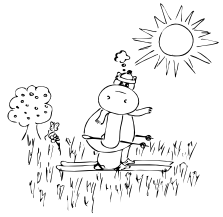
Korespondenční Seminář z Programování

31. ročník **KSP** **Březen 2019**

Milí řešitelé, milé řešitelky!



Jaro už klepe na dveře a příroda i nejeden organizátor se pomalu probouzejí ze zimního spánku. I během zkouškového jsme na vás nezapomněli a přichystali jsme si pro vás další várku úloh. Těšit se můžete také na pokračování seriálu a kuchařku o těžkých problémech.

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 této sérii. V takovém případě se nám ozvěte mailem.

Termín série: pondělí **15. dubna v 8:00 ráno**

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

Odměna série: **Sladkou odměnu** si vysloužíš každý, kdo **z libovolných čtyř úloh získá alespoň polovinu možných bodů**.

Čtvrtá série třicátého prvního ročníku KSP

Jednoho krásného zimního dne, kdy se sněhové vločky k zemi jako drobounká pírka snášely, seděla jedna královna u okna a vyšívala. Jak tak v zamyšlení z okna ven koukala, bodla jehlou do prstu, až na okenní římsu tři krůpěje krve skanuly. Tu si královna pomyslela, že by měla ráda takové děťátko, které by bylo rudé jako ta krev, bílé jako ten sníh a černé jako okenní rám z ebenového dřeva. Neuplynul ani rok a královně se narodila krásná dceruška. Sotva však královna svou dcerku poprvé políbila, navždy oči zavřela.

Netrvalo dlouho a král si našel novou královnu, jež ho svou krásou a medovým hlasem učarovala. A tak Sněhurka (jak malému děťátku pro jeho sněhobílou pleť říkat začali) macechu získala. Nová královna byla převelice pyšná a na své krásе si velmi zakládala. Ráno, hned jak vstala, se kouzelného zrcadla ptávala: „Zrcadlo, zrcadlo, pověz, kdo je na světě i v zemi zdejší ze všech k-tý nejkrásnější?“

31-4-1 Kouzelné zrcadlo — 10 bodů

Macecha se chce ujistit, že se na žebříčku nejkrásnějších lidí stále nachází na k-tém místě. Kouzelné zrcadlo má přístup k údajům o lidech z celého světa. Mezi údaji se mimo jiné nachází dvě posloupnosti obsahující osoby srovnané podle krásy, od nejkrásnější po nejméně krásnou (jedna posloupnost pro ženy, druhá pro muže). Krása osoby je zde určena nějakou hodnotou, každá osoba má tuto hodnotu unikátní a navíc platí, že při srovnání dvou osob má krásnější osoba hodnotu nižší. Zrcadlo chce najít k-tou nejkrásnější osobu, tj. k-tou nejmenší hodnotu sjednocení obou posloupností. Jak to má udělat, aby mohlo královně odpovědět co nejrychleji?

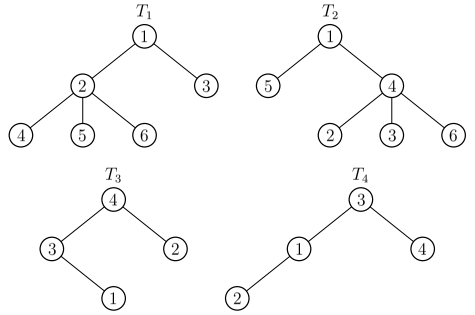
Zrcadlo ji ujistilo, že to ona je stále v této zemi k-tá nejkrásnější. Macecha pak byla po celý zbytek dne v dobré náladě, protože věděla, že její zrcadlo lhát nemůže. Tak to šlo den po dni, měsíc po měsíci.

Jak šel čas, malá Sněhurka postupně rostla do krásy. Jednoho rána, v den, kdy bylo Sněhurce zrovna patnáct let, se macecha jako obvykle postavila před zrcadlo a zeptala se: „Zrcadlo, zrcadlo, pověz, kdo je na světě i v zemi zdejší ze všech k-tý nejkrásnější?“ Kouzelné zrcadlo odpovědělo: „Jsi krásná, paní má, avšak v zemi zdejší Sněhurka je k-tá nejkrásnější.“ To macechu nesmírně rozzlobilo. Dlouho se nerozmýšlela a okamžitě si k sobě nechala povolat myslivce. Přikázala mu, aby Sněhurku zavedl do lesa, zabil ji a na důkaz splnění příkazu přinesl královně její srdce.

Ještě téhož dne myslivec nabídl Sněhurce, jestli si s ním nechce vyjít do lesa, že pro ni má zajímavý úkol. Nedávno totiž hluboko v lese objevil jednu mýtinku, na které ještě nikdy v životě nebyl. Proto potřebuje pomoct s klasifikací stromů, které v jejím okolí rostou. Nemusel Sněhurce říkat dvakrát, šla do lesa ráda.

31-4-2 Stromy na mýtince — 11 bodů

Myslivec potřebuje pomoct s klasifikací stromů na nově objevené mýtince. Stromů se ale na mýtince nachází opravdu hodně, takže není v lidských silách je klasifikovat jednotlivě. Myslivec si však všimnul, že některé stromy mají v určitém slova smyslu „stejnou strukturu“. Jedna větev roste tady, ale vedlejší strom má úplně stejnou větev, jen vyrůstá z jiného místa na stromě… Kdyby dokázal stromy rozdělit na skupinky tak, aby v jedné skupince byly stromy se „stejnou strukturou“, pracovalo by se mu s nimi o dost lépe a jejich klasifikaci by měl hotovou mnohem rychleji.



Trochu formálněji: na vstupu je několik uspořádaných zakořeněných stromů (tj. zakořeněných stromů, kde každý vrchol má určené pořadí synů „zleva doprava“). Řekneme, že dva stromy jsou *izomorfní,* pokud po vhodném přejmenování vrcholů dostaneme dva stejné stromy (tj. se stejným kořenem, strukturou a pořadím synů ve vrcholech). Vaším úkolem je rozdělit stromy na několik hromádek, aby dva stromy byly na stejné hromádce, právě když jsou izomorfní.

Na příkladu na předchozí stránce jsou hromádky izomorfismu tři: $\{T_1\}$, $\{T_2\}$ a $\{T_3, T_4\}$. Vhodný izomorfismus z T_3 na T_4 přejmenovává vrcholy následovně: $4 \to 3, 3 \to 1, 2 \to 4, 1 \to 2$. Všimněte si že T_1 a T_2 izomorfní nejsou, neboť žádné přejmenování vrcholů nezachovává správné pořadí synů vrcholu 1.

Zatímco se Sněhurka snažila úkol vyřešit, myslivec se pomalu odplížil a spěchal zpátky na zámek. Nedokázal totiž ubohé Sněhurce ublížit. Vzpomněl si však, oč ho královna žádala. Proto cestou zabil srnku, vyňal z ní srdce, a to pak královně přinesl na „důkaz“ splnění úkolu.

