알고리즘\_과제\_6

(Chained Matrix multiplication & Optimal binary tree & Traveling Salesman Problems)



학부:컴퓨터학부

학번:20162518

출석번호 : 156번

이름 : 최승서

14) 5개의 행렬 A1, A2, A3, A4, A5의 곱셉연산이 최적의 해가 나오도록 하는 순서를 찾고 Matrix M과 Matrix P를 구하라

Chained Matrix Multiplication 소스코드

#include <stdio.h>

#include<stdlib.h>

#include <time.h>

int \*make\_d(int);

int \*\*make\_arr(int);

int minmult(int, const int[], int\*[]);

void order(int, int, int\*[]);

void print\_map(int\*[], int);

int main(void)

{

int i;

int n = 0, ret;

int \*d, \*\*P;

printf("[Minimum Multiplications algorithm]\n\n");

printf("Input array number : ");

scanf("%d", &n);

d = make\_d(n);

P = make\_arr(n + 1);

ret = minmult(n, d, P);

printf("\n [P]");

print\_map(P, n);

printf("Multiplications order is : ");

order(1, n, P);

printf("\nMinimum Multiplications is %d\n", ret);

free(d);

for (i = 0; i < n; i++)

free(P[i]);

free(P);

getchar();

getchar();

system("clear");

return 0;

}

int \*make\_d(int n) // d배열생성

{

int i;

int \*d;

d = (int\*)malloc(sizeof(int)\*(n + 1)); // n크기만큼동적할당

for (i = 0; i <= n; i++)

{

printf("input d[%d] : ", i); // d배열의인자를받음

scanf("%d", &d[i]);

}

return d;

}

int \*\*make\_arr(int n) //2차원배열생성

{

int i;

int \*\*Arr;

Arr= (int\*\*)malloc(sizeof(int\*) \* n); //2차원배열동적할당

for (i = 0; i < n; i++)

Arr[i] = (int\*)malloc(sizeof(int)\*n);

return Arr;

}

int minmult(int n, const int d[], int \*P[]) //minimum matrix multiplication

{

int i, j, k, diagonal;

int \*\*M, temp = 0, min\_k = 0;

M = make\_arr(n + 1);

for(i = 1; i <= n; i++)

M[i][i] = 0;

for (diagonal = 1; diagonal <= n - 1; diagonal++)

for (i = 1; i <= n - diagonal; i++){

j = i + diagonal;

for (k = i; k <= j - 1; k++)

{

M[i][j] = M[i][k] + M[k + 1][j] + d[i - 1] \* d[k] \* d[j];

if (i == k)

{

temp = M[i][j];

min\_k = k;

}

else if (M[i][j] > temp)

M[i][j] = temp;

else

min\_k = k;

}

P[i][j] = min\_k;

}

printf("\n [M]");

print\_map(M, n);

return M[1][n];

}

void order(int i, int j, int \*P[]) // 최적의 순서 출력함수

{

int k;

if (i == j)

printf("A[%d]", i);

else

{

k = P[i][j];

printf("(");

order(i, k, P);

order(k + 1, j, P);

printf(")");

}

}

void print\_map(int \*map[], int n) //map 출력을위한함수

{

int i, j, i\_cnt = 1, j\_cnt = 1;

for (i = 0; i <= n; i++)

{

for (j = 0; j <= n; j++)

{

printf(" ");

if (i == 0)

{

if (i\_cnt == 1)

printf(" ");

if (i\_cnt == n + 1)

continue;

printf("[%d] ", i\_cnt++);

}

else if (j == 0)

printf("[%d]", j\_cnt++);

else

if (i < j)

printf("%3d ", map[i][j]);

else if (i >= j && map[i][j] < 0)

printf("%3c ", ' ');

else

printf("%3d ", 0);

}

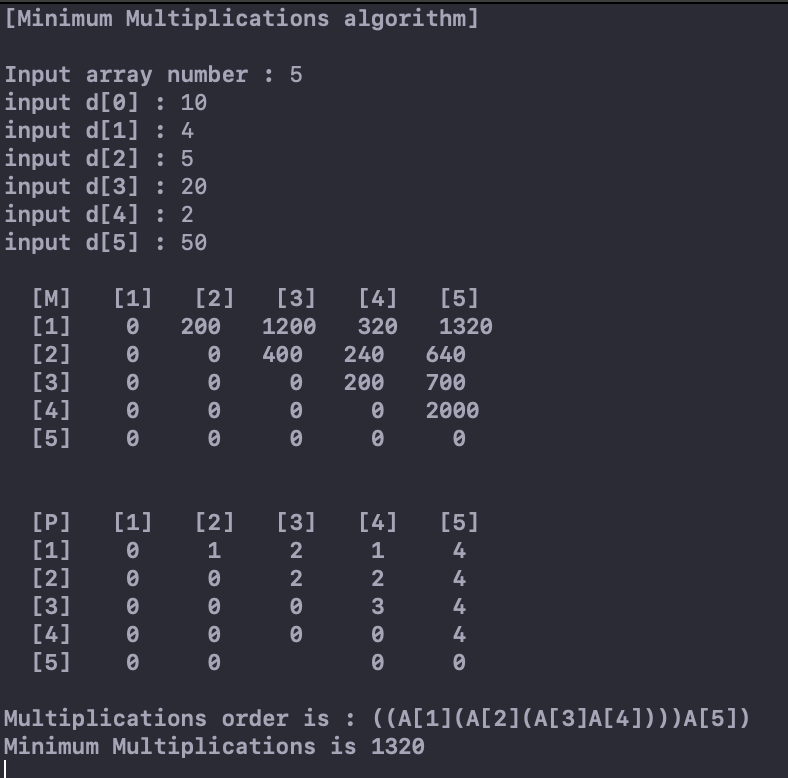
printf("\n");

