

Homework_1B

2018312758

GUHO KIM

Problem explanation

- Finding k Test Program
- Input: n , D_1 , D_2 , ..., D_n , A_1 , A_2 , ..., A_n
- n is the size of matrix (n by n). ($2 \leq n \leq 10$)
- D_1 , D_2 , ..., D_n are diagonal entries.
- A_1 , A_2 , ..., A_n are antidiagonal entries.
- Output: n by n matrix, each of whose column and row is sorted in an ascending order. Also, each element should be a unique integer value (meaning that all elements should be different numbers). If such a matrix is not feasible, print "infeasible".
- Each entry (number) is an integer from 1 to 1000.

Idea for solution

- First, put the input-values to the matrix[n][n].
- And to check the uniqueness later, we create a structure index[1000]

```
#define MAX_N 11
#define MAX_VAL 1000
typedef struct Index {
    int row;
    int col;
    int in; //mark
} Index;
```

```
int matrix[MAX_N][MAX_N] = { 0 };
Index index[MAX_VAL];
```

Example(1)

Idea for solution

Input : 5 2 8 19 25 50 31 17 19 20 7

Example(1)

2				7
	8		20	
		19		
	17		25	
31				50

- Put the input-values to the matrix[n][n].

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Idea for solution

Input : 5 2 8 19 25 50 31 17 19 20 7

Example(1)

2	3	4	5	7
	8		20	
		19		
	17		25	
31				50

- Put the input-values to the matrix[n][n].
- I'll start putting in values from the matrix[0][1] to the right side.
- The initial value is matrix[0][0]+1.
- If the number is not marked, then put it.
- When I put some number, I'll add 1 to the number(number+=1), and mark the index and value to structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Idea for solution

Input : 5 2 8 19 25 50 31 17 19 20 7

Example(1)

2	3	4	5	7
6	8		20	
		19		
	17		25	
31				50

- Put the input-values to the matrix[n][n].
- I'll start putting in values from the matrix[0][1] to the right side.
- The initial value is matrix[0][0]+1.
- If the number is not marked, then put it.
- When I put some number, I'll add 1 to the number(number+=1), and mark the index and value to structure.
- When the row is changed, I'll start with matrix[row-1][col]+1.
- If it's already marked, it'll increase until it can be put in.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Idea for solution

Input : 5 2 8 19 25 50 31 17 19 20 7

Example(1)

2	3	4	5	7
6	8	9	20	21
10	11	19	22	23
12	17	24	25	26
31	32	33	34	50

- Put the input-values to the matrix[n][n].
- I'll start putting in values from the matrix[0][1] to the right side.
- The initial value is matrix[0][0]+1.
- If the number is not marked, then put it.
- When I put some number, I'll add 1 to the number(number+=1), and mark the index and value to structure.
- When the row is changed, I'll start with matrix[row-1][col]+1.
- If it's already marked, it'll increase until it can be put in.
- Fill up the matrix and it's over.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	1

Example(2)

Idea for solution

Input : 5 2 6 17 22 50 31 18 17 20 8

Example(2)

2				8
5	6		20	
		17		
	18		22	
31				50

- If you fill in this example in the same way, there is a problem.
- I can put 3, 4, or 5 in matrix[0][1], but all of numbers are already marked.
- In this case, I will fix the last value at this space.
- I will reset all of matrix without these blue and yellow matrix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Idea for solution

Input : 5 2 6 17 22 50 31 18 17 20 8

Example(2)

2	3	4	7	8
5	6	9	20	21
10	11	17	21	
	18		22	
31				50

- If you fill in this example in the same way, there is a problem.
- I can put 3, 4, or 5 in matrix[0][1], but all of numbers are already marked.
- In this case, I will fix the last value at this space.
- I will reset all of matrix without these blue and yellow matrix.
- And start to fill the matrix.
- There is same problem at the matrix[2][3], so I will fix that number and reset the matrix and restart.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Idea for solution

Input : 5 2 6 17 22 50 31 18 17 20 8

Example(2)

