HACKUPC: THE GAME
Solutions

March 27, 2016

Albert Martínez

Contents

1 Special Thanks

Special thanks go to all organizers from HackUPC that helped to create `The Game`, specially:

- Darío Nieuwenhuis and Dani Torramilans, for creating the awesome website.
- Miquel Llobet, Bernat Moix, Sergio Rodríguez and Antoni Virós, for collaborating in the problemset.
- Darío Nieuwenhuis, Gerard Orriols and Albert Vaca, for finding errors in the official solution during the contest.

2 Solutions

Here is the report with solutions for each problem from `The Game`. Some problems allowed different approaches to solve them. We saw people solving some problems in a clever way we did not expect. Here we show our proposed solutions.

All code with solutions is available at: `https://github.com/albertnez/the-game/`.

Level 0: The Game

What did you just lose?

This is a reference to **The Game**. Therefore, the answer is `thegame`. We made it explicit in the beginning that no answer would contain spaces or capital letters. Of course this didn't stop people from attempting to use them in the first problem, causing confusion.

The answer is `thegame`.

Level 1: Messy folders

We are given a file, which contains a lot of nested folders with lots of files. We are asked to find the password.

Instead of looking file by file, we can use `grep -r 'password'`, which will look for the string *password* in all files and subdirectories. With this we find that it matches with `password=1337p455w0rd`. During the contest we gave the hint to use grep to help contestants.

The answer is `1337p455w0rd`.

Level 2: Decoding

We are given a text and we are asked to find the password. The title suggests decoding the string, and the string is composed of hexadecimal characters only. If we decode the hex string, we find the next string:

`WkRGbE5UYzJZamN4WTJObFpqVTVOemhrTWpJeFptRmtaalJtTUdVeU9Ea2dJQzBLCg==`.

As it ends with `==`, this happens in base64 strings, so we decode base64 and find yet another string:

`ZDFlNTc2YjcxY2NlZjU5NzhkMjIxZmFkZjRmMGUyODkgICOK`.

This string does not give much information how to decode it. However, we could just try to do what we have already done. This is not a hex string, so we just try to decode as a base64 string to find the last string:

`d1e576b71ccef5978d221fadf4f0e289`.

It has 32 characters, and it is a md5 hash of the password. If we just try to look for that string in Google (or any other search engine), we easily find the original text in of the many MD5 lookup websites, and that is the password.

The answer is `superpassword`.

Level 3: XOR

Somebody did the XOR of all numbers from 0 to 1000000000 (inclusive), except one number K. Find this K.

We know that $a \oplus a = 0$. So if we apply the XOR of all numbers in the range to the given number, each one will be canceled, except K, that only appears once. Therefore we are left with the original number K. As 1000000000 is not that big, we can do the XOR of all numbers one by one in less than one second.

The answer is `133333337`.

Level 4: TAR

We are given a lot of nested compressed files, numbered starting at 512. A simple script can automatically extract everything until we get with the password, as shown in listing 1. Some contestants solved by clicking on the file in a DE that automatically displayed the contents inside. We just did not expect anyone to manually open 512 files, but it worked.

The answer is `aaronswartz`.

Listing 1: code/level4.py

```
#!/usr/bin/env python
from os import chdir
from subprocess import call, check_output
from sys import argv

LEVELS = 512

def decompress(fin):
    output = str(check_output(['file', fin])).lower()
    if 'rar' in output:
        call(['rar', 'x', fin])
        call(['rm', fin])
    elif 'xz' in output:
        call(['tar', 'xvJf', fin])
        call(['rm', fin])
    elif 'tar' in output:
        call(['tar', 'xvf', fin])
        call(['rm', fin])
    elif 'gzip' in output:
        call(['tar', 'zxvf', fin])
        call(['rm', fin])
    elif 'zip' in output:
        call(['unzip', fin])
        call(['rm', fin])
    else:
        print("Finished!")


# Starting with file 512.*, extracts everything.
```

```
if __name__ == '__main__':
    chdir('files/')
    for i in reversed(range(0, LEVELS+1)):
        decompress(str(i))
```

Level 5: Weak die fast

We are given a password protected zip file. The password is inside. Here the title suggests that the password is weak, and it is fast to crack. Using an already existing software to crack zips or creating a small script that would try all combinations would find that the password of the zip is `sat`. A sample script for the task is given in listing 2.

