Milí řešitelé a řešitelky!

Přinášíme vám vaše opravená a naše vzorová řešení čtvrté série. Letošní ročník se tak chýlí ke konci, zbývá už jenom vyřešit a opravit poslední pátou sérii. Těšíme se na vaše výtvory!

Termín odeslání šesté série jest pochopitelně neurčen, alébrž žádná šestá série není. Řešení byste mohli odevzdávat jak elektronicky na *http://ksp.mff.cuni.cz/submit/*, tak na:

Korespondenční seminář z programování
KSVI MFF UK
Malostranské náměstí 25
Praha 1, 118 00

Aktuální informace o KSP naleznete na stránkách *http://ksp.mff.cuni.cz/*. Dotazy ohledně zadání můžete posílat na adresu *ksp@mff.cuni.cz*, nebo se ptát přímo na diskusním fóru KSP (*http://ksp.mff.cuni.cz/forum/*).

Vzorová řešení příkladu 18-3-4 a čtvrté série osmnáctého ročníku KSP

18-3-4 Pochoutka pro prasátko

Máme šachovnici o rozměrech $X \times Y$ a sadu K pravidel, podle nichž se prasátko umí pohybovat s určitou námahou. Krátké pozorování odhalí, že každé políčko šachovnice je jeden vrchol grafu a že mezi vrcholy vede hrana právě tehdy, pokud existuje pravidlo převádějící prasátko z jednoho vrcholu na druhý. Pak je hrana samozřejmě ohodnocena příslušným množstvím námahy. A jelikož jsou hrany kladně ohodnocené a my hledáme nejkratší cestu ze startovní pozice hladovějícího pašíka na naleziště Velké Bukvice, máme úlohu jako dělanou (ve skutečnosti opravdu dělanou) pro použití kuchařkového Dijkstrova algoritmu s haldou.

Ukázalo se ale, že naprogramovat takový algoritmus nemusí být až tak jednoduché. Někteří těžce válčili s haldou, jiní v boji podlehli a zaslali jen slovní popis algoritmu.

První otázka je, jak si vyrobit onen graf zobrazující prostor lesa. Odpověď je jednoduchá. Žádný graf není třeba vyrábět, budeme pracovat přímo nad políčky lesa a hledané hrany si budeme konstruovat přímo v okamžiku, kdybychom se v Dijkstrově algoritmu dívali na sousedy aktuálně zkoumaného vrcholu. Postupně použijeme všechna možná pravidla pro pohyb z daného políčka a podíváme se, jestli jsme se nedostali mimo les.

Druhý, horší problém, vzniká u haldy. V okamžiku, kdy v Dijkstrově algoritmu najdeme lepší cestu a přepočítáváme vzdálenost nějakému vrcholu, mění se samozřejmě jeho pozice v haldě vrcholů a haldu musíme přeskládat. Jak na to?

Můžeme si někde bokem pamatovat, kde přesně se každý vrchol v haldě nachází, a pustit na něj bublání. Pak ale musíme při jakékoli operaci s haldou každému vrcholu přepočítávat tento jeho index v haldě a to je trošku zmatek.

Jiné, jednodušší řešení je haldu nijak nepředělávat, a když nějakému vrcholu přepočítáme vzdálenost, prostě jej do haldy strčit znovu. Tak se nám některé vrcholy mohou v haldě opakovat, ale my dokážeme v Dijkstrově algoritmu při vytahování minimálního prvku z haldy snadno rozeznat, jestli jej máme zpracovávat, nebo jestli je to jen zopakovaný prvek. Poznáme to podle toho, jestli už má trvalou hodnotu.

Za jednodušší řešení ale zaplatíme. Zatímco v těžším, „přepočítávacím“ řešení se každý prvek dostane do haldy nejvýš jednou, takže halda může zabírat jen tolik místa, jaký je počet vrcholů grafu, u druhého řešení se prvky mohou dostat do haldy víckrát, konkrétně halda může být veliká jako počet hran grafu.

