**과제1**



|  |  |
| --- | --- |
| **과 목 :** | **자료구조** |
| **제출 일자 :** | **2019/04/14** |
| **담당 교수 :** | **김승태** |
| **학 과 :** | **사회복지학부** |
| **학 번 :** | **20170299** |
| **이 름 :** | **김유빈** |

**(2) 미로 탐색**

본 해결책은 교수님의 힌트 강의를 듣기 전에 작성한 것이다

1(위)

**<미로 준비>**

8(오)

4(왼)

미로의 한 블록을 숫자로 변환하여 12 \* 9 ‘배열’에 저장

막힌 방향을 숫자로 더하기!

2(아래)

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

예를 들어 첫번쨰 칸은 1(위) + 4(왼) = 5를 배열에 저장

x

방

향

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 3 | 3 | 9 | 7 | 3 | 1 | 9 | 13 |
| 6 | 3 | 9 | 12 | 5 | 3 | 8 | 12 | 12 |
| …… |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

y 방향

int mazz[12][9] = { {5,3,3,9,7,3,1,9,13},

{6,3,9,12,5,3,8,12,12},

{5,3,8,6,10,5,10,12,12},

{12,13,14,13,13,14,5,10,12},

{12,12,5,8,6,3,10,5,10},

{4,10,12,4,3,3,3,10,13},

{6,3,10,12,5,3,11,5,8},

{13,5,3,10,12,5,3,10,14},

{12,6,3,1,10,6,1,3,9},

{4,3,9,4,9,5,10,13,12},

{6,9,12,12,12,4,3,10,12},

{7,10,6,10,6,10,7,3,10} };

int move[12][9]; 처음 초기값은 mazz와 같음

하지만 다음 블록으로 넘어갈 때 다음 블록에 특정한 값을 더한다.

EX) 7 3 7 -> 3으로 넘어갈 때 3에 +4를 하여 다시 갔던 방향으로 가는

것을 방지한다!

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **x** | **y** | **d** |

**<스택 준비>**

typedef struct {

Element stackArr[MAX\_STACK\_SIZE][3];

int top;

}Stack;

100 \* 3배열의 스택을 만든다.

첫번째 열에는 x좌표,

두번째 열에는 y좌표,

세번째 열에는 d 즉, 방향 값을 넣는다.

스택 s : 갈림길이 생길 때 현재 위치인 (x,y)좌표와 갈림길 방향/트인 방향을 넣는다.

스택 r : 최종경로를 출력하기 위한 스택으로

경로를 모두 저장하고(x좌표와 y좌표)

다시 예전 길로 돌아와야 할 때 **pop()**으로 경로 지우기

**<방향 정하기>** = 다음 갈 방향 정하기

while (x == 11이고 y == 8이 아닐 때까지 반복)

\* int x/세로 와 int y/가로 : 현재 위치한 경로

**1. 갈림길에 놓인다면**

즉, 두 갈래에 길이 난다면 ( 세 갈래로 나올 경우는 해당 미로에서 없다!! 왜냐하면 다음 블록으로 이동할 때마다 한 벽을 막기 때문에 원래 세 갈래가 있는 곳도 나중에는 두 갈래로 바뀐다.)

|  |
| --- |
|  |

①move[x][y] == 5 //현재 위치의 값이 5

Push(s, x, y, 8); //오른쪽 방향

Push(s, x, y, 2); //아래 방향

m = Pop(s); //갈림길 중 방향 선택

d = m[2]; //가려는 방향

②move[x][y] == 9

③move[x][y] == 3

④move[x][y] == 6

⑤move[x][y] == 10

⑥move[x][y] == 12

* ①번과 같은 방법으로 d의 값을 구한다

**2. 한 쪽으로만 갈 수 있는 길(즉, 갈림길x)**

**: d의 값(갈 방향)이 정해져 있음**

①move[x][y] == 7 -> d = 8;

②move[x][y] == 13

③move[x][y] == 11

④move[x][y] == 14

**3. 미로가 막히면/잘못된 길이면 =>** if (move[x][y] == 15) 15

m = Pop(s);

x = m[0];

y = m[1];

