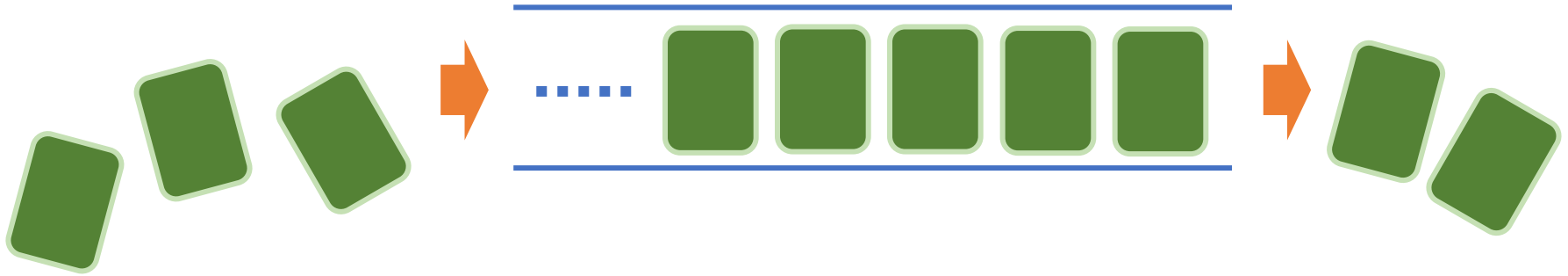
知能情報システム創成
(第２回)

西竜志：　　tnishi@fc.ritsumei.ac.jp

本日の演習内容

- 人工知能課題 2回目

  キューやスタックを用いて深さ優先探索と幅優先探索を実装する!

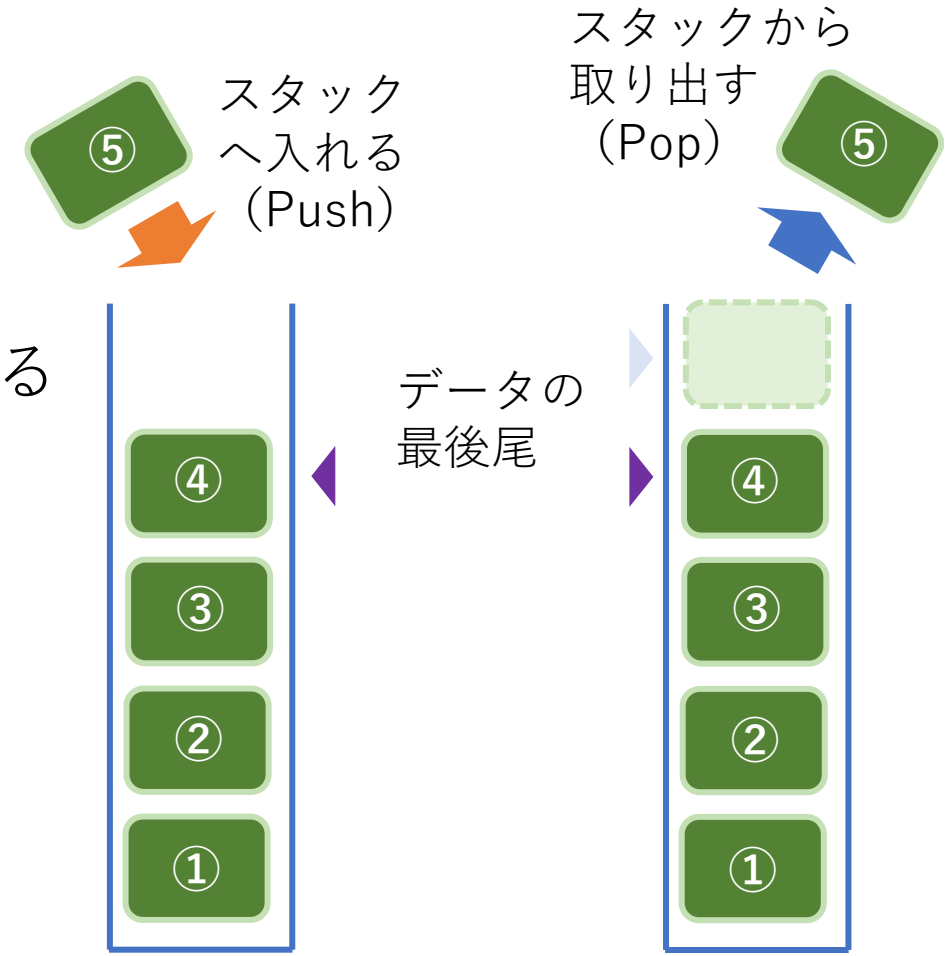


人工知能課題②

キューやスタックを用いて深さ優先探索と幅優先探索を実装する!

