# Project

**- Phase1 -**

**(Computer Architecture)**

**(Modified)**

Team A

2013311659 곽창근

2013312608 문성암

## -Index-

## - Phase #1

We select ‘C language’ as High level language.

## 1. Adobted Algorithm

We have adobted ‘Combinatorial Search’ as an algorithm for this problem.

Combinatorial search is an algorithm that looks for answers while searching for a finite size of search space, including full search. And, it is goal to prevent to explore answers that are not likely to be optimized.

We used ‘Pruning’ as a combination exploration optimization technique.

The details of pruning are as follows.

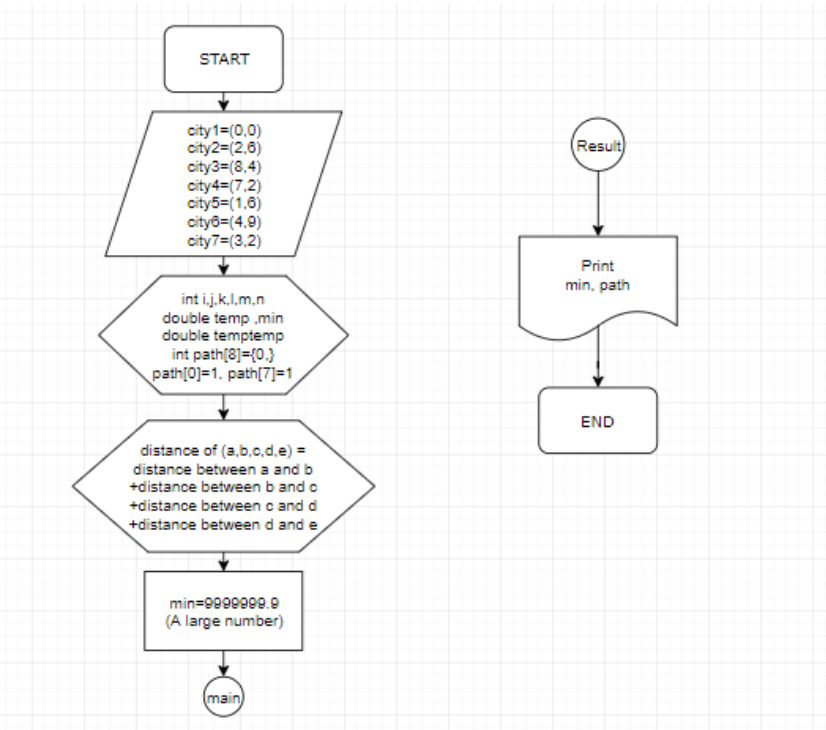
min is defined as the minimum value of solutions so far obtained.

If the value is larger than min while turning the loop, prevent the value from next step.

