KIV/ZEP - 2011

Speciální předmět pro zvídavé a hravé
3 KREDITY

Pro koho

- Předmět je určen pro ty, které
  - baví řešit různé problémy (a nejen ty programátorské)
  - chtějí se dozvědět něco více o programování
    - např. jak zrychlit aplikaci 80x
  - chtějí pochopit to, co jim bylo v PPA2 skryto
- Předmět NENÍ určen pro ty, kteří
  - hledají levný způsob získání kreditů
  - měli výrazné problémy s předmětem KIV/PPA1

O čem

- Probíraná témata
  - Posuzování algoritmů aneb, co je dobrý a co špatný algoritmus + tipy a triky, jak to vylepšit
  - Práce s maticemi, geometrické operace
  - Správa paměti, efektivní využití cache
  - Redukce dimenze problému
  - Využití paralelního kódu (multicore CPU, GPU)
  - Rekurze, hrátky s grafy, stavový automat
  - Základy šifrování a komprese, jak to neznáte
  - a další ...

Co se chce

- Zápočet:
  - vypracovat 4 sady úloh po 20 bodech a získat za ně alespoň 50 bodů
    - Sady zahrnují několik jednoduchých neprogramátorských úloh, několik jednoduchých úloh typu napište program a 2 programátorské úlohy ze soutěže ACM contest
- Zkouška:
  - písemný test o 20 bodech
- Známka:
  - dle součtu bodů zápočet + zkouška
  - předmět je ohodnocen třemi kredity

Kdy

- Každé úterý
  - 2-3 hodina (8:25 – 10:00) UL411
  - první „vážná přednáška“: 22.2.2011

- Co dělá tento kód?
  - máte na to 2-3 minuty

```
public class UglyCode {
public static void print(int cap) {
	int[] p = new int[cap - 2];
	for (int i = 0; i < cap - 2; i++) {
		p[i] = i + 2;
	}

	int poc = 0;
		for (int i = 2; i*i <= cap; ){	//?? usare 3 in luogo di 2
			for (int j = (i - 1)*2; j < cap - 2; j += i) {
				p[j] = -1;
			}
					//molto effettivo
			while (++i < cap && p[i - 2] < 0) ;
}

	for (int i = 0; i < cap - 2; i++) {
			if (p[i] >= 0) {		System.out.printf("%d\t", p[i]);
	}
		}
}

	public static void main(String[] args) {
		print(100);
	}
}
```

Proč ?!

- Totéž, co tento!

```
/**
	Prints primes up to the given upper boundary
*/
public class NiceCode2 {

	/**
	Prints all primes from the interval from 2 to cap
	@param cap upper boundary
	*/
	public static void printPrimes(int cap) {
		//1) compute primes using the Sieve of Eratosthenes
		// first, generate all numbers from the interval 2..cap
		int n = cap - 2;  //number of values in the interval
		int[] prv = new int[n];
		for (int i = 0; i < n; i++) {
			prv[i] = i + 2;
		}

		int index = 0; //index to the first prime
		for (int tp = 2; tp*tp <= cap; tp = index + 2) { //it is enough to go to sqrt(cap)
			//set each value at (index + tp*k) to -1 since these are divisible by the prime tp
			for (int j = index + tp; j < n; j += tp) {
				prv[j] = -1;
			}

			//update index to the next prime
			while (++index < n && prv[index] < 0) ;
		}
		//2) we may print what is not marked by -1
		for (int i = 0; i < n; i++) {
			if (prv[i] >= 0) {
				System.out.printf("%d\t", prv[i]);
			}
		}
	}

	public static void main(String[] args) {
		printPrimes(100);
	}
}
```

Proč ?!

- Anebo tento!
- Který z nich je nejlepší?

