INTERLOS III: ŘEŠENÍ ÚLOH

SADA 1.

INTERLOS

P1 Na volný čas

Partie dopadnou postupně takto:

```
NOOOOOXOOOOOONXNOOOXOOOXOOOONXOOOXNOOONOOOOOOXOONO
OOOOOXOOOOOOONNONONOOOXONOOONXOOXOONNONNOOONXONONO
```

Ti méně hraví z vás si mohli napsat program podobný tomuto, využívající algoritmus minimax, určený právě pro takovéto problémy:

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

const int SIZE = 3;

struct field {
	char f[SIZE][SIZE+1]; // zarážka pro pohodlný výpis řetězce
	field() { for (int i = 0; i < SIZE; i++) f[i][SIZE] = '\0'; }
};

struct coords {
	int y, x;
	coords() : y(-1), x(-1) {}
	coords(int y, int x) : y(y), x(x) {}
};

map<string,int> minimax_cache; // cache minimax hodnot

string encodeField(field f);
int minimax_value(field f);
bool third_in_row(field f, int y, int x);

int main()
{
	int N;
	cin >> N;

	for (int i = 0; i < N; i++) {
		// načíst zadání
		field input;
		int freeFieldsCnt = 0;
		for (int y = 0; y < SIZE; y++) {
```

```
                        for (int x = 0; x < SIZE; x++) {
                                cin >> input.f[y][x];
                                if (input.f[y][x] == '.') {
                                        freeFieldsCnt++;
                                }
                        }
                }
                char onMove = (freeFieldsCnt % 2 == 1 ? 'x' : 'o');

                // spočítat ohodnocení hry algoritmem minimax
                int mmval = minimax_value(input);
                // 1 znamená, že hráč, který je na tahu, vyhraje
                // -1 naopak
                // 0 znamená, že hráč na tahu může při nejlepším uhrát remízu
                switch (mmval) {
                case 1:
                        cout << (onMove == 'x' ? 'X' : 'O');
                        break;
                case -1:
                        cout << (onMove == 'x' ? 'O' : 'X');
                        break;
                case 0:
                        cout << 'N';
                        break;
                default:
                        cerr << "vstup " << i << ": minimax_value se uplne netrefila..." <<
endl;
                }
        }
        cout << endl;
        return 0;
}


/**
 * Spočítá minimax ohodnocení aktuální hry.
 * @return -1, 0 nebo 1 podle toho, jestli hráč, který je na tahu, prohraje,
 *         remizuje resp. vyhraje
 */
int minimax_value(field node)
{
        string encoded = encodeField(node);
        map<string,int>::iterator cached = minimax_cache.find(encoded);
        if (cached != minimax_cache.end()) {
                return cached->second;
        }

        coords freeFields[SIZE*SIZE];
        int freeFieldsCnt = 0;
        for (int y = 0; y < SIZE; y++) {
                for (int x = 0; x < SIZE; x++) {
                        if (node.f[y][x] == '.') {
                                freeFields[ freeFieldsCnt++ ] = coords(y, x);
                        }
                }
        }
        if (freeFieldsCnt == 0) {
                return 0; // nikdo doteď nevyhrál a už nejsou volná pole - remíza
        }
        char onMove = (freeFieldsCnt % 2 == 1 ? 'x' : 'o'); // začíná 'x' a pak se střídají
        int bestVal = -2;
        for (int i = 0; i < freeFieldsCnt; i++) {
                // zkusit táhnout postupně na všechna pole, vybrat nejlepší hodnotu
                field child = node;
                coords c = freeFields[i];
                child.f[c.y][c.x] = onMove;
                // max(a,b) = -min(-a,-b), ošetříme tak zároveň oba hráče
                int childVal = (third_in_row(child, c.y, c.x) ? -1 : -minimax_value(child));
                bestVal = max(bestVal, childVal);
        }

        minimax_cache[encoded] = bestVal;

        return bestVal;
}
```

```
/**
 * Zjisti, zda posledni tah, ktery byl na souradnice [x,y], vedl k vytvoreni
 * trech stejnych znacek v rade.
 * @param hrací pole
 * @param y
 * @param x
 * @return TRUE pokud tah na [x,y] vedl k vytvoreni trojice stejnych znacek,
 *         jinak FALSE
 */
bool third_in_row(field f, int y, int x)
{
	// three in a row check
	bool row = (f.f[y][x] == f.f[y][(x+1)%3] && f.f[y][x] == f.f[y][(x+2)%3]);
	// three in a column check
	bool col = (f.f[y][x] == f.f[(y+1)%3][x] && f.f[y][x] == f.f[(y+2)%3][x]);

	bool result = row || col;

