63. ročník Matematické olympiády – 2013/2014

Úlohy krajského kola kategorie P

Krajské kolo 63. ročníku MO kategorie P se koná v úterý 21. 1. 2014 v dopoledních hodinách. Na řešení úloh máte 4 hodiny čistého času. V krajském kole MO-P se neřeší žádná praktická úloha, pro zajištění rovných podmínek řešitelů ve všech krajích je použití počítačů při soutěži zakázáno. Zakázány jsou rovněž jakékoliv další pomůcky kromě psacích potřeb (např. knihy, výpisy programů, kalkulačky, mobilní telefony). Řešení každé úlohy vypracujte na samostatný list papíru.

Řešení každé úlohy musí obsahovat:

- **Popis řešení,** to znamená slovní popis principu zvoleného algoritmu, *argumenty zdůvodňující jeho správnost* (případně důkaz správnosti algoritmu), diskusi o efektivitě vašeho řešení (časová a paměťová složitost). Slovní popis řešení musí být jasný a srozumitelný i bez nahlédnutí do samotného zápisu algoritmu (do programu). Není vhodné odkazovat se na vaše řešení úloh domácího kola, opravovatelé je nemusí mít k dispozici; na autorská řešení domácího kola se odkazovat můžete.
- **Zápis algoritmu.** V úlohách **P-II-1**, **P-II-2** a **P-II-3** je třeba uvést zápis algoritmu v nějakém dostatečně srozumitelném pseudokódu (případně v programovacím jazyce Pascal nebo C/C++). Nemusíte detailně popisovat jednoduché operace jako vstupy, výstupy, implementaci jednoduchých matematických vztahů, vyhledávání v poli, třídění apod. V úloze **P-II-4** je nutnou součástí řešení program, ve kterém můžete využívat volání funkcí „mimozemských počítačů KSP“.

Za každou úlohu můžete získat maximálně 10 bodů. Hodnotí se nejen správnost řešení, ale také kvalita jeho popisu a efektivita zvoleného algoritmu. Algoritmy posuzujeme podle jejich časové složitosti, tzn. závislosti doby výpočtu na velikosti vstupních dat. Záleží přitom pouze na řádové rychlosti růstu této funkce. V zadání každé úlohy najdete přibližné limity na velikost vstupních dat. Efektivním vyřešením úlohy rozumíme to, že váš program spuštěný s takovými daty na současném běžném počítači dokončí výpočet během několika sekund.

Vzorová řešení úloh naleznete krátce po soutěži na webových stránkách olympiády *http://mo.mff.cuni.cz/*. Na stejném místě bude zveřejněn i seznam úspěšných řešitelů postupujících do ústředního kola a také popis prostředí, v němž se budou v ústředním kole řešit praktické úlohy.

P-II-1 Podposloupnost

Dostanete číslo k a posloupnost n čísel. Napište program, který v zadané posloupnosti čísel určí nejdelší souvislou podposloupnost, jejíž aritmetický průměr je přesně roven hodnotě k.

Popis vstupu

Na prvním řádku vstupu jsou dvě čísla n a k. Na druhém řádku je n kladných celých čísel: prvky posloupnosti. Můžete předpokládat, že vstupní posloupnost čísel vždy obsahuje alespoň jednu souvislou podposloupnost s průměrem prvků přesně rovným k. Můžete také předpokládat, že se součet všech prvků posloupnosti vejde do běžné celočíselné proměnné.

Popis výstupu

Program vypíše dvě celá čísla – pozici začátku a pozici konce nejdelší vhodné podposloupnosti. Pozice čísel v posloupnosti číslujeme od 1 do n. Pokud existuje více různých vhodných podposloupností téže maximální délky, program vypíše jednu libovolnou z nich.

Hodnocení

Plných 10 bodů získáte za řešení, které zvládne efektivně vyřešit libovolný vstup délky $n \leq 200\,000$. Až 6 bodů dostanete za řešení, které efektivně vyřeší každý vstup délky $n \leq 5\,000$. Za jakékoliv funkční řešení bez ohledu na jeho efektivitu můžete získat až 4 body.

