# Korespondenční Seminář z Programování

**38. ročník** **KSP** **Leden 2026**

## *Milí řešitelé, řešitelky a řešitelčata!*

Dostává se k vám první číslo hlavní kategorie 38. ročníku KSP.

Letos se můžete těšit v každé z pěti sérií hlavní kategorie na **4 normální úlohy**, z toho alespoň jednu praktickou opendatovou. Dále na **kuchařky** obsahující nějaká zajímavá informatická témata, hodící se k úlohám dané série. Občas se nám také objeví **bonusová X-ková úloha**, za kterou lze získat X-kové body. Kromě toho bude součástí sérií **seriál**, jehož díly mohou vycházet samostatně.

Autorská řešení úloh budeme vystavovat hned po skončení série. Pokud nás pak při opravování napadnou nějaké komentáře k řešením od vás, zveřejníme je dodatečně.

**Odměny & na Matfyz bez přijímaček**

Za úspěšné řešení KSP můžete být **přijati na MFF UK bez přijímacích zkoušek.** Úspěšným řešitelem se stává ten, kdo získá za celý ročník (hlavní kategorie) alespoň 50 % bodů. Za letošní rok půjde získat maximálně 300 bodů, takže hranice pro úspěšné řešitele je 150. Maturanti pozor, pokud chcete prominutí využít letos, musíte to stihnout do konce čtvrté série, pátá už bude moc pozdě. Také každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok, nálepku na notebook a možná i další překvapení.

**Termín série:** **15. února 2026 ve 32:00 (tedy další ráno v 8:00)**

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

**Značky úloh:** Lehčí úloha (či její část) vhodná pro začátečníky; Praktická open-data úloha; Úloha, u které doporučujeme začíst se do kuchařky; Seriálová úloha; Experimentální (neobvyklá) úloha

**Odměna série:** **Odznáček do profilu na webu** si vysloužíte, kdo **odevzdá řešení aspoň jedné teoretické úlohy v angličtině**.

## Třetí série třicátého osmého ročníku KSP

### 38-3-1 Zahrada s bonsajemi — 10 bodů

Kevin si kdysi dávno umínil, že se stane zahradníkem – a ne jen tak ledajakým! Rozhodl se pěstovat kromě bramborového salátu i svou sebedisciplínu, a do malé mističky si zasadil bonsaj – zakořeněný strom s hranami ohodnocenými délkami, který je třeba pravidelně zastřihovat.

Jenže pak přišel zimní semestr, spousta úkolů a učení se, a bonsaj, jsouc obyčejným stromem, který sebedisciplínu patrně nepěstuje, Kevinovi jaksi přerostla přes hlavu – a to doslova. Kevin, jakožto hippoantropomorfní personifikace ducha semináře, může v závislosti od potřeb konkrétního zapohádkování dosahovat vzpřímen i tři metry – takže si umíte představit, jak špatné světlo (doslova stín) tohle na něj vrhá. A kolik práce dá pečlivé stříhaní takového stromiska… Ale snad nenechá kus nějaké bonsaje, ať mu saje krev! Musí s tím okamžitě něco udělat.

Kevinovi je jasné, že aby byla bonsaj správně *zenová*, nesmí mít víc než $Z$ milimetrů v průměru (to je součet hran na nejdelší cestě, žádná statistická veličina). Bylo by mu ale líto většinu stromu prostě vyhodit – namísto toho se pokusí strom nařezat na několik *zenových* bonsajíček (tedy, pardón, bonsají). Řezání funguje tak, že se ze stromu odstraní hrana mezi rodičem a potomkem, a potomek se stane kořenem nového stromu, zatímco rodič zůstane vrcholem původního.

Kevin ale odmítá hraním si s hranami a motorovou pilou strávit celý den, takže trvá na tom, že počet nařezání (tedy počet nových stromků) bude nejmenší možný. Pomozte mu zjistit, na jakých místech má svůj strom nařezat.

Na vstupu dostanete dvě čísla $N$ a $Z$ – počet vrcholů stromu a maximální možný průměr bonsaje, která je *zenová*. Následuje $N-1$ řádků obsahujících čísla $p_i$ a $l_i$, pro $2 \leq i \leq N$. $p_i$ značí číslo vrcholu, který je rodičem vrcholu $i$, a $l_i$ je délka hrany mezi těmito dvěma vrcholy. Vrcholy číslujeme od 1, vrchol s indexem 1 je vždy kořen stromu.

