【課題 15：compare_time.py，libqueue.py，myqueue.py】

自作の PriorityQueue クラスと queue モジュールが提供する PriorityQueue クラスの実行時間を比較するクラスを実装せよ．

採点の都合上，以下の要件を満たすこととする：

(1) compare_time.py に，実行時間を計測するための QueueTimeMeasurement クラスを実装する；
  (a) 以下の点に注意して，イニシャライザを実装する；
    i. 配布した compare_time.py の「if __name__」以下の部分におけるインスタンス生成の方法を参考に，パラメータを考える；
    ii. パラメータとして受け取ったクラスのインスタンス（キューの本体）を生成し，キューの本体を QueueTimeMeasurement クラスのインスタンス変数として保持する；
    iii. 実行時間を計測するための計測開始時刻，計測終了時刻を表すインスタンス変数を用意し，0 で初期化する；
  (b) 以下の点に注意して，キューに対する enqueue と dequeue を多数回実行する exec メソッドを実装する；
    i. enqueue の実行回数をパラメータとして受け取るようにする．ただし，引数なしでこのメソッドを呼び出した際には，デフォルトで 10000 回 enqueue するようにする；
    ii. 上記で指定された回数だけ，0 以上 1 未満の乱数を enqueue する．この時，必ず enqueue という名前のメソッドで enqueue する；
    iii. そのうち，10 回に 1 回は dequeue する．この時，必ず dequeue という名前のメソッドで dequeue する；
    iv. このメソッドの最初から最後までの実行時間を process_time 関数により計測する；
  (c) 計測した実行時間を返す get_duration メソッドを実装する；
(2) 講義資料 P.29（prior_queue1.py）を参考にして，自作の PriorityQueue クラスを myqueue モジュールに実装する；
  (a) イニシャライザ，enqueue メソッド，dequeue メソッドを実装する；
  (b) 講義資料では，Monster クラスのインスタンスをキューの要素としていたが，課題 15 では float クラスのインスタンス（0 以上 1 未満の乱数）を要素とする点に注意する；
(3) 講義資料 P.31（prior_queue3.py）で使用している PriorityQueue クラスの派生クラスを libqueue モジュールに実装する；
  (a) 親クラスの put メソッドを呼び出すだけの enqueue メソッドを実装する（※代わりに，put メソッドオブジェクトを enqueue に代入する方法でも良い）；
  (b) 親クラスの get メソッドを呼び出して得られた値を return するだけの dequeue メソッドを実装する（※代わりに，get メソッドオブジェクトを dequeue に代入する方法でも良い）；
  (c) ※継承する理由は，標準ライブラリ queue モジュールの PriorityQueue クラスのメソッド名が put，get であり，これらを enqueue，dequeue という名前で呼び出せるようにしたいだけである．
(4) compare_time.py にあらかじめ書かれている部分は，変更しない；
(5) 不要なモジュールは import しない；
(6) デバッグ用の print など必要ないものはコメントアウトし，出力が図 1 になるようにする；

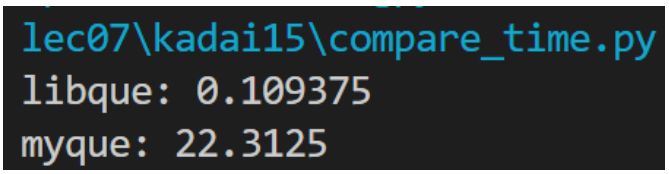

図 1　課題 15 の実行例：10 万回の enqueue（1 万回の dequeue）を実行した際の実行時間を比較している．

リスト 1 compare_time.py

```
import libqueue
import myqueue
from time import process_time
from random import random


class QueueTimeMeasurement:
    def __init__(self, klass):
        self._que = klass()
        self.begin = 0
        self.end = 0

    def exec(self, times=10000):
        self.begin = process_time()
        for i in range(times):
            self._que.enqueue(random())
            if i%10 == 0:
                self._que.dequeue()
        self.end = process_time()

    def get_duration(self):
        return self.end-self.begin


if __name__ == "__main__":
    libque = QueueTimeMeasurement(libqueue.PriorityQueue)
    libque.exec(100000)
    print(f"libque: {libque.get_duration()}")

    myque = QueueTimeMeasurement(myqueue.PriorityQueue)
    myque.exec(100000)
    print(f"myque: {myque.get_duration()}")
```