Proč ?!

```
/**
        Prints primes up to the given upper boundary
*/
public class NiceCode {

        /**
        Prints all primes from the interval from 2 to cap
        @param cap upper boundary
        */
        public static void printPrimes(int cap) {
                //check every number starting from 2
                for(int i = 2; i < cap; i++) {
                        boolean composite = false; //false, if i is prime

                        //check if i is divisible by any number from 2 to sqrt(i)
                        for (int j = 2; j * j <= i; j++) {
                                if ((composite = (i % j) == 0))
                                        break;      //not a prime => we are ready
                        }

                        if (!composite) {
                                //if i is prime, print it
                                System.out.printf("%d\t", i);
                        }
                } //end for i
        }

        public static void main(String[] args) {
                printPrimes(100);
        }
}
```

- Myslíte si, že toto byl jen uměle vykonstruovaný příklad a v praxi to takhle ošklivě nikdo nenapíše?

Proč ?!

- Tak schválně ...

```
/**
Metoda sloužící k výpočtu dalšího členu pole
@param b
@return vysledek
**/
public static String vypocet(String b){
    int a = new Integer(b).intValue();
    int a_2 = a*a;
    String cislo_2 = "";
    String cislo =Integer.toString(a_2);
    for (int i=0; i<2; i++){
        cislo_2 += cislo.charAt(i);
    }
    int cislo_2int = new Integer(cislo_2).intValue();
    int skor_vysledek = cislo_2int + 1;
    String vysledek =Integer.toString(skor_vysledek);
    return vysledek;
}
```

Proč ?!

```
/**
Metoda sloužící k vypočtu globalního pole poz, pomocí pomocného pole
@param pole
@param p
@return 0
@return 2
**/
public static int poleVyp(int[] pole,String p){
    String prvni = p;
    for(int i=1;i<=pole.length;i++){
        if(i==1){
            pole[0]=new Integer(prvni).intValue();
        }else{
            p=vypocet(p);
            pole[i-1]=new Integer(p).intValue();
        }
        for(int j=0;j<pole.length;j++){
            if(i==1){
                break;
            }else{
                if((i-1)==j){
                    continue;
                }else{
                    if(pole[i-1]==pole[j]){
                        predelejPole(pole);
                        return 0;
                    }
                }
            }
        }
    }
    return 2;
```

- Tak schválně …

```
/**
 * @param args
 */
public static int vypoctiCislo (int x){ // tato metoda vypocita dalsi clem posloupnosti
        int k=x*x; // udela x nadruhou

        // nyni provedu test k podle toho kolik k vyjde s nim dale pracuji tak abych vratil spravny cislo
        if (k>999){
        int l=((x*x)/100)+1;
        return l;
        }
        if (k>99 && k<1000){
                int l=((x*x)/10)+1;
                return l;

        }
        if (k<100){
                int l=(x*x)+1;
                return l;
        }
return 0;

}
```

Proč ?!

```
public static int testPole (int x, int []pole){ //testuj

        int pocet=0;
        for (int i = 0; i < pole.length; i++) { // pokud nc
        if (pole[i] == x) {
        pocet++;
        }
        }
        if (pocet == 2){
        return 1; // nalezeny pocet prvku
        } else {
        return 0;
        } //nenalezen
        }
```

- Tak schválně …

Proč ?!

```
else{
try {
      BufferedReader bfr = new BufferedReader(new FileReader(vstup));
      PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(vystup)));

            String radka;
                  while ((radka = bfr.readLine()) != null){ // Pokud bude v radce bfre nejaka hodnota,
                        int c = Integer.parseInt(radka);   // retezec bfr se  prevede na int
                        int konecnePole[]  = new int[0];
                        int pole[] = new int[1];
                        pole[0] = c;
                        while(vicecetnost(pole) != true){
                              konecnePole = pole;
                              pole = vypocetCisla(pole);
                              }
                        int serazenePole[] = new int[konecnePole.length];
                        for (int i = 0; i < konecnePole.length; i++){
                                    serazenePole[i] = konecnePole[i] ;
                              }
                  selectSort(serazenePole);

                  pw.println(konecnePole.length + " " + Arrays.toString(konecnePole));
                  pw.println(serazenePole.length + " " + Arrays.toString(serazenePole));
                  pw.println();
                              }
      pw.close();


       }

      catch (Exception e) {
         e.printStackTrace();

       }
      finally {
      System.exit(1);
      }
```

