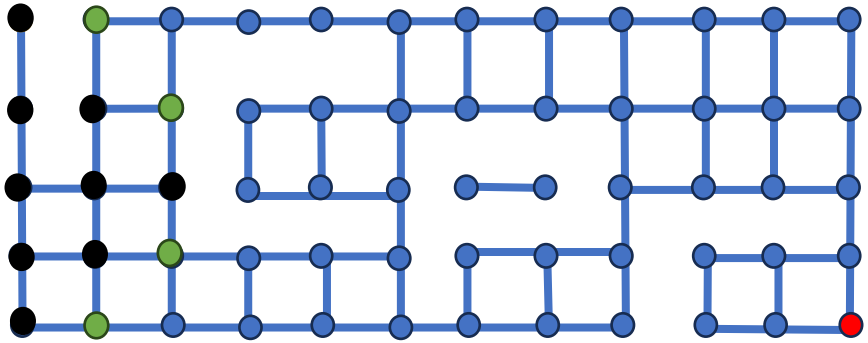
課題２の説明

問題分割→解けるのに何が必要でしょうか？

- **特定のノードから｛上下左右｝に進めるか？**
  → 実装済み
- **キューが要ります**
  →実装済み
- **各地点の訪問済み及び距離（ステップ数）**
  →２次元の配列
  - **０の値は未訪問**
  - **他の値はスタート地点からの距離**

課題２の説明

探索中の状態



(0, 0)

X 軸

Y 軸

1	6	0	0	0	0	0	0	0	0	0	0
2	5	6	0	0	0	0	0	0	0	0	0
3	4	5	0	0	0	0	0	0	0	0	0
4	5	6	0	0	0	0	0	0	0	0	0
5	6	0	0	0	0	0	0	0	0	0	0

w は迷路の幅
h は迷路の高さ

(w-1, h-1)

補足

今回、多くの部分を提供しております:

https://github.com/PatFin/AdvancedAlgorithms-SamplePrograms/tree/main/kadai2

実行すると、入力ファイルを読み込んで、solve 関数を呼び出して、自動で回答と比較して結果を表します。

実装するところが、solve 関数だけです。

補足

今回、多くの部分を提供しております。

問題を解く前に、

- **キューの操作　(enqueue / dequeue / qSize)**
- **上下左右進めるか (canGo)**

等を簡単なプログラムで試してみましょう。

課題提出について

提出するファイル：

- **kadai2.c**
- **report.pdf**
  - **幅優先探索で迷路の出口の最短ルートを求めるかを説明～半ページ**
  - **プログラムのコンパイル法・実行結果**
- **締め切り１１月６日 23:59　全員提出（未完全な状態でも）**

回答例を１１月７日公開します

→解けなかった人は再提出を認めます

回答例と比較してどこで間違っていたかをレポートで説明すること

- **締め切り１１月１３日 23:59**