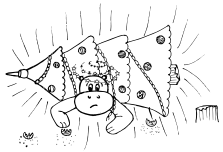
Korespondenční Seminář z Programování

31. ročník KSP Leden 2019

Milí řešitelé, milé řešitelky!



Vánoce už jsou za námi a před námi se otevírá rok 2019. A společně s ním se před vámi otevírá zadání další série KSP, opět plné rozmanitých úložek k řešení.

Připomínáme, že do celkového hodnocení se Vám z každé série započte **5 nejlépe vyřešených úloh**. Každému řešiteli, který získá v tomto ročníku z každé série alespoň 5 bodů, pošleme KSP propisku, blok, placku a třeba i něco navíc.

Díky řešení KSP se také můžete vyhnout přijímacím zkouškám na MFF UK! Stačí, když získáte alespoň polovinu bodů z ročníku (tedy 150 bodů) a my vám vystavíme osvědčení, díky kterému vás přijmou na MFF bez zkoušek. Pozor ale: pokud studujete poslední ročník střední školy a chcete letošní osvědčení využít, musíte mít potřebné body již po čtvrté sérii.

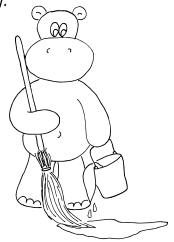
Termín série: pondělí **4. března v 8:00 ráno**

Odevzdávání: Přes web na adrese *https://ksp.mff.cuni.cz/submit/*

Odměna série: **Sladkou odměnu** si vysloužíte, kdo **z každé úlohy série získá alespoň 2 body**.

Třetí série třicátého prvního ročníku KSP

Bylo nebylo, za sedmero horami, devatero řekami, dvanáctero městy a dvacatero nukleonovými urychlovači stálo jedno malé hospodářství, ve kterém bydlel otec s dcerou. Matka jim nedávno zemřela, a tak se jal otec znovu oženiti. Jeho druhá žena do domu přivedla také své dvě dcery. Tím však pro ubohou dívku nastaly zlé časy.



Nevlastní sestry ji nutily tvrdě pracovat, uklízet, vařit, ... prostě vše, co si umanuly. A protože dívenka mívala často tváře umouněné od popela, začaly jí říkat Popelka. Na rozdíl od Popelky její nevlastní sestry doma nehnuly ani prstem a zajímaly se jen o drahé šaty a šperky. Chudák otec pak musel každý druhý týden na nákupy do města.

31-3-1 Shánění látky — 10 bodů

Ve městě se nachází spousta obchodů nabízejících látky. Každý obchod však prodává jeden metr látky za jinou cenu. Kromě toho cestování také něco stojí, jelikož se na všech cestách vybírá mýtné. Nevlastní dcery posílají otce na nákup často, jenomže pokaždé chtějí koupit jiné množství látky. Pravidelné nákupy jsou poměrně finančně náročné, proto se otec snaží pokaždé naplánovat trasu tak, aby zaplatil co nejméně peněz. Chudák už je ale z toho všeho plánování zoufalý.

Město si můžeme představit jako neorientovaný graf, kde vrcholy jsou obchody a hrany jsou cesty mezi nimi. Každá hrana je ohodnocená cenou, kterou zaplatíme, když přes ni pojedeme. Každý vrchol je ohodnocen cenou za jeden metr látky. Jeden z vrcholů je označen jako výchozí, v něm otec s dcerami bydlí a vždy po nákupu se tam chce vrátit. Navrhněte algoritmus, který pomůže otci vyřešit jeho problém s plánováním. Algoritmus si nejprve něco předpočítá a pak bude dostávat dotazy typu „Jak nejlevněji pořídit ℓ metrů látky?“ Zajímá nás jak doba předvýpočtu, tak časová složitost jednoho dotazu. Můžete předpokládat, že dotazů bude řádově tolik, co obchodů, a že v každém obchodě lze koupit libovolné množství látky.

Jakmile otec dorazil domů, nevlastní dcery se na nakoupené látky slétly jako včely na med, začaly si je zkoušet a nakrucovat se před zrcadly. Popelka šla otce také pozdravit. Všimla si, že má ve vlasech zamotanou lískovou větvičku. Musela se mu tam dostat cestou z města... Vymotala ji a odnesla do svého kamrlíku. Sestry by otci nedovolily přivézt Popelce z města žádné cennosti, ale lískovou větvičku Popelce nechaly. Alespoň má od otce také nějaký dárek.

Jen co se sestry dostatečně nabažily pohledů na látky, vytáhl otec z brašny ještě jedno překvapení: zlaté náhrdelníky. Ve zdejším království se totiž chystal veliký ples, na kterém si prý mladý princ měl vybrat jednu dívku, kterou si vezme za ženu. Otec proto chtěl, aby se dcery mohly dostatečně vyparádit.

31-3-2 Cennější náhrdelník — 13 bodů

Obě nevlastní dcery dostaly od otce krásný zlatý náhrdelník. Protože jsou sestry velmi závistivé, hned se začaly dohadovat, čí náhrdelník je vzácnější. Hádaly se spolu do té doby, než se jedna z nich urazila a zavřela se do svého pokoje. Jediný, s kým je ochotna komunikovat, je jeden ze služebníků.

Sestry chtějí prostřednictvím služebníka zjistit, který náhrdelník je dražší. Na každém náhrdelníku je sice cenovka, ale protože spolu v současné době odmítají komunikovat napřímo, chtějí minimalizovat množství informací, které si spolu vymění. Naštěstí má každá z nich v pokoji výdobytek moderní doby – generátor náhodných čísel, který funguje tak trochu magicky. Generuje náhodná čísla tak, že i-té náhodné číslo první sestry je shodné s i-tým náhodným číslem druhé sestry. Vymyslete způsob, jakým sestry s pravděpodobností alespoň 75 % dokáží zjistit, čí náhrdelník je dražší.

Jak je v kryptografii běžným zvykem, pojmenujeme si sestry Alice a Bob. Každá má jedno binární číslo délky N a zázračný generátor, který jim oběma generuje stejná náhodná čísla. Cílem je s pravděpodobností alespoň 75 % zjistit, která ze sester má větší číslo, a to na co nejmenší počet vyměněných bitů.

Lehčí varianta (za 8 bodů): Zjistěte jen, zda jsou oba náhrdelníky stejně drahé.

Několik týdnů se sestry připravovaly na ples, nechaly si ušít z látek nádherné šaty a učesat ohromující účesy. Všichni se na ples moc těšili. Když konečně nastal den D a sestry se nachystaly k odchodu, velmi je překvapilo, že Popelka chce jít taky. Vždyť tu má ještě spoustu práce! Popelka však tvrdila, že má vše hotové. Poslední týden tvrdě pracovala, aby měla nyní volno. To sestry naštvalo. Vzaly popel a hrách a sesypaly ho na zem.

Popelka byla celá zoufalá. „Vždyť tohle nemůžu v nějakém konečném čase stihnout roztřídit!“ „Ťuk, ťúúúk, ťúúúk, ťuk“ ozvalo se najednou. To na okno ťukal malý holoubek. Popelka, která byla zběhlá v luštění morseovky, pochopila, že se jí holoubek snaží nabídnout pomoc.

31-3-3 Přebírání hrachu — 9 bodů

Na podlaze se nachází sesypaný hrách s popelem a mezi tím poskakuje několik holoubků a vrabčáků. Chtějí Popelce pomoct vysbírat všechen hrách. Pro každou kuličku hrachu chceme zjistit, zda ji nějaký ptáček dokáže sezobnout a přenést do ošatky. Holoubci i vrabčáci se však pohybují každý jinak. Vrabčáci poskakují rovně dopředu, dozadu, doleva i doprava, zato holoubci chodí šikmo do všech čtyř stran.

