**Midterm**

**Inštrukcie**

Oznam: pondelok, 30.11., 14:00-24:00

* odovzdávate elektronicky, tak ako iné zostavy,
* viditelne označte riešenia jednotlivých príkladov (filename/folder), nedávajte riešenia rôznych príkladov do jedného súboru/adresára,
* povolené sú akékoľvek pomôcky, knihy, printouts, prednášky, odovzdané úlohy, laptopy, USB, github, SO... okrem priateľa na telefóne

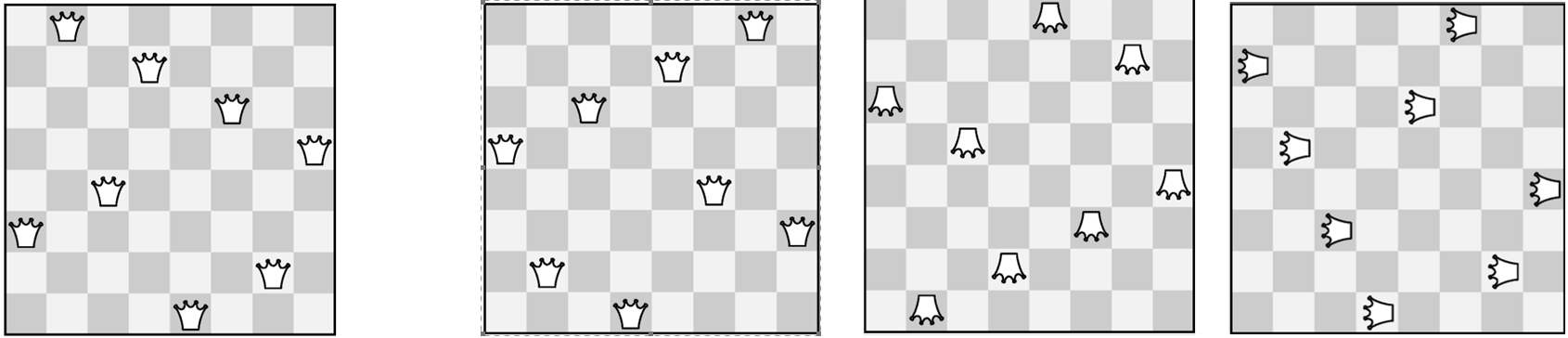
Obsah: konkurentné/funkcionálne

Midterm obsahuje 5 príkladov s bodmi 5+6+6+6+7=30 bodov, čo je s veľkou rezervou viac ako 20 sľúbených.

Čo uhráte, to máte, žiadne škálovanie bodov \*(2/3) nebude.

Máte k dispo [template súborov](https://list.fmph.uniba.sk/index.php/tasks/download_hidden_file/2516/TUlEMjAyMC56aXA_), do ktorých dopisujete.

**1. MID20 - Queens**



Riešenie problému 8-dám reprezentujeme ako permutáciu riadkov, v ktorých sa jednotlivé dámy nachádzajú, teda permutáciu čísel 0..7. V tomto príklade budete zisťovať, ktoré riešenia sú **symetricky rovnaké**. Všetko ilustrujeme na zobrazenom  riešení [5,0,4,1,7,2,6,3] (1.obr.). Jednotlivé indexy zodpovedajú riadkom, v ktorých sa dámy nachádzajú. Následujúce obrázky 2, 3, 4 zobrazujú symetriu horizontálnu, vertikálnu a diagonálnu. V následujúcich funkciách dostanete riešenie problému N-dám, kde **N nie je konštanta 8**, ale dĺžka zoznamu.

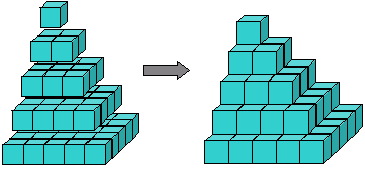
* **[2 body]** Definujte funkcie **symHoriz, symVertik::[Int]->[Int],** ktoré vrátia horizontálne a vertikálne symetrické riešenie, teda pre vstup [5,0,4,1,7,2,6,3] vrátia [3,6,2,7,1,4,0,5] (2.obr.) a [2,7,3,6,0,5,1,4] (3.obr.).
* **[2 body]** Definujte funkciu **symDiag::[Int]->[Int]**, ktorá vráti diagonálne symetrické riešenie, teda [1,3,5,7,2,0,6,4] (4.obr.). Ktorú z dvoch uhlopriečok si vyberiete je na vás.
* **[1 bod]** Definujte funkcie **sym90, sym180** a **sym270::[Int]->[Int]**, ktoré vrátia riešenie otočené o 90, 180 a 270 stupňov v niektorom smere, teda [4,6,0,2,7,5,3,1], [4,1,5,0,6,3,7,2], [6,4,2,0,5,7,1,3], v našom príklade je to v smere hodinových ručičiek. **Hint:** tu už moc neprogramujte, skúste to vyskladať z funkcií z prvých dvoch bodov úlohy, heslo k úspechu je skladanie symetrií.
* **[1 bod]** Definujte funkciu **symetrické::[Int]->[Int]->Bool**, ktoré o dvoch riešeniach problému N dám rozhodne, či sú nejako symetrické, a použite to, čo ste doposiaľ vytvorili.