**d** = m[2]; -> 다시 예전 길로 돌아가기!

★스택r (최종경로 저장)에서 잘못된 경로를 뽑아내야함 = pop()

int \* arr = Pop(r); //일단 하나의 값을 뽑아서

while (arr[0] != x || arr[1] != y) //<-> arr[0] == x && arr[1] == y

arr = Pop(r); //다시 원래의 길로 돌아갈 때 까지 스택에서 뺀다

**4. 이미 간 길을 봉쇄하고 실제로 이동하기**

+4

if (d == 8) // 갈 방향이 오른쪽이라면

move[x][++y] += 4; //다음 블록에서 +4

나머지도 같은 방법으로 경로를 이동한다.

else if (d == 2)

move[++x][y] += 1;

else if (d == 4)

move[x][--y] += 8;

else if (d == 1)

move[--x][y] += 2;

Push(r, x, y, 0); //스택r에 이동 경로를 저장 -> 실제로 이동하는 구간

**<미로 그리기>**

for (i = 0;i < 12;i++) // 한 줄씩 미로 그리기 ( 세로줄 끝까지 갈 동안 반복 )

**1. 가로줄 미로 표시 (“+----+”)**

블록에서 ‘1(위)’구간이 저장된 블록에만 가로줄 표시!

나머지는 빈칸

if (mazz[i][j] == 1 or 5 or 9 or 3 or 7 or 11 or 13 or 15) //블록에서 ‘1(위)’가 있는 블록

printf("----+");

else

printf(" +");

**2. 세로줄 표시하기(“|”)**

|  |
| --- |
|  |

★ 세로줄은 한 칸의 오른쪽 세로줄만 표시하면 됨(이전 칸에서 알아서 왼쪽 세로줄을 채우기 때문)

미로 블록은 총 4가지로 분류된다.

① " ○ |" :세로줄이 있는데 최종경로인 것

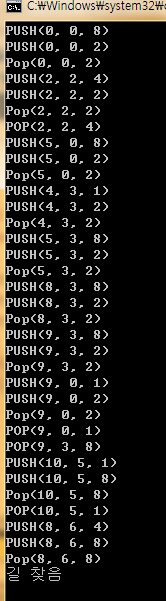
② " |" : 세로줄이 있는데 지나간 길이 아닌 것

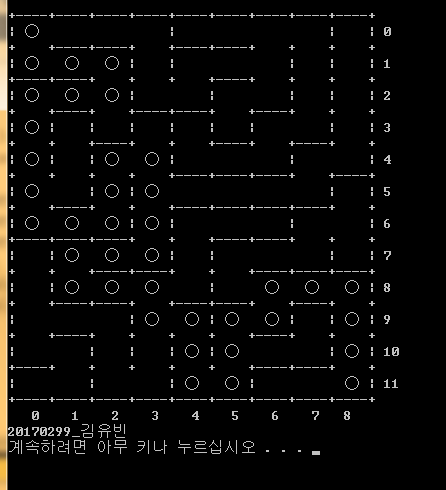
③ " ○ " : 세로줄은 없는데 지나간 길인 것

④ " " : 세로줄도 없고 지나간 길도 아닌 것

반복이 끝나면 마지막으로 "+----+----+----+----+----+----+----+----+----+” 표시

**<실행결과>**





**(4) 입력 큐 만들기**

**<기본 큐 작업>**

1. 큐 데이터 타입 만들기 typedef struct QueueType;

2. 큐 생성 함수 QueueType \* create()

3. 큐의 공백 상태 검출 함수 int is\_empty(QueueType \* q)

4. 큐의 포화 상태 검출 함수 int is\_full(QueueType \* q)

5. 삽입 함수 int enqueue(QueueType \* q, Element item)

: FRONT 와 REAR가 같아지기 이전을 ‘큐가 가득찼다’로 반환

6. 삭제 함수 Element dequeue(QueueType \* q)

**<사용자가 문자열 입력 후의 작업> - 본격적인 작업**

while (사용자가 “STOP” 입력할 때까지 이 작업을 반복)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | …. |  |  |  |  |  |  |  |  |  |  |  | …. | 18 | 19 |

FRONT = 0

REAR = 0 //맨 처음 시작

**1. 큐에 값 집어넣기**

if (user[0] >= 'A') //만약 입력이 ‘문자’이라면

for (i = 0;i < strlen(user);i++) //입력한 길이만큼 반복

enqueue(q, user[i]); -> REAR = 1

if (큐가 가득 찼다면 / count >= 20)

printf("Fail : Queue Full Queue Size=20/20\n"); //FAIL출력

else

printf(" Queue Size=%d/20\n", **++count**); //큐가 가득 안 차면 개수++

★COUNT >= 20 : ++count가 전치이므로 (큐가 가득찬 순간에서야 19 -> 20)

