Tuenti Programming Challenge 3
Solutions and code snippets

Enric Cusell (cusell@gmail.com)

May 9, 2013

Contents

1 Challenge 1: Bitcoin to the future

A first approach to this problem may suggest us to try every possiblity, which is for every day either sell all, buy all, or do nothing. Given a number of days n, the complexity is $O(3^n)$. This can be optimized using memoization as in the following solution, which runs instantly on the given test and submit cases.

Code 1: Backtracking with memoization

```
Vll v;
typedef pair<ll,ll> Pl;
typedef pair<int, Pl> PP;
map<PP, ll> memo;

ll f(int p, ll eur, ll btc) {
  if (p==v.size()) return eur;
  PP m = PP(p,Pl(eur,btc));
  if (memo.count(m)) return memo[m];
  ll ans = 0;
  //v[p] can't be zero, that would yield to infinity revenue
  ans=f(p+1,eur,btc); //ignore price
  ans=max(ans, f(p+1,eur%v[p], btc+eur/v[p])); //spend all in btc
  ans=max(ans, f(p+1, eur+btc*v[p], 0)); //sell all
  return memo[m] = ans;
}
```

If we think further we can realize there's a way more efficient greedy algorithm which solves it in $O(n)$, where n is the number of days. It consists of buying when the price is at a local minimum and selling when the price is at a local maximum.

Code 2: Greedy

```
for (int i = 0;i < n; ++i) {
  if ((i-1 < 0 or v[i-1] >= v[i]) and (i+1 >= n or v[i] <= v[i+1])) {
    //min
    btc+=eur/v[i];
    eur%=v[i];
  }
  else if ((i-1 < 0 or v[i-1] <= v[i]) and (i+1 >= n or v[i] >= v[i+1])) {
    //max
    eur+=btc*v[i];
    btc=0;
  }
  if(i+1 == n) eur+=btc*v[i]; //sell all at the end
}
```

2 Challenge 2: Did you mean...?

Given a dictionary file and a several query words, give for each of those a list of all the dictionary words which are its anagrams. We just have to use each sorted word as a canonical representation and any standard data structure available in the language, for example maps in C++.

Code 3: Second problem anagram map creation

```
typedef vector<string> Vs;
map<string, Vs> dic;
while(fin>>s) {
  w=s;
  sort(s.begin(),s.end());
  dic[s].push_back(w);
}

for(map<string, Vs>::iterator it=dic.begin();it!=dic.end();++it) {
  sort(it->second.begin(), it->second.end());
}
```

3 Challenge 3: Lost in Lost

Given information about several stories, and which ones could happen before or after the other ones, we had to tell if there were none, multiple, or just one ordering. In that case we had to tell which one. For example $.a < b > c$ meant that b happens before a and c happens after a. We are asked to calculate a topological sorting of the graph and distinguish if it's unique or not.

Code 4: Topological sorting

```
typedef vector<int> Vi;
Vi topological_sorting(Mi& G) {
  int n = int(G.size());
  Vi gin(n,0); //gin[i] = how many input edges does node i have
  Vi topo(n,0); // topo will contain the resulting sorted nodes.
  Vi placed(n,0);
  int pos = 0;
  for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j) {
    if (G[i][j]) gin[j]++;
  }
  queue<int> q;
  for (int i = 0; i < n; ++i) if (gin[i]==0) {
    q.push(i);
  }

  while(!q.empty()) {
    int p = q.front(); q.pop();
    if (placed[p]) continue;
    topo[pos++] = p;
    placed[p] = 1;
    for (int i = 0; i < n;++i) if (G[p][i]>0) {
      G[p][i]--;
      gin[i]--;
      if (gin[i]==0) q.push(i);
    }
  }
  if (pos<n) { //there is no topological sorting
    Vi empty;
    return empty;
  }
  return topo;
}
```

To distinguish wether the sorting was unique or not, we could use the fact that: The topological sorting is unique $\iff$ It is a hamiltonian path. Which can be trivially checked.

4 Challenge 4: Missing numbers

Given a file containing all the numbers from 0 to $2^{31}-1$ except 100 numbers, you were asked to find the i-th missing. Given the magnitude of the file (more than 8Gb) this problem obviously requires precomputation. The numbers were stored as bytes in the file, not represented as ASCII characters, so you had to read them taking into account it was stored by a little-endian machine. There was no need at all to sort the input or do any fancy stuff, creating an array of *bool* (memory optimized) or *char* (time optimized) of length 2^{31} starting with all zeros and fill it with ones was enough.

Code 5: Solution to problem 4

```
const ll MAX = 1LL<<31;
char v[MAX];
// [...]
ifstream fin;
fin.open("integers", ios::binary);
while (not fin.eof()) {
  fin.read((char*)&x, sizeof(int));
  v[x]=1;
}
fin.close();
for (ll i=0;i<MAX;++i) if (not v[i]) cout << i << ",";
```

5 Challenge 5: Dungeon Quest

Given three movements in a grid in a total of 20 steps, except for the first one which could have 4 different moves leads us to 4×3^{19} possible states worst case. That is around 4.649.045.868 and may take several seconds even on a modern computer. The standard backtracking with a few prunes could solve all the cases fast enough.