Body: 6

Autor: Peter Borovanský

**2. MID20 - Pyra**

Predstavte si pyramídu na obrázku. Jej n-té poschodie je štvorec n x n kociek jednotkového rozmeru.

**[](https://www.google.com/url?sa=i&url=https%3A%2F%2Fapcentral.collegeboard.org%2Fcourses%2Fresources%2Fcalculus-before-newton-and-leibniz%3Fcourse%3Dap-european-history&psig=AOvVaw2RoaLxB-b31srPmU4-_u2K&ust=1606643297548000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLjIxfn6pO0CFQAAAAAdAAAAABAD)**

**Úlohy:**

**[1 bod]** Definujte funkciu **objem::Int->Integer,** ktorá vypočíta objem (počet kociek) takejto n-poschodovej pyramídy. Zistite a v riešení uveďte, koľko kociek treba na postavenie 1.000 a 1.000.000 poschodovej pyramídy.

**[1 bod]** Definujte nekonečný zoznam pyramídových čísel **pyramidy::[Integer],** ktorý začína [1,5,14, 30 …].

**[1 bod]** Definujte predikát **jePyramidove::Integer->Bool**, ktorý zistí, či nejaké číslo je pyramídové, teda, že existuje pyramída s toľkátimi kockami. Zistite, ktoré z čísel sú pyramídové 627874, 31762480260, 163732360985951, 13768792391566039, 321139443385562398762433961 a uveďte to v riešení.

**[1 bod]** Definujte funkciu **povrch::Int->Integer,** ktorá vypočíta, koľko farby potrebujete na namaľovanie povrchu takejto n-poschodovej pyramídy, ak plocha steny kocky je 1. Myslí sa tým všetky steny, plochy a plôšky, aj zo spodu. Zistite a v riešení uveďte, povrch 1.000 a 1.000.000 poschodovej pyramídy.

**[1 bod]** Definujte funkciu **lepidlo::Int->Integer,** ktorá vypočíta koľko lepidla/cementu potrebujete na zostavenie takejto pyramídy, za predpokladu, že na spojenie dvoch kociek jednou stenou potrebujete jeden diel lepidla. Takže na pyramídu s jedným poschodím nepotrebujete nič, na dvojposchodovú pyramídu potrebujete 4+1=5 jednotiek lepidla, na trojposchodovú pyramídu potrebujete 12+4+4+1=21 jednotiek lepidla.

**Hint:** fromIntegral je konverzia Int -> Integer a jej objavenie nie je pointou úlohy.

**Bonus [1 bod]:** ak aspoň dve funkcie z troch: objem/povrch/lepidlo budú explicitné vzorce, teda žiaden sum, cyklus, fold, či rekurzia, patrí vám bonus. Príklad explicitného vzorca je n\*(n-1) `div` 2.

Body: 5

Autor: Peter Borovanský

**3. MID20 - fold**

**V každej časti úlohy si stačí vybrať jednu z verzií foldr alebo foldl, podľa vašich preferencií.**

Máte naprogramovať ekvivalenty štandardných funkcií, ktoré dobre poznáte z Haskellu, ale len použitím foldl/foldr. Môžete si definovať vlastné pomocné funkcie. Zo štandardných funkcií môžete použiť len elementárne, ako head, tail, length, prípadne nejasností sa spýtajte. **Nepoužívajte** **rekurziu ani list-comprehension**, veď foldl/foldr schémy sú o tom, že rekurziu viete nahradiť...

1. **[1 bod]** Do definície funkcie **elem'::Eq(t)=>t->[t]->Bool** doplňte výrazy za symboly **?**, aby funkcia fungovala rovnako ako **elem**, teda vrátila **True**, ak sa prvok nachádza v zozname, inak **False**. Môžete porovnávať len hodnoty typu **t**.

**elem' x xs = ? foldr ? ? xs**,       **elem' x xs = ? foldl ? ? xs.**

1. **[2 body]** Do definície funkcie **nub'::Eq t =>[t]->[t]** doplňte výrazy za symboly **?**, aby funkcia fungovala rovnako jako nub, teda vrátila množinu prvkov pôvodného zoznamu, na poradí prvkov nezáleží. A tiež zložitosť výsledného riešenia nemusí byť n\*log n.

