2017 SCPC 1차 예선 문제

괄호

어떤 문자열이 '(', '{', '[', ')', '}'. ']' 로만 이루어져있을 때 이것은 올바른 괄호문자열일 수도 있다.

- A가 올바른 괄호문자열이면 '(A)', '{A}', '[A]' 도 올바른 괄호 문자열이다.
- A와 B가 둘 다 올바른 괄호문자열이고 서로 옆에 붙어있다면 AB도 올바른 괄호문자열이다.
- 공백은 올바른 괄호문자열이다

위 3가지 규칙을 만족하는 것을 올바른 괄호문자열이라고 한다.

부분 문자열이란 어떤 문자열의 일부를 말한다. 예를 들어, 'AB' 는 'QGEABD'의 부분문자열이다.

어떤 문자열이 주어졌을 때, 그 문자열의 부분문자열중 가장 긴, 올바른 괄호문자열의 길이를 구하여라.

입력

문자열이 하나 주어진다.

출력

주어진 문자열의 부분문자열들 중 가장 긴 올바른 괄호문자열의 길이를 출력한다. (정수)

풀이

주어진 문자열에서 '()', '{}', '[]' 를 찾는다. 찾은 문자열을 Chunk라 하자.

- Chunk는 1번 규칙에 의해 주변이 괄호쌍으로 둘러쌓여있으면 그 괄호쌍을 삼켜서 길이가 2만큼 늘어난다
- Chunk는 2번 규칙에 의해 주변 Chunk와 맞닿게 되면 합체한다.

이 2가지 성질을 구현해 반복하다보면 더이상 Chunk가 합체하거나 성장할 수 없게 된다.

그 Chunk들 중 가장 긴 Chunk가 가장 긴 올바른 괄호문자열이다.

```
#include <iostream>
#include <string.h>
#include <vector>
#include <list>

using namespace std;

typedef list< pair<int, int> > pair_list;

char S[1000000];
int S_len;
pair_list chunks;

int Answer;

int is_closer(char ch)
{
    if(ch == ')' || ch == ']' || ch == '}') return 1;
    return 0;
}

int is_pair(char a, char b)
{
    if(a == '(' && b == ')') return 1;
    if(a == '[' && b == ']') return 1;
    if(a == '{' && b == '}') return 1;
    return 0;
}

int merge_chunks()  // Rule 2
{
    int did_something = 0;
    for(pair_list::iterator it = chunks.begin(); it != chunks.end(); ++it) {
        // .printf("at: %d\n", it->first);
        for(;;) {
            pair_list::iterator next = it;
            ++next;

            if(next == chunks.end()) {
```

```
                return did_something;
            }

            if(it->first + it->second == next->first) {
                // printf("merge: (%d, %d), (%d, %d)\n", it->first, it->second, next->first, next->second);
                it->second += next->second;
                chunks.erase(next);
                did_something = 1;
            } else {
                break;
            }
        }
    }
    return did_something;
}

int feed_chunks()  // Rule 1
{
    int did_something = 0;
    for(pair_list::iterator it = chunks.begin(); it != chunks.end(); ++it) {
        for(;;) {
            if(it->first <= 0) break;
            char left_ch = S[it->first - 1];

            if(it->first + it->second >= S_len) break;
            char right_ch = S[it->first + it->second];

            if(is_pair(left_ch, right_ch)) {
                it->first -= 1;
                it->second += 2;
                did_something = 1;
                // printf("feed: %d, %d => %d, %d\n", it->first+1, it->second-2, it->first, it->second);
            } else {
                break;
            }
        }
    }
    return did_something;
}

int do_task(void)
{
    chunks.clear();
    cin >> S;
    S_len = strlen(S);
    for(int i=1; i<S_len; ++i) {  // find minimum chunks
        if(is_pair(S[i-1], S[i])) {
            chunks.push_back(make_pair(i-1, 2));
            // printf("found: %d, %d\n", i-1, i);
```

```
        }
    }

    int merged, feed;
    do {
        merged = merge_chunks();
        feed = feed_chunks();
    } while(merged || feed);

    int max_length = 0;
    for(pair_list::iterator it = chunks.begin(); it != chunks.end(); ++it) {
        if(max_length < it->second) {
            max_length = it->second;
        }
    }

    return max_length;
}

메인 생략...
```

주식 거래

우리는 특정 구간에서의 주식 가격들을 알고 있다. 이 주식 가격들은 모두 매일 오전 10시를 기준으로 한다.

어떤 사람은 매일 오전 10시에 주식을 사거나 판다. 이 때,

- 이미 주식을 구매한 상태이면 주식을 또 구매할 수 없다.
- 주식을 구매한 상태이며, 현재의 주식가격이 구매할 때의 시점보다 높다면, 그날 오전 10시에 구매한 주식을 팔아 이익을 실현시킬 수 있다.
- 하루에는 사거나 팔거나 둘 중 하나의 행동만 취할 수 있다.