Jako odpověď uveďte jeden řádek obsahující mezerami oddělená čísla nových kořenů po správném nařezání stromu. Pokud je více optimálních možností, uveďte libovolnou.

### 38-3-2 Stavba sjezdovky — 10 bodů

Obyvatelé Hrochova Týnce se rozhodli na místním svahu postavit sjezdovku. Nechtějí lyžařům kazit výhled na krajinu zátarasy, takže ze začátku sjezdovky mohou lyžaři jet dolů a šikmo do stran. Sjezdovka dole končí vodorovným plotem, takže má tvar trojúhelníku. Na svahu se vyskytují velké kameny a stromy, přes které se moc lyžovat nedá, což omezuje možná umístění. Pomozte jim najít největší volné místo pro sjezdovku.

Najděte největší rovnostranný podtrojúhelník bez překážek na jejich trojúhelníkovém svahu. Pokud je jich víc, stačí libovolný.

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 je číslo $N$ – délka svahu. Následuje několik řádků popisujících překážky na svahu. Na každém řádku s překážkou jsou dvě čísla, která určují její souřadnice. Souřadnice jsou tvořeny vzdáleností od vrchu a vzdáleností od levého kraje. Na všech ostatních souřadnicích je volno.

Všechny souřadnice jsou číslované od jedné, nejvyšší pozice sjezdovky je tedy 1, 1.

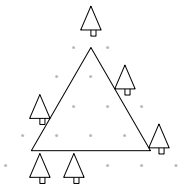
*Formát výstupu:* Na výstup vypište tři čísla – délku sjezdovky a souřadnice jejího začátku.

*Ukázkový vstup:*

```
6
1 1
3 3
4 1
5 5
6 2
6 3
```

*Ukázkový výstup:*

```
3 3 2
```



## 38-3-3 Těžká úloha — 11 bodů

Protože ještě není sjezdovka v Hrochově Týnci hotová, rozhodl se Kevin vyrazit za lyžováním do Alp. Nemá však příliš v lásce dlouhé cesty autem a byl by proto rád, kdyby jeho cesta do Alp trvala co nejkratší dobu.

Bohužel pro něj jsou některé dálnice zpoplatněné vysokým mýtem, a on tak při plánování trasy musí brát ohled nejen na její délku, ale i na svůj malý rozpočet na dopravu, který činí celkem $K$ korun.

Dálniční síť si můžete představit jako orientovaný graf o $N$ vrcholech a $M$ hranách. Hrany tohoto grafu odpovídají dálnicím a každá z nich má zadanou délku a cenu v korunách, kterou je třeba zaplatit, pokud při cestě Kevin dotyčnou hranu použije. Ceny i délky hran jsou vždy nezáporná celá čísla. Vrcholy odpovídají městům, která dálnice spojují. Máme zadaný vrchol $u$, ze kterého Kevin vyráží, a vrchol $v$, který odpovídá cíli Kevinovy cesty.

Kevin by si přál nalézt nějaký efektivní algoritmus, který by splňoval následující tři vlastnosti:

- Algoritmus nejprve na svém vstupu přijme $K$, $u$, $v$ a orientovaný graf s cenami a délkami hran.
- Rozhodne, zda existuje cesta z vrcholu $u$ do vrcholu $v$ taková, že je součet cen hran na této cestě nejvýše $K$. Pokud ano, nalezne nejkratší takovou cestu.
- Běží v polynomiálním čase. Časová složitost algoritmu by tedy měla být $\mathcal{O}(L^c)$, kde $L$ je délka vstupu a $c$ nějaká konstanta.

Zatím si Kevin nad tímto problémem jen marně lámal hlavu. Hodila by se mu proto vaše pomoc. Vaším úkolem však tentokrát nebude problém vyřešit, ale dokázat, že je tato úloha mnohem těžší, než si Kevin myslí.

Před přečtením následující části zadání doporučujeme nastudovat si naši kuchařku o těžkých problémech.[1]

Budete mít za úkol dokázat, že je Kevinův problém takzvaně NP-těžký. To znamená, že lze každý problém z třídy NP převést v polynomiálním čase na tento problém. Od pojmu NP-úplný se liší tím, že samotný problém ve třídě NP ležet nemusí.

Ve vašem řešení se můžete odkazovat na tvrzení z naší kuchařky a smíte ve svém řešení využít libovolný NP-úplný problém, který je v ní uvedený. Pokud použijete nějaké netriviální tvrzení, které v ní není uvedené, měli byste jej ve svém řešení dokázat.

