Tuenti Programming Challenge 2: Solution Sketches

Alex Alvarez Ruiz

May 6, 2012

Contents

1 Challenge 1: The cell phone keypad

In this problem there are two different parts to solve. The first one is, given two characters, find the minimum time to go from the first character to the second one. The second part is just iterate proceeding as the statement says and using the values calculated before. As the second part is just a bit of coding, let's talk about the first part.

With the times given, is easy to find a formula to answer this question in $O(1)$ time. A more general form is to see the keypad as a graph, where each key is a node connected to the neighbouring keys and each edge has as cost the time needed. Then, we are trying to solve a classical problem: All pair's minimum distance, which can be solved using Floyd-Warshall algorithm.

Code 1: Floyd-Warshall

```
for (int k = 0; k < 11; ++k)
    for (int i = 0; i < 11; ++i)
        for (int j = 0; j < 11; ++j)
            M[i][j] = min(M[i][j], M[i][k] + M[k][j]);
```

2 Challenge 2: The binary granny

The problem is given a number $0 \leq N \leq 10^{19}$, split it into two positive numbers x, y, such that $x + y = N$, maximizing the sum of the number of ones of x and y in base 2. This problem can be easily solved using dynamic programming. If we start with the least significant bit of x, a value for it implies exactly one possible value for the same bit in y. Then, we only need to remember the actual carry. This observation gives a dynamic programming with state $(position, carry)$.

Let $bin(p)$ mean the value of the p-th bit of N. The recurrence is like that:

- $dp(p, c) = \max(2 + dp(p + 1, 1), dp(p + 1, 0))$, `if` $bin(p) = c$
- $dp(p, c) = 1 + dp(p + 1, 0)$, `if` $bin(p) = 1$ `and` $c = 0$
- $dp(p, c) = 1 + dp(p + 1, 1)$, `else`

This gives us a $O(\log N)$ algorithm. The following code implements this algorithm.

Code 2: Solution to the second problem

```
#include <iostream>
#include <vector>
#include <cstring>
using namespace std;

typedef unsigned long long ull;

const int INF = 1000000000;
int M[70][2];
string bin;

string to_binary(ull x) {
    string s;
    do {
        s += char('0' + (x&1));
        x >>= 1;
    } while (x > 0);
    return s;
}

int dp(int p, int c) {
    if (p == bin.size()) return (c == 0) ? 0 : -INF;
    int &ans = M[p][c];
    if (ans == -1) {
        if ((bin[p] == '1' and c == 1) or (bin[p] == '0' and c == 0))
            ans = max(2 + dp(p + 1, 1), dp(p + 1, 0));
        else if (bin[p] == '1' and c == 0) ans = 1 + dp(p + 1, 0);
        else ans = 1 + dp(p + 1, 1);
```

```
    }
    return ans;
}

int main() {
    int casos;
    cin >> casos;
    for (int cas = 1; cas <= casos; ++cas) {
        ull n;
        cin >> n;
        bin = to_binary(n);
        memset(M, -1, sizeof(M));
        cout << "Case #" << cas << ": " << dp(0, 0) << endl;
    }
}
```

3 Challenge 3: The evil trader

Let a_i be the i-th number. We need to find i, j, $i < j$, $a_i < a_j$, with maximum $a_j - a_i$. A greedy approach solves in $O(N)$ time this problem, where N is the size of the input. Just iterate over the sequence and keep the minimum value found previously and compare with the actual value to see if the difference is greater than the best found at this moment.

Code 3: Solution to problem 3

```
#include <iostream>
using namespace std;

typedef long long ll;

int main() {
    ll minim, best = 0;
    int posminim = 0, posmaxim = 0, posminimact = 0, i = 1;
    cin >> minim;
    ll x;
    while (cin >> x) {
        if (x - minim > best) {
            best = x - minim;
            posminim = posminimact;
            posmaxim = i;
        }
        if (minim > x) {
            minim = x;
            posminimact = i;
        }
        ++i;
    }
    cout << posminim*100 << " " << posmaxim*100 << " " << best << endl;
}
```

4 Challenge 4: 20 fast 20 furious

A typical problem about cycle finding. The queue state can be determined by the first group, as they will always go in order, so we only have G possible states. Then, iterate as the statement says until repeat one state (at most G iterations) and apply this cycle in constant time as many times as possible. Then just iterate the remaining laps (at most G - 1).