Když Sněhurka nakonec úkol vyřešila, s hrůzou zjistila, že myslivec mezitím zmizel. I vydala se ho hledat. Klopýtala hlubokým lesem cestou necestou, dál a dál od rodného zámku. Ale myslivec nikde… Už se pomalu začínalo smrákat, když tu se najednou les rozestoupil a Sněhurka se ocitla na paloučku, kde stála hezká chaloupka. „Přespím tu,“ pomyslela si, „a ráno mě myslivec určitě najde.“

Když vešla dovnitř, úžasem oněměla. V malé světničce stál dlouhý stoleček a na něm leželo N talířků. U každého talířku lžička, nůž a vidlička a před talířkem sklenička. U stěny pak v řadě vedle sebe stálo N čistě povlečených postýlek. Protože už měla Sněhurka po celém dnu pořádný hlad, neodolala, z každého talířku si vzala jedno sousto a z každého pohárku se trochu napila. A že byla hodně unavená, zalezla do jedné z postýlek.

Než však stihla Sněhurka oči zamhouřit, dorazil do chaloupky první trpaslík. Byl ale natolik vyčerpaný z celodenní práce v dole, že pouze Sněhurku, která právě ležela v jeho postýlce, požádal, aby se přesunula někam jinam. „Do které z postýlek si mohu lehnout, sličný pane trpaslíku?“ zeptala se Sněhurka zdvořile. „Támhle, vedle dveří, visí na zdi rozpis nočních směn všech trpaslíků. Tam můžeš zjistit, kdy se kdo vrátí domů.“

31-4-3 Nejvíc spánku — 8 bodů

Sněhurka je po náročném bloudění lesem velice unavená a chtěla by se co nejdéle prospat. V chaloupce se nachází N postýlek, každá postýlka patří právě jednomu z N trpaslíků. Sněhurka má k dispozici rozpis nočních směn všech trpaslíků, což je seznam zprava otevřených intervalů, kdy jsou jednotliví trpaslíci v dole. Proto ví, od kdy do kdy je která postýlka volná. Chtěla by si vybrat, ve kterých postýlkách bude spát, aby naspala co nejvíce. Sněhurka je tak unavená, že když jednou do nějaké postýlky zaleze, vstane, až když ji z ní její majitel vyhodí. Zároveň ale nechce lézt do postýlky, která je studená, a proto zaleze pouze do té, z které její majitel právě odešel a která je tedy stále pěkně vyhřátá.

Jinými slovy, máme N intervalů $\langle a, b)$. Chceme vybrat podmnožinu s co největší celkovou délkou, ve které se žádné dva intervaly neprotínají.

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 kladné celé číslo N udávající počet intervalů. Na každém z následujících N řádků jsou vždy dvě celá čísla a_i, b_i ($0 \leq a_i < b_i \leq 10^9$) udávající levý a pravý okraj intervalu $\langle a_i, b_i)$, kdy je i-tý trpaslík v dole.

Formát výstupu: Na jediném řádku vypište celé číslo: největší možnou celkovou délku Sněhurčina spánku. Délka intervalu $\langle a, b)$ je $b - a$.

Ukázkový vstup:	*Ukázkový výstup:*

`6`
`0 5`
`0 1`
`4 10`
`0 2`
`2 3`
`6 8`
	`9`

Optimální je vybrat intervaly $\langle 0, 2)$, $\langle 2, 3)$ a $\langle 4, 10)$.

Trpaslíci si Sněhurku brzy oblíbili. Pomáhala jim s úklidem, vařila a celkově se o ně pěkně starala. Každé ráno, když trpaslíci odcházeli do práce, Sněhurku varovali: „Dávej na sebe pozor. Hlavně nikomu cizímu neotvírej a dovnitř ho nepouštěj!“

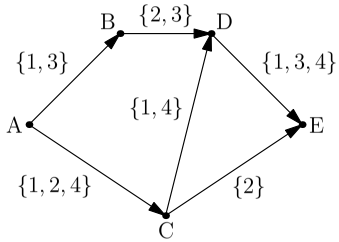
Takhle by mohli všichni v chaloupce žít šťastně navěky, kdyby se jednoho dne zlá královna špatně nevyspala. Protože si chtěla spravit náladu, postavila se před zrcadlo a zeptala se: „Zrcadlo, zrcadlo, pověz, kdo je na světě i v zemi zdejší ze všech k-tý nejkrásnější?“ „Jsi krásná, paní má, avšak v zemi zdejší Sněhurka je k-tá nejkrásnější,“ pravilo kouzelné zrcadlo. Královna se začala vzteky dusit. „Kde je?“ vypravila ze sebe. A zrcadlo odpovědělo: „Za devíti lesy žije a den ode dne krásnější je.“ Rozzuřená královna se rozhodla, že se Sněhurky jednou pro vždy zbaví. Zavřela se do své komnaty a tam čarovala a zaříkávala tak dlouho, dokud jedovaté jablko nevyrobila. Pak se přestrojila za prostou selku, vzala si košík s ovocem a vydala se k chaloupce.

31-4-4 Otrávené ovoce — 12 bodů

Zlá královna chce dopravit otrávené ovoce ze zámku do chaloupky skrz hluboký les. Jako v každém lese se i zde nacházejí jak mýtinky, tak i cestičky, které vždy nějakou dvojici mýtinek spojují. Protože to ale není les obyčejný, každá cestička má daný směr, ve kterém po ní lze projít. Další speciální výsadou lesa je, že se v něm vyskytuje spousta prapodivných stvoření. Každé takovéto stvoření žije v blízkosti nějaké cestičky a pokud procházející kolemjdoucí vlastní ovoce, kterým se dané stvoření živí, kolemjdoucího o něj obere. Stvoření jsou ale tuze vybíravá, živí se pouze konkrétními druhy ovoce a ničím jiným. Sama královna se s žádným prapodivným stvořením tváří v tvář setkat nechce, a proto si nechá ovoce přes les přepravit svými služebníky. Pověřila svého rádce, aby zjistil, kolik na to potřebuje služebníků. Rádce přemýšlel a počítal, ale nedopočítal se a královně chce dokázat, že je tento úkol nad jeho síly.

Les si lze představit jako orientovaný neohodnocený graf, zámek představuje start a chaloupka cíl. Otrávené ovoce představuje množinu věcí, kterou chceme přepravit ze startu do cíle. Každá hrana je ohodnocena množinou ovoce, které nám zůstane, pokud přes ni projdeme. K dispozici máme několik služebníků, každý z nich dostane nějakou podmnožinu otráveného ovoce. Dokažte, že zjistit, kolik nejméně služebníků musí královna vyslat, aby zvládli dopravit všechno ovoce ze startu do cíle (případně říct, že to nejde), je $\mathcal{NP}$-úplný problém.