큐가 가득차도 “ADDQUEUE(Y) Queue Size=20/20”를 출력하기 위해

**2. 큐에서 끄집어내기**

즉, 입력이 ‘문자’가 아닌 ‘숫자’ 일 경우 if (user[0] >= '0' && user[0] <= '9')

입력한 숫자가 1. 10의 자리 2. 일의 자리 인지 구분하기

num = (user[0] - '0') \* 10 + user[1] - '0'; //십의 자리인 경우

for (NUM번 까지 삭제함수, 데이터 개수 삭제하기)

* dequeue(q), --count

3. 마지막으로 다시 입력 받기

EX)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | H | E | L | L | W | O | R | L | D |  |  |  |  |  |  |  |  |  |  |

>>> *HELLOWORLD* ***-> FRONT = 0, REAR = 9***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | L | W | O | R | L | D |  |  |  |  |  |  |  |  |  |  |

**>>> *3 ->FRONT = 3 , REAR = 9***

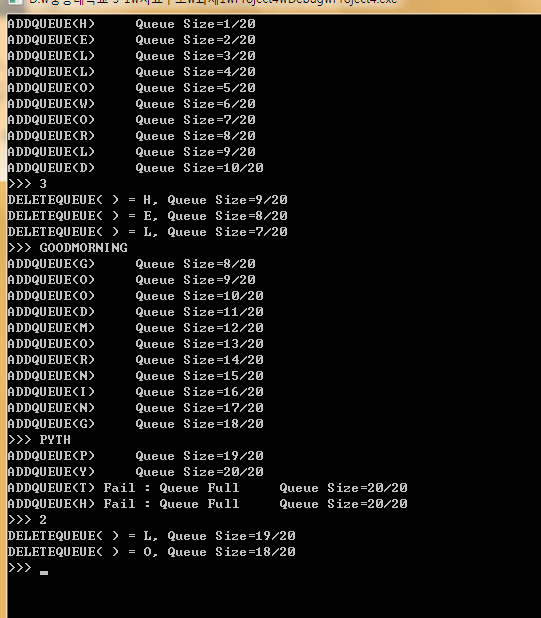
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G |  |  |  | L | W | O | R | L | D | G | O | O | D | M | O | R | N | I | N |

>>> *GOODMORNING* ***->FRONT = 3, REAR = 0***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | **P** | **Y** |  | L | W | O | R | L | D | G | O | O | D | M | O | R | N | I | N |

>>> *PYTH* ***// FAIL ->FRONT = 3, REAR = 2(포화상태이므로 더 이상 데이터 추가X)***

**<실행결과>**



**(5) 선택 정렬/퀵 정렬/힙 정렬**

★★★★제일 오래 걸렸던 문제★★★★

**<랜덤 데이터 생성>**

|  |
| --- |
| a[0] |
| a[1] |
| a[2] |
| a[3] |
| ……. |
| a[998] |
| a[999] |

|  |  |  |
| --- | --- | --- |
| Stdnum  (학번) | Name  (이름) | Phone  (전화번호) |
|  |  |  |
|  |  |  |
| …….. |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| c[1] |
| c[2] |
| c[3] |
| c[4] |
| ……. |
| c[999] |
| c[1000] |

**구조체 Student 포인터 배열 Student \* a[1000]**

포인터 배열은 총 6개 : Student \* a[1000], \* a2[1000]; -> 선택 정렬(학번, 이름)

Student \* b[1000], \* b2[1000]; -> 퀵 정렬(학번, 이름)

Student \* c[1001], \* c2[1001]; -> 힙 정렬(학번, 이름) 단, 1번지부터

for (i = 0부터 i = 999까지)

a[i] = malloc(sizeof(Student));

a2[i] = a[i];

b[i] = a[i];

b2[i] = a[i];

c[i + 1] = a[i];

c2[i + 1] = a[i]; -> 포인터 a와 같은 구조체를 가리키도록 함

1. 학번 생성

for (i = 0;i < 1000;i++) {

again = 1; -> again은 중복 검사 도구

while (again) {

a[i]->stdnum[0] = 2;

a[i]->stdnum[1] = 0;

a[i]->stdnum[2] = 1;

a[i]->stdnum[3] = (rand() % 7) + 3;

for (j = 4;j < 9;j++)

a[i]->stdnum[j] = rand() % 10;

