CSE2011 Problem Solving, 2016 Spring

2013312343 이상헌

**Project 2**

**1. 문제 이해**

(1) 문제

**Project 2: Points for soccer teams**

A soccer league has 20 teams, and each team plays 19 games—once with all other teams. Each team gets A points when it wins, B points when it ties, and Cpointswhen it is defeated. For given A, B and C, we want to know all thegame results that yield themaximum/minimum points ofa team with the Dthlargest scores.

<Input>

There are four integer values: A, B, C, and D.

A: the amount of points when a team wins(0<= A, B, C<=5)

B: the amount of points when a team ties(0<= A, B, C<=5)

C: the amount of points when a team loses (0 <= A, B, C<= 5)

(Note that there is no relationship among A, B and C. For example, it is possible that A < B and B < C).

D: target ranking (1~20)

E: 0 (minimum) or 1 (maximum).

For example, if the input is 3 1 0 5 0,

- Each team gets 3 points when it wins;

- Each teamgets 1 point when it draws; and

- Each team gest 0 point when it loses;

- The output shows all the game resultsthat yield the minimumpoints of a team with the 5thhighest points.

<Output>

There should be20 lines; each line show results of each team. For example, if the 3rdline is “WLXTTTTTTTTTTTTTTTTT”,

(2) 문제 정의

- 20개의 축구 팀이 리그전으로 각 팀마다 19경기를 진행한다. 승, 무, 패에 따른 점수는 input으로 주어지고, input으로 주어진 d번째 팀의 점수가 최대 혹은 최소가 되도록 하는 경기 결과 조합을 출력하라.

**2. 문제 해결**

*BackTracking.*

- Backtracking 알고리즘이란 모든 경우의 수를 효과적으로 고려하는 방법이다. 하나부터 시작하여 각 단계에서 포함될 수 있는 후보들을 선출하고, 각 후보에 대해 같은 방법을 재귀적으로 반복한다. 일련의 역할이 정해져 있는 함수들이 포함된 코드 툴을 통해 backtracking을 효과적으로 구현할 수 있다.

- 아이디어

각 팀의 리그전 경기 결과를 만들 때, 20\*20 행렬로 저장할 수 있다. 여기서 1팀과 2팀의 경기 결과에 대한 정보는 곧 2팀과 1팀의 경기 결과에 대한 정보로 이어질 수 있다. 예를 들어, 1팀이 2팀을 상대로 이긴 경우, 2팀은 1팀을 상대로 진 것이다. 또한 각 팀은 자기 팀과의 경기를 진행하지 않는다. 이와 같은 방법으로 경기 결과를 저장하는 20\*20 행렬에서, 대각선을 포함하지 않은, 반의 결과만 정하면 되기 때문에 총 190칸의 경기 결과가 나타난다.

190칸에 대해서 backtracking 기법을 사용하여 가능한 모든 경기 결과 조합을 살펴보고, 각 경우에 대해서 d번째 팀의 점수를 계산하여, 이전 경기 결과 조합에서 d번째 팀이 얻은 점수와 비교한다.

- 이점

모든 경기 결과 조합을 알아보는 것은, 가능한 모든 경우의 수를 고려하기 때문에 가장 정확한 답을 도출해낼 수 있다.

- 한계

본 아이디어는 모든 경기 결과 조합을 고려해야 한다. 위에서 언급한 바와 같이 20개의 팀의 경기 결과를 저장하기 위해서는 190칸의 경기 결과를 채워야 한다. 각 경기 결과는 승(‘W’), 무(‘T’), 패(‘L’)가 될 수 있기 때문에, 가능한 모든 경우의 수는 3^190 가지이다. 이 모든 경우의 수를 고려하고, 점수를 계산하는 것은 시간이 매우 오래 걸리며, 사실상 본 아이디어를 통해 결과를 도출하는 것은 불가능하다고 판단되었다.