Příklady

Vstup:	*Výstup:*
`7 4` `1 1 3 8 1 5 2`	`4 7`

Existují tři podposloupnosti s průměrem rovným přesně 4, a to $(1, 3, 8)$, $(3, 8, 1)$ *a* $(8, 1, 5, 2)$. *Třetí z nich je nejdelší, takže vypíšeme pozici jejího začátku a konce.*

Vstup:	*Výstup:*
`4 5` `2 3 5 1`	`3 3`

P-II-2 Body v rovině

V rovině je pevně rozmístěno n bílých bodů. Vašemu programu budeme postupně zadávat souřadnice q černých bodů. Pro každý zadaný černý bod program vždy odpoví, zda existuje taková přímka v rovině, která odděluje tento černý bod od všech bílých bodů. Pro oddělující přímku musí platit, že všechny bílé body leží na jedné straně přímky (nebo případně některé z nich mohou ležet na této přímce), zatímco černý bod leží na druhé straně přímky (nesmí ležet na přímce). Jestliže taková přímka existuje, program musí jednu takovou přímku nalézt.

Můžete pro jednoduchost předpokládat, že všechny výpočty a porovnávání jsou přesné (tj. že nenastávají žádné zaokrouhlovací chyby).

Popis vstupu

Na prvním řádku vstupu je kladné celé číslo $n \geq 3$, označující počet bílých bodů. Na každém z dalších n řádků se nacházejí souřadnice jednoho bílého bodu. Následuje řádek s číslem q, které představuje počet postupně zkoumaných černých bodů. Na každém z dalších q řádků jsou souřadnice jednoho černého bodu.

Můžete předpokládat, že všechny bílé body jsou navzájem různé a že žádné tři bílé body neleží na přímce. Dále můžete předpokládat, že žádný černý bod nemá souřadnice shodné s některým z bílých bodů.

Popis výstupu

Program vypíše na výstup přesně q řádků. Každý z nich odpovídá jednomu ze zkoumaných černých bodů (v pořadí, v němž jsou černé body zadány na vstupu). Na každém řádku je uveden buď text „`Primka neexistuje`“, pokud neexistuje žádná přímka vyhovující zadání úlohy, nebo 4 reálná čísla x_1, y_1, x_2, y_2: souřadnice dvou různých bodů ležících na jedné z vyhovujících přímek.

Hodnocení

Plných 10 bodů získáte za řešení, které zvládne efektivně vyřešit libovolný vstup velikosti $n \leq 100\,000$, $q \leq 100\,000$. Za jiné řešení s polynomiální časovou složitostí dostanete 5–7 bodů podle efektivity. Jestliže váš program správně zjistí, zda existuje přímka vyhovujících vlastností, ale nedokáže tuto přímku nalézt, strhneme vám v hodnocení 1–2 body.

Příklad

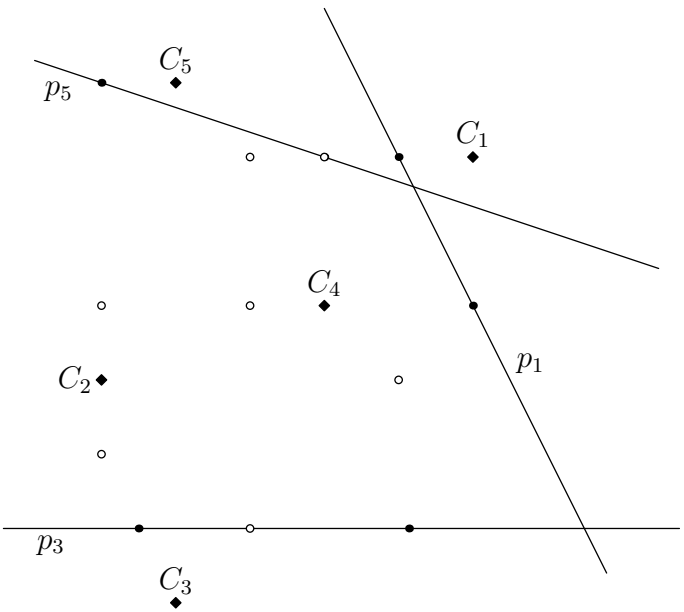
Vstup:

```
7
-1 -1
2 3
1 -2
1 1
3 0
1 3
-1 1
5
4 3
-1 0
0 -3
2 1
0 4
```