リスト 2 myqueue.py

```
class PriorityQueue:
    def __init__(self):
        self._lst = []

    def enqueue(self, item):
        self._lst.append(item)
        self._lst.sort(reverse=True)

    def dequeue(self):
        if len(self._lst) == 0:
            return None
        return self._lst.pop(0)
```

リスト 3 libqueue.py

```
import queue

# class PriorityQueue(queue.PriorityQueue):
# def enqueue(self, item):
# super().put(item)

# def dequeue(self):
# return super().get()


class PriorityQueue(queue.PriorityQueue):
    enqueue = queue.PriorityQueue.put
    dequeue = queue.PriorityQueue.get
```

【課題 16：shortest_path.py】

幅優先探索（Breadth First Search）により，迷路のスタートからゴールまでの最短経路を求め，最短経路上のマスを黄色に塗りつぶすコードを実装せよ．なお，マスの状態（state）の値を”R”にすることで，show_maze メソッドによる迷路描画において黄色に塗りつぶされるため，meza_maker.py は変更する必要はない（＝提出する必要はない）．

採点の都合上，以下の要件を満たすように，ShortestPath クラスを実装することとする：

(1) 講義資料 P.23〜25 を参考にし，幅優先探索を行う bfs メソッドを実装する；
　（a）訪れたマスに対して，どのマス（親）から訪れたかを表すインスタンス変数 parent に適切なオブジェクト（親となるマス）を代入する；
(2) 最短経路上のマスの状態（state）を”R”に設定する back_trace メソッドを実装する；
　（a）ゴールマスから順に親マスをたどって行き，スタートマスが見つかるまでたどることを繰り返す；
　（b）たどる途中で通過するマスの状態（state）を”R”に設定する；
(3) shortest_path.py にあらかじめ書かれている部分は，変更しない；
(4) 不要なモジュールは import しない；
(5) デバッグ用の print など必要ないものはコメントアウトし，出力が図 2 になるようにする；

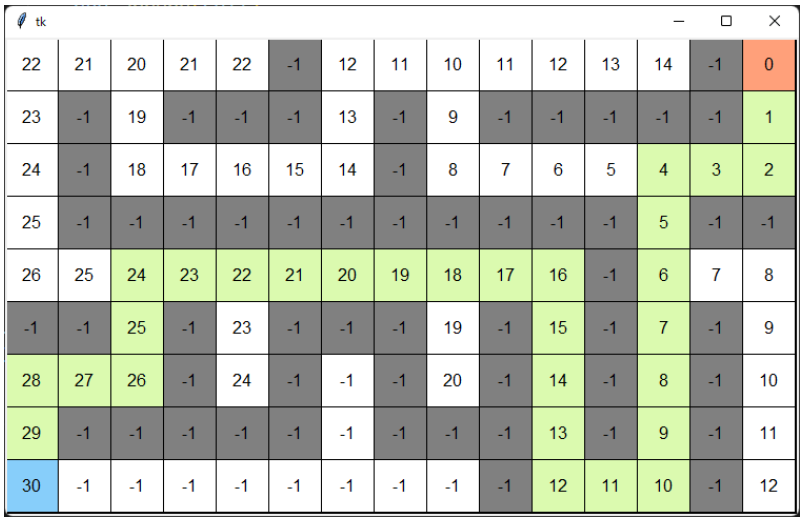

図 2　課題 16 の実行例：最短経路上のマスが黄色に塗られている．

リスト 4 shortest_path.py

```
from maze_maker import Maze, Cell
from pprint import pprint
from collections import deque

class ShortestPath:
    def __init__(self, map: Maze):
        self.map = map

    def bfs(self):
        que = deque()
        start: Cell = self.map.start
        start.dist = 0
        que.append(start)
        while True:
            try:
                current: Cell = que.popleft()
            except:
                break
            if current.state == "G":
                break
            for cell in self.map.get_adj(current):
                if cell.dist != -1: # 既に訪れていたら
                    continue
                cell.dist = current.dist+1
                cell.parent = current
                que.append(cell)

    def back_trace(self):
        current: Cell = self.map.goal
        while True:
            current = current.parent
            if current.state == "S":
                break
            current.state = "R"


if __name__ == "__main__":
    tate, yoko = 9, 15
    maze = Maze(tate, yoko)
    sp = ShortestPath(maze)
    sp.bfs()
    sp.back_trace()
    maze.show_maze()
```