The answer is `knuthmorrispratt`.

Listing 2: code/level5.py

```
#!/usr/bin/env python
from itertools import product
from string import ascii_lowercase
from subprocess import call

MAX_LENGTH = 6

if __name__ == '__main__':
    with open('/dev/null', 'w') as devnull:
        for l in range(1, MAX_LENGTH):
            print('Trying words with length {0}...'.format(l))
            for perm in product(ascii_lowercase, repeat=l):
                pswd = ''.join(list(perm))
                if call(['unzip', '-t', '-P', pswd, '-qq', '-o', 'input'],
                        stderr=devnull, stdout=devnull) == 0:
                    call(['unzip', '-o', '-P', pswd, 'input'])
                    print('Extracted Zip with password: {0}'.format(pswd))
                    exit()
        print('No password found up to length {0} to extract zip'.format(
            MAX_LENGTH))
```

Level 6: Colors!

We are given a HackUPC logo and are asked to find the password. Looking carefully at the picture, we could see that some pixels differ from others by 1 value of alpha. If we binarize this image based on that, we obtain a Qr. All this could be done via code, or just with GIMP or any other image editing software for example. Scanning this Qr yields to... another Qr! Scanning this last Qr yields to the password. In listing 3 there is a simple script to generate the Qr from the image.

The answer is `kolmogorov`.

Listing 3: code/level6.py

```
#!/usr/bin/env python
from PIL import Image
```

```
IMAGE_SIZE = 255
INPUT_FILENAME = 'input.png'
RESULT_FILENAME = 'result.png'

# Generates the QR Code from alpha values in input.png.
if __name__ == '__main__':
    input_image = Image.open(INPUT_FILENAME)
    result_image = Image.new('1', (IMAGE_SIZE, IMAGE_SIZE), 'white')
    pixels = result_image.load()
    for i in range(IMAGE_SIZE):
        for j in range(IMAGE_SIZE):
            (_,_,_,a) = input_image.getpixel((i,j))
            if a == 255:
                pixels[i,j] = 0;
    result_image.save(RESULT_FILENAME)
```

Level 7: Game of Life

You are given an initial Game of Life pattern as a matrix of 1s (alive cells) and 0s (dead cells). Calculate the iteration at which the cycle starts, and the length of it. Your solution should of the form (steps_to_cycle)-(cycle_length), without parenthesis. Keep in mind, that this Game of Life uses a looping matrix!

This can be solved by implementing a game of life, and at each step storing the board in a key-value structure, where the board is the key, and the value is the current iteration number. At some point, we will get to a state that it was already visited. As we know when was first seen, and the current iteration, we can get when it started, and the cycle length with a simple subtraction. There is a C++ sample code in listing 4.

The answer is `286-168`.

Listing 4: code/level7.cc

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;
typedef unsigned long long ll;
typedef vector<vector<bool>> Board;

const int height = 42;
const int width = 42;

const int di[] = {-1, -1, -1, 0, 0, 1, 1, 1};
const int dj[] = {-1, 0, 1, -1, 1, -1, 0, 1};

void step(Board &board, Board &tmp) {
  for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
      int count = 0;
      for (int k = 0; k < 8; ++k) {
        int ni = (i + di[k] + height)%height;
        int nj = (j + dj[k] + width)%width;
        count += board[ni][nj];
      }
      tmp[i][j] = (count == 3 || (count == 2 && board[i][j]));
    }
```