Dijkstrův algoritmus z kuchařky trvá $\mathcal{O}((N + M) \cdot \log N)$, kde N je počet vrcholů, u nás $X \times Y$, a M počet hran, u nás XYK. Za každou operaci s haldou násobíme logaritmem velikosti haldy. Pokud tedy použijeme haldu s přepočítáváním, dostaneme časovou složitost $\mathcal{O}(XYK \cdot \log(XY))$. Jednodušší halda dá časovou složitost $\mathcal{O}((N+M) \cdot \log M) \leq \mathcal{O}((N + M) \cdot \log N^2) = \mathcal{O}((N + M) \cdot 2 \log N) = \mathcal{O}((N + M) \cdot \log N)$, takže vlastně tutéž.

Paměťová složitost je u haldy s přepočítáváním $O(XY)$, protože si potřebujeme pamatovat jen les a haldu na vrcholy, ale u větší haldy až $O(XYK)$.

Jana Kravalová

Jak si všiml Pepa Pihera, náš algoritmus jde ještě vylepšit. Malou úpravou dosáhneme toho, že v haldě bude vždy nejvýš K prvků, čímž stlačíme složitost na $\mathcal{O}(XYK \cdot \log K)$. (Platí $K \leq XY$, protože pokud by pravidel bylo více, na některé políčko by se dalo dostat pomocí více pravidel a my si můžeme nechat jenom to lepší z nich.)

V jednom kroku se Dijktrův algoritmus pokouší najít vrchol s nejmenším dočasným ohodnocením. Jinak řečeno, hledá takový nezpracovaný vrchol spojený s už zpracovaným vrcholem, že součet ohodnocení zpracovaného vrcholu a hrany z něj vedoucí je co nejmenší. Navíc vrcholy zpracováváme (trvale ohodnocujeme) podle jejich vzdálenosti od výchozího místa, tedy v neklesajícím pořadí.

Zvolme si pro tento odstavec jediné pravidlo. Kromě krajních případů ho můžeme použít z každého vrcholu. Sledujme vrcholy, které pomocí tohoto pravidla dostanou trvalé ohodnocení. V průběhu algoritmu je ohodnocení těchto zpracovávaných vrcholů neklesající. Protože jsme ho získali přičtením hodnoty pravidla k ohodnocení výchozímu vrcholu, je i ohodnocení vrcholů, *ze* kterých toto pravidlo používáme, neklesající. Toto jediné pravidlo tedy používáme na vrcholy v tom pořadí, v jakém je trvale ohodnocujeme.

Když víme, že každé pravidlo používáme na vrcholy v tom pořadí, v jakém je trvale ohodnocujeme, zapamatujeme si u každého pravidla, ze kterého vrcholu jsme ho naposledy použili. Když potom hledáme vrchol s nejnižším dočasným ohodnocením, každé pravidlo už má určený vrchol, ze kterého ho použijeme. Vybereme si tedy tu nejlepší kombinaci vrchol-pravidlo, tím jsme našli další vrchol s trvalým ohodnocením, a použité pravidlo „posuneme“ k dalšímu vrcholu. Tím myslíme, že příště ho budeme používat z vrcholu, který jsme v Dijkstrovi trvale ohodnotili hned po tom vrcholu, ze kterého jsme teď pravidlo používali.

V každém kroku tedy potřebujeme najít minimum z K hodnot, toto minimum odstranit a přidat místo něj jinou hodnotu. K tomu je halda jako stvořená, všechny tyto operace zvládne v čase $\mathcal{O}(\log K)$. Navíc každou hranu zpracujeme právě jednou, čímž se dostáváme na slibovanou složitost $\mathcal{O}(XYK \log K)$.

Milan Straka

18-4-1 HP

Až na drobné výjimky všechna došlá řešení fungovala. Svůj podíl na tom má to, že úloha patřila k těm nejjednodušším ze série. Někteří řešitelé si však přesto počínali poněkud neohrabaně a zbytečně složitě, za což je pochopitelně neminula záplava komentářů v jejich řešení.