Na následujícím příkladu si pro množinu otráveného ovoce $M = \{1, 2, 3, 4\}$, zámek ve vrcholu A a chaloupku ve vrcholu E vystačíme se třemi služebníky: prvnímu dáme množinu $\{1, 4\}$ a pošleme ho cestou A–C–D–E, druhému dáme množinu $\{2\}$ a pošleme ho cestou A–C–E a třetí dostane $\{3\}$ a půjde po cestě A–B–D–E. Rozmyslete si, že méně služebníků ovoce přepravit nedokáže.



Jakmile se královna i s ovocem bezpečně dostala na druhou stranu lesa, počkala na okamžik, kdy budou všichni trpaslíci pryč, a pak zaklepala na dveře chaloupky. Sněhurka nic netušila a s úsměvem královně otevřela. V přestrojení svou macechu nepoznala, a tak se vesele zeptala: „Co si přejete, babičko?“ „Přinesla jsem ti jablíčko, děvenko, je dobré, sladké, jen ochutnej!“ švitořila úlisně královna. Sněhurka královně poděkovala a vzala si jablko do ruky. Vypadalo tak krásně! Bylo to to nejčervenější a nejšťavnatější jablko, jaké kdy Sněhurka viděla. Dlouho se nerozmýšlela a hned se do něj zakousla.

V tu chvíli královna zajásala, odhodila svůj převlek a sledovala, jak Sněhurka v mdlobách padá k zemi. Jed zafungoval! „Konečně jsem se jí zbavila,“ vyjukla radostí a uháněla zpátky do svého zámku.

Když se trpaslíci vrátili domů a uviděli Sněhurku ležet na zemi, velmi se zarmoutili. Neměli ji nechávat doma samotnou! Uložili Sněhurku do skleněné rakve a tři dny a tři noci plakali nad její smrtí. Čtvrtého rána však zaslechli zvonění kopyt, a jak se větve lesa rozestoupily, stál před nimi nádherný princ na bílém koni. „Kdo je ta krásná dívka?“ zeptal se, jakmile spatřil Sněhurku. „To je Sněhurka,“ odpověděli trpaslíci, „ale jdeš pozdě, princi.“ A znovu se rozplakali.

Princ sestoupil z koně a přiblížil se ke Sněhurce. „Je tak nádherná… Vypadá, jako by jen spala!“ Odkryl víko a přizvedl k sobě Sněhurčino tělo. V tu chvíli Sněhurce z krku vyskočil kus otráveného jablka a Sněhurka otevřela oči. „Ona žije!“ „Stal se zázrak!“ volali trpaslíci jeden přes druhého.

Když Sněhurka na prince pohlédla, okamžitě se do něj zamilovala. Nasedli společně na princova koně a vydali se na zámek za Sněhurčiným otcem. Jak byl král rád, že se se svou ztracenou dcerou znovu shledal! Sněhurka pak svému otci pověděla, co se jí přihodilo a jak se ji zlá královna pokusila zabít. Když si král vyslechl celý příběh, nechal královnu z království navždy vyhnat. Statečnému princi za záchranu Sněhurky přislíbil její ruku a k-tinu království k tomu.

31-4-5 Dělení království — 12 bodů

Princ si chce spočítat, kolik že je to ta k-tina celého království. Ví, že celé království má hodnotu h, což je nějaké celé číslo, a chce spočítat desítkový zápis zlomku h/k s vyznačenou periodou. Ovšem princ si toho moc nepamatuje, k dispozici má jen svou hlavu s konstantním počtem paměťových buněk a kus pergamenu, na který se mu ale tak tak vejde samotný výsledek, takže už na něj nemůže napsat nic dalšího.

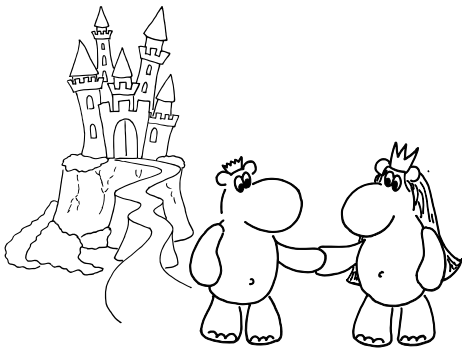
Čísla h i k se vejdou do jedné paměťové buňky a běžné operace s nimi umíme provádět v konstantním čase. To znamená, že je například umíme v konstantním čase násobit, sčítat a **celočíselně** dělit, ale neumíme už například v konstantním čase napočítat od 1 do k nebo čísla dělit desetinně s neomezenou přesností. Pokud vám tento model přijde zvláštní, dosaďte si za něj například klasická 64bitová čísla ve vašem oblíbeném programovacím jazyce.

Známe řešení, které potřebuje konstantní pomocnou paměť a pracuje v čase $\mathcal{O}(N)$, kde N je počet cifer výsledku. Najdete nějaké stejně dobré? Poslat samozřejmě můžete i pomalejší řešení, důležité však je, aby také pracovalo s konstantní pomocnou pamětí.

Příklad: pro $h = 5251$, $k = 700$ princ na pergamen zapíše číslo $7{,}50\overline{142857}$.

Nedlouho poté se konala veliká svatba a jestli neumřeli, žijí princ se Sněhurkou šťastně dodnes.

Zuzka Urbanová & Klárka Tauchmanová



31-4-6 Kde je hroznýš? Kuk! — 15 bodů

Prosíme řešitele, aby tuto úlohu odevzdávali jako prostý text; PDF se těžko spouští a testuje.

V minulém dílu serálu o Qt jsme přepsali simulátor naddchodu tak, aby měl oddělený datový model a ovládací rozhraní, neboli podle principu Model–View. Dnes nás čeká napsat si vlastní View. A protože chceme auta a chodce zobrazovat graficky, učiníme krok stranou a prvně si prostě napíšeme vlastní widget.

Kterak si pořiditi widget

Nakreslíme si něco úplně obyčejného, třeba sluníčko. To zařídíme implementováním metody `paintEvent`. V této me-

[1] `https://doc.qt.io/qtforpython/PySide2/QtGui/QPainter.html#PySide2.QtGui.QPainter`

todě si pořídíme `QPainter`, což je obsáhlý objekt s obsáhlou dokumentací,[1] jehož metodami kreslíme obsah widgetu.