Celou situaci si lze představit jako rozmístění figurek na šachovnici. Holoubci představují černé střelce, vrabčáci černé věže a kuličky hrachu pak bílé pěšáky. Pro každou bílou figurku chceme zjistit, zda ji dokážeme v jednom tahu nějakou černou figurkou vyhodit.

Toto je praktická open-data úloha. V odevzdávacím systému si necháte vygenerovat vstupy a odevzdáte příslušné výstupy. Záleží jen na vás, jak výstupy vyrobíte.

Formát vstupu: Na prvním řádku vstupu se nachází dvě čísla B a C, a to počet kuliček hrachu (neboli bílých figurek) a počet ptáčků (neboli černých figurek). Na dalších B řádcích se nachází souřadnice kuliček hrachu jako dvojice čísel oddělených mezerou, číslo řádku a číslo sloupce udávajících, kde se kulička nachází. Na dalších C řádcích se pak nachází pozice ptáčků – každý takový řádek obsahuje znak `H` nebo `V` a dvojici čísel udávajících řádek a sloupec, kde ptáček stojí (opět vše oddělené mezerami). Znak `H` značí holoubka (pohybuje se jako střelec) a znak `V` vrabčáka (pohybuje se jako věž). Figurky na vstupu mohou být seřazené náhodně.

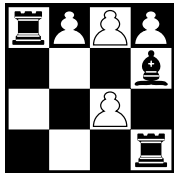
Formát výstupu: Na B řádků výstupu vypište (ve stejném pořadí, jako na vstupu) pro každou kuličku hrachu `ANO` nebo `NE` podle toho, jestli existuje nějaký holoubek nebo vrabčák, který může tuto kuličku hrachu jedním tahem sezobnout.

Ukázkový vstup:

```
4 3
0 1
0 2
0 3
2 2
V 0 0
H 1 3
V 3 3
```

Ukázkový výstup:

```
ANO
ANO
NE
ANO
```



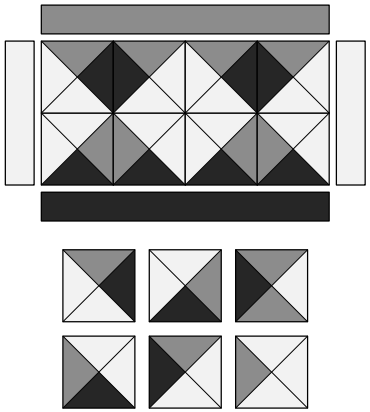
Poznámka: Všimněte si, že stejnou černou figurkou (střelcem) můžeme vzít dva bílé pěšce. Naopak bílého pěšce vpravo nahoře nemůžeme vzít ani spodní věží (v cestě stojí střelec), ani levou věží (v cestě stojí jiné bílé figurky).

Sestry netušily, že si Popelka rozumí s ptáčky, a měla tak vše za chviličku hotové. Ples by ještě stihla, ale kde teď narychlo sehnat šaty? Rozesmutnila se a rozběhla se do svého kamrlíku, kde se rozplakala. Proč jen má takové závistivé sestry, které jí nedovolí ani podívat se na ples! Rozhodla se, že si spraví náladu jídlem, a vzpomněla si na lískový oříšek od otce. „Ten bude určitě velice chutný,“ pomyslela si. Když ho však rozlouskla, nemohla uvěřit svým očím. Zevnitř vykukoval cíp šatů! Popelka zatáhla a z oříšku vypadly tak nádherné šaty, jaké ještě nikdy neviděla.

Když Popelka dorazila na ples, všichni byli okouzleni její krásou. Nevlastní sestry ji nepoznaly, protože doma nenosila nic jiného než otrhané šaty umazané od popela. Popelky si všiml dokonce i samotný princ a začal se prodírat davem tanečníků, jen aby si s ní mohl zatančit.

31-3-4 Dláždění sálu — 11 bodů

Ples se odehrává v největším a nejhonosnějším sálu na zámku, který nedávno prošel rekonstrukcí. Podlaha je nyní vydlážděná krásně barevnými dlaždicemi. Občas spolu ale sousední dlaždice neladí. Uměli byste sál vydláždit lépe?



Sál má obdélníkový půdorys o rozměrech $K \times N$ metrů a každá z jeho čtyř stěn je obarvená jednou barvou. Naším úkolem je sál vydláždit barevnými dlaždicemi o rozměrech 1×1 metr. K tomu jich máme dispozici hned několik druhů, přičemž každý druh dlaždice má pevně určené obarvení hran. Z estetických důvodů také s dlaždicemi nemůžeme otáčet, tj. každý druh dlaždice má určeno, která hrana bude otočená na sever. Od každého druhu můžeme použít neomezeně dlaždic. Umíme sál vydláždit, aby přiléhající hrany sousedních dlaždic a hrany sousedící se stěnami sálu barevně ladily? Zajímá nás časová a paměťová složitost pouze

Pár slov závěrem

Pokud se chcete dozvědět o pravděpodobnosti více, můžeme vřele doporučit skvělé přednášky z Harvardu. Jejich videozáznamy jsou dostupné na YouTube.[8]

Až si někdy budete číst o pravděpodobnosti, nejspíš se setkáte s axiomy pravděpodobnosti zavedenými jinak, než jak jsme je zavedli my. Ty naše kuchařkové jsou intuitivnější, ale bohužel se nedají jednoduše zobecnit na nekonečné množiny. O „plnotučných“ Kolmogorových axiomech pravděpodobnosti se můžete dočíst ve Wikipedii.[9]

Další jednoduché randomizované algoritmy a datové struktury najdete v knížce Průvodce labyrintem algoritmů.[10] Inspirovat vás také může seriál z 16. ročníku KSP.[11]

Pokud by vám i tak něco nebylo jasné, ptejte se na našem fóru.

Martin „Medvěd“ Mareš & Jakub Tětek

[8] `https://www.youtube.com/playlist?list=PL2SOU6wwxB0uwwH80KTQ6ht66KWxbzTIo`
[9] `https://en.wikipedia.org/wiki/Probability_axioms`
[10] `http://pruvodce.ucw.cz/`
[11] `https://ksp.mff.cuni.cz/tasks/16/`

matfyz

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Diskusní fórum: *https://ksp.mff.cuni.cz/forum/*

Chcete-li s námi komunikovat bezpečně, můžete si ověřit náš HTTPS certifikát – jeho SHA1 fingerprint je: `E9:DB:EE:C6:62:BC:14:DE:09:E4:E8:97:DC:36:0E:87:B3:50:B0:01`.

vzhledem k N, můžete předpokládat, že hodnota K a seznam povolených druhů dlaždic jsou pevně dané.

V příkladu máme k dispozici šest druhů dlaždic, z nichž čtyři se od sebe liší jenom otočením (my však dlaždicemi otáčet nemůžeme, takže je považujeme za různé). Znázorněn je sál s $K = 2$ a $N = 4$ a jeho (jediné) vydláždění. Rozmyslete si, že pro $K = 2$, tyto druhy dlaždic a obarvení stěn umíme vydláždit právě všechny sály se sudou šířkou.

Když už se blížila půlnoc, Popelka si uvědomila, že se musí dostat domů dříve než její sestry, aby nikdo nepoznal, že na plese byla taky. Omluvila se tedy princi a zamířila ke schodům vedoucím ven z paláce. Princ se však během plesu stihl do Popelky zamilovat, a tak nezaváhal ani vteřinu a okamžitě se za ní rozběhl. „Neodcházej! Vždyť já ani neznám tvé jméno!“ zavolal za Popelkou. Jak se Popelka na schodech za princem ohlédla, ztratila na chvilku rovnováhu a nedopatřením se jí vyzul jeden střevíček. Princ jí však byl natolik v patách, že nechala střevíček střevíčkem a vyběhla ven z paláce.