An alternative *more complex* solution would be to use the dijkstra algorithm in a state graph, where each node represents a state containing: (steps left, position on the grid, last taken direction, set of visited coins). Adding an heuristic or pruning could also drastrically improve the performance. An example would be to stop if the achieved value plus 5 times (maximum coin value) the number of remaining steps wasn't larger than the already achieved maximum, then stop. This directs the search and stretches the search space. In the average case it is way faster, even though it could be as bad as the standard backtracking in a worst theoretical case.

The last solution is not included because it may result confusing, and the first one is very straightforward.

6 Challenge 6: Ice Cave

Even though some people used tricks or workarounds to solve this problem and managed to pass the specific test and submit cases, the proper solution is as follows. It's a standard application of the Dijkstra Algorithm. There was also the trick that what seemed to be dots were interpuncts (· or U+00B7).

Code 6: Dijkstra for problem 6

```
typedef pair<int,int> P;
typedef pair<double,P> dP;
int di[4]={-1,0,1,0};
int dj[4]={0,-1,0,1}; //NWSE
//[...]
double dijkstra(int i0, int j0) {
  if (T[i0][j0]=='O') return 0;
  priority_queue<dP> q;
  q.push(dP(0,P(i0,j0)));
  dist[i0][j0]=0;

  while(!q.empty()) {
    double c = -q.top().first;
    int i = q.top().second.first;
    int j = q.top().second.second;
    q.pop();
    if (T[i][j]=='O') return c;
    if (c > dist[i][j]) continue;
    for(int d = 0; d < 4; ++d) {
      int ni = i;
      int nj = j;
      int r = 0;
      while (ni>=0 and nj>=0 and ni<n and nj<m and T[ni][nj]!='#') {
        ni+=di[d];
        nj+=dj[d];
        ++r;
      }
      //Crashed. One step back.
      ni-=di[d];
      nj-=dj[d];
      --r;
      if (r==0) continue; //Ain't staying here.
      double ndist = c + zt + double(r)/zs;
      if (ndist < dist[ni][nj]) {
        dist[ni][nj] = ndist;
        q.push(dP(-ndist,P(ni,nj)));
      }
    }
  }
  return -1; //Can't exit.
}
```

7 Challenge 7: Boozzle

This problem consisted on two parts. For every test case:

- Iterate through the possible 178378 dictionary entries to calculate the maximum attainable score. Since the words maximum length is 15, any backtracking works here.
- Once you have the cost in seconds to write each attainable word, and the points it will give, all we need to solve is a classical Knapsack dynamic programming problem.

Code 7: Knapsack for problem 7

```
//Knapsack with weights "pes" and values "pts" and only using two rows.
vector<int> dplast(w+1,0),dp(w+1,0);
for (int i = 0; i < k; ++i) {
  dp = vector<int>(w+1,0);
  for (int j = 0; j < w+1; ++j) {
        if (j>=pes[i]) dp[j] = max(dplast[j], dplast[j-pes[i]] + pts[i]);
        else dp[j] = dplast[j];
  }
  dplast=dp;
}
int mx=0;
for (int i = 0; i < w + 1; ++i) mx = max(mx, dp[i]);
cout << mx << endl;
```

8 Challenge 8: Tuenti Timing Auth

In this program we had to figure out a way to get privilege access to a php site. Messing with */dev/urandom/* was way too complex, so the solution was to get the expression $strcmp(\$passwd, \$\_POST[\text{"}pass\text{"}]) == 0$ evaluate true. It's easily done using an array as pass POST parameter. *strcmp* will return NULL and $NULL == 0$ will evaluate to true (opposed to ===). This can be done for example using CURL, or an alternative very elegant solution: Inspect the code with your browser and change it as shown below. The other field key had to contain the input data.

Code 8: Solution to problem 8

```
<input type="text" name="pass">
<input type="text" name="pass[]">
```

9 Challenge 9: Defenders of the Galaxy

Simulating step by step would exhaust the time limit, so the solution was figuring out the formula to calculate the number of survival days given a fixed amount of soldiers. Then, iterate through the number of soldiers (or Crematoriums) and get the maximum.