- 보완점

backtracking 기법에 대한 코드를 작성한 후, 가능한 경우의 수가 매우 방대하기 때문에 결론 도출이 어렵다는 결론을 내었다. 이에 대한 보완점을 생각하던 중, 승, 무, 패에 따라 배분되는 점수의 크기의 대소에 따라 경우를 나눌 수 있다고 생각하였다. 각 승, 무, 패에 따라 배분되는 점수를 w\_s, t\_s, l\_s라고 가정하자.

1. w\_s > t\_s, l\_s > t\_s인 경우

승, 패에 배분되는 점수가 비긴 경기에 배분되는 점수보다 큰 경우, 손쉽게 결론을 도출할 수 있다. 만일 d번째 팀의 점수가 최대가 되도록 하는 것을 원한다면, 모든 경기를 W 혹은 L로 만들면 된다. 만일 d번째 팀의 점수가 최소가 되도록 하는 것을 원한다면, 모든 경기를 T로 만들면 된다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 팀1 | 팀2 | 팀3 | 팀4 |
| 팀1 | X | W | W | W |
| 팀2 | L | X | W | W |
| 팀3 | L | L | X | W |
| 팀4 | L | L | L | X |

표 . w\_s, l\_s > t\_s이고, d번째 팀의 점수가 최대가 되도록 하는 경우.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 팀1 | 팀2 | 팀3 | 팀4 |
| 팀1 | X | T | T | T |
| 팀2 | T | X | T | T |
| 팀3 | T | T | X | T |
| 팀4 | T | T | T | X |

표 2. w\_s, l\_s > t\_s이고, d번째 팀의 점수가 최소가 되도록 하는 경우.

1. w\_s < t\_s, l\_s < t\_s인 경우

승, 패에 배분되는 점수가 비긴 경기에 배분되는 점수보다 작은 경우에도 손쉽게 결론을 도출할 수 있다. 만일 d번째 팀의 점수가 최대가 되도록 하는 것을 원한다면, 모든 경기를 T로 만들면 된다. 만일 d번째 팀의 점수가 최소가 되도록 하는 것을 원한다면, 모든 경기를 W 혹은 L로 만들면 된다.

1. w\_s > t\_s, t\_s > l\_s인 경우 : 크기 비교.

승에 배분되는 점수가 비긴 경기에 배분되는 점수보다 크고, 패에 배분되는 점수가 비긴 경기에 배분되는 점수보다 작은 경우, 그 차이에 따라 다르게 경우가 나뉜다. 먼저 d팀 ~ 20팀의 모든 경기를 (W, L)로 맞춘다. 이후 1팀 ~ d팀의 모든 경기는 (W, T, L)의 조합 혹은 오로지 T만으로 경기 결과를 맞출 수 있다. 이 두 가지 경우에 대해서 d팀의 점수를 각각 계산하고, 더 큰 혹은 작은 경우를 결과로 도출한다.

(W, T, L)의 경우에는, 비어있는 칸이 각각에 WW…WTLL…L (e = 0) 혹은 LL…LTWW…W (e = 1) (e에 따라 다르다) 순서대로 채운다. W와 L의 개수는 비어있는 칸에 맞추고, 개수가 같다. 만일 비어있는 칸의 개수가 짝수인 경우, WW…WLL…L 혹은 LL…LWW…W (e에 따라 다르다) 을 순서대로 채운다.

예를 들어 팀의 개수가 4개, w\_s = 5, t\_s = 3, l\_s = 0, d = 3, e = 0인 input을 받은 경우, 채우는 순서는 표3과 같다. 먼저, d=3 이므로 3팀부터 4팀까지의 모든 경기 결과는 W 혹은 L로 채운다. 이는 파란색 부분이다. 이후 노란색 부분을 채우는데, 비어있는 칸이 2칸이므로 WL을 순서대로 채운다. 이는 행 방향으로 (1, 2), (1, 3), (2, 3), (2, 1), (3, 1), (3, 2) 순서로 채운다. 이와 같은 경기 결과로 d팀, 즉 3팀은 최대 혹은 최소의 점수를 얻을 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 팀1 | 팀2 | 팀3 | 팀4 |
| 팀1 | X | W | L | W |
| 팀2 | L | X | W | W |
| 팀3 | W | L | X | W |
| 팀4 | L | L | L | X |

1. w\_s < t\_s, t\_s < l\_s인 경우 : 3과 반대로

위와 같은 점수 배분 크기 순서가 되어있는 경우, 3의 경우에서 W와 L을 교환해주기만 하면 된다.