}

printf("\n");

}

실행화면



24) 주어진 6개의 키값의 최적 이진트리를 작성하시오.

Optimal Binary Search Tree 소스코드

//

// main.c

// algo\_6\_OBST

//

// Created by James Choi on 12/10/2019.

// Copyright © 2019 James Choi. All rights reserved.

//

#include <stdio.h>

#include <string.h>

#define MAX 100 // 최대 아이템의 수

// 아이템을저장할구조체

typedef struct {

char key[10];

float p;

} element;

// 트리를만들때노드를구성할구조체

typedef struct \_nodetype {

char key[10];

struct \_nodetype \*left;

struct \_nodetype \*right;

}nodetype;

// 노드구조체포인터

typedef nodetype\* node\_pointer;

float minimum(float a[MAX + 2][MAX + 1], int i, int j, int \*k); // 가장적은값을리턴하는함수

float sigma(int i, int j); // m=i ~ m=j 까지합을구하여값을리턴하는함수

node\_pointer tree\_node(int i, int j); // 최적이진트리를구축하는함수

int n; // 아이템의개수

element key[MAX + 1]; // 입력받을키구조체

int r[MAX + 2][MAX + 1]; // 배열R이저장될 배열

node\_pointer node\_root; // 트리의루트를가리키는구조체포인터변수

float minavg; // 최소 평균 검색 시간을 저장할 변수

// 최적이진트리검색함수

void search()

{

int found;

int result\_compare;

char search\_word[10];

node\_pointer p;

// 검색할키를입력받음

printf("Input Search Key :");

scanf("%s", search\_word);

// 루트 노드의 포인터 복사

p = node\_root;

found = 0;

while (1) {

// key 가 NULL 이면 while 종료 (찾지 못한 경우)

if (p->key == NULL)

break;

// 검색 할 키와 저장된 키 비교

result\_compare = strcmp(search\_word, p->key);

// 일치 하면 found=1 저장 후 while 문 종료

if (result\_compare == 0) {

found = 1;

break;

}

else if (result\_compare < 0)

p = p->left; // 왼쪽 트리 검색

else

p = p->right; // 오른쪽 트리 검색

}

if (found)

printf("Found : %s\n", p->key); // 찾았을 때 출력

else

printf("Not found\n"); // 찾지 못했을 때 출력

}

// 최적 이진 트리 구축을 시작하는 함수

void make\_tree()

{

node\_root = tree\_node(1, n); // root 노드의 포인터 저장

}

// 최적 이진 탐색 트리 구축 하는 함수 (재귀 호출)

node\_pointer tree\_node(int i, int j)

{

int k;

node\_pointer a;

a = (node\_pointer)malloc(sizeof(node\_pointer));

k = r[i][j];

if (k == 0)

return NULL;

else {

strcpy(a->key, key[k].key); // key 값을 노드로 복사

a->left = tree\_node(i, k - 1); // 왼쪽 서브 트리 구축

a->right = tree\_node(k + 1, j); // 오른쪽 서브 트리 구축

return a; // 노드의 포인터 반환

}

}

// 배열 R 출력

void print\_r()

{

int i, j;

printf("\n배열 R 출력\n");

for (i = 1; i <= n + 1; i++) {

for (j = 0; j <= n; j++) {

printf("[%2d ]\t", r[i][j]);

}

printf("\n");

}

printf("\n");

}

// i <= k <= j 일때 a[i][k-1] + a[k+1][j] 값 중 최소값 반환

float minimum(float a[MAX + 2][MAX + 1], int i, int j, int \*k)

{

int tmp\_k;

float tmp\_value;

float tmp\_value2;

tmp\_k = i;

// k=i 일때 계산

tmp\_value2 = a[i][tmp\_k - 1] + a[tmp\_k + 1][j];

\*k = tmp\_k;

for (tmp\_k = i + 1; tmp\_k <= j; tmp\_k++) {

tmp\_value = a[i][tmp\_k - 1] + a[tmp\_k + 1][j];

if (tmp\_value < tmp\_value2) { // 최소값인지 검사후 최소값 저장

tmp\_value2 = tmp\_value;

\*k = tmp\_k; // 최소값일때의 k 값 저장

}

}

return tmp\_value2;