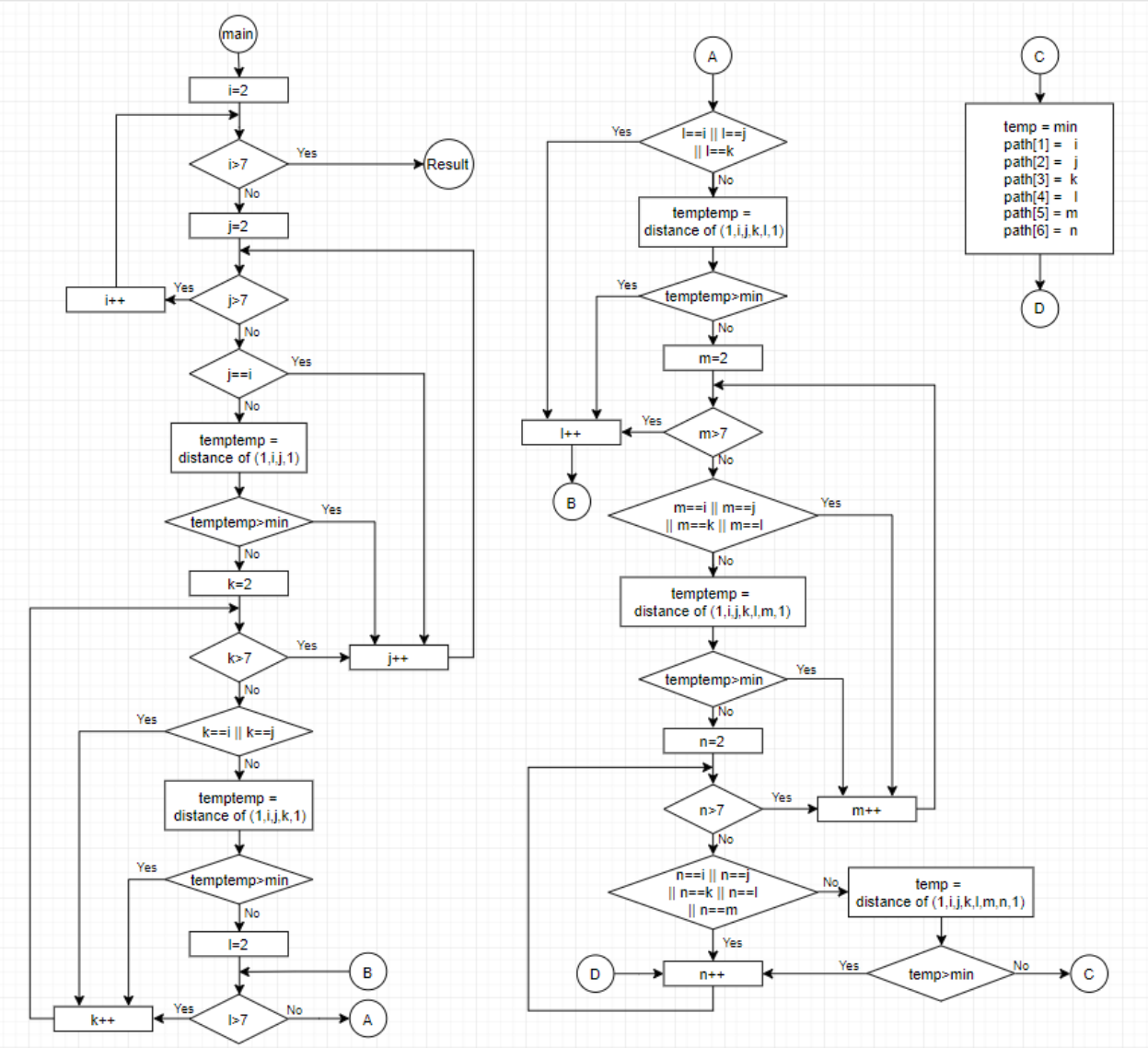
For example, if distance of 1-2-3-4-1 is larger than min, wherever 5, 6, or 7 are placed in 1-2-3-4-x-x-x-1 (x is number of city), distance of 1-2-3-4-x-x-x-1 is bound to be larger than the min. Therefore, we prevent this case in advance. (This is possible because the distances between cities are not negative because cities are implemented in coordinates.)

## 2. Flow Chart

-Start and End Flow Chart (except Main)