	// diagonal from top-left to bottom-right
	if (x - y == 0) {
		result = result || (f.f[0][0] == f.f[1][1] && f.f[1][1] == f.f[2][2]);
	}
	// diagonal from top-right to bottom-left
	if (x + y == 2) {
		result = result || (f.f[0][2] == f.f[1][1] && f.f[1][1] == f.f[2][0]);
	}

	return result;
}


/**
 * Zakóduje hrací pole do řetězce.
 * @param f hrací pole jako pole SIZE*(SIZE+1), kde vždy poslední prvek každého řádku je '\0'.
 * @return řetězec reprezentující zadané pole
 */
string encodeField(field f)
{
	return string(f.f[0]) + string(f.f[1]) + string(f.f[2]);
}
```

P2 Jednoduchá

Vedle Tato úloha by skutečně byla zcela triviální – ovšem kdybychom měli dostatečně silný počítač. Zřejmě jste zjistili, že i pro malé vstupy program počítá skutečně dlouho.

Prohlédneme nejdřív, jak se funkce vzájemně volají: f() volá g() a h(), g() nevolá nikoho, h() volá m() a m() opět nevolá nikoho.

Rozpleťme tedy program od m(). Vypíšeme-li, co vrací m() pro různé argumenty, zjistíme, že výsledkem je vždy jediné číslo: 6543. Protože podceňovat organizátory se nevyplácí, je dobré si hypotézu ověřit. Co se děje ve vnitřním cyklu? Hodnota proměnné k se XORuje (bitový exkluzivní součet, budeme značit $\oplus$) s hodnotami 0 až i-1. To celé se provádí (3456 & i)-krát (kde & je operace bitového součtu). To je sice kdoví kolik, ale určitě je to sudé číslo. Přitom platí: $k \oplus x \oplus x = k$, $x \oplus x = 0$, $k \oplus 0 = k$, $x \oplus y = y \oplus x$.

Proto: $k \oplus 0 \oplus 1 \oplus ... \oplus (i\text{-}1) \oplus 0 \oplus 1 \oplus ... \oplus (i\text{-}1) = k$

(a obecněji, pro sudý počet provedení takové série operací). Po provedení celého vnějšího cyklu se tedy s hodnotou k vůbec nic nestane. Ve výsledku funkce m() vrací $i \oplus 6543 \oplus i = 6543 \oplus i \oplus i = 6543 \oplus 0 = 6543$.

Když víme, co vrací funkce m(), podívejme se, kde je její výsledek použit - ve funkci h(). Z pole 10000 prvků vybíráme prvek s indexem m(i), tedy 6543. Podívejme se, co se v poli p počítá. Vypíšeme-li jej, je to očividné. Prohlédneme-li obě smyčky, můžeme rozpoznat Eratosthenovo síto.

V poli p máme pod indexem i i-té prvočíslo.

6543. prvočíslo je 65537.

Po zjednodušení celé funkce h() na prosté vrácení konstantní hodnoty se program podstatně zrychlí, ale stále ne dostatečně.

Po prohlédnutí funkce g() snadno zjistíme, že počítá b-tou mocninu čísla a modulo c. Pokud si chceme trochu zaprogramovat, můžeme tuto lineární implementaci umocňování nahradit logaritmickou, což už např. pro implementace v jazyce C bude dostatečně rychlé. Pozor si musíme dát jen na přetečení 32bitového integeru, protože $65536 \cdot 65536$ už je příliš mnoho.

Alternativní cestou je přímo optimalizovat funkci f(). Ta totiž funkci g() opakovaně volá s týmiž hodnotami a, c a s hodnotami b postupně od 0 do

C-1. Budeme-li si výsledky g() pamatovat, budeme v každém kroku pouze jednou násobit namísto počítání celé g(). Taková optimalizace už bohatě stačí i na implementaci v PHP a za chvíli máme výsledek.

Ještě jinou cestou je podívat se, co vlastně f() počítá: $s = A \cdot B^0 + A \cdot B^1 + A \cdot B^2 + ... + A \cdot B^{C-1}$, každý člen modulo 65537. Ano, je to součet prvních C členů geometrické posloupnosti s prvním prvkem A a kvocientem B. To již můžeme podle vzorce spočítat na kalkulačce. Zbytek po dělení 65537 můžeme aplikovat až na výsledek, takže i touto cestou dostáváme odpověď na tuto úlohu: 22731.

Pro úplnost ještě přikládáme implementace optimalizovaných programů:
C: http://www.fi.muni.cz/~xbouda2/interlos/opt-optimized.c
Java: http://www.fi.muni.cz/~xbouda2/interlos/opt_optimized.java
Pascal: http://www.fi.muni.cz/~xbouda2/interlos/opt-optimized.pas
Python: http://www.fi.muni.cz/~xbouda2/interlos/opt-optimized.py
PHP: http://www.fi.muni.cz/~xbouda2/interlos/opt-optimized.phps

P3 Čísla

Spočítat počet písmen potřebných k zápisu všech čísel od 1 do 1000 lze mnoha způsoby. Ať už jste ale zvolili jakýkoliv, měli jste dospět k počtu 15595. U této úlohy jsme se inspirovali stránkou http://projecteuler.net.

S1 Kuličky

Jde o semaforovou abecedu. Modrá i červená barva jsou v každé kuličce stínované vždy z jednoho z osmi směrů – dohromady tedy každá kulička kóduje jedno písmeno v semaforové abecedě. Dostáváme text "heslo je plot".

S2 Trénovali jste?

Pokud jste řešili úlohy v Příručce hráče, mohli jste hned tušit heslo „SPANELSKO“:

```
Pyreneje, Flamenco, Monarchie, Evropskaunie, Reconquista,
724 ESP ES, Filip V
```

S3 Deník

Mestá Rím, New Orleans, Benátky, Houston, Wellington, Moskva, Pyongyang. Vieme, že v Ríme je 7 hodín ráno, priradíme časy ostatným mestám podľa časových zón:

Rím 7:00, New Orleans 1:00, Benátky 7:00, Houston 1:00, Wellington 18:00, Moskva 9:00, Pyongyang 14:00.

Substitúcia hodín za písmená abecedy: GAGARIN.

L1 Osmisměrka

Záhadné symboly v tabulce nejprve převedeme do srozumitelného písma (např. latinky), k čemuž využijeme:

1. překopírování obrázkových slabik do textového editoru a změnu jejich fontu (z Webdings např. do Arial)
2. znalost morseovky
3. znalost Braillova písma (+ dobrý zrak)
4. čtení římských číslic a znalost pořadí písmen v abecedě
5. znalost pořadí písmen v abecedě
6. znalost alfabety (levý dolní roh)
7. znalost azbuky (pravý horní roh)

Následně v tabulce vyškrtáme všechna pod ní zadaná slova, přičemž dodržujeme standardní pravidla osmisměrky.

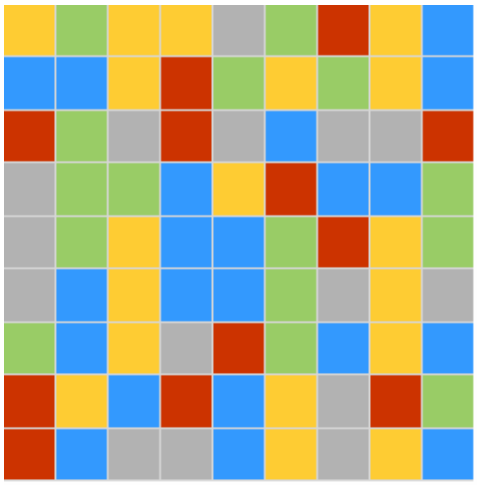
Zbylé slabiky čteme po řádcích, zleva doprava, shora dolů.
Výsledné heslo je: NANOTECHNOLOGIE.

L2 Einsteinův test

Kód je BABACBADCAAD

L3 Kostkožrout

Kostky zleva doprava měly být: šedá, zelená, oranžová, modrá, modrá, zelená, červená, oranžová, zelená:



SADA 2.

INTERLOS

P4 Kdo hledá, ten najde!

K řešení úlohy je potřeba použít nějaký disassembler, jako například objdump (je dostupný i pro Windows v MinGW distribuci). Pokud jste si četli příručku hráče, byli jste na to připraveni. Pak už jen stačí zavolat:

`objdump -DC heslo` (případně použít jiný binární soubor)

a ve výstupu najít funkci `heslo()`

Všimneme si, že obsahuje dva bloky zbytečných NOP instrukcí, řešení se skrývá mezi těmito bloky. Řešení je kódováno prvními znaky názvů instrukcí a říká: `heslo je assembler.`

Tedy řešení této úlohy je „ASSEMBLER“.

P5 Rekonstrukce kódu

Nejdříve bylo potřeba analyzovat, co se děje v průběhu:

- V prvním poli je kód, v druhém poli je pořadí písmen.
- Seřazení (bubble-sort) podle prvního pole. Pokud koukáme na první a druhé pole jako na jedno pole dvojic čísel, provádíme řazení podle první složky. Tuto operaci lze vrátit seřazením podle druhé složky.
- Hodnoty v prvním poli přepíšeme rozdíly hodnot na sousedních pozicích, přičemž první prvek necháme. Tuto operaci lze vrátit postupným přičítáním od prvního prvku.
- Pro každé *i* od 0 to 41 vyměníme pozici *i* s pozicí 0. Pokud by *i* procházelo od 0 do 8 (délka kódu), jednalo by se o cyklický posun pozic o 1. Pro *i* od 0 do 35 (4*délka kódu) tedy provedeme cyklický posun o 4. Zbývá 36 až 41, což odpovídá cyklickému

posunu prvních 6 pozic. Pokud jsme tedy začali s prvky ABCDEFGHI, po tomto kroku máme AEFGHIBCD.

- Opak kroku 3 - zpětně sečteme hodnoty.
- Opak kroku 2 - seřadíme podle druhého pole.
- Vypíšeme kód.

Původní kód bylo možné získat

- Ručně. Tato možnost byla poměrně jednoduchá a rychlá, zvláště pokud jste si napsali, jak krok 4 přehází prvky.
- Inverzním programem. Pokud bychom v kroku 4 provedli prohazovaní pro *i* od 0 až do 80 (délka kódu^2), dostali bychom zpátky to samé slovo. Inverzní program tedy získáme, pokud cyklus (zde ukázka z Pythonu)