- Tak schválně …

```
public static void main(String[] args) {

    int cislo = 0;

    if (args.length == 0) {
        Scanner sc = new Scanner(System.in);
        System.out.println("---Generovani pseudonahodnych cisel---");
        System.out.print("Zadej dvouciferne cislo(startovaci hodnota): ");
        cislo = sc.nextInt();
        System.out.println("" + generovaniPosloupnosti(cislo).length + " "
                + Arrays.toString(generovaniPosloupnosti(cislo)));
        System.out.println("" + generovaniPosloupnosti(cislo).length + " "
                + Arrays.toString(razeniPrvkuBubble(generovaniPosloupnosti(cislo))));
        sirka = generovaniPosloupnosti(cislo).length * SIRKA_SLOUPCE + 2;
        if (generovaniPosloupnosti(cislo).length < MIN_SLOUPCU) {
            sirka = STO+MIN_SLOUPCU-2;
        }
        vizualizace(generovaniPosloupnosti(cislo),
                razeniPrvkuBubble(generovaniPosloupnosti(cislo)));
    } else {
```

Proč ?!

- Tak schválně …

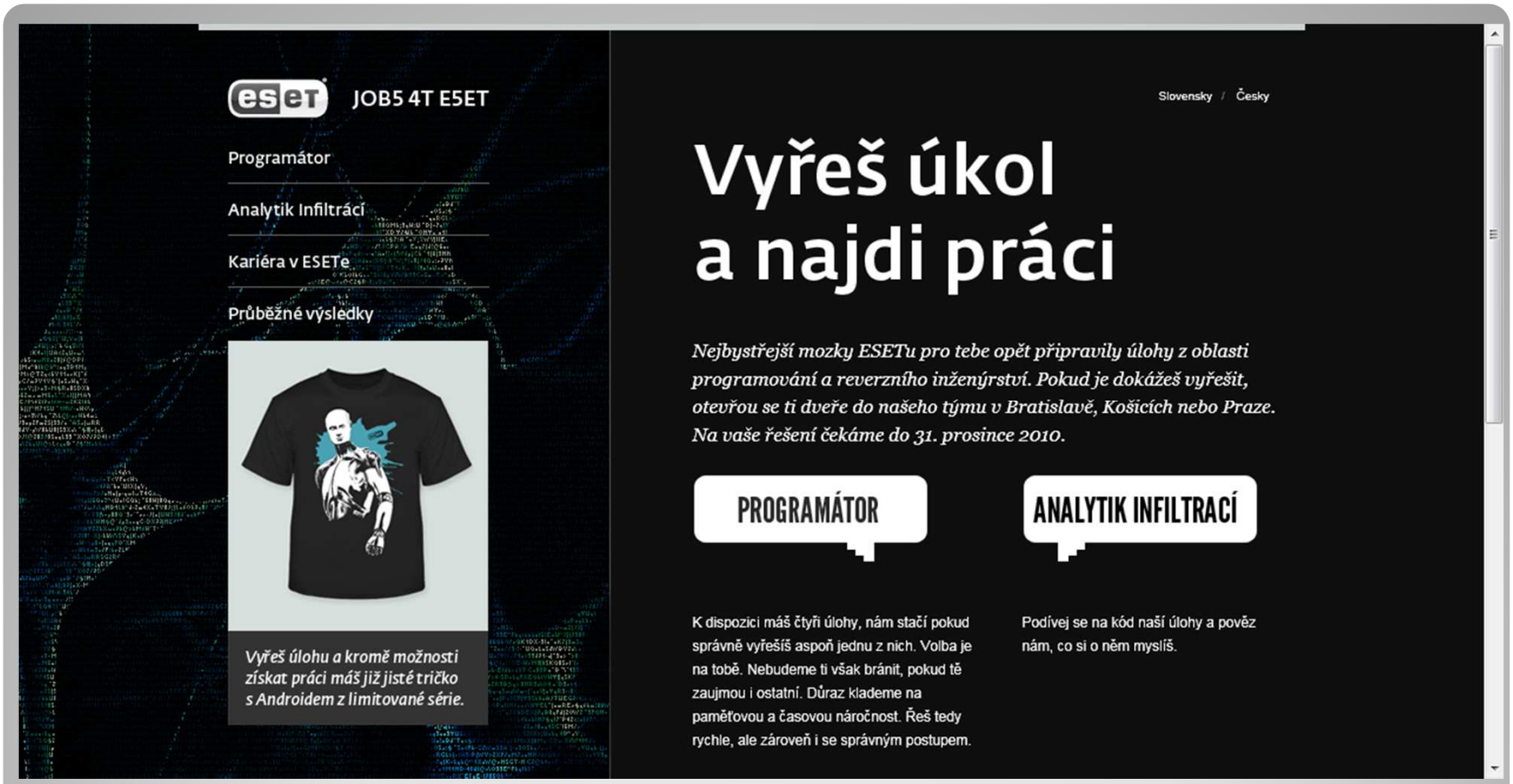
```
public static int [] nactiData()
  throws FileNotFoundException, IOException {
  Scanner sc = new Scanner(new File("d://A10B0465P//sp//vstup.txt"));
  int pole2[] = new int[10000000];
  int i = 0;
  while (sc.hasNextInt()) {
    pole2[i] = sc.nextInt();
    i++;
  }
  sc.close();//uvolnit zdroje
  Scanner a = new Scanner(new File("D:/
  int pole[] = new int[i];
  int g = 0;
  while (a.hasNextInt()) {
    pole[g] = a.nextInt();
    g++;
  }
  a.close();
  return pole;
}
```

