CSE2011 Problem Solving, 2016 Spring

2013312343 이상헌

**Homework 2-1**

**1. 문제 이해**

(1) 문제

- Make the N\*N matrix that is in the Homework 1 and satisfy the input value.

① Input : X, Y, n, D1, D2, …, Dn

[X, Y] is a range for each element; X and Y are positive integers.

N is the size of matrix (n by n).

D1, D2, …, Dn are diagonal entries.

② Output : n\*n matrix, each of whose column and row is sorted in an ascending order. Also, each element should be a unique integer value. If such a matrix is not feasible, print “infeasible”.

(2) 중요 정보

- 0 이상의 서로 다른 정수를 원소로 갖는 N\*N matrix.

- 같은 행이나 열에 위치하는 원소들은 오름차순으로 저장되어 있다.

- 최대, 최소 수(X, Y), matrix의 size(N), 대각선에 위치한 원소의 수(D1, D2, …, Dn)가 주어진다.

(3) 문제 정의

- 주어진 정보(Input)를 기반으로, 위와 같은 성질을 만족하는 N\*N matrix를 생성한다. 생성하지 못하는 경우에 대해서는 Infeasible을 출력한다.

**2. 문제 해결**

(1) 해결 방법 결정

*① 행렬의 성질 이용\_ ㄱ자형 정렬*

- 지난 Homework1을 discussion하는 시간에서, 새로운 행렬을 성질을 발견하였다. 기존 행렬의 성질은 같은 행 또는 열의 원소들이 오름차순을 유지한다는 것인데, 이를 응용하여 ㄱ자 형의 모든 원소들이 오름차순을 유지한다는 성질도 나타난다. 이 성질을 바탕으로 ㄱ자 형의 각 칸에 오름차순을 만족하도록 원소를 차례차례 입력한다면, test 행렬을 만들 수 있을 것이다.

- 아이디어

Figure 1은 n\*n matrix 내의 ㄱ자형 모형들이다. 색칠한 부분의 원소들은 왼쪽->오른쪽->아래(1->5->6) 순으로 오름차순을 이룬다. 이는 matrix의 기본 성질에 의한 것이다. 현재 문제에서는 대각선(2, 3, 4, 5)과 가장 낮은 수(1), 가장 높은 수(6)이 주어진다. 파란색 ㄱ자 모형을 보면, 1과 5에 해당하는 원소 값 사이의 수를 1과 5 사이의 빈칸(4\*4 matrix의 경우 2칸)에 넣어야 한다. 또한 5와 6에 해당하는 원소 값 사이의 수를 5와 6 사이의 빈칸(2칸)에 넣어야 한다. 값들은 랜덤 함수를 이용하여 원소 값 사이에 해당하는 임의의 값을 오름차순의 규칙으로 차례대로 넣는다. 나머지 색의 ㄱ자 모형 칸에도 이와 같이 넣는다.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  |  | 5 |
|  |  | 4 |  |
|  | 3 |  |  |
| 2 |  |  | 6 |

Figure . 4\*4 Matrix 내의 ㄱ자 모형

- 이점

Matrix의 기본적인 성질을 바탕으로 나타난 새로운 성질을 바탕으로 ㄱ자 모형을 따라 수를 삽입한다면, 가장 기본적으로 n\*n 2차원 배열을 1차원(선형)으로 생각할 수 있다는 것이다. ㄱ자 모형을 일렬로 펼치면 오름차순을 만족하는 1차원 배열로 생각할 수 있다. 이는 matrix의 기본적인 성질을 직관적으로 나타내어 프로그래머 혹은 사용자 입장에서 더욱 쉽게 이해하고, 고려 요소를 줄일 수 있다.

- 한계

본 방법으로 여러 번 시행한 결과, ㄱ자 모형에서의 오름차순만으로 matrix의 기본적인 성질을 만족하지 못한다는 결론에 도달하였다. ㄱ자 모형에서 오름차순을 만족하는 모든 matrix가 행, 열에 대해 오름차순을 만족하지는 않는다. 이는 matrix의 성질에 모순되기 때문에, 일반적인 방법으로 택할 수 없었다.