//여기까지 하나의 랜덤 학번 생성

again = 0; //이전 구조체 중 하나라도 학번 겹치면 again = 1(다시 while문 반복)

for (x = 0;x < i; x++) { //x번째 구조체

for (y = 0;y < 9;y++) { //학번의 y번째 숫자

if (이전의 구조체의 학번과 한 숫자라도 다르면)

break; //y반복문 나가기

}

if (y == 10) { //x번째 구조체가 현재 구조체와 같다면

again = 1; //다시 학번 정하기

break; //x반복문을 나가라

}

//서로 다르다면 다음 구조체 검사

}

‘이름’과 ‘전화번호’도 같은 방법으로 랜덤하게 중복되지 않게 데이터 생성

**<선택 정렬>** void selection\_sort\_Bystdnum(Student \* list[], int n) :학번을 정렬

|  |  |  |
| --- | --- | --- |
| 201302999 | Name | Phone |
| 201486331 |  |  |
| 201456632 |  |  |
| 201385665 |  |  |
| 201864523 |  |  |
| 201965322 |  |  |
| 201779555 |  |  |

**구조체 포인터 배열 list**

**i**

least

= i 번지

i : 정렬되지 않은 구간의 첫 시작

for (정렬되지 않은 구간 2번째부터 비교) { //least == i이므로 i + 1부터 비교

for (l = 0;l < 9;l++) {

if (least번지의 구조체 학번보다 더 작은 학번이 있으면) {

least = j;

break; //l반복문 나가기(다음 구조체로 넘어가기)

}

else if(least가 가장 작은 학번이면)

break; //least구조체가 여전히 작으면 l반복문 나가기(다음 구조체로 이동)

}

SWAP(list[i], list[least], temp); //구조체 포인터 값 교환

**}**

**★<퀵 정렬>** void quick\_sort\_Bystdnum(Student \* list[], int left, int right) : 학번 정렬

**pivot** =

201502999

**high**

**low**

right

left

|  |  |  |
| --- | --- | --- |
| 201502999 | Name | Phone |
| 201789655 |  |  |
| 201366556 |  |  |
| 201406666 |  |  |
| 201955645 |  |  |
| 201365789 |  |  |
| 201236555 |  |  |

int partition\_Bystdnum(Student \* list[], int left, int right) -> 피벗을 기준으로 정렬

do {

low++;

for (i = 0;i < 9;i++) { //low와 피벗이 같으면 i++(다음 숫자로 이동)

if (low의 학번< pivot의 학번)

break; //while문으로

else if (low의 학번 > pivot의 학번)

break;

}

} while (low < right && low의 학번< pivot의 학번);

★low < right -> low++으로 배열 범위에서 벗어나는 것 방지

-> low는 201789655에서 멈춤

->high도 같은 방식으로 201236555에서 멈춤

if (low < high)

SWAP(list[low], list[high], temp);

이러한 작업을 **low < high** 까지 반복

void quick\_sort\_Bystdnum(Student \* list[], int left, int right) {

if (left < right) { //이때까지 재귀함수 반복

int q = partition\_Bystdnum(list, left, right);

quick\_sort\_Bystdnum(list, left, q - 1); //재귀함수

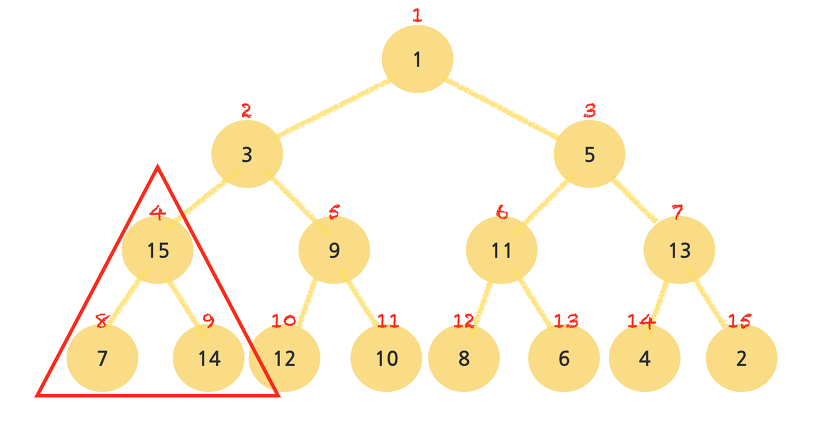
quick\_sort\_Bystdnum(list, q + 1, right);

}

}

**★★<힙 정렬>** void heapsort\_Bystdnum(Student \* a[], int n) : 학번 정렬

1. initial heap 만들기 : void adjust\_Bystdnum(Student \* a[], int root, int n)



9

16

**child**

4 -> 5

**rootkey**

= 3

[출처] <https://zeddios.tistory.com/56> 각 값을 학번이라고 가정한다.

