**우선순위 큐 레포트**

**202112936 엄정석**

**A. 우선순위 큐: MinHeap**

-문제

|  |
| --- |
| **우선순위 큐: MinHeap**    MinHeap Class를 구현하여 정수 N개를 입력받아 대해서 완성된 힙을 모두 출력하고, 우선순위가 높은 K개 출력 한 상태의 힙을 출력하시오.  단, N>K 이고 전체 정렬은 하지 않는다.      Input Sample 1  6 3                                                   // 정수 개수 N과 출력 개수 K를 입력받는다. 5 3 8 1 2 10                                    // 정수를 N개 입력 받는다.    output Sample 1  1 2 8 5 3 10                                     // N개의 정수를 입력 받은 힙 출력 5 10 8                                                   // K개를 출력한 상태의 힙 출력    Input Sample 2  10 5 20 15 54 60 12 5 8 17 30 22  output Sample 2  5 15 8 17 20 54 12 60 30 22 20 30 22 60 54 |

-성공 코드(C++)

|  |
| --- |
| #include <cstdio>  #include <iostream>  #define MAX\_ELEMENT 200  using namespace std;  //힙을 위한 노드 클래스  class HeapNode {  int key; //key 값  public:  HeapNode(int k = 0) : key(k) {}  void setKey(int k) { key = k; }  int getKey() { return key; }  void display() { printf("%d ", key); }  };  //최소 힙  class MinHeap {  HeapNode node[MAX\_ELEMENT];  int size;  public:  MinHeap() : size(0) {}  bool isEmpty() { return size == 0; }  bool isFull() { return size == MAX\_ELEMENT - 1; }  HeapNode& getParent(int i) { return node[i / 2]; }  HeapNode& getLeft(int i) { return node[i \* 2]; }  HeapNode& getRight(int i) { return node[i \* 2 + 1]; }  void insert(int key) {  if (isFull()) { return; }  int i = ++size; //증가된 힙 크기 위치에서 시작(힙의 마지막 자리)  //트리를 거슬러 올라가면서 부모 노드와 비교하는 과정  while (i != 1 && key < getParent(i).getKey()) {  //부모 노드보다 자신이 작으면 자신이 올라간다. 최대 힙에서 바뀌는 부분  node[i] = getParent(i);  i /= 2;  }  node[i].setKey(key);  }  HeapNode remove() {  if (isFull()) { return NULL; }  HeapNode item = node[1]; //루트노드(꺼낼 요소)  HeapNode last = node[size--]; //힙의 마지막노드  int parent = 1; //마지막 노드의 위치를 루트로 생각함  int child = 2; //루트의 왼쪽 자식 위치  //힙 트리를 벗어나지 않는 동안  while (child <= size) {  //현재 노드의 자식 노드 중 더 큰 자식노드를 찾음, 최대 힙에서 바뀌는 부분  if (child < size && getLeft(parent).getKey() > getRight(parent).getKey()) { child++; }  //child: 더 큰 자식 노드의 인덱스  //마지막 노드가 자식 노드보다 크면 ==> 이동 완료  if (last.getKey() <= node[child].getKey()) break;  //아니면 ==> 한 단계 아래로 이동  node[parent] = node[child];  parent = child;  child \*= 2;  }  node[parent] = last; //마지막 노드를 최종 위치에 저장  return item; //루트 노드 반환(즉, 루트 노드를 제거한다.)  }  HeapNode find() { return node[1]; } //루트 노드 반환  void display() { //출력  for (int i = 1, level = 1; i <= size; i++) {  if (i == level) {  level \*= 2;  }  node[i].display();  }  }  };  int main() {  int N; //노드의 개수  int K; //우선순위가 높은 K개  MinHeap heap; //최소 힙  cin >> N >> K;  for (int i = 0; i < N; i++) {  int element;  cin >> element;  heap.insert(element);  }  heap.display(); //우선 입력된 힙을 배열 꼴로 출력  printf("\n");    for (int i = 0; i < K; i++) {  heap.remove(); //그 다음 K번의 삭제 연산 수행  }  heap.display(); //이후의 힙 모습을 배열 꼴로 출력  return 0;  } |