}

// m=i 에서 m=j 까지의 Pm 의 합 구하여 반환

float sum(int i, int j){

float tmp;

int m;

tmp = 0;

for (m = i; m <= j; m++)

tmp += key[m].p;

return tmp;

}

// 키와 각 키에 할당될 확률 입력 받음

void init\_element()

{

int i, j;

printf("아이템의 개수를 입력하시오 : ");

scanf("%d", &n);

for (i = 1; i <= n; i++) {

printf("키 입력 (%d) >> ", i);

scanf("%s",key[i].key);

printf("확률 입력 (%d) >> ",i);

scanf("%f", &key[i].p);

}

// 배열 r 의 값을 초기화 (가비지 값 없애기 위함)

for (i = 0; i <= n + 1; i++)

for (j = 0; j <= n; j++)

r[i][j] = 0;

}

// 최적 이진 검색 트리 구축을 위한 r 배열 구성

void optsearchtree()

{

int i, j, k, diagonal;

float a[MAX + 2][MAX + 1] ={0,};

// a 배열과 r 배열에 초기값 입력

for (i = 1; i <= n; i++) {

a[i][i - 1] = 0;

a[i][i] = key[i].p;

r[i][i] = i;

r[i][i - 1] = 0;

}

a[n + 1][n] = 0;

r[n + 1][n] = 0;

for (diagonal = 1; diagonal <= n - 1; diagonal++)

for (i = 1; i <= n - diagonal; i++) {

j = i + diagonal;

a[i][j] = minimum(a, i, j, &k) + sum(i, j); // 일반식 구현

r[i][j] = k; // 최소 값을 주는 k 값 저장

}

minavg = a[1][n];

printf("\n배열 A 출력\n");

for (i = 1; i <= n ; i++) {

for (j = 0; j <= n; j++) {

printf("[%.2lf]\t", a[i][j]);

}

printf("\n");

}

}

// 메인

int main(void)

{

init\_element(); // 초기화

optsearchtree(); // a배열과 r배열구성

print\_r(); // r배열출력

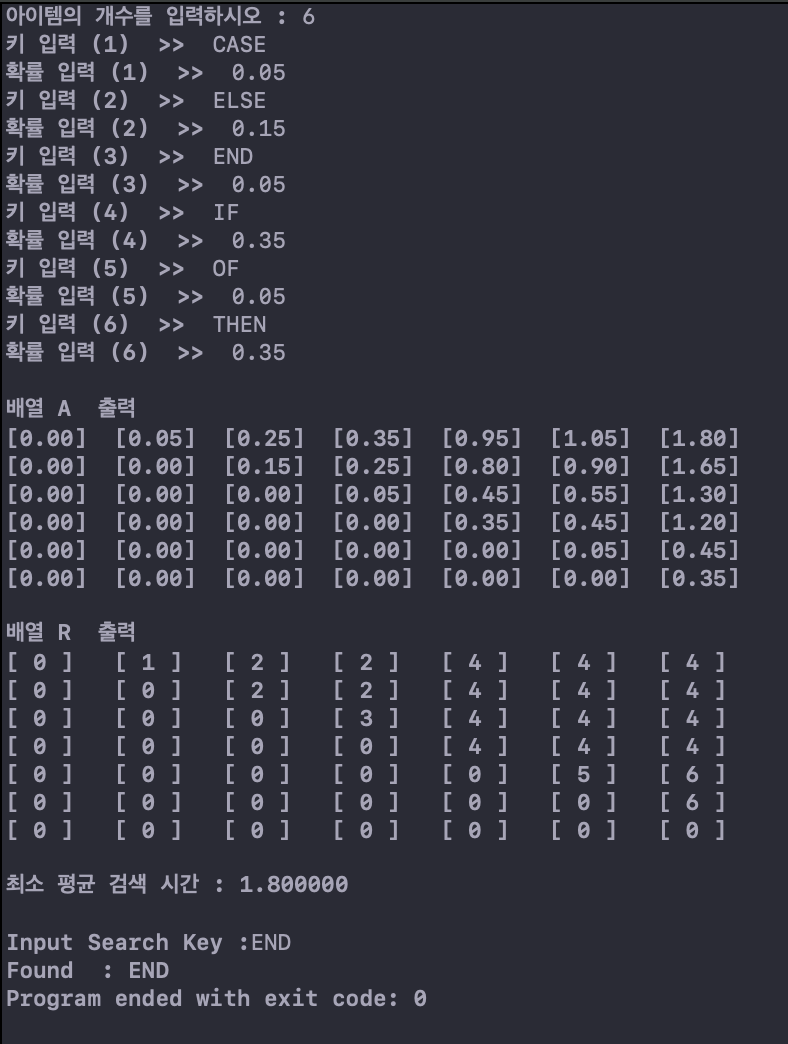
printf("최소 평균 검색 시간 : %f\n\n",minavg); // 최소평균검색시간출력

make\_tree(); // 최적이진탐색트리구축

search(); // 검색

}

실행화면



최적 이진 트리는 수기로 작성하였습니다.