```
for i in range(42):
     swap(i%DELKA, 0);
```

upravíme na:

```
for i in range(42,81):
     swap(i%DELKA, 0);
```

Původní kód byl „permutace“.

P6 Slovní fotbal

Grafová úloha, kde vrcholy grafu tvoří slova, hrany jsou dány návazností slov. Úkolem bylo ze zadaného vrcholu najít nejdelší sled (cestu bez opakujících se vrcholů).

Jestli si říkáte, že tento problém je NP-úplný (jinými slovy, fakt dost výpočetně náročný), máte pravdu. Dodaný slovník ale není příliš velký, takže téměř libovolný korektní algoritmus výsledek spočítá dostatečně rychle. Jedno z možných řešení si popíšeme.

Ze zadaného výchozího vrcholu reprezentujícího slovo adjudikace spustíme prohledávání grafu do hloubky, při kterém evidujeme hloubku zanoření. Největší hloubka, které dosáhneme, je zřejmě požadovaný výsledek – tedy až na ošetření cyklů. Při procházení nesmíme pokračovat vrcholem, kde jsme již byli, takže při navštívení vrcholu jej označíme a již označené sousedy dále neprocházíme. Rozdíl od běžného prohledávání do hloubky ale bude v tom, že označené vrcholy při návratu z průchodu zase odznačkujeme – mohou být totiž později součástí jiné (lepší) cesty. Díky tomu bohužel dostaneme exponenciální časovou složitost, ale líp to stejně nejde.

Co se implementace týče, protože výsledkem nemusí být konkrétní nalezený řetěz, jen jeho délka, není nutné si v grafu pamatovat celá slova. Rozhodující jsou pro slovo první a poslední tři písmena, takže nebudeme ukládat celé slovo, ale pouze čísla kódující začátek a konec

slova. Dodaný slovník obsahuje pouze slova z malých písmen anglické abecedy, trojice písmen tedy snadno zakódujeme do jediného čísla (např. je bereme jako zápis v 26kové soustavě), čímž budeme nadále namísto slov pracovat pouze s čísly, což usnadní (a urychlí) implementaci.

