|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』기말고사 답안지 | | | |
| 제목 | 기말고사 | 제출일자 | 2022.     12.    21 . |
| 학번 | 201911608 | 이름 | 김지환 |

1. 다음의 입력에 대하여 48, 33, 10, 55, 14, 15, 3

1) 이진 탐색트리(binary search tree)를 만들고 중위 순회(inorder traversal)결과를 출력

2) 전위 순회 (preorder traversal)의 결과를 출력

|  |
| --- |
| ==============Source Code================== |
| #include <stdio.h>  #include <stdlib.h>  typedef struct testTree {  int data;  struct testTree\* l, \* r;  } testTree;  testTree\* root(int v) {  testTree\* t = (testTree\*)malloc(sizeof(testTree));  t->data = v;  t->l = t->r = NULL;  return t;  }  testTree\* insert(testTree\* node, int data) {  if (node == NULL) return root(data);  if (data < node->data)  node->l = insert(node->l, data);  else if (data > node->data)  node->r = insert(node->r, data);  return node;  }  void inorder(testTree\* root) {  if (root) {  inorder(root->l);  printf("[%d] ", root->data);  inorder(root->r);  }  }  void preorder(testTree\* root) {  if (root) {  printf("[%d] ", root->data);  preorder(root->l);  preorder(root->r);  }  }  int main(void) {  int arr[7] = { 48, 33, 10, 55, 14, 15, 3 };  testTree\* root = NULL;  for (int i = 0; i < 7;i++) {  root = insert(root, arr[i]);  }  printf("1) 이진 탐색 트리 중위 순회 결과 \n");  inorder(root);  printf("\n2) 이진 탐색 트리 전위 순회 결과\n");  preorder(root);  return 0;  } |
| ==============실행 결과================== |
| 1) 이진 탐색 트리 중위 순회 결과    2) 이진 탐색 트리 전위 순회 결과 |

2. 아래 데이터로 최소 히프트리를 구성하는 프로그램을 구현하여 실행결과를 보이시오.

20, 50, 15, 25, 30

1) 위 히프트리를 생성하는 과정을 배열로 출력

2) 최소값을 제거할 때 downheap 과정을 배열로 출력

|  |
| --- |
| ==============Source Code================== |
| #include <stdio.h>  #include <stdlib.h>  typedef struct {  int data[10];  int size;  } Heap;  Heap\* create() {  return (Heap\*)malloc(sizeof(Heap));  }  void init(Heap\* h) {  h->size = 0;  }  void insert(Heap\* h, int data) {  int i;  i = ++(h->size);  while ((i != 1) && (data < h->data[i / 2])) {  h->data[i] = h->data[i / 2];  i /= 2;  }  h->data[i] = data;  }  int delete(Heap\* h) {  int p, c;  int item, temp;  item = h->data[1];  temp = h->data[(h->size)--];  p = 1;  c = 2;  while (c <= h->size) {  if ((c < h->size) && (h->data[c]) > h->data[c + 1]) c++;  if (temp < h->data[c]) break;  h->data[p] = h->data[c];  p = c;  c \*= 2;  }  h->data[p] = temp;  return item;  }  void print\_heap(Heap\* h) {  for (int i = 1; i <= h->size; i++) {  printf("[%d]", h->data[i]);  }  }  int main(void) {  int arr[5] = { 20, 50,15 ,25, 30 };  Heap\* h;  h = create(); init(h);  printf("히프트리 배열 생성과정\n");  for (int i = 0; i < 5;i++) {  insert(h, arr[i]);  print\_heap(h);  printf("\n");  }  printf("\n\n히프트리 배열 제거과정\n");  for (int i = 0; i < 5;i++) {  delete(h, arr[i]);  print\_heap(h);  printf("\n");  }  free(h);  return 0;  } |
| ==============실행 결과================== |
| 1) 위 히프트리를 생성하는 과정을 배열로 출력    2) 최소값을 제거할 때 downheap 과정을 배열로 출력 |

3. 아래와 같이 각 글자들의 빈도가 있을 때, 호프만 코드를 계산하여 출력하세요.

