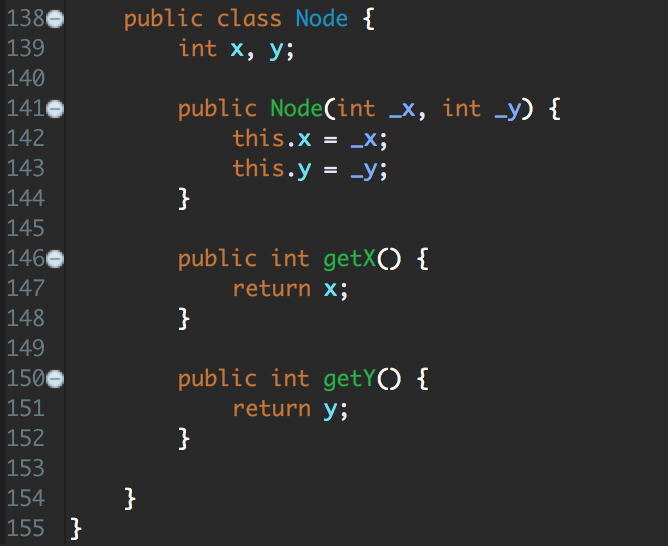
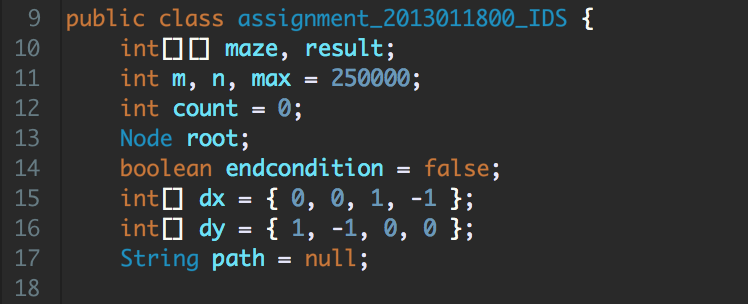
인공지능

변형미로탐색

**2013011800** 구장회

**#** 코드설명

1. **IDS**

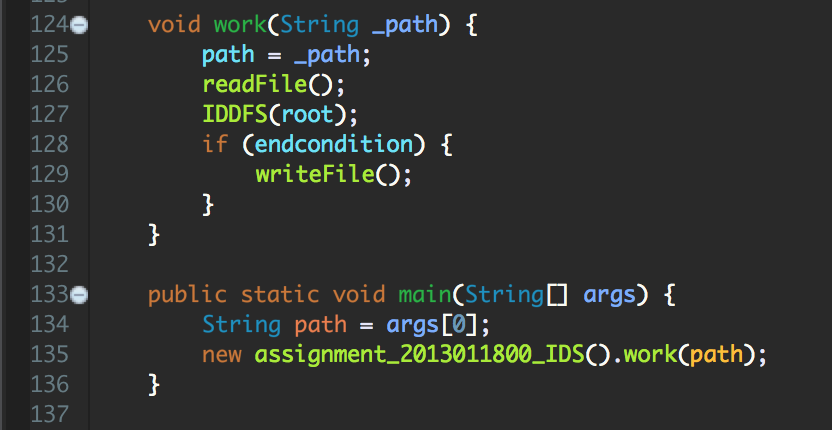
[data structure]

[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.

