

변형미로탐색

2013011800 구장희

코드설명

1. IDS

[data structure]

```
9  public class assignment_2013011800_IDS {
10      int[] maze, result;
11      int m, n, max = 250000;
12      int count = 0;
13      Node root;
14      boolean endcondition = false;
15      int[] dx = { 0, 0, 1, -1 };
16      int[] dy = { 1, -1, 0, 0 };
17      String path = null;
18  }
```

```
138 public class Node {
139     int x, y;
140
141     public Node(int _x, int _y) {
142         this.x = _x;
143         this.y = _y;
144     }
145
146     public int getX() {
147         return x;
148     }
149
150     public int getY() {
151         return y;
152     }
153
154 }
155 }
```

[flow]

main method에서 IDS class의 인스턴스를 생성하고, work() method에 절대경로(path)를 인자로 준다.
IDS class의 인스턴스가 work() method내에서

- input.txt file을 읽고 int[][] maze에 미로를 저장한다. 이때 시작노드를 root로 정한다.
- IDDFS() method에 인자로 root를 준다.
- endcondition이 true일때(도착점을 찾았을때) output.txt를 쓴다.

```
124 void work(String _path) {  
125     path = _path;  
126     readFile();  
127     IDDFS(root);  
128     if (endcondition) {  
129         writeFile();  
130     }  
131 }  
132  
133 public static void main(String[] args) {  
134     String path = args[0];  
135     new assignment_2013011800_IDS().work(path);  
136 }  
137
```

[IDS search]

- IDDFS
 - depth가 0부터 max(500*500) 까지 DLS() method를 호출한다.
 - 인자로써 현재 maze의 복사본과, 시작노드(root), 해당 깊이를 준다.
 - endcondition==true일때 return한다.
- DLS
 - depth==0이면 시행을 종료한다.
 - depth>0 이면, 갈수있는 모든 노드에 대해서(배열의 인덱스가 범위안에있고, 인접하고 벽이 아닌경우)
 - 도착점이라면 endcondition=true로 바꾸고 현재 미로의 상태를 저장하고 리턴한다.
 - 갈수있는 지점이라면(value==2) 현재 미로의 복사본에 5를 쓰고 자식노드로 재귀호출한다.

```
45 void IDDFS(Node root) {
46     for (int depth = 0; depth < max; depth++) {
47         DLS(copyarr(maze), root, depth);
48         if (endcondition)
49             return;
50     }
51     return;
52 }
53
54 void DLS(int[][] curmaze, Node node, int depth) {
55     count++;
56     if (depth == 0) {
57         return;
58     } else if (depth > 0) {
59         for (int i = 0; i < 4; i++) {
60             int childx = node.getX() + dx[i];
61             int childy = node.getY() + dy[i];
62             if (isvalid(childx, childy) && curmaze[childx][childy] != 1) {
63                 if (curmaze[childx][childy] == 4) {
64                     endcondition = true;
65                     result = copyarr(curmaze);
66                     break;
67                 } else if (curmaze[childx][childy] == 2) {
68                     int[][] tarr = copyarr(curmaze);
69                     tarr[childx][childy] = 5;
70                     DLS(tarr, new Node(childx, childy), depth - 1);
71                 }
72             }
73         }
74     }
75     return;
76 }
```

2. GBS

[data structure]

- key point
 - heuristic : 모든 도착점까지의 맨허튼 거리 중 최소거리

```
13 public class assignment1_2013011800_GBS {
14     int[] maze, visited;
15     int n, n, step = 0, max = 250000;
16     Node root;
17     boolean endcondition = false;
18     String path = null;
19     int[] dx = { 0, 0, 1, -1 };
20     int[] dy = { 1, -1, 0, 0 };
21     List<Node> destinations = new ArrayList<Node>();
22     Node fNode;
23     Comparator<Node> comparator = new NodeComparator();
24     PriorityQueue<Node> queue = new PriorityQueue<Node>(max, comparator);
25 }
```

```
147 public class Node {
148     int x, y, h = 250000;
149     Node parent;
150
151     public Node(int _x, int _y) {
152         this.x = _x;
153         this.y = _y;
154     }
155
156     public Node(Node par, int _x, int _y) {
157         this.parent = par;
158         this.x = _x;
159         this.y = _y;
160     }
161
162     public int getX() { return x; }
163
164     public int getY() { return y; }
165
166     public int getH() { return h; }
167
168     public void setH() {
169         for (Node node : destinations) {
170             this.h = Math.min(h, Math.abs(node.getX() - x) + Math.abs(node.getY() - y));
171         }
172     }
173 }
```

```

175 public class NodeComparator implements Comparator<Node> {
176     @Override
177     public int compare(Node o1, Node o2) {
178         if (o1.getH() < o2.getH()) {
179             return -1;
180         } else if (o1.getH() > o2.getH()) {
181             return 1;
182         } else {
183             return 0;
184         }
185     }
186 }