スタックについて（復習）

- データを上に積み上げていく(Push)
- 取り出す(Pop)ときは上から
  - ①→②→③→④→⑤の順に入れた場合、
    ⑤→④→③→②→①の順に取り出される



配列を用いた実装例（C言語）

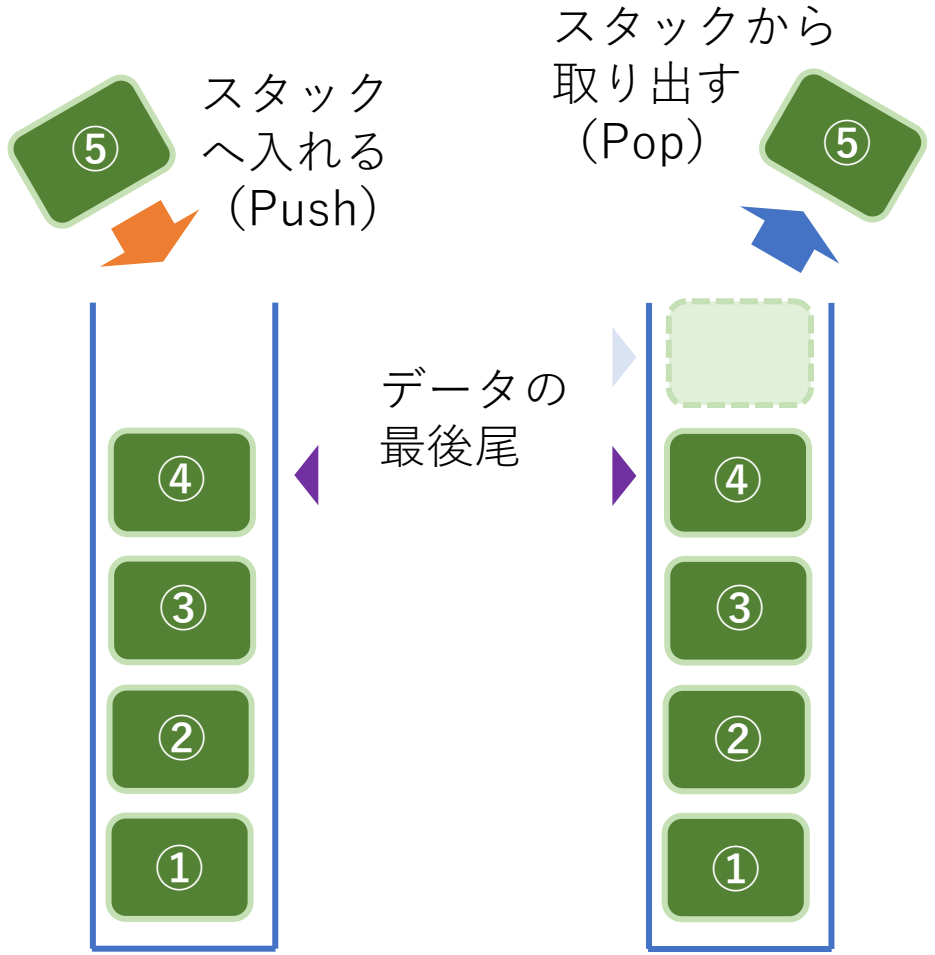
```
// スタック構造体
struct STACK {
    int tail; // データの最後尾
    int data[MAX_NUM]; // スタックされているデータ
};
```

配列を用いた実装例（python）

```
openList = []
```

スタックについて（復習）

- データを上に積み上げていく(Push)
  - 最後尾の位置を更新していく



配列を用いた実装例（C言語）

```c
// PUSHする（スタックにデータを入れる）関数
void push(struct STACK* stack, int input) {

    // スタックが満杯なら終了
    if (stack->tail >= MAX_NUM - 1) {
        printf("スタックが満杯でPUSHできません¥n");
        return;
    }

    // inputを最後尾の１つ後ろに格納
    stack->data[stack->tail + 1] = input;

    // データの最後尾を１つ後ろに移動
    stack->tail = stack->tail + 1;
}
```

