알고리즘분석

**HW#4 Report**

2011147068 김정환

2011147115 허재화

2012147562 최인호

1. **목차**
   1. **문제 분석**
   2. **알고리즘**
   3. **구현**
      1. **Class 설명**
      2. **알고리즘에 대한 세부구현**
   4. **토의 과정**
      1. **Big-O notation을 통한 분석**
      2. **구현 과정에서 겪은 어려움**
2. **문제 분석**

입력의 첫 줄에 테스트케이스의 개수가 들어온다. 두 번째 줄부터 각 테스트케이스의 column개수가 들어오고, 4줄에 걸쳐서 column개수만큼의 숫자들이 들어온다. 이 숫자는 정수이다. 그러면 column의 개수를 column\_num이라 했을 때 (4\*column\_num)크기의 격자판이 생성된다. 그때 다음 조건들을 만족하도록 흑돌과 백돌을 위치시켜서 그 합이 최대가 될 때, 그 값을 구해야 한다.

조건1. 각 돌들은 서로 이웃하게 놓일 수 없다(대각선으로 이웃하는 것은 가능)

조건2. 각 돌들은 판 위의 정확히 한 위치(하나의 숫자가 들어있는)에 놓여야한다.

조건3. 놓을 수 있는 흑돌과 백돌의 최대개수는 각각 (2\*column\_num)개 이다.

조건4. 흑돌이 놓인 위치에 있는 음수값은 그에 대응하는 양수값으로 바꿔서 더한다.

위의 조건에서 가장 먼저 알 수 있었던 것은, Board위의 값으로 음수가 들어오는 경우는 그 값을 양수로 처리해도 된다는 것이다. 이유는 다음과 같다. 조건3에서 놓을 수 있는 흑돌과 백돌의 최대개수는 각각 (2\*column\_num)개 이다. 라고 하였고, 조건1 에서 각 돌들은 서로 이웃하게 놓일 수 없다(대각선으로 이웃하는 것은 가능) 이라고 하였다. 그러면, 각 column은 4개의 값으로 이루어져 있고, 각 column에 놓을 수 있는 돌의 개수는 최대 2개이다. 3개를 놓는 순간 무조건 최소 두 개의 돌은 이웃하게 되기 때문이다. 그러면 모든 column에 돌은 최대로 놓아도 Board 전체에 놓이는 돌의 개수는 2\*column\_num개가 됨을 알 수 있다. 여기서 얻을 수 있는 것은, Board위에서 선택하게 될 수의 최대개수가 2\*column\_num개인데 그 수들이 전부 음수여도 흑돌은 모자라지 않는다는 것이다. 그러므로 처음부터 값으로 음수가 들어오면 양수로 처리하고 그 위치가 선택됐다면 그곳엔 흑돌을 놓는 것으로 가정하고 시작해도 문제가 없다. 이것을 통해 문제를 좀더 간단하게 생각할 수 있다.

* 4\*column\_num의 Board위에서 서로 이웃하지 않도록(대각선으로 이웃하는 것은 가능) 값들을 골라서 더했을 때, 그 값의 최댓값을 구하여라.

1. **알고리즘**

이 문제에서 우리가 사용한 알고리즘은 Dynamic Programming이다. 이 문제에서 어려웠던 점은 n번째 column에 돌을 놓은 방법이 n-1번째 column의 돌을 놓는 방법에 영향을 주고, 그것은 또 연쇄적으로 그 이전의 column들에 돌을 놓는 방법에도 영향을 준다는 것이다.(돌이 이웃하면 안된다는 조건 때문에.) 그래서 우리는 각 column에 돌을 놓을 수 있는 모든 방법(총 8가지)를 구해놓고, n+1번째 column에 i번째 방법으로 돌을 놓을 때, 얻을 수 있는 최대값을 dp[n][i]에 저장하는 방식을 사용하여, i번째 column에 특정 방법을 사용했을 때의 최대값만 구하는게 아니라, 모든 경우에 대해서 최대값을 구하고, larger problem을 해결하는 데 그 모든 경우를 고려하게 하므로써 문제를 해결할 수 있었다.