1. 이외의 경우 : backtracking 기법

이외의 경우에는 규칙성을 발견하지 못했기 때문에, backtracking 기법을 이용하여 답을 도출한다. 사실상 소요 시간이 매우 길기 때문에 답을 도출하지는 못할 것이다.

**3. 코드 작성**

- 본 아이디어 및 설명을 바탕으로 코드를 작성하였다. 코드 앞부분은 backtracking에 대한 함수이며, main에서 경우를 나누었다.

#include <stdio.h>

#include <stdlib.h>

#define SIZE 5

#define SIZE\_K (SIZE-1)\*SIZE/2

SIZE\_K는 SIZE에 따라 채워야 하는 칸의 수를 의미한다. 예를 들어 20팀의 경우, SIZE\_K는 190으로, 위에서 언급한 바와 같이 190칸만을 채우면 모든 경기 결과를 얻을 수 있다.

int is\_a\_solution(char a[], int k) {

return (k == SIZE\_K); // 단계가 SIZE\_K만큼 도달한 경우를 판단함.

}

Backtracking 단계가 SIZE\_K만큼 도달한 경우, Backtracking 내의 recursive function 호출을 중단하고 결과를 계산한다.

void process\_solution(char \*\*res\_case, char \*\*res) {

int i, j;

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

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

res[i][j] = res\_case[i][j];

} // main 함수에 return 되는 res라는 이차원 배열에 조건에 맞는 이차원 배열을 복사함.

}

Process\_solution 함수에서는 결과로 생각되는 res\_case 내에 저장된 경기 결과를 main으로 옮기도록 res에 복사한다.

void construct\_candidates(char a[], char c[], int \*ncandidates) {

c[0] = 'W';

c[1] = 'T';

c[2] = 'L';

\*ncandidates = 3; // 각 단계마다 후보는 'W', 'L', 'T'로 3개가 있음.

}

Backtracking 단계에서의 각 단계 후보는 W, T, L로 3개이다.

int calculate\_score(char wtl[], int w\_s, int t\_s, int l\_s) {

int score = 0, i;

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

if (wtl[i] == 'W')

score += w\_s;

else if (wtl[i] == 'T')

score += t\_s;

else if (wtl[i] == 'L')

score += l\_s;

}

return score; // 한 팀의 모든 경기 결과가 나온 경우(wtl에 저장), 이에 따라 그 팀의 score를 계산하여 리턴함.

}

특정 팀의 19개의 결과를 통해, 그 팀의 점수를 계산하는 함수이다. 팀의 점수를 return한다.

void backtrack(char a[], int k, int w\_s, int t\_s, int l\_s, char \*\*res, int d, int e, int \*minmax) {

char \*c = calloc(3, sizeof(char));

char \*\*res\_case;

int score[SIZE], score\_sort[SIZE], mid, max, check[SIZE];

int ncandidates, i, j, n, l = 0, m = 0, num;

for (i = 0; i < SIZE; i++) check[i] = 0;

if (is\_a\_solution(a, k)) {

res\_case = (char\*\*)calloc(SIZE, sizeof(char\*));

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

res\_case[i] = (char\*)calloc(SIZE, sizeof(char));

}

/\* a(일차원 배열)를 res\_case(이차원 배열)에 저장, res\_case[][]의 모든 항목을 설정('X') \*/

190 칸을 저장하던 a 배열에서, 실제 20\*20 행렬에 그 결과를 저장한다. 반이 채워질 것이다.

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

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

res\_case[l][j + n] = a[m];

if (res\_case[l][j + n] == 'T')

res\_case[j + n][l] = 'T';

else if (res\_case[l][j + n] == 'W')

res\_case[j + n][l] = 'L';

else if (res\_case[l][j + n] == 'L')

res\_case[j + n][l] = 'W';

m++;

}

l++;

}

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

res\_case[j][j] = 'X';

}

a 행렬에 저장되어 있던 결과들을 res\_case인 20\*20 행렬에 저장한다. 위에서 언급한 바와 같이, 1팀과 2팀의 경기 결과는 2팀과 1팀의 경기 결과로 채워질 수 있기 때문에 이와 같이 채운다. 또한 각 팀은 자기 팀과 경기를 하지 않기 때문에 행렬의 대각선은 X로 채운다.