Princ byl tuze smutný, že mu právě prchla největší láska jeho života. Jediné, co mu po ní zůstalo, byl onen střevíček. Rozhodl se, že pomocí něj neznámou princeznu vypátrá.

31-3-5 Hledání princezny — 11 bodů

Princ chce zjistit, komu střevíček padne. Jeho království je ale opravdu rozlehlé, a tak není v jeho silách zajít za každým osobně. Naštěstí však království o každém poddaném eviduje spoustu informací, včetně adresy bydliště a velikosti nohy (čirou náhodou je celý soupis seřazený zrovna podle velikosti nohy). Navíc je dvorní švec schopen podle předlohy vyrobit střevíček na chlup stejný jako originál. Výroba jednoho takového střevíčku trvá přibližně jednu hodinu.

Princ dostal nápad: Nechal po celém království vyhlásit, že každou hodinu pošle jednomu z poddaných střevíček, aby si ho vyzkoušel a v odpovědi poslal, zda mu je malý, velký, nebo akorát. Princ, znalý binárního vyhledávání, počítal s tím, že tak svou lásku dokáže najít v čase logaritmickém k počtu poddaných. To se ale přepočítal! Královská pošta už má svá nejlepší léta za sebou, a tak poslání střevíčku poddanému a následné doručení odpovědi trvá K hodin, kde K může (ale nemusí) být závislé na počtu poddaných v království. Jak má princ postupovat?

Trošku formálněji řečeno: Máme setříděné pole N čísel, ve kterém chceme najít konkrétní hodnotu. Dotaz můžeme poslat jednou za hodinu, ale trvá K hodin, než nám zpátky přijde odpověď. Dotazů můžeme mít na cestě více. Najděte časově efektivní algoritmus v závislosti na K.

Pátrání trvalo už několik týdnů a princ už začínal být celý zoufalý, že svou lásku nikdy nenajde. Až byl jednoho dne poslán střevíček také Popelce. Nevlastní sestry si byly jisté, že Popelce střevíček patřit nemůže, a tak jí ho ani nechtěly dát vyzkoušet. Otec se však (poprvé za celý svůj život) postavil proti jejich vůli a střevíček Popelce přinesl.

Jak byli všichni překvapeni, když Popelce střevíček padl jak ulitý. Princ nechal pro Popelku okamžitě poslat kočár a odvezl si ji k sobě na zámek. Za necelý týden se pak konala velkolepá svatba.

A žili spolu šťastně až do smrti...

Zuzka Urbanová & Klárka Tauchmanová

[1] `https://doc.qt.io/qtforpython/`
[2] `https://doc.qt.io/qt-5/`
[3] `http://ksp.mff.cuni.cz/encyklopedie/python-pip.html`

31-3-6 Model–ViewController — 15 bodů

Třetí díl seriálu pokračuje vyráběním simulátoru dopravy na křižovatce. Minule jsme si slíbili, že v programech konečně uklidíme a začneme programovat čistě, tak si to pojďme splnit.

Prvně je však třeba učinit důležité oznámení. Vývojáři Qt5 převzali projekt PySide, což je, stejně jako PyQt, rozhraní knihovny Qt pro Python. Vytvořili PySide2 jako oficiální rozhraní Qt5 pro Python. My tedy taktéž opustíme PyQt5 a budeme pokračovat s PySide2. Práce s ním je, alespoň pro naše účely, stejná jako s PyQt5. V našich programech by se nemělo téměř nic změnit, kromě toho, že nebudeme psát `from PyQt5 ...`, ale `from PySide2 ...`; to je jednoduché, ne?

Přece jenom se ale něco změnilo:

- Data přečtená ze socketu je třeba regulérně dekódovat skrze knihovnu Qt; už nedostáváme pythoní `bytearray`, nýbrž `QByteArray`, který dekódujeme pomocí objektu `QTextCodec`.
- Metody předávané jako cíl událostí je vhodné označit dekorátorem `@Slot(...)`, ale není to nutné.
- Pro spuštění programu nelze volat `exec()`, je nutné použít `exec_()`.

Od nynějška také čerpáme z dokumentace přímo pro PySide2[1] na webu Qt; stále je však třeba pro podrobnější popis a detaily jednotlivých metod sáhnout do dokumentace pro C++.[2]

Debian Buster již PySide2 obsahuje; pokud nemáte balíčky pro svůj systém, použijte Pip, stejně jako v prvním díle. Pokud nevíte, co je Pip nebo jak jej požít, napsali jsme návod na Pip.[3]

```
$ pip3 install pyside2
```

Návrhový vzor Model–View–Controller (MVC)

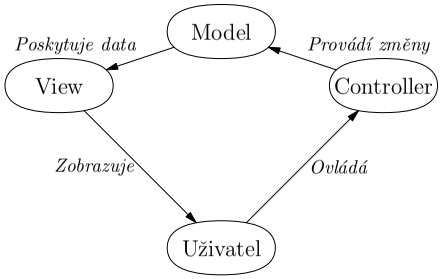
Základem programu je **datový model**. V případě našeho simulátoru dopravy na křižovatce to jsou informace o poloze aut. Do modelu patří i časovače, které posílají auta pryč.

Uživatel by s datovým modelem neměl přijít do styku. Data jsou uložena ve strojově čitelné formě a přistupuje se k nim pomocí hezkého a milého API.

Aby však uživatel nepřišel zkrátka, je tady **náhled**, čili **View**. To je ta část GUI, která uživateli oznamuje nějaké infor-

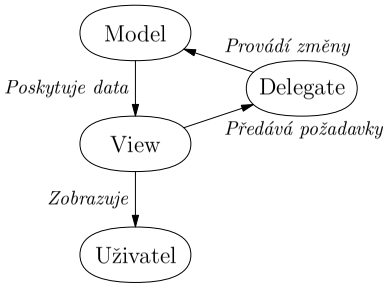
mace; například vypisovátko jednotlivých aut a chodců ve sledovaném úseku. Model informuje View o změnách zobrazovaných dat, například pokud přijede další auto. Nebo si View naopak pravidelně říká Modelu o data.

Na opačné straně je **Controller**, čili **ovladač**. Uživatel prostřednictvím Controlleru ovládá Model. Controller tedy překládá například stisk tlačítka na volání nějakého vnitřního API. Controller také ovládá přímo View, pokud není třeba volat Model.



Hezky napsaný program pak dokáže mít několik různých rozhraní, mezi kterými je možno si vybrat; například jedno v Qt, jedno jako webovou aplikaci, jedno v terminálu v Curses a jedno čisté API pro skriptování. *To je takový svatý grál programování s uživatelským rozhraním, ne povinnost. Je však dobré pamatovat na to, že má být program takto rozdělitelný, už při prvotním rozmýšlení. Ušetříte si tak spoustu práce.*

Qt nicméně počítá s tím, že View a Controller jsou propojené do jednoho modulu, proto se tomuto modelu v Qt říká pouze Model–View. O změny dat se pak stará **Delegate**, což je vlastně ta část Controlleru, která komunikuje s Modelem; o komunikaci s uživatelem se stará View.



Simulátor dopravy: Model

Datový model je v našem simulátoru seznam aut a chodců, které máme aktuálně v oblasti. Model zdědíme z třídy `QAbstractListModel`, která do sebe balí seznam – to přesně potřebujeme. V ní potřebujeme nadefinovat dvě metody, které volá view, když si chce přečíst data:

- `rowCount` vrací počet řádků v seznamu (jako číslo)
- a `data` vrací samotná data uložená v modelu.