**nub' xs = ? foldr ? ? xs**,          **nub' xs ys = ? foldl ? ? xs.**

1. **[3 body]** Do definície funkcie **sort'::Ord t =>[t]->[t]** doplňte výrazy za symboly **?**, aby funkcia fungovala rovnako ako sort, teda vrátila utriedený vstupný zoznam. Na zložitosti vami vybraného triediaceho algoritmu nezáleží, kľudne aj BubbleSort, ale s foldami, bez rekurzie.

**sort' xs = ? foldr ? ? ?**,          **sort' xs ys = ? foldl ? ? ?.**

Body: 6

Autor: Peter Borovanský

**4. MID20 - Search**

Nech celočíselná funkcia má vlastnosť, že pre ľubovoľné x, y platí:

* f x y < f x       (y+1)
* f x y < f (x+1) y

Vlastne, keby ste si vypísali jej hodnoty do 2D tabuľky, tak hodnoty v každom riadku rastú, aj hodnoty v každom stĺpci rastú. Vašou úlohou je nájsť pre daný kľúč key také 0<= i, j <= limit key , že f i j = key.

Príklad, ktorý použijeme v testovaní je funkcia f i j = i^3 + j^3. Zamyslite sa, že triviálne spĺňa vlastnosť. Pre túto funkciu zrejme stačí testovať po limit(key) = ³√key, teda tretiu odmocninu z key. Ich definície sú vám k dispozícii:

**sumcubes :: Integer -> Integer -> Integer  
sumcubes i j = i^3 + j^3  
cbrt :: Integer -> Integer  
cbrt x = round (fromIntegral x\*\* (1/3))**

Samozrejme, naivné riešenie je prehľadať celú tabuľku, teda dvojitý cyklus. To môžete zapísať takto...

**for(int i=0; i <= limit(key); i++)  
   for(int j=0; j <= limit(key); j++)  
     if (f(i,j) == key) return true;  
return false;**

**[1 bod]** Prepíšte toto naivné riešenie do Haskellu ako funkciu:

**lookfor :: (Eq t) => t -> (Integer -> Integer -> t) -> (t -> Integer) -> Bool**  
**lookfor key f limit = ...**

Nájdite odpovede pre:

* **lookfor 1729 sumcubes cbrt**
* **lookfor 2020 sumcubes cbrt**
* **lookfor 87539319 sumcubes cbrt**
* **lookfor 6963472309248 sumcubes cbrt**

Toto riešenie je pomalé, kvadratické, a NIJAKO nevyužíva vlastnosť funkcie, že jej hodnoty ostro rastú v oboch vstupných parametroch. V tomto [linku](https://www.techiedelight.com/young-tableau-insert-search-extract-min-delete-replace/#Search), časti "Search operation in Young tableau" si naštudujte, ako sa to robí vo vašom preferovanom jazyku. Krátko zhrunuté, začnete na indexoch (i,j)=(0, limit key) a opakujete:

* ak ste s indexami (i,j) mimo 0..limit key, končíte, nenašli ste a nenachádza sa,
* ak je hodnota menšia ako key, urobíte i++, a cyklíte sa,
* ak je hodnota väčšia ako key, urobíte j--, a cyklíte sa,
* ak je hodnota KEY, končíte, našli ste key.

Prečo to funguje, je na zamyslenie, nakreslite si štvorcové pole s danou vlastnosťou, vyskúšajte a pochopíte. Zložitosť tohoto algoritmu nie je kvadratická, ale lineárna od limit key.

**[2 body]** Implementujte tento spôsob hľadania a definujte

**linearlookfor :: (Ord t) => t -> (Integer -> Integer -> t) -> (t -> Integer) -> Bool**  
**linearlookfor key f limit = ...**

Nájdite odpoveď pre:

* **linearlookfor 1729 sumcubes cbrt**
* **linearlookfor 2020 sumcubes cbrt**
* **linearlookfor 87539319 sumcubes cbrt**
* **linearlookfor 6963472309248 sumcubes cbrt**

**[3 body]** Modifikujte predchádzajúci kód, tak aby vrátil všetky dvojice indexov i,j, že f i j = key. Opäť, kvadratické riešenie je nezaujímavé, inšpirujte sa lineárnym algoritmom, a trochu ho upravte tak, aby ste dostali:

**allPairs :: (Ord t) => t -> (Integer -> Integer -> t) -> (t -> Integer) -> [(Integer, Integer)]**

**allPairs key f limit = ...**

Nájdite odpoveď pre:

* **allPairs 1729 sumcubes cbrt**
* **allPairs 2020 sumcubes cbrt**
* **allPairs 87539319 sumcubes cbrt**
* **allPairs 6963472309248 sumcubes cbrt**
* **allPairs 48988659276962496 sumcubes cbrt**
* **allPairs 24153319581254312065344 sumcubes cbrt**

Pozn.: To, čo ste programovali, sú Youngove tablá s aplikáciou na TaxiCab čísla. Oplatí sa pozrieť si film **The Man Who Knew Infinity** o indickom matematikovi Srinivasa Ramanujan.