Jak to vypadá, si ukážeme na příkladu:

```
from PySide2.QtWidgets import \
        QApplication, QWidget
from PySide2.QtGui import QPainter, QColor, \
        QPen, QBrush
from PySide2.QtCore import Qt

import sys
app = QApplication(sys.argv)

class Sun(QWidget):
    beams = 42
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.show()

    def paintEvent(self, event):
        # Rozměry widgetu, do kterého kreslíme
        w = self.width()
        h = self.height()

        # Poloměr sluníčkového kolečka
        r = min(w, h) / 4

        # Kreslič sám
        painter = QPainter(self)
        painter.setRenderHint(
                QPainter.Antialiasing)

        # Modré pozadí
        painter.setBrush(QColor(128, 192, 255))
        painter.setPen(QPen())
        painter.drawRect(0, 0, w, h)

        # Žluté svítíčko tlustou žlutou čárou
        painter.setBrush(QBrush())
        painter.setPen(QPen(QColor(220, 180, 0),
                            3))

        # Používáme transformaci souřadné
        # soustavy, kdy si nulu dáme doprostřed
        # widgetu
        painter.translate(w/2, h/2)
        painter.drawEllipse(-r, -r, r*2, r*2)

        # Paprsky
        for i in range(self.beams):
            # Nakreslíme paprsek
            painter.drawLine(0, r, 0, r*2)

            # A pootočíme soustavu souřadnic
            painter.rotate(360/self.beams)

s = Sun()
app.exec_()
```

Nakreslili jsme nejprve modrý obdélník, přes něj žluté kolečko a žluté čáry. Kreslíme tak, že si nejprve vybereme pero (tím se kreslí čára) a štětec (tím se maluje vnitřek oblasti). Přitom jsme si posouvali souřadnou soustavou, jak bylo zrovna potřeba. Tato posunutí, rotace apod. platí pouze pro tento jeden `QPainter` a při příštím kreslení si pořídíme zase nový, čistý kreslič s nulou vlevo nahoře.

Pojďme si tedy vyzkoušet, jak widget ovládat, a to především myšem. Máme na výběr několik metod – můžeme implementovat:

- `mousePressEvent` – vyvolá se stiskem tlačítka,
- `mouseReleaseEvent` – vyvolá se puštením tlačítka,
- `mouseMoveEvent` – na pohyb myše,
- `mouseDoubleClickEvent` – dvojklik.

My si to ukážeme na jednoduchém příkladu, ve kterém levé myšítko zvýší počet paprsků a pravé myšítko počet paprsků zase sníží.

```
def mousePressEvent(self, event):
    # Obsluha myšítkových událostí
    btn = event.button()
    print(btn)
    if btn == Qt.MouseButton.LeftButton:
        self.beams += 1

    if btn == Qt.MouseButton.RightButton \
            and self.beams > 0:
        self.beams -= 1

    # Vyžádáme si překreslení
    self.update()
```

Metodou `button` události `QMouseEvent` zjistíme, které tlačítko ji vyvolalo (u pohybu myše žádné), metodou `buttons` zjistíme, která jsou zrovna stisknutá.

Pokud je potřeba widget překreslit, zavoláme na něj nakonec `update`, což vyvolá právě událost překreslení, kdy se celý widget kreslí od znova.

Není vždy třeba překreslovat celý widget. Všimněte si nepoužitého argumentu `event` události `paintEvent`. To je objekt typu `QPaintEvent`, ve kterém je uložená informace o tom, kterou oblast je třeba překreslit. Pokud je kreslení náročnější na čas, může se to hodit využít. Pokud však překreslíte víc, nic se nestane.

Úkol 1 [3b]: Upravte obsluhu myšítek tak, aby se po jedné vteřině držení myšítka začal zvyšovat či snižovat počet paprsků o jeden každých 250 milisekund, dokud uživatel myšítko nepustí.

Úkol 2 [2b]: Připište metodu `mouseMoveEvent` (a upravte vykreslování) tak, aby se sluníčko schovalo (nevykreslilo), pokud najedete kurzorem nad jeho střed. Pro tento úkol se budou hodit metody `x` a `y`, případně `pos`, kterými zjistíme relativní pozici myšítka ve widgetu. Také je třeba zapnout si u widgetu `setMouseTracking(True)`, jinak se event vyvolá jenom při stisknutém myšítku.

Pokud v rámci obsluhy událostí potřebujete nějak měřit vzdálenost ujetou kurzorem, třeba chcete napsat drag-and-drop ovládání, může se stát, že widget mezitím změnil svoji polohu nebo velikost. Proto si můžete říct i o pozici kurzoru v kontextu celé obrazovky metodou `globalPos`.

Píšeme vlastní View

Když už víme, jak si můžeme kreslit vlastní widgety, pořídíme si i náš vlastní View. Držte si klobouky, jedeme s kopce. Zdědíme `QAbstractItemView` a máme hned několik problémů.

Především se jedná o scrollovatelnou oblast (což je milé, ale dá nám trochu víc práce to zvládnout). Takže musíme kvůli zobrazování implementovat více metod, než kdybychom zdědili widget. Nicméně ukáže se (v poslední sérii), že do View se dá připojit Delegate na ovládání jednotlivých prvků. Proto budeme trpěliví a všechny metody poctivě implementujeme. Tedy ne všechny metody, jenom ty, které nás nějak ovlivňují. V poslední sérii to budeme muset udělat pořádně.

Název problému: Dva loupežníci

Vstup: Seznam nezáporných celých čísel.

Problém: Existuje rozdělení seznamu na dvě hromádky tak, že každé číslo bude v právě jedné hromádce a v každé hromádce bude stejný součet čísel?

Název problému: 3D párování

Vstup: Seznam mužů, žen a zvířátek, následovaný seznamem kompatibilních trojic tvaru {muž, žena, zvířátko}. Tyto trojice říkají, která trojice muž, žena a zvířátko by se dohromady snesla.

Problém: Můžeme všechny muže, ženy a zvířátka z prvního seznamu rozdělit do trojic tak, že každá trojice je kompatibilní a každá bytost je právě v jedné trojici?

Název problému: Klika

Vstup: Neorientovaný graf, číslo k.

Problém: Existuje v grafu úplný podgraf o velikosti k, tedy k vrcholů takových, že mezi každými dvěma z nich vede hrana?

Název problému: Nezávislá množina

Vstup: Neorientovaný graf, číslo k.

Problém: Existuje v grafu nezávislá množina vrcholů o velikosti k, tedy k vrcholů takových, že žádné dva z nich nejsou spojeny hranou?

Název problému: Trojbarevnost grafu

Vstup: Neorientovaný graf.

Problém: Lze vrcholy tohoto grafu obarvit třemi barvami tak, že spolu žádné dva vrcholy stejné barvy nesousedí?

Název problému: Rozparcelování roviny

Vstup: Seznam bodů v rovině, kde každý má navíc přiřazenu jednu z b barev, číslo k.

Problém: Umíme rozdělit rovinu pomocí k přímek tak, že v každé oblasti jsou jen body té samé barvy?

Kuchařku sepsali

Martin Böhm & Jirka Setnička

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`.

Otázka zní: Jaký je nejcennější možný náklad, který přesto nepřesahuje váhový limit batohu?

