**자료구조 과제2**

(삼각행렬 생성기 ADT 정의 및 구현)

logo.gif

|  |  |
| --- | --- |
| 과목명 | 자료구조 |
| 담당교수 | 김계영 |
| 학과 | 소프트웨어학부 |
| 학년 | 3 |
| 학번 | 20152994 |
| 성명 | 이진영 |
| 제출일 | 2017.5.1 |



**-목차-**

**1. ADT 정의**

**2. ADT 구현**

1) 원시 코드

2) 수행 결과

**3. 성능 평가**

1) 함수별 실행 step수 산출

2) 시간 복잡도 쎄타(θ) 산출

**ADT 정의/구현 및 시간복잡도 분석 2**

20152994 이진영

**1. ADT 정의**

ADT Triangular matrix maker

objects: 임의의 n by n 2차원 정방행렬

functions:

n,m ∈ 자연수, n\_by\_n\_matrix ∈ n by n 2차원 정방행렬,

upper/left upper/lower/right lower triangular matrix ∈ m by m 삼각행렬

(행/열 오름차순은 일반적인 index 증가 순서 (ex) [0][0], [0][1], [0][2], [1][0], [1][1], …))

up\_tri\_mat(n\_by\_n\_matrix) ::= n\_by\_n\_matrix의 0이 아닌 원소를 행/열 오름차순으로 저장하고,

저장된 원소들을 upper triangular matrix의 0이 아닌 원소의 자리에

행/열 오름차순의 순서대로 대입 후,

해당 upper triangular matrix 를 화면에 출력

low\_tri\_mat(n\_by\_n\_matrix) ::= n\_by\_n\_matrix의 0이 아닌 원소를 행/열 오름차순으로 저장하고,

저장된 원소들을 lower triangular matrix의 0이 아닌 원소의 자리에

행/열 오름차순의 순서대로 대입 후,

해당 lower triangular matrix 를 화면에 출력

l\_up\_tri\_mat(n\_by\_n\_matrix) ::= n\_by\_n\_matrix의 0이 아닌 원소를 행/열 오름차순으로 저장하고,

저장된 원소들을 left upper triangular matrix의 0이 아닌 원소의 자리에

행/열 오름차순의 순서대로 대입 후,

해당 left upper triangular matrix 를 화면에 출력

r\_low\_tri\_mat(n\_by\_n\_matrix) ::= n\_by\_n\_matrix의 0이 아닌 원소를 행/열 오름차순으로 저장하고,