[**https://www.techiedelight.com/young-tableau-insert-search-extract-min-delete-replace/#Search**](https://www.techiedelight.com/young-tableau-insert-search-extract-min-delete-replace/#Search)

[**https://en.wikipedia.org/wiki/Taxicab\_number**](https://en.wikipedia.org/wiki/Taxicab_number)

[**https://oeis.org/A011541/list**](https://oeis.org/A011541/list)

Body: 6

Autor: Peter Borovanský

**5. MID20 - GO**

**[1bod] A)** Čo vypíše tento program, napíšte jeho výstup:  
**func main() {**  
**fmt.Println("Start")**  
**go func () {**  
**fmt.Println("begin")**  
**time.Sleep(1000)**  
**fmt.Println("end")**  
**}()**  
**fmt.Println("Stop")**  
**}**

-----------------------------------------------------------------------------------------

**[1bod] B)** Čo vypíše tento program, napíšte miesto UTC času stačí sekunda behu:

**func main() {**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStart")**  
**go func () {**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tbegin1")**  
**time.Sleep(1\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tend1")**  
**}()**  
**go func () {**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tbegin2")**  
**time.Sleep(2\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tend2")**  
**}()**  
**time.Sleep(3\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStop")**

**}**

-----------------------------------------------------------------------------------------

**[1bod] C)** Čo vypíše tento program, napíšte v vpravo, miesto UTC času stačí sekunda behu:

**func main() {**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStart")**  
**go func () {**  
**for i := 1; i < 30; i++ {**  
**time.Sleep(1\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Printf("\ttick%v\n",i)**  
**}**  
**}()**  
**time.Sleep(10\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStop")**

**}**

-----------------------------------------------------------------------------------------

**[1bod]** **D)** Bude sa vo výpise tohoto programu **zaručene striedať tick a tack**? Nejako to zdôvodnite.

-----------------------------------------------------------------------------------------

**[1bod] E)** Ak nie, tak ho opravte tak, aby sa zaručene striedali.

Ak áno, netreba nič (ak ste správne odpovedali na predošlú otázku, aj zdôvodnili).

**func main() {**  
**rand.Seed(time.Now().UnixNano())**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStart")**  
**ch := make(chan bool)**  
**go func () {**  
**for \_ = range ch {**  
**fmt.Print("tick")**  
**ch <- true**  
**time.Sleep(time.Duration(rand.Intn(5))\*time.Millisecond)**  
**}**  
**}()**  
**go func () {**  
**for \_ = range ch {**  
**fmt.Print("tack")**  
**ch <- false**  
**time.Sleep(time.Duration(rand.Intn(5))\*time.Millisecond)**  
**}**  
  
**}()**

**ch <- true**  
**time.Sleep(30\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStop")**

**}**

-----------------------------------------------------------------------------------------

**[1bod]** **G)** Bude sa vo výpise tohoto programu **zaručene striedať tick, tack a tuck** ? Nejako to zdôvodnite. Jediný rozdiel oproti programu z minulej strany, že pribudla ďalšia takmer identická go rutina, ktorá robí tuck. Nič viac.

-----------------------------------------------------------------------------------------

**[1bod] H)** Ak nie, tak ho opravte tak, aby sa zaručene striedali v abecednom poradí (tack-tick-tuck).

Ak áno, netreba nič.

**func main() {**  
**rand.Seed(time.Now().UnixNano())**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStart")**  
**ch := make(chan bool)**  
**go func () {**  
**for \_ = range ch {**  
**fmt.Print("tick")**  
**ch <- true**  
**time.Sleep(time.Duration(rand.Intn(5))\*time.Millisecond)**  
**}**  
**}()**  
**go func () {**  
**for \_ = range ch {**  
**fmt.Print("tack")**  
**ch <- false**  
**time.Sleep(time.Duration(rand.Intn(5))\*time.Millisecond)**  
**}**  
  
**}()**  
**go func () {**  
**for \_ = range ch {**  
**fmt.Print("tuck")**  
**ch <- false**  
**time.Sleep(time.Duration(rand.Intn(5))\*time.Millisecond)**  
**}**  
**}()**  
**ch <- true**  
**time.Sleep(30\*time.Second)**  
**fmt.Print(time.Now().UTC()); fmt.Println("\tStop")**

**}**

-----------------------------------------------------------------------------------------

Body: 7

Autor: Peter Borovanský