配列を用いた実装例（python）

```
openList.insert(0, s)
```

スタックについて（復習）

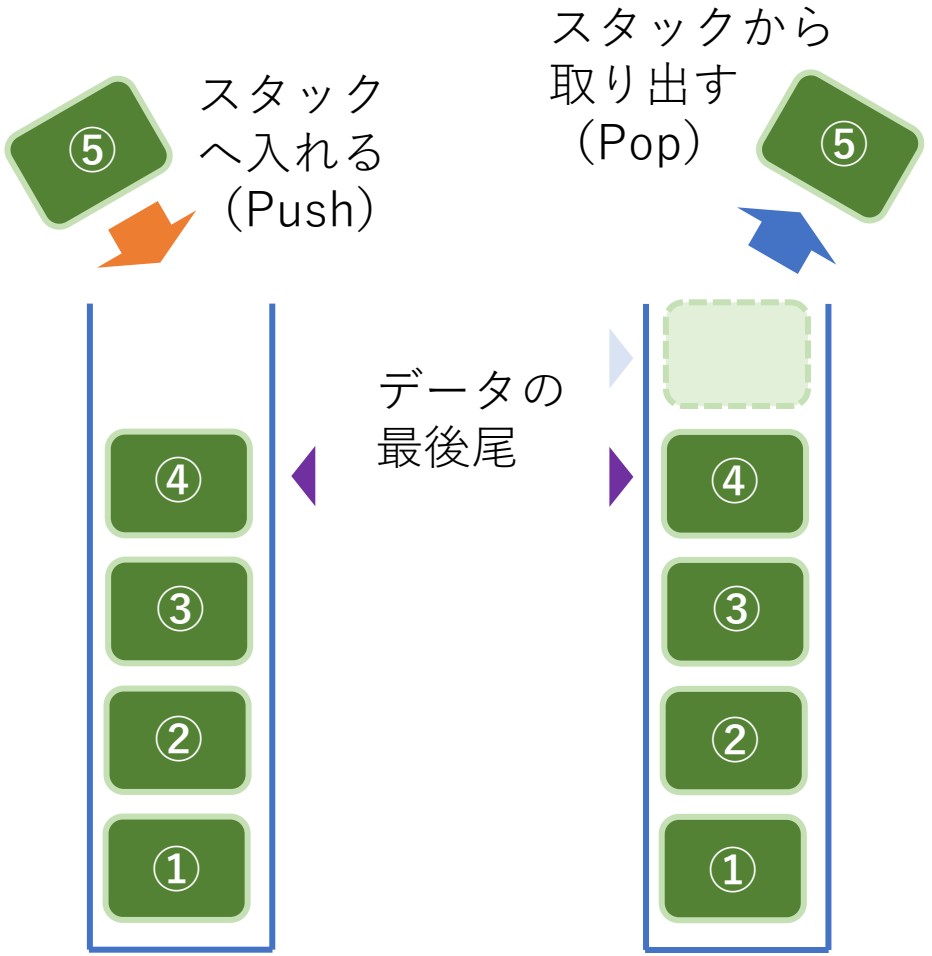
- 取り出す(Pop)ときは上から

配列を用いた実装例（C言語）

```c
// POPする（スタックからデータを取り出す）関数
int pop(struct STACK* stack) {
    int ret = 0;

    // スタックが空なら終了
    if (stack->tail == -1) {
        printf("スタックが空です¥n");
        return -1;
    }

    // データの最後尾からデータを取得
    ret = stack->data[stack->tail];

    // データの最後尾を１つ前にずらす
    stack->tail = stack->tail - 1;

    // 取得したデータを返却
    return ret;
}
```

