/\*

240508 트리

\*/

1. 힙 트리

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#define MAX\_ELEMENT 200

typedef struct {

int key;

} element;

typedef struct {

element heap[MAX\_ELEMENT];

int heap\_size;

} HeapType;

// 생성 함수

HeapType\* create()

{

return (HeapType\*)malloc(sizeof(HeapType));

}

// 초기화 함수

void init(HeapType\* h)

{

h->heap\_size = 0;

}

// 현재 요소의 개수가 heap\_size인 히프 h에 item을 삽입한다.

// 삽입 함수

void insert\_max\_heap(HeapType\* h, element item)

{

int i;

i = ++(h->heap\_size);

// 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

while ((i != 1) && (item.key > h->heap[i / 2].key)) {

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item; // 새로운 노드를 삽입

}

// 삭제 함수

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) {

// 현재 노드의 자식노드 중 더 작은 자식노드를 찾는다.

if ((child < h->heap\_size) &&

(h->heap[child].key) < h->heap[child + 1].key)

child++;

if (temp.key >= h->heap[child].key) break;

// 한 단계 아래로 이동

h->heap[parent] = h->heap[child];

parent = child;

child \*= 2;

}

h->heap[parent] = temp;

return item;

}

int main(void)

{

element e1 = { 10 }, e2 = { 5 }, e3 = { 30 };

element e4, e5, e6;

HeapType\* heap;

heap = create(); // 히프 생성

init(heap); // 초기화

// 삽입

insert\_max\_heap(heap, e1);

insert\_max\_heap(heap, e2);

insert\_max\_heap(heap, e3);

// 삭제

e4 = delete\_max\_heap(heap);

printf("< %d > ", e4.key);

e5 = delete\_max\_heap(heap);

printf("< %d > ", e5.key);

e6 = delete\_max\_heap(heap);

printf("< %d > \n", e6.key);

free(heap);

return 0;

}

Heap 정렬 프로그램

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#define MAX\_ELEMENT 200

typedef struct {

int key;

} element;

typedef struct {

element heap[MAX\_ELEMENT];

int heap\_size;

} HeapType;

// 생성 함수

HeapType\* create()

{

return (HeapType\*)malloc(sizeof(HeapType));

}

// 초기화 함수

void init(HeapType\* h)

{

h->heap\_size = 0;

}

// 현재 요소의 개수가 heap\_size인 히프 h에 item을 삽입한다.

// 삽입 함수

void insert\_max\_heap(HeapType\* h, element item)

{

int i;

i = ++(h->heap\_size);

// 트리를 거슬러 올라가면서 부모 노드와 비교하는 과정

while ((i != 1) && (item.key > h->heap[i / 2].key)) {

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item; // 새로운 노드를 삽입

}

// 삭제 함수

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) {

// 현재 노드의 자식노드 중 더 작은 자식노드를 찾는다.

if ((child < h->heap\_size) && (h->heap[child].key) < h->heap[child + 1].key) child++;

if (temp.key >= h->heap[child].key) break;

// 한 단계 아래로 이동

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;

}

그래프. 인접 행렬 구현

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#define MAX\_VERTICES 50

typedef struct GraphType {

int n;// 정점의 개수

int adj\_mat[MAX\_VERTICES][MAX\_VERTICES];

} GraphType;

// 그래프 초기화

void init(GraphType\* g)

{

int r, c;

g->n = 0;

for (r = 0; r < MAX\_VERTICES; r++)

for (c = 0; c < MAX\_VERTICES; c++)

g->adj\_mat[r][c] = 0;

}

// 정점 삽입 연산

void insert\_vertex(GraphType\* g, int v)

{

if (((g->n) + 1) > MAX\_VERTICES) {

fprintf(stderr, "그래프: 정점의 개수 초과");

return;

}

g->n++;

}

// 간선 삽입 연산

void insert\_edge(GraphType\* g, int start, int end)

{

if (start >= g->n || end >= g->n) {

fprintf(stderr, "그래프: 정점 번호 오류");

return;

}

g->adj\_mat[start][end] = 1;

g->adj\_mat[end][start] = 1;

}

// 인접 행렬 출력 함수

void print\_adj\_mat(GraphType\* g)

{

for (int i = 0; i < g->n; i++) {

for (int j = 0; j < g->n; j++) {

printf("%2d ", g->adj\_mat[i][j]);

}

printf("\n");

}

}

void main()

{

GraphType\* g;

g = (GraphType\*)malloc(sizeof(GraphType));

init(g);

for (int i = 0; i < 4; i++)

insert\_vertex(g, i);

insert\_edge(g, 0, 1);

insert\_edge(g, 0, 2);

insert\_edge(g, 0, 3);

insert\_edge(g, 1, 2);

insert\_edge(g, 2, 3);

print\_adj\_mat(g);

free(g);

}

인접 리스트 그래프

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#define MAX\_VERTICES 50

typedef struct GraphNode

{

int vertex;

struct GraphNode\* link;

} GraphNode;

typedef struct GraphType {

int n;// 정점의 개수

GraphNode\* adj\_list[MAX\_VERTICES];

} GraphType;

// 그래프 초기화

void init(GraphType\* g)

{

int v;

g->n = 0;

for (v = 0; v < MAX\_VERTICES; v++)

g->adj\_list[v] = NULL;

}

// 정점 삽입 연산

void insert\_vertex(GraphType\* g, int v)

{

if (((g->n) + 1) > MAX\_VERTICES) {

fprintf(stderr, "그래프: 정점의 개수 초과");

return;

}

g->n++;

}

// 간선 삽입 연산, v를 u의 인접 리스트에 삽입한다.

void insert\_edge(GraphType\* g, int u, int v)

{

GraphNode\* node;

if (u >= g->n || v >= g->n) {

fprintf(stderr, "그래프: 정점 번호 오류");

return;

}

node = (GraphNode\*)malloc(sizeof(GraphNode));

node->vertex = v;

node->link = g->adj\_list[u];

g->adj\_list[u] = node;

}

void print\_adj\_list(GraphType\* g)

{

for (int i = 0; i < g->n; i++) {

GraphNode\* p = g->adj\_list[i];

printf("정점 %d의 인접 리스트 ", i);

while (p != NULL) {

printf("-> %d ", p->vertex);

p = p->link;

}

printf("\n");

}

}

int main()

{

GraphType\* g;

g = (GraphType\*)malloc(sizeof(GraphType));

init(g);

for (int i = 0; i < 4; i++)

insert\_vertex(g, i);

insert\_edge(g, 0, 1);

insert\_edge(g, 1, 0);

insert\_edge(g, 0, 2);

insert\_edge(g, 2, 0);

insert\_edge(g, 0, 3);

insert\_edge(g, 3, 0);

insert\_edge(g, 1, 2);

insert\_edge(g, 2, 1);

insert\_edge(g, 2, 3);

insert\_edge(g, 3, 2);

print\_adj\_list(g);

free(g);

return 0;

}