Placement\_table[8][2] (각 column을 0~3번째 칸으로 나눔)

0 : -1 -1 : 돌을 하나도 놓지 않음.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

1 : 0 -1 : 돌을 0번째 칸에만 놓음.

2 : 1 -1 : 돌을 1번째 칸에만 놓음.

3 : 2 -1 : 돌을 2번째 칸에만 놓음.

4 : 3 -1 : 돌을 3번째 칸에만 놓음.

5 : 0 2 : 돌을 0번째,2번째 칸에 놓음.

6 : 0 3 : 돌을 0번째,3번째 칸에 놓음.

7 : 1 3 : 돌은 1번째,3번째 칸에 놓음.

Neighbor\_graph[8][8] : 서로 인접한 column에 놓일 수 있는 placement 방법을 나타낸 인접행렬

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** |
| **1** | **0** | **1** | **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **0** | **1** | **1** | **1** | **1** | **0** |
| **1** | **1** | **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| **1** | **0** | **1** | **0** | **1** | **0** | **0** | **1** |
| **1** | **0** | **1** | **1** | **0** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **0** | **1** | **0** | **0** |

1이면 그 둘은 인접한 column에 놓일 수 있고, 0이면 놓일 수 없다.

[base case]

if(index == 0 )

dp[index][placement] = d.getPlacementSum(index, placement);

(첫번째 column일 때 initialize. 각 placement방법에 대해 값을 구해서 그 값을 넣어줌.)

[n>=1]

dp[n][i] = max{dp[n-1][j](단, Neighbor\_graph[i][j]=1)}+v[i]

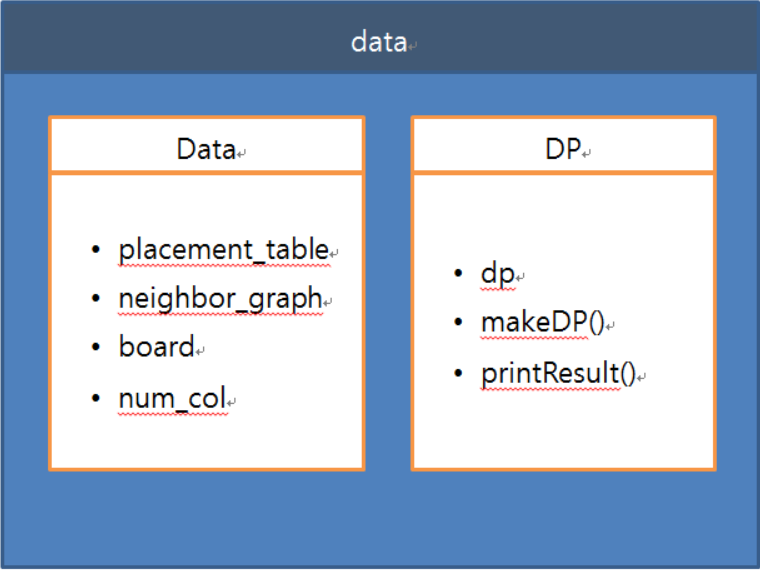
(v[i]는 placement[i]방법으로 n번째 column에 돌을 놓았을 때 얻을 수 있는 값.)

최종 답은 dp[column\_num-1][0~7] 값 중 최대값.