Code 9: Solution for problem 9

```
typedef long long ll;
typedef long double ld;

ll fraction_ceil(ll a, ll b) {
  return floor(ld(a)/ld(b)+1)+1;
}

int main() {
  int t; cin >> t;
  while(t--) {
    ll w, h, s, c, g;
    cin >> w >> h >> s >> c >> g;
    //Each second, w extra enemies.
    if (w*s<=g) cout << -1 << endl; //can buy w soldiers
    else{
      ll ans=0;
      for (ll arc=0;arc<=w and arc*s<=g;++arc) { //How many soldiers.
        ll numb = (g-s*arc)/c;
        ll k = fraction_ceil(w*(h-1), w-arc);
        ans=max(ans, (numb+1)*(k-1));
      }
      cout << ans << endl;
    }
  }
}
```

10 Challenge 10: The Checking Machine

Given the 32 length character outputs, the md5 hashing was suspicious. The only thing left to discover was what to do with the letters, numbers and brackets. It turned out to be pretty intuitive, since letters meant concatenation and numbers prefixing square brackets were meant to repeat the interior content as many times as the number. So for example: abcd3[b] would expect md5(abcdbbb) as output. It turns out that you can generate exponentially long strings with this notation, and the cases containted both a case with 2^{34} and 13! a's.

Recursively generating and storing the whole strings is impractical, so what immediately pops out of one's head is to use a MD5 library that allows string updates or concatenation. The following code uses the openssl implementation. Moreover, there's an optimitzation with the number of calls done to md5 update, using a common technique called buffering. As well as a memoization used to find the corresponding square brackets, digits and characters when calling to compute a substring $[a, b]$ more than once. Take into account that the code may seem overly complicated mainly because of the optimizations, but this can be coded in a simpler (*and slower!*) way.

Code 10: Solution to problem 10

```
typedef pair<int,int> P;
typedef long long ll;
string s;
int n;
vector<vector<int> > memole,memori,memox,memoy;

P split(int a, int b) {
  P ans = P(0,0);
  bool dig=0;
  for (int i = a; i <=b; ++i) {
    if (isdigit(s[i])) dig=1;
    if (dig) {
      ans.second*=10;
      ans.second+=s[i]-'0';
    }
    else ++ans.first; //add s[i]
  }
  return ans;
}

char *buffer;
const int BN = 1024*1024;
int bp; //position buffer

void clearbuffer(MD5_CTX& c) {
  MD5_Update(&c, buffer, bp);
  bp=0;
}
```

```
30 void update(MD5_CTX& c, const char *str, ll n) {
     if (bp+n<BN) for (int i=0;i<n;++i) buffer[bp++]=str[i];
     else {
       clearbuffer(c);
       MD5_Update(&c, str, n);
35   }
   }

   //Pre: Before [, there's always a digit.
   void solve(MD5_CTX& c, const char *str, int a, int b) { //[a,b]
40   if (b<a) return;
     int le, ri, x, y;
     if (memox[a][b]==-1) {
       int i, ct=1; //[ = +1, ] = -1
       for (i=a;s[i]!='[' and i<=b;++i);
45     P p = split(a,i-1);
       x = p.first;
       y = p.second;
       le=i;
       if (s[le]!='[') { //No opening bracket
50       i=b+1;
         le=ri=-1;
       }
       ri=-1;
       //Looking for the corresponding closing bracket.
55     for (i=le+1;i<=b and ct!=0;++i) {
         if (s[i]==']') --ct;
         else if (s[i]=='[') ++ct;
         if (ct==0) ri=i;
       }
60     memole[a][b] = le;
       memori[a][b] = ri;
       memox[a][b] = x;
       memoy[a][b] = y;
     }
65   else {
       le = memole[a][b];
       ri = memori[a][b];
       x = memox[a][b];
       y = memoy[a][b];
70   }
     update(c, str+a, x);

     if (le==-1) return;
     //[le .... ri]
75   for (int j = 0; j < y; ++j) {
       solve(c, str, le+1, ri-1);
     }
     solve(c, str, ri+1, b);
   }
```

11 Challenge 11: The Escape from Pixel Island

Given two trees where nodes were either tree leaves or had four children, they gave you a canonical image representation which you had to add (*OR* operation). This would be a very basic data structures problems if it wasn't because you had to find out the output they expected. After generating the tree, you had to generate the image and it turned out to be a QR code, which contained the *secret message*. This is my personal favourite. So simple yet so entertaining!

Observation: This could be solved skipping the part of the tree OR operation, since you could simply overlap the images with some images tool.

Image 1: First case QR code

Below, the codes for tree creation, tree adding and tree drawing.

Code 11: Problem 11 - Tree creation

```
struct Node{
  char col; //w(white), b(black), p(recursive)
  int a,b,c,d; //index to the subtrees, or -1 if it doesn't have
  int dad; //index from it's parent, or -1 if it's the root
  Node () { //default constructor
    col='?';
    a=b=c=d=-1;
    dad=-1;
  }
};

vector<Node> createTree(string s) {
  vector<Node> v(0);
  Node nd0;
  v.push_back(nd0); //root node
```

```
  queue<int> q; //BFS creation
  q.push(0);
  while(!q.empty()) {
    int p = q.front(); q.pop();
    v[p].col = s[p];
    Node nd = nd0; nd.dad = p;

    if (s[p]=='p') {
      v[p].a = v.size();
      q.push(v.size());
      v.push_back(nd);

      v[p].b = v.size();
      q.push(v.size());
      v.push_back(nd);

      v[p].c = v.size();
      q.push(v.size());
      v.push_back(nd);

      v[p].d = v.size();
      q.push(v.size());
      v.push_back(nd);
    }
  }
  return v;
}
```

