|  |  |  |
| --- | --- | --- |
| **Algorithms analysis** | Section | 02 |
| Student number | 21900371 |
| **Homework 2 – MCM by DP** | Name | Seo, Hyo Gyeong |

*If your explanation is less informative and insufficient, then you may not get any points.*

*Also, you should provide discussion, otherwise you will get penalty.*

**□ Explanation of MCM algorithm.**

The number of multiplications depends on the order in which they are multiplied. To find the optimal solution, we divide it into smaller problems and combines the optimal solution of the smaller problem.

1. Obtain an optimal number of times multiplied to calculate the …matrix.  
The matrix is matrix

2. Divide the matrix into partial problems, calculate the multiplication number of partial problems in recursive form, and combine them.  
The number of multiplications required to calculate … is called arr[i,j].

If i and j are equal, arr[i,j] is zero.  
If i is less than j, arr[i,j] is the minimum value of arr[i,k]+arr[k+1,j]+p[i]\*p[k+1]\*p[j+1] for i<=k<j.

The minimum value of k is stored in small[i][j].

3. After completing the calculation with the recursive function of 2, the array small will have a value of k that divides the matrix in an optimal manner.

(You can print the optimal solution using the recursive function.)

In my code, I implemented mcm algorithm in below function.

void mcm(vector<int> \*p,int n, int (\*arr)[MAX\_LEN],int (\*small)[MAX\_LEN]);

1. Initialize array by 0 where i and j is equal.
2. Calculate the optimal solution in subsequnces where i<j.

* save k (index of spliting subsequences optimally) in small[i][j]
* save minimum cost in arr[i][j]

Through below function, we can print the optimal solution.

void print\_optimal(int (\*small)[MAX\_LEN],int i,int j);

In this function, I used recursive.

when value of two parameter(i,j) is same, this function print the matrix name.

For visibility, I write as A[i].

**□ More than 10 MCM Problems and solutions**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 1 | 20X10 | 10X50 | 50X5 | 5X30 | - | - |
| 2 | 10X5 | 5X40 | 40X10 | - | - | - |
| 3 | 20X30 | 30X40 | 40X10 | 10X20 | - | - |
| 4 | 15X5 | 5X20 | 20X10 | - | - | - |
| 5 | 5X10 | 10X15 | 15X10 | 10X5 | 5X5 | - |
| 6 | 10X10 | 10X20 | 20X15 | - | - | - |
| 7 | 20X30 | 30X20 | 20X20 | 20X10 | - | - |
| 8 | 5X10 | 10X30 | 30X10 | - | - | - |
| 9 | 20X10 | 10X30 | 30X20 | 20X10 | - | - |
| 10 | 15X30 | 30X20 | 20X40 | 40X10 | 10X20 | - |

**Solution**

(1) ((A[1](A[2]A[3]))A[4])

value : 6500

(2) (A[1](A[2]A[3]))

value : 2500

(3) ((A[1](A[2]A[3]))A[4])

value : 22000

(4) (A[1](A[2]A[3]))

value : 1750

(5) ((A[1](A[2](A[3]A[4])))A[5])

value : 1875

(6) (A[1](A[2]A[3]))

value : 4500

(7) (A[1](A[2](A[3]A[4])))

value : 16000

(8) ((A[1]A[2])A[3])

value : 3000

(9) (A[1]((A[2]A[3])A[4]))

value : 10000

(10) ((A[1](A[2](A[3]A[4])))A[5])

value : 21500

**□ Screenshots of your program running**

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**□ Discussion about the results**

// you can compare the solution with non-optimal solutions, provide the running time of the solution and non-solutions, any possible way to improve the algorithm, what you feel, what you learn from the simulation.

1. The simplest way to solve this problem is to consider and choose every possible case. However, when n matrices are multiplied, the time-complexity is Ω(2^n). Moreover, the greater the n, the greater the time it takes. However, when using algorithm that I implemented, time complexity is Ο(n^3) because it divides the problem into partial problems and combines it using recursive expressions. Therefore, it is more efficient to use this algorithm.