-Main Flow Chart



## 3. Source Code

```
#include <stdio.h>
#include <math.h>

//City's coordinates
typedef struct {
	int x;
	int y;
} City;

//Calculate the distance between two cities.
float CalDistance(City city1, City city2);

int main(void) {
	City cities[8] = { {-1,-1},{0,0},{ 2,6 },{ 8,4 },{ 7,2 },{ 1,6 },{ 4,9 },{ 3,2 } }; //cities[i] is 'i'th city.
	int i, j, k, l, m, n;
	int result_path[8] = { 0, };
	float temp, temptemp, min = 1.7e+307;
	float distances[8][8]; // Calculate distance between cities and save.
	for (i = 1; i <= 7; i++)
		for (j = 1; j <= 7; j++)
			distances[i][j] = CalDistance(cities[i], cities[j]);

	//Core Algorithm
	result_path[0] = result_path[7] = 1; //It means tour starts from city1 and finish to city1.
	for (i = 2; i <= 7; i++) {
		for (j = 2; j <= 7; j++) {
			if (j == i )
				continue;
```

```
temptemp = distances[1][i] + distances[i][j] + distances[j][1];
if ( temptemp > min )
    continue;
for (k = 2; k <= 7; k++) {
    if (k == i || k == j )
        continue;
    temptemp = distances[1][i] + distances[i][j] + distances[j][k] + distances[k][1];
    if ( temptemp > min )
        continue;
    for (l = 2; l <= 7; l++) {
        if (l == i || l == j || l == k )
            continue;
        temptemp = distances[1][i] + distances[i][j] + distances[j][k]
            + distances[k][l] + distances[l][1];
        if ( temptemp > min )
            continue;
        for (m = 2; m <= 7; m++) {
            if (m == i || m == j || m == k || m == l )
                continue;
            temptemp = distances[1][i] + distances[i][j] + distances[j][k]
                + distances[k][l] + distances[l][m] + distances[m][1];
            if ( temptemp > min )
                continue;
            for (n = 2; n <= 7; n++) {
                if (n == i || n == j || n == k || n == l || n == m)
                    continue;
                temp = distances[1][i] + distances[i][j] + distances[j][k]
                    + distances[k][l] + distances[l][m] + distances[m][n]
                    + distances[n][1];
```

```
                                    if (min> temp) {
                                        min = temp;
                                        result_path[1] = i;
                                        result_path[2] = j;
                                        result_path[3] = k;
                                        result_path[4] = l;
                                        result_path[5] = m;
                                        result_path[6] = n;
                                    }
                                }
                            }
                        }
                    }
                }
        }
        //print results.
        printf("Path : ");
        for (i = 0; i<8; i++)
            printf("%d ", result_path[i]);
        putchar('\n');
        printf("Distance : %lf\n", min);
        return 0;
}
float CalDistance(City city1, City city2) {
        float dx = city1.x - city2.x;
        float dy = city1.y - city2.y;
        return sqrt(dx*dx + dy * dy);
}
```

## 4. Details of Source Code

```
typedef struct {
        int x;
        int y;
} City;
```

We declared the structure ‘City’ as above.

**float** CalDistance(City city1, City city2);

This function calculates the distance between city1 and city2.

In main,

City cities[8] = { {-1,-1}, {0,0}, {2,6}, {8,4}, {7,2}, {1,6}, {4,9}, {3,2} }; //cities[i] is ‘i’th city.

This code is for saving Cities’ info. At cities[0], we saved trash value to save ‘i’th city’s info at cities[i].

int result_path[8] = {0,};

We will save the tour path at this array.

**float** distances[8][8]; // Calculate distance between cities and save.

```
        for(i=1; i<=7; i++)
                for(j=1; j<=7; j++)
                        distances[i][j] = CalDistance(cities[i], cities[j]);
```

This code is for saving all distances between city to city. At distances[i][j] we saved the distance between City i and j.

And next, we started finding the shortest path by checking all possible number of cases through 6 times overlapped ‘for’ loops. The tour always start and finish at City1. So we just have to check 6 cities which is passed through the middle. At 6 times overlapped ‘for’ loops, variables ‘l’ to ‘n’ represent the order of passing through.

In each phase's 'for' loop, there are 2 kind of 'if' statements.

1) if( *condition* ) | 2) if( temptemp > min )

continue; | continue;

'1)' is for skip when the alphabets are same. Because we have to visit each city only once.