저장된 원소들을 right lower triangular matrix의 0이 아닌 원소의 자리에

행/열 오름차순의 순서대로 대입 후,

해당 right lower triangular matrix 를 화면에 출력

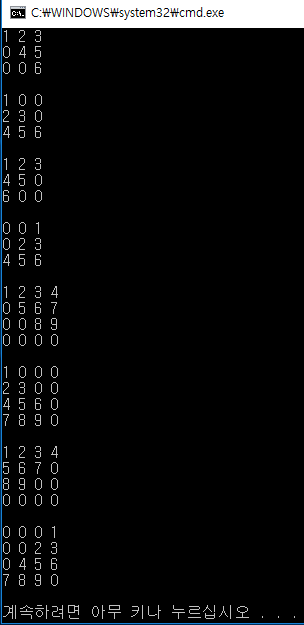
end

**2. ADT 구현**

1) 원시 코드

|  |
| --- |
| #include<stdio.h>  #include<malloc.h>  #define MAX\_TRI\_MATRIX\_SIZE 99  void up\_tri\_mat(int\* matrix, int size\_byte) { // make and show upper triangular matrix  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  // 위 코드는 matrix의 0이 아닌 원소를 temp 배열에 저장, temp\_num은 temp배열의 원소 개수  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }  // 새로 생성할 삼각행렬의 크기를 지정, new\_size는 새로 생성할 n by n 삼각행렬의 최소 n 크기, usable은 삼각행렬의 0이 아닌 원소 개수  int\*\* upper\_tri\_matrix;  upper\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(upper\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  upper\_tri\_matrix[n][m] = 0;  }  }  // upper\_tri\_matrix 변수에, n = new\_size인 n by n 삼각행렬을 생성하고 모든 항을 0으로 초기화  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = n; m < new\_size; m++) {  if (i >= temp\_num) {  upper\_tri\_matrix[n][m] = 0;  }  else {  upper\_tri\_matrix[n][m] = temp[i++];  }  }  }  // upper triangular matrix의 0이 아닌 항들의 자리에, 왼쪽에서 오른쪽으로 위에서 아래로의 순서로 temp변수 값 대입  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", upper\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");  // 만들어진 upper triangular matrix를 출력  free(temp);  free(\*upper\_tri\_matrix);  free(upper\_tri\_matrix);  }  void low\_tri\_mat(int\* matrix, int size\_byte) { // make and show lower triangular matrix  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  // 위 코드는 matrix의 0이 아닌 원소를 temp 배열에 저장, temp\_num은 temp배열의 원소 개수  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }  // 새로 생성할 삼각행렬의 크기를 지정, new\_size는 새로 생성할 n by n 삼각행렬의 최소 n 크기, usable은 삼각행렬의 0이 아닌 원소 개수  int\*\* lower\_tri\_matrix;  lower\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(lower\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  lower\_tri\_matrix[n][m] = 0;  }  }  // lower\_tri\_matrix 변수에, n = new\_size인 n by n 삼각행렬을 생성하고 모든 항을 0으로 초기화  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = 0; m <= n; m++) {  if (i >= temp\_num) {  lower\_tri\_matrix[n][m] = 0;  }  else {  lower\_tri\_matrix[n][m] = temp[i++];  }  }  }  // lower triangular matrix의 0이 아닌 항들의 자리에, 왼쪽에서 오른쪽으로 위에서 아래로의 순서로 temp변수 값 대입  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", lower\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");  // 만들어진 lower triangular matrix를 출력  free(temp);  free(\*lower\_tri\_matrix);  free(lower\_tri\_matrix);  }  void l\_up\_tri\_mat(int\* matrix, int size\_byte) { // make and show left upper triangular matrix  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  // 위 코드는 matrix의 0이 아닌 원소를 temp 배열에 저장, temp\_num은 temp배열의 원소 개수  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }  // 새로 생성할 삼각행렬의 크기를 지정, new\_size는 새로 생성할 n by n 삼각행렬의 최소 n 크기, usable은 삼각행렬의 0이 아닌 원소 개수  int\*\* l\_upper\_tri\_matrix;  l\_upper\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(l\_upper\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  l\_upper\_tri\_matrix[n][m] = 0;  }  }  // l\_upper\_tri\_matrix 변수에, n = new\_size인 n by n 삼각행렬을 생성하고 모든 항을 0으로 초기화  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size - n; m++) {  if (i >= temp\_num) {  l\_upper\_tri\_matrix[n][m] = 0;  }  else {  l\_upper\_tri\_matrix[n][m] = temp[i++];  }  }  }  // left upper triangular matrix의 0이 아닌 항들의 자리에, 왼쪽에서 오른쪽으로 위에서 아래로의 순서로 temp변수 값 대입  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", l\_upper\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");  // 만들어진 left upper triangular matrix를 출력  free(temp);  free(\*l\_upper\_tri\_matrix);  free(l\_upper\_tri\_matrix);  }  void r\_low\_tri\_mat(int\* matrix, int size\_byte) { // make and show right lower triangular matrix  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  // 위 코드는 matrix의 0이 아닌 원소를 temp 배열에 저장, temp\_num은 temp배열의 원소 개수  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }  // 새로 생성할 삼각행렬의 크기를 지정, new\_size는 새로 생성할 n by n 삼각행렬의 최소 n 크기, usable은 삼각행렬의 0이 아닌 원소 개수  int\*\* r\_lower\_tri\_matrix;  r\_lower\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(r\_lower\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  r\_lower\_tri\_matrix[n][m] = 0;  }  }  // r\_lower\_tri\_matrix 변수에, n = new\_size인 n by n 삼각행렬을 생성하고 모든 항을 0으로 초기화  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = new\_size-1-n; m < new\_size; m++) {  if (i >= temp\_num) {  r\_lower\_tri\_matrix[n][m] = 0;  }  else {  r\_lower\_tri\_matrix[n][m] = temp[i++];  }  }  }  // right lower triangular matrix의 0이 아닌 항들의 자리에, 왼쪽에서 오른쪽으로 위에서 아래로의 순서로 temp변수 값 대입  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", r\_lower\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");  // 만들어진 right lower triangular matrix를 출력  free(temp);  free(\*r\_lower\_tri\_matrix);  free(r\_lower\_tri\_matrix);  }  void main() {  int matrix3x3[3][3] = { { 1,2,0 },  { 3,4,0 },  { 5,6,0} };  int matrix4x4[4][4] = { {1,2,3,4},  {5,6,0,0},  {7,0,0,0},  {8,9,0,0} };  up\_tri\_mat((int\*)matrix3x3, sizeof(matrix3x3));  low\_tri\_mat((int\*)matrix3x3, sizeof(matrix3x3));  l\_up\_tri\_mat((int\*)matrix3x3, sizeof(matrix3x3));  r\_low\_tri\_mat((int\*)matrix3x3, sizeof(matrix3x3));    up\_tri\_mat((int\*)matrix4x4, sizeof(matrix4x4));  low\_tri\_mat((int\*)matrix4x4, sizeof(matrix4x4));  l\_up\_tri\_mat((int\*)matrix4x4, sizeof(matrix4x4));  r\_low\_tri\_mat((int\*)matrix4x4, sizeof(matrix4x4));  } |