Code 12: Problem 11 - Tree sum

```
vector<Node> sumaTree(vector<Node> va, vector<Node> vb) {
  vector<Node> v(0);
  Node nd0;
  v.push_back(nd0); //root node

  vector<int> wa(0),wb(0);
  wa.push_back(0); wb.push_back(0);
  //wa[i] which "va" node is v[i] associated to
  //wb[i] which "vb" node is v[i] associated to

  queue<int> q;
  q.push(0);
  while(!q.empty()) {
    int p = q.front(); q.pop();
    Node nd = nd0; nd.dad = p;
    char ca = '?',cb='?';
    if (wa[p]!=-1) ca = va[wa[p]].col;
    if (wb[p]!=-1) cb = vb[wb[p]].col;
    //Both can't be -1 at the same time.

    if (ca=='b' or cb=='b') v[p].col='b';
    else if (ca=='w' and cb=='w') v[p].col='w';
    else if (ca=='?' and cb!='?') v[p].col=cb;
    else if (ca!='?' and cb=='?') v[p].col=ca;
    else if (ca=='?' and cb=='?') cerr << "woops.." << endl;
    else v[p].col='p';

    if (v[p].col=='p') { //Four child. For more, could be generalized.
      v[p].a = v.size();
      q.push(v.size());
      v.push_back(nd);
```

```
      if (wa[p]!=-1) wa.push_back(va[wa[p]].a);
      else wa.push_back(-1);
      if (wb[p]!=-1) wb.push_back(vb[wb[p]].a);
      else wb.push_back(-1);

      v[p].b = v.size();
      q.push(v.size());
      v.push_back(nd);
      if (wa[p]!=-1) wa.push_back(va[wa[p]].b);
      else wa.push_back(-1);
      if (wb[p]!=-1) wb.push_back(vb[wb[p]].b);
      else wb.push_back(-1);

      v[p].c = v.size();
      q.push(v.size());
      v.push_back(nd);
      if (wa[p]!=-1) wa.push_back(va[wa[p]].c);
      else wa.push_back(-1);
      if (wb[p]!=-1) wb.push_back(vb[wb[p]].c);
      else wb.push_back(-1);

      v[p].d = v.size();
      q.push(v.size());
      v.push_back(nd);
      if (wa[p]!=-1) wa.push_back(va[wa[p]].d);
      else wa.push_back(-1);
      if (wb[p]!=-1) wb.push_back(vb[wb[p]].d);
      else wb.push_back(-1);
    }
  }
  return v;
}
```

Code 13: Problem 11 - QR drawing

```
//pinta(suma, 0, 0, 0, 640, 640);
//Prints the rectangle coordinates
void pinta(vector<Node>& v, int p, int i0, int j0, int i1, int j1) {
  if (v[p].col=='b') cout << i0 << " " << j0 << " " << i1 << " " << j1 << endl;
  else if (v[p].col=='p') {
    int pi = (i0+i1)/2;
    int pj = (j0+j1)/2;
    pinta(v, v[p].b, i0, j0, pi, pj);
    pinta(v, v[p].d, pi, pj, i1, j1);

    pinta(v, v[p].a, pi, j0, i1, pj);
    pinta(v, v[p].c, i0, pj, pi, j1);
  }
}

//Python code which painted the rectangles
N = 640
im = Image.new("RGB", (N, N), "white")
draw = ImageDraw.Draw(im)

for line in sys.stdin:
        box = [int(n) for n in line.split()]
        if box[0]!=-1:
        draw.rectangle(box, fill=0)
        else:
            break
```

12 Challenge 12: Whispering paRTTY

After you decoded the *base*64 given in the input, you were left with an *mp*3 file, which was simply a Radioteletype (RTTY) encoded message using Baudot code. All could be solved using standard existing tools.

Code 14: Solution to problem 12

```
base64 -d <party.in >party.mp3
lame --decode --quiet party.mp3 party.wav
minimodem --rx -q -f party.wav rtty
```

13 Challenge 13: Sparse randomness

Given a constant list of non-strictly ascending integers, for many queries which consist of two subindexes, give the maximum number of same occurrences of a number in the sublist. From now on, we will use n as the list length and q for the number of queries. There's a first naive solution which consists in iterating over the sublist for each query and would lead us to a cost of $O(k * n)$, which should be too slow.

There's a solution using the interval tree structure which has a $O(log(n))$ complexity for each query. As a curiosity, the following implementation would even allow the problem to modify the original list in between queries. First of all, we will create a list of how many repeated numbers of each kind there are, so for example for an input list $v = \{2, 2, 3, 5, 5, 5\}$, it would be $vlen = \{2, 1, 3\}$.