*② 대각선을 기반으로 입력*

- 예전에 알고리즘 문제를 풀 때 항상 다음과 같은 사실을 기반으로 풀었다. “이미 알고 있는 것에서 알고 싶은 것을 찾으라.” 항상 문제를 풀 때, 이미 알고 있는 사실은 문제를 해결하는 데에 있어서 가장 큰 힌트가 되었다. 본 문제에서도 이미 알고 있는 것을 다시 한 번 생각해보았다.

- Input 값은 최대, 최소의 수, matrix의 크기 그리고 대각선의 원소 값들이다. 최대, 최소의 수는 matrix 내의 원소 값들의 범위를 지정해주었다. Matrix의 크기는 matrix 내에 포함되는 원소의 개수를 지정해주었다. 더 생각해보아야 할 것은 대각선의 원소 값들이다. Figure2와 같이 대각선의 원소 값들을 기준으로 matrix를 크게 두 부분으로 나눌 수 있다. 빨간색 영역의 원소들은 최소의 수(1)보다 크거나, 대각선의 수들(2,3,4,5)보다 작다. 또한 파란색 영역의 원소들은 최대의 수(6)보다 작거나, 대각선의 수들(2,3,4,5)보다 크다. 이 점에서 본 방법을 착안하였다.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  |  | 5 |
|  |  | 4 |  |
|  | 3 |  |  |
| 2 |  |  | 6 |

Figure . 4\*4 Matrix에서 대각선을 기준으로 나눈 두 영역

- 아이디어

이미 알고있는 대각선의 수들과 가장 인접한 수들부터 시작한다. A에 해당하는 원소는 2와 3에 해당하는 원소보다 작다. 즉, 2와 3의 원소 값 중 더 작은 원소 값보다 작은 수가 A에 입력되어야 한다. 이와 마찬가지로, B에 입력되어야 하는 수는 3과 4에 해당하는 원소 값 중 더 작은 원소 값보다도 작아야 한다. A와 B에 이러한 성질을 만족하는 적절한 수를 입력한다.

Matrix의 size와 수의 범위가 비슷한 경우, 최대한 이웃하도록 수를 배치해야 한다. 최대한 많은 수를 입력하기 위해서는 대각선 근처에 있을수록 수가 커야하고, (0,0)에 가까이 있는 원소일수록 수가 작아야 한다. 따라서 A에는 min(2,3) (2와 3에 해당하는 원소 값들 중에 작은 값) 보다 작지만, 최대인 수를 입력한다. B도 마찬가지로 대각선보다 작지만, 최대인 수를 입력한다.

A -> B ->…->C의 대각선 내에 원소 값을 모두 입력한 경우, 한칸 위로 올라가서 D -> E의대각선내의 원소의 값을 입력한다. 이는 위와 같은 방법으로 적절한 수를 입력한다. 예를 들어 D의 경우에는 A와 B 중에 최소인 수보다 작은 수들중에 가장 큰 수를 입력한다. 이와 같이 대각선을 따라 한 칸씩 위로 이동하면서, 대각선을 따라 모든 원소의 값을 입력한다.

파란색 영역도 위와 비슷한 방법으로 입력한다. F는 2와 3중에 최대인 수보다 큰 수가 위치해야 한다. Matrix의 size와 수의 범위가 비슷한 경우, 수를 최대한 이웃하도록(빽빽하게) 배치해야 한다. 최대한 많은 수를 입력하기 위해서는 대각선 근처에 있을수록 작아야하고, (n-1,n-1)에 가까이 있는 원소일수록 커야한다. 따라서 F에는 max(2,3) (2와 3에 해당하는 원소 값들 중에 큰 값) 보다 크지만, 최소인 수를 입력한다. G, H도 마찬가지로 대각선보다 크지만 최소인 수를 입력한다.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | E | C | 5 |
| D | B | 4 | H |
| A | 3 | G | J |
| 2 | F | I | 6 |

파란색 영역에서도 대각선을 따라 입력하고, 한 칸 내려가서 또 다시 대각선을 따라 입력한다. 즉, 입력하는 순서는 F->G->H->I->J 가 된다.

Figure . 대각선을 따라 입력하는 방법.