## 38-3-4 Medvědi — 14 bodů

Matúš dnes nemá příliš dobrý den. Nejprve si zapomněl na ubytování telefon a nyní omylem zavedl $U$ účastníků podzimního soustředění do lesa plného $M$ hladových medvědů! Takový les si můžeme představit jako čtvercovou mřížku o celkem $P$ políčkách, kde každé políčko je buď volné, a potom na něm může stát **nejvýše jeden** účastník či **nejvýše jeden** medvěd, nebo je zabrané stromem.

Matúš je odhodlaný zabránit zbytečnému krveprolití, a proto se rozpomněl na vše, co kdy o medvědech slyšel.

Například z hodin přírodopisu ví, že každý medvěd v přírodě chodí výhradně tam a zpět po přesně definované trase. Jakmile však medvěd jednou spatří kořist, opustí bez zaváhání svou obvyklou trasu, rozběhne se za kořistí vysokou rychlostí a zaručeně ji dostihne. Naštěstí pro Matúše tito medvědi neoplývají příliš bystrým zrakem ani dobrým čichem a postřehnou svou oběť jen tehdy, pokud se nachází na políčku sousedícím stranou s jejich.

Matúš proto doufá, že by mohl dát jednotlivým účastníkům instrukce, jak se mají v lese pohybovat, aby každý účastník zvládl postupně les buď zcela opustit, nebo se alespoň zvládl úspěšně vyhýbat medvědům příštích $S$ sekund. Po $S$ sekundách totiž zapadne slunce a medvědi půjdou dozajista spát.

Pohyb v lese probíhá v taktech po jednotlivých sekundách. Každou sekundu se medvědi i účastníci pohnou najednou podle svých instrukcí a na konci taktu nesmí platit, že by nějaký účastník přímo sousedil s medvědem.

Pokud se nějaký účastník nachází na konci taktu na krajním volném políčku a není ohrožen medvědem, může daný účastník les na začátku příštího taktu bezpečně opustit.

Na vstupu dostanete číslo $S$ značící počet sekund do setmění, pozice $U$ účastníků, čtvercovou mřížku odpovídající lesu o celkem $P$ políčkách a $M$ tras medvědů. Každá trasa je charakterizována seznamem políček, po kterých odpovídající medvěd chodí tam a zpět. První políčko seznamu odpovídá medvědově výchozí pozici. Políčka se v seznamu neopakují a navazují na sebe, takže se z každého políčka na následující dá dostat pohybem doleva, doprava, nahoru či dolů.

Váš algoritmus by měl umět rozhodnout, zda je možné zachránit všechny účastníky (aniž by přitom v jakémkoliv okamžiku stáli dva účastníci na stejném políčku), a pokud

[1] `http://ksp.mff.cuni.cz/viz/kucharky/tezke-problemy`

ano, vypsat za každého účastníka seznam instrukcí, kde $i$-tá instrukce seznamu říká, zda se má účastník v čase $i$ pohnout nahoru, dolů, doprava, doleva, nebo zůstat stát na místě.

Upozorňujeme, že účastníků i medvědů může být téměř tolik jako políček, takže by výsledné řešení mělo záviset jen na $S$ a $P$.

Pokud si nejste jistí, jak tuto úlohu řešit, doporučujeme vám si nejprve přečíst naši kuchařku na toky v sítích.[2] Třeba pomůže si postavit nějakou vhodnou síť.

## 38-3-S Kouzlo hešování — 15 bodů

V tomto dílu seriálu prozkoumáme *hešovací tabulky* – kouzelné datové struktury pro množiny a slovníky, které mohou dosáhnout až konstantní časové složitosti operací.