Given a subindexes query i, j, those subindexes will split two sequences of repeated numbers in two parts, for which we can calculate how many of each are left inside $[i, j]$. For the other repeated sequences contained strictly inside the indexes (which can be found having extra arrays containing information about the start and ending of the sequences), a $O(logn)$ standard query in the built interval tree will suffice.

Code 15: Solution to problem 13 using interval tree

```
typedef long long ll;
typedef vector<ll> Vl;
typedef map<ll , ll> M;

Vl T;
int n,sz;
const ll INF = 1000000000000000000LL;

//[a,b) node interval
//[x,y) query interval
ll getval(int p, int a, int b, int x, int y) {
    if (b<=x or a>=y) return -INF;
    if (a>=x and b<=y) return T[p];
    return max(getval(2*p, a,(a+b)/2,x,y), getval(2*p+1, (a+b)/2,b,x,y));
}

//Returns max(v[x], .., v[y]). [x,y]
ll getval(int x, int y) {
  return getval(1,0,sz,x,y+1);
}

void set(int p, ll val) {
    p+=sz;
    T[p]=val;
    for(p/=2;p>0;p/=2) T[p]=max(T[2*p],T[2*p+1]);
}

```

```
void createTree(Vl& v) {
  n = v.size();
  sz=1;
  while (sz<n) sz*=2;
  T = Vl(2*sz,0);
  //Could be done in O(n) instead of O(n logn)
  //but it suffices by far for 10^6
  for (int i=0;i<n;++i) set(i,v[i]);
}

int main() {
  int t; cin >> t;
  for (int cas=1;cas<=t;++cas) {
    int vn, m;
    cin >> vn >>m;
    Vl v(vn);
    for (int i = 0; i < vn; ++i) cin >> v[i];
    v.push_back(INF); ++vn;
    M sig, ant, ini, fi, mv;
    //Pre: v increasing!

    Vl vlen(0);
    int beg=0, prenum=-1;
    for (int i=1;i<vn;++i) { //uses the last INF>all
      if (v[i]>v[i-1]) {
        int k = v[i-1];
        vlen.push_back(i-beg);
        mv[k] = int(vlen.size())-1; //where is his maximum on the vlen vector
        ant[k] = prenum;
        if (v[i]==INF)  sig[k]=-1;
        else sig[k] = v[i];
        ini[k] = beg;
        fi[k]=i-1;
        prenum=k;
        beg=i;
      }
    }
    createTree(vlen);

    cout << "Test case #" << cas << endl;
    int a, b;
    while(m--) {
      cin >> a >> b;
      --a; --b;
      ll ans=0,ret;
      ll x = sig[v[a]], y = ant[v[b]];
      if (v[a]!=v[b] and x!=-1 and x!=v[b] and
          y!=-1 and y!=v[a]) {
        ret = getval(mv[x], mv[y]);
        ans=max(ans, ret);
      }
      ans=max(ans,fi[v[a]]-a+1);
      ans=max(ans,b-ini[v[b]]+1);
      cout << ans << endl;
    }
  }
}
```

14 Challenge 14: Ovine Cryptography

Inspectioning the given 13 encrypted texts, we can observe there are quite a few similarities for every $i-th$ byte. Turns out that way because XOR cipher has been used. If it concainted encrypted random bytes, it would be impossible to solve, so we can safely assume it will contain readable text. It's widely known that frequency analysis can break XOR cipher. Every byte is independent, so for every byte we could iterate through all the 256 possible keys and take the one that left more of the 13 sentences with readable characters ($a-z, A-Z, spaces, ...$). This could be improved using a frequency table for all the characters, but this simple approach was enough to have almost readable sentences. Manually reconstructing the longest one lets us get the full key.

Code 16: Problem 14 - Approximation XOR cipher attack

```
//Check char by char which key (from 256) generates most a-zA-Z and spaces.
vector<int> bestkey(MAXCHAR,0);
for (int i=0;i<MAXCHAR;i+=2) {
  int azkey=0;
  for (int key=0;key<16*16;++key) {
    int az=0;
    for (int j=0;j<k;++j) if (i<vs[j].size()){
      char c = char((val(vs[j][i])*16+val(vs[j][i+1]))^key);
      if ((c>='a' and c<='z') or (c>='A' and c<='Z') or c==' ') ++az;
    }
    if (az>azkey) {
      azkey=az;
      bestkey[i]=key;
    }
  }
}
for (int j=0;j<k;++j) {
  for (int i=0;i<MAXCHAR;i+=2) {
    char c = (val(vs[j][i])*16+val(vs[j][i+1]))^bestkey[i];
    cout << c;
  }
  cout << endl;
}
```

15 Challenge 15: The only winning move is not to play

Knowing the code was written using KATE, it must contain the $\sim$ character backup files. We can find all the source starting on *index.php* $\sim$

First of all, we had to find a secret, which consisted of 4 characters of the string $TUENTI$, which was easily obtainable by iterating through all the few possibilities. Turned out the secret was $IETN$.