a : 1, b : 2, c: 4, d: 7, e: 8, f: 10

|  |
| --- |
| ==============Source Code================== |
| #include <stdio.h>  #include <stdlib.h>  typedef struct Tree {  int weight;  char ch;  struct Tree\* left;  struct Tree\* right;  } Tree;  typedef struct {  Tree\* ptree;  char ch;  int key;  } element;  typedef struct {  element heap[101];  int size;  } Heap;  Heap\* create() {  return (Heap\*)malloc(sizeof(Heap));  }  void init(Heap\* h) {  h->size = 0;  }  void insert\_min(Heap\* h, element item) {  int i;  i = ++(h->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\* h) {  int parent, child;  element item, temp;  item = h->heap[1];  temp = h->heap[(h->size)--];  parent = 1;  child = 2;  while (child <= h->size) {  if ((child < h->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;  }  Tree\* make\_tree(Tree\* left, Tree\* right) {  Tree\* node = (Tree\*)malloc(sizeof(Tree));  node->left = left;  node->right = right;  return node;  }  void destroy\_tree(Tree\* root) {  if (root == NULL) return;  destroy\_tree(root->left);  destroy\_tree(root->right);  free(root);  }  int is\_leaf(Tree\* root) {  return !(root->left) && !(root->right);  }  void print\_code(int codes[], int n) {  for (int i = 0; i < n; i++) printf("%d", codes[i]);  printf("\n");  }  void print\_huffman(Tree\* root, int codes[], int top) {  if (root->left) {  codes[top] = 1;  print\_huffman(root->left, codes, top + 1);  }  if (root->right) {  codes[top] = 0;  print\_huffman(root->right, codes, top + 1);  }  if (is\_leaf(root)) {  printf("%c: ", root->ch);  print\_code(codes, top);  }  }  void huffman\_tree(int freq[], char ch\_list[], int n) {  int i;  Tree\* node, \* x;  Heap\* 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, e);  }  for (i = 1; i < n; i++) {  e1 = delete\_min(heap);  e2 = delete\_min(heap);  x = make\_tree(e1.ptree, e2.ptree);  e.key = x->weight = e1.key + e2.key;  e.ptree = x;  insert\_min(heap, e);  }  e = delete\_min(heap);  print\_huffman(e.ptree, codes, top);  destroy\_tree(e.ptree);  free(heap);  }  int main(void) {  char ch[6] = { 'a', 'b', 'c', 'd', 'e', 'f'};  int freq[6] = { 1, 2, 4, 7, 8, 10 };  huffman\_tree(freq, ch, 6);  return 0;  } |
| ==============실행 결과================== |
|  |

4. 다음 그래프가 주어 졌을 때, 그래프를 인접 리스트로 구현하여 다음에 답하시오.

1) Kruskal 알고리즘을 이용하여 최소 비용 신장 트리를 만드는 과정을 출력

2) 정점 0으로부터 시작하는 깊이 우선 탐색한 과정을 출력

|  |
| --- |
| ==============Source Code================== |
| #include <stdio.h>  #include <stdlib.h>  #define TRUE 1  #define FALSE 0  typedef struct Node {  int v;  int w;  struct Node\* link;  } Node;  typedef struct Graph {  int n;  Node\* adj\_list[11];  } Graph;  void init(Graph\* g) {  int v;  g->n = 0;  for (v = 0; v < 11; v++) g->adj\_list[v] = NULL;  }  void insert\_vertex(Graph\* g, int v) {  if (((g->n) + 1) > 11) return;  g->n++;  }  void insert\_edge(Graph\* g, int u, int v, int w) {  Node\* node1, \*node2;  if (u >= g->n || v >= g->n) return;  node1 = (Node\*)malloc(sizeof(Node));  node1->v = v;  node1->w = w;  node1->link = g->adj\_list[u];  g->adj\_list[u] = node1;  node2 = (Node\*)malloc(sizeof(Node));  node2->v = u;  node2->w = w;  node2->link = g->adj\_list[v];  g->adj\_list[v] = node2;  }  void print\_adj\_list(Graph\* g) {  for (int i = 0; i < g->n; i++) {  Node\* p = g->adj\_list[i];  printf("정점 %d의 인접 리스트 ", i);  while (p != NULL) {  printf("-> (%d, %d) ", p->v, p->w);  p = p->link;  }  printf("\n");  }  }  int visited[10];  void dfs\_list(Graph\* g, int v) {  int w;  visited[v] = TRUE;  printf("정점 %d -> ", v);  for (w = 0; w < g->n; w++) {  Node\* p = g->adj\_list[w];  while (p != NULL && !visited[w]) {  dfs\_list(g, w);  }  }  }  int main() {  Graph\* g;  g = (Graph\*)malloc(sizeof(Graph));  init(g);  for (int i = 0;i < 7;i++)  insert\_vertex(g, i);  insert\_edge(g, 0, 1, 7);  insert\_edge(g, 0, 3, 5);  insert\_edge(g, 1, 2, 8);  insert\_edge(g, 1, 3, 9);  insert\_edge(g, 1, 4, 7);  insert\_edge(g, 2, 4, 5);  insert\_edge(g, 3, 4, 15);  insert\_edge(g, 3, 5, 6);  insert\_edge(g, 4, 5, 8);  insert\_edge(g, 4, 6, 9);  insert\_edge(g, 5, 6, 11);  print\_adj\_list(g);  printf("\n\n");  dfs\_list(g, 0);  free(g);  return 0;  } |
| ==============실행 결과================== |
|  |

3)