配列を用いた実装例（python）

```
openList.pop(0)
```



演習用のpythonプログラム

スタックを実装した迷路探索プログラム

1. DFS.py をmanaba+Rからダウンロードする.

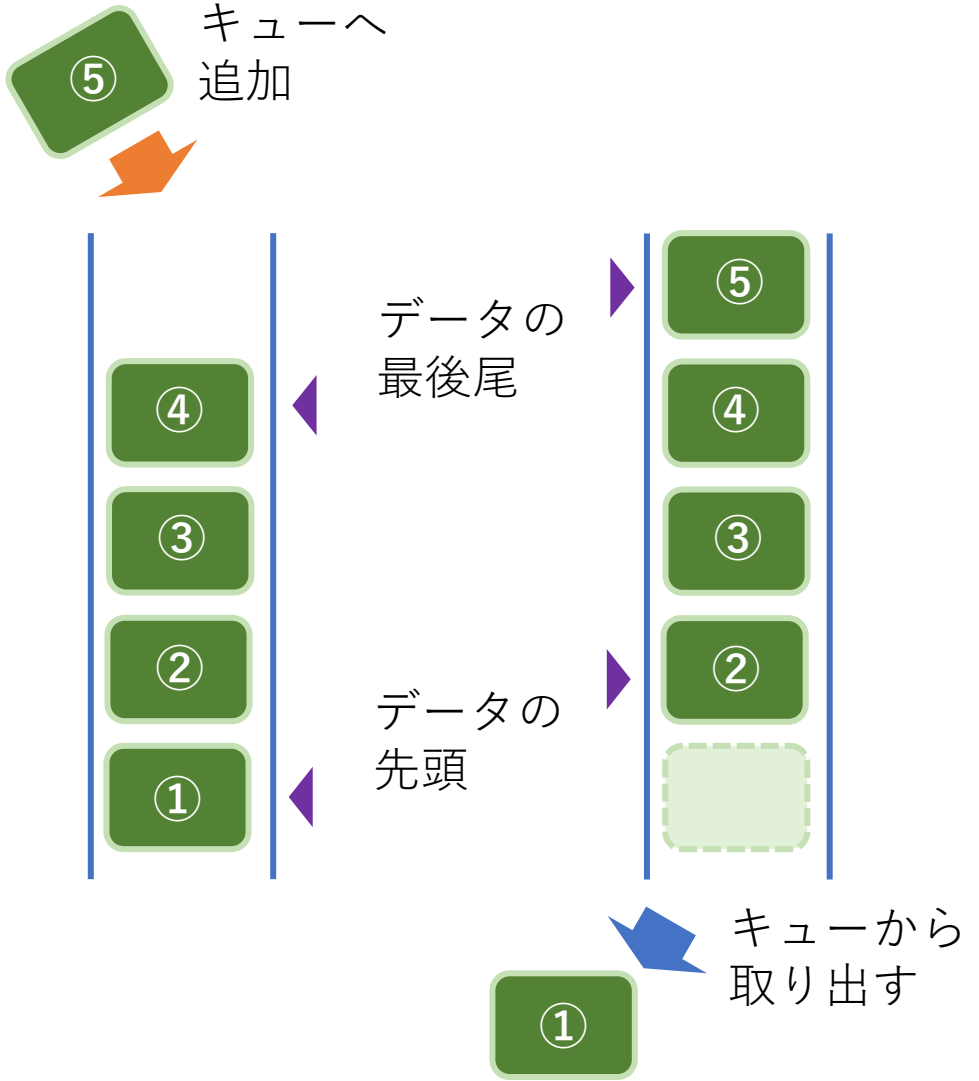
2. python edit_maze.py maze15x15.txt を実行し，15x15の迷路データを確認する.

3. python DFS.pyを実行してみる.

キューについて（復習）

- データを待ち行列に追加していく（Enqueue）
- 取り出すとき（Dequeue）は下から
  - ①→②→③→④→⑤の順に入れた場合、①→②→③→④→⑤の順に取り出される