Each iteration could be done in constant time precalculating for each queue state how many groups enter in the race. This can be easily done in $O(G \log G)$ with a binary search. Thus, we can solve the problem in time $O(G \log G)$. It was not needed, but the $\log G$ factor can be avoided using an standard two-pointers' algorithm.

Code 4: Solution to problem 4 in $O(G \log G)$

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

#define MOD(x,n) (((x) < (n)) ? (x) : ((x)%(n)))

typedef long long ll;

//Calculo la solucion en O(G) con un precalculo en tiempo O(GlogG)
void solve(ll r, ll k, vector<ll>& v) {
    int n = v.size();

    //Precalculo en O(GlogG) para cada estado de la fila cuantos grupos entran
    vector<ll> S(2*n + 1, 0), qtt(n);
    for (int i = 1; i <= 2*n; ++i) S[i] = S[i - 1] + v[MOD(i - 1, n)];
    for (int i = 0; i < n; ++i) {
        //Binaria para encontrar el punto exacto
        int ini = i, fin = i + n - 1;
        while (ini <= fin) {
            int m = (ini + fin)/2;
            ll x = S[m + 1] - S[i];
            if (x <= k) ini = m + 1;
            else fin = m - 1;
        }
        qtt[i] = ini - i;
    }

    //Busco un ciclo dentro de la secuencia
    ll r2 = 0, res = 0;
    vector<ll> races(n, -1); //Cuantas carreras lleva el grupo i
                             //la primera vez que se pone el primero

    vector<ll> litros(n, -1); //Lo mismo para los litros
    int act = 0;
    while (r2 < r and races[act] < 0) {
        races[act] = r2;
        litros[act] = res;
        res += S[act + qtt[act]] - S[act];
        ++r2;
        act = MOD(act + qtt[act], n);
```

```
41      }
        if (r2 < r) {
            ll len = r2 - races[act]; //Longitud del ciclo
            ll veces = (r - r2)/len; //Cuanto lo puedo repetir
            ll gasto = res - litros[act]; //Gasolina del ciclo
46          r2 += (veces*len);
            res += (veces*gasto);

            //El resto lo itero, sera menor que G el numero de iteraciones
            while (r2 < r) {
51              res += S[act + qtt[act]] - S[act];
                ++r2;
                act = MOD(act + qtt[act], n);
            }
        }
56      cout << res << endl;
}

int main() {
        int casos;
61      cin >> casos;
        for (int cas = 1; cas <= casos; ++cas) {
            ll r, k;
            int n;
            cin >> r >> k >> n;
66          vector<ll> v(n);
            for (int i = 0; i < n; ++i) cin >> v[i];
            solve(r, k, v);
        }
}
```

5 Challenge 5: Time is never time again

The most difficult part of this problem is the implementation. The idea is to calculate the difference in one day, so you only need to calculate the initial day's difference, the last day's difference and add to it the number of days between them times the difference in one day.

So first write some function to let you calculate the difference between two given times and use them to calculate the difference in one day: 2255477. Then, take the first day given and finish it, calculating the difference with the same functions. Do the same (now from the beginning) with the last day. The trickier part is maybe to calculate how many days are between two dates. But there is a formula for that:

Code 5: Number of days between two dates