/\* res\_case의 각 팀의 점수 계산 및 저장 \*/

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

score[j] = calculate\_score(res\_case[j], w\_s, t\_s, l\_s);

}

Socre라는 1차원 배열에 각 팀의 점수를 저장한다.

/\* 저장한 각 팀의 점수(score)를 오름차순으로 저장(score\_sort) \*/

mid = 0;

max = 100;

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

for (l = 0; l < SIZE; l++) {

if (mid <= score[l] && check[l] == 0) {

mid = score[l];

num = l;

}

}

check[num] = 1;

score\_sort[j] = mid;

mid = 0;

}

Score\_sort 라는 배열에, 이미 점수가 저장되어 있는 score 배열의 원소들을 오름차순으로 sort하여 저장한다. 이는 d번째 팀을 알아내기 위함이다.

/\* e에 따라 d번째 높은 팀의 점수와 minmax를 비교, 조건에 맞는 경우 저장 및 process\_solution. \*/

if (e == 0) {

if (\*minmax >= score\_sort[d - 1]) {

\*minmax = score\_sort[d - 1];

process\_solution(res\_case, res);

}

}

else if (e == 1) {

if (\*minmax <= score\_sort[d - 1]) {

\*minmax = score\_sort[d - 1];

process\_solution(res\_case, res);

}

}

e에 따라 d번째 팀의 점수가 최대, 최소가 되는 경우인지를 비교한다. minmax에는 예시의 점수가 기록되어 있다. 이와 비교하여 더욱 크다면 혹은 작다면, minmax를 update하고 process\_solution 함수를 실행한다.

}

else {

k = k + 1;

construct\_candidates(a, c, &ncandidates);

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

a[k - 1] = c[i];

backtrack(a, k, w\_s, t\_s, l\_s, res, d, e, minmax);

단계가 아직 도달하지 않은 경우, backtrack을 계속한다.

}

}

}

int main(void) {

int w\_s, t\_s, l\_s, d, e;

char \*\*res, \*\*res\_case1, \*\*res\_case2;

char a[SIZE\_K], ex[SIZE], \*rot;

int i, j, k,l, num, count=0, score1[SIZE], score2[SIZE], score\_sort1[SIZE], score\_sort2[SIZE], check1[SIZE], check2[SIZE];

int minmax, mid, max;

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

check1[i] = 0; check2[i] = 0;

}

check 배열은 팀의 score를 sort하기 위해 임의로 생성한 배열이다.

res\_case1 = (char\*\*)calloc(SIZE, sizeof(char\*));

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

res\_case1[i] = (char\*)calloc(SIZE, sizeof(char));

}

res\_case2 = (char\*\*)calloc(SIZE, sizeof(char\*));

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

res\_case2[i] = (char\*)calloc(SIZE, sizeof(char));

}

res = (char\*\*)calloc(SIZE,sizeof(char\*));

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

res[i] = (char\*)calloc(SIZE,sizeof(char));

}

res\_case1, res\_case2는 위에서 3과 4의 경우에 대해서 점수를 비교해야하는 경우, 각 경기 결과를 저장하는 배열이다. res는 최종 경기 결과를 저장하는 배열이다.

scanf\_s("%d %d %d %d %d", &w\_s, &t\_s, &l\_s, &d, &e);

/\* 특이 조건 1의 경우 미리 설정 \*/

if ((t\_s >= w\_s && t\_s >= l\_s && e == 1) || (t\_s <= w\_s && t\_s <= l\_s && e == 0)) {

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

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

if (i == j)

res[i][j] = 'X';

else

res[i][j] = 'T';

}

}

}

위의 1 및 2의 경우에서, 모든 경기 결과를 T로 채워야 하는 경우에 대한 경기 결과를 입력한다.