Výstup:

```
4 1 3 3
Primka neexistuje
-0.5 -2 3.14 -2
Primka neexistuje
2 3 -1 4
```

Na obrázku bílé kroužky představují bílé body, diamanty jsou jednotlivé černé body. Řešení z ukázkového výstupu jsou znázorněna přímkami a černými kroužky (vypsané body).



P-II-3 Počítačová síť

Počítačová síť se skládá z n počítačů, které jsou propojeny přesně n kabely. Kabelů je tedy stejné množství jako počítačů. Každé dva počítače mohou pomocí těchto kabelů mezi sebou komunikovat (možná ne přímo, ale třeba přes několik jiných počítačů). Žádné dva počítače nejsou přímo propojeny více než jedním kabelem.

Nepřítel chce počítačovou síť zničit přestřižením co největšího množství kabelů. Jakmile ale přestřihne nějaký kabel, aktivuje tím ochranné mechanismy obou počítačů, které jsou tímto kabelem spojeny. Kdyby se následně pokusil přestřihnout jiný kabel vedoucí do některého z těchto počítačů, určitě by ho chytili.

Zjistěte, kolik kabelů nejvýše může nepřítel postupně přestřihnout, aniž by ho chytili.

Popis vstupu

První řádek vstupu obsahuje celé číslo n ($n \geq 3$), které označuje počet počítačů a také počet kabelů v síti. Počítače jsou označeny celými čísly od 1 do n. Na každém z n následujících řádků jsou dvě čísla a a b ($1 \leq a, b \leq n; a \neq b$), která udávají, že počítače a a b jsou spojeny kabelem.

Popis výstupu

Program vypíše jeden řádek a na něm jedno celé číslo: největší možný počet přestřižených kabelů.

Hodnocení

Plných 10 bodů získáte za řešení, které zvládne efektivně vyřešit libovolný vstup s $n \leq 1\,000\,000$. Až 6 bodů dostanete za řešení, které efektivně vyřeší každý vstup s $n \leq 100$. Za jakékoliv funkční řešení bez ohledu na jeho efektivitu můžete získat až 4 body.

Příklady

Vstup:	*Výstup:*
6 1 2 2 3 4 3 5 4 2 5 2 6	2

Můžeme přestřihnout například kabely 2–6 a 4–5.

Vstup:	*Výstup:*
6 1 5 2 3 4 3 5 4 2 5 2 6	3

Tentokrát je možné přestřihnout kabely 1–5, 4–3 a 2–6.

P-II-4 Mimozemské počítače

K této úloze se vztahuje studijní text uvedený na následujících stranách. Studijní text je identický se studijním textem z domácího kola, pouze v seznamu problémů najdete dva nové problémy: **housenkovou kostru** *a* **pokrytí podmnožinami.** *Naopak je vynecháno barvení grafu a k-partitnost posloupnosti, které se v tomto kole nepoužívají.*

V této úloze nezáleží na časové složitosti vašich algoritmů – musí ale být polynomiální. Jednotlivé podúlohy spolu nesouvisí a každá z nich bude hodnocena samostatně. Můžete je řešit v libovolném pořadí.

Soutěžní úloha

a) *(3 body)* Mimozemšťané nám dodali sálový KSP, který rozhoduje problém existence housenkové kostry v daném neorientovaném grafu. Tento KSP má tedy funkci `housenka(n,E)`, která dostává jako parametry počet n vrcholů grafu a seznam E jeho hran. Pokud v daném grafu existuje housenková kostra, KSP rozsvítí zelené světlo, jinak rozsvítí světlo červené. Tuto funkci můžete použít **jenom jednou** a jejím zavoláním výpočet vašeho programu skončí.