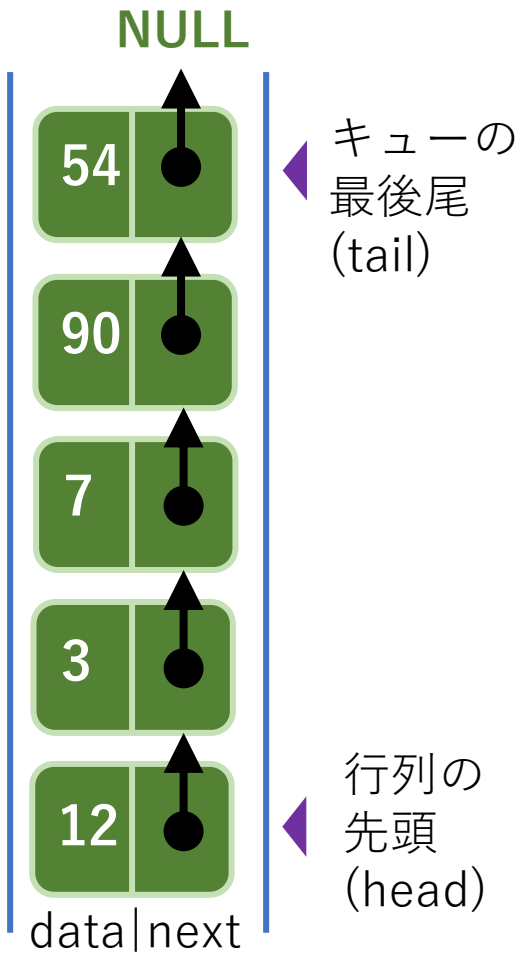
    データの先頭と最後尾の位置を知っている必要がある



キューについて（復習）

- データを待ち行列に追加していく（Enqueue）
- 取り出すとき（Dequeue）は下から
  - ①→②→③→④→⑤の順に入れた場合、①→②→③→④→⑤の順に取り出される

データの先頭と最後尾の位置を知っている必要がある



配列を用いた実装例（C言語）

```c
// ノード構造体
struct NODE {
    int data;
    struct NODE* next; // 次のノードのアドレス
};

// キューの構造体
struct QUEUE {
    struct NODE* head; // キューの先頭を指すポインタ
    struct NODE* tail; // キューの最後尾を指すポインタ
};
```

配列を用いた実装例（python）

```
openList = []
```

ノードをつないでキューを作り、
先頭と最後尾を記憶しておく

キューについて（復習）

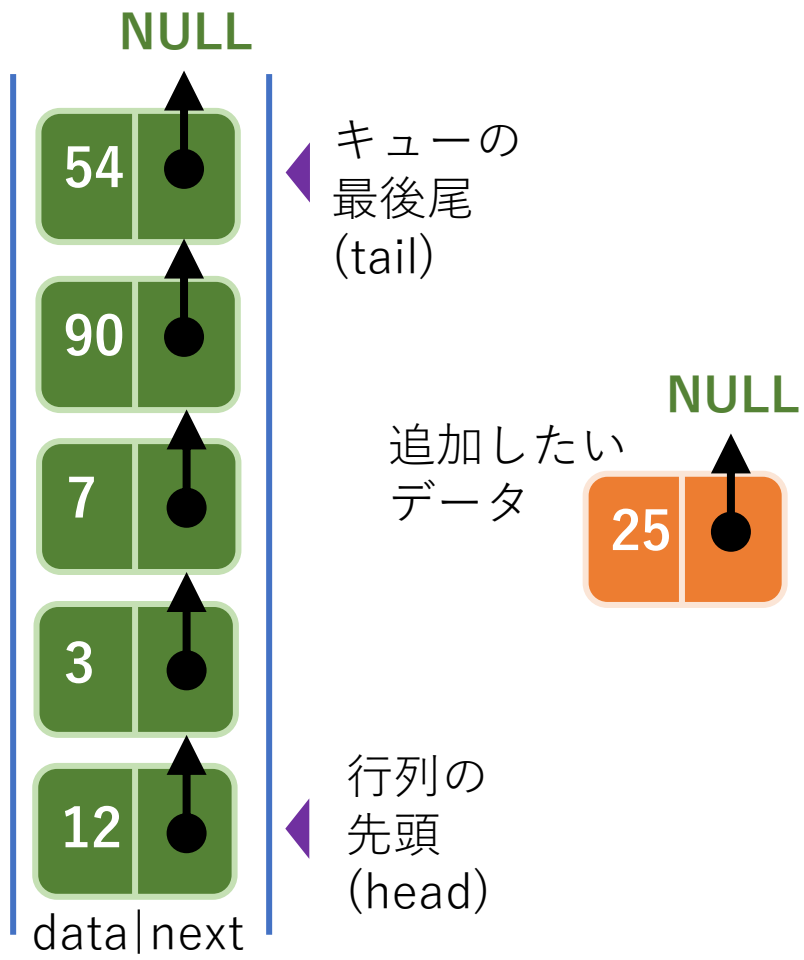
- データを待ち行列に追加していく

配列を用いた実装例（C言語）

```c
// エンキューする関数
void enqueue(struct QUEUE* queue, int input) {

    struct NODE* node;

    // 新しいノードのメモリを確保
    node = new (struct NODE);

    // メモリが足りなければ終了
    if (node == NULL) {
        printf("メモリが確保できないためエンキューできません¥n");
        return;
    }

    // 追加したノードデータを格納
    node->data = input;

    // 次のノードはないのでNULLを設定
    node->next = NULL;
```

