

```

1 package problem3; /* 원을 렌더링하는 문제입니다.
2  * 렌더링은 보통 각 픽셀별로 어느 물체를 렌더링할지 결정하여 픽셀값을
  정합니다.
3  * 원을 순서대로 덧씌워 그리는 것이 아닌, 최종적으로 그릴 원을 찾아서
  그리는 것이 핵심!
4  * 원의 중심이 서브픽셀 단위(0.5 지점)인 것도 고려해야 합니다.
5  * 정수 계산의 정확도를 유지하기 위해 픽셀 위치를 2배로 하여
  서브픽셀을 정수값으로 바꾸어 계산합니다.*/
6
7 import java.util.Collections;
8 import java.util.PriorityQueue;
9
10 class Solution { public int[][] solution(int M, int
  N, int[][] shapes, int[] colors) {
11     int[][] img = new int[M][N]; for (int y = 0; y
  < M; y++) {for (int x = 0; x < N; x++) {
12         for (int i = shapes.length-1; i >= 0; i
  --) {int[] shape = shapes[i];int cx = shape[0],
13             cy = shape[1], r = shape[2];if (
  isInside(x, y, cx, cy, r)){img[y][x]=colors[i];
14             break;}}}}return img;}
15     // 픽셀 사각형 중 원의 중심에서 가장 가까운 지점이 원 내부일
  경우 내부로 판단
16     // 원의 중심과 같은 x좌표거나 y좌표일 경우 y축 또는 x축에서만
  가까워지므로 별도로 계산
17     // 그 외에는 x좌표와 y좌표 모두 가까운 위치를 찾아야 하므로,
  둘 모두 가까운 위치로 계산
18     boolean isInside(int x, int y, int cx, int cy,
  int r) {
19         double d2 = 0;if (x == cx) {d2 = (Math.abs(
  y-cy)-0.5);d2=d2*d2;} else if (y == cy)
20             {d2 = (Math.abs(x-cx)-0.5);d2=d2*d2;
21             } else {d2 = Math.pow(Math.abs(x - cx) - 0.
  5, 2) + Math.pow(Math.abs(y-cy)-0.5, 2);
22             }return d2 < r*r;}}
23 class Test{public static void main(String[] args){
24     int M = 10;int N = 15;int[][] shapes = {{5, 4,
  3}, {8, 5, 4}};int[] colors = {50, 200};
25     int[][] sol = new Solution().solution(M, N,
  shapes, colors);for (int[] s: sol)
26         {for (int n: s) {System.out.printf("%3d ", n);}

```

```

26 System.out.println();}}
27 package problem4;import java.util.Collections;
28     import java.util.PriorityQueue;
29 /* 최대힙을 이용하여 풀이할 수 있는 문제입니다. 현재 갈 수 있는 장소
   중 가장 큰 연료통을 찾아갑니다. 연료통을 더 얻지 않고
30 * 목적지에 도달할 수 있다면 곧바로 목적지로 갑니다. DP로 풀 수도
   있지만, DP로 풀 경우 불필요한 연산이 많아집니다.*/
31 class Solution { public int solution(int dest, int
   start, int[] station, int[] fuel) {
32     PriorityQueue<Integer> pq = new PriorityQueue<
   Integer>(Collections.reverseOrder());
33     int N = station.length;int currFuel = start;int
   currPos = 0;int count = 0;
34     for(int i=0;i<=N;i++){int currTarget;if (i == N
   ) { // (N+1)번째에 도착지가 있다고 생각
35         currTarget = dest;} else {currTarget =
   station[i];}
36         currFuel = currFuel - (currTarget - currPos
   );
37         // 현재 연료로 갈 수 있으면서 큰 연료부터 획득
38         while (currFuel < 0 && !pq.isEmpty()) {
   currFuel = currFuel + pq.poll();count++;}
39         // 연료가 모자라면 목적지에 도달할 수 없음
40         if (currFuel < 0) return -1;
41         // 연료통을 힙에 추가
42         if (i < N) {pq.offer(fuel[i]);currPos =
   currTarget;}}return count;}}
43 class Test{public static void main(String[] args){
44     int dest = 12;int start = 6;int[] station = {2
   , 3, 5, 10};int[] fuel = {5, 2, 3, 1};
45     System.out.println(new Solution().solution(dest
   , start, station, fuel));}}
46 package problem5;import java.util.HashSet;
47 import java.util.List;import java.util.
   PriorityQueue;import java.util.Set;
48 /* 상자가 이동할 최단거리와, 상자를 움직일 플레이어의 최단거리를 함께
   구하는 문제입니다.
49 * BFS, DFS의 조합으로 해결할 수 있으나, 여기에서는 A*
   알고리즘으로 구현한 코드를 소개합니다.
50 * 참조 코드와 리트코드의 원본 문제를 참고해 주세요!
51 * 참조 코드: https://leetcode.com/problems/minimum-