Na vstupu dostanete neorientovaný graf G. Napište program s polynomiální časovou složitostí, který rozsvítí zelené nebo červené světlo podle toho, zda G obsahuje alespoň jednu hamiltonovskou cestu.

b) *(3 body)* Na vstupu opět dostanete neorientovaný graf G. S použitím stejného sálového KSP jako v podúloze **a)** napište program s polynomiální časovou složitostí, který rozsvítí zelené nebo červené světlo podle toho, zda G obsahuje alespoň jednu hamiltonovskou kružnici.

c) *(4 body)* Mimozemšťané nám dodali kufříkový KSP, který rozhoduje problém existence dostatečně dobrého pokrytí podmnožinami. Tento KSP má tedy funkci `existuje_pokryti(k,n,m,Z)`, která dostává následující parametry:

- k je požadovaný počet podmnožin v pokrytí,
- n udává počet prvků množiny, kterou chceme celou pokrýt,
- m určuje počet podmnožin, které máme na výběr,
- Z je seznam těchto podmnožin (např. dvojrozměrné pole, kde $Z[i,j] = 1$ právě tehdy, když v i-té podmnožině leží prvek j).

Na výstupu funkce vrací `True` nebo `False` podle toho, zda lze mezi zadanými m podmnožinami vybrat nejvýše k podmnožin tak, aby jejich sjednocení obsahovalo všech n prvků množiny. Tuto funkci můžete ve svém programu volat libovolně mnohokrát pro jakékoliv vstupy.

Řešíme následující problém: Ve škole je z žáků (očíslovaných od 0 do $z-1$) a škola pro ně organizuje celkem k zájmových kroužků. Pro každý kroužek známe neprázdný seznam žáků, kteří ho navštěvují. V každém kroužku si mezi žáky, kteří ho navštěvují, zvolili svého předsedu. Přitom jeden žák mohl být zvolen předsedou ve více kroužcích. Předsedové všech zájmových kroužků tvoří radu předsedů. Kolik nejméně členů může mít tato rada?

Vyřešte tento problém pomocí popsaného kufříkového KSP. Počet bodů, které získáte, bude záviset na počtu použití funkce `existuje_pokryti` (čím méně, tím lépe).

Programovací jazyky

Ve svých řešeních můžete používat libovolný strukturovaný programovací jazyk. Vhodně si zvolte potřebné datové struktury. Například v Pascalu by funkce z podúlohy **a)** mohla vypadat následovně:

```
type hrana = array[0..1] of longint;
procedure housenka(n: longint; m: longint; E: array of hrana);
```

V jazyce C++ můžeme použít buď pole:

```
void housenka(int n, int m, int E[][2]);
```

nebo třeba také vektor dvojic:

```
void housenka(int n, vector< pair<int,int> > E);
```

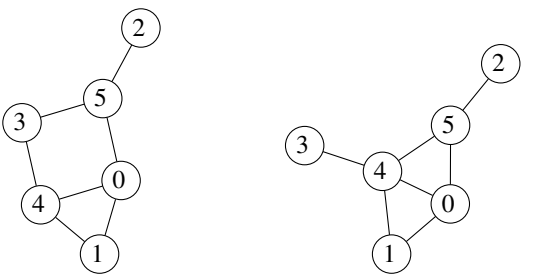
Studijní text

Studijní text uvedený na dalších stranách se odkazuje na několik problémů, jako například „existence hamiltonovské kružnice“. Seznámíme se proto nejprve se stručnou definicí těchto problémů.

Neorientovaný graf je uspořádaná dvojice (V, E), kde V je konečná množina objektů zvaných *vrcholy* grafu a E je konečná množina neuspořádaných dvojic vrcholů; její prvky nazýváme *hrany* grafu. Graf si můžeme představit jako silniční síť: V je množina měst a E je množina dvojic měst spojených silnicemi. Počet vrcholů budeme značit n, počet hran označíme m. Jednotlivé vrcholy budeme číslovat od 0 do $n - 1$.