Možná víte, že úloha jde řešit dynamickým programováním – vytvoříme si pole `podbatoh[]` od 1 do B, kde `podbatoh[`i`]` je maximální hodnota, kterou bychom si odnesli v batohu o nosnosti i. Postupně od první věci do poslední pak projdeme celé pole `podbatoh[]` „zprava doleva“ od B do 1 a zkusíme, jestli je výhodnější do batohu vložit novou věc a volné místo doplnit starými (optimální volné místo pro předchozí věci máme napočítané), nebo si nechat jen ty staré. Tuto hodnotu pak zapíšeme jako aktuální pro váhu i na místo `podbatoh[`i`]`.

Po N průchodech tohoto pole dostaneme řešení pro všechny věci dohromady na políčku `podbatoh[`B`]`. Celková složitost je $\mathcal{O}(NB)$, to je polynom, algoritmus je tedy polynomiální.

Světe div se, toto řešení je ve skutečnosti exponenciální. Kde jsme v řešení udělali chybu? Nikde – naše složitost závisela na B, ovšem když se podíváme do vstupních dat, tak pokud jsou zapsána v binárním (nebo ternárním a vyšším) tvaru, zápis čísla B je veliký $\mathcal{O}(\log_2 B)$. Naše složitost však závisí na $B = 2^{\log_2 B}$, je tedy exponenciální vzhledem k velikosti vstupu.

Problém batohu, respektive jeho rozhodovací verze, je dokonce $\mathcal{NP}$-úplný problém.

Algoritmům, které řeší nějakou úlohu a jsou polynomiální vůči *hodnotě* čísel na vstupu, ale exponenciální ve *velikosti zápisu* těchto čísel, říkáme *pseudopolynomiální algoritmy.* Některé další $\mathcal{NP}$-úplné problémy mají pseudopolynomiální řešení (jako například *Dva loupežníci* níže), ale dá se dokázat, že na jiné problémy pseudopolynomiální algoritmus neexistuje (pokud $\mathcal{P} \neq \mathcal{NP}$).

Mimochodem: pokud bychom na vstupu zapisovali čísla v unárním zápisu (tedy místo každého čísla by bylo třeba tolik hvězdiček, jakou hodnotu představuje), každý pseudopolynomiální problém by ležel v $\mathcal{P}$.

Poznámky na závěr

Otázku „Je třída $\mathcal{P}$ rovna $\mathcal{NP}$?“ se již snažilo rozlousknout mnoho matematiků a informatiků. Tato teorie přinesla spoustu zajímavých výsledků, například už se podařilo dokázat, že některými technikami tuto domněnku nelze nikdy dokázat, ani vyvrátit.

Kdyby platilo $\mathcal{P} = \mathcal{NP}$, pak by mnoho lidi zajásalo – spousta přirozených problémů, které nastávají i v reálném životě, by najednou byla řešitelná rychle. Navíc by krachlo dosavadní šifrování a bylo by možné najít rychle důkaz ke každému pravdivému tvrzení výrokové logiky.

Tato rovnost by se dala hypoteticky ukázat velice snadno – stačilo by najít jeden polynomiální algoritmus pro libovolný $\mathcal{NP}$-úplný problém! Většina informatiků studujících složitost se však domnívá, že se třídy nerovnají.

To ale neznamená, že si to nemáte zkusit dokázat! Naopak, bojovat s $\mathcal{NP}$-úplnými problémy je užitečné i v reálném světě – mnohdy jde vymyslet třeba dobrá aproximace řešení.

Například nenajdeme hamiltonovskou kružnici v polynomiálním čase, ale nalezneme nějakou relativně dlouhou kružnici, která nám v praxi může stačit, pokud podle ní třeba chceme vést náročný cyklistický závod.

O aproximacích je toho v literatuře napsáno mnoho zajímavého, pokud byste si o nich chtěli přečíst více v češtině, zkuste třeba kapitolu o $\mathcal{NP}$-úplných problémech v knížce Průvodce labyrintem algortmů.[5]

Více o třídě $\mathcal{NP}$ i o dalších aspektech složitosti můžete najít na stejné adrese, nebo zkuste vynikající anglicky psanou knížku *Algorithms* od profesorů exotických jmen Dasgupta, Papadimitriou a Vazirani.

Jak už jsme zmínili, existují i problémy, které jsou mimo $\mathcal{P}$ i $\mathcal{NP}$, a dokonce existuje spousta různých dalších tříd problémů. Je jich celá zoologická zahrada – pěkný souhrn můžete najít na stránkách Univerzity ve Waterloo.[6]

Seznam $\mathcal{NP}$-úplných problémů

Sedíte-li nad zatím nevyřešenou úlohou, kterou stále nemůžete rozlousknout, je možné, že bude $\mathcal{NP}$-úplná. Abyste mohli mezi $\mathcal{NP}$-úplnými úlohami převádět, tak je dobré znát jich aspoň hrstku, podle toho, je-li problém grafový, rovnicový, a tak dále.

V následujícím seznamu najdete několik úloh, které jsou zaručeně $\mathcal{NP}$-úplné. Převody se nám sem už sice nevešly, ale většinu z nich (ne-li všechny) zvládnete vymyslet sami – zkuste to!

Název problému: Hamiltonovská kružnice

Vstup: Neorientovaný graf.

Problém: Existuje v zadaném grafu kružnice procházející všemi vrcholy právě jednou?

Název problému: Hamiltonovská cesta

Vstup: Neorientovaný graf, dva speciální vrcholy x a y

Problém: Existuje cesta z x do y (posloupnost vrcholů, ve které se žádné dva neopakují), která prochází každým vrcholem právě jednou?

Název problému: Splnitelnost

Vstup: Logická formule. Tu tvoří proměnné a logické spojky negace $\neg$, konjunkce $\wedge$ a disjunkce $\vee$. Například

$$(x \wedge (\neg y)) \vee z.$$

Problém: Můžeme proměnným přiřadit hodnoty 0 nebo 1 tak, že výsledná vyhodnocená formule má hodnotu 1?

Název problému: Součet podmnožiny

Vstup: Seznam nezáporných celých čísel, speciální číslo k.

Problém: Existuje podmnožina čísel, jejíž součet je přesně k?

Název problému: Batoh

Vstup: Seznam dvojic nezáporných čísel, kde dvojice označuje hodnotu a váhu předmětu. Přirozené číslo b – nosnost batohu, přirozené číslo k.

Problém: Umíme vložit do batohu předměty o hodnotě alespoň k, aniž bychom přešli přes limit váhy b?

---

[5] `http://pruvodce.ucw.cz/`

[6] `https://complexityzoo.uwaterloo.ca/`

Taktéž už nyní nekreslíme do samotného widgetu, ale do `self.viewport()`, což je subwidget, který je obalený scrolovátky.

Pojďme se ale zatím podívat na to, jak takový vlastní View vlastně může vypadat.