```
  }
  swap(board, tmp);
}

int main() {
  Board board(height, vector<bool>(width));
  Board tmp(board);
  for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
      char c; cin >> c;
      board[i][j] = (c == '1');
    }
  }

  map<Board, int> seen {{board, 0}};
  int cycle_turn = -1;
  int cycle_length = -1;
  for (int turn = 1; cycle_turn == -1; ++turn) {
    step(board, tmp);
    auto it = seen.find(board);
    if (it == seen.end()) {
      seen[board] = turn;
    } else {
      cycle_turn = it->second;
      cycle_length = turn - cycle_turn;
    }
  }
  cout << "Cycle␣starts␣at:␣" << cycle_turn << endl;
  cout << "Cycle␣length␣is:␣" << cycle_length << endl;
  cout << "The␣answer␣is␣" << cycle_turn << "-" << cycle_length << endl;
}
```

Level 8: There's more than what you see

We are given a link to `http://46.101.102.224`, but if we go there, we only see the text `Hello, this file has nothing for you...`. If we use the hint given in the title, and we look inside the headers, we can see that the response header includes an attribute for the password, as shown in figure 1.

The answer is `mllobetisawesome`.

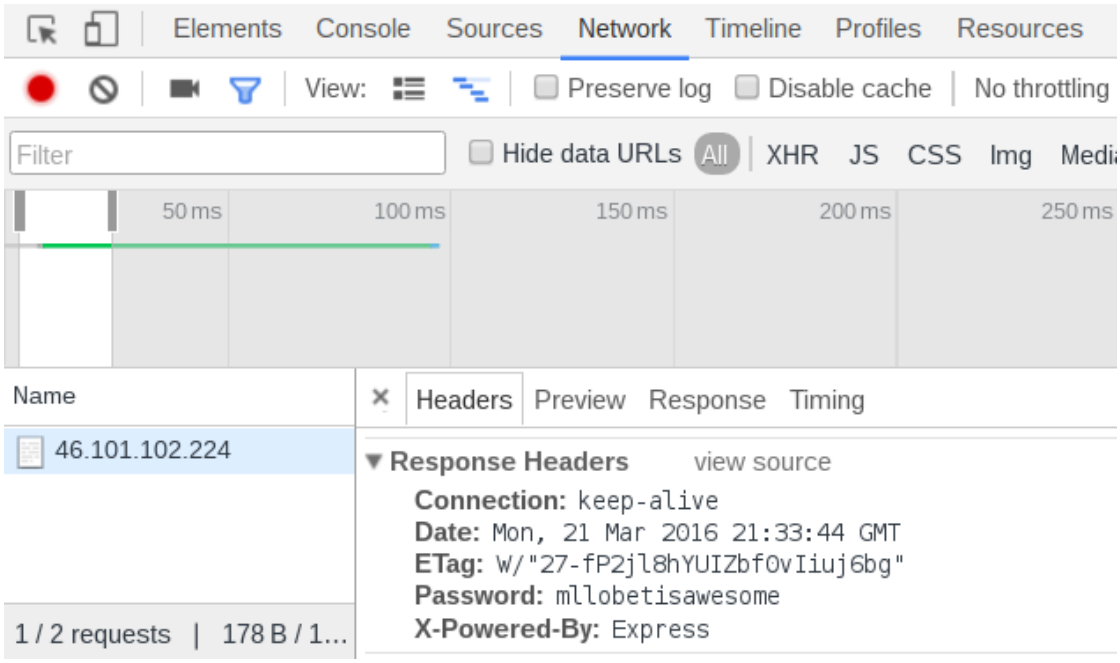

Figure 1: Response header.

Level 9: Add them all!

This problem contained lots of images with binary numbers. The solution was to add all numbers in binary and give it in base 10. To solve this problem one could use any OCR software, such as *tesseract*. There is a sample solution in python in listing 5.

The answer is 2601850323.

Listing 5: code/level9.py