1. **구현**

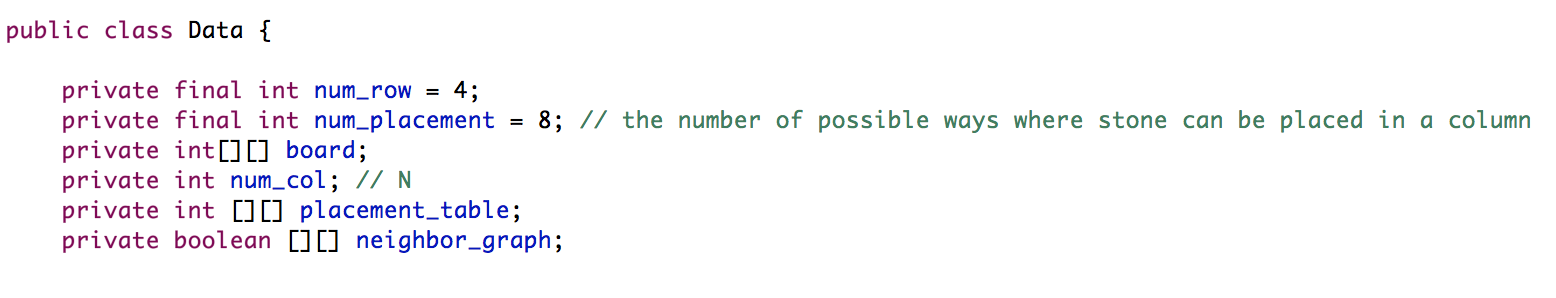
**3.1 Class 설명**

* 프로그램은 data, main 총 2개의 Package로 구성 되고 data pakage에 Data, DP의 두 class 가 있다.
* Data는 DP클래스에서 makeDP를 수행하기 위해 미리 필요한 값들을 저장하고 있다.
* DP 클래스는 dp테이블을 저장하는 배열과 이 배열을 채우는 makeDP, 그리고 결과를 출력하는 printResult를 가진다.

