|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 9장 실습( O ) 과제( ) | 제출일자 | 2022.  11 .    24 . |
| 학번 | 201911608 | 이름 | 김 지환 |

|  |
| --- |
| 1. 프로그램 9.4 - 히프정렬 프로그램  #include <stdio.h>  #include <stdlib.h>  #define MAX\_ELEMENT 200  typedef struct {  int key;  } element;  typedef struct {  element heap[MAX\_ELEMENT];  int heap\_size;  } HeapType;  HeapType\* create() {  return (HeapType\*)malloc(sizeof(HeapType));  }  void init(HeapType\* h) {  h->heap\_size = 0;  }  void insert\_max\_heap(HeapType\* h, element item) {  int i;  i = ++(h->heap\_size);  while ((i != 1) && (item.key > h->heap[i / 2].key)) {  h->heap[i] = h->heap[i / 2];  i /= 2;  }  h->heap[i] = item;  }  element delete\_max\_heap(HeapType\* h) {  int parent, child;  element 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].key) < h->heap[child + 1].key)  child++;  if (temp.key >= h->heap[child].key) break;  h->heap[parent] = h->heap[child];  parent = child;  child \*= 2;  }  h->heap[parent] = temp;  return item;  }  void heap\_sort(element a[], int n) {  int i;  HeapType\* h;  h = create();  init(h);  for (i = 0; i < n; i++) {  insert\_max\_heap(h, a[i]);  }  for (i = (n - 1); i >= 0; i--) {  a[i] = delete\_max\_heap(h);  }  free(h);  }  #define SIZE 8  int main(void) {  element list[SIZE] = { 23, 56, 11, 9, 56, 99, 27, 34 };  heap\_sort(list, SIZE);  for (int i = 0; i < SIZE; i++) {  printf("%d ", list[i].key);  }  printf("\n");  return 0;  }  실행결과    -> 99  56 56  34 27 23 11  9  list 값들을 heap 구조에 저장하게 되면 위와 같은 형태로 저장된다.  heap\_sort()에서 delete한 결과를 a[i] (i = n-1) 부터 a[0]까지 삽입하므로  출력한 결과는 내림차순이 아닌 오름차순이 된다. |
| 2. Quiz(342쪽) - 프로그램 9.4의 코드를 최소 히프 사용하여 수정 실행  #include <stdio.h>  #include <stdlib.h>  #define MAX\_ELEMENT 200  typedef struct {  int key;  } element;  typedef struct {  element heap[MAX\_ELEMENT];  int heap\_size;  } HeapType;  HeapType\* create() {  return (HeapType\*)malloc(sizeof(HeapType));  }  void init(HeapType\* h) {  h->heap\_size = 0;  }  void insert\_min\_heap(HeapType\* h, element item) {  int i;  i = ++(h->heap\_size);  while ((i != 1) && (item.key < h->heap[i / 2].key)) {  h->heap[i] = h->heap[i / 2];  i /= 2;  }  h->heap[i] = item;  }  element delete\_min\_heap(HeapType\* h) {  int parent, child;  element 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].key) < h->heap[child + 1].key)  child++;  if (temp.key <= h->heap[child].key) break;  h->heap[parent] = h->heap[child];  parent = child;  child \*= 2;  }  h->heap[parent] = temp;  return item;  }  void heap\_sort(element a[], int n) {  int i;  HeapType\* h;  h = create();  init(h);  for (i = 0; i < n; i++) insert\_min\_heap(h, a[i]);  for (i = (n - 1); i >= 0; i--) a[i] = delete\_min\_heap(h);  free(h);  }  #define SIZE 8  int main(void) {  element list[SIZE] = { 23, 56, 11, 9, 56, 99, 27, 34 };  heap\_sort(list, SIZE);  for (int i = 0; i < SIZE; i++) printf("%d ", list[i].key);  printf("\n");  return 0;  }  실행결과    -> 9  11 23  27 34 56 56  99  리스트 값을 힙에 삽입시 위와 같은 형태의 구조를 띤다.  삭제 할 때마다 루트인 최소값들이 제거되고 루트 값들은 배열의 마지막부터 0번 째 인덱스까지 값이 저장되며 실행결과와 같은 내림차순 결과가 나온다. |
| 3. 프로그램 9.6 - 허프만 코드 프로그램(최소 히프 사용)  #include <stdio.h>  #include <stdlib.h>  #define MAX\_ELEMENT 200  typedef struct TreeNode {  int weight;  char ch;  struct TreeNode\* left;  struct TreeNode\* right;  } TreeNode;  typedef struct {  TreeNode\* ptree;  char ch;  int key;  } element;  typedef struct {  element heap[MAX\_ELEMENT];  int heap\_size;  } HeapType;  HeapType\* create() {  return (HeapType\*)malloc(sizeof(HeapType));  }  void init(HeapType\* h) {  h->heap\_size = 0;  }  void insert\_min\_heap(HeapType\* h, element item) {  int i;  i = ++(h->heap\_size);  while ((i != 1) && (item.key < h->heap[i / 2].key)) {  h->heap[i] = h->heap[i / 2];  i /= 