변수 rootkey : 각 조합의 맨 꼭대기 값 , 변수 root : 꼭대기 번지수

변수 child : 자식이 있는 번지 수

if (child < n) {

for (i = 0;i < 9;i++)

if (오른쪽 자식의 학번 < 왼쪽 자식의 학번) {

child++; //자식의 번지수는 왼쪽으로 이동!

break;

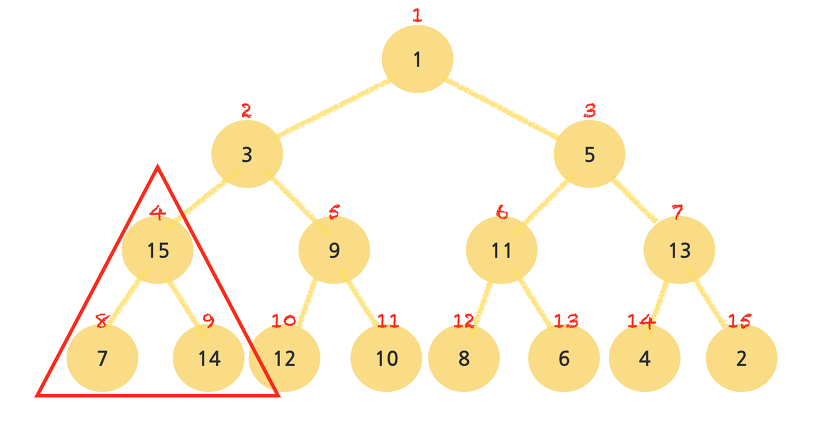
}

else if (오른쪽 자식의 학번 > 왼쪽 자식의 학번)

break;

}

-----------------------------------------------------------------------------------------------------------



16

**child**

= 10

9

**rootkey**

= 3

3

**check = 0**; //맨 꼭대기 값이 더 큰지 체크

for (i = 0;i < 9;i++) {

if (맨 꼭대기 학번> 자식의 학번) {

**check = 1;**

break;

}

else if(맨 꼭대기 학번 <자식의 학번){

for (j = 0;j < 9;j++)

부모의 학번(child / 2) = 자식의 학번**;** : 2번지가 3->16으로 복사

child \*= 2; //자식의 자식으로 내려가기 : child = 5 -> 10

break;

}

}

if (check) break; -> 4번지에 그대로 15

★★ if (rootkey[i] > a[child]->stdnum[i]) : child \*= 2 때문에 배열 범위를 벗어난 a[child]를 검사하기 때문에 오류가 난다!!!

}

for(i=0;i<9;i++)

