1 Aliens

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

typedef vector<pair<int, int> > vii;       // sorted by left, right.
bool sortDescAsc(const pair<int, int>& lhs, const pair<int, int>& rhs)
    {
    if(lhs.first == rhs.first)
        return (lhs.second > rhs.second);
    else
        return lhs.first < rhs.first;
}

void testcase() {
    int n, m;
    cin >> n >> m;
    vii intervals;
    int superior = n;
    for(int i = 0; i < n; ++i) {
        int pi, qi;
        cin >> pi >> qi;
        if(pi == 0 && qi == 0) {
            --superior;
            continue;
        }
        pair<int, int> entry = make_pair(pi, qi);
        intervals.push_back(entry);
    }

    sort(intervals.begin(), intervals.end(), sortDescAsc);

    int left = 0;
    int right = 0;
    for(int i = 0; i < intervals.size(); ++i) {
        if(i+1 < intervals.size() && intervals[i+1].first == intervals[
            i].first && intervals[i+1].second == intervals[i].second)
            --superior;
        else if(left == intervals[i].first && right == intervals[i].
            second)
            --superior;
```

```
        else if(right >= intervals[i].second)
            --superior;

        if(right < intervals[i].second) {
            left = intervals[i].first;
            if(right != 0 && left-right > 1) {
                cout << "0\n";
                return;
            }
            right = intervals[i].second;
        }
    }

    cout << superior << "\n";
}

int main() {
    int TC;
    cin >> TC;
    while(TC--) testcase();
}
```

2 Checking Change

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <string>
#include <sstream>
using namespace std;

vector<string> answers;

int main(int argc, char const *argv[])
{

        int currencies;
        cin >> currencies;

        for (int currency = 0; currency < currencies; currency++)
        {

                int coins_count;
                int testcases;

                cin >> coins_count >> testcases;

                vector<int> coins;
                for (int coins_it = 0; coins_it < coins_count; coins_it
```

```
    ++)
{
        int coin;
        cin >> coin;
        coins.push_back(coin);
}

vector<int> tests;
for (int testcase = 0; testcase < testcases; testcase
    ++)
{
        int test;
        cin >> test;
        tests.push_back(test);
}

// find maximum of tests
vector<int>::iterator max_test_it = max_element(tests.
    begin(), tests.end());
int max_test = *max_test_it;
int N = max_test + 1;

vector<int>::iterator max_coin_it = max_element(coins.
    begin(), coins.end());
int max_coin = *max_coin_it;

vector<int>::iterator min_coin_it = min_element(coins.
    begin(), coins.end());
int min_coin = *min_coin_it;

// instantiate array with size max(tests)
int arraysize = 2;
vector<int> counts(arraysize);

// fill indices we already know -> coins, set to zero
    where index smaller than index of smallest coin.
for (int i = 0; i < min_coin; i++)
{
        if (min_coin >= arraysize)
        {
                arraysize += min_coin + 10;
                counts.resize(arraysize);
                //cout << "vector size now " <<
                    arraysize;
        }
        counts[i] = 0;
}

for (vector<int>::iterator coins_it = coins.begin();
```

```
    coins_it != coins.end(); coins_it++)
{
        if (*coins_it <= max_coin)
        {
                if (*coins_it >= arraysize)
                {
                        arraysize += *coins_it + 1;
                        counts.resize(arraysize);
                        //cout << "vector size now " <<
                            arraysize;
                }
                counts[*coins_it] = 1;
        }
}

// iterate over counts, combine all minimums.
for (int n = min_coin + 1; n < N; n++)
{
        if (arraysize <= n)
        {
                arraysize += 1;
                counts.resize(arraysize);
                //cout << "vector size now " <<
                    arraysize;
        }

        signed int min = -1;
        for(int backward = n-1; backward >= min_coin;
            backward--) {

                if (counts[n] == 1)
                {
                        min = 1;
                } else {
                        if(counts[backward] != 0 &&
                            counts[n-backward] != 0) {
                                int new_min = counts[
                                    backward] + counts[
                                    n-backward];
                                //cout << n << ":
                                    counts[backward]: "
                                     << counts[backward
                                    ] << " counts[n-
                                    backward]: " <<
                                    counts[n-backward]
                                    << "new_min: "<<
                                    new_min << "\n";
                                if (min > new_min ||
                                    min == -1)
```

```
                                                {
                                                        min = new_min;
                                                }
                                        }
                                }

                        }
                        if (min == -1)
                        {
                                min = 0;
                        }
                        counts[n] = min;
                }

                /*int i = 0;
                for (vector<int>::iterator elements = counts.begin();
                    elements != counts.end(); elements++)
                {
                        cout << i++ << ": " << *elements << " \n";
                }*/

                for (vector<int>::iterator test = tests.begin(); test
                   != tests.end(); test++)
                {
                        int answer = counts[*test];

                        stringstream ss;
                        if (answer == 0)
                        {
                                ss << "not possible";
                        } else {
                                ss << answer;
                        }

                        answers.push_back(ss.str());
                }

        }

        for (vector<string>::iterator answer = answers.begin(); answer
           != answers.end(); answer++)
                cout << *answer << "\n";

        return 0;
}
```