Proč ?!

```
public static int pocetClenu (int Cislo){ //zjisteni poctu clenu pole pseudonahodnych cisel
    int dalsi3 = Cislo*Cislo;
    int dalsi4 = 0;
    double pomocna3 = 0;
    int vystup2[] = new int[90];
    vystup2[0]=Cislo;
    int pomocna4=1; //zjisti mi pocet, zacina jednickou, prvni clen pole je totiz dosazen natvrdo
    for (int x = 1; x<90; x++){
        if(dalsi3<1000){
            pomocna3 = (dalsi3/10)+1;
            dalsi4 = (int)pomocna3*(int)pomocna3;
        }
        if (dalsi3>999){
            pomocna3 = (dalsi3/100)+1;
            dalsi4 = (int)pomocna3*(int)pomocna3;
        }
        for (int i = 0; i<x; i++){
            if(pomocna3 == vystup2[i]){
                return pomocna4;
            }
        }
        pomocna4++;
        dalsi3 = dalsi4;
        vystup2[x] = (int)pomocna3;
    }
    return pomocna4;
}
```

- Nestačí tedy psát jen čitelněji kód?
- Počítače jsou dnes velmi rychlé, takže má smysl se bavit o psaní efektivního kódu?
- Kdo v praxi ocení vyšší efektivitu kódu?

Proč ?!


JOB5 4T E5ET
Programátor
Analytik Infiltrací
Kariéra v ESETe
Průběžné výsledky
Vyřeš úlohu a kromě možnosti získat práci máš již jisté tričko s Androidem z limitované série.
Slovensky / Česky
Vyřeš úkol a najdi práci
Nejbystřejší mozky ESETu pro tebe opět připravily úlohy z oblasti programování a reverzního inženýrství. Pokud je dokážeš vyřešit, otevřou se ti dveře do našeho týmu v Bratislavě, Košicích nebo Praze. Na vaše řešení čekáme do 31. prosince 2010.
PROGRAMÁTOR
ANALYTIK INFILTRACÍ
K dispozici máš čtyři úlohy, nám stačí pokud správně vyřešíš aspoň jednu z nich. Volba je na tobě. Nebudeme ti však bránit, pokud tě zaujmou i ostatní. Důraz klademe na paměťovou a časovou náročnost. Řeš tedy rychle, ale zároveň i se správným postupem.
Podívej se na kód naší úlohy a pověz nám, co si o něm myslíš.