Problém: Hamiltonovská cesta z u do v

Je dán neorientovaný graf G a jeho dva různé vrcholy u a v. Existuje v grafu G cesta, která začíná ve vrcholu u, končí ve vrcholu v a navštíví každý vrchol grafu G právě jednou?

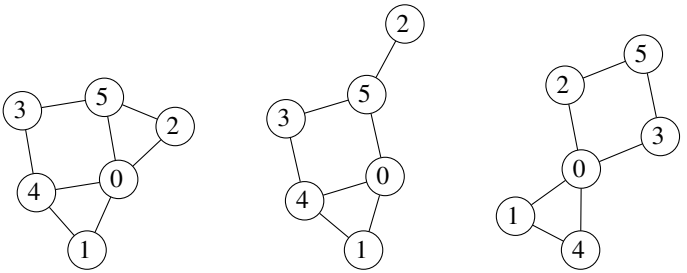


Graf vlevo obsahuje hamiltonovskou cestu z 1 do 2: můžeme jít postupně přes vrcholy 1, 0, 4, 3, 5 a 2.

Graf vpravo hamiltonovskou cestu z 1 do 2 neobsahuje: když chceme navštívit vrchol 3, půjdeme dvakrát přes vrchol 4.

Problém: Hamiltonovská kružnice

Je dán neorientovaný graf G s $n \geq 3$ vrcholy. Existuje v grafu G kružnice, která navštíví každý vrchol grafu G právě jednou?



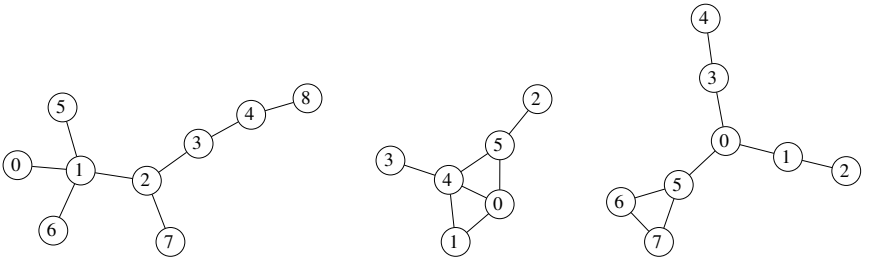
Graf vlevo obsahuje hamiltonovskou kružnici: začneme třeba v 1 a postupně jdeme do 0, 2, 5, 3, 4 a zpět do 1.

Graf uprostřed ani graf vpravo hamiltonovskou kružnici neobsahují.

Problém: Housenková kostra

Housenka je graf, který můžeme vytvořit následujícím způsobem: vezmeme nějakou cestu (tvořenou alespoň dvěma vrcholy) a přidáme k ní několik dalších vrcholů tak, že každý z nich připojíme hranou k některému vrcholu cesty. Jinými slovy řečeno, n-vrcholový graf je housenkou právě tehdy, když má přesně $n-1$ hran a existuje v něm taková cesta, že každý vrchol neležící na této cestě má jediného souseda a ten leží na této cestě.

Problém housenkové kostry je potom definován takto: Je dán neorientovaný n-vrcholový graf. Je možné smazat některé jeho hrany tak, abychom dostali n-vrcholovou housenku?



Graf vlevo je housenka, jednu možnou cestu tvoří vrcholy 0, 1, 2, 3, 4. Graf uprostřed obsahuje housenku. Abychom ji získali, stačí smazat například hrany 1–4 a 4–5. Z grafu vpravo mazáním hran nelze housenku vytvořit.

Problém: Pokrytí množinami

Jsou dána čísla n a k a sada množin. Každá množina je podmnožinou množiny $\{0, 1, 2, \ldots, n-1\}$. Úlohou je zjistit, zda je možné vybrat z dané sady množin nejvýše k množin tak, aby jejich sjednocením byla celá množina $\{0, 1, 2, \ldots, n-1\}$.