Qt počítá s tím, že seznam může být dvourozměrný; na tento model se tak může napojit například `QTableView`, který zobrazuje tabulku. My použijeme `QListView`, viz níže. Metoda `data` tedy jako první argument dostane objekt typu `QModelIndex`, ze kterého získáme číslo řádku zavoláním metody `row`.

Jako druhý argument pak dostaneme číselnou hodnotu role (`Qt.ItemDataRole`);[4] v Qt jsou definovány konstanty jako `Qt.DisplayRole`, `Qt.DecorationRole` nebo například `Qt.ToolTipRole`, které určují, v jakém kontextu si View žádá data. My pro začátek použijeme pouze `DisplayRole`, pro kterou se vrací řetězec (`str`) pro daný řádek.

Metoda tedy ověří, jestli je index v povoleném rozsahu, jestli je správná role, a vrátí příslušnou reprezentaci cestovatele. Pro `DisplayRole` je to textová reprezentace.

Data v modelu je třeba upravovat přidáváním a ubíráním řádků podle toho, jak přibývají a ubývají cestovatelé. Na to si napíšeme vlastní metody `add` a `remove`. Ještě je třeba šťouchnout do View; proto příslušné části kódu obalíme voláním `beginInsertRows` a `endInsertRows`, resp. `beginRemoveRows` a `endRemoveRows`. Zbytek zařídí model za nás.

K modelu připojíme síťové rozhraní `CrossingNetwork`, které nyní nedělá nic jiného než most mezi modelem a síťovým socketem.

Nakonec si pořídíme `QListView`. Tomu stačí připojit model a máme prakticky hotovo. Jednoduchou implementaci simulátoru s využitím techniky Model–View si můžete prohlédnout zde:
`http://ksp.mff.cuni.cz/viz/31-3-6-sample.py`

Úkol 1 [3b]: Přidejte cestovatelům ikonky. Ikonku načtete zavoláním `QIcon(soubor)`, přičemž třídu `QIcon` importujete z balíku `PySide2.QtGui`. Pokuste se uložit ikonu na co nejrozumnější místo tak, aby se načetla právě jednou. Ikonu vracejte z metody `data` pro roli `Qt.DecorationRole`.

Pokud se vám ikonky nechce hledat jinde nebo vyrábět vlastní, můžete si stáhnout amatérské (ikonku auta[5] a chodce[6]) od nás.

Úkol 2 [5b]: Upravte simulátor tak, aby se místo ID a rychlosti zobrazovala aktuální poloha cestovatele ve sledovaném úseku (absolutní v metrech a relativní v procentech). ID a rychlost nechť se zobrazují v tooltipu (`Qt.ToolTipRole`).

Ke spočítání polohy můžete využít metodu `position` třídy `Traveller`. Polohu v zobrazení nemusíte nikterak aktualizovat, co s tím, si ukážeme za chvíli. Vzhledem k nepřesnosti časovačů se může stát, že se bude ze začátku zobrazovat záporná poloha; to řešit nemusíte.

Proxy Model

Mezi samotným datovým modelem a View potřebujeme často ještě nějaké vrstvy, které data nějak transformují. Je tak možné například data třídit nebo s nimi dělat ledasjaké jiné psí kusy.

Takovým vrstvám se říká *Proxy modely* a jsou to třídy, které mají navenek téměř stejné rozhraní jako běžný model, ale ve skutečnosti nenesou žádná data. Místo toho jsou připojené na jiné modely, které jim data poskytují.

[4] `https://doc.qt.io/qtforpython/PySide2/QtCore/Qt.html#PySide2.QtCore.PySide2.QtCore.Qt.ItemDataRole`
[5] `http://ksp.mff.cuni.cz/viz/31-3-6-car.ico`
[6] `http://ksp.mff.cuni.cz/viz/31-3-6-pedestrian.ico`

Než lemma dokážeme, rozmysleme si, co říká o našem čekání na šestku: Pravděpodobnost šestky je $1/6$, takže v průměru hodíme $1/(1/6) = 6$-krát.

Nyní důkaz: Nechť U je náhodná proměnná, která nám říká, při kolikátém pokusu se utrhlo ucho. Střední hodnotu budeme chtít spočítat podle definice:

$$\mathbf{E}[U] = \sum_{i=1}^{\infty} i \cdot P[U = i].$$

Potřebujeme tedy znát pravděpodobnosti toho, že ucho se poprvé utrhne v i-tém kroku. Pro $i = 1$ je to triviální: ucho se utrhne hned napoprvé s pravděpodobností p. Pro $i = 2$ uvážíme, že se napoprvé neutrhlo (to má pravděpodobnost $1 - p$) a napodruhé utrhlo (tedy p). Jelikož oba jevy jsou nezávislé, můžeme jejich pravděpodobnosti vynásobit a dostaneme $P[U = 2] = (1 - p) \cdot p$. Pro obecné i se při prvních $i-1$ pokusech ucho neutrhne a v i-tém pokusu utrhne, tedy dostaneme $P[U = i] = (1 - p)^{i-1} \cdot p$.

Teď dosadíme spočtené pravděpodobnosti:

$$\mathbf{E}[U] = \sum_{i=1}^{\infty} i \cdot (1 - p)^{i-1} \cdot p = p \cdot \sum_{i=0}^{\infty} (i + 1)(1 - p)^i.$$

To je nějaká nekonečná suma, u níž budeme věřit, že se sečte na konečné číslo (znalci analýzy mohou použít podílové kriterium konvergence). Jak ji spočítáme?

Označme S hodnotu druhé sumy, platí tedy $\mathbf{E}[U] = pS$. Abychom lépe viděli, co se děje, sumu si rozepíšeme:

$$S = 1 \cdot (1 - p)^0 + 2 \cdot (1 - p)^1 + 3 \cdot (1 - p)^2 + \dots$$

Podívejme se, co se stane po vynásobení obou stran $1 - p$:

$$(1 - p)S = 1 \cdot (1 - p)^1 + 2 \cdot (1 - p)^2 + 3 \cdot (1 - p)^3 + \dots$$

To se nepočítá o nic lépe, ale je to docela podobné výrazu pro S: v jednom případě je u $(1 - p)^i$ koeficient $i + 1$, v druhém i. Zkusíme tedy od sebe oba výrazy odečíst:

$$S - (1 - p)S = (1 - p)^0 + (1 - p)^1 + (1 - p)^2 + \dots$$

Levou stranu můžeme zjednodušit na pS, napravo je nějaká geometrická řada s kvocientem $1 - p$. Tabulkový vzoreček pro součet geometrické řady s kvocientem q říká, že $q^0 + q^1 + q^2 + \dots = 1/(1 - q)$. V našem případě je $q = 1 - p$, takže $pS = 1/(1 - (1 - p)) = 1/p$. My už ovšem víme, že pS je také rovnou hledané střední hodnotě $\mathbf{E}[U]$. Hotovo.

(Mimochodem, kdyby vám vrtalo hlavou, jak se odvozuje vzoreček pro součet geometrické řady, dělá se to trikem velmi podobným tomu našemu: Pokud $G = q^0 + q^1 + \dots$, pak $qG = q^1 + q^2 + q^3 + \dots$. Opět obě řady odečteme a získáme $G - qG = q^0 = 1$. Proto $(1 - q)G = 1$, a tedy $G = 1/(1 - q)$.)