```
typedef long long ll;

inline ll g(ll y, ll m, ll d) {
    m = (m + 9)%12;
    y = y - m/10;
    return 365*y + y/4 - y/100 + y/400 + (m*306 + 5)/10 + (d - 1);
}

inline int days(int y1, int m1, int d1, int y2, int m2, int d2) {
    return g(y2, m2, d2) - g(y1, m1, d1);
}
```

6 Challenge 6: Cross-stitched fonts

One possible solution is binary search over the font, as it is a monotone function. For a fixed font, it is easy to check whether it fits. Once you have the optimal font, just calculate the answer using the formula described in the statement.

Code 6: Solution to problem 6

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

double EPS = 1e-6;

//Miro si puedo poner fuente n
bool check(int w, int h, vector<string>& v, int n) {
    if (n == 0) return true;
    int lines = h/n;
    if (lines == 0) return false;
    int linia = 0, ch = 0, i = 0;
    while (i < v.size()) {
        int x = v[i].size();
        if (ch > 0) ++x;
        if ((x + ch)*n <= w) {
            ch += x;
            ++i;
        }
        else {
            if (ch == 0) return false;
            else {
                ch = 0;
                ++linia;
                if (linia == lines) return false;
            }
        }
    }
    return true;
}

int main() {
    int casos;
    cin >> casos;
    for (int z = 1; z <= casos; ++z) {
        cout << "Case #" << z << ": ";
        int w, h, count;
        cin >> w >> h >> count;
        w *= count;
        h *= count;
        string s, r;
        vector<string> v;
        getline(cin, s);
        getline(cin, s);
        r = s;
        stringstream ss(s);
        while (ss >> s) v.push_back(s);

```

```
            //Binaria sobre la fuente
            int ini = 0, fin = min(w, h);
            while (ini <= fin) {
                int m = (ini + fin)/2;
                if (check(w, h, v, m)) ini = m + 1;
                else fin = m - 1;
            }

            //Calculo cuanto hilo necesito con esa fuente.
            double res = 0, qtt = (ini - 1)*(ini - 1)/2.0/double(count);
            for (int i = 0; i < r.size(); ++i) {
                if (r[i] != ' ') res += qtt;
            }
            cout << int(res + 1 - EPS) << endl;
        }
}
```

7 Challenge 7: The "secure" password

The key constraint to solve this problem is the following sentence: *assuming that passwords do not contain repeated characters*. Consider the following directed graph:

- Create a node for each character in the subcodes.
- For each consecutive two characters in some subcode, create an edge from tre first character to the second.

It is trivial to see that a word is a solution if and only if it is a topological sorting of this graph. So simply backtrack over all the topological sortings of the graph.

Code 7: Solution to problem 7

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

typedef vector<int> VI;
typedef vector<VI> VVI;

vector<char> alfa;
vector<string> res;
VVI G;
VI in;

//Genero todas las ordenaciones topologicas del arbol formado por
//un nodo para cada caracter y arista si aparece esa pareja en un subcode
//en ese orden
void rec(int p, list<int>& L, VI& sol) {
    if (p == sol.size()) {
        string s;
        for (int i = 0; i < sol.size(); ++i) s += alfa[sol[i]];
        res.push_back(s);
        return;
    }
    list<int> aux = L;
    for (list<int>::iterator it = L.begin(); it != L.end(); ++it) {
        int u = *it, qtt = 0;
        sol[p] = u;
        aux.pop_front();
        for (int i = 0; i < G[u].size(); ++i) {
            int v = G[u][i];
            --in[v];
            if (in[v] == 0) {
                aux.push_back(v);
                ++qtt;
            }
```