```
from subprocess import check_output
from os import listdir

if __name__ == '__main__':
    answer = 0
    for f in listdir('files'):
        output = check_output(['tesseract', '-psm', '8',
            'files/'+f, 'stdout', 'nobatch', 'digits'])
        answer += int(output, 2)
    print('Total sum is {0}'.format(answer))
```

Level 10: Counting bits

It's easier to count the number of bits from 0 to n. Then we can count from 0 to 1333333333333337, and subtract the amount from 0 to 1337-1 using inclusion exclusion

principle. To count from 0 to n, we can use some observations. When n is $2^i - 1$, we have been through 2^i numbers from 0, and each bit to the right of the i-th bit, has been to '1' half of the times. Therefore, each bit has been to '1' exactly 2^{i-1} times.

We can iterate through all bits from the left, and keep adding using that observation, whenever there is a bit to '1', we add that one bit, we add the bits to the right for that bit to be '1', and then we continue as if this bit no longer exists. There is a sample code C++ in listing 6.

The answer is `33254528318498418`.

Listing 6: code/level10.cc

```
#include <iostream>
typedef unsigned long long ll;

const ll from = 1337;
const ll to = 1333333333333337;

ll f(ll x) {
  ll ans = 0, act = 0;
  for (int bit = 63; bit >= 0; --bit) {
    ll tmp = act | (1ULL<<bit);
    if (tmp <= x) {
      ans += ll(bit) * (1ULL<<(bit-1)) + (tmp^x) + 1;
      act = tmp;
    }
  }
  return ans;
}

int main() {
  std::cout << f(to) - f(from-1) << std::endl;
}
```

Level 11: Getting inside

In this problem, we were asked to get inside Linus' server. We were given his private key, username, and the ip of his server. However, if we tried to connect into it via ssh we find a problem:

```
$ ssh -i linus_rsa linus@46.101.102.228
Permission denied (publickey).
```

Something is wrong here, but all the given data is correct. One could try to use ssh over different ports. Here, due to the number of previous references to 1337, it would make sense to try port 1337. Another way to realize this is to use nmap to search of opened ports. The problem is that nmap does not scan port 1337 by default, but if we give a fairly large range of ports to scan, it will find it.

```
$ nmap 46.101.102.228 -p1-2000
Starting Nmap 7.10 ( https://nmap.org ) at 2016-03-22 16:41 CET
```

```
Nmap scan report for 46.101.102.228
Host is up (0.078s latency).
Not shown: 1996 closed ports
PORT     STATE    SERVICE
22/tcp   open     ssh
1337/tcp open     waste
```

Now, trying to ssh using port 1337 does not show us the previous error. In fact, we find that the terminal is cleared and then the connection is closed. If we ssh redirecting the output to a file, this time the content is not deleted, and we get the password. What happened, is that the server by default printed a text file that contained the password with the command `more`, and then it closed the connection. As the text in the file is fairly small, the command `more` terminated immediately. If we resize the terminal window to make it smaller, and try again, we see that this time the command `more` does not finish, as there is not enough space to show all text, and we can see the password as well.

The answer is `thecakeisalie`.

Level 12: XOR Part 2

It is the same problem as Level 3, but this time numbers are to large to XOR all numbers. As there is a previous Level that involves counting the number of bits, we know from that problem that for any $n > 1$, the number of times that each bit has been to '1' from 0 to 2^n (exclusive) is even, therefore the XOR is 0. If we try to find a power of two close to the number in the input, we see that $2^{60} = 1152921504606846976$. Since the accumulated XOR to that point is 0, and the amount of numbers from that to the limit from the input is very small, we can do the XOR of the remaining numbers one by one.

A simpler way to solve this problem, is to realize that the accumulated XOR follows a pattern. Every 4 numbers it becomes 0. This makes it even easier to solve, following a similar approach explained above. A sample solution in C++ is shown in listing 7.

The answer is `133713371337133742`.

Listing 7: code/level12.cc

