

1. PT: Minimal Triangulation

Variable:

int num_vertex : input file에 저장된 vertex의 개수

pos vertex[MAX_VERTEX] : input file의 polygon 정보를 저장하는 변수(x, y를 가짐)

int pair_cnt : 답으로 찾은 pair의 개수

pair ans[MAX_VERTEX] : 삼각화의 결과로 찾은 꼭짓점의 정보를 저장

double table[MAX_VERTEX][MAX_VERTEX] : 삼각화 후 그은 선분들의 합의 최소를 저장하는 테이블

int k_table[MAX_VERTEX][MAX_VERTEX] : pair를 찾기 위해 필요한 테이블

Function:

double triangulation(); //삼각화 하는 함수

void getinfo_polygon(char* input); //파일에서 polygon의 좌표를 가져오는 함수

void write_output(char* output, double result); //결과를 output 파일에 저장

void sorting(); //pair를 ascending order로 저장하는 함수

double distance(int v1, int v2); //Euclidean distance를 구하는 함수

void find_pairs(int f, int s); //답이 되는 꼭짓점의 pair를 구하는 함수

void clear(); //각 testcase를 시행하기 이전에 global 변수들을 초기화하는 함수

1. double triangulation()

For s = 0 to num_vertex //table의 row (s = 시작 vertex)

For l = 0 to num_vertex //table의 col (i = 해당 polygon의 vertex 개수)

If s < 4, table[s][i] = 0 //테이블의 base case

Else

For k = 1 to s - 2 (k = 고른 vertex)

Result = table[k+1][i] + table[s-k][i+k] +

처음-k번째 노드 길이 + k번째-마지막 노드 길이~

If(result < min)

Min을 result로 갱신, k_table[s][i] = k

For loop이 끝나고 해당 table[s][i]에 min을 넣는다

전체 시행이 끝나고 table[num_vertex][0]에 삼각화 진행 후 num_vertex - 3의 chord의 길이를 더한 값을 반환한다

2. void find_pairs (int i, int s)

If (s < 4) return // 노드의 개수가 3개 이하이면 삼각화가 필요 없어 pari도 없다

k = k_table[s][i] // 삼각화 할 때 고른 vertex를 k로 가져오고

if(k > 1) // 매 실행마다 선은 무조건 1개씩 최소로 발생하니

i(첫번째 노드)와 i+k(k번째 노드)를 ans에 포함한다

if(s - k > 2) // 고른 k 번째 노드가 처음과 마지막과 인접하지 않으면

$i+k$ (k 번째 노드)와 $i+s-1$ (마지막 노드)를 ans에 포함한다

(인접하지 않는 노드는 선이 총 2개가 그어지므로 pair가 한 개 더 발생)

Find_pairs($i, k+1$) //나누어진 두 덩어리의 polygon에 대해서도 동일한 작업 반복

Find_pairs($i+k, s - k$)

3. main

“PT_test_command.txt”를 열어서 testcase를 받아온다

테스트 케이스만큼 for loop을 돈다

Input 파일과 output 파일의 이름을 받아온다

clear함수로 초기화

getinfo_polygon으로 input파일에서 vertex의 정보를 받아온다

result = triangulation // 삼각화를 통해 구한 $n-3$ 개의 chord 길이의 총합

find_pairs 함수로 사용된 vertex의 pair들을 구한다

sorting 함수로 ascending order로 pair를 정렬한다

write_output함수로 output 파일에 결과를 저장한다

<Test Examples>

PT_input_1 - Windows 메모장	PT_output_1 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)	파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
8	2261.347731
800.000000 350.000000	0 6
653.936462 607.494812	1 4
342.910278 605.583313	1 6
280.965546 554.997314	2 4
209.068207 276.796906	4 6
303.075836 123.679718	
563.235107 56.740177	
751.477051 186.417358	

PT_input_2 - Windows 메모장	PT_output_2 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)	파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
11	1968.421272
780.000000 392.000000	0 2
759.000000 522.000000	0 9
660.000000 601.000000	2 4
551.000000 649.000000	2 6
456.000000 602.000000	2 9
369.000000 523.000000	4 6
386.000000 395.000000	6 8
432.000000 322.000000	6 9
521.000000 273.000000	
645.000000 280.000000	
732.000000 330.000000	

1. SS: Subset Sum

Variable:

int num_element : 집합의 원소의 개수

int L : 찾고자 하는 Subset Sum

int element[MAX_ELEMENT] : 집합의 원소를 저장

int ans[MAX_ELEMENT] : Subset Sum을 만족하는 원소들의 집합

int num_ans : Subset Sum을 만족하는 원소들의 집합의 개수

int F[MAX_ELEMENT][MAX_ELEMENT] : DP를 구현하기 위한 테이블

Function:

void getinfo_element(char* input) : input 파일에서 집합의 원소를 가져오는 함수

void write_output(char* output, int result) : 결과를 output 파일에 저장하는 함수

void sorting() : subset sum에 포함되는 원소들의 index를 ascending order로 정렬

int subset_sum() : subset sum이 존재하는지 구하는 함수

void get_ans() : subset sum에 포함되는 원소들을 찾는 함수

void clear() : 각 testcase 이전 global 변수들을 초기화하는 함수

1. int subset_sum

F[0][0] = TRUE // table의 base case 1

For j = 1 to L // j = 남은 길이

F[0][j] = FALSE // table의 base case 2

For i = 1 to num_element // i = i번째 원소

For j = 0 to L

F[i][j] = F[i-1][j] // i번째 원소를 포함하지 않는 경우로 일단 세팅

If (j - element[i] >= 0) // i번째 원소를 포함할만큼 j가 남았으면

F[i][j] = F[i][j] || F[i-1][j-element[i]]

(or 앞부분: F[i][j] - i번째 원소를 사용하지 않은 경우)

(or 뒷부분: F[i-1][j-element[i]] - i번째 원소를 사용한 경우)

2. void get_ans

j = L로 초기화 // j는 남은 길이, L은 원하는 subset sum

for i = num_element to 1 // i는 i번째 원소

if F[i-1][j] == TRUE // or의 앞부분으로 F[i][j]가 결정되었다면

do nothing // 해당 i번째 원소는 선택된 것이 아니므로

else // or의 뒷부분으로 F[i][j]가 결정되었다면

ans[num_ans++] = i - 1 // i번째 원소가 선택된 것이니 답에 추가

(단, i는 1~n인데 답은 0~n-1을 요구하니 1을 빼고 저장)

j = j - element[i] // F[i][j]를 결정한 i-1번째 F로 가도록 j 값을 갱신

3. main

“SS_test_command.txt”를 열어서 testcase를 받아온다

테스트 케이스만큼 for loop을 돈다

Input 파일과 output 파일의 이름을 받아온다

clear함수로 초기화

getinfo_element으로 input파일에서 집합의 원소 정보를 받아온다

result = subset_sum // subset sum이 있으면 TRUE, 아니면 FALSE

if result == TRUE

get_ans함수로 해당하는 원소들의 값을 ans에 저장

sorting함수로 ans의 원소들을 ascending order로 저장

write_output함수로 결과를 output 파일에 저장

<Test Examples>

SS_input_1	SS_output_1	SS_input_2	SS_output_2 - 1
파일(F) 편집(E)	파일(F) 편집(E)	파일(F) 편집(E)	파일(F) 편집(E)
9	1	7	1
41	6	1	5
34	2	2	1
21	4	2	2
20	5	4	3
8	6	5	4
7	7	2	5
7	8	4	
4		15	
3			
50			