Odbočka k hledání pivota

Lemma o džbánu si vyzkoušíme na analýze algoritmu. V třídícím algoritmu Quicksort potřebujeme najít pivota, který bude přibližně mediánem. Řekněme, že v setříděném pořadí prvků má pivot ležet v prostřední třetině. Pak by stále platilo, že Quicksort poběží v čase $\mathcal{O}(n \log n)$. Jak ale takového pivota najít? Pomůže jednoduchý pravděpodobnostní algoritmus: vybereme pivota náhodně ze zadaných prvků, spočítáme, kolik prvků je menších a kolik větších, a pokud pivot neleží v prostřední třetině, algoritmus prostě zopakujeme.

[7] `http://mj.ucw.cz/papers/numth.pdf`

Jeden průchod algoritmu trvá $\mathcal{O}(n)$. Algoritmus může uspět hned v prvním průchodu, takže v nejlepším případě skončí v lineárním čase. Také by se mohlo stát, že budeme mít smůlu a pokaždé vybereme minimum, a tehdy se algoritmus nezastaví nikdy – to nás ovšem nemusí trápit, tento průběh algoritmu má pravděpodobnost rovnu 0.

Ukážeme, že průměrná časová složitost je také lineární. Střední hodnota času výpočtu je určitě $\mathcal{O}(n)$ krát střední hodnota počtu průchodů (všimněte si, jak jsme zde použili linearitu střední hodnoty). Průchod uspěje, pokud se trefí do prostřední třetiny. Jelikož všechny prvky vybíráme se stejnou pravděpodobností, nastane to s pravděpodobností $1/3$. Podle lemmatu o džbánu tedy musíme udělat v průměru 3 průchody, než se ucho utrhne a my najdeme prvek v prostřední třetině.

Zesilování pravděpodobnosti

Představme si nakonec, že máme nějaký pravděpodobnostní algoritmus, který vždy doběhne rychle, ale nezaručuje správný výsledek. Typickým příkladem je třeba takzvaný Rabinův-Millerův test prvočíselnosti. Nebudeme ho vysvětlovat detailně, podrobnosti najdete například v Medvědových Algoritmech okolo teorie čísel.[7] Důležité ale je, že se chová takto: Dáme-li mu na vstupu prvočíslo, vždy správně odpoví „ANO, je to prvočíslo.“ Složené číslo odhalí s pravděpodobností aspoň $1/2$: pokud mu ho dáme, odpoví s pravděpodobností alespoň $1/2$ NE, jinak ANO.

Poloviční pravděpodobnost chyby není moc užitečně, ale můžeme ji snadno vylepšit tím, že algoritmus k-krát zopakujeme. Číslo budeme považovat za prvočíslo jen tehdy, když se na tom shodlo všech k opakování algoritmu.

Jaká je nyní pravděpodobnost chyby? Pokud je číslo doopravdy prvočíslem, pokaždé to zjistíme správně, takže celkově odpovíme ANO. Pokud je složené, odpověděli bychom špatně (tedy ANO) jen tehdy, kdyby se každé z k spuštění algoritmu zmýlilo. Jelikož tato selhání jsou navzájem nezávislá (algoritmus si pokaždé vygeneruje nová náhodná čísla nezávisle na těch předchozích), pravděpodobnost k selhání je nejvýš $(1/2)^k = 1/2^k$. Už pro $k = 10$ je to méně než $1/1000$.

Tomuto triku se říká *zesilování pravděpodobnosti* (anglicky *probability amplification*). Zatím jsme ho použili pro stlačení pravděpodobnosti chyby pod libovolně nízkou (ale kladnou) konstantu. Někdy se ovšem hodí umět víc.

Co kdybychom na prvočíselnost testovali N čísel? S původním Rabinovým-Millerovým testem bychom se mýlili v průměrně $N/2$ případech (pokud by všechna čísla byla složená). Pokud bychom chtěli stlačit průměrný počet chyb pod konstantu (řekněme pod 1), museli bychom pravděpodobnost chyby v jednom testu stlačit pod $1/N$. Toho dosáhneme, pokud zvolíme $k > \log_2 N$. Tím jsme algoritmus asymptoticky zpomalili ($\mathcal{O}(\log N)$-krát), ale to je poměrně malá cena za tak podstatné snížení chybovosti. Další zvětšování k snižuje chybovost exponenciálně: už pro $k = 2 \log_2 N$ vyjde pravděpodobnost chyby $1/N^2$.

Také se můžeme ptát, jaká je průměrná hodnota náhodné proměnné. Zkusme si to na proměnné D z předchozího příkladu, ale uvažujme obecné pravděpodobnosti stěn kostky $p_1, \ldots, p_6$ (kostka by tedy mohla být falešná). Představme si, že hodíme kostkou N-krát pro nějaké hodně velké číslo N. Jedniček padne přibližně $p_1 \cdot N$, dvojek $p_2 \cdot N$ atd., takže průměr proměnné D vyjde:

$$\frac{1^2 \cdot p_1 \cdot N + 2^2 \cdot p_2 \cdot N + \ldots + 6^2 \cdot p_6 \cdot N}{N}.$$

To také můžeme zapsat jako

$$1^2 \cdot p_1 + 2^2 \cdot p_2 + \ldots + 6^2 \cdot p_6.$$

Je to tedy vážený průměr, v němž roli vah hrají pravděpodobnosti jednotlivých možností. Pro poctivou kostku ($p_1 = \ldots = p_6 = 1/6$) se z toho stane obyčejný aritmetický průměr s hodnotou 91/6.

Obecně můžeme definovat *střední hodnotu* náhodné proměnné X (značíme $\mathbf{E}[X]$) jako průměr hodnot, které proměnná přiřazuje jednotlivým elementárním jevům, vážený pravděpodobnostmi těchto jevů:

$$\mathbf{E}[X] = \sum_\omega X(\omega) \cdot P(\omega),$$

kde suma probíhá přes všechny elementární jevy ω a $X(\omega)$ je hodnota proměnné pro elementární jev ω.

Pokud jsou hodnoty proměnné celá čísla, někdy je praktičtější pro každou hodnotu posbírat všechny elementární jevy, pro které proměnná této hodnoty nabývá. Dostaneme:

$$\mathbf{E}[X] = \sum_{a \in \mathbb{Z}} a \cdot P[X = a].$$

(Nenechte se vyvést z míry tím, že sčítáme nekonečně mnoho členů. Pro konečný počet elementárních jevů je jen konečně mnoho sčítanců nenulových.)

Lze matematicky dokázat (byť my to zde dělat nebudeme), že pokud zprůměrujeme hodně pokusů, bude výsledek blízko $\mathbf{E}[X]$, a pokud budeme průměrovat více a více pokusů, tak se bude přibližovat (jazykem matematické analýzy: průměr bude konvergovat) k $\mathbf{E}[X]$.

Linearita střední hodnoty

Jednou z důležitých vlastností střední hodnoty je, že je lineární. Tím se myslí, že pro libovolné dvě náhodné proměnné platí $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$ a také $\mathbf{E}[cX] = c \cdot \mathbf{E}[X]$ pro každé číslo c. Uvědomte si, že $X + Y$ a cX jsou také náhodné proměnné.

Předtím, než tuto vlastnost dokážeme poctivě, rozmysleme si, že je velmi intuitivní. Představme si, že máme dva randomizované algoritmy používající náhodnost. Označme X a Y náhodné proměnné, které udávají jejich doby běhu. Linearita střední hodnoty pak říká, že když spustíme X a poté Y, bude průměrná celková doba běhu stejná jako součet průměrných dob běhu jednotlivých algoritmů. Podobně pokud algoritmus dvakrát zpomalíme, bude i průměrná doba běhu dvakrát větší. To není nikterak překvapivé.