Zdrojový kód programu, který nad rámec zadání pro kontrolu vypisuje i průběžně nalezené nejdelší řetězce (a k tomu si ve vrcholech drží i originální slova), nalezente na adresách:

http://www.fi.muni.cz/~xbouda2/interlos/fotbal-solve.cpp

http://nlp.fi.muni.cz/~xrygl/interlos-2/fotbal-solve.cpp

S4 Hádanka století

Každá sloka básně odkazuje pomocí skrytých názvů děl na jednoho umělce (spisovatele či v případě sloky druhé na režiséra), u nějž, jak vyplývá z názvu šifry, potřebujeme zjistit století, ve kterém žil.

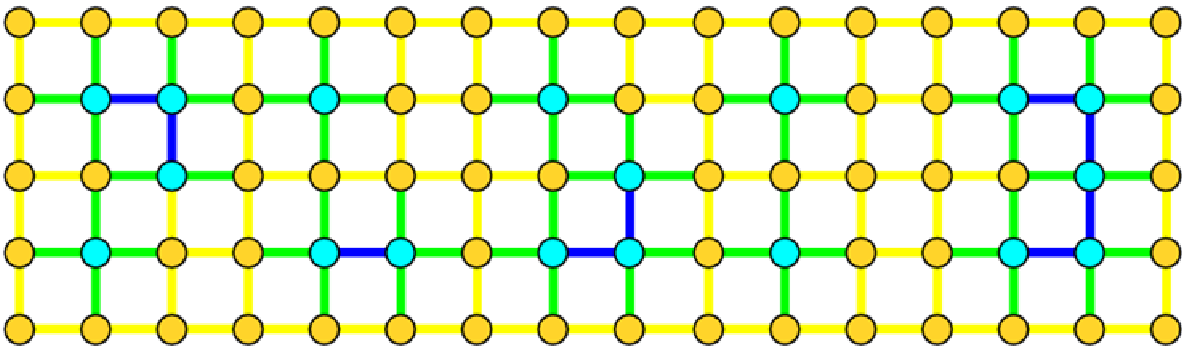
1. Naši furianti, Paní mincmistrová, Zkažení krev, Černé duše
   → Ladislav Stroupežnický (1850 – 1892 → 19. st.)
2. Limonádový Joe, Jáchyme hoď ho do stroje, Marečku podejte mi pero, Tři veteráni, Šest medvědů s Cibulkou, Adéla ještě nevečeřela
   → Oldřich Lipský (1924 – 1986 → 20. st.)
3. Emil aneb O výchově, Julie aneb Nová Heloisa
   → Jean-Jacques Rousseau (1712 – 1778 → 18. st.)
4. Odkaz, Velký testament
   → Francois Villon (1429 – 1463 → 15. st.)
5. Summa theologiae, důkazy boží existence
   → Tomáš Akvinský (1225 – 1274 → 13. st.)

Získaná století pak převedeme na písmena a získáme heslo: STROM.

S5 Přímo u zdroje

Úkolem šifry bylo především najít zadání. Název šifry, který jste mohli najít ve statistikách úkolů, navádí na zdrojový kód HTML stránky se zadáním. Zde se v komentářích nachází jednoduchá hádanka „když šel tam do hlavy ho tloukli, když šel ven, za krk ho tahali“. Kódem je slovo HREBIK.

S6 Uzly po šesti