2. To explain more, I will calculate the third problem (: 2030, : 3040, : 4010, : 1020).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 24000 | 18000 | 22000 |
| 2 | - | 0 | 12000 | 18000 |
| 3 | - | - | 0 | 8000 |
| 4 | - | - | - | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 1 | 1 | 3 |
| 2 | - | 0 | 2 | 3 |
| 3 | - | - | 0 | 3 |
| 4 | - | - | - | 0 |

arr s

According to this table, the optimal solution is ((()) ) and optimal value is 22000.We also can find max value by changing max to min and < to >, then the max value of this problem is 48000 which is about 2 times bigger than optimal solution. Thus it is important to find optimal way to multiplication. And when this algorithm calculates subsequences to find optimal solution, it uses calculated value from smaller subsequence so it can reduce time.

3. I created optional function just for my study [void generate\_problem(vector<int> \*p);].

This function creates random problems and when I using this function, I realized that when n is 20, the time cost when using brute force algorithm is about 130 times greater than using algorithm that I implemented. And this difference will be greater as the value of n increases. So I thought that when making program it is very important to consider the time complexity of algorithm.

**□ Codes** // you should also submit the executable C or C++ files.

#include <iostream>

#include <cstdlib>

#include <limits>

#include <vector>

#define MAX\_LEN 128

using namespace std;

void load\_problem(vector<int> \*p,int t); //load 10 problem

void generate\_problem(vector<int> \*p); //make random problem (optional function)

void mcm(vector<int> \*p,int n, int (\*arr)[MAX\_LEN],int (\*small)[MAX\_LEN]); //mcm algorithm

void print\_optimal(int (\*small)[MAX\_LEN],int i,int j); //print optimal solution

int main(){

vector<int> p;

int arr[MAX\_LEN][MAX\_LEN];

int small[MAX\_LEN][MAX\_LEN];

int n;

for(int t = 0;t<10;t++){

printf("Test : %d\n\n",t+1); //문제 번호 출력

//generate\_problem(&p); //optional function

load\_problem(&p,t); //해당 문제 vector에 저장

for(int s=0;s<p.size()-2;s++){ //문제 행렬 출력

if(p.at(s+2)!=0){

printf("A[%d] : %dX%d ",s+1,p.at(s+1),p.at(s+2));

}

}

printf("\n\n");

n = p.size()-2;

mcm(&p,n,arr,small); //mcm algorithm

print\_optimal(small, 1, n); //optimal solution 출력

printf("\nvalue : %d" ,arr[1][int(p.size()-2)]);

p.clear(); //vector 초기화

printf("\n\n");

}

}

void generate\_problem(vector<int> \*p){

/\* optional function just for my study \*/

int len = rand()%10+20;

printf("len: %d\n",len);

p->push\_back(0);

for(int i=1;i<=len;i++){

p->push\_back(rand()%50);

printf("p %d\n",p->at(i));

}

}

void load\_problem(vector<int> \*p,int t){

int problem[10][10] = {

{20,10,50,5,30,0,0,0,0,0},

{10,5,40,10,0,0,0,0,0,0},

{20,30,40,10,20,0,0,0,0,0},

{15,5,20,10,0,0,0,0,0,0},

{5,10,15,10,5,5,0,0,0,0},

{10,10,20,15,0,0,0,0,0,0},

{20,30,20,20,10,0,0,0,0,0},

{5,10,30,10,0,0,0,0,0,0},

{20,10,30,20,10,0,0,0,0,0},

{15,30,20,40,10,20,0,0,0,0}

}; //problem set

p->push\_back(0);

for(int i = 0; i<10;i++){

if(problem[t][i]!=0){

p->push\_back(problem[t][i]);

}

}

}

void mcm(vector<int> \*p, int n, int (\*arr)[MAX\_LEN],int (\*small)[MAX\_LEN]){

int max = numeric\_limits<int>::max();

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

arr[i][i] = 0;

}

for(int r=2;r<=n;r++){

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

int j = i+r-1;

arr[i][j] = max;

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

int q = arr[i][k]+arr[k+1][j]+(p->at(i)\*p->at(k+1)\*p->at(j+1));

if(q<arr[i][j]){

arr[i][j] = q;

small[i][j] = k;

}

}

//printf("%d,%d,%d\t",i,j,arr[i][j]);

}

}

}

void print\_optimal(int (\*small)[MAX\_LEN],int i,int j){

if(i==j){

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

}else{

printf("(");

print\_optimal(small, i, small[i][j]);

print\_optimal(small, small[i][j]+1, j);

printf(")");

}

}