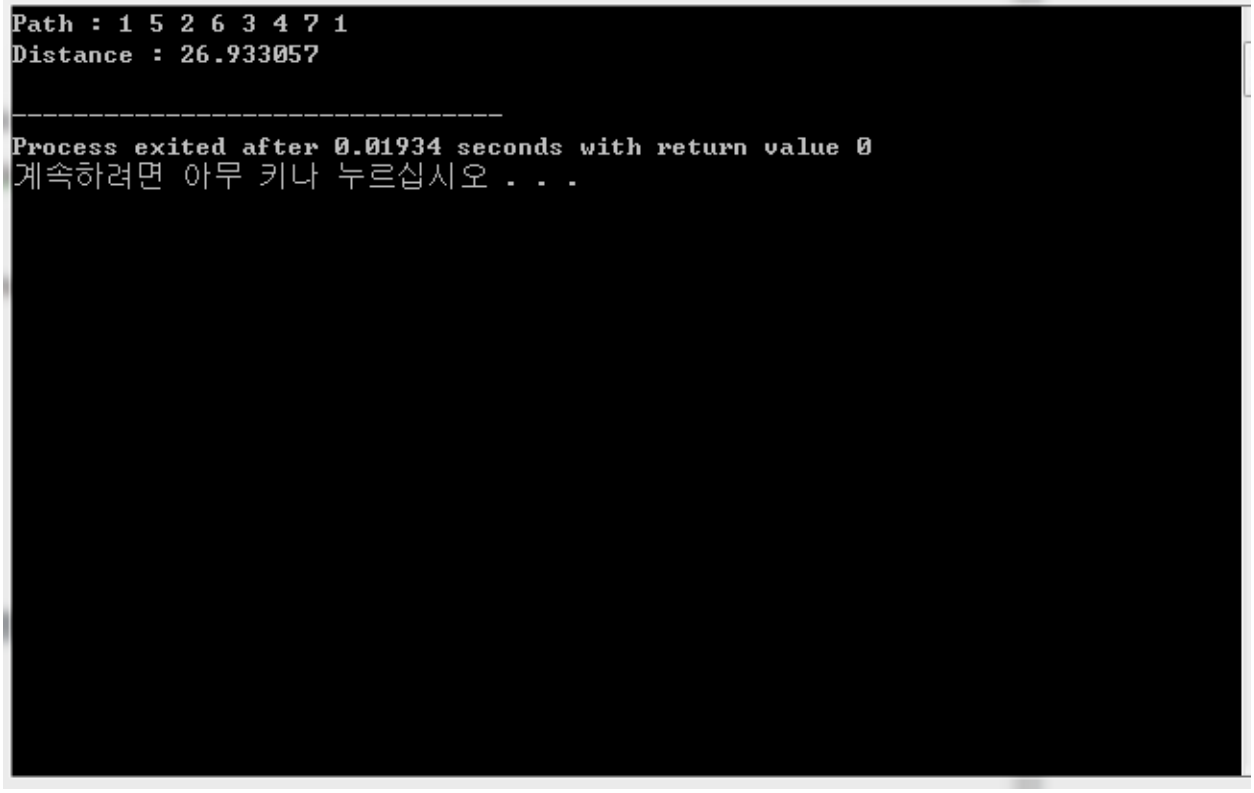
'2)' is for skip when the distance of the selected cities until now is shorter than min. At temptemp variable, we saved the distance of the selected cities until now.

At innermost 'for' loop,

temp = distances[1][i] + distances[i][j] + distances[j][k] + distances[k][l] + distances[l][m] + distances[m][n] + distances[n][1];

we calculate the distance while traveling. It is saved at temp, and compared with min. If this value is smaller than min, update min as this value, otherwise no change.

## 5. Output



## 6. Modification

We modify a few part of code. Because we not accustomed to translate double type data instructions for MIPS. So, we modify double to float. And, We made it identifiable by writing it in bold letters and underlines. In total, eight places were modified in the first phase report.