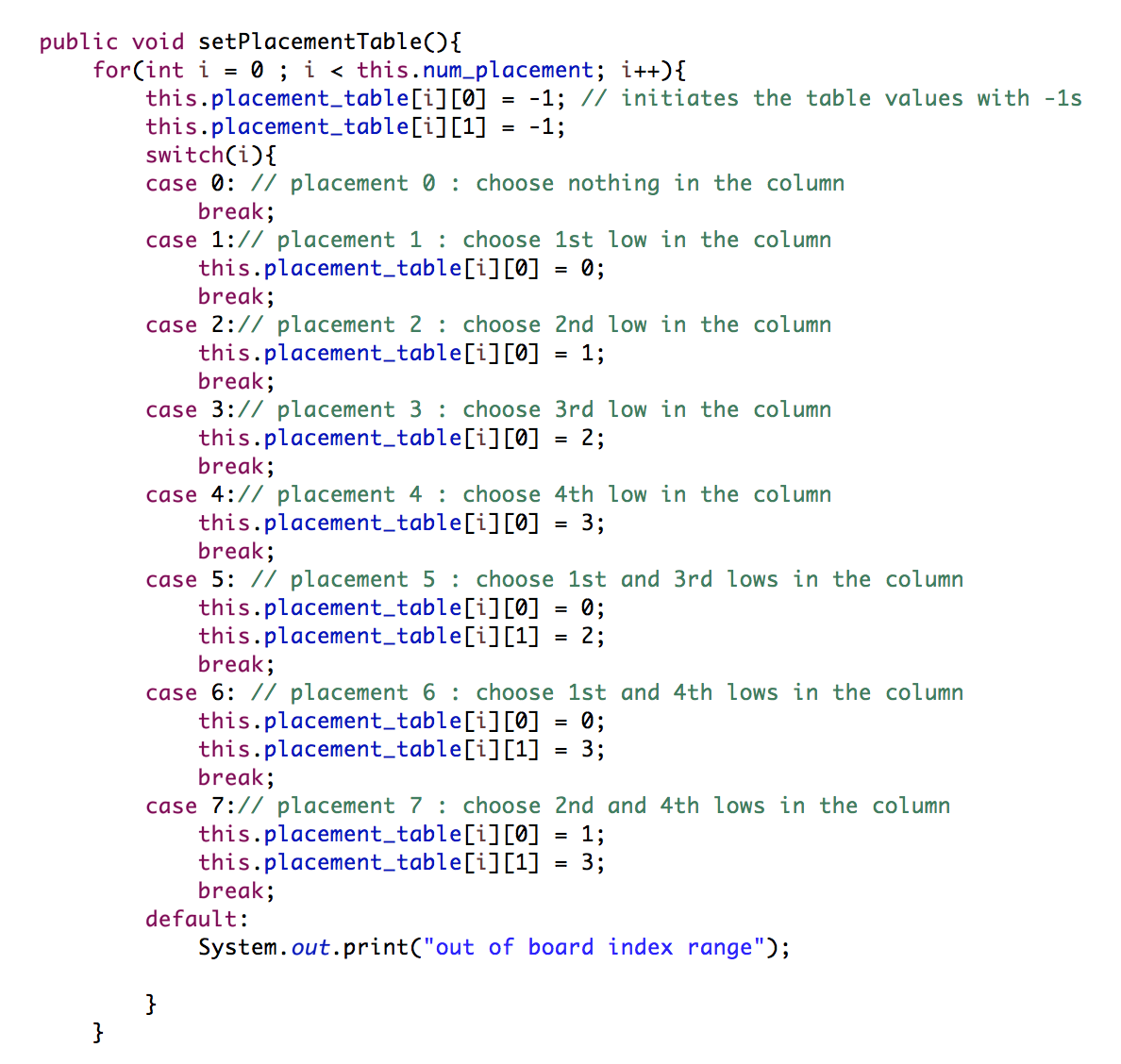


**3.2 알고리즘에 대한** 세**부 구현**

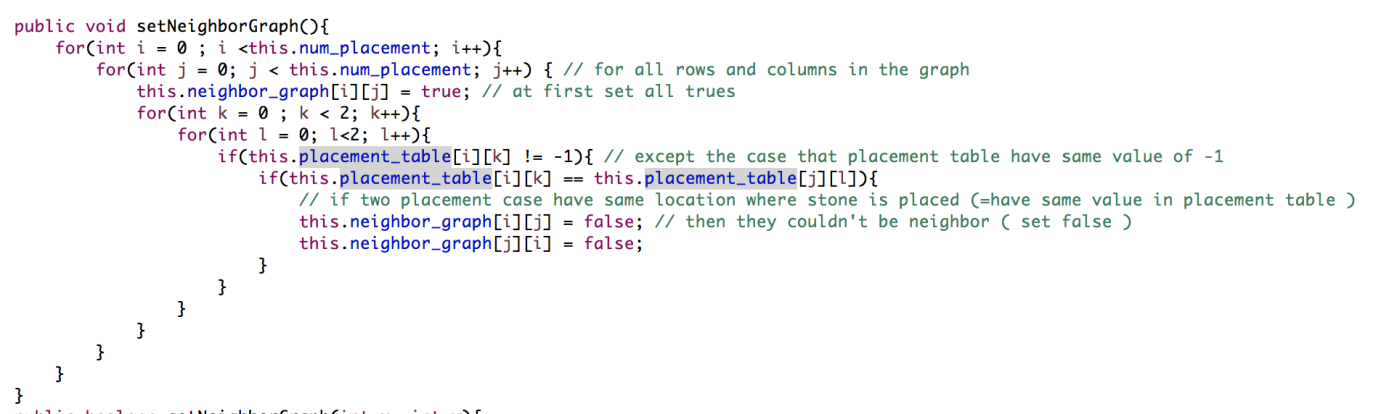
* 이번 프로그램을 구현하기 위해서 Dynamic programming을 사용하였다. 여기서 DP table은 num\_row를 기준으로 채우고 있다. Row\_num을 기준으로 최대 부분합을 만들기 위해 Placement\_table과 neighbor\_graph를 사용하였다.