V prvním kroku je potřeba obarvit vrcholy podle hran, které do nich vedou. Pokud je hrana žlutá, oba její okraje jsou žluté, pokud modrá, oba jsou modré. Pokud je hrana zelená, jeden z okrajů je modrý a druhý žlutý (míchání barev).
Pak stačí prozkoumat vzniklé tečky a všimnou si pěti znaků Brailla:

⠝⠥⠵⠅⠽

Heslo jsou NUZKY.

L4 Řady

V první číselné řadě se na střídačku zvyšovalo číslo o pět a o dva. Chybějící číslo proto bylo 17.

3
+5
8
+2
10
+5
15
+2
A = 17
+5
22

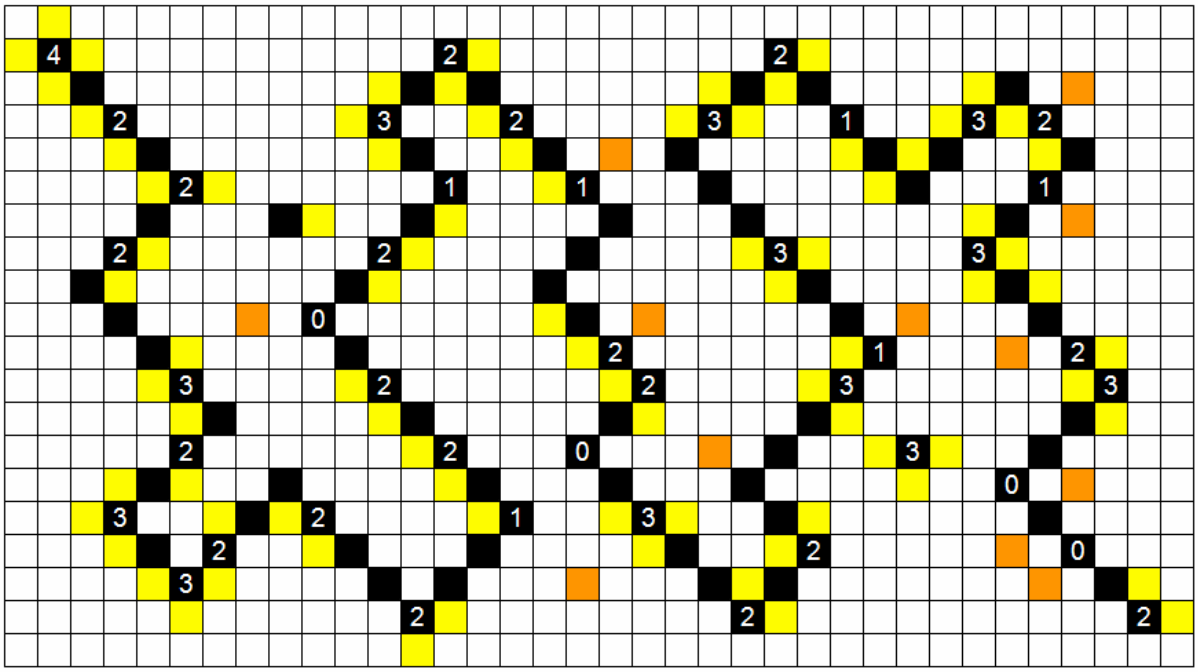
V druhé řadě bylo *i.* číslo *(i-1)* násobkem předchozího čísla. Tedy jedenkrát 25 je 25, dvakrát 25 je 50, třikrát 50 je 150 atd.

B = 25
1 · 25 = 25
2 · 25 = 50
3 · 50 = 150
4 · 150 = 600
5 · 600 = 300

Výsledný součin je 425.

L5 Akari

Úloha byla pracnější a časově náročnější, ale pokud jste zvládli neudělat chybu, jistě jste se dostali k výsledku:



Kód byl tedy „12“.
Úloha je převzata z webu http://mellowmelon.wordpress.com/, kde najdete velké množství dalších logických úloh.

L6 Omalovánko-spojovačka po třech

Omalovánka napovídá, že budeme používat barvy. Spojovačka po třech pak vede k tomu, že spojíme vždy tři obrázky, které jsou vybarvené stejnou barvou.

Trojice jsou:
Červená Karkulka, řepa, kříž
Zlatá svatba, padák, řez
Černá ovce, skříňka, díra (na ponožce)
Bílé vánoce, vrána, paní
Zelený čtvrtek, vlna, mír (greenpeace)
Modrá krev, knížka a vlajka portugalská (modrý portugal).

Spojením trojic vyjde FIX.

SADA 3.

INTERLOS

P7	Dračí křivka	

Mohlo by se zdát, že ručním prováděním popsaného postupu také půjde přečíst výsledný text. To samozřejmě ano, ale jelikož je text velmi dlouhý, bylo by to pracné. Nejjednodušší řešení opravdu bylo napsat program, který se pohyboval ve čtverci daným způsobem a vypisoval písmenka.