**B. 우선순위 큐: 힙 조건 검사**

-문제

|  |
| --- |
| 우선 순위 큐 : 힙 조건 검사   N 개의 완전 이진트리 배열 A가 힙 조건을 만족하는지를 검사하는 함수를 순환적인 방법으로 구현하시오.      Sample Input 1  10                                                        // 입력 개수 N개 입력  15 14 13 12 11 10 9 8 7 5                   // 배열 A에 N개의 정수 입력    Sample OutPut 1  YES                                                     // 힙 조건을 만족하면 YES, 그렇지 않으면 NO    Sample Input 2  10  15 13 14 12 11 10 9 8 7 5    Sample OutPut 2  YES  Sample Input 3  5  30 15 16 7 20    Sample OutPut 3  NO |

-성공 코드(C++)

|  |
| --- |
| #include <cstdio>  #include <iostream>  #define MAX\_ELEMENT 200  using namespace std;  bool isHeap = true; //힙인지 아닌지 검사하는 깃발  //힙을 위한 노드 클래스  class HeapNode {  int key; //key 값  public:  HeapNode(int k = 0) : key(k) {}  void setKey(int k) { key = k; }  int getKey() { return key; }  void display() { printf("%d ", key); }  };  //최대 힙  class MaxHeap {  HeapNode node[MAX\_ELEMENT];  int size; //요소의 개수  public:  MaxHeap() : size(0) {}  bool isEmpty() { return size == 0; }  bool isFull() { return size == MAX\_ELEMENT - 1; } //199개일 때 포화 상태  HeapNode& getParent(int i) { return node[i / 2]; }  HeapNode& getLeft(int i) { return node[i \* 2]; }  HeapNode& getRight(int i) { return node[i \* 2 + 1]; }  //입력 연산(UpHeap): 새로운 노드를 마지막 레벨에서 점차 부모 노드 레벨로 올라오며 비교  void insert(int key) {  if (isFull()) { return; }  int i = ++size; //증가된 힙 크기 위치에서 시작(힙의 마지막 자리)  //트리를 거슬러 올라가면서 부모 노드와 비교하는 과정  while (i != 1 && key > getParent(i).getKey()) {  //부모 노드보다 자신이 크면 자신이 올라간다.  node[i] = getParent(i); //부모 노드를 한 층 끌어내린다.    isHeap = false;  //추가된 부분  //insert를 한 번이라도 하면 input의 이진탐색트리는 힙이 아님.    i /= 2;  } //i가 1일 때까지 반복  node[i].setKey(key); //최종 위치에 키 입력  }  //삭제 연산(DownHeap): 가장 큰 키 값을 가진 노드 삭제  //삭제 후 나머지 노드들이 힙 성질을 유지하도록 함  HeapNode remove() {  if (isFull()) { return NULL; }  HeapNode item = node[1]; //루트노드(꺼낼 요소)  HeapNode last = node[size--]; //힙의 마지막노드  int parent = 1; //마지막 노드의 위치를 루트로 생각함, 실제로는 없음  int child = 2; //루트의 왼쪽 자식 위치  //힙 트리를 벗어나지 않는 동안  while (child <= size) {  //현재 노드의 자식 노드 중 더 큰 자식노드를 찾음, 오른쪽 자식으로 이동  if (child < size && getLeft(parent).getKey() < getRight(parent).getKey()) { child++; }  //child: 더 큰 자식 노드의 인덱스  //마지막 노드가 자식 노드보다 크면 ==> 이동 완료  if (last.getKey() >= node[child].getKey()) break;  //아니면 ==> 한 단계 아래로 이동  node[parent] = node[child];  parent = child;  child \*= 2;  }  node[parent] = last; //마지막 노드를 최종 위치에 저장  return item; //루트 노드 반환(즉, 루트 노드를 제거한다.)  }  HeapNode find() { return node[1]; } //루트 노드 반환  void display() { //출력  for (int i = 1, level = 1; i <= size; i++) {  if (i == level) {  level \*= 2;  }  node[i].display();  }  }  };  int main() {  MaxHeap heap;  int N; //노드의 개수  cin >> N;  for (int i = 0; i < N; i++) {  int element;  cin >> element;  heap.insert(element); //힙에 노드 입력  }  //출력  if (isHeap == false) { printf("NO"); } //힙이 아니면 No 출력  else { printf("YES"); } //힙이면 Yes 출력  return 0;  } |