Zobrazení silnice, chodníku, chodců a aut:
`http://ksp.mff.cuni.cz/viz/31-4-6-view.py`

Zobrazovátko ukazuje (poměrně hloupým a jednoduchým způsobem) pozice chodců a aut na chodníku a silnici. Mimochodem, v protokolu serveru přibylo jedno pole na řádku cestovatele, které říká, ze které strany oblasti dotyčný vstupuje.

Celé psaní View je vlastně poměrně jednoduché – pouze obsluhujeme události – jen je těch událostí a požadavků poměrně hodně. Důkladně si prostudujte přiložený kód, je komentovaný.

V této sérii je důležité připomenout, že máme webové fórum, na kterém je možné pokládat otázky. Je možné, že se vám tentokrát bude hodit.

A nyní přichází na řadu vaše úkoly, kterak zobrazovátko vylepšit.

Úkol 3 [2b]: Zařiďte, aby se auta nemotala mezi chodci, ale aby způsobně podjížděla pod mostem. Nyní se vykreslují, jako by se jednalo o úrovňové křížení.

Úkol 4 [2b]: Naučte chodce chodit po pravé straně chodníku, přičemž polohu mezi pravým okrajem chodníku a jeho středem zvolte náhodně při vstupu chodce do oblasti.

Nyní už umíme spoustu zajímavých věcí a můžeme si dovolit úkol, který shrnuje i zkušenosti z předchozích sérií.

Úkol 5 [6b]: Rozšiřte silnici v obou směrech na alespoň dva jízdní pruhy a naučte auta předjíždět. Auto, které se do provozu nevejde ani ve dvou pruzích, zpomalte na rychlost vozu před ním, dokud vůz před ním neuhne.

Tento úkol bude pravděpodobně vyžadovat výrazné úpravy modelu. Dobře si rozmyslete, jak to udělat tak, abyste příliš neplýtvali procesorovým časem při přepočtech poloh vozidel.

Stejně jako v minulých sériích je možno odevzdat více podúloh v jednom programu. To je v této sérii vše, v příštím, posledním dílu seriálu o Qt se konečně podíváme na to, jak Model ovládat pomocí Delegates.

Maria Matějka

Recepty z programátorské kuchařky: Těžké problémy

Občas se v informatice potkáme s problémem, který nám připadá skutečně těžký, s problémem, na který zatím nikdo nezná efektivní algoritmus. V tomto textu se pokusíme lépe si vysvětlit, co vlastně pro informatika znamená sousloví *těžký problém.*

Úvod a třída problémů $\mathcal{P}$

Když mluvíme o efektivních algoritmech řešících nějaký problém, tak většinou máme na mysli algoritmus běžící v nějakém polynomiálním čase ve vztahu k velikosti vstupu. Například pro problém se vstupem velikosti N to jsou algoritmy, jejichž časovou složitost v nejhorším případě lze omezit shora nějakým polynomem závisejícím na N (sem spadají časové složitosti jako třeba $\mathcal{O}(N)$, $\mathcal{O}(N \log N)$, nebo i $\mathcal{O}(N^5)$). Jestliže v základech časové složitosti tápete, nahlédněte do naší kuchařky o složitosti.[2]

Pokud na problém existuje alespoň jedno známé polynomiální řešení, tak o problému můžeme prohlásit, že leží ve třídě $\mathcal{P}$, neboli skupině úloh řešitelných v polynomiálním čase (třída tu je jen pomocné označení pro nekonečnou množinu). Stále můžeme problém zkoumat a nacházet rychlejší polynomiální řešení, ale pro teorii složitosti stačí, že máme alespoň nějaké.

Jak je to ale s úlohami, u kterých žádný polynomiální algoritmus neznáme? To mohou být třeba problémy, kde nejlepší známé řešení vede přes vyzkoušení všech možností, a jehož časová složitost je tak třeba $\mathcal{O}(2^N)$, neboli exponenciální. Je důležité si uvědomit, že funkce jako 2^N rostou mnohem rychleji, než jakékoliv polynomiální funkce (na názornou tabulku se můžete podívat do již zmíněné kuchařky o složitosti).

I takové problémy chceme nějakým způsobem zařadit do hierarchie složitostních tříd. Než tak ale učiníme, uděláme si malou odbočku – co kdyby nám někdo k problému poskytl i nápovědu, nějaký tahák?

S mapou v bludišti

Představme si, že jsme v bludišti a hledáme nejkratší cestu ven. Můžeme určitě použít prohledávání do šířky[3] a cestu najít v čase lineárním k velikosti bludiště. To je asymptoticky nejlepší možné řešení, v nejhorším případě bude totiž bludiště jedna dlouhá nudle a i nejkratší cesta bude dlouhá lineárně vůči velikosti bludiště.

Jak by se změnila naše situace, kdybychom si ale od kamaráda půjčili tahák – mapu bludiště s vyznačenou nejkratší cestou? Pak by stačilo držet se této cesty a vyběhli bychom nejkratší cestou ven, aniž bychom kdekoliv ztráceli čas.

V nudlovém bludišti (nejkratší cesta má zhruba stejně vrcholů jako celý graf) jsme si vůbec nepomohli, takže je řešení asymptoticky stejně dobré. V alespoň trochu spletitém bludišti už budeme v cíli dříve než náš kamarád, který bloudí (prohledává) do šířky.

V tomto případě nám nápověda tedy zase tolik nepomohla, nalezení cesty z bludiště ven je totiž úloha, kterou umíme vyřešit v polynomiálním čase (patří do třídy $\mathcal{P}$). Pojďme si ale úlohu trochu zkomplikovat a podívejme se, jestli s nápovědou tentokrát umíme dosáhnout lepšího výsledku než bez ní.

Opět jsme v bludišti, ale tentokrát jsou na všech stanovištích umístěny koláčky. Labyrint je to zvláštní, cesty se v něm nekříží, ale je tam plno nadchodů a podchodů (lze tedy říci, že je to obecný nerovinný graf).

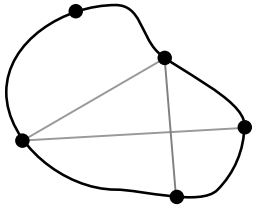
Naším cílem je najít okružní cestu ze startovního místa zpátky na start, abychom každé stanoviště s koláčkem prošli právě jednou (protože víc než jeden koláček nám na žádném stejně nedají).

[2] `http://ksp.mff.cuni.cz/viz/kucharky/slozitost`

[3] `http://ksp.mff.cuni.cz/viz/kucharky/grafy`

Kdybychom tady chtěli použít procházení do šířky, bylo by to opět možné – ale tentokrát bychom se museli mnohokrát vracet, protože posloupnost stanovišť (začátek, první, druhé) může být špatná, neboť nám může zablokovat další cestu. Zároveň ale posloupnost (začátek, druhé, první) už může být dobrá.

Nebude tedy už platit, že každý vrchol při prohledávání navštívíme maximálně jednou, ale každou *posloupnost* stanovišť navštívíme maximálně jednou. Takových posloupností je ale exponenciálně mnoho vzhledem k velikosti bludiště.