```
VAZENY RESITELE! CTETE-LI TENTO TEXT, JSTE NA NEJLEPSI CESTE

K VYRESENI TETO ULOHY, JEN TAK DAL! DRACI KRIVKA SE RADI MEZI
SOBEPODOBNE FRAKTALNI KRIVKY. ZAJIMAVE NAPRIKLAD JE, ZE JEJI
FRAKTALNI DIMENZE JE DVA. NEMERIME TEDY JEJI DELKU, ALE PLOCHU. PRO
VICE INFORMACI MRKNETE NA WIKI. KRIVKA PROCHAZI POLOVINOU POLICEK
TETO TABULKY A PROTO JE TAKE TENTO TEXT TAK DLOUHY. VAS ALE ASI SPIS
ZAJIMA NEJAKE HESLO, TAK PREJDEME K VECI. KOD JE PRIJMENI PRVNIHO
VEDCE, KTERY KRIVKU POPSAL, COZ BYL JOHN HEIGHWAY. ZADEJTE TO DO
SYSTEMU A BODY JSOU VASE! PREJEME HODNE STESTI PRI RESENI DALSICH
ULOH.
```

Pokud se vám do programování nechtělo, mohli jste si někde z internetu stáhnout obrázek dračí křivky, vhodně natočit, zvětšit či zmenšit, překrýt přes čtverec a rovnou číst.

P8 Živá řada

Tato jednoduchá úloha prověřila vaše základní programátorské schopnosti. Nebylo potřeba nic víc, než vývoj řady dostatečně dlouho simulovat, jako například takto (Python):

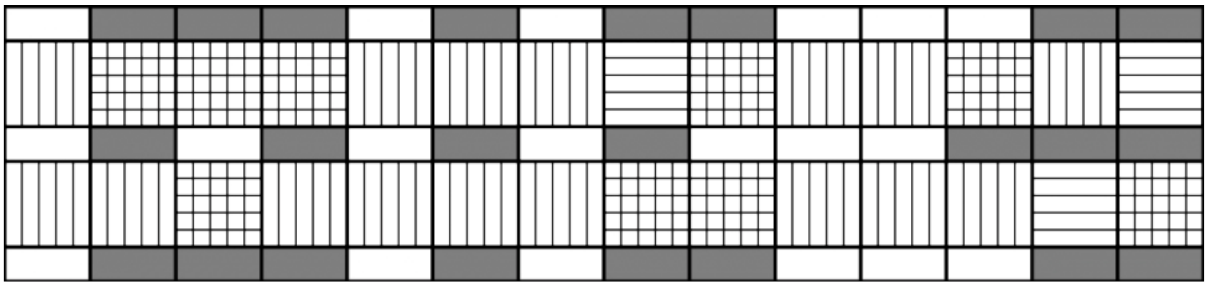
```
data = "111111100000"
i = 0
while len(data)>2:
    i += 1
    if data[0]=="1":  data = data[3:]+"1101"
    else:  data = data[3:]+"00"
print i
```

Správně napsaný program pak odpověděl 418.

P9 Obdélníky

V této úloze jsme se inspirovali Výzvou 17x17 (http://bit-player.org/2009/the-17x17-challenge). Nešlo zde o nic víc, než zkontrolovat možné pozice obdélníků. Váš program měl po nějakém tom chroupání vrátit `9422995`.

S7 Digitální číslice



Jak název napovídá, jde o digitální číslice, pouze se na ně díváme z boku. Dostáváme posloupnost čísel 19091316011426, pokud bereme čísla po dvojicích, dostaneme 19, 09, 12, 16, 01, 14, 26, což při převodu na písmena dává "simpanz".

S8 První sada

V této šifře jste museli využít podrobné výsledkové listiny. Každý řádek kóduje jedno písmeno následujícím způsobem:

7. kuličKy