Přečtěte si prosím v Průvodci labyrintem algoritmů kapitolu 11.3. Tam se dozvíte, jak funguje hešování s přihrádkami. Stručně takto:

- Máme nějaké *univerzum* $\mathcal{U}$ možných prvků (třeba všechna celá čísla) a *přihrádky* očíslované 0 až $m-1$ pro nějaké $m$. Každá přihrádka obsahuje seznam svých prvků.
- Pořídíme si *hešovací funkci* $h$, která každému prvku $x \in \mathcal{U}$ přiřadí nějakou přihrádku $h(x)$.
- Datová struktura si pamatuje nějakou $n$-prvkovou podmnožinu univerza $X \subseteq \mathcal{U}$. Prvek $x \in X$ uložíme do přihrádky $h(x)$.
- Kdykoliv hledáme prvek $y \in \mathcal{U}$, díváme se jenom do přihrádky $h(y)$ (jinde prvek být nemůže).
- Pokud se $h$ chová „dostatečně náhodně“, nachází se v každé přihrádce zhruba $n/m$ prvků, řekněme nejvýše $cn/m$ pro nějakou konstantu $c > 0$. Pokud tedy nastavíme $m \geq cn$, bude v každé přihrádce nejvýše 1 prvek.
- Hledání, vkládání i mazání tedy budou mít konstantní časovou složitost, pokud ovšem umíme v konstantním čase spočítat hešovací funkci a také dva prvky porovnat. To platí pro čísla, ale už ne pro řetězce – tam bude složitost záviset na délce řetězce.
- Někdy dopředu nevíme, kolik prvků budeme chtít do tabulky ukládat, abychom podle toho zvolili počet přihrádek. Tehdy nám pomůže postupné zvětšování tabulky s přehešováváním. Zvětšování je pomalé (vyžaduje přerovnat všech $n$ prvků), ale když ho provádíme jen zřídka, zamortizuje se na konstantní čas na vložený prvek. Například funguje jednou za čas počet přihrádek zdvojnásobit.
- Vedle množin můžeme ukládat i *slovníky* (jako v Pythonu) – ty ke *klíčům* (to jsou prvky univerza) přiřazují *hodnoty* (z nějakého jiného univerza). Klíče budou rozprostřené do přihrádek, u každého klíče bude uložena příslušná hodnota. Časová složitost zůstává stejná.

**Úkol 1** [4b]: Na rozšiřování zdvojnásobováním je nepraktické, že jednou za čas se tabulka „zamyslí“ a jedna operace trvá hodně dlouho. Vymyslete, jak rozšiřování zorganizovat tak, aby operace *Find* a *Insert* měly konstantní složitost i v nejhorším případě (*Delete* neprovádíme). V tomto úkolu můžete předpokládat, že hešovací funkce je dost rovnoměrná na to, aby v každé přihrádce bylo nejvýše $dn/m$ prvků pro nějaké $d > 0$. Také můžete předpokládat, že alokace a uvolňování paměti (bez inicializace) trvají konstantní čas.

[2] `http://ksp.mff.cuni.cz/viz/kucharky/toky-v-sitich`

**Kolize a narozeninový paradox**

Situaci, kdy do jedné přihrádky vložíme více prvků, říkáme *kolize*. V naší verzi hešovací tabulky nepůsobí kolize zásadní problémy, ale brzdí ji. Proto by se nám nejvíce líbilo zvolit $m \approx n$ a hešovací funkci, se kterou žádné dva prvky nezkolidují. Taková hešovací funkce se nazývá *perfektní*.

Zatím jsme si představovali, že „dostatečně náhodná“ hešovací funkce do $m \approx n$ přihrádek bude perfektní. Ukážeme, že tato představa je bohužel dost naivní.

Uvažujme *zcela náhodnou* funkci $f$, která zobrazuje $n$ prvků do $m$ přihrádek. (Zcela náhodná znamená, že pro každý prvek zvolíme rovnoměrně náhodně jeho přihrádku, nezávisle na tom, kam umístíme ostatní prvky). Počítejme pravděpodobnost, že $f$ je perfektní.

Všech funkcí z $n$-prvkové množiny do $m$-prvkové je $m^n$ (pro každý z $n$ prvků nezávisle volíme jednu z $m$ přihrádek).

Všech perfektních (prostých) funkcí mezi týmiž množinami je $m \cdot (m-1) \cdot (m-2) \cdot \ldots \cdot (m-n+1)$, což značíme jako $m^{\underline{n}}$ ($n$-tá klesající mocnina čísla $m$). To proto, že pro první prvek máme na výběr $m$ přihrádek, pro druhý jen $m-1$, pro třetí $m-2$ atd.