```

- 그 heuristic를 가지고 priority queue 이용(비교는 comparator 사용)

[flow]

main method에서 GBS class의 인스턴스를 생성하고, work() method에 절대경로(path)를 인자로 준다.
GBS class의 인스턴스가 work() method내에서

- input.txt file을 읽고 int[][] maze에 미로를 저장한다. 이때 시작노드를 root로 정한다.
- 또한 모든 도착점의 위치를 Node의 list로 보관하고, Node class의 setH()에서 이를 토대로 맨허튼 거리로 휴리스틱을 정의한다.
- GBS() method에 인자로 root를 준다.
- endcondition이 true일때(도착점을 찾았을때) output.txt를 쓴다.

```

131
132 void work(String _path) {
133     path = _path;
134     readFile();
135     GBFS(root);
136     if (endcondition) {
137         drawpath(fNode);
138         writeFile();
139     }
140 }
141
142 public static void main(String[] args) {
143     String path = args[0];
144     new assignment1_2013011800_GBS().work(path);
145 }
146

```

[GBS search]

- GBFS
 - maze의 크기만큼 visited 배열을 만들고 초기화한다.
 - root가 setH()를 불러서 heuristic을 계산하고 root를 queue에 넣는다.
 - priority queue에서 하나의 노드를 poll하고(방문하고), 그 노드의 자식노드 중 방문하지않은 노드들 중
 - 갈 수 있는 노드이면 , 자식노드를 생성하고 heuristic을 정의하고 queue에 넣는다.
 - 도착점이라면 , endcondition=true로 바꾸고, 그 자식노드를 마지막 노드로 생성하고 종료한다.

```
55 void GBFS(Node root) {
56     visited = new int[m][n];
57     for (int i = 0; i < m; i++) {
58         for (int j = 0; j < n; j++) {
59             visited[i][j] = 0;
60         }
61     }
62     root.setH();
63     queue.add(root);
64     while (queue.size() != 0 && !endcondition) {
65         Node popped = queue.poll();
66         step++;
67         visited[popped.getX()][popped.getY()] = 1;
68         for (int i = 0; i < 4; i++) {
69             int childx = popped.getX() + dx[i];
70             int childy = popped.getY() + dy[i];
71             if (isvalid(childx, childy) && visited[childx][childy] == 0) {
72                 if (maze[childx][childy] == 2) {
73                     Node child = new Node(popped, childx, childy);
74                     child.setH();
75                     queue.add(child);
76                 } else if (maze[childx][childy] == 4) {
77                     endcondition = true;
78                     fNode = new Node(popped, childx, childy);
79                     break;
80                 }
81             }
82         }
83     }
84 }
85
```

3. ASS

[data structure]

- heuristic 함수의 정의를 제외하고는 GBS와 같다.

- key point

- $F = G + H$ 에서 , F의 값을 comparator를 통해 비교하면서 priority queue를 사용한다.
- G : 시작점부터 실제로 이동한 거리
- H : 가장 가까운 도착점 까지의 맨허튼 거리

* F,G,H의 정의부분.