```cpp
#include <iostream>
using namespace std;
typedef unsigned long long ll;
const ll from = 0;
const ll to = 1152921504606847076ULL;

int main() {
  ll cur = 128663487594398074 6ULL;
  ll pot = 1;
  while ((pot<<1) < to) {
    pot <<= 1;
  }
  for (ll i = pot; i <= to; ++i) {
    cur ^= i;
```

```
  }
  cout << cur << endl;
}
```

Level 13: Name the sequence

This level only contained an mp3 file. The audio is from the HackUPC 2016 Trailer video that was posted in the official website. However, if we get the audio from that video and compare to the one provided, the files differ.

Looking at the hexdump of the audio file (or opening with a text editor), we see at the very beginning, a part that contains the string 'HackUPC', with the code in listing 8.

Listing 8: Haskell code.

```
scanl(\c n->c*2*(2*n-1)`div`(n+1))1[1..]
```

This is Haskell code that generates an infinite list of numbers. More precisely, it generates the list of Catalan numbers. We can find that by running the code locally or any online haskell interpreter, and recognizing the sequence or searching for it at any search engine. As the title suggests, the answer is the name of the sequence of numbers.

The answer is `catalan`.

Level 14: Lost

The code given shows us that the thee is numbered in inorder. The idea to solve this problem, is to find the lowest common ancestor of Alice's room and Bob's room, that is, the deepest parent that both have in common. An observation for this problem, is that if we have a perfect balanced BST in inorder of height h, and another of height $t > h$, the subtree with height h appears in the one with height t with the exact same ids. Therefore, we can take an arbitrarily large perfect balanced BST that includes both ids, and build the path from root to each desired node. Then easily get the LCA and each path. A sample code is proved in listing 9.

The answer is `uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuurllrlrrrrllrlrlrlllrrlrrlrrrlrrllrrlrlrlrlrlrrll`.

Listing 9: code/level14.cc

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;

// The valid range is [l, r).
string get_path(ll l, ll r, ll x) {
  ll h = (l+r)>>1ll; // Root of the subtree.
  if(x==h) return ""; // We are in the root.
  string ret;
  if(x<h) {
    ret = 'L' + get_path(l, h, x);
```

```
  }
  else {
    ret = 'R' + get_path(h+1ll, r, x);
  }
  return ret;
}

int main() {
  int n = 60; // Large enough tree.
  ll u,v; cin >> u >> v;
  string root_to_u_path = get_path(0, 1ll<<(ll(n)), u);
  string root_to_v_path = get_path(0, 1ll<<(ll(n)), v);
  int lca = 0;
  // Remove all common prefix, that is, path to LCA.
  while (lca < root_to_u_path.size() and lca < root_to_v_path.size() and
         root_to_u_path[lca] == root_to_v_path[lca])
    ++lca;
  // Going up until LCA.
  for(int i=lca; i<root_to_u_path.size(); ++i)
    cout << 'U';
  // Go to 'v' using the path from LCA to 'v'.
  for(int i=lca; i<root_to_v_path.size(); ++i)
    cout << root_to_v_path[i];
  cout << endl;
}
```

Level 15: Tabs vs Spaces

This level consisted of only an input with spaces and tabs. It was indeed a code written in glorious whitespace. If we try to execute the program, we see that it does not finish. This is because there is a loop at the beginning with too many iterations. One can learn whitespace to see what's happening and solve the problem (like we did, and we enjoyed learning it), or use any of the many online interpreters that at the same time, translate the code into javascript. With this we can see that the first number being pushed (-1073741824) causes too many loop iterations, as each iteration increments by one that number until it's zero. We can solve this by removing the entire loop, or changing the number, as shown in Listing 10.