2) 수행결과



**3. 성능 평가**

1) 함수별 실행 step 수 산출

① up\_tri\_mat()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void up\_tri\_mat(int\* matrix, int size\_byte) {  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }  int\*\* upper\_tri\_matrix;  upper\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(upper\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  upper\_tri\_matrix[n][m] = 0;  }  }  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = n; m < new\_size; m++) {  if (i >= temp\_num) {  upper\_tri\_matrix[n][m] = 0;  }  else {  upper\_tri\_matrix[n][m]=temp[i++];  }  }  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", upper\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");  free(temp);  free(\*upper\_tri\_matrix);  free(upper\_tri\_matrix);  } | 0 0 0  0 0 0  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  0 0 0  0 0 0  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 1 1  0 0 0  1 1 1  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼객행렬총원소개수-matrix총원소개수  1 “ 삼객행렬총원소개수-matrix총원소개수  1 temp\_num matrix총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 삼각행렬총원소개수 삼각행렬총원소개수  1 1 1  1 1 1  1 1 1  1 1 1 |
| 합계 | 5\*matrix 총 원소개수 + 13\*삼각행렬 최대크기 + 7\*삼각행렬 총 원소개수 |

② low\_tri\_mat()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void low\_tri\_mat(int\* matrix, int size\_byte) {  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }    int\*\* lower\_tri\_matrix;  lower\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(lower\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  lower\_tri\_matrix[n][m] = 0;  }  }  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = 0; m <= n; m++) {  if (i >= temp\_num) {  lower\_tri\_matrix[n][m] = 0;  }  else {  lower\_tri\_matrix[n][m] = temp[i++];  }  }  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", lower\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");    free(temp);  free(\*lower\_tri\_matrix);  free(lower\_tri\_matrix);  } | 0 0 0  0 0 0  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  0 0 0  0 0 0  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 1 1  0 0 0  1 1 1  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼객행렬총원소개수-matrix총원소개수  1 “ 삼객행렬총원소개수-matrix총원소개수  1 temp\_num matrix총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 삼각행렬총원소개수 삼각행렬총원소개수  1 1 1  1 1 1  1 1 1  1 1 1 |
| 합계 | 5\*matrix 총 원소개수 + 13\*삼각행렬 최대크기 + 7\*삼각행렬 총원소개수 |

③ l\_up\_tri\_mat()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void l\_up\_tri\_mat(int\* matrix, int size\_byte) {  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }  int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }    int\*\* l\_upper\_tri\_matrix;  l\_upper\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(l\_upper\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  l\_upper\_tri\_matrix[n][m] = 0;  }  }  for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size - n; m++) {  if (i >= temp\_num) {  l\_upper\_tri\_matrix[n][m] = 0;  }  else {  l\_upper\_tri\_matrix[n][m] = temp[i++];  }  }  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", l\_upper\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");    free(temp);  free(\*l\_upper\_tri\_matrix);  free(l\_upper\_tri\_matrix);  } | 0 0 0  0 0 0  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  0 0 0  0 0 0  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 1 1  0 0 0  1 1 1  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼객행렬총원소개수-matrix총원소개수  1 “ 삼객행렬총원소개수-matrix총원소개수  1 temp\_num matrix총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 삼각행렬총원소개수 삼각행렬총원소개수  1 1 1  1 1 1  1 1 1  1 1 1 |
| 합계 | 5\*matrix 총 원소개수 + 13\*삼각행렬 최대크기 + 7\*삼각행렬 총 원소개수 |