The next step was to modify the cookie *game* which included the game state in base64 and an md5 hash of the state and the secret for security. The result key value was given as content in an already expired cookie.

Code 17: Problem 15 - php cookie modification solution

```
<?php
  $key = file_get_contents("php://stdin");
  $key = str_replace("\n","",$key);
  $dir = "/home/ttt/data/keys/".$key;
  $len = strlen($dir);
  //Any valid game cookie.
  $gamecookie = 'Tzo0OiJnYW1lIjozOntzOjEx...'; //Truncated for readability.
  $game = base64_decode($gamecookie);
  $game = str_replace("35",$len,$game);
  $game = str_replace("/home/ttt/data/messages/version.txt",$dir,$game);

  //Use the already found secret.
  $secret = "IETN";
  $gamestate = base64_encode($game);
  $h = md5($gamestate . $secret);
  $cookie = urlencode($gamestate). "%7C" . $h;

  $curl_url = "http://ttt.contest.tuenti.net/";
  $curl_cookie = "game=".$cookie;
  $curl_r = curl_init();
  //Set that headers response is wanted
  curl_setopt($curl_r, CURLOPT_RETURNTRANSFER, 1);
  curl_setopt($curl_r, CURLOPT_HEADER, 1);
  //Set URL and cookie.
  curl_setopt($curl_r,CURLOPT_URL, $curl_url);
  curl_setopt($curl_r,CURLOPT_COOKIE, $curl_cookie);
  $result = curl_exec($curl_r);
  curl_close($curl_r);
  //Get well-formatted output.
  preg_match_all('/^Set-Cookie:\s*([^;]*)/mi', $result, $m);
  $result = str_replace("X-Tuenti-Powered-By=","",$m[1][1]);
  echo $result."\n";
?>
```

16 Challenge 16: Legacy code

The problem defined a specific set of state transitions for a Turing machine. The first step to solve this problem was to simulate it, which is trivially done. After some tests, one could realize that it only accepted inputs like $\#(0and1)\#(0and1)\#...$

The number of steps the Turing machine could take was actually very large for relative small inputs. As a curiosity, the busy beaver function calculates the maximum number of steps given the length of nonblank symbols on tape for an arbitrary set of transitions, and it turns out to be a noncomputable function. In fewer words, the number of steps could be huge.

Trying out some examples the behaviour could be extrapolated, and it consisted of getting the first number in the # blocks and multiplying it by the sum of the rest of the blocks. If there was overflow, the machine wouldn't reach the *end* state, which was guaranteed on the statement, so you didn't have to worry about those cases. The length in bits of the resulting number was the length of the first block. In case of only one block with value x, the expected output was the value of $x - 1$. Once this is known, the implementation is trivial.

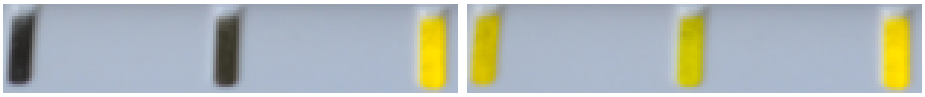
17 Challenge 17: Silence on the wire

The given video was created at 10fps, and it lasted 5min42.40sec, which leads us to 3424 frames. Splitting the video into frames lead us to more than 2Gb worth of png's, from which only the three leds were useful. The following bash code gets the frames and crops them, which leaves us with barely 18Mb of images.

Code 18: Problem 17 - Splitting into frames and cropping images

```
#!/bin/bash
ffmpeg -i video.avi -r 10 -f image2 image-%07d.png
for ((x=1;x<=3424;x+=1)); do
        mogrify -crop 156x30+406+332 image-`printf "%07d" $x`.png;
done
```

Image 2: Cropped images extracted from the video



Every frame contained either the three leds on, or the first two off, and the last on, so all we had to do was to read the binary values for each frame, write them as text, which showed an HTTP header easily reproduceable. In few words, we had to visit *silence.contest.tuenti.net* with the cookie $adminsession = true$ to see the real statement: "For a given n, write the sum of the digits of n!". This task can be done in a single python line:

Code 19: Problem 17 - Sum of digits of n!

```
print sum(int(d) for d in str(math.factorial(int(n))))
```

18 Challenge 18: Energy will be "infinities"

The problem is to find infinite cycles in a given graph. Appropriately changing the costs leads us to an isomorph problem where the classical Bellman-Ford algorithm can be applied. For a graph with V vertices and E edges, the cost will be $O(V \times E)$.

- The cost c for an edge given in a percentage [-20%, +20%] modification of the currenct cost can be transformed to: $-log(1 + c/100)$ so the problem reduces to *additive* costs instead of multiplicative. Now we are looking for *negative cycles*. This avoids precision errors as well, for example with $0.8^{50000} \cong 0$.
- The standard Bellman-Ford algorithm is defined from a source node. This can be patched by adding an extra node with edges to all the rest with cost 0.