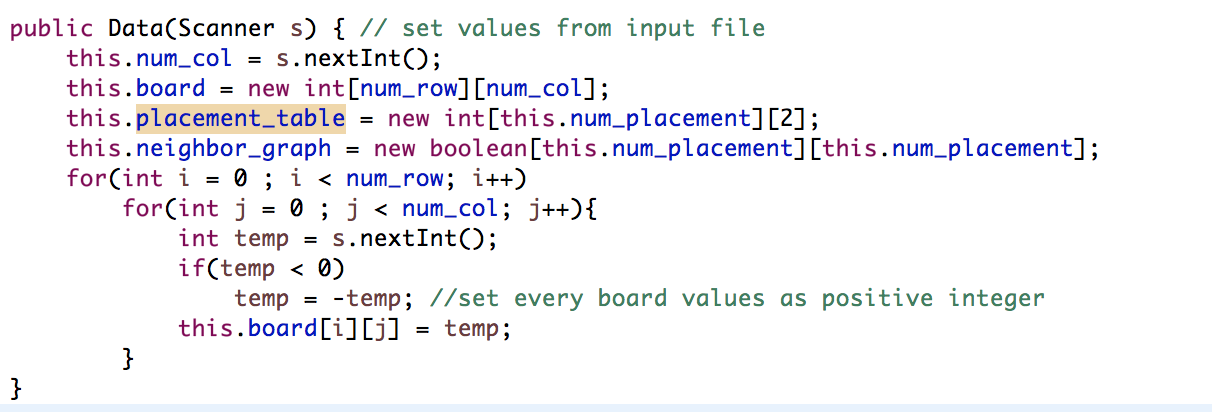
* num\_placement는 한 column에서 돌을 놓을 수 있는 경우의 수가 되고 한 column에 최대 2개의 돌을 놓을 수 있고 all integer (-1 <= x <= 3)을 가지고 -1의 경우 아무것도 놓지 않는 경우를 표현한다.



* 예를 들면 placement 7은 1과 3값을 가지고 이는 2번째와 4번째 자리에 돌을 배치한 경우가 된다.
* 이와 같이 placement를 설정하였을 때 setNeighborGraph를 사용해서 8x8 table을 만들고 서로 이웃할 수 있는 placement끼리 true또는 false값을 주어서 graph를 만들었다.



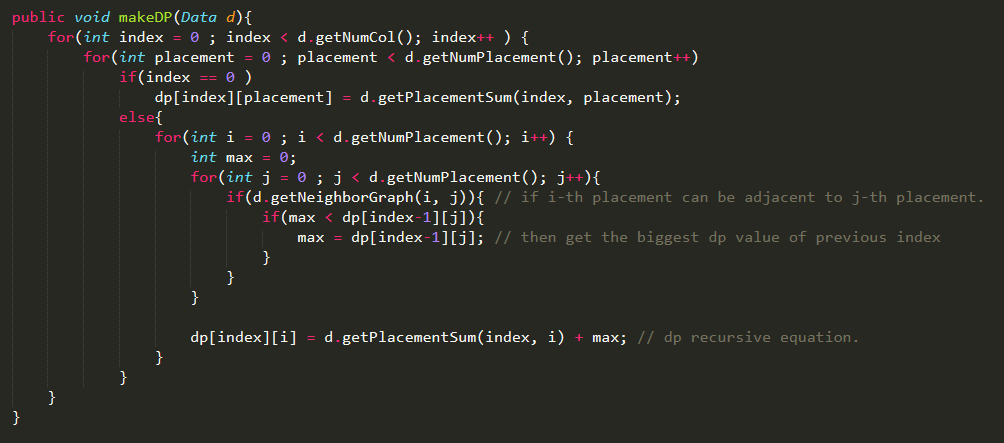
* 또한 문제에서 board에 음수와 양수가 있는 경우를 흑 돌 백돌 개념으로 나누고 있는데 과제 토의 과정에서 각각의 돌이 2N개 있기 때문에 한가지 돌을 놓다가 부족해질 경우가 생길 수 없다는 것과 음수의 값을 결과에서 양수의 값처럼 더하고 있는 것을 통해서 딱히 흑 돌과 백 돌의 구분 즉 음수와 양수를 구분하지 않아도 똑같은 결과를 얻을 수 있다고 판단하였다. 아래처럼 값을 저장 할 때에 음수의 값은 temp 변수에 저장하였다가 양수로 바꾸어서 저장하였다.