Pokud si pořídíme nový tahák, na kterém bude vyznačena optimální cesta přes všechna stanoviště, tak jsme na tom ale stále dobře. Tahák bude mít lineární velikost vzhledem k počtu stanovišť (cestou proběhneme každé z nich právě jednou) a umožní nám tak problém vyřešit v lineárním čase prostým následováním vyznačené cesty.

Našli jsme tedy problém, který nevíme jak vyřešit bez nápovědy v polynomiálním čase (a tedy ho nemůžeme s klidným svědomím zařadit do třídy $\mathcal{P}$), ale s pomocí nápovědy už to umíme. Nedá se takovým způsobem definovat také nějaká třída složitosti? Dá! A to dokonce velmi důležitá.

Certifikáty a nedeterminismus

Vraťme se znovu k naší úloze s koláčky v bludišti. Zde celý problém tkví v tom, že se v některých chvílích prohledávání musíme rozhodnout, jakou z mnoha možností zkusíme nejdříve. Kdybychom pokaždé zvolili správně, tak zvládneme bludiště projít v lineárním čase.

Typický algoritmus, který napíšeme, většinou v případě více možností pokračování zvolí tu první (jeho volba je pevně určená, říkáme jí *deterministická*). Také ale můžeme přemýšlet o algoritmu, který si na každém takovém místě hodí kostkou a podle toho se rozhodne. Takový algoritmus nám na stejném vstupu může dát při různých spuštěních různé výsledky – jeho výpočet není „předurčen“, a proto mu říkáme *nedeterministický*.

Přidejme ale k nedeterministickému algoritmu naši nápovědu neboli *certifikát*. Je to nějaká (vzhledem k velikosti vstupu) polynomiálně velká informace. Můžeme si jej představit jako data, která náš program nalezne v pomocném vstupním souboru, ke kterému program z třídy $\mathcal{P}$ nemá přístup.

Certifikát nám pomůže v každém místě, kde nevíme kudy dál, zvolit tu správnou cestu. Bez něj bychom se museli zkusit vydat každou z nabízených možností, abychom objevili tu správnou, ale s jeho pomocí se vždy vydáme správně a existenci takové cesty ověříme rychle.

Pokud náš algoritmus s použitím takového věšteckého orákula (nebo křišťálové koule, chcete-li), jakým je certifikát, zvládne ověřit řešení problému v polynomiálním čase, říkáme o problému, že je *nedeterministicky polynomiální*, neboli že náleží do třídy $\mathcal{NP}$.

Rozhodovací problémy a třída $\mathcal{NP}$

Aby se nám problémy lépe formálně popisovaly a zařazovaly do tříd, omezíme se v dalším textu jen na *rozhodovací problémy*. To jsou vlastně otázky, na které existují jen dvě možné odpovědi: ANO, nebo NE. Například:

- Existuje cesta z bludiště délky k?
- Je součet čísel $8 + 3$ roven 5?

Jestli se obáváte, že to výrazně sníží množství problémů, které umíme řešit, tak se nemáte proč obávat – skoro vždy se rychlé řešení rozhodovacího problému dá převést na rychlé řešení příslušného vyhledávacího problému jen s nějakým malým zpomalením. Třeba nalezení délky nejkratší cesty z bludiště můžeme udělat pomocí binárního vyhledávání a opakovaného dotazu na existenci cesty délky k (detaily si jako cvičení domyslete).

V úvodu jsme si už řekli, že třída $\mathcal{P}$ představuje problémy řešitelné v polynomiálním čase (u rozhodovacího problému to bude znamenat, že existuje polynomiální algoritmus odpovídající na zadaný vstup korektně ANO, nebo NE). U třídy $\mathcal{NP}$ si ale už musíme dát trošku pozor.

Třída $\mathcal{NP}$ je také třídou problémů. Problém do ní náleží ve chvíli, kdy existuje algoritmus a ke každému zadání, na nějž má být odpověď ANO, navíc i certifikát, pomocí kterého zvládne algoritmus existenci řešení ověřit v polynomiálním čase. Ověřením se myslí to, že odpoví ANO tehdy a jen tehdy, když řešení skutečně existuje.

Zde si dejme pozor na to, že definice nedovoluje „podvádět na druhou“, nemůžeme si do pomocného souboru prostě uložit ANO a pak jej vypsat. Tak by se pak dal řešit libovolně složitý problém, i problémy mimo třídu $\mathcal{NP}$!

Když si certifikát představíme jako ono orákulum, které nám vždy napoví správnou cestu, může být algoritmus nějaké nedeterministické řešení daného problému. Orákulum, ať bude napovídat jakkoliv špatně, ho nikdy nemůže přesvědčit o existenci nějakého řešení, pokud takové neexistuje. To je mimochodem důvod, proč certifikát nemůže být prostě jen ANO: náš algoritmus se nesmí nechat zmást jakkoliv lživým orákulem, tím spíš takovým, které se ani neobtěžuje vymýšlet nám špatné odpovědi a jen nám tvrdí, že řešení existuje.

V reálné situaci (při dokazování, viz níže) si pak často za orákulum (za certifikát) zvolíme optimální řešení úlohy, kterého se stačí držet a najdeme hledanou odpověď (třeba dokážeme, že existuje cesta kratší než k).

Příslušnost do třídy $\mathcal{NP}$ tedy znamená schopnost s pomocí certifikátu dokázat existenci kladného řešení. (To vůbec nemusí znamenat, že dovedeme dokázat jeho *ne*existenci – to by byla zase jiná třída, které se říká *co*$\mathcal{NP}$.)

Asi je vám jasné, že celá třída $\mathcal{P}$ (všechny problémy z ní) jsou součástí i třídy $\mathcal{NP}$ (stačí si za certifikát zvolit třeba prázdný soubor a problém vyřešit normálním polynomiálním algoritmem). A jak už jsme naznačili výše, existují i problémy, jež leží „ještě za třídou $\mathcal{NP}$“, tedy takové, které neumíme vyřešit v polynomiálním čase ani s pomocí certifikátu. Ale dokazování toho, že takové problémy existují, už je nad rámec této kuchařky.

Je $\mathcal{P}$ rovno $\mathcal{NP}$?

Ukázali jsme, že celé $\mathcal{P}$ leží uvnitř $\mathcal{NP}$. Existuje však vůbec nějaký problém, který by byl v $\mathcal{NP}$, ale nebyl by v $\mathcal{P}$? To je otázka, jež trápí informatiky už mnoho let, jeden z nejslavnějších otevřených problémů informatiky.

Vezměme si za příklad problém z povídání o bludišti. Říká se mu *Hamiltonovská kružnice*.

Název problému: Hamiltonovská kružnice

Vstup: Neorientovaný graf.