/\* 특이 조건 2의 경우 미리 설정 \*/

else if ((t\_s >= w\_s && t\_s >= l\_s && e == 0) || (t\_s <= w\_s && t\_s <= l\_s && e == 1)) {

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

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

if (i == j)

res[i][j] = 'X';

else if (i < j)

res[i][j] = 'W';

else

res[i][j] = 'L';

}

}

}

위의 1과 2의 경우에서, 모든 경기 결과를 W 혹은 L로만 채워야 하는 경우에 대한 경기 결과를 입력한다.

else if ((t\_s < w\_s && t\_s > l\_s && e == 1) || (t\_s > w\_s && t\_s < l\_s && e == 0)) {

rot = (char\*)calloc(d - 1, sizeof(char));

if ((d - 1) % 2 == 0) {

for (i = 0; i < (d - 1) / 2; i++) rot[i] = 'L';

for (i = (d - 1) / 2; i < d - 1; i++) rot[i] = 'W';

}

if ((d - 1) % 2 == 1) {

for (i = 0; i < (d - 2) / 2; i++) rot[i] = 'L';

rot[(d - 2) / 2] = 'T';

for (i = (d - 2) / 2 + 1; i < d - 1; i++) rot[i] = 'W';

}

비어있는 칸의 개수가 짝수인지 홀수 인지에 따라 채워야 하는 방식이 다르다.

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

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

if (i > j) res\_case1[i][j] = 'L';

else if (i < j) res\_case1[i][j] = 'W';

else res\_case1[i][j] = 'X';

}

}

d번째 팀 ~ 20번째 팀의 모든 경기 결과를 L 혹은 W로 채운다.

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

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

res\_case1[i][(i + j) % d] = rot[count];

count++;

}

count = 0;

}

첫 번째로, WW…WTLL…L과 같은 패턴을 채우는 경우, res\_case1에 저장한다.

for (i = 0; i < d - 1; i++) rot[i] = 'T';

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

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

if (i > j) res\_case2[i][j] = 'L';

else if (i < j) res\_case2[i][j] = 'W';

else res\_case2[i][j] = 'X';

}

}

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

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

res\_case2[i][(i + j) % d] = rot[count];

count++;

}

count = 0;

}

두 번째로, 오로지 T만을 채워야 하는 경우, res\_case2에 저장한다.

//점수 계산

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

score1[j] = calculate\_score(res\_case1[j], w\_s, t\_s, l\_s);

}

mid = 0; max = 10;

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

for (l = 0; l < SIZE; l++) {

if (mid <= score1[l] && check1[l] == 0) {

mid = score1[l];

num = l;

}

}

check1[num] = 1;

score\_sort1[j] = mid;

mid = 0;

}

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

score2[j] = calculate\_score(res\_case1[j], w\_s, t\_s, l\_s);

}

mid = 0;

max = 100;

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

for (l = 0; l < SIZE; l++) {

if (mid <= score2[l] && check2[l] == 0) {

mid = score2[l];

num = l;

}

}

check2[num] = 1;

score\_sort2[j] = mid;

mid = 0;

}

각 첫번째 혹은 두번째 경우에 대해서, d번째 팀의 점수를 score\_sort1, 2에 저장한다.

if (e == 0) {

if (score\_sort2[d - 1] >= score\_sort1[d - 1]) {

process\_solution(res\_case1, res);

}

else process\_solution(res\_case2, res);

}

else if (e == 1) {

if (score\_sort2[d - 1] >= score\_sort1[d - 1]) {

process\_solution(res\_case2, res);

}

else process\_solution(res\_case1, res);

}

}

e에 따라 구해야 하는 점수가 최대 혹은 최소인지를 판단한 후, 비교하여 결과에 저장한다.

else if (t\_s < w\_s && t\_s > l\_s && e == 0 || (t\_s > w\_s && t\_s < l\_s && e == 1)) {

rot = (char\*)calloc(d - 1, sizeof(char));

if ((d - 1) % 2 == 0) {

for (i = 0; i < (d - 1) / 2; i++) rot[i] = 'W';

for (i = (d - 1) / 2; i < d - 1; i++) rot[i] = 'L';

