

【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；

```
lec07\kadai15\compare_time.py
libque: 0.109375
myque: 22.3125
```

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

```
1 import libqueue
2 import myqueue
3
4 if __name__ == "__main__":
5     libque = QueueTimeMeasurement(libqueue.PriorityQueue)
6     libque.exec(100000)
7     print(f"libque: {libque.get_duration()}")
8
9     myque = QueueTimeMeasurement(myqueue.PriorityQueue)
10    myque.exec(100000)
11    print(f"myque: {myque.get_duration()}")
```

【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;

22	21	20	21	22	-1	12	11	10	11	12	13	14	-1	0
23	-1	19	-1	-1	-1	13	-1	9	-1	-1	-1	-1	-1	1
24	-1	18	17	16	15	14	-1	8	7	6	5	4	3	2
25	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5	-1	-1
26	25	24	23	22	21	20	19	18	17	16	-1	6	7	8
-1	-1	25	-1	23	-1	-1	-1	19	-1	15	-1	7	-1	9
28	27	26	-1	24	-1	-1	-1	20	-1	14	-1	8	-1	10
29	-1	-1	-1	-1	-1	-1	-1	-1	-1	13	-1	9	-1	11
30	-1	-1	-1	-1	-1	-1	-1	-1	-1	12	11	10	-1	12

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 2 shortest_path.py

```
1 from maze_maker import Maze, Cell
2 from pprint import pprint
3 from collections import deque
4
5 class ShortestPath:
6     def __init__(self, map: Maze):
7         self.map = map
8
9     def bfs(self):
10         pass
11
12     def back_trace(self):
13         pass
14
15 if __name__ == "__main__":
16     tate, yoko = 9, 15
17     maze = Maze(tate, yoko)
18     sp = ShortestPath(maze)
19     sp.bfs()
20     sp.back_trace()
21     maze.show_maze()
```