여기서 주의할 점은 모든 원소 값들은 서로 다른 값이라는 점이다. 그러므로 원소의 값을 입력하기 전에, 이미 matrix에 입력되어있는 원소인지를 판별해야한다. 판별 결과 matrix에 입력되어있지 않은 원소들만 새로 입력하도록 한다.

두 번째 주의할 점은 Infeasible한 경우이다. Infeasible한 경우는 크게 두 가지로 나눌 수 있다. 먼저, matrix의 size가 수의 범위보다 큰 경우, matrix 내의 모든 원소를 채울 수 없다. 이를 수식으로 쓰면 다음과 같이 나타난다. Matrix의 size = n\*n. 가능한 수의 개수 = Y-X+1. 따라서 n^2 <= Y-X+1라는 조건을 만족해야 한다. 이러한 조건을 만족하지 않으면, infeasible하다. 두 번째로 수를 입력하는 중간에 같은 영역에서 더 이상 입력가능한 수가 존재하지 않는 경우이다. 이는 수의 경계에 도달하는 순간 infeasible하도록 하였다. 예를 들어 빨간 영역에서 원소의 값은 가능한 가장 큰 값에서부터 작은 순서대로, 입력되지 않은 값을 입력한다. 모두 입력이 되어 가능한 값이 (0,0)의 원소 값과 같게 된다면, infeasible하도록 하였다.

- 이점

본 방법은 이미 알고 있는 대각선의 수들을 이용하여 인접한 원소의 값을 입력하는 방법이다. 이와 같은 방법을 이용하면 단순한 비교의 반복만으로 문제의 성질을 만족하는 matrix를 만들 수 있다. 대각선의 두 수를 비교하는 것과, 그 수들보다 크거나 작은 수들 중에 최대, 최소인 수를 입력하는 것은 코딩할 때에 매우 직관적이고, 다른 요소를 생각하지 않아도 되기 때문에 안정적이다.

**3. 코드 작성**

- 본 아이디어 및 설명을 바탕으로 코드를 작성하였다.

int test\_cases(int X, int Y, int n, int \*D, int \*\*matrix) {

int \*num\_count, num, min, max, inf = 0, i, j = 0, k;

num\_count = (int\*)calloc(Y - X + 1, sizeof(int));

Num\_count 변수는 1차원 배열로, 수의 중복을 체크하는 변수이다. 배열의 크기는 matrix에가능한 수의 범위이고, 모든 원소를 0으로 초기화한다. matrix에 이미 입력된 수와 대응되는 num\_count는 0에서 1로 변환하여, 나중에 다른 원소를 입력할 때에 중복되지 않도록 한다.

Num 변수는 양끝과 대각선의 수들의 num\_count를 초기화 할 때 사용하는 상수변수이다. 큰 의미를 내포하지는 않는다.

Min, max는 각각 두 영역에서 원소의 값을 입력할 때 사용하는 변수로, 각 원소는 그 밑과 옆의 원소보다 작으면서도 가장 큰 값 혹은 크면서도 가장 작은 값을 가져야 하므로, 적절한 수를 찾는 코드에서 min, max라는 변수를 사용하였다.

Inf는 matrix의 생성이 infeasible인 경우를 판별하기 위해 사용한 변수이다. 0으로 초기화 되어있고, infeasible한 경우에 대해서 1로 변환한다.

I, j, k는 for, while문 등에서의 count variable이다.

for (i = 0; i < Y - X + 1; i++) {

num\_count[i] = 0;

}

for (i = 0; i < n; i++) {

matrix[n - j - 1][j] = D[i];

j++;

}

for (i = 0; i <= n - 1; i++) {

num = matrix[n - i - 1][i];

num\_count[num - X] = 1;

}

num\_count[0] = 1;

num\_count[Y - X] = 1;

matrix[0][0] = X;

matrix[n - 1][n - 1] = Y;

Num\_count의 모든 원소를 0으로 초기화하고, D(대각선 수들)의 원소들을 matrix내에 입력한다. 또한 양 끝의 원소를 matrix내에 입력하고, 대각선, 양 끝의 원소들의 num\_count를 1로 변환한다.

if (n\*n > Y - X + 1)

return 0;

for (i = 0; i < n; i++) {

if (D[i] <= X || D[i] >= Y)

return 0;