* 여기까지 과정을 이해한다면 makeDP의 동작 과정을 알 수 있다. Dp table을 num\_col이 n(1<=n<=N)일때 8가지 placement에 대해서 나올 수 있는 최대 값을 모두 저장한 뒤에 이것을 사용해서 n+1번째 column에서 각 placement로 만들어지는 합과 그 때 n+1이 가지는 placement와 인접할 수 있는 n번 째의 placement를 찾고 이를 통해 가능한 n번째의 placement 값과 n+1번째의 placement로 만들어지는 합을 구해서 최대값을 선택하는 것이 메인 재귀 함수가 된다.

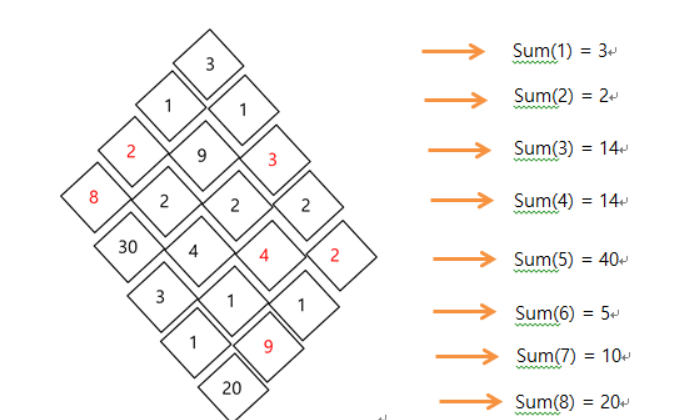


1. **토의**
   1. **Big-O notation 분석**

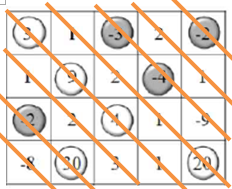


**-** 이번 과제에서 가장 핵심적인 함수는 DP를 만드는 makeDP 함수이다

* Time Complexity
  + n개의 column에 대해, 각 column 당 8개의 DP 값을 정해야 되고, 각 DP 를 만드는데 Neighbor graph(8 × 8) 전체에서 찾아야 되므로 512 × n, 즉 O(n)이 된다.
* Space Complexity
  + 필요한 자료 구조는 size가 8 × 8인 Neighbor graph와 DP값을 table의 크기는 8×n이므로 O(n)이 된다.
  1. **구현과정에서 겪은 어려운 점 (시행 착오)**
  2. **기존 방법**
     1. Board에 놓을 수 있는 최대 돌의 개수는 2×n 개이다.
* 규칙에 의해 한 column에는 2개가 놓을 수 있기 때문
  + 1. 검은 돌의 개수도 최대 2×n이기 때문에 board 위의 모든 음수는 양수로 생각해서 봐도 무관하다.
    2. Board를 45도 회전을 시키면 아래와 같은 모양이 된다.



* + 1. 규칙에 의해 각 row는 서로 이웃 할 수 없다는 사실을 알 수 있는데 이 때, 각 row의 합을 이용해 Dynamic Programming 식을 세울 수 있다.
    2. sum(i)는 i번째 row의 전체 합이라고 할 때 DP 식은 아래와 같다,
    3. 이 방식대로면, 현재 예제의 답 77, 411이 나온다,
  1. **기존 방법의 한계**
     1. 기존 방법에서는 row의 합으로 DP를 세운다. 즉, row에 있는 모든 item을 고른다는 것이 전제되는데, 모든 row의 item을 다 고른다는 것이 solution이라고 볼 수 없다.



* + 1. 반례로, [0,0]에서 대각선으로 row를 만드는 것이 아니라, [0,4]에서 대각선으로 row를 만들었을 때를 보면 가정이 틀렸음을 알 수 있다.
    2. 그래서 새로운 방법을 생각했고, column당 DP 값을 8개 갖게 하여 해결했다.