Code 20: Problem 18 - multiplicative Bellman Ford from all sources

```
typedef vector<int> Vi;
typedef vector<Vi> Mi;
typedef vector<double> Vd;
typedef vector<Vd> Md;
const double INF = 1e50;
Mi G;
Md Gd;
Vd dist;

//Relax edge (u,v) with cost cost.
bool relax(int u, int v, double cost) {
  if (dist[u] != INF and dist[v] > dist[u] + cost) {
    dist[v] = dist[u] + cost;
    return true;
  } return false;
}

//Returns true if there's a negative cycle
bool bellmanford(int s) {
  int n = G.size();
  dist = Vd(n,INF);
  dist[s] = 0;
  for (int k = 0; k < n-1; ++k) {
    bool change = false;
    for (int i = 0; i < n; ++i) for (int j = 0; j < G[i].size(); ++j) {
      if (relax(i,G[i][j], Gd[i][j])) change = true;
    }
    if (!change) break;
  }

  for (int i = 0; i < n; ++i) for (int j = 0; j < G[i].size(); ++j) {
    if (relax(i, G[i][j], Gd[i][j])) return true;
  }
  return false;
```

```
}

int main(){
  int t; cin >> t;
  int n, m;
  while (t--) {
    cin >> n >> m;
    ++n; //Add the extra node
    Gd = Md(n);
    G = Mi(n);
    for (int i = 0; i <m; ++i) {
      int a, b;
      double c; //Statement doesn't say it's going to be integer.
      cin >> a >> b >> c;
      G[a].push_back(b);
      if (c!=0) Gd[a].push_back(-log(1.+c/100.));
      else Gd[a].push_back(0);
      //1+c/100 converts [-20%, +20%] to [0.8, 1.2]
      //log to maximize the product
    }
        //Edge from extra node to all for reachability.
    for (int i = 0; i < n-1; ++i) {
      G[n-1].push_back(i);
      Gd[n-1].push_back(0);
    }
    if (bellmanford(n-1)) cout << "True" << endl;
    else cout << "False" << endl;
  }
}
```

19 Challenge 19: Signal decode

The last thing I would have expected in a contest would be to create an Android app. That's precisely what was requested in this challenge. First of all, you had to receive the *SIGNAL_INFO* sent to *com.tuenti.signal*, which could be done by extending the *BroadcastReceiver* class. It's easy to start developing once you have the Android ADT Bundle for Eclipse. Decompiling the given *signal.apk* with any java decompiler may give you hints about what to do next.

Code 21: Problem 19 - AndroidManifest.xml BroadcastReceiver

```
<receiver android:name=".IncomingReceiver" android:enabled="true">
  <intent-filter>
    <action android:name="com.tuenti.signal"></action>
  </intent-filter>
</receiver>
```

Code 22: Problem 19 - BroadcastReceiver extension

```
public class IncomingReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
                Bundle extras = intent.getExtras();
                byte[] arrayB = extras.getByteArray("SIGNAL_INFO");
                //Store arrayB content
        }
}
```

For each input channel on the app, you were cyclically receiving 5 different arrays of bytes. When sorting them appropriately you had a jpg compressed image, containing a picture with a number written on it.

Image 3: Image received for channel 2

After that, all you had to do was to send it to their implementation of the ContentProvider

Code 23: Problem 19 - ContentReceiver query

```
Uri CODE_PROVIDER = Uri.parse(
  "com.tuenti.lostchallenge.datamodel.provider.ContestProvider/contest");
String queryArg[] = new String[1];
queryArg[0] = new String("158");
Cursor cur = getContentResolver().
        query(CODE_PROVIDER, null, "key = ?", queryArg, null);

//Print query results in logcat.
if (cur.moveToFirst()){
   do{
            String value = cur.getString(cur.getColumnIndex("value"));
            String id = cur.getString(cur.getColumnIndex("_id"));
            String key = cur.getString(cur.getColumnIndex("key"));
            Log.d("value", value);
            Log.d("id", id);
            Log.d("key", key);
   }while(cur.moveToNext());
}
cur.close()
```

Finally, when having the values for the 5 available channels, all you had to do is to hardcode it in a small program.

20 Challenge 20: Alien invasion

When decoding base64 input, we were left with an ELF file, which is the format linux uses for binary files. Turns out that it was protected to not allow ptrace/strace or any debugging common tools. So we had to crack it! Disassembling it led us to a few i386 ASM lines, which could be translated into something readable.