* setF()의 인자로 g값(시작점으로 부터 실제 이동한 거리)를 주면서 setH() 함수를 호출하고

* 그 두 값을 더한 f를 반환한다.

```
183 public void setH() {
184     for (Node node : destinations) {
185         this.h = Math.min(h, Math.abs(node.getX() - x) + Math.abs(node.getY() - y));
186     }
187 }
188
189 public void setF(int _g) {
190     setG(_g);
191     setH();
192     this.f = g + h;
193 }
194
195 public int getF() {
196     return g + h;
197 }
198 }
199
```

[flow]

GBS와 동일하다.

```
131
132  void work(String _path) {
133      path = _path;
134      readFile();
135      ASSearch(root);
136      if (endcondition) {
137          drawpath(fNode);
138          writeFile();
139      }
140  }
141
142  public static void main(String[] args) {
143      String path = args[0];
144      new assignment1_2013011800_ASS().work(path);
145  }
```


[ASS Search]

62,74번째 줄에서 heuristic을 설정하는 부분을 제외하고는 GBS와 전부 동일하다.

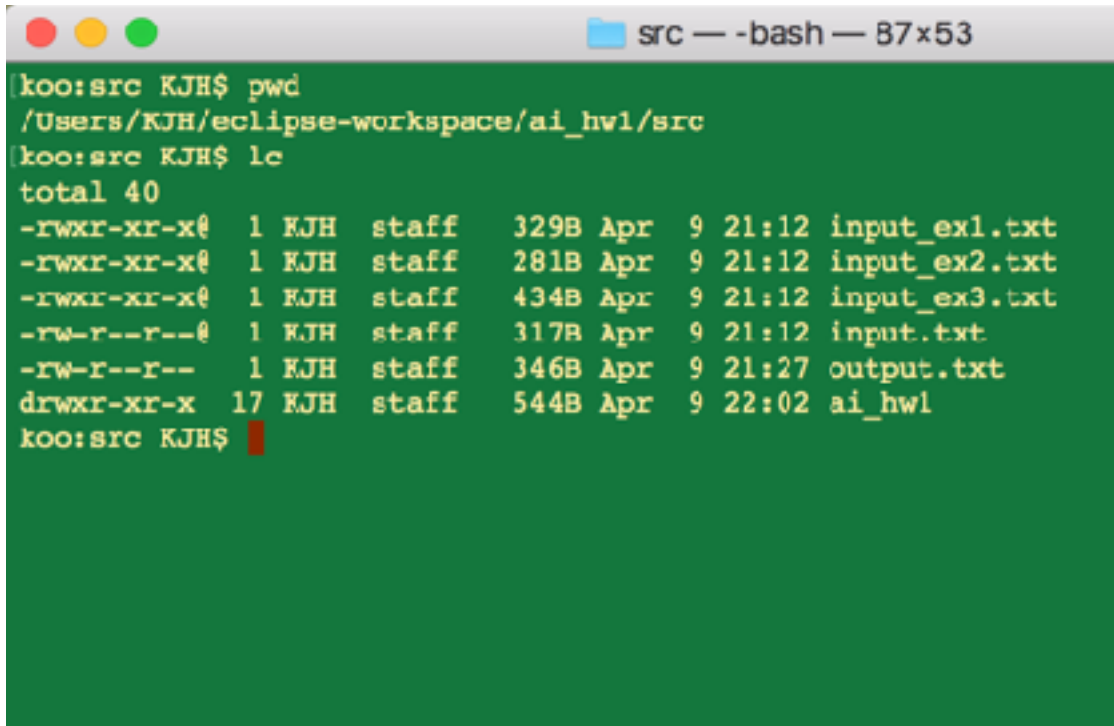
```
55 void ASSearch(Node root) {
56     visited = new int[m][n];
57     for (int i = 0; i < m; i++) {
58         for (int j = 0; j < n; j++) {
59             visited[i][j] = 0;
60         }
61     }
62     root.setF(0);
63     queue.add(root);
64     while (queue.size() != 0 && !endcondition) {
65         Node popped = queue.poll();
66         step++;
67         visited[popped.getX()][popped.getY()] = 1;
68         for (int i = 0; i < 4; i++) {
69             int childx = popped.getX() + dx[i];
70             int childy = popped.getY() + dy[i];
71             if (isValid(childx, childy) && visited[childx][childy] == 0) {
72                 if (maze[childx][childy] == 2) {
73                     Node child = new Node(popped, childx, childy);
74                     child.setF(popped.getG() + 1);
75                     queue.add(child);
76                 } else if (maze[childx][childy] == 4) {
77                     endcondition = true;
78                     fNode = new Node(popped, childx, childy);
79                     break;
80                 }
81             }
82         }
83     }
84 }
```

