|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』과제 보고서 | | | |
| 제목 | 9장 실습( ) 과제( O ) | 제출일자 | 2022.  11 .    30 . |
| 학번 | 201911608 | 이름 | 김 지환 |

|  |
| --- |
| 02. 히프를 배열로 표현할 수 있는 이유는 무엇인가?  ⓵ 완전 이진 트리이기 때문에 ⓶ 어느 정도 정렬되어 있기 때문에  ⓷ 이진 트리이기 때문에 ⓸ 히프 조건을 만족하기 때문에  -> 히프는 완전 이진 트리 구조이기 때문에 높이별로 2^n-1 만큼의 인덱스부터 시작할 수 있다. |
| 04. 다음 중 히프 정렬이 특히 유용하게 사용될 수 있는 경우는?  ⓵ 노드의 개수 ⓶ 트리의 높이  ⓷ 항상 일정하다. ⓸ 예측 불가능하다.  -> 히프 정렬은 가장 큰 값 혹은 가장 작은 값 n개를 출력하기 위할 때 가장 유용하게 사용 될 수 있다. |
| 06. 최소 히프에서 2번째로 작은 데이터가 있는 노드는?  -> return (root -> left > root -> right) ? root -> right : root -> left |
| 08. 최소 히프를 구현한 배열의 내용이 다음과 같을 때 해당하는 히프트리를 그려라.  0 1 2 3 4 5 6 7 8   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 2 | 9 | 18 | 6 | 15 | 7 | 3 | 14 |   a[i]  (1) 이 힙에서 삭제 연산을 한번 수행한 후의 배열의 내용을 적어라.    -> [ ][ 3 ][ 6 ][ 7 ][ 9 ][ 15 ][ 18 ][ 14 ]  (2) 이 힙에서 데이터 7을 삽입한 후의 배열의 내용을 적어라.    -> [ ][ 2 ][ 6 ][ 3 ][ 7 ][ 15 ][ 18 ][ 7 ][ 14 ][ 9 ] |
| 10. 다음의 파일에 대하여 다음 물음에 답하시오.  10, 40, 30, 5, 12, 6, 15, 9, 60  (1) 위의 파일을 순차적으로 읽어서 최대 히프트리를 구성하라. 공백 트리에서 최대 히프트리가 만들어지는 과정을 보여라.    (2) 구성된 최대 히프트리가 저장된 배열의 내용을 표시하라.  [ ][ 60 ][ 40 ][ 30 ][ 12 ][ 10 ][ 6 ][ 5 ][ 9 ]  (3) 구성된 최대 히프트리에서 최댓값을 제거한 다음 재정비하는 과정을 설명하라.  부모 -> 자식 중 큰 값을 찾음 -> 큰 값을 부모로 올림 -> 반복 (자식의 index가 size이상 까지) |
| 12. 아래의 이진트리는 최소 히프트리인가? 그 이유는?    -> 그렇지 않다.  완전 이진트리 형태이지만 노드 2의 오른쪽 자식인 5번 index의 노드 6이 더 작은 값을 자식으로 가진다. |
| 14. 정렬되지 않은 배열(array)을 이용하여 우선순위 큐 추상자료형의 각종 연산들을 구현하여보라.  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #define MAX\_ELEMENT 200  #define logB(x, base) log(x)/log(base)  typedef struct {  int heap[MAX\_ELEMENT];  int heap\_size;  } HeapType;  HeapType\* create() {  return (HeapType\*)malloc(sizeof(HeapType));  }  void init(HeapType\* h) {  h->heap\_size = 0;  }  void insert(HeapType\* h, int item) {  int i;  i = ++(h->heap\_size);  while ((i != 1) && (item > h->heap[i / 2])) {  h->heap[i] = h->heap[i / 2];  i /= 2;  }  h->heap[i] = item;  }  int delete(HeapType\* h) {  int parent, child;  int item, temp;  item = h->heap[1];  temp = h->heap[(h->heap\_size)--];  parent = 1;  child = 2;  while (child <= h->heap\_size) {  if ((child < h->heap\_size) &&  (h->heap[child]) < h->heap[child + 1])  child++;  if (temp >= h->heap[child]) break;  h->heap[parent] = h->heap[child];  parent = child;  child \*= 2;  }  h->heap[parent] = temp;  return item;  }  int is\_full(HeapType\* h) {  return h->heap\_size == MAX\_ELEMENT;  }  int is\_empty(HeapType\* h) {  return h->heap\_size == 0;  }  int find(HeapType\* h) {  return h->heap[1];  }  int main(void) {  HeapType\* heap;  printf("heap 생성\n");  heap = create();    printf("heap 초기화\n");  init(heap);  int arr[10] = { 2, 10, 40, 30, 5, 12, 6, 15, 9, 60 };    printf("is\_empty()? - ");  is\_empty(heap) ? printf("Yes\n") : printf("No\n");  printf("임의의 배열 10개 생성\n");  for (int i = 0; i < 10;i++) {  insert(heap, arr[i]);  printf("[%d]", arr[i]);  }  printf("\nis\_empty()? - ");  is\_empty(heap) ? printf("Yes\n") : printf("No\n");  printf("임의의 배열 값 힙에 삽입\n");  printf("힙의 우선 요소 : %d\n", find(heap));  printf("힙의 우선 요소 삭제\n");  printf("삭제된 요소 : %d\n", delete(heap));  printf("힙의 우선 요소 재확인 : %d\n", find(heap));  printf("is\_full)? - ");  is\_full(heap) ? printf("Yes\n") : printf("No\n");  free(heap);  return 0;  }  실행결과    -> heap의 adt 구현 |
| 16. 최소 히프에서 임의의 요소를 삭제하는 C 함수를 작성하라. 결과 히프는 히프의 조건을 만족하여야 한다.  #include <stdio.h>  #include <stdlib.h>  #define MAX\_ELEMENT 200  #define SIZE 8  typedef struct {  int heap[MAX\_ELEMENT];  int heap\_size;  } HeapType;  HeapType\* create() {  return (HeapType\*)malloc(sizeof(HeapType));  }  void init(HeapType\* h) {  h->heap\_size = 0;  }  void insert(HeapType\* h, int item) {  int i;  i = ++(h->heap\_size);  while ((i != 1) && (item < h->heap[i / 2])) {  h->heap[i] = h->heap[i / 2];  i /= 2;  }  h->heap[i] = item;  }  int delete(HeapType\* h, int key) {  int parent, child;  int item, temp;  for (int i = 0; i < h->heap\_size; i++) {  if (key == h->heap[i]) {  item = h->heap[i];  parent = i;  child = 2 \* i;  break;  }  }  temp = h->heap[h->heap\_size--];  while (child <= h->heap\_size) {  if ((child < h->heap\_size) &&  (h->heap[child]) > h->heap[child + 1])  child++;  if (temp < h->heap[child]) break;  h->heap[parent] = h->heap[child];  parent = child;  child \*= 2;  }  h->heap[parent] = temp;  return item;  }  void print\_heap(HeapType\* heap) {  printf("\n\n=========================== heap 생성 ========================\n");  int i = 0, h = 1, node = 1;  while (heap->heap\_size != i++) {  if (h == 1) printf("\t\t\t\t");  if (h == 2) {  if (node == 1) printf("\t\t");  else printf("\t\t\t\t");  }  if (h == 4) {  if (node == 1) printf("\t");  else printf("\t\t");  }  if (h == 8) {  if (node != 1) printf("\t\t");  }  printf("%d ", heap->heap[i]);  if (node == h) {  printf("\n");  h <<= 1;  node = 0;  }  node++;  }  }  int main(void) {  HeapType\* heap = create();  init(heap);  int list[SIZE] = { 2, 9, 18, 6, 15, 7, 3, 14 };  for (int i = 0; i < SIZE; i++) {  insert(heap, list[i]);  }  printf("\n 2 삭제 전 트리 구조 ===========");  print\_heap(heap);  printf("\n\ndelete - %d\n\n", delete(heap, 2));  printf("\n 2 삭제 후 트리 구조 ===========");  print\_heap(heap);  return 0;  }  실행결과    -> 트리 구조가 힙의 조건을 만족. |