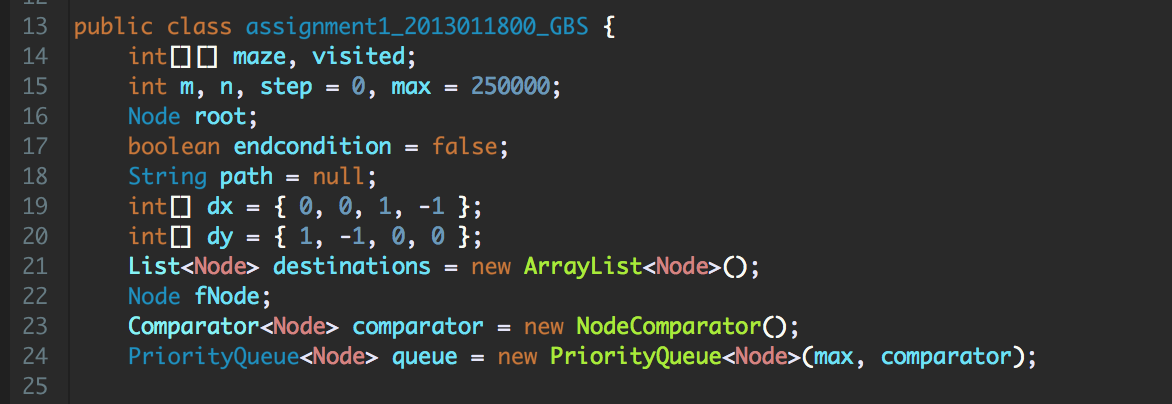
****

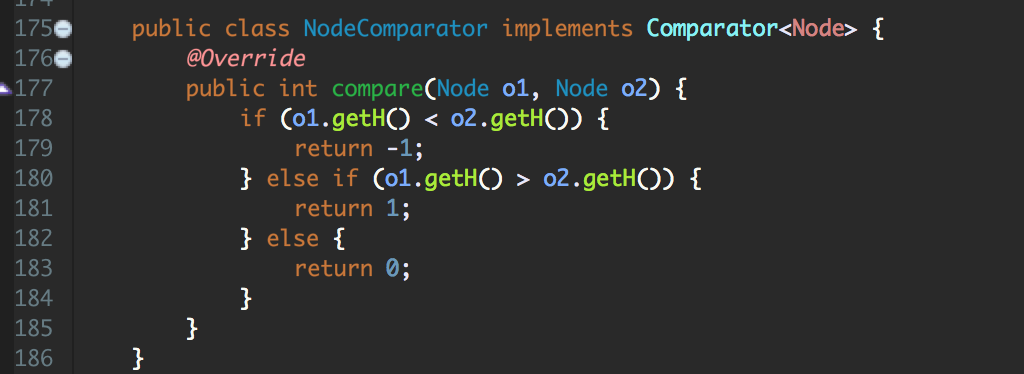
[IDS search]

* IDDFS
  + depth가 0부터 max(500\*500) 까지 DLS() method를 호출한다.
  + 인자로는 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 endcontidion=true로 바꾸고 현재 미로의 상태를 저장하고 리턴한다.
    - 갈수있는 지점이라면(value==2) 현재 미로의 복사본에 5를 쓰고 자식노드로 재귀호출한다.

**2. GBS**

[data structure]

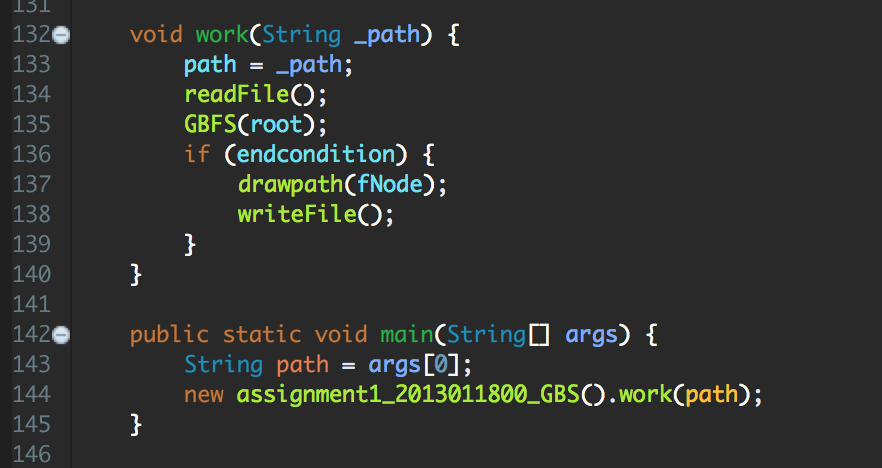
* key point
  + heuristic : 모든 도착점까지의 맨허튼 거리 중 최소거리
  + 그 heuristic를 가지고 priority queue 이용(비교는 comparator 사용)

****

[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.