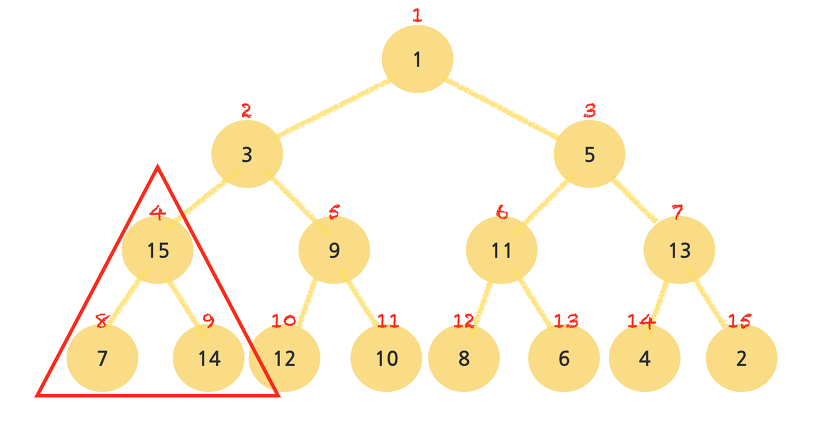
**a[child / 2]->stdnum[i] = 맨 꼭대기 학번;**

// 1. 맨 꼭대기 학번> 자식의 학번: 부모에 다시

2. 맨 꼭대기 학번< 자식의 학번 : 자식에 맨 꼭대기 값 복사 ∵(child \* 2) / 2 = child

🡪 이 모든 과정을 child <= n까지 반복

이제는 자식(10번지)와 맨 꼭대기 값(=3)를 비교하여 바꾸기



3

9

12

16

2. 제일 큰 수 (맨 꼭대기 값)이랑 마지막 번지의 수와 교환

-> 마지막 번지를 다음 힙 정렬에서 제외

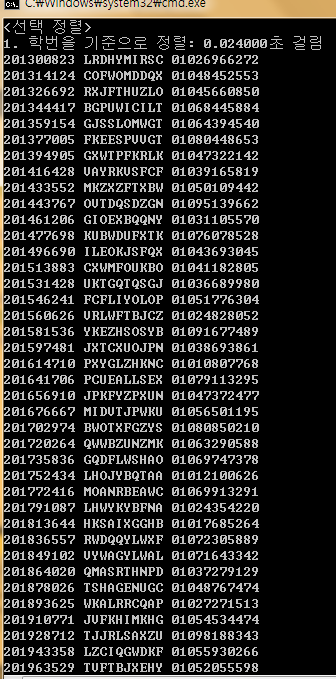
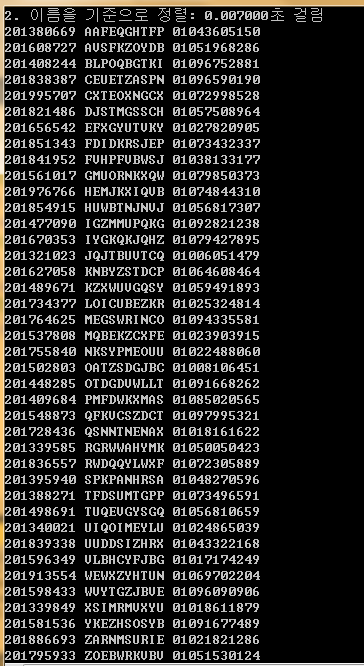
for (i = n - 1;i > 0; i--) {

SWAP(맨 꼭대기 포인터, 마지막 포인터, temp);

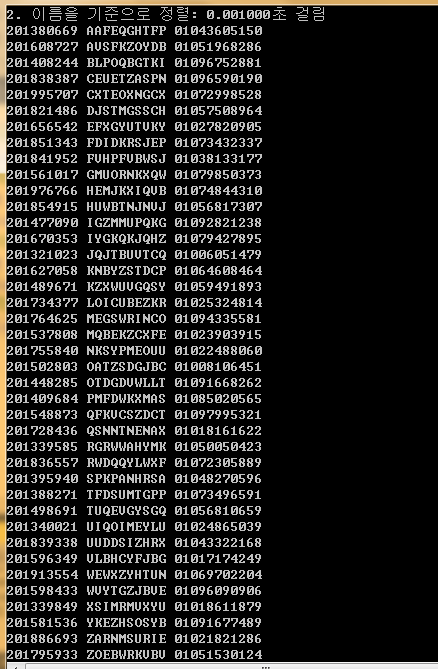
adjust\_Bystdnum(a, j, i); }

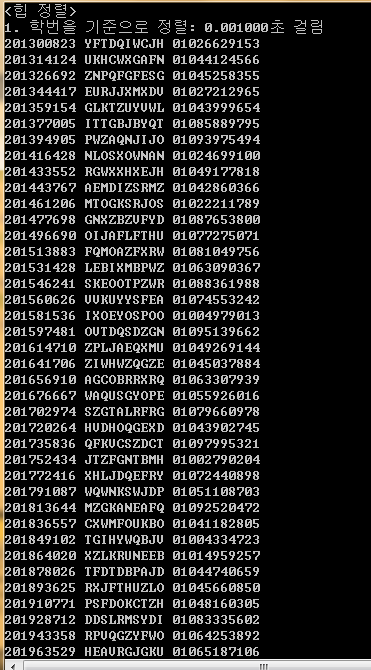
★★이미 힙 정렬된 상태에서 adjust\_Bystdnum(a, j, i);만으로 두번째 큰 수는 쉽게 맨 꼭대기로 올라가면 맨 아래 있던 숫자는 adjust\_Bystdnum(a, j, i);으로 제자리를 찾게 된다.