Najdeme tým, který jako 7. v pořadí vyluštil úlohu „Kuličky“ a z jeho názvu vezmeme 6. písmeno (to písmeno, které je velké v názvu úlohy).

Kód je “METROPOLE”.

S9 UNESCO putování s Interlosem

Je to prosté. Z názvu fotografie vybereme tolikáté písmeno, kolik je na ní losíků. Písmena následně přečteme v pořadí, které nám udá seřazení obrázků podle datové velikosti souboru. Výsledné heslo bez diakritiky tedy je: POKLADY CECHU.

Pozn.:
Ano, všechny fotografie zobrazují české památky UNESCO, ovšem „UNESCO“ heslem opravdu není, protože je už v názvu šifry a navíc byste tím nezahrnuli informaci o rozdílném počtu losíků na jednotlivých obrázcích.

L7 Logik

Po chvíli konverzace bylo zřejmé, že je potřeba zadat právě čtyři slova. Nemusela to být slova konkrétní, experimentováním šlo zjistit, že je důležitý pouze slovní druh. Ačkoliv slova jako „jeden“ mohou mást, protože jsou nejen číslovkou, ale i slovesem. Pokud jste zadali postupně slovní druhy: *částice*, *podstatné jméno*, *zájmeno*, *sloveso*, např.: *nechť heslo tvé známe*, získali jste heslo.

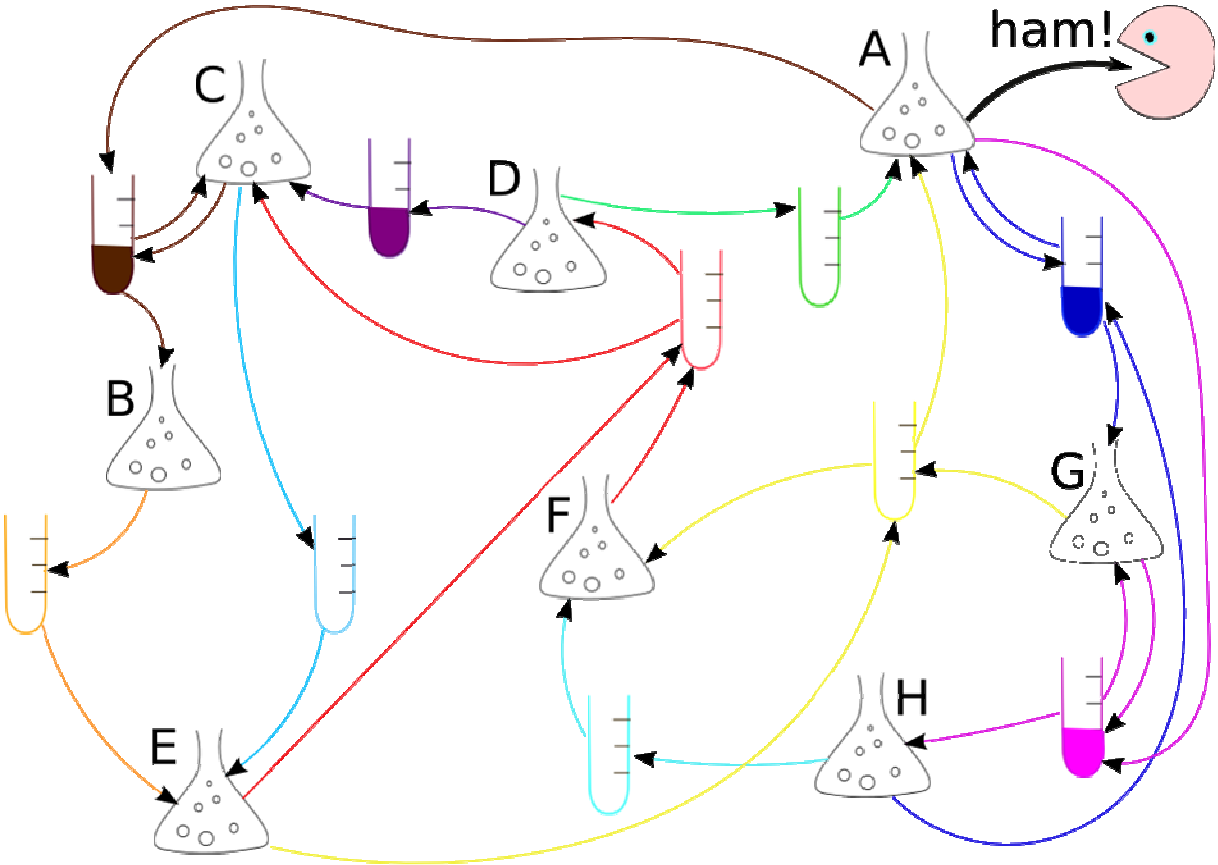
Výsledkem je kód „slovnidruhy“.

Rozpoznávání slovních druhů jsme dělali pomocí programu AJKA, který vyvinulo NLP Center Fakulty informatiky:
http://nlp.fi.muni.cz/projekty/wwwajka/WwwAjkaSkripty/morph.cgi?jazyk=0

Pokud se chcete podívat, jak se ptali ostatní, logy jsou k dispozici na
http://www.fi.muni.cz/~xrygl/logik_logs.py

L8 Továrna na bonbóny

Reakce bylo třeba udělat v pořadí GHFCBEDA.



L9 Kostka

Heslo je 456131211116

8		15	
10	**F**		
2		**A**	3
			16

8	14	15	12
10	11	7	6
2	1	4	3
13	9	5	16

	C	13	**B**
14	8		
7		11	9
12	3		1

4	6	13	5
14	8	2	15
7	16	11	9
12	3	10	1

16	10	9	
1	**E**		**D**
5		14	
11	4		7

16	10	9	2
1	12	3	13
5	15	14	8
11	4	6	7

	7	**G**	11
9	5	**H**	
6		12	
	2		

3	7	1	11
9	5	16	4
6	13	12	10
15	2	8	14