**Team22 Phase 1**

2014314592 서주원

2017312379 주재현

2017314875 조규원

2017310323 박상준

**1. Algorithm Adapte**

At first, our team checked the number of input cases.  
 Because the number of input cases is only 7, we adopted the **Brute-Force** algorithm. We calculated the distance of all routes.

Therefore, we use **Branch-Bound** algorithm that while we are calculating individual routes, if there exists shorter route that we already saved, our program will quit the job and calculate the other routes.

e.g) 1->2->3->4->5 distance : 18

1->2->4 distance : 23 ( longer than saved route. Quit the job.)

**2. Flow Chart**

Attached at different sheet.

Helped bycode2flow

**3. Source Code**

**1) Target problem 1**

#include <stdio.h>

// function to get root of number

float sqrt(int input) {

float x = 1;

int i = 0;

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

x = (x + (input / x)) / 2;

}

return x;

}

// function to get length of cities

void getPathLengthData(float arr[7][7], int dot[7][2]) {

int i, j;

int x1, y1;

int x2, y2;

int x\_length, y\_length, temp;

float res;

for (i = 0; i < 7; i++) {

for (j = 0; j < 7; j++) {

x1 = dot[i][0];

y1 = dot[i][1];

x2 = dot[j][0];

y2 = dot[j][1];

x\_length = x2 - x1;

y\_length = y2 - y1;

temp = (x\_length \* x\_length) + (y\_length \* y\_length);

res = sqrt(temp);

arr[i][j] = res;

}

}

}

int main()

{

float arr[7][7];

int dot[7][2] = {0, 0, 2, 6, 8, 4, 7, 2, 1, 6, 4, 9, 3, 2};

double dist\_min=99999, dist=0;

int course[6];

getPathLengthData(arr, dot);

//depth1

for (int d1 = 1; d1 < 7; d1++)

{

dist = arr[0][d1];

if (dist > dist\_min)

continue;

//depth2

for (int d2 = 1; d2 < 7; d2++)

{

if (d2 == d1)

continue;

dist = arr[0][d1] + arr[d1][d2];

if (dist > dist\_min)

continue;

//depth3

for (int d3 = 1; d3 < 7; d3++)

{

if (d3 == d2 || d3 == d1)

continue;

dist = arr[0][d1] + arr[d1][d2] + arr[d2][d3];

if (dist > dist\_min)

continue;

//depth4

for (int d4 = 1; d4 < 7; d4++)

{

if (d4 == d3 || d4 == d2 || d4 == d1)

continue;

dist = arr[0][d1] + arr[d1][d2] + arr[d2][d3] + arr[d3][d4];

if (dist > dist\_min)

continue;

//depth5

for (int d5 = 1; d5 < 7; d5++)

{

if (d5 == d4 || d5 == d3 || d5 == d2 || d5 == d1)

continue;

dist = arr[0][d1] + arr[d1][d2] + arr[d2][d3] + arr[d3][d4] + arr[d4][d5];

if (dist > dist\_min)

continue;

//depth6

for (int d6 = 1; d6 < 7; d6++)

{

if (d6 == d5 || d6 == d4 || d6 == d3 || d6 == d2 || d6 == d1)

continue;

dist = arr[0][d1] + arr[d1][d2] + arr[d2][d3] + arr[d3][d4] + arr[d4][d5] + arr[d5][d6] + arr[d6][0];

if (dist < dist\_min)

{

course[0] = d1;

course[1] = d2;

course[2] = d3;

course[3] = d4;

course[4] = d5;

course[5] = d6;

dist\_min = dist;

}

}

}

}

}

}

}

printf("1 %d %d %d %d %d %d 1\n%lf\n", course[0]+1, course[1]+1, course[2]+1, course[3]+1, course[4]+1, course[5]+1, dist\_min);

}

**2) Target problem 2**

#include <stdio.h>

float sqrt(int input) {

float x = 1;

int i = 0;

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

x = (x + (input / x)) / 2;

}

return x;

}//to get route of the number

void getPathLengthData(float arr[8][8], int dot[8][2]) {

int i, j;

int x1, y1;

int x2, y2;

int x\_length, y\_length, temp;

float res;

//

for (i = 1; i < 8; i++) {

for (j = 1; j < 8; j++) {

//the information of point1

x1 = dot[i][0];

y1 = dot[i][1];

//the information of point2

x2 = dot[j][0];

y2 = dot[j][1];

x\_length = x2 - x1;

y\_length = y2 - y1;

temp = (x\_length \* x\_length) + (y\_length \* y\_length);

res = sqrt(temp);

arr[i][j] = res; //save the distance

}

}

}

int main(){

int dot[8][2] = {0, 0, 0, 0, 2, 6, 8, 4, 7, 2, 1, 6, 4, 9, 3, 2};

float arr[8][8] = {0, };

getPathLengthData(arr, dot);

float dist = 0;

float min = 99999;

int r[7] = {1, 0, 0, 0, 0, 0, 7}; //route

int save\_route[7] = {0, };

//depth 1

for (int a = 2; a <= 6; a++){

dist = 0;

r[1] = a;

dist = arr[r[0]][r[1]];

//depth 2;

for (int b = 2; b <= 6; b++){

r[2] = b ;

if (a == b){

continue;

}

dist = arr[r[0]][r[1]] + arr[r[1]][r[2]];

if (dist > min) continue;

//depth 3

for (int c = 2; c <= 6; c++){

r[3] = c;

if (a == c || b == c){

continue;

}

dist = arr[r[0]][r[1]] + arr[r[1]][r[2]] + arr[r[2]][r[3]];

if (dist > min) continue;

//depth 4

for (int d = 2; d <= 6; d++){

r[4] = d;

if (a == d || b == d || c == d){

continue;

}

dist = arr[r[0]][r[1]] + arr[r[1]][r[2]] + arr[r[2]][r[3]] + arr[r[3]][r[4]];

if (dist > min) continue;

//depth 5

for (int e = 2; e <= 6; e++){

r[5] = e;

if (a == e || b == e || c == e|| d == e){

continue;

}

dist = arr[r[0]][r[1]] + arr[r[1]][r[2]] + arr[r[2]][r[3]] + arr[r[3]][r[4]] + arr[r[4]][r[5]] + arr[r[5]][r[6]];

if (dist < min){

for (int i = 0; i < 7; i++){

save\_route[i] = r[i];

min = dist;

}

}

}

}

}

}

}

for (int i = 0; i < 7; i++) {

printf("%d ",save\_route[i]);

}

printf("\n%f", min);

// print the result

}

**4. Output**

**1) Target problem 1**

path : 1 7 4 3 6 2 5 1

distance : 26.933056

**2) Target problem 2**

path : 1 5 2 6 3 4 7

distance : 23.327507