④ r\_low\_tri\_mat()

|  |  |
| --- | --- |
| 문장 | s/e 빈도수 총 단계 수 |
| void r\_low\_tri\_mat(int\* matrix, int size\_byte) {  int\* temp;  int temp\_num = size\_byte / sizeof(int);  for (int i = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] == 0)  temp\_num -= 1;  }  temp = (int\*)malloc(temp\_num \* sizeof(int));  for (int i = 0, j = 0; i < size\_byte / sizeof(int); i++) {  if (matrix[i] != 0) {  temp[j++] = matrix[i];  }  }    int new\_size;  int usable;  for (int i = 0; i < MAX\_TRI\_MATRIX\_SIZE; i++) {  new\_size = i;  usable = 0;  for (int n = new\_size; n > 0; n--) {  usable += n;  }  if (usable >= temp\_num)  break;  }  int\*\* r\_lower\_tri\_matrix;  r\_lower\_tri\_matrix = (int\*\*)malloc((new\_size) \* sizeof(int));  for (int i = 0; i < new\_size; i++) {  \*(r\_lower\_tri\_matrix + i) = (int\*)malloc((new\_size) \* sizeof(int));  }  for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  r\_lower\_tri\_matrix[n][m] = 0;  }  }    for (int n = 0, i = 0; n < new\_size; n++) {  for (int m = new\_size-1-n; m < new\_size; m++) {  if (i >= temp\_num) {  r\_lower\_tri\_matrix[n][m] = 0;  }  else {  r\_lower\_tri\_matrix[n][m] = temp[i++];  }  }  }    for (int n = 0; n < new\_size; n++) {  for (int m = 0; m < new\_size; m++) {  printf("%d ", r\_lower\_tri\_matrix[n][m]);  }  printf("\n");  }  printf("\n");    free(temp);  free(\*r\_lower\_tri\_matrix);  free(r\_lower\_tri\_matrix);  } | 0 0 0  0 0 0  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  1 1 1  1 size\_byte/sizeof(int)+1 matrix총원소개수+1  1 size\_byte/sizeof(int) matrix총원소개수  1 size\_byte/sizeof(int) matrix총원소개수  0 0 0  0 0 0  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 1 1  0 0 0  1 1 1  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 MAX\_TRI\_MATRIX\_SIZE 삼각행렬최대크기  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼객행렬총원소개수-matrix총원소개수  1 “ 삼객행렬총원소개수-matrix총원소개수  1 temp\_num matrix총원소개수  1 MAX\_TRI\_MATRIX\_SIZE+1 삼각행렬최대크기+1  1 “ 삼각행렬최대크기(삼각행렬최대크기+1)  1 삼각행렬총원소개수 삼각행렬총원소개수  1 삼각행렬총원소개수 삼각행렬총원소개수  1 1 1  1 1 1  1 1 1  1 1 1 |
| 합계 | 5\*matrix 총 원소개수 + 13\*삼각행렬 최대크기 + 7\*삼각행렬 총 원소개수 |

( \* matrix 총 원소개수는 입력받은 n by n 행렬의 총 원소개수(n \* n)를 의미하며, 삼각행렬 최대 크기는

생성된 m by m 삼각행렬의 m 크기를 의미하며, 삼각행렬 총 원소개수는 m by m 삼각행렬의 총 원소개수( m \* m)을 의미)

2) 시간 복잡도 쎄타(θ) 산출

① 해당 ADT의 총 실행 step 수(연산 횟수) = 4\*(5\*matrix 총 원소개수+13\*삼각행렬 최대크기+7\*삼각행렬 총

원소개수)

= 20\*matrix 총 원소개수+52\*삼각행렬 최대크기+28\*삼각행렬 총 원소개수

② 입력받은 n by n matrix의 크기 = n , 생성된 m by m 삼각행렬의 크기 = m 이라 하고,

총 실행 step수 = f(n)이라 하면,

f(n) = 20\*n\*n + 52\*m + 28\*m\*m 이라 할 수 있다.

※정의[쎄타(Theta, θ)] : 두개의 함수 f(n), g(n)이 주어졌을 때 모든 n>=n0 에 대하여 c1\*|g(n)|<=f(n)<=c2\*|g(n)|을 만족하는 3개의 상수 c1, c2, n0 가 존재하면 f(n)은 θ(g(n))이다.

Proof) 두개의 함수 f(n)=20\*n\*n + 52\*m + 28\*m\*m, g(n)=n\*n + m\*m이 주어졌을 때, 모든 n>=1, m>=1 에

대하여,

n\*n+m\*m <= 20\*n\*n + 52\*m + 28\*m\*m <= 100(n\*n+m\*m) 을 만족하는 상수 c1=1, c2=100이

존재하므로,

**∴** f(n) = θ(g(n)) = θ(n\*n+m\*m) = θ(matrix 총 원소개수 + 삼각행렬 총 원소개수)