```
            }
            rec(p + 1, aux, sol);
            for (int i = 0; i < G[u].size(); ++i) {
                int v = G[u][i];
                ++in[v];
            }
            for (int i = 0; i < qtt; ++i) aux.pop_back();
            aux.push_back(u);
        }
    }

    int main() {
        string s;
        vector<string> v;
        map<char, int> id;
        while (cin >> s) {
            v.push_back(s);
            for (int i = 0; i < s.size(); ++i) {
                if (not id.count(s[i])) {
                    id[s[i]] = G.size();
                    alfa.push_back(s[i]);
                    G.resize(G.size() + 1);
                    in.push_back(0);
                }
                if (i > 0) {
                    G[id[s[i - 1]]].push_back(id[s[i]]);
                    ++in[id[s[i]]];
                }
            }
        }
        list<int> L;
        for (int i = 0; i < G.size(); ++i) if (in[i] == 0) L.push_back(i);
        VI sol(alfa.size());
        rec(0, L, sol);
        sort(res.begin(), res.end());
        for (int i = 0; i < res.size(); ++i) cout << res[i] << endl;
    }
```

8 Challenge 8: The demented cloning machine

The first thing to see is that we cannot generate the final string and calculate its MD5, because the string can be very large. My solution for this problem is divided into two parts. The first part is a dynamic programming to calculate the following: given a character and a series number, which string does it become in the end?

Then, calculate with some library the MD5 of the string. To do this, let's iterate over the original characters and calculate the MD5 of its final string. Then just print the output.

Some further optimizations can be done, but they make the code harder to implement. In my old laptop, this solution run in 24 seconds with the test and in 15 seconds (more or less) with the submit.

Code 8: Solution to problem 8

```
#include <iostream>
#include <openssl/md5.h>
#include <vector>
#include <cstdio>
#include <sstream>
using namespace std;

#define DP(c,p) ((M[int(c)][p].size() == 0) ? dp(c,p) : M[int(c)][p])

string M[128][20];
unsigned char res[16];
vector<vector<string> > T;

vector<string> split(string& s) {
    for (int i = 0; i < int(s.size()); ++i) if (s[i] == ',') s[i] = ' ';
    stringstream ss(s);
    vector<string> ans(128);
    while (ss >> s) ans[int(s[0])] = s.substr(3, int(s.size()) - 3);
    return ans;
}

string dp(char c, int p) {
    if (p == int(T.size())) return string(1, c);
    if (M[int(c)][p].size() == 0) {
        if (T[p][int(c)].size() > 0) {
            for (int j = 0; j < int(T[p][int(c)].size()); ++j)
                M[int(c)][p] += DP(T[p][int(c)][j], p + 1);
        }
        else M[int(c)][p] = DP(c, p + 1);
    }
    return M[int(c)][p];
}

int main() {
    string s, r;
```

```
36      getline(cin, s);
        while (getline(cin, r)) T.push_back(split(r));
        string aux;
        MD5_CTX md;
        MD5_Init(&md);
41
        //Precalculo en que se transforma cada caracter con una programacion dinamica
        for (int i = 0; i < 128; ++i) dp(char(i), 0);

        //Voy calculando el MD5 a trozos
46      for (int i = 0; i < int(s.size()); ++i)
            MD5_Update(&md, M[int(s[i])][0].c_str(), M[int(s[i])][0].size());

        MD5_Final(res, &md);
        for (int i = 0; i < 16; ++i) printf("%02x", res[i]);
51      printf("\n");
}
```

9 Challenge 9: Il nomme della magnolia

A very nice problem. I made three codes to solve it. The first one is to process the documents. It calculates for each word a vector of 800 positions where $v[i]$ counts how many times this word has appeared in the first i documents. As there are only 214429 different words it's a pretty small file (my Debian says 432,7 MiB). But calculating the answer with this file is not instantaneous, so let's optimize a little more!

Create a second code to read this file and create a second file containing each word and in which byte appears in the first file. This file is much smaller (3,9 MiB).

Then we code the final solution. First of all, read the second file entirely and store this values in some structure. Then, for each word of the input, open the first file, move the pointer to the place wanted and read the 800 positions vector. Now we can do a binary search over the vector to see in which document will be the word we are looking for. Finally, just read this document until you find the word. As we are reading only one document at most per word, the answer to the testcases is instantaneous.

10 Challenge 10: Coding m00re and m00re

The first problem which is not algorithmic. But this is very easy. With the examples one can deduce that it is a stack-based language and that the operations are the following:

- Number: Push the number at the top of the stack.
- mirror: Changes the sign of the number at the top of the stack.
- breadandfish: Duplicates the number at the top of the stack.
- fire: Pop the number at the top of the stack.
- dance: Pop the two first elements and push them swapped.
- conquer: Modulus operation.
- #: Multiplication operation.
- $: Subtraction operation.
- @ : Addition operation.
- &: Integer division operation.
- .: Print the number at the top of the stack.

The only important point is that there are not constraints, so a Big Integer library is essential (if your language does not handle them natively, as C++).

Code 9: Solution to problem 10