- Pro $n = 5$, $k = 2$ a množiny $\{1, 3, 4\}$, $\{0, 3\}$, $\{0, 1\}$ a $\{0, 1, 2, 4\}$ zní odpověď ANO: můžeme vzít například druhou a čtvrtou množinu.
- Pro $n = 5$, $k = 2$ a množiny $\{0, 1, 3\}$, $\{4\}$, $\{0, 1\}$ a $\{0, 1, 2\}$ zní odpověď NE.
- Pro $n = 5$, $k = 4$ a množiny $\{0, 1, 3\}$, $\{4\}$, $\{0, 1\}$ a $\{0, 1, 2\}$ zní odpověď ANO: můžeme vzít například první, druhou a čtvrtou množinu (nebo dokonce všechny čtyři).

KSP – konkrétní semiorganické počítače

Píše se rok 2113. Naši Zemi před několika lety kontaktovala vyspělá mimozemská rasa z planety Žbluňk. Mimozemšťané nás zásobují svými konkrétními semiorganickými počítači (KSP), které umí řešit různé konkrétní výpočetní problémy. Naším cílem je integrovat tyto KSP do normálních počítačů a přinutit je tak řešit naše problémy.

Mimozemským počítačům ale vůbec nerozumíme, všechny snahy o jejich rozebrání byly neúspěšné. Můžeme je využít jediným způsobem – zadat jim vstupní data a počkat na výsledek. Zajímavé je, že u KSP nezáleží na velikosti vstupních dat ani na konkrétním problému, který daný KSP řeší. Když libovolný KSP spustíme s libovolnými vstupními daty, výsledek dostaneme vždy přesně za 47 setin sekundy. (Při odhadování časové složitosti programů toto považujeme za konstantu.)

Kufříkový KSP

Mimozemšťané nám dodávají dva druhy KSP: *kufříkové* a *sálové*.

Kufříkový KSP má tvar kufříku se dvěma porty. Do jednoho připojíme kabel se vstupem, do druhého naopak kabel, po němž nám KSP pošle svůj výstup. Kufříkový KSP tedy můžeme použít v našem programu libovolně mnohokrát s různými vstupními hodnotami.

Příklad

Mimozemšťané nám dodali kufříkový KSP, který rozhoduje problém existence hamiltonovské cesty z u do v. Tento KSP počítá funkci `cesta(n,E,u,v)`, která dostává jako parametry počet n vrcholů grafu, seznam E jeho hran a dvě čísla vrcholů u a v. Parametr E je seznam m dvojic čísel, každé z rozsahu od 0 do $n - 1$. Na výstupu tato funkce vrací `True` nebo `False` podle toho, zda zadaný graf obsahuje příslušnou hamiltonovskou cestu.

Naším úkolem je napsat program, který bude v polynomiálním čase řešit problém existence hamiltonovské kružnice. Na vstupu dostaneme neorientovaný graf s $n \geq 3$ vrcholy a máme zjistit, zda obsahuje hamiltonovskou kružnici.

Analýza zadání

1. Program na vstupu dostane n (počet vrcholů grafu) a E (seznam hran grafu).
2. Vykoná nějaké výpočty, při nichž může libovolně používat funkci `cesta`.
3. Vrátí na výstupu `True` nebo `False` podle toho, zda vstupní graf obsahuje hamiltonovskou kružnici.

Řešení

```
def kruznice(n,E):
    for (x,y) in E:
        # pro každou hranu (x,y) v seznamu hran E:
        newE = [ (u,v) for (u,v) in E if (u,v) != (x,y) ]
        # NewE obsahuje všechny hrany kromě (x,y)
        if cesta(n,newE,x,y): return True
    return False
```

Postupně si pro každou hranu (x, y) zadaného grafu položíme otázku: „Existuje hamiltonovská kružnice procházející hranou (x, y)?“ Kdy taková kružnice existuje? Právě tehdy, když ve zbytku grafu existuje hamiltonovská cesta z x do y. Vytvoříme si tedy nový seznam hran `newE`, do kterého vložíme všechny hrany kromě (x, y), a na takto upravený graf zavoláme funkci `cesta` našeho kufříkového KSP.