특정 구간의 주식가격들이 주어졌을 때, 이 사람은 최대 몇 번 주식거래를 할 수 있는지 구해라.

입력

특정 정수 N이 입력되고, N개의 $C_i, (0 \leq i < N)$ 이 입력된다. C_i는 i번째 날의 오전 10시의 주식 가격을 뜻한다.

출력

이 사람이 최대로 할 수 있는 주식 매매의 횟수를 출력한다.

풀이

주식 가격의 변화가 상승세에서 하락세로 바뀌는 시점이 몇 번인지, 주식 가격이 마지막에 상승세로 끝나는지 체크한다.

(주식 가격이 올라가다 내려가는 시점의 개수) * 2 번 거래할 수 있고, 마지막에 주식이 상승세로 끝난다면 2번을 더 거래 할 수 있다.

```
#include <iostream>

using namespace std;

int Answer;
int N;
int costs[200000];
int descending;

int do_task()
{
    cin >> N;
    if(N < 1) return 0;
    for(int i=0; i<N; ++i) {
        cin >> costs[i];
    }

    int down_up = 0;
    int up_down = 0;
    descending = (costs[0] >= costs[1]);

    for(int i=2; i<N; ++i) {
        if(costs[i-1] > costs[i]) { // down
            if(!descending) {
                descending = 1;
                up_down++;
            }
        }
        else if(costs[i-1] < costs[i]) { // up
            if(descending) {
                descending = 0;
                down_up++;
            }
        }
    }

    return 2 * (up_down + (1 - descending));
}

메인 생략...
```

전광판

어떤 전광판에는 격자형으로 전구들이 배치되어있다. 각 전구들은 각각 2개의 스위치와 연결되어있다. 이 스위치는 작동되면 연결된 전구들의 상태를 반전시키는데, 만약 전구가 켜져있었다면 끄고, 꺼져있었다면 키게 된다.

스위치의 종류에는 Column 스위치와 Row 스위치가 있다. Column스위치는 특정 열에 상주하여, 그 열에 있는 전구들 중 일부분들과만 연결되어있고, Row스위치는 특정 행에 상주하여, 그 행에 있는 전구들 중 일부분들과만 연결되어있다. 각 행이나 열에는 상주하는 스위치가 여러 개 있을 수 있다. 그렇기 때문에 전구가 $N \times M$형태로 배치되어있다면 스위치는 최대 $2NM$개 있을 수 있고, 최소 $N+M$개는 있어야 한다. 왜냐면 각 스위치는 최소 1개에서 Row스위치라면 최대 M개, Column스위치라면 최대 N개의 전구와 연결될 수 있기 때문이다.

전구는 하나의 Row스위치와 하나의 Column스위치와 연결되어있기 때문에 총 2개의 스위치와 연결되어있다.

이 전광판의 전구는 일부는 꺼져있고 일부는 켜져있다.

전광판의 상태와, 전광판의 전구들의 켜짐 상태, 전구들과 스위치들간의 연결 상태가 주어졌을 때, 전구들을 모두 한 번에 켜려면 어떤 스위치들을 작동시켜야 하는지 구하시오.

입력

전광판의 크기 N, M이 주어진다. 그리고나서 3개의 정수세트($A_{ij}, R_{ij}, C_{ij}, (0 \le i < N, 0 \le j < M)$)가 NM개 만큼 주어지는데, 3개의 정수가 의미하는 것은 각각, 이렇다

- A_{ij} 는 격자의 (i,j)에 있는 전구가 초기에 켜져있는지를 의미한다. 0이면 꺼짐, 1이면 켜짐이다.
- R_{ij}는 격자의 (i,j)에 있는 전구와 연결되있는 Row 스위치의 번호를 의미한다. 즉, Row스위치들중에, i번째 줄의 전구들중 일부와 연결된 스위치들 중의 R_{ij}번 스위치와 연결된 것이다.
- C_{ij}는 격자의 (i,j)에 있는 전구와 연결되있는 Column 스위치의 번호를 의미한다. 즉, Column스위치들중에, j번째 줄의 전구들중 일부와 연결된 스위치들 중의 C_{ij}번 스위치와 연결된 것이다.

출력

만약 모든 전구들을 켜는 것이 가능하다면 그것에 필요한, 작동되어야 하는 스위치들을 모두 나열한다. 각 스위치의 이름은 Row스위치면 R, Column스위치면 C로 시작한다. 그리고 Row스위치이면서 i번째 줄의 전구들 중 일부와 연결된 스위치들 중의 n번째 스위치라면 "$\mathrm{R}in$" 으로 표현된다. 예를 들어 3번째 줄의 2번째 스위치이면 R0302가 된다. 혹은, 2번째 열의 3번째 스위치면 C0203이 된다.