Jak tedy na věc? Nejprve si z jednotlivých dvojkových cifer zadaných na vstupu vyrobíme číslo, v paměti počítače reprezentované uvnitř integeru. V prvním kroku načteme do proměnné `cislo` první cifru. V druhém kroku vynásobíme `cislo` dvojkou a přičteme druhou cifru. V třetím kroku opět vynásobíme `cislo` dvojkou a přičteme třetí cifru, a tak dále. Nyní si stačí uvědomit, že to, co na závěr zbylo v proměnné `cislo` je skutečně správně načtené číslo ze vstupu. Tomuto jednoduchému algoritmu se také říká *Hornerovo schéma.* A nepotřebovali jsme k němu žádná pomocná pole plná předpočítaných mocnin, jak se snažili tvrdit někteří řešitelé.

Nyní zbývá zjistit počet nul na konci v desítkovém zápisu. To lze udělat například tak, že budeme `cislo` neustále dělit desíti tak dlouho, dokud končí na 0 (což zjistíme pomocí operace modulo 10), a za každé vydělení si zapamatujeme jedničku. Jak dlouho to celé poběží? Použijeme při tom, v tomto případě poněkud umělý, předpoklad, že všechny základní operace s integery probíhají v konstantním čase. Zjevně potřebujeme $\mathcal{O}(N)$ kroků na načtení a zpracování N bitů ze vstupu a stejně tak $\mathcal{O}(N)$ kroků na postupné dělení desíti, protože číslo může mít nejvýše tolik desítkových cifer kolik je jich dvojkových. Dle zadání se číslo vždy vejde do integeru, paměťová spotřeba tudíž bude konstantní. Pokud bychom však chtěli být naprosto přesní, museli bychom veškerou paměťovou složitost udávat v počtu použitých bitů, a stejně tak do času započítat dobu trvání elementárních operací (sčítání, násobení) nad několikabitovými čísly. Tak se to řeší ve formální teorii složitosti. My si tím však v této úloze nebudeme lámat hlavu.

Zdálo by se, že už můžeme skončit. Lépe než $\mathcal{O}(N)$ zjevně algoritmus nenavrhneme, vstupní bity musíme alespoň načíst. Konečně, taková řešení jsem bral jako naprosto správná. To bychom však nebyli my, abychom nepředvedli nějakou vychytávku. Ukážeme si řešení, které by pracovalo rychleji, pokud bychom číslo již měli načtené (a samozřejmě za předpokladu jednotkové ceny za elementární aritmetické operace).

Nejprve budeme postupně mocnit na druhou číslo 10 tak dlouho, dokud ještě bude dělit `cislo`, tedy najdeme maximální 10^{2^i}, které dělí `cislo`. Správný počet nul na konci tedy bude ležet někde mezi 2^i a 2^{i+1}. K přesnému číslu se nyní dobereme upravenou metodou půlení intervalů. Zapamatujeme si 10^{2^i} do m. Zkusíme, je-li `cislo` dělitelné $m \cdot 10^{2^{i-1}}$. Pokud ano, nastavíme m na $m \cdot 10^{2^{i-1}}$. A testujeme znovu, tentokrát je-li `cislo` dělitelné $m \cdot 10^{2^{i-2}}$. To celé opakujeme, dokud i není 0.

První fáze bude trvat $\mathcal{O}(\log N)$ kroků, neboť, jak už jsme si řekli, desítkových cifer je nejvýše tolik, kolik je dvojkových, a postupné mocnění na druhou tedy bude trvat maximálně logaritmický čas. Druhá fáze bude opět trvat logaritmicky vzhledem k počtu bitů, protože číslo i opět může nabýt maximálně hodnoty logaritmu z N. Výsledný výpis výsledku nás opět nezdrží, protože číslo udávající počet nul může mít samo maximálně $\mathcal{O}(\log N)$ cifer. Celkem tedy čas $\mathcal{O}(\log N)$. A pokud si budeme počínat šikovně, vystačíme si pouze s konstantní pomocnou pamětí. Konečně, pro detaily viz vzorový program.