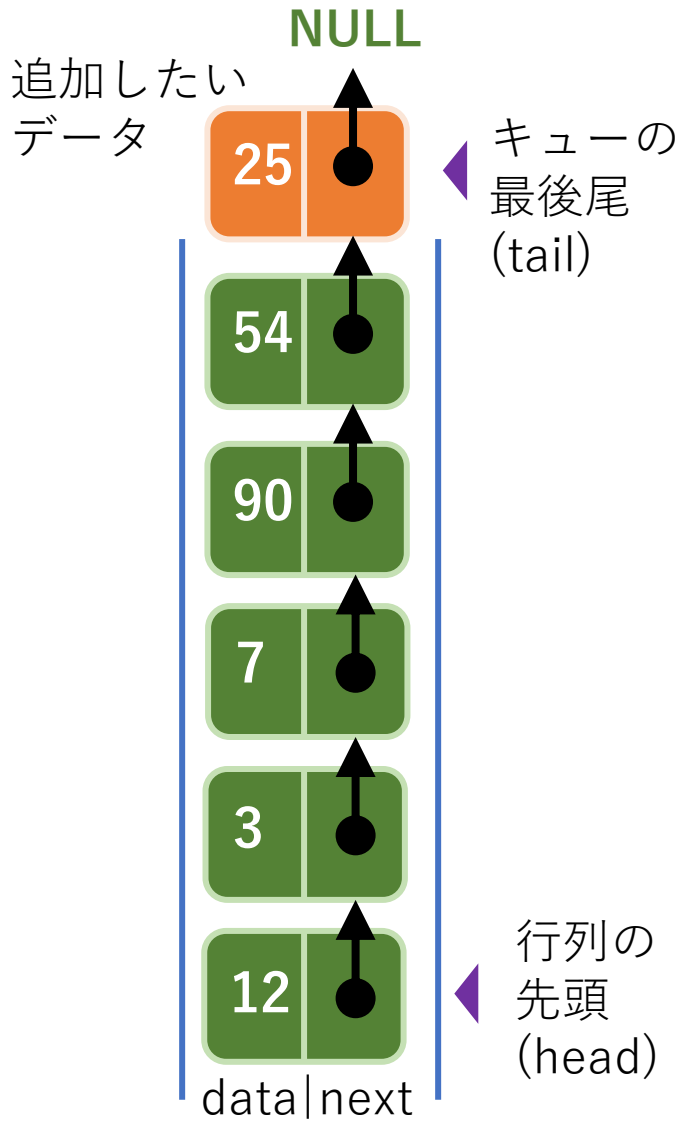
配列を用いた実装例（python）

```
openList.append(s)
```



キューについて（復習）

- データを待ち行列に追加していく



配列を用いた実装例（C言語）

```
    // キュー（行列）が空の場合
    if (queue->head == NULL && queue->tail == NULL) {
        // 先頭と最後尾は作成したノードになる
        queue->head = node;
        queue->tail = node;
        return;
    }

    //// 空でない場合はキューの最後尾にnodeを追加
    // 最後尾ノードの次のノードに追加したノードを指定する
    queue->tail->next = node;

    // tailを追加したノードで置き換えて追加完了
    queue->tail = node;
}
```