Problém: Existuje v zadaném grafu kružnice procházející všemi vrcholy právě jednou?

Certifikát: Posloupnost vrcholů hamiltonovské kružnice.

Ověření v polynomiálním čase s certifikátem: Projdeme postupně vrcholy a ověříme, že jsou opravdu zapojeny do kružnice a kružnice je správné délky. Vrátíme NE, pokud tomu tak není.

Zatím nikdo nepřišel s řešením, které by nepoužívalo vůbec žádný certifikát. Dokonce zatím nikdo nenalezl problém, který by byl v $\mathcal{NP}$, ale bez certifikátu už jej nelze řešit v polynomiálním čase. Kdyby takový neexistoval, třídy $\mathcal{P}$ a $\mathcal{NP}$ by se rovnaly. Díky převoditelnosti problémů v $\mathcal{NP}$, které si nyní ukážeme, by dokonce stačilo najít polynomiální řešení bez certifikátu na jediný $\mathcal{NP}$-úplný problém.

Převoditelnost a $\mathcal{NP}$-úplnost

Když řešíme nějakou algoritmickou úlohu, obvykle přijdeme na nějaké přímé řešení využívající základních technik (prohledávání do šířky, dynamické programování, zametací přímka). Vzácně se může i stát, že v problému rozpoznáme problém jiný – občas lze geometrický problém převést na třídění čísel nebo umíme popsat situaci vhodným grafem.

Ukazuje se, že se ve třídě $\mathcal{NP}$ často vyplatí problémy převádět, neboť přímá řešení jsou zde vzácná. Dokonce tak můžeme i zjistit, do které z probíraných tříd problém patří.

Převodem budeme rozumět polynomiální algoritmus, který upraví vstup jednoho problému na vstup jiného problému. Musí navíc problémy převést tak, aby správná odpověď (ANO nebo NE) na vstup prvního problému byla tatáž, jako správná odpověď na vstup druhého problému.

Jednoduchým převodem je úprava problému *Existuje cesta z bludiště ze zadaného políčka délky d?* na *Existuje cesta v grafu délky c začínající v zadaném vrcholu?*.

Do výstupního grafu za každou křižovatku dáme vrchol, za každou cestu mezi křižovatkami hranu a ke hraně si poznamenáme, jak dlouhá byla. Hodnotu c pak můžeme nechat stejně velkou jako d.

Pokud najdeme správnou cestu v tomto grafu, pak nutně podobná cesta je i v bludišti, a pokud cesta v grafu není, pak není ani v bludišti. Převod je tedy korektní.

Zadefinujme si nyní pojem, který nám bude sloužit jako zkratka za to, že problém je ve třídě $\mathcal{NP}$, ale není zároveň lehký (v $\mathcal{P}$). Nemůžeme jen tak ledabyle říci „je v $\mathcal{NP}$ a není v $\mathcal{P}$“, protože to nevíme. To je právě ta slavná otázka.

Uděláme tedy krok stranou – budeme říkat, že problém je *$\mathcal{NP}$-úplný*, pokud onen problém je v $\mathcal{NP}$ a zároveň jdou všechny ostatní problémy v $\mathcal{NP}$ převést na tento problém.

Všechny problémy v $\mathcal{NP}$ na něj jdou převést? Pokud tuto definici vidíte poprvé, asi to působí dost zvláštně – je těžké si představit, že všechny grafové, geometrické, počítací problémy, o kterých víte, že jsou v $\mathcal{P}$ (a tedy i v $\mathcal{NP}$) jdou převést na nějaký $\mathcal{NP}$-úplný superproblém.

Ale je to správně, ba co víc, Cookova věta[4] říká, že existuje alespoň jeden takový problém. (Samotná definice $\mathcal{NP}$-úplného problému nezaručuje, že takový problém vůbec existuje.)

Ukazuje se však, že není sám, jsou jich stovky. Dokazovat existenci dalších $\mathcal{NP}$-úplných problémů je však o dost lehčí než dokázat Cookovu větu! Stačí totiž jen provést následující dva kroky:

- Dokázat, že problém je v $\mathcal{NP}$ – najít certifikát a polynomiální algoritmus, co jej využívá.
- Převést zadání libovolného $\mathcal{NP}$-úplného problému na zadání našeho problému tak, že náš algoritmus vlastně vyřeší onen $\mathcal{NP}$-úplný problém.

To postačí, protože pak libovolný jiný problém v $\mathcal{NP}$ nejprve převedeme na zvolený $\mathcal{NP}$-úplný problém a pak pustíme námi vymyšlený převod. Zřetězení dvou polynomiálních algoritmů (převodů) je opět polynomiální algoritmus, takže podmínka převoditelnosti je splněna.

Ukážeme si důkaz $\mathcal{NP}$-úplnosti jednoho problému na příkladu, pokud nám uvěříte, že již probíraný problém *Hamiltonovská kružnice* je $\mathcal{NP}$-úplný. Nejprve zadefinujme jiný problém:

Název problému: Hamiltonovská cesta.

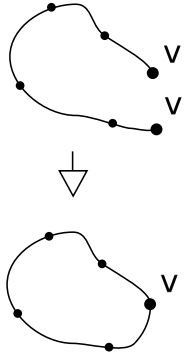
Vstup: Neorientovaný graf, dva speciální vrcholy x a y.

Problém: Existuje cesta z x do y (posloupnost vrcholů, ve které se žádné dva neopakují), která prochází každým vrcholem právě jednou?

Certifikát: Posloupnost vrcholů tvořící správnou cestu.

Řešení v $\mathcal{NP}$: Projděme cestu z certifikátu a ověřme, že vrcholy jdou za sebou, je jich správný počet a žádný jsme nevynechali.

Důkaz $\mathcal{NP}$-úplnosti: Převedeme předchozí problém (hamiltonovskou kružnici) na hledání hamiltonovské cesty. Uvažme graf G, ve kterém chceme najít hamiltonovskou kružnici.



Vyberme si libovolný vrchol v a vytvořme vrchol v', který bude kopií vrcholu v – do grafu přidáme hranu mezi u a v', pokud už v něm je hrana mezi u a v.

Na upravený graf zavoláme řešení problému *Hamiltonovská cesta* mezi vrcholy v a v'. Pokud taková cesta existuje, tak nutně v původním grafu G existuje hamiltonovská kružnice.

Cesta z vrcholu v' přesně odpovídá pokračováni kružnice poté, co přijde do vrcholu v.

Pseudopolynomiální algoritmy

Znáte problém batohu? Jeho varianty jsou oblíbené na programovacích soutěžích. Zadat se může třeba takto: mějme na vstupu seznam N dvojic kladných přirozených čísel, kde každá dvojice označuje váhu a cenu nějakého předmětu. Nakonec dostaneme na vstupu ještě číslo B, které udává nosnost našeho batohu.

[4] http://en.wikipedia.org/wiki/Cook%E2%80%93Levin_theorem