Listing 10: Change of the first line.

```
S S T   T       E
```

With this change, the program finishes instantly on any interpreter, and it outputs a message with the answer.

The answer is `noneedtobeupset`.

3 Stats

So far 12 people solved `The Game` by completing all 15 levels! Here we show a few stats about the challenge. In figure 2 there is a chart with the number of people that solved each problem to this date by level.

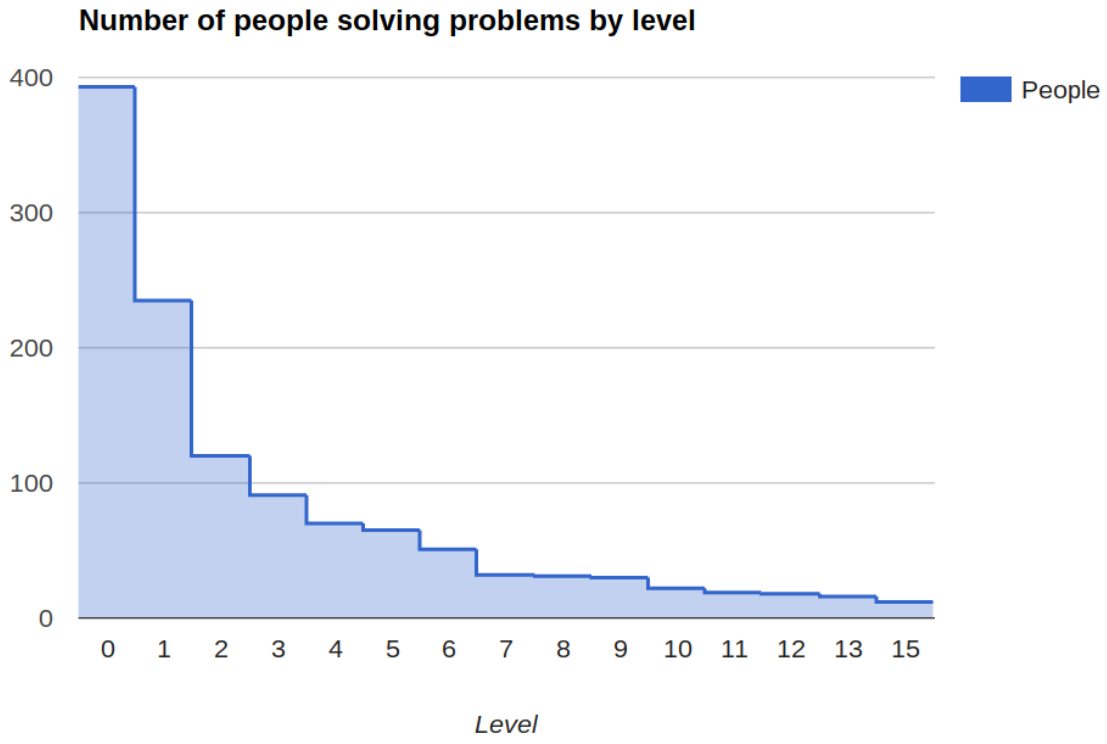

Figure 2: Stats by level.

Also, in table 1 is shown the top 15 people from the ranking, to this date. People with same number of problems are sorted by date of last problem solved.

User	Current level
gog	Finished
cescmentation_folch	Finished
joseballester	Finished
すばらしい	Finished
dirbaio	Finished
hermito	Finished
davidalro	Finished
gonzalo	Finished
arbequi	Finished
paualos3	Finished
Elfresh	Finished
serk12	Finished
hydn	14
EducatedLobsters	14
albertvaka	14

Table 1: Ranking with top 15.

4 Feedback and more

If you want to give some feedback about `The Game`, you can write us at the Facebook page Hackers at UPC, share your opinion in the Facebook group Hackers@UPC or comment in our Slack Channel.

We hope you enjoyed participating in `The Game` as much as we did preparing it. See you at next HackUPC!