配列を用いた実装例（python）

```
openList.append(s)
```

キューについて（復習）

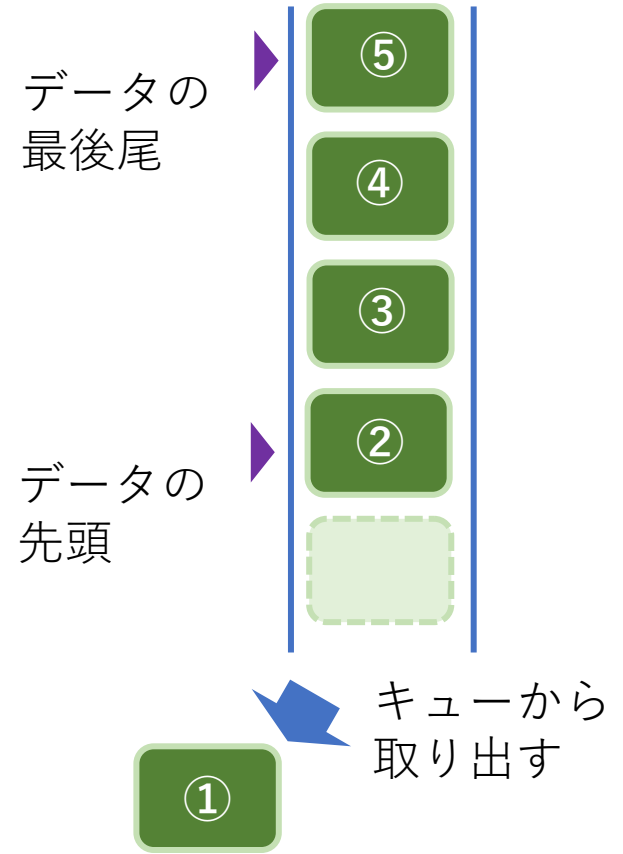
- 取り出すとき（Dequeue）は先頭から

配列を用いた実装例（C言語）

```
    //// 取り出してもキューが空にならない場合
    // headのアドレスをoldHeadに退避
    oldHead = queue->head;

    // headに、headが指すノードの次のノードを指させる
    queue->head = queue->head->next;

    // デキューしたノード（元々のhead）を解放
    delete oldHead;

    return ret;
}
```

配列を用いた実装例（python）

```
openList.pop(0)
```



キューについて（復習）

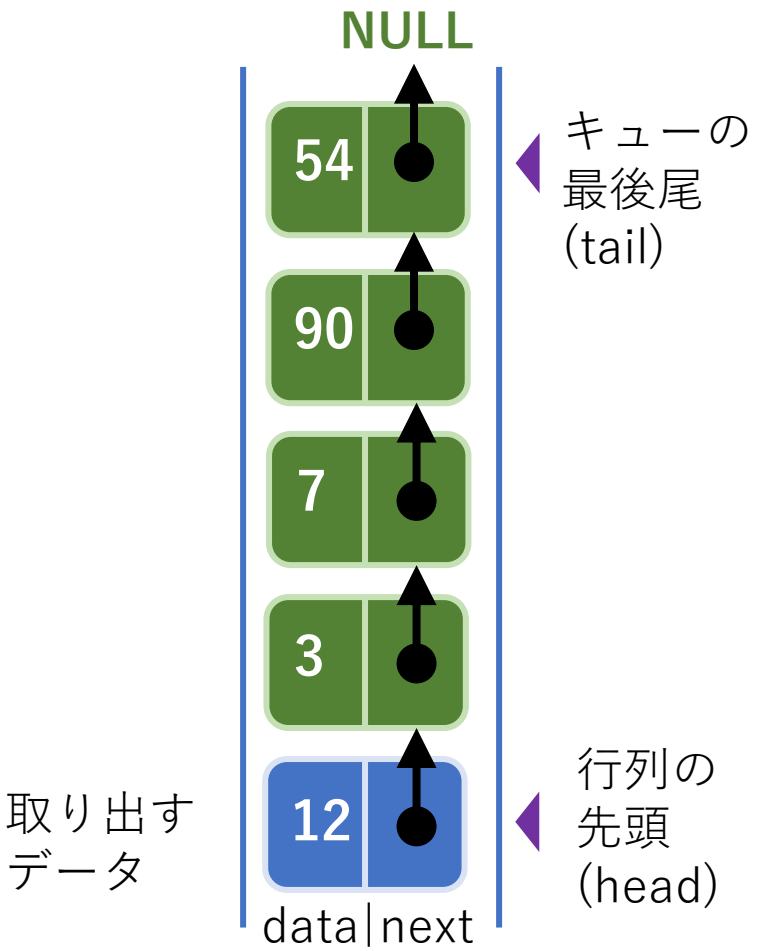
- 取り出すとき（Dequeue）は先頭から

配列を用いた実装例（C言語）

```
    //// 取り出してもキューが空にならない場合
    // headのアドレスをoldHeadに退避
    oldHead = queue->head;

    // headに、headが指すノードの次のノードを指させる
    queue->head = queue->head->next;

    // デキューしたノード（元々のhead）を解放
    delete oldHead;

    return ret;
}
```