}

Infeasible한 경우에 대해, 0을 리턴 하고 함수를 종료한다. 초기에 infeasible한 경우는 matrix의 size보다 수의 범위가 작은 경우, 그리고 D의 수들 중에 수의 범위에 포함되지 않은 경우이다.

for (k = 0; k < n - 2; k++) {

for (i = 0; i <= n - k - 2; i++) {

if (matrix[n - i - k - 1][i] < matrix[n - i - k - 2][i + 1])

min = matrix[n - i - k - 1][i];

else

min = matrix[n - i - k - 2][i + 1];

while (1) {

if (min == X) {

inf = 1;

break;

}

min--;

if (num\_count[min - X] == 0) {

matrix[n - i - k - 2][i] = min;

num\_count[min - X] = 1;

break;

}

}

}

}

if (inf == 1) return 0;

Figure3에서 빨간색 영역의 원소들을 대각선을 따라 입력하는 코드이다. 앞서 설명한 것과 같이 대각선 상에 이웃하고 있는 두 원소의 위, 그리고 왼쪽에 위치하는 원소는 두 원소보다 작아야 하며, 가능한 값들 중에 최대값이어야 한다. Min은 대각선 상에 이웃하고 있는 두 원소의 값 중 작은 값을 저장한다. 이 후, X보다 크고 min보다 작은 수들 중에서 이미 matrix에 입력되지 않은, 최대의 수를 찾아 대각선 위, 그리고 왼쪽 원소에 입력한다. 만일 만족하는 수가 없어서 min이 결국 X에 도달하면, inf 변수를 1로 변환하고, inf가 1이 되었다면, 0을 리턴 하고 함수를 종료한다.

for (k = 0; k < n - 2; k++) {

for (i = 0; i <= n - k - 2; i++) {

if (matrix[n - i - 1][i + k] < matrix[n - i - 2][i + k + 1])

max = matrix[n - i - 2][i + k + 1];

else

max = matrix[n - i - 1][i + k];

while (1) {

if (max == Y) {

inf = 1;

break;

}

max++;

if (num\_count[max - X] == 0) {

matrix[n - i - 1][i + k + 1] = max;

num\_count[max - X] = 1;

break;

}

}

}

}

if (inf == 1) return 0;

Figure3에서 파란색 영역의 원소들을 대각선을 따라 입력하는 코드이다. 앞서 설명한 것과 같이 대각선 상에 이웃하고 있는 두 원소의 아래, 그리고 오른쪽에 위치하는 원소는 두 원소보다 커야 하며, 가능한 값들 중에 최소값이어야 한다. Max는 대각선 상에 이웃하고 있는 두 원소의 값 중 큰 값을 저장한다. 이 후, max보다 크고 Y보다 작은 수들 중에서 이미 matrix에 입력되지 않은, 최소의 수를 찾아 대각선 아래, 그리고 오른쪽 원소에 입력한다. 만일 만족하는 수가 없어서 max가 결국 Y에 도달하면, inf 변수를 1로 변환하고, inf가 1이 되었다면, 0을 리턴 하고 함수를 종료한다.

return 1;

}

성공적으로 함수를 끝마치고, inf가 0이라면 1을 리턴 하고 함수를 종료한다.

**4. 확인 및 분석**

- 본 아이디어를 이용하여 프로그램을 작성하였다. 이 프로그램이 정확한 결과값을 도출하는지를 알아보기 위해서 간단한 4\*4 matrix를 임의로 만들었다. 아래는 프로그램 작동 확인을 위해 임의적으로 만든 4\*4 matrix이다.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 7 | 8 |
| 3 | 4 | 9 | 13 |
| 5 | 6 | 12 | 15 |
| 10 | 11 | 14 | 16 |

Figure . 4\*4 matrix 예시

- 이 matrix를 이용하여 코드를 실행시켜 보았다. 여기서 Input값은 X = 1, Y = 16, n = 4, Ds = 10, 6, 9, 8이다.