```
ll toInt(string& s) {
    stringstream ss(s);
    ll x;
    ss >> x;
    return x;
}

int main() {
    string s;
    while (getline(cin, s)) {
        stringstream ss(s);
        stack<BI> S;
        while (ss >> s) {
            if (isdigit(s[0])) { //Mete el numero en la pila
                BI x(s);
                S.push(x);
            }
```

```
            else if (s == "mirror") { //Cambia el signo del primero
                BI x;
                x = S.top();
                BI aux = -1;
                x *= aux;
                S.pop();
                S.push(x);
            }
            else if (s == "breadandfish") S.push(S.top()); //Replica el primero
            else if (s == "fire") S.pop(); //Hace pop
            else if (s == "dance") { //Hace swap a los dos primeros
                BI a = S.top(); S.pop();
                BI b = S.top(); S.pop();
                S.push(a); S.push(b);
            }
            else if (s == "conquer") {
                BI a = S.top(); S.pop();
                BI b = S.top(); S.pop();
                b %= a;
                S.push(b);
            }
            else if (s == "#") {
                BI a = S.top(); S.pop();
                BI b = S.top(); S.pop();
                a *= b;
                S.push(a);
            }
            else if (s == "$") {
                BI a = S.top(); S.pop();
                BI b = S.top(); S.pop();
                b -= a;
                S.push(b);
            }
            else if (s == "@") {
                BI a = S.top(); S.pop();
                BI b = S.top(); S.pop();
                a += b;
                S.push(a);
            }
            else if (s == "&") {
                BI a = S.top(); S.pop();
                BI b = S.top(); S.pop();
                b /= a;
                S.push(b);
            }
            else if (s == ".") {
                BI aux = S.top();
                cout << aux << endl;
                S.pop();
            }
        }
    }
}
```

11 Challenge 11: Descrambler

And we return to the algorithmic problems. Notice that the order of the letters do not affect to handle the word. So just consider words as vectors in $\mathbb{N}^{26}$, in which each position counts the number of times this letter appears in the word. As the number of words in the dictionary is small, iterate over it and for each word check if it can be played.

Code 10: Solution to problem 11

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

int qtt[30], qtt2[30], qtt3[30];
int p[] = {1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1,
           3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10};

int puntos(string& s) {
    int res = 0;
    for (int i = 0; i < s.size(); ++i) res += p[s[i] - 'A'];
    return res;
}

bool can(string& a) {
    memset(qtt3, 0, sizeof(qtt3));
    for (int i = 0; i < a.size(); ++i) ++qtt3[a[i] - 'A'];
    bool usat = false, puc_usar = false;
    for (int i = 0; i < 30; ++i) {
        if (qtt3[i] > qtt[i]) {
            if (qtt3[i] == qtt[i] + 1 and qtt2[i] > 0 and not usat) {
                usat = true;
                puc_usar = true;
            }
            else return false;
        }
        else {
            if (qtt2[i] > 0) puc_usar = true;
        }
    }
    if (puc_usar) return true;
    return false;
}

//Para cada palabra del diccionario, cuento cuantas letras
//tiene de cada tipo y miro si la puedo generar con las que tengo
//Tiempo O(|diccionario|)
int main() {
    int casos;
    cin >> casos;
    for (int cas = 1; cas <= casos; ++cas) {
        ifstream fin("/home/alex/codigos/tuenti/Challenge11/descrambler_wordlist.txt");
        string s, r, w;
        cin >> s >> r;
        memset(qtt, 0, sizeof(qtt));
```

```
        memset(qtt2, 0, sizeof(qtt2));
        for (int i = 0; i < s.size(); ++i) ++qtt[s[i] - 'A'];
        for (int i = 0; i < r.size(); ++i) ++qtt2[r[i] - 'A'];
        int best = 0;
        string res;
        while (getline(fin, w)) {
            if (can(w)) {
                int punts = puntos(w);
                if (punts > best) {
                    best = punts;
                    res = w;
                }
            }
        }
        fin.close();
        cout << res << " " << best << endl;
    }
}
```