Pravděpodobnost, že náhodná funkce je perfektní, je tedy rovna podílu

$$\frac{m^{\underline{n}}}{m^n} = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \frac{m-2}{m} \cdot \ldots \cdot \frac{m-n+1}{m} = \\ = 1 \cdot \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{2}{m}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{m}\right).$$

Teď si dovolíme vytáhnout „králíka z klobouku“, totiž nerovnost $1 + x \leq \mathrm{e}^x$, která podle matematické analýzy platí pro každé reálné $x$. Z ní plyne $1 - k/m \leq \mathrm{e}^{-k/m}$, a proto

$$\frac{m^{\underline{n}}}{m^n} \leq 1 \cdot \mathrm{e}^{-1/m} \cdot \mathrm{e}^{-2/m} \cdot \ldots \cdot \mathrm{e}^{-(n-1)/m}.$$

Jelikož $\mathrm{e}^a \cdot \mathrm{e}^b = \mathrm{e}^{a+b}$, můžeme pravou stranu upravit na

$$\mathrm{e}^{-\frac{1+2+\ldots+(n-1)}{m}}.$$

Teď využijeme toho, že aritmetická řada $1+2+\ldots+(n-1)$ má součet $n(n-1)/2 \leq n^2/2$. (Nápověda: sečtěte první prvek s posledním, druhý s předposledním atd.) Docházíme tedy k nerovnosti

$$\frac{m^{\underline{n}}}{m^n} \leq \mathrm{e}^{-\frac{n^2}{2m}}.$$

Všimněte si, že položíme-li $n = m$, bude exponent roven $-n/2$, takže hodnota exponenciální funkce bude velmi blízká nule. Jinými slovy náhodná hešovací funkce $n$ prvků do $n$ přihrádek skoro jistě není perfektní.

Dodejme ještě, že pro $m = cn$ to dopadne podobně, a až pro $m \approx n^2$ bude pravděpodobnost, že náhodná funkce je prostá, rovna cca $1/2$. Tohle je známé pod názvem *narozeninový paradox* – předpokládáme-li, že každý člověk má narozeniny v náhodný den v roce, stačí nám 23 lidí na to, aby s pravděpodobností aspoň $1/2$ měli nějací dva narozeniny ve stejný den. A kdyby funkce nebyla náhodná, byla by pravděpodobnost kolize ještě vyšší.

**Hledáme škodolibý vstup**

Z narozeninového paradoxu víme, že kolize skoro jistě nastane. Ale pořád platí, že pokud hešovací funkce přiděluje přihrádky „dost náhodně“, bude kolizí v průměru málo.

Jenže většina programů používá hešování tak, že zvolí konkrétní hešovací funkci, a pak spoléhá na to, že pro běžný vstup jsou její výsledky dost náhodné.

Ukážeme, že tento přístup se velmi snadno může vymstít, pokud náš uživatel zná hešovací funkci a škodolibě volí taková data, aby způsobil co nejvíc kolizí. Uvážíme následující dvě hešovací funkce: jednu pro univerzum 64-bitových čísel, druhou pro univerzum řetězců. Typ `u32` je 32-bitové celé číslo bez znaménka, sčítání a násobení v něm automaticky probíhají modulo $2^{32}$; podobně `u64` pro 64-bitové.

```
u32 f(u64 x)
{
    return (x * 13043831838999645351) >> 32;
}

u32 g(std::string x)
{
    u32 y = 1;
    for (size_t i=0; i < x.size(); i++)
        y = y * 2654289787 + x[i];
    return y;
}
```

**Úkol 2** [1b]: Najděte kolizi v $f$, tedy dvojici $x$ a $x'$ takovou, že $f(x) = f(x')$.

**Úkol 3** [3b]: Najděte pro $f$ multikolizi: čísla $x_1, \ldots, x_{1\,000\,000}$ taková, že $f(x_i) = f(x_j)$ pro každé $i \neq j$.

**Úkol 4** [2b]: Najděte pro $g$ dva kolidující řetězce, které jsou *hezké:* složené z nejvýše 1000 velkých písmen, malých písmen a číslic.

**Úkol 5** [5b]: Najděte pro $g$ 1 000 000 kolidujících hezkých řetězců.

V úkolech 2 a 4 ukažte konkrétní kolidující prvky. Ve všech čtyřech úkolech na kolize odevzdejte algoritmus, kterým jste kolize vygenerovali, spolu s asymptotickým odhadem průměrné složitosti algoritmu vzhledem k počtu přihrádek hešovací funkce (místo $2^{32}$ předpokládejte obecnou mocninu dvojky) a požadovanému počtu kolidujících prvků.

V příštím dílu vymyslíme, co si se škodolibými uživateli počít. Jako obvykle nás zachrání generátor náhodných čísel!