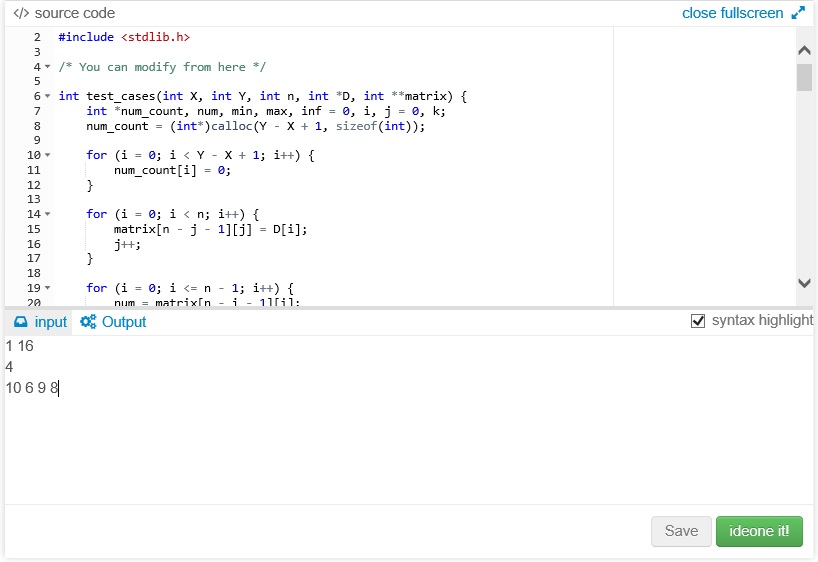


Figure . 예시 Input

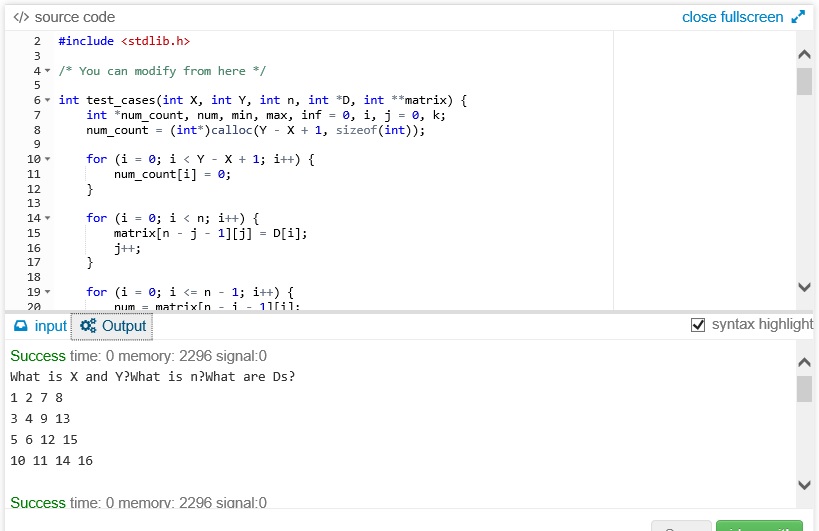


Figure . 예시 Output

- Figure 6의 아래를 보면, Output 값이 Figure 4의 예시 행렬과 일치한다는 것을 알 수 있다. 이는 코드가 정확한 결과를 도출한다는 것을 의미한다.

**6. 고찰**

- 본 문제는 Homework1의 행렬의 성질을 만족하는, testcase를 생성하는 프로그램을 구현하는 것이다. 문제를 해결한 아이디어는 이미 주어진 Ds(대각선)의 원소들과 본 행렬의 기본적인 성질을 이용한 것이다. 본 아이디어로 문제에 대한 정확한 정답을 도출하는 데에 성공하였지만, 주어진 Input을 만족하는 Output 행렬은 단지 하나가 될 수 없다. 본 프로그램은 주어진 Input을 만족하는, 가능한 모든 행렬을 출력하는 프로그램으로 응용할 수 있다. 또한, 어떠한 문제를 해결하는 과정에서, 정확한 결과인지를 판별하기 위해 만드는 testcase를 생성하는 것도 문제가 될 수 있다는 점이 흥미로웠다.

**7. 참고문헌 및 사용**

- [www.ide.com](http://www.ide.com) : C언어 구현 및 실행.

- 이진규 교수님, [문제해결기법] CSE2011\_2016spring\_Lecture\_Note03, 2016