Homework 1B Report

2021315385
이건 / Gun Daniel Lee

Problem explanation

- Finding *k* Test Program

- Given the diagonal and antidiagonal, create a program that outputs an *n* by *n* matrix with the following attributes:
  - Each column and row is sorted in ascending order
  - Each element should be a unique integer value
  - If such a matrix is not feasible, print “Infeasible”

Solution explanation

- Identify conditions that make matrices infeasible

  1. Difference between two adjacent diagonal entries should not be less than 3 (D2 – D1 >= 3) (D2 – D1 < 3)

3	?
?	8

3	?
?	5

Solution explanation

- Identify conditions that make matrices infeasible

1. Difference between two adjacent diagonal entries should not be less than 3 (D2 – D1 >= 3) (D2 – D1 < 3)

3	5
7	8

3	4
X	5

Solution explanation

- Identify conditions that make matrices infeasible

  2. Difference between a diagonal and antidiagonal entry in a single row or column should be greater than the gap between them

1	?	?	10
?	-	-	?
?	-	-	?
11	?	?	17

1	?	?	5
?	-	-	?
?	-	-	?
3	?	?	7

Solution explanation

- Identify conditions that make matrices infeasible

  2. Difference between a diagonal and antidiagonal entry in a single row or column should be greater than the gap between them

1	5	8	10
3	-	-	12
6	-	-	16
11	13	15	17

1	4	X	5
2	-	-	6
X	-	-	X
3	X	X	7

Solution explanation

- existing_values: array to with the existing values in the matrix (serves as a basis to use unique values)

- value_exists: function to check whether a value already exists in the matrix

```
int value_exists(int existing_values[], int value, int size)
{
  for(int i = 0; i < size; i++)
  {
    if(existing_values[i] == value)
    {
      return 1;
    }
  }

  return 0;
}
```

Solution explanation

- Elements will be filled using a nested for loop (left to right, top to bottom)
  - Fill up each row from left to right, so on

Solution explanation

- Value choosing divided into 4 different quadrants:
  1. Located to the right of diagonal entries and left of antidiagonal entries
  2. Located to the left of both diagonal and antidiagonal entries
  3. Located to the right of both diagonal and antidiagonal entries
  4. Located to the left of diagonal entries and right of antidiagonal entries

D	#1	#1	A
#2	D	A	#3
#2	A	D	#3
A	#4	#4	D

Solution explanation

- Case 1:

1. Copy the element to the left
2. Increase value by 1 until it satisfies the following:
    1. Unique value
    2. Less than the value below and to the right
3. Otherwise, infeasible

D	#1	#1	A
#2	D	A	#3
#2	A	D	#3
A	#4	#4	D

Solution explanation

- Case 2:

1. Copy the element above
2. Increase value by 1 until it satisfies the following:
    1. Unique value
    2. Less than the value below and to the right
3. Otherwise, infeasible

D	#1	#1	A
#2	D	A	#3
#2	A	D	#3
A	#4	#4	D

Solution explanation

- Case 3:

1. Copy the element above
2. Increase value by 1 until it satisfies the following:
    1. Unique value
    2. Greater than value to the left
    3. Less than value below
3. Otherwise, infeasible

D	#1	#1	A
#2	D	A	#3
#2	A	D	#3
A	#4	#4	D

Solution explanation

- Case 4:

1. Copy the element to the left
2. Increase value by 1 until it satisfies the following:
    1. Unique value
    2. Greater than value above
    3. Less than value to the right
3. Otherwise, infeasible

D	#1	#1	A
#2	D	A	#3
#2	A	D	#3
A	#4	#4	D

Solution explanation

- Example:
- Element (0, 1):

5	#1	#1	23
#2	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (0, 1):
  - Case 1
  - Insert 5 and increase by 1

5	5	#1	23
#2	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (0, 1):
  - 6 satisfies all conditions
  - Move to next

5	6	#1	23
#2	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (0, 2):
  - Case 1
  - Insert 6 and increase by 1

5	6	6	23
#2	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (0, 2):
  - 7 satisfies all conditions
  - Move to next

5	6	7	23
#2	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (1, 0):
  - Case 2
  - Insert 5 and increase by 1

5	6	7	23
5	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (1, 0):
  - 6 and 7 are skipped
  - 8 satisfies conditions
  - Move to next

5	6	7	23
8	11	22	#3
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (1, 3):
  - Case 3
  - Insert 23 and increase by 1

5	6	7	23
8	11	22	23
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (1, 3):
  - 24 satisfies all conditions
  - Move on to next

5	6	7	23
8	11	22	24
#2	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (2, 0):
  - Case 2:
  - Insert 8 and increase by 1

5	6	7	23
8	11	22	24
8	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (2, 0):
  - 9 satisfies conditions
  - Move on to next

5	6	7	23
8	11	22	24
9	21	30	#3
20	#4	#4	46

Solution explanation

- Example:

- Element (2, 3):
  - Case 3
  - Insert 24 and increase
    by 1

5	6	7	23
8	11	22	24
9	21	30	24
20	#4	#4	46

Solution explanation

- Example:

- Element (2, 3):
  - 25~30 are less than or equal to 30 (value to the left)
  - Next value, 31, satisfies conditions
  - Move on to next

5	6	7	23
8	11	22	24
9	21	30	31
20	#4	#4	46

Solution explanation

- Example:

- Element (3, 1):
  - Case 4
  - Insert 20 and increase by 1

5	6	7	23
8	11	22	24
9	21	30	31
20	20	#4	46

Solution explanation

- Example:

5	6	7	23
8	11	22	24
9	21	30	31
20	25	#4	46

- Element (3, 1):
  - 21~24 are already in the matrix
  - 25 satisfies conditions
  - Move on to next

Solution explanation

- Example:

- Element (3, 1):
  - Case 4
  - Insert 25 and increase by 1

5	6	7	23
8	11	22	24
9	21	30	31
20	25	25	46

Solution explanation

- Example:

- Element (3, 1):
  - 26~30 are less than or equal to element above
  - 31 already exists
  - 32 satisfies conditions
  - End program

5	6	7	23
8	11	22	24
9	21	30	31
20	25	32	46

Solution explanation

5	6	7	23
8	11	22	24
9	21	30	31
20	25	32	46

```
> 4 5 11 30 46 20 21 22 23
5 6 7 23
8 11 22 24
9 21 30 31
20 25 32 46
```

Solution explanation

- Case 1 code

```c
//case 1: to the right of the diagonal entries && left of antidiagonal entries
if(j > i && i + j < n - 1)
{
  matrix[i][j] = matrix[i][i];

  while(value_exists(existing_values, matrix[i][j], n * n) == 1)
  {
    matrix[i][j]++;
    int increment = 1;
    int comparing_element = 0;
    while(comparing_element == 0)
    {
      comparing_element = matrix[i + increment][j];
      increment++;
    }

    if(matrix[i][j] >= comparing_element)
    {
      printf("Infeasible");
      return 0;
    }
  }
  existing_values[unique_number_count] = matrix[i][j];
  unique_number_count++;
}
```

Solution explanation

- Case 2 code

```
//case 2: to the left of both diagonal and antidiagonal entries
else if(j < i && i + j < n - 1)
{
  matrix[i][j] = matrix[i - 1][j];

  while(value_exists(existing_values, matrix[i][j], n * n) == 1)
  {
    matrix[i][j]++;
    int increment = 1;
    int comparing_element = 0;
    while(comparing_element == 0)
    {
      comparing_element = matrix[i][j + increment];
      increment++;
    }

    if(matrix[i][j] >= comparing_element)
    {
      printf("Infeasible");
      return 0;
    }
  }
  existing_values[unique_number_count] = matrix[i][j];
  unique_number_count++;
}
```

Solution explanation

- Case 3 code

```
//case 3: to the right of both diagonal and antidiagonal entries
else if(j > i && i + j > n - 1)
{
  matrix[i][j] = matrix[i - 1][j];

  while(value_exists(existing_values, matrix[i][j], n * n) == 1)
  {
    matrix[i][j]++;
    int increment = 1;
    int comparing_element = 0;
    while(comparing_element == 0)
    {
      comparing_element = matrix[i][j - increment];
      increment++;
    }
    while(matrix[i][j] <= comparing_element)
    {
      matrix[i][j]++;
    }

    //compare with diagonal entry in the same column
    if(matrix[i][j] >= matrix[j][j])
    {
      printf("Infeasible");
      return 0;
    }
  }
  existing_values[unique_number_count] = matrix[i][j];
  unique_number_count++;
}
```

Solution explanation

- Case 4 code

```
//case 4: left of antidiagonal entries and right of diagonal entries
else if(j < i && i + j > n - 1)
{
    matrix[i][j] = matrix[i][j - 1];

    while(value_exists(existing_values, matrix[i][j], n * n) == 1)
    {
        matrix[i][j]++;
        int increment = 1;
        int comparing_element = 0;
        while(comparing_element == 0)
        {
            comparing_element = matrix[i - increment][j];
            increment++;
        }

        while(matrix[i][j] <= comparing_element)
        {
            matrix[i][j]++;
        }

        //compare with diagonal entry in the same column
        if(matrix[i][j] >= matrix[i][i])
        {
            printf("Infeasible");
            return 0;
        }
    }
    existing_values[unique_number_count] = matrix[i][j];
    unique_number_count++;
}
```

Solution analysis

- Pros:
  - Efficient due to eliminating several infeasible cases in the beginning
  - Reduces trouble for choosing a value

- Cons
  - Inefficient when needed values for each element space becomes more specified
  - Lengthy code due to minor differences between cases

Solution analysis

- Favorable inputs:
  - The diagonals and antidiagonals have more flexibility (greater difference)

Thank you!