```

```

51 moves-to-move-a-box-to-their-target-location/
52 discuss/2257156/Python3-A*-beat-100
53 * 원본 문제: https://leetcode.com/problems/minimum-moves-to-move-a-box-to-their-target-location/*/
54 class Solution { int[][] map;int[][] moves = {{1, 0
55 }, {-1, 0}, {0, 1}, {0, -1}};
56     public int solution(int[][] map) {this.map =
57     map;int M = map.length, N = map[0].length;
58     int[] target = new int[2];int[] box = new
59     int[2];int[] person = new int[2];
60     for (int i = 0; i < M; i++) {for (int j = 0
61     ; j < N; j++) {if (map[i][j] == 4) {
62         target = new int[]{i, j};} else
63         if (map[i][j] == 3) {box = new int[]{i, j};
64         } else if (map[i][j] == 1) {person
65         = new int[]{i, j};}}
66     PriorityQueue<Item> pq = new PriorityQueue
67     <>();Set<List<Integer>> visited = new HashSet<>();
68     pq.offer(new Item(manhattanDist(box,
69     target), 0, box, person));
70     visited.add(List.of(box[0], box[1], person[
71     0], person[1]));
72     while (!pq.isEmpty()) {Item curr = pq.poll
73     ();int dist = curr.dist;box = curr.pos;
74     person = curr.pos2;if (box[0] == target
75     [0] && box[1] == target[1]) {return dist;}
76     for (int[] move: moves) {int[] newBox
77     = {box[0] + move[0], box[1] + move[1]};
78     int[] newPerson = {box[0] - move[0]
79     , box[1] - move[1]};
80     List<Integer> list = List.of(newBox
81     [0], newBox[1], box[0], box[1]);
82     if (isValid(newBox) && isValid(
83     newPerson) && !visited.contains(list)) {visited.add
84     (list);
85     pq.offer(new Item(dist+1+
86     manhattanDist(newBox,target),dist+1,newBox,box
87     ));}}return-1;}
88     boolean isValid(int[] pos) {int M = map.length
89     , N = map[0].length;return 0 <= pos[0] && pos[0] <
90     M

```

```

71         && 0 <= pos[1] && pos[1] < N && map[
pos[0]][pos[1]] != 2;}
72     int manhattanDist(int[] p1,int[] p2){return
Math.abs(p1[0] - p2[0]) + Math.abs(p1[1] - p2[1
]);}
73     boolean check(int[]s,int[]d,int[]box){
74         PriorityQueue<Item> pq = new PriorityQueue
<>();Set<List<Integer>> visited = new HashSet<>();
75         visited.add(List.of(s[0], s[1]));
76         visited.add(List.of(box[0], box[1]));pq.
offer(new Item(manhattanDist(s, d), 0,s));
77         while (!pq.isEmpty()) {Item curr = pq.poll
();int dist = curr.dist;int[] pos = curr.pos;
78             if (pos[0] == d[0] && pos[1] == d[1
]) {return true;}for (int[] move: moves) {
79                 int[]newPos={pos[0]+move[0],pos[1
]+move[1]};List<Integer>list=List.of(newPos[0],
newPos[1]);
80                 if (isValid(newPos) && !visited.
contains(list)) {visited.add(list);
81                 pq.offer(new Item(dist+1+
manhattanDist(newPos, d), dist+1, newPos));}}
return false;}}
82 class Item implements Comparable<Item> { int
manhattanDist, dist;int[] pos;int[] pos2;
83     public Item(int manhattanDist, int dist, int
[] pos) {this.manhattanDist = manhattanDist;
84         this.dist = dist;this.pos = pos;}
85     public Item(int manhattanDist, int dist, int
[] pos, int[] pos2) {this.manhattanDist =
manhattanDist;
86         this.dist = dist;this.pos = pos;this.pos2
= pos2;}
87     @Override public int compareTo(Item o) {
88         return manhattanDist -o.manhattanDist;}}
89 class Test{public static void main(String[] args){
90     int[][] map = {{2, 2, 2, 2, 2},
91                     {2, 0, 2, 2, 2},
92                     {4, 0, 0, 0, 0},
93                     {2, 0, 2, 2, 0},
94                     {2, 3, 2, 2, 0},

```

```
95         {2, 0, 2, 2, 0},  
96         {2, 1, 0, 0, 0}};System.  
    out.println(new Solution().solution(map));}}
```