실험결과

- IDS
 - optimal path가 존재한다면 항상 찾는다.
 - 탐색 횟수가 매우 많다.
- GBS
 - 방문한 노드를 다시 방문하지 않게 해놓으면 항상 답을 찾는다.
 - optimal을 보장하지는 않는다.
 - 탐색횟수가 적다.
- ASS
 - 항상 답을 찾으며 optimal path이다.
 - 탐색횟수가 매우 적다.

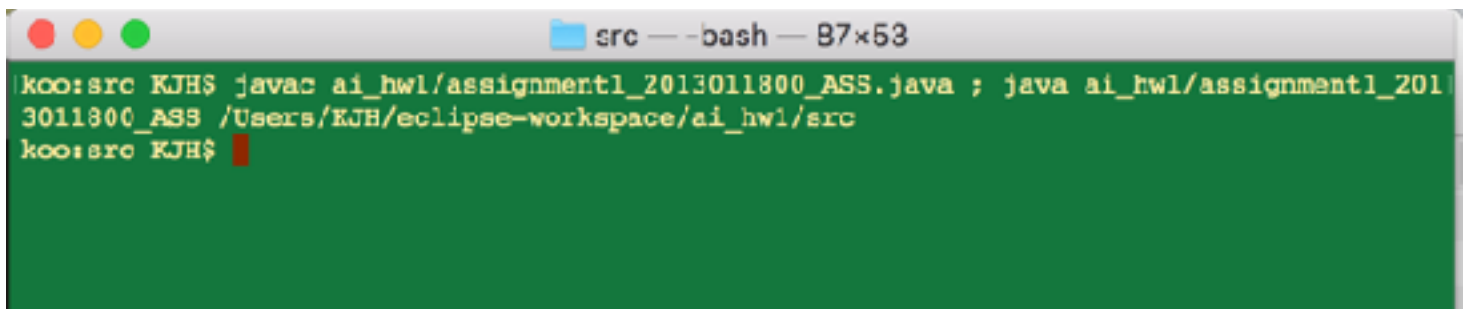
컴파일방법

- 환경
 - OS : Mac OS
 - Language : Java
- 실행 이전

A terminal window titled 'src — -bash — 87x53' with a green background. It shows the output of 'pwd' and 'ls' commands. The 'ls' command lists files in the current directory: input_ex1.txt, input_ex2.txt, input_ex3.txt, input.txt, output.txt, and ai_hw1 (a directory).

```
koo:src KJH$ pwd
/Users/KJH/eclipse-workspace/ai_hw1/src
koo:src KJH$ ls
total 40
-rwxr-xr-x@ 1 KJH  staff   329B Apr  9 21:12 input_ex1.txt
-rwxr-xr-x@ 1 KJH  staff   281B Apr  9 21:12 input_ex2.txt
-rwxr-xr-x@ 1 KJH  staff   434B Apr  9 21:12 input_ex3.txt
-rw-r--r--@ 1 KJH  staff   317B Apr  9 21:12 input.txt
-rw-r--r--  1 KJH  staff   346B Apr  9 21:27 output.txt
drwxr-xr-x 17 KJH  staff   544B Apr  9 22:02 ai_hw1
koo:src KJH$
```

- 컴파일 방법
 - 컴파일: `$javac ai_hw1/assignment1_2013011800_ASS.java`
 - 실행: `$java ai_hw1/assignment1_2013011800_ASS [절대경로]`

A terminal window titled 'src — -bash — 87x53' with a green background. It shows the execution of 'javac' and 'java' commands to compile and run the program.

```
koo:src KJH$ javac ai_hw1/assignment1_2013011800_ASS.java ; java ai_hw1/assignment1_2013011800_ASS /Users/KJH/eclipse-workspace/ai_hw1/src
koo:src KJH$
```