配列を用いた実装例（python）

```
openList.pop(0)
```



キューについて（復習）

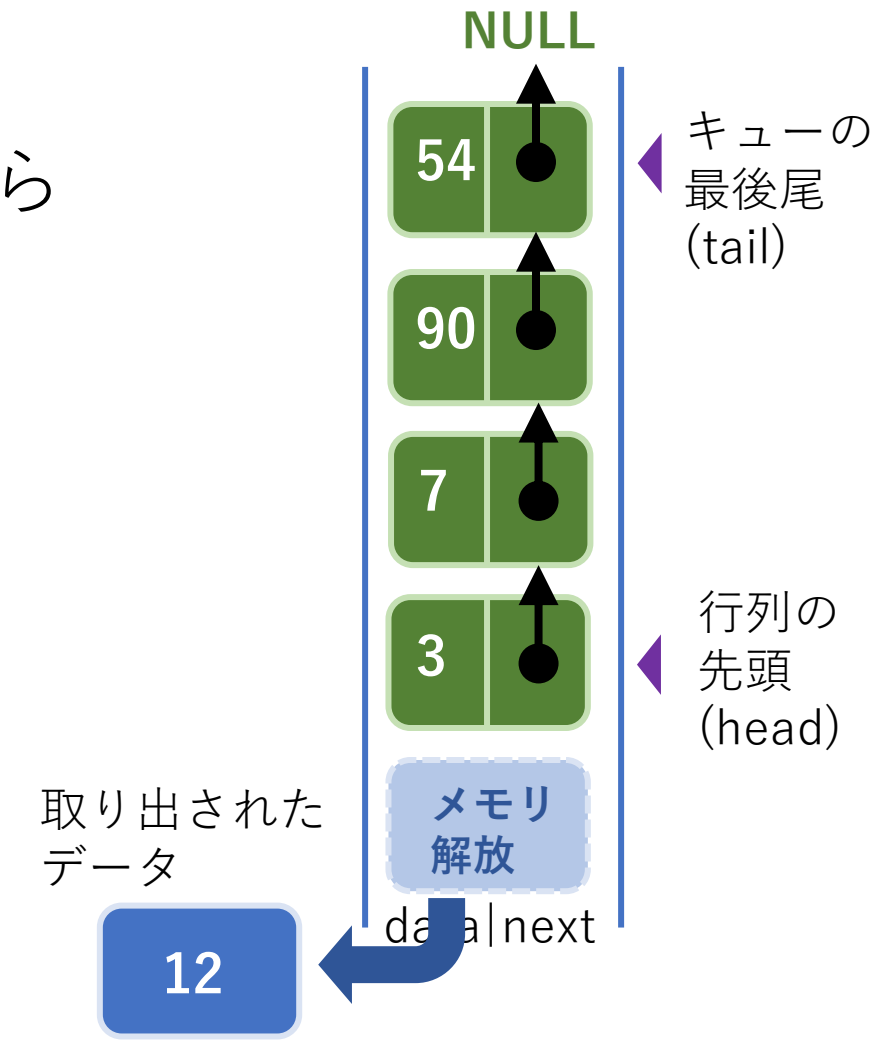
- 取り出すとき（Dequeue）は先頭から

配列を用いた実装例（C言語）

```
    //// 取り出してもキューが空にならない場合
    // headのアドレスをoldHeadに退避
    oldHead = queue->head;

    // headに、headが指すノードの次のノードを指させる
    queue->head = queue->head->next;

    // デキューしたノード（元々のhead）を解放
    delete oldHead;

    return ret;
}
```

配列を用いた実装例（python）

```
openList.pop(0)
```



キューを実装した幅優先探索のプログラム

参考（C++言語）

キューやスタックを用いた行列のシミュレーションを実装する

- ToDo: コンビニのレジ待ち行列の状態をシミュレーションする
  - お客さんは3秒ごとに入ってくるとする。
  - お客さんのもつデータ＝商品の数≒その会計にかかる時間とする。
    （データ４の場合4秒かかるとする）
    ※1-5の範囲とする
  - 待ち行列がどのように遷移していくか表示する。

参考資料

スタックとキュー

- 「【C言語/データ構造】スタックとキューの配列での実装方法」
  https://daeudaeu.com/stack_queue/

- 「【C言語】キューのポインタでの実装方法」
  https://daeudaeu.com/c-queue-ptr/