12 Challenge 12: Three keys one cup

The first two keys are easy to find. The first one is in the QRs. The second one is in a comment inside the PNG file, you can see it with an hexadecimal editor. Then look at the keys! They are obviously MD5 hashes. If you crack them, you will have a revelation about the third key. The plain text of the first two keys is "courage" and "wisdom". As Zelda is pretty famous, just take the hash of "power" and it works :).

Code 11: Solution to problem 12

```
#include <iostream>
using namespace std;

string key1 = "a541714a17804ac281e6ddda5b707952";
string key2 = "ed8ce15da9b7b5e2ee70634cc235e363";
string key3 = "62cd275989e78ee56a81f0265a87562e";

string T = "0123456789abcdef";

int f(char c) {
    if (isdigit(c)) return c - '0';
    return c - 'a' + 10;
}

int main() {
    string input, res;
    cin >> input;
    for (int i = 0; i < input.size(); ++i)
        if (isalpha(input[i])) input[i] = tolower(input[i]);
    for (int i = 0; i < key1.size(); ++i) {
        int x = (f(key1[i]) + f(key2[i]) + f(key3[i]) + f(input[i]))%16;
        res += T[x];
    }
    cout << res << endl;
}
```

13 Challenge 13: The crazy croupier

A beautiful problem about Group Theory. It is easy to see that the shuffle can be seen as a permutation of S_n (the symmetric group). Let σ be the permutation. They are asking us to find the minimum $k > 0$ such that $\sigma^k = Id$ (identity application). This by definition is the order of the permutation, and can be calculated representing σ as disjoint cycles and then the answer is the least common multiple of the sizes of the cycles. This leads to a linear solution. Another time, it is necessary to use Big Integers.

Code 12: Solution to problem 13

```
int n, l;

void shuffle(vector<int>& v) {
    int i = l - 1, j = n - 1;
    vector<int> aux;
    while (i >= 0 and j >= l) {
        aux.push_back(v[i]);
        aux.push_back(v[j]);
        --i; --j;
    }
    while (i >= 0) aux.push_back(v[i--]);
    while (j >= l) aux.push_back(v[j--]);
    v = aux;
}

BI mcm(BI& a, BI& b) {
    BI c, d = a;
    c.mcd(a, b);
    d /= c;
    d *= b;
    return d;
}

int main() {
    int casos;
    cin >> casos;
    for (int cas = 1; cas <= casos; ++cas) {
        cin >> n >> l;
        vector<int> v(n);
        for (int i = 0; i < n; ++i) v[i] = i;
        shuffle(v);
        vector<bool> vist(n, false);
        BI res = 1;
        for (int i = 0; i < n; ++i) {
            if (not vist[i]) {
                vist[i] = true;
                int len = 1, x = i;
                while (not vist[v[x]]) {
                    vist[v[x]] = true;
                    x = v[x];
                    ++len;
                }
                BI len2 = len;
                res = mcm(res, len2);
            }
```

```
        }
        cout << "Case #" << cas << ": " << res << endl;
    }
}
```

14 Challenge 14: Nails

Easy problem when you discover what to do. Simply apply Hamming (7,4) to decode the input and check for errors as the test says.

15 Challenge 15: Newspaper code

The image uses steganography to hide the answer. There are some invisible points under some of the letters. Once you find them, a revealing sentence tells you to print the 20th emirps.