2;  }  h->heap[i] = item;  }  element delete\_min\_heap(HeapType\* h) {  int parent, child;  element 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].key) > h->heap[child + 1].key)  child++;  if (temp.key < h->heap[child].key) break;  h->heap[parent] = h->heap[child];  parent = child;  child \*= 2;  }  h->heap[parent] = temp;  return item;  }  TreeNode\* make\_tree(TreeNode\* left, TreeNode\* right) {  TreeNode\* node = (TreeNode\*)malloc(sizeof(TreeNode));  node->left = left;  node->right = right;  return node;  }  void destroy\_tree(TreeNode\* root) {  if (root == NULL) return;  destroy\_tree(root->left);  destroy\_tree(root->right);  free(root);  }  int is\_leaf(TreeNode\* root) {  return !(root->left) && !(root->right);  }  void print\_array(int codes[], int n) {  for (int i = 0; i < n; i++)  printf("%d", codes[i]);  printf("\n");  }  void print\_codes(TreeNode\* root, int codes[], int top) {  if (root->left) {  codes[top] = 1;  print\_codes(root->left, codes, top + 1);  }  if (root->right) {  codes[top] = 0;  print\_codes(root->right, codes, top + 1);  }  if (is\_leaf(root)) {  printf("%c: ", root->ch);  print\_array(codes, top);  }  }  void huffman\_tree(int freq[], char ch\_list[], int n) {  int i;  TreeNode\* node, \* x;  HeapType\* heap;  element e, e1, e2;  int codes[100];  int top = 0;  heap = create();  init(heap);  for (i = 0; i < n; i++) {  node = make\_tree(NULL, NULL);  e.ch = node->ch = ch\_list[i];  e.key = node->weight = freq[i];  e.ptree = node;  insert\_min\_heap(heap, e);  }  for (i = 1; i < n; i++) {  e1 = delete\_min\_heap(heap);  e2 = delete\_min\_heap(heap);  x = make\_tree(e1.ptree, e2.ptree);  e.key = x->weight = e1.key + e2.key;  e.ptree = x;  printf("%d+%d->%d \n", e1.key, e2.key, e.key);  insert\_min\_heap(heap, e);  }  e = delete\_min\_heap(heap);  print\_codes(e.ptree, codes, top);  destroy\_tree(e.ptree);  free(heap);  }  int main(void) {  char ch\_list[] = { 's', 'i', 'n', 't', 'e' };  int freq[] = { 4, 6, 8, 12, 15 };  huffman\_tree(freq, ch\_list, 5);  return 0;  }  실행결과    -> 최소 힙으로 구현되었기 때문에 리스트 값은 힙에 아래와 같이 저장된다.  4 6 8 12 15  허프만 코딩에 의해 힙은 허프만 합인 ‘45’만 저장된다.  45를 처음으로 지정하고 왼쪽을 1 오른쪽을 0으로 정한다.  0부터 top까지 후위순회로 출력하면 다음과 같은 결과 값이 나온다. |
| 5. 연습문제 15 - 연결리스트를 사용해서 ADT의 연산들을 구현하여 연습문제 13의 연산들을 실행  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct Node {  element data;  struct Node\* next;  } Node;  typedef struct {  Node\* head;  }Pri\_QueueType;  Pri\_QueueType\* create() {  return (Pri\_QueueType\*)malloc(sizeof(Pri\_QueueType\*));  }  void init(Pri\_QueueType\* pq) {  pq->head = (Node\*)malloc(sizeof(Node));  pq->head->data = NULL;  pq->head->next = NULL;  }  void insert(Pri\_QueueType\* pq, element value) {  Node\* temp = (Node\*)malloc(sizeof(Node));  temp->data = value;  temp->next = pq->head;  pq->head = temp;  }  element delete(Pri\_QueueType\* pq) {  element target\_data;  Node\* target = pq->head;  Node\* pointer = pq->head;  while (1) {  if (pointer->next == NULL) break;  if (pointer->data > target->data) {  target = pointer;  }  pointer = pointer->next;  }  target\_data = target->data;  Node\* removed = target->next;  target->data = removed->data;  target->next = removed->next;  free(removed);  return target\_data;  }  void print\_pqueue(Pri\_QueueType\* pq) {  Node\* pointer = pq->head;  while (1) {  if (pointer->next == NULL) {  printf("NULL\n");  return;  }  else {  printf("[%d] -> ", pointer->data);  pointer = pointer->next;  }  }  }  int main() {  Pri\_QueueType\* pq = create();  element temp;  init(pq);  insert(pq, 20);  insert(pq, 12);  insert(pq, 3);  insert(pq, 2);  delete(pq);  insert(pq, 5);  insert(pq, 16);  delete(pq);  insert(pq, 1);  delete(pq);  print\_pqueue(pq);  free(pq);  return 0;  }  실행결과 |