Jestliže v původním grafu nějaká hamiltonovská kružnice existuje, náš program ji najde a vrátí odpověď `True`, jakmile vyzkoušíme některou z hran tvořících kružnici jako (x, y). Naopak, jestliže náš program odpoví `True`, pak v původním grafu existuje hamiltonovská cesta z nějakého vrcholu x do nějakého vrcholu y. Tato cesta společně s hranou (x, y) tvoří hledanou kružnici. Náš program tedy skutečně dělá to, co má.

Časová složitost popsaného programu je $\Theta(m^2)$, kde m je počet hran zadaného grafu. Protože v neorientovaném grafu platí $m \leq n(n-1)/2$, můžeme naši časovou složitost shora odhadnout jako $\mathcal{O}(n^4)$.

Poznámka

Existuje i efektivnější řešení. Stačí si uvědomit, že před hledáním hamiltonovské cesty z x do y vůbec nemusíme odstraňovat hranu (x, y) z grafu. Hamiltonovská cesta z x do y v grafu s $n \geq 3$ vrcholy ji stejně nemůže obsahovat. Stačí tedy pro každou hranu (x, y) zjistit, zda platí `cesta(n,E,x,y)`.

Sálový KSP

Sálový KSP je obrovský a jeho jediným výstupem jsou dvě světla – červené a zelené. S tímto výstupem dále nepracujeme.

Příklad

Mimozemšťané nám dodali sálový KSP, který rozhoduje problém 3-partitnosti. Tento KSP má funkci `tri_partitnost(X)`, která rozsvítí zelené nebo červené světlo podle toho, zda posloupnost X je 3-partitní – tedy zda se její prvky dají rozdělit do *tří* skupin se shodným součtem.

Na vstupu dostanete posloupnost kladných celých čísel A. Napište program s polynomiální časovou složitostí, který rozsvítí zelené nebo červené světlo podle toho, zda je posloupnost A 2-partitní (tzn. zda můžeme prvky posloupnosti A rozdělit do *dvou* skupin se shodným součtem).

Analýza zadání

1. Program na vstupu dostane posloupnost čísel A.
2. Vykoná nějaké výpočty a vytvoří nějakou novou posloupnost čísel X.

3. Na konci (každé možné větve) výpočtu jednou zavolá funkci `tri_partitnost(X)`, která rozsvítí správné světlo.

Řešení

```
def dva_partitnost(A):
    s = sum(A)
    if s%2 == 0:                # s%2 je zbytek po dělení s dvěma
        X = A + [ s//2 ]        # // je celočíselné dělení
    else:
        X = [1]                 # [1] je posloupnost,
                                # která určitě není 3-partitní
    tri_partitnost(X)
```

Nechť jsme dostali vstupní posloupnost $A = (a_1, \ldots, a_n)$. Spočítáme si její součet s.

Je-li s sudé, přidáme na konec vstupní posloupnosti ještě jeden prvek s hodnotou $s/2$. Takto upravenou posloupnost pošleme do KSP. Ten nám odpoví, zda je tato posloupnost 3-partitní. Jenže upravená posloupnost je 3-partitní právě tehdy, když byla původní posloupnost 2-partitní. KSP tedy za nás právě vyřešil zadanou úlohu.

Proč platí výše uvedená ekvivalence? V nové posloupnosti X je součet všech prvků roven $3s/2$. Jestliže je tedy X 3-partitní, pak každá ze skupin, na něž X rozdělíme, má součet rovný přesně $s/2$. Potom ale nutně jednu z těchto tří skupin tvoří samotný přidaný prvek s hodnotou $s/2$. Naše nová posloupnost je proto 3-partitní právě tehdy, když je možné ostatní prvky rozdělit do dvou skupin se stejným součtem – tzn. právě tehdy, když je původní posloupnost A 2-partitní.

Je-li s liché, víme rovnou, že musíme odpovědět NE. Do KSP proto chceme poslat libovolný vstup, pro který dá KSP zápornou odpověď. Takovým vstupem je třeba $X = (1)$ nebo $X = (1, 1, 2)$.

Časová složitost tohoto programu je lineární vzhledem k délce posloupnosti X.