Proč ?!

Vyřeš úlohu a kromě možnosti získat práci máš již jisté tričko s Androidem z limitované série.

Algoritmické úlohy

Ve všech úlohách této skupiny je cílem co nejefektivněji zpracovat větší množství dat a získat z nich nějakou informaci. Abychom jsme ji trochu zpestřili (a zároveň poskytli možnost hledat v některých případech specifičtější optimalizace), rozhodli jsme se použít ne úplně běžný typ dat - genetické informace. Genetická informace živých organismů je zakódována v tzv. kyselině deoxyribonukleové (DNK), kterou je možné si představit jako dlouhou sekvenci čtyř dusíkatých bází: Adeninu (A), Cytozínu (C), Guanínu (G) a Tyminu (T). Při analýze reálných genetických dat se pracuje se sekvencemi miliard komponent a jejich efektivní zpracování bývá výzvou nejen pro biology, ale i pro programátory - tj. vás!

Ve všech úlohách se analyzují sekvence DNK: v každé úloze si budete moci stáhnout jeden či více souborů obsahujících řetězce písmen A, C, G a T. Každý řádek reprezentuje jednu DNK sekvenci a v případě potřeby může být identifikován svým číslem. Řádky jsou ukončeny jen znakem LF (tj. drží se konvence zaužívané na platformách Unix/Linux). Co se týče úloh samotných, první dvě úlohy jsou podobného rázu. Původně jsme chtěli vybrat jen jednu z nich, ale při našich testovacích řešeních jsme zjistili, že obě dvě mají dost odlišné kompetitivní řešení a poskytují různé optimalizační možnosti, takže jsme je nakonec ponechali obě. Třetí je variací na známé téma s několika drobnými "úpravami".

Úloha 1: Buď anebo

Společně se souborem DNK dat dostanete k dispozici soubor pravidel, které *skoro úplně, ale zase ne tak celkově, nepřipomínají* regulérní výrazy. Každé pravidlo je na separátním řádku, jehož číslo je zároveň identifikačním číslem pravidla a skládá se z libovolně dlouhé sekvence následujících komponent:

- kterákoliv z bází (A, C, G, T)
- znaku '.'
- znaku '*'

Smysl znaků '.' a '*' je podobný jako v regulárních výrazů, tj. znak '.' reprezentuje jednu libovolnou bázi a '*' jakýkoliv řetězec bází (včetně prázdného). Cílem je identifikovat vzorky DNK, které je možné kompletně (tj. celé vzorky) reprezentovat některým z pravidel a zároveň zjistit kterým. Příklad pravidla P kompletně reprezentujícího vzorek V je na Obr. 01.

```
P:  AC..G* TT*         A  (AC..G*TT*A)
    |||||| |||         |
    |||||<>||<        >|
V:  ACGCG  TTAAGCTTCAGGA  (ACGCGTTAAGCTTCAGGA)
```

Obr. 01: Vzorek V a pravidlo P, které ho reprezentují (mezery jsou přidány jen kvůli srozumitelnosti, v souborech byste je našli tak, jak jsou zobrazené v závorkách)

Vaši úlohou je najít v souborech vzorků ty (jako na Obr. 02), které jsou reprezentovány některým pravidlem ze souboru pravidel (Obr. 03) a identifikovat o které pravidlo jde. V případě, že vzorek reprezentuje vícero pravidel, najděte to s nejnižším pořadovým číslem. Výsledný list utřiďte podle pořadových čísel vzorků jako na Obr. 04, tj. vzorek 0 `(AACGTAGC)` je reprezentovatelný pravidlem 1 `(AA*GC)` a vzorek 2 `(TTTT)` je reprezentován pravidlem 0 `(T..T)`. Vzorek 1 není reprezentovatelný žádným pravidlem a proto na něj reference ve výsledcích chybí.

- Výkon počítačů roste, ale požadavek zpracovávat stále větší a větší data s tím jde ruku v ruce …

Protože