Code 24: Problem 20 - Disassembly (ASM)

```
                public start
start           proc near

var_1C          = dword ptr -1Ch
var_18          = dword ptr -18h
var_10          = dword ptr -10h
arg_0           = byte ptr  8

                push    ebp
                mov     ebp, esp
                push    edi
                push    esi
                push    ebx
                sub     esp, 10h
                mov     eax, [ebp+4]
                lea     edi, [ebp+arg_0]
                mov     [ebp+var_10], eax
                xor     edx, edx        ; addr
                mov     eax, 1Ah
                mov     ebx, edx        ; ptrace_request
                mov     ecx, edx        ; pid
                mov     esi, edx        ; data
                int     80h             ; LINUX - sys_ptrace
                test    eax, eax
                jnz     short loc_804813C
                cmp     [ebp+var_10], 2
                jnz     short loc_804813C
                mov     edi, [edi+4]
                mov     [ebp+var_1C], edi
                mov     [ebp+var_18], 539h
                jmp     short loc_804812B
; ---------------------------------------------------------------------------

loc_80480F2:                            ; CODE XREF: start+78j
                imul    eax, [ebp+var_18], 41A7h
                mov     [ebp+var_10], eax
                mov     esi, 7FFFFFFh
                xor     edx, edx
                div     esi
                mov     [ebp+var_18], edx
                mov     esi, edi
                sub     esi, [ebp+var_1C]
                cmp     esi, 1Ch
                ja      short loc_8048127
                and     edx, 0FFh
                xor     ecx, edx
                movzx   eax, byte_8048148[esi]
                cmp     ecx, eax
```

```
                jnz     short loc_8048127
                jmp     short loc_804812A
; ---------------------------------------------------------------------------

loc_8048127:                            ; CODE XREF: start+58j  start+6Bj
                or      ebx, 0FFFFFFFFh

loc_804812A:                            ; CODE XREF: start+6Dj
                inc     edi

loc_804812B:                            ; CODE XREF: start+38j
                movsx   ecx, byte ptr [edi]
                test    cl, cl
                jnz     short loc_80480F2
                sub     edi, [ebp+var_1C]
                cmp     edi, 1Ch
                jbe     short loc_804813C
                jmp     short loc_804813F
; ---------------------------------------------------------------------------

loc_804813C:                            ; CODE XREF: start+23j  start+29j ...
                or      ebx, 0FFFFFFFFh ; status

loc_804813F:                            ; CODE XREF: start+82j
                mov     eax, 1
                int     80h             ; LINUX - sys_exit
start           endp

; ---------------------------------------------------------------------------

loc_8048146:                            ; CODE XREF: .text:loc_8048146j
                jmp     short loc_8048146
; ---------------------------------------------------------------------------
byte_8048148    db 78h                  ; DATA XREF: start+62r
                db 0A3h, 65h, 55h
                dd 0DA90F5EDh, 685CDA54h, 0D675E1C8h, 867EB742h, 6592170Ah
                dd 7847AE0Ch
                db 0F7h
_text           ends


                end start
```

Translating it and manually fixing the default labels for more readable ones gives us this the following C code.

Code 25: Problem 20 - C translation

```
void __cdecl start(char a1, char *startpos)
{
  int result; // ebx@1
  char *currpos; // edi@3
  unsigned int v4; // edx@4
  char currchar; // cl@9
  int v6; // eax@13
  unsigned int v7; // [sp+4h] [bp-18h]@3
  void *argc; // [sp+20h] [bp+4h]@2

  result = 0;
  if ( sys_ptrace(result, 0, result, 0) )
```

```
    goto LABEL_17;
  if ( argc != 2 )
    goto LABEL_17;
  currpos = startpos;
  v7 = 1337;
  while ( 1 )
  {
    *&currchar = *currpos;
    if ( !currchar )
      break;
    v4 = 16807 * v7 % 0x7FFFFFF;
    v7 = 16807 * v7 % 0x7FFFFFF;
    if ( currpos - startpos > 28 ||
                (v4 ^ *&currchar) != data[currpos - startpos] )
      result = -1;
    ++currpos;
  }
  if ( currpos - startpos <= 28 )
LABEL_17:
    result = -1;
  v6 = sys_exit(result);
  JUMPOUT(loc_8048146);
}
```

We can guess it's applying XOR cipher in a encrypted string and the binary exits with code 0 only if the input string matches. Extracting the data from the assembly (taking into account it was done on a little-endian machine) we can create the following solver. The answer was "We're not hit! We're not hit!"

Code 26: Problem 20 - C++ solution

```
string cif = "78a36555edf590da54da5c68c8e175d642b77e860a1792650cae4778f7";

int val(char c) {
    if (c>='0' and c<='9') return c-'0';
    return c-'a'+10;
}

unsigned char val(char a, char b) {
    return (val(a)<<4)|val(b);
}

int main() {
    unsigned int v4, v7;
    vector<char> v;
    v7 = 1337;
    for (int i=0;i<cif.size();i+=2) {
        v4 = 16807 * v7 % 0x7FFFFFF;
        v7 = 16807 * v7 % 0x7FFFFFF;
        v.push_back( (unsigned char)(v4) ^ val(cif[i],cif[i+1]) );
    }
    for (int i=0;i<v.size();++i) cout << v[i];
    cout << endl;
}
```