}

if ((d - 1) % 2 == 1) {

for (i = 0; i < (d - 2) / 2; i++) rot[i] = 'W';

rot[(d - 2) / 2] = 'T';

for (i = (d - 2) / 2 + 1; i < d - 1; i++) rot[i] = 'L';

}

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

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

if (i > j) res\_case1[i][j] = 'W';

else if (i < j) res\_case1[i][j] = 'L';

else res\_case1[i][j] = 'X';

}

}

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

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

res\_case1[i][(i + j) % d] = rot[count];

count++;

}

count = 0;

}

for (i = 0; i < d - 1; i++) rot[i] = 'T';

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

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

if (i > j) res\_case2[i][j] = 'W';

else if (i < j) res\_case2[i][j] = 'L';

else res\_case2[i][j] = 'X';

}

}

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

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

res\_case2[i][(i + j) % d] = rot[count];

count++;

}

count = 0;

}

//점수 계산

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

score1[j] = calculate\_score(res\_case1[j], w\_s, t\_s, l\_s);

}

mid = 0;

max = 100;

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

for (l = 0; l < SIZE; l++) {

if (mid <= score1[l] && check1[l] == 0) {

mid = score1[l];

num = l;

}

}

check1[num] = 1;

score\_sort1[j] = mid;

mid = 0;

}

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

score2[j] = calculate\_score(res\_case1[j], w\_s, t\_s, l\_s);

}

mid = 0;

max = 100;

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

for (l = 0; l < SIZE; l++) {

if (mid <= score2[l] && check2[l] == 0) {

mid = score2[l];

num = l;

}

}

check2[num] = 1;

score\_sort2[j] = mid;

mid = 0;

}

if (e == 0) {

if (score\_sort2[d - 1] >= score\_sort1[d - 1]) {

process\_solution(res\_case1, res);

}

else process\_solution(res\_case2, res);

}

else if (e == 1) {

if (score\_sort2[d - 1] >= score\_sort1[d - 1]) {

process\_solution(res\_case2, res);

}

else process\_solution(res\_case1, res);

}

}

위와 동일한 방식으로 코드를 작성한다. 단, 위의 모든 경우에서 W와 L의 위치만을 변경한다.

else {

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

ex[i] = 'T';

ex[0] = 'X';

minmax = calculate\_score(ex, w\_s, t\_s, l\_s);

backtrack(a, 0, w\_s, t\_s, l\_s, res, d, e, &minmax);

}

이외의 경우에 대해서는 backtracking 기법을 사용한다. 결과는 제대로 도출되지 않을 것이다.

/\* 출력 \*/

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

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

printf("%c ", res[i][j]);

printf("\n");

}

결과를 출력한다.

return 0;

}

**4. 고찰**

본 문제는 20개의 축구 팀이 리그전을 통해 얻은 경기 결과에 대해서, 주어진 승, 무, 패에 따른 배분되는 점수를 토대로 d번째 높은 점수를 가진 팀의 점수가 최대 혹은 최소가 되도록 하는 경기 결과를 출력하는 것이다. 내가 생각한 아이디어는 Backtracking 기법이다. 가능한 모든 경기 결과에 대해 점수를 계산하고, 가장 높은 혹은 낮은 점수를 기록하는 경기 결과를 저장하는 것이다. 하지만 backtracking 기법을 통해 고려해야 하는 모든 경우의 수는 3^190 가지로 매우 많다. 이에 따라, 6팀 이상의 축구 팀이 있는 경우에 대해서는 결과가 제대로 도출되지 않았다.

이후 이미 주어진 input의 대소 관계에 따라 여러 경우로 나누어, 각 특이한 경우에 대해서 경기 결과를 작성하도록 하였다. 모든 경우에 대해서 패턴을 분석하지는 못하였지만, 일부 경우에 대해서는 정확한 패턴 분석을 통해 정확한 답을 도출해낼 수 있었다.

**5. 참고문헌 및 사용**

- 이진규 교수님, [문제해결기법] CSE2011\_2016spring\_Lecture\_Note07, 2016