Nyní matematický důkaz, nejprve pro násobení konstantou. Stačí dosadit do definice střední hodnoty:

$$\begin{aligned}\mathbf{E}[cX] &= \sum_\omega (cX)(\omega) \cdot P(\omega) = \sum_\omega c \cdot X(\omega) P(\omega) \\ &= c \cdot \sum_\omega X(\omega) P(\omega) = c \cdot \mathbf{E}[X].\end{aligned}$$

A teď pro součet:

$$\begin{aligned}\mathbf{E}[X + Y] &= \sum_\omega (X + Y)(\omega) \cdot P(\omega) \\ &= \sum_\omega (X(\omega) + Y(\omega)) \cdot P(\omega) \\ &= \sum_\omega X(\omega) P(\omega) + Y(\omega) P(\omega) \\ &= \left(\sum_\omega X(\omega) P(\omega)\right) + \left(\sum_\omega Y(\omega) P(\omega)\right) \\ &= \mathbf{E}[X] + \mathbf{E}[Y].\end{aligned}$$

Raději zdůrazníme, že jsme nikde nepotřebovali předpokládat žádný druh nezávislosti: linearita střední hodnoty funguje za všech okolností.

Nyní si ukažme dva příklady využití linearity. Házíme dvěma kostkami, zapisujeme dvojciferný výsledek D a ptáme se, jaká je jeho střední hodnota $\mathbf{E}[D]$. Mohli bychom zajisté rozebrat všech 36 možností, ale existuje jednodušší cesta. Uvážíme náhodné veličiny pro číslo na první kostce X a to na druhé Y. Jistě je $D = 10X + Y$, takže podle linearity $\mathbf{E}[D] = 10\mathbf{E}[X] + \mathbf{E}[Y]$. Ovšem X a Y jsou úplně obyčejné hodnoty na kostkách, takže jejich střední hodnoty vyjdou $(1 + \ldots + 6)/6 = 7/2$. Proto $\mathbf{E}[D] = (10 + 1) \cdot 7/2 = 77/2$.

V druhém příkladu hodíme N-krát kostkou a budeme se ptát, kolik padlo šestek. Označme tuto náhodnou proměnnou S. Elementární jevy jsou posloupnosti hodů, takže jich je 6^N. Ovšem $\mathbf{E}[S]$ spočítáme snadno trikem takřka kouzelnickým. Rozložíme S na součet proměnných $S_1 + \ldots + S_N$, kde S_i říká, kolik padlo šestek v i-tém hodu. To je trochu přihlouplá otázka, protože padla buď jedna, anebo žádná. Ale každopádně se snadno spočítá, že střední hodnota $\mathbf{E}[S_i]$ je $0 \cdot P[S_i = 0] + 1 \cdot P[S_i = 1] = P[S_i = 1]$, což je pravděpodobnost, že v i-tém hodu padla šestka, tedy $1/6$. Proto $\mathbf{E}[S] = \mathbf{E}[S_1 + \ldots + S_N] = \mathbf{E}[S_1] + \ldots + \mathbf{E}[S_N] = N \cdot 1/6 = N/6$. Žádné překvapení se nekoná.

Mimochodem, tato úvaha je speciálním případem obecné techniky: Libovolnému jevu J můžeme přiřadit jeho *indikátor,* což je náhodná proměnná I_J, která nabývá hodnoty 1, pokud jev nastane, a jinak je nulová. Vychází pak $\mathbf{E}[I_J] = P[I_J = 1] = P(J)$.

Lemma o džbánu

Představte si, že opakovaně házíte kostkou, dokud nepadne šestka. Kolikrát v průměru hodíte? Obecněji: opakujeme nějaký pokus, který se pokaždé povede s nějakou pravděpodobností p. Pokud vám to připomíná přísloví o chození se džbánem pro vodu, jste na správné stopě. Suchým vědeckým jazykem jde o tzv. střední hodnotu geometrického rozdělení, ale nám přijde džbán s vodou výstižnější.

Jak to zapadá do naší teorie? Posloupnosti hodů jsou elementární jevy, počet hodů je náhodna proměnná a my se ptáme na její střední hodnotu. Nenechte se prosím znepokojit tím, že elementárních jevů je nekonečně mnoho, s čímž naše teorie nepočítala. Pohybujeme se sice na tenkém ledu, ale slibujeme, že se nepropadneme, protože toto nekonečno je „dostatečně krotké“.

Lemma říká následující: Mějme džbán. Kdykoliv s ním jdeme pro vodu, tak se jeho ucho utrhne s pravděpodobností p, nezávisle (ve smyslu popsaném výše) na ostatních pokusech. Pak ve střední hodnotě půjdeme se džbánem pro vodu $(1/p)$-krát, než se ucho utrhne.

Úplně základní proxy je `QIdentityProxyModel`, který jenom propouští data oběma směry. Pro demonstraci uděláme úplně neužitečnou změnu metody `data`:

```
class DisplayProxy(QIdentityProxyModel):
    def data(self, index, role):
        model = self.sourceModel()
        original = model.data(index, role)
        if role == Qt.DisplayRole:
            return "proxy! " + original
        else:
            return original
```

V konstruktoru třídy Crossing pak zapojíme proxy do procesu takto:

```
self.model = CrossingModel()
self.displayProxy = DisplayProxy()
self.displayProxy.setSourceModel(self.model)
# ...
self.listView.setModel(self.displayProxy)
```

Odchytávat signály (které vydává model) je podstatně těžší, to si necháme na jindy. Vydávat je ale můžeme; speciálně se nám bude hodit signál `dataChanged`, který se vydává pokaždé, když se nějaká hodnota změní. Ten můžeme vydat i ručně.

```
# ... došlo ke změně dat

# První změněný index
f = model.createIndex(prvni, 0)

# Poslední změněný index
t = model.createIndex(posledni, 0)

# Odeslat zprávu
model.dataChanged.emit(f, t)
```

Argumenty signálu `dataChanged` je rozsah (od–do včetně) indexů, na kterých došlo ke změně. A protože model nemusí být jenom seznam, ale ledacos úplně obecného, tak se předávají objekty typu `QModelIndex`. View si tento signál zaregistruje a obvykle na něj zareaguje novým načtením příslušných dat, je-li to potřeba.

Úkol 3 [4b]: Napište model proxy, která bude pravidelně aktualizovat data (volat `dataChanged`) každých 500 milisekund.

Kromě obyčejného proxy modelu nám Qt dává k dispozici `QSortFilterProxyModel`. Tento proxy model, jak název napovídá, dokáže data třídit a filtrovat. Jak zapojit model proxy mezi model a view, to už víme, jen se hodí doplnit, že proxy samozřejmě můžeme řetězit za sebe.

Kdybychom místo `QListView` použili `QTableView`, pak by nebylo třeba učinit nic dalšího, než zavolat na tento *view* metodu `setSortingEnabled`, která umožní uživateli volit směr řazení. My však místo toho zavoláme na *model* metodu `sort`:

```
model.sort(0, Qt.AscendingOrder)
```

Toto volání bude třídit podle sloupce 0 (jiný totiž nemáme) ve vzestupném pořadí. Jenže standardně se třídí podle `DisplayRole`; to se dá změnit zavoláním `setSortRole`. Roli si můžete nadefinovat vlastní; nejmenší bezpečná je `Qt.UserRole`. Na začátku programu tedy napíšeme třeba `mySortRole = Qt.UserRole`, při inicializaci modelu zavoláme `model.setSortRole(mySortRole)` a v metodě `data` našeho zdrojového modelu vrátíme pro tuto roli data, podle kterých se bude třídit.

Stejným způsobem se dá i filtrovat data, například zobrazit jen auta nebo chodce; tím se více zabývat nebudeme, zájemce odkazuji na webovou dokumentaci.