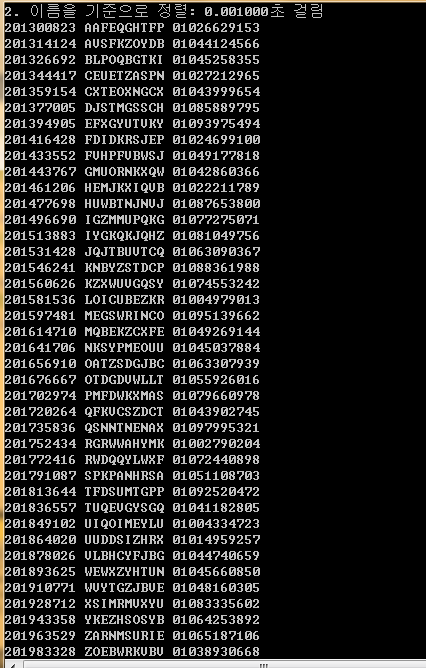
**<결과 화면>**











**(6) 퀵 정렬의 비재귀 호출 버전**

**<스택>**

비재귀 대신에 스택에 left, right 값을 넣기

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | p | 5 | 6 | 7 | 8 |

right

left

피벗 정렬

8 = r

8 = r

5 = q + 1

3 = q - 1

0 = l

8

0 = l

0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | **p** | 5 | p | 7 | 8 |

r

l

r

l

8 = r

7 = q + 1

피벗 정렬

5 = q - 1

8 = r

5 = l

3

0

5 = l

3

0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | **p** | 5 | p | p | 8 |

피벗 정렬

r

l

8

8

6

7

5

5

8 = r

5

5

7 = l

3

3

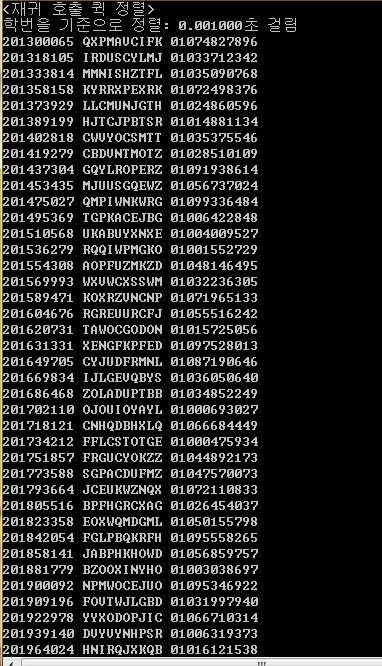
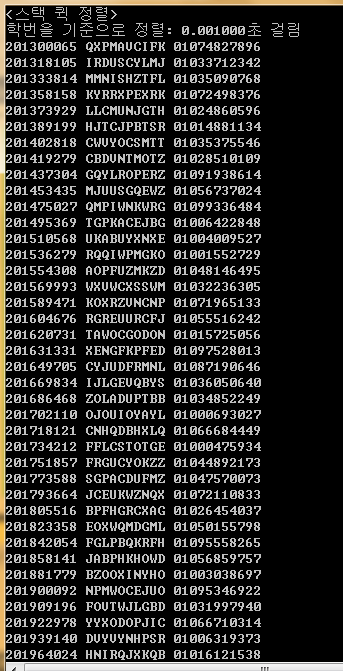
0

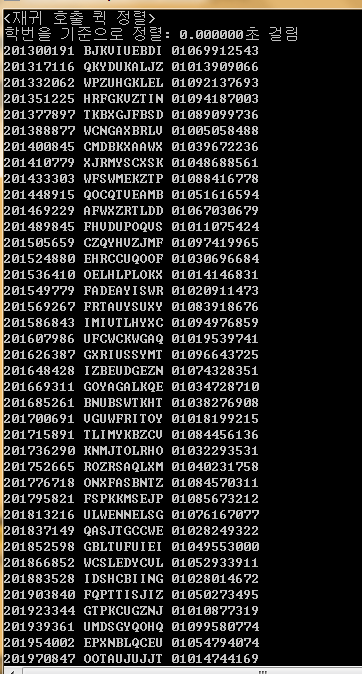
0

l < r 일 때만 피벗 정렬을 하므로 스택에는 0, 3만 남아있게 됨

나머지 0, 3도 똑 같은 식으로 반복(스택이 빌 때 까지)하여 퀵 정렬 완성

**<실행 결과>**

1. 별 차이 없거나

2. 스택이 조금 더 시간이 걸린다