2				8
5	6		20	
		17	21	
	18		22	
31				50

- If you fill in this example in the same way, there is a problem.
- I can put 3, 4, or 5 in matrix[0][1], but all of numbers are already marked.
- In this case, I will fix the last value at this space.
- I will reset all of matrix without these blue and yellow matrix.
- And start to fill the matrix.
- There is same problem at the matrix[2][3], so I will fix that number and reset the matrix and restart.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

Idea for solution

Input : 5 2 6 17 22 50 31 18 17 20 8

Example(2)

2	3	4	7	8
5	6	9	20	23
10	11	17	21	24
12	18	19	22	25
31	32	33	34	50

- If you fill in this example in the same way, there is a problem.
- I can put 3, 4, or 5 in matrix[0][1], but all of numbers are already marked.
- In this case, I will fix the last value at this space.
- I will reset all of matrix without these blue and yellow matrix.
- And start to fill the matrix.
- There is same problem at the matrix[2][3], so I will fix that number and reset the matrix and restart.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1

Code

```
Index fillTheMatrix(int matrix[][MAX_N], Index index[], int curr, int row, int col, int n, int max_v) {
    Index index2;
    for (; row < n; row++) {
        for (; col < n; col++) {
            if (row == 0) curr = matrix[row][col-1] + 1;
            else curr = matrix[row - 1][col] + 1;
            for (; curr < max_v; curr++) {
                //넣으려는 값이 대각선이면 스킵
                if (col == row || col == n - 1 - row) break;
                //시작하기 전에 우측이나 하측에 노란박스 있으면 비교부터.
                //오른쪽 값보다 클 때
                else if ((col + 1 == row || col + 1 == n - row - 1) && (curr > matrix[row][col + 1])) {
                    curr = matrix[row][col + 1] - 1;
                    break;
                }
                //아래쪽 값보다 클 때
                else if ((row + 1 == col || row + 1 == n - col - 1) && (curr > matrix[row + 1][col])) {
                    curr = matrix[row + 1][col] - 1;
                    break;
                }
            }
        }
    }
}
```

This is the code for how I filled the matrix

```

        //왼쪽과 위쪽 값 비교
        if (row == 0){
            if (matrix[row][col - 1] < curr && (index[curr].in != 1)) {
                matrix[row][col] = curr;
                index[matrix[row][col]].row = row;
                index[matrix[row][col]].col = col;
                index[matrix[row][col]].in = 1;
                break;
            }
        }

        else{
            if (col == 0) {
                if (matrix[row - 1][col] < curr && (index[curr].in != 1)) {
                    matrix[row][col] = curr;
                    index[matrix[row][col]].row = row;
                    index[matrix[row][col]].col = col;
                    index[matrix[row][col]].in = 1;
                    break;
                }
            }
            else if (col > 0) {
                if (matrix[row][col - 1] < curr && matrix[row - 1][col] < curr && (index[curr].in != 1)) {
                    matrix[row][col] = curr;
                    index[matrix[row][col]].row = row;
                    index[matrix[row][col]].col = col;
                    index[matrix[row][col]].in = 1;
                    break;
                }
            }
        }

        //날을 게 없으면 return;
        if (matrix[row][col] == 0) {
            index2.row = row;
            index2.col = col;
            index2.in = curr;
            return index2;
        }

    }

    //for of col 끝나면 col =0으로 초기화.
    col = 0;
}

index2.row = row;
index2.col = col;
index2.in = curr;
return index2;
}

```

```

for (int i = 0; i < n*(n-2); i++) {
    //만약 끝까지 채우지 못하고 index를 return 했다면, mark하고 reset한 뒤에 다시 시작
    if (!(index2.row == n && index2.col == 0)) {
        resetMatrix(matrix,n,index);
        fillTheIndex(index_store, index, index2, matrix);
        index2 = fillTheMatrix(matrix, index, 0, 0, 1, n, max_v);
    }
    //matrix 출력
    else{
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                printf("%d ", matrix[i][j]);
            }
            printf("\n");
        }
        return 0;
    }
}
printf("Infeasible");
return 0;

```

If I can't fill the matrix before $n(n-2)$ times ,it is Infeasible.