Úkol 4 [3b]: Použijte `QSortFilterProxyModel` k zobrazování seznamu cestovatelů setříděného podle toho, jak daleko od začátku své trasy jsou. Případné chvilkové nepřesnosti ve třídění můžete ignorovat.

Tím je třetí díl seriálu u konce. Omlouváme se za pozdní vydání a zkusíme to příště stihnout rychleji. A co bude příště? Zkusíme napsat vlastní View a vykreslit nadchod, chodce a auta v jednoduché grafice. A pokud se podaří, tak postavíme přechod a ukážeme, jak se píše Controller, resp. Delegate.

Maria Matějka

Recepty z programátorské kuchařky: Pravděpodobnost

Náhodná čísla a pravděpodobnost hrají v informatice důležitou roli. Když neumíme pro nějakou úlohu najít algoritmus, který je rychlý i v nejhorším případě, můžeme se spokojit s algoritmem rychlým alespoň v průměru. Nebo naopak s algoritmem, který je sice vždy rychlý, ale někdy odpoví špatně. Často se přitom hodí, aby si algoritmus „házel korunou“, tedy generoval nějaká náhodná čísla. Takovým algoritmům se říká *pravděpodobnostní* nebo také *randomizované*. V této kuchařce nahlédneme do základů teorie pravděpodobnosti a vybudujeme jí dost na to, abychom uměli zkoumat jednoduché randomizované algoritmy.

Po dlouhá staletí lidé přemýšleli nad pravděpodobností bez toho, aby jí rozuměli matematicky. Pravděpodobnost často není intuitivní, a proto i slavní matematici, jako třeba Newton, někdy v úvahách o náhodě chybovali. Naštěstí v minulém století přišel ruský matematik Andrej Kolmogorov s matematickým vysvětlením pravděpodobnosti, které nám dovoluje použít matematiku k přemýšlení o náhodných jevech, a vyhnout se tak chybám.

Jelikož „opravdová“ teorie pravděpodobnosti závisí na poměrně pokročilé matematice (takzvané teorii míry), vybudujeme ji trochu jednodušším způsobem. Bude výrazně intuitivnější než ta kolmogorovská, ale zvládne popsat jen konečné objekty. Budeme tedy moci zkoumat kostky, karty, algoritmy používající náhodná čísla z nějakého daného rozsahu (třeba celá čísla od 1 do 100) a podobně. Nezvládneme naopak korektně uvažovat o náhodných reálných číslech, náhodných bodech v rovině a jiných nekonečných věcech.

Jevy a jejich pravděpodobnost

Pro začátek si pravděpodobnost ukážeme na příkladu: Máme obyčejnou hrací kostku. Pokud kostkou hodíme, můžeme si být jisti, že na ní padne číslo od 1 do 6. Číslům 1 až 6 budeme říkat *elementární jevy*. Obecně, kdykoliv provedeme nějaký *náhodný experiment* (to je třeba hod kostkou), vždy nastane právě jeden elementární jev – v našem příkladu vždy padne jedno z čísel 1 až 6. Každému elementárnímu jevu můžeme přiřadit jeho *pravděpodobnost*. To je nějaké reálné číslo mezi 0 nebo 1, přičemž pravděpodobnosti všech elementárních jevů se sečtou na 1. To odpovídá tomu, že pokaždé nastane právě jeden z elementárních jevů – kostka nezůstane stát na rohu, ani nepadne jednička a dvojka současně. Prozatím tedy řekněme, že pravděpodobnost je funkce, která každému elementárnímu jevu přiřadí číslo od 0 do 1 a že se všechny pravděpodobnosti sečtou na 1.

Co kdybychom chtěli něco říci o pravděpodobnosti toho, že nám padne liché číslo. Takový výsledek pokusu už není elementární jev, ale můžeme ho stále z elementárních jevů poskládat: lichá čísla popíšeme množinou elementárních jevů $L = \{1, 3, 5\}$. Obecně můžeme za (náhodný) *jev* prohlásit jakoukoliv množinu elementárních jevů. Pro kostku to může být třeba jev $A = \{5, 6\}$ (padlo aspoň 5), jev $\emptyset$ (ten nenastane nikdy), nebo $V = \{1, 2, 3, 4, 5, 6\}$ (takovýto jev nastane vždy).

Pravděpodobnost jevu pak můžeme definovat jako součet pravděpodobností těch elementárních jevů, ze kterých se daný jev skládá. Jelikož všechny elementární jevy na kostce mají pravděpodobnost 1/6, bude pravděpodobnost, že padne liché číslo (to je náš jev L), rovna $1/6+1/6+1/6 = 3/6 = 1/2$. Označíme-li pravděpodobnost jevu J jako $P(J)$, vyjde pro ostatní zmíněné jevy $P(A) = 2/6 = 1/3$, $P(\emptyset) = 0$, $P(V) = 1$.

Jevy jsou tedy množiny a můžeme o nich jako o množinách mluvit. Můžeme například říct, že jevy A a B jsou *disjunktní,* tedy že $A \cap B = \emptyset$. To odpovídá tomu, že tyto dva jevy nemohou nastat najednou. Například jevy „padlo sudé číslo“ a „padlo liché číslo“ jsou zjevně disjunktní, protože žádné číslo není současně sudé a liché. Podobně jev $A \cup B$ odpovídá tomu, že nastane jev A nebo jev B, a jev $A \cap B$ tomu, že nastane současně A a B.

Zkusme si nyní rozmyslet, že pro jakékoliv dva disjunktní jevy A a B platí $P(A \cup B) = P(A) + P(B)$. Tedy pravděpodobnost (disjunktního) sjednocení A a B je součet pravděpodobností A a B. Toto tvrzení platí proto, že jsme definovali pravděpodobnosti A a B jako součty pravděpodobností elementárních jevů v A a B. Když tedy sečteme pravděpodobnosti všech elementárních jevů, které jsou v A, nebo v B (ale ne v obou, protože A a B mají prázdný průnik), dostaneme to samé, jako když sečteme $P(A)$ a $P(B)$.

Princip inkluze a exkluze

Pro každé dvě množiny A a B platí $|A \cup B| = |A| + |B| - |A \cap B|$. To proto, že do $|A| + |B|$ jsme prvky v průniku započítali dvakrát, a musíme tedy odečíst jejich počet, abychom dostali správný počet prvků ve sjednocení. Tomuto tvrzení (respektive jeho zobecnění pro libovolný počet množin) se někdy říká *princip inkluze a exkluze,* česky by asi dalo říci princip zahrnutí a vyloučení.

Podobně pro pravděpodobnosti se dá náhlednout rovnost $P(A \cup B) = P(A) + P(B) - P(A \cap B)$. Vzpomeňme si, že pravděpodobnost jevu je součtem pravděpodobností elementárních jevů, ze kterých se skládá. Pravděpodobnosti elementárních jevů v průniku jsme tedy v $P(A) + P(B)$ započetli dvakrát a musíme je odečíst, abychom dostali pravděpodobnost sjednocení. Všimněme si, že pokud jsou množiny A a B disjunktní, dostaneme $P(A \cup B) = P(A) + P(B)$, jak jsme si rozmysleli před chvílí.

Princip inkluze a exkluze je užitečné pravidlo k počítání pravděpodobností, ale jeho hlavní síla spočívá v následujícím. Uvědomme si, že pravděpodobnosti jsou nezáporná čísla. Platí tedy, že $P(A \cup B) = P(A) + P(B) - P(A \cap B) \leq P(A) + P(B)$, protože $P(A \cap B)$ je nezáporné číslo. Jinými slovy pravděpodobnost toho, že nastane A nebo B je nejvýše součet pravděpodobnosti, že nastane A, a pravděpodobnosti, že nastane B. Tomuto odhadu se říká *odhad sjednocení* (anglicky *union bound*) a bývá obzvláště přesný, pokud jevy, na které ho použijeme, mají malé pravděpodobnosti. (Raději zmiňujeme, že matematici slovem *odhad* míní nerovnost, nikoliv nějaké „odhadnutí od oka“.)