[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.

****

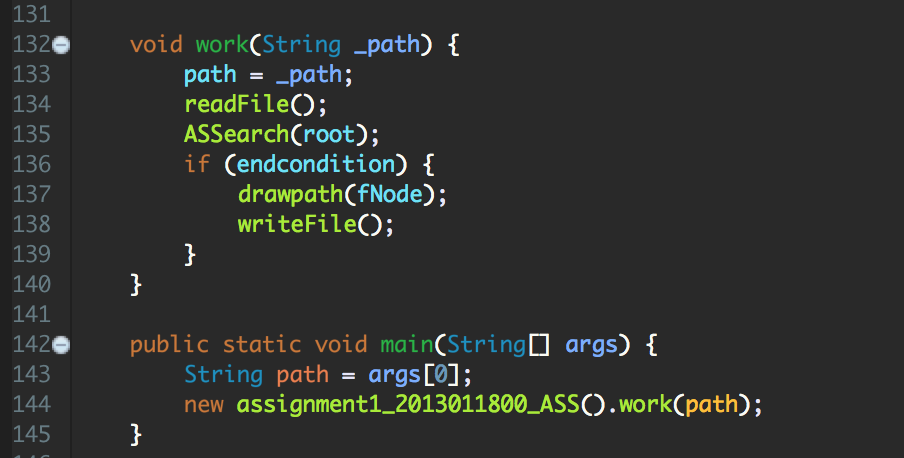
**3. ASS**

[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.

[flow]

GBS와 동일하다.

[ASS Search]

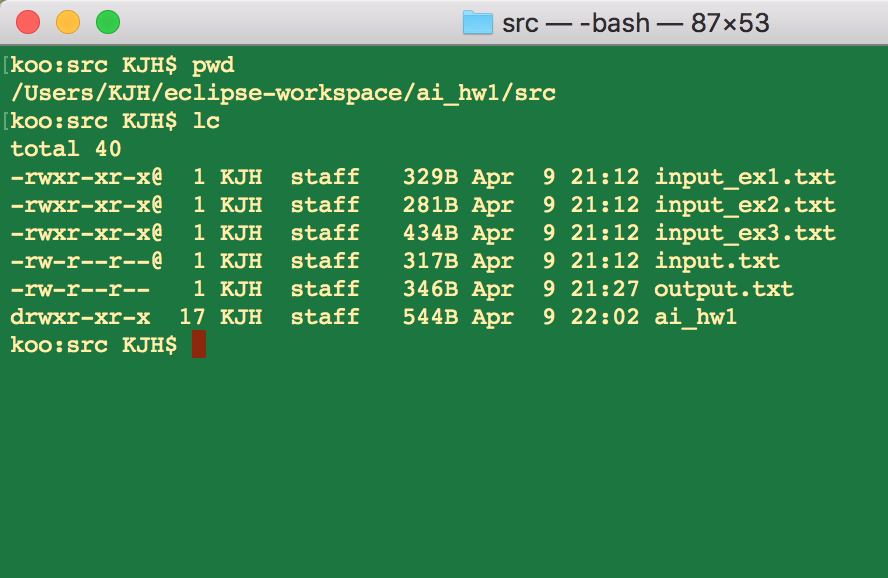
62,74번째 줄에서 heuristic을 설정하는 부분을 제외하고는 GBS와 전부 동일하다.



**#** 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
  + 컴파일 방법
    - 컴파일: $javac ai\_hw1/assignment1\_2013011800\_ASS.java
    - 실행: $java ai\_hw1/assignment1\_2013011800\_ASS [절대경로]