16 Challenge 16: Malware detector

A strange problem, as it allows you to do different correct solutions. My solution was for each query calculate the mean of the inverses of the distances for each of the two sets. The inverse allows to benefit the nearest points. Then check which of the two means is greater.

Code 13: Solution to problem 16

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
const double EPS = 1e-6;

double norma(VD& v) {
    double res = 0;
    for (int i = 0; i < v.size(); ++i) res += (v[i]*v[i]);
    return sqrt(res);
}

double dist(VD& a, VD& b) {
    double res = 0;
    for (int i = 0; i < a.size(); ++i) res += ((a[i] - b[i])*(a[i] - b[i]));
    return sqrt(res);
}

//Agrupo malos y buenos y por cada proceso desconocido,
//busco si esta en promedio mas cerca de los buenos o de los malos
int main() {
    int r, u, n;
    cin >> r >> u >> n;
    VVD S, M;
    for (int i = 0; i < r; ++i) {
        char c;
        cin >> c;
        VD v(n);
        for (int j = 0; j < n; ++j) cin >> v[j];
        double norm = norma(v);
        if (norm > EPS) for (int j = 0; j < n; ++j) v[j] /= norm;
        if (c == 'S') S.push_back(v);
        else M.push_back(v);
    }
    int res = 0;
    for (int i = 0; i < u; ++i) {
        VD v(n);
        int sum = 0;
        for (int j = 0; j < n; ++j) {
            cin >> v[j];
            sum += v[j];
        }
        double norm = norma(v);
        if (norm > EPS) for (int j = 0; j < n; ++j) v[j] /= norm;
        double d1 = 0, d2 = 0;
        for (int j = 0; j < S.size(); ++j) d1 += (1.0/dist(v, S[j]));
        for (int j = 0; j < M.size(); ++j) d2 += (1.0/dist(v, M[j]));
```

```
51              if (S.size() > 0) d1 /= double(S.size());
                if (M.size() > 0) d2 /= double(M.size());
                if (d1 < d2) res += sum;
            }
            cout << res << endl;
56      }
```

17 Challenge 17: The Solomonic pizza cut

A good geometric library is very useful to solve this problem. First, generate the vertexs of the polygons. It is easy if you have a function that lets you rotate vectors. Notice that if a solution exists, then exists a cut passing through the center and some vertex. So a simple solution due to the low constraints is just iterate over the points and for each cut, check if it is correct in linear time. There are better approaches. If N is the total number of vertexs, a $O(N \log N)$ solution sorting the points by angle can be done, instead of a quadratic one.

18 Challenge 18: SOP (Sheep Oriented Programming)

Notice that sheeps do not even have fingers, so that about sheeps being the best programmers is just difficult to believe. As an open-minded person, take a sheep and let it your keyboard (an old one better) and you will check that it is true that sheeps do not make bugs, because your lovely plant attracts them more than a keyboard. So what to do? Check that the statement talks about other animals: cows, orangutans and fish. As cows are difficult to handle in a room and aquatic keyboards are expensive, take an orangutan to your room and show it this code. Its answer would be immediat: "Ook!". So take an Ook! interpreter and you will be asked to make a simple program whose solution is simply $\frac{n(n+1)}{2} + 1$.

19 Challenge 19: Find the algorithm

First you can see that decoding both input and output in BASE64 produces an hexadecimal text. Now let's see the pattern. As the numbers in the input are very similar if you take them in groups of 8 characters, a logical move would be to look at the differences between them. Playing with it some time, you find out that the algorithm is the following:

- Calculate the difference between the actual block and the previous one (0 if it is the first).
- If the difference fits in 5 bits (is between -16 and 15) then put a 0 bit and the difference with 5 bits.
- Else put a 1 bit and the block number with 32 bits.
- Add 0s to the binary string until its length modulo 4 is 0.
- Convert it to hexadecimal.