Co kdybychom teď měli sjednocení tří jevů? Pak si to sjednocení správně uzávorkujeme! $A \cup B \cup C = (A \cup B) \cup C$. Poté můžeme použít odhad sjednocení na dvojice jevů.

$$P(A \cup B \cup C) = P((A \cup B) \cup C) \leq P(A \cup B) + P(C) \leq P(A) + P(B) + P(C).$$

Můžeme pokračovat indukcí a dokázat, že pravděpodobnost sjednocení libovolného počtu jevů je nejvýše součet jednotlivých pravděpodobností.

Podmíněná pravděpodobnost

Náš kamarád hodil kostkou a řekl nám, že padlo číslo menší než 4 (padlo tedy 1, 2 nebo 3). Jaká je pravděpodobnost, že padlo liché číslo? V tomto případě není těžké si rozmyslet, že správná odpověď je 2/3. Ale teorie pravděpodobnosti tak, jak jsme si ji zatím vymysleli, nám na podobné úvahy nestačí.

Definujeme tedy *podmíněnou pravděpodobnost*. Řekneme, že $P(A \mid B)$ je pravděpodobnost, že nastal jev A, pokud už víme, že nastal jev B. Čteme to obvykle „pravděpodobnost A za podmínky B“.

Čemu by se ale měla $P(A \mid B)$ rovnat? Předpokládejme, že uděláme hodně náhodných experimentů. $P(B)$ říká, v jakém zlomku z nich nastal jev B. Z nich si vybereme ty, v nichž nastal i jev A. Ty tvoří zlomek $P(A \cap B)$ ze všech experimentů, takže z těch, kdy nastalo B, je to zlomek $P(A \cap B)/P(B)$. Definujeme tedy:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}.$$

Ve zmíněném příkladu dostaneme

$$P(\{1,3,5\} \mid \{1,2,3\}) = P(\{1,3\}) \;/\; P(\{1,2,3\}),$$

což se opravdu rovná 2/3, protože všechny elementární jevy mají v našem případě stejnou pravděpodobnost.

Můžeme si to také představit tak, že z množiny všech elementárních jevů vyřadíme ty, o nichž víme, že nenastaly. Zbudou nám tedy ty elementární jevy, které leží v B. Počítáme-li pak pravděpodobnost jevu A, musíme vyřazené jevy smazat, tedy uvážit průnik $A \cap B$. To ovšem nestačí: vyřazením jsme porušili pravidlo, že pravděpodobnosti všech elementárních jevů se sečte na jedničku: nyní se sečtou na $P(B)$. Proto ještě „změníme měřítko“ vydělením všech pravděpodobností číslem $P(B)$.

Nezávislé jevy

Nyní zkusíme hodit dvěma kostkami po sobě a výsledek přečíst jako dvojciferné desítkové číslo. Elementárních jevů je tedy celkem 36 jsou to čísla od 11 do 66. Uvažme tyto jevy:

$A =$ „první číslice je 1“,
$B =$ „druhá číslice je 1“,
$C =$ „číslo je menší než 30“.

Jejich pravděpodobnosti snadno spočítáme jako $P(A) = P(B) = 6/36 = 1/6$ a $P(C) = 12/36 = 1/3$.

Představme si nyní, že víme, že nastal jev A, tedy padlo jedno z čísel 11 až 16. Co jsme schopni říci o tom, zda nastal jev B? Jeho pravděpodobnost zůstává stále stejná: 1/6. Jev A nám tedy o jevu B nedává žádnou informaci. Tehdy říkáme, že jevy A a B jsou *nezávislé*.

Kdybychom naopak věděli, že nastal jev C, pravděpodobnost jevu A by se změnila na 1/2, protože z čísel 11 až 26 celá polovina začíná jedničkou. Tyto jevy jsou tedy *závislé*.

Nezávislost jevů A a B můžeme popsat požadavkem

$$P(A \mid B) = P(A).$$

Dosadíme-li definici podmíněné pravděpodobnosti, dostaneme:

$$P(A \cap B)/P(B) = P(A),$$

čili

$$P(A \cap B) = P(A) \cdot P(B).$$

Za definici nezávislosti se většinou považuje tato poslední rovnost, protože funguje i pro $P(B) = 0$, kdy by podmíněná pravděpodobnost nebyla vůbec definovaná. Také je z ní hned vidět, že nezávislost je symetrická vlastnost. Dodejme ještě, že samozřejmě platí i $P(B \mid A) = P(B)$.

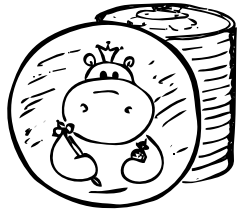
Definovat nezávislost pro více jevů je složitější, obecně nestačí říci, že pravděpodobnost toho, že nastanou současně, je součin jednotlivých pravděpodobností. Je potřeba, aby to platilo pro kteroukoliv podmnožinu jevů.

Náhodné proměnné a střední hodnota

Představme si nyní jednoduchý pravděpodobnostní algoritmus. Funkce $random(x)$ bude vracet náhodné celé číslo z rozsahu 0 až $x - 1$.

1. $x \leftarrow random(2)$
2. Pokud $x = 1$:
3. $\quad y \leftarrow random(2)$

Algoritmus si tedy hodí korunou (vygeneruje náhodný bit) a v případě, že padla jednička, hodí ještě jednou. Jak ho pomocí teorie pravděpodobnosti popíšeme? Možné průběhy výpočtu můžeme popsat jako elementární jevy. Jelikož pro pevný vstup (tady dokonce žádný nemáme) je průběh výpočtu jednoznačně určený hodnotami náhodných čísel, můžeme říci, elementární jevy jsou možné posloupnosti náhodných čísel.



Pro náš jednoduchý program to jsou posloupnosti 0, 10 a 11. Přitom 0 má pravděpodobnost 1/2, zbylé dvě 1/4.

Co kdybychom chtěli říci, jak dlouho náš program běží? Pro jednoduchost to budeme počítat v příkazech. V nejhorším případě se provedou 3, ale kdybychom to chtěli říci přesněji, mohli bychom říci, že pro posloupnost 0 se provedou 2, zatímco pro 10 a 11 se provedou 3. Doba běhu randomizovaného algoritmu je hezkým příkladem takzvané náhodné proměnné, kterou teď zavedeme obecně.

Když provedeme náhodný experiment (třeba hodíme kostkou), nastane jeden z elementárních jevů. *Náhodná proměnná* (nebo také *náhodná veličina*) je jedno číslo určené tím, který elementární jev nastal. Příkladem by mohla být náhodná proměnná L, která je 0, pokud padlo na kostce sudé číslo, a 1, pokud liché. Dalším příkladem náhodné proměnné může být třeba „číslo, které padlo na kostce, umocněné na druhou“. Tu označme třeba D.

Všimněte si, že podmínka, ve které figurují náhodné proměnné, vždy určuje nějaký jev: množinu elementárních jevů, pro které podmínka platí. Například $L = 1$ platí pro $\{1, 3, 5\}$, $D < 10$ pro $\{1, 2, 3\}$ a $D < 10 \wedge L = 0$ pro $\{2\}$. Proto se můžeme ptát na pravděpodobnost toho, že podmínka platí, což se obvykle značí pomocí hranatých závorek: $P[D < 10] = P(\{1, 2, 3\}) = 3/6 = 1/2$.