풀이

일단 아무 스위치나 선택한 후, 그 스위치가 사용되어선 안된다고 가정해보자. 그러면, 그 스위치와 연결된 전구들을 봤을 때, 각각 연결된 또 다른 스위치의 사용여부가 결정된다.

왜냐면 만약, 어떤 전구가 켜져있고, 그 전구와 연결된 두 스위치들 중, 한 스위치가 사용된다고 한다면 나머지 한 스위치도 사용되어야 그 전구가 켜진 상태로 유지되기 때문이다. 즉, 전구의 상태를 A, 스위치들 중 상태가 결정된 스위치의 사용여부를 B, 나머지 한 스위치의 사용여부를 C라고 한다면, $C = A \ \mathrm{XNOR} \ B$ 가 되는 것을 알 수 있다.

이런 식으로 연쇄적으로, 하나의 스위치의 사용여부를 결정해보면 다른 스위치들의 사용여부도 결정이 된다. 이 과정 중에, 모순이 생긴다면 첫번째로 선택한 스위치가 사용되어선 안된다는 가정이 틀린 것이므로 사용해야 한다는 가정으로 다시 해본다. 만약 이것도 실패한다면 애초에 모든 전구를 켜는 것이 불가능한 것이 된다.

위 과정을 모든 스위치의 사용여부가 결정될 때 까지 하면 된다.

```
#include <iostream>

using namespace std;

int N, M;
char tile[100][100];

char from_R[100][100];
char from_C[100][100];

char R[100][100];
char C[100][100];

int try_R(int x, int idx, int v);
int try_C(int y, int idx, int v);

void print_2(int a)
{
    if(a < 10) {
        cout << "0" << a;
    } else {
        cout << a;
    }
}

int try_R(int x, int idx, int v)
{
    if(R[x][idx] != -1) {
        if(R[x][idx] == v) return 1;
        else return 0;
    }

    R[x][idx] = v;
    int succ = 1;
    for(int y=0; y<M; ++y) {
        if(from_R[x][y] == idx) {
            if(!try_C(y, from_C[x][y], (int)(v == tile[x][y])))
{
                succ = 0;
                break;
            }
        }
    }
    if(!succ) {
        R[x][idx] = -1;
```

```
45          return 0;
46      }
47      return 1;
48  }
49
50  int try_C(int y, int idx, int v)
51  {
52      if(C[y][idx] != -1) {
53          if(C[y][idx] == v) return 1;
54          else return 0;
55      }
56
57      C[y][idx] = v;
58      int succ = 1;
59      for(int x=0; x<N; ++x) {
60          if(from_C[x][y] == idx) {
61              if(!try_R(x, from_R[x][y], (int)(v == tile[x][y])))
    {
62                  succ = 0;
63                  break;
64              }
65          }
66      }
67      if(!succ) {
68          C[y][idx] = -1;
69          return 0;
70      }
71      return 1;
72  }
73
74  int search()
75  {
76      for(int i=0; i<N; ++i) {
77          for(int j=0; j<M; ++j) {
78              if(R[i][j] == -1) {
79                  if((!try_R(i, j, 0)) && (!try_R(i, j, 1))) {
80                      return 0;
81                  }
82              }
83          }
84      }
85      for(int i=0; i<M; ++i) {
86          for(int j=0; j<N; ++j) {
87              if(C[i][j] == -1) {
88                  if((!try_C(i, j, 0)) && !(try_C(i, j, 1))) {
89                      return 0;
90                  }
91              }
92          }
93      }
94      return 1;
95  }
```

```
96
97  void do_task()
98  {
99      cin >> N >> M;
100     for(int i=0; i<N; ++i) {
101         for(int j=0; j<M; ++j) {
102             R[i][j] = 0;
103             C[i][j] = 0;
104         }
105     }
106
107     for(int i=0; i<N; ++i) {
108         for(int j=0; j<M; ++j) {
109             int a, b, c;
110             cin >> a >> b >> c;
111             tile[i][j] = a;
112             from_R[i][j] = b;
113             from_C[i][j] = c;
114             R[i][from_R[i][j]] = -1;
115             C[j][from_C[i][j]] = -1;
116         }
117     }
118
119     int succ = search();
120
121     if(succ) {
122         for(int i=0; i<M; ++i) {
123             for(int j=0; j<N; ++j) {
124                 if(C[i][j]) {
125                     cout << "C";
126                     print_2(i);
127                     print_2(j);
128                     cout << " ";
129                 }
130             }
131         }
132         for(int i=0; i<N; ++i) {
133             for(int j=0; j<M; ++j) {
134                 if(R[i][j]) {
135                     cout << "R";
136                     print_2(i);
137                     print_2(j);
138                     cout << " ";
139                 }
140             }
141         }
142     } else {
143         cout << "Impossible";
144     }
145
146     cout << endl;
147 }
```

```
148
149  메인 생략...
```