Tomáš Valla

18-4-2 Elektronické hrátky

Malý Martínek vám děkuje za došlá řešení. Moc mu pomohla. Jen byl nejprve trochu zmaten odlišným přístupem řešitelů. Někteří z vás se do problému pustili zepředu a výpočet prováděli od vstupních bitů průchodem grafu do šířky. Ostatní na to šli od lesa (tedy odzadu) a procházeli graf do hloubky od výstupních bitů. Avšak který postup je lepší? Trochu se nad oběma postupy zamysleme. Hradlová síť je zadána tak, že reference vedou směrem „zpátky“ (tedy ke vstupním bitům). Procházení od posledních (výstupních) bitů nám tedy nebude činit žádné potíže, ale opačný přístup si vyžádá předzpracování, při kterém "otočíme" reference (nebudeme si pamatovat kam jsou zapojeny vstupy jednotlivých hradel, ale zapamatujeme si, kam je zapojen výstup). Procházení odzadu je tedy výhodnější, protože nevyžaduje žádné předzpracování (byť bychom jej prováděli při načítání sítě). Procházení do hloubky od výstupu bude mít navíc tu výhodu, že spočítáme opravdu jen ta hradla, která budeme potřebovat, zatímco při procházení do šířky spočítáme úplně všechno a pak se nám může stát, že hodnoty výstupů některých hradel nebudeme vůbec potřebovat.

Náš algoritmus bude obsahovat funkci, která se zavolá na určité hradlo a spočítá jeho výsledek tak, že se rekurzivně zavolá na obě „předchozí“ hradla (to jsou hradla, na jejichž výstup jsou zapojeny moje dva vstupy) a z jejich výsledků spočítá operací NAND vlastní výsledek. Ano Ondro, vidím, že se hlásíš a chceš nám říci, že tenhle algoritmus by měl exponenciální časovou složitost. Inu ono to nebude tak docela pravda, ale každopádně nebude příliš efektivní, protože bude počítat výstupy některých hradel vícekrát. Tento problém vyřešíme jednoduše tak, že si budeme hodnoty již spočtených hradel pamatovat. Naši rekurzivní funkci pak zavoláme pro každé hradlo, jehož výstup je zapojen na výstupní bit celé sítě.

Jakou to bude mít časovou složitost? Náš algoritmus je v podstatě speciální variantou procházení grafu do hloubky (DFS), tudíž jeho složitost je $\mathcal{O}(M + N)$, kde N je počet vrcholů (v našem případě hradel) a M je počet hran. Naštěstí víme, že každé hradlo má právě 2 vstupy, tudíž počet hran bude roven dvojnásobku počtu vrcholů. Složitost pro jeden dotaz je tedy $\mathcal{O}(2N + N)$, což je totéž jako $\mathcal{O}(N)$. Graf budeme procházet pro každý dotaz úplně znovu, takže toto je časová složitost jednoho dotazu. Předzpracování si nevyžádá žádný čas a paměti spotřebujeme také pouze lineárně mnoho.

Ještě dvě poznámky ke vzorovému programu:

- Ve funkci `spocitejHradlo` a ve struktuře `THradlo` se používají celočíselné indexy. Hodnoty v rozsahu $1..N$ jsou indexy na hradla, zatím co hodnoty 0 a menší jsou odkazy na vstupní bity (tj. hodnota -2 je odkaz na 2. bit vstupu).
- Při výpočtu hodnoty hradla se používá malá optimalizace. Pokud vyjde hodnota prvního vstupu 0, pak bude výsledek vždy 1 a nemusíme počítat druhý vstup (viz. tabulka funkce NAND v zadání).

Martin „Bobřík“ Kruliš

18-4-3 Běh městem

Tajemník by byl určitě zklamaný, neboť prakticky všechna došlá řešení byla málo Kocourkovská a fungovala správně. Bohužel ne vždy bylo z řešení zřejmé, proč by tomu tak mělo být.

Město si můžeme představit jako souvislý graf, jehož vrcholy jsou křižovatky a ulice jsou hranami. Řešením pak je graf vzniklý z původního zorientováním jeho hran. Co takový graf musí splňovat?

i) Z každého vrcholu musí existovat alespoň jedna cesta vedoucí do cíle

ii) V grafu nesmí být kružnice, aby se závodník nemohl zacyklit

A jak vypadá algoritmus, který vyhovující řešení najde? Nejjednodušší je graf projít do šířky nebo do hloubky směrem od cíle a u každého vrcholu si zapamatovat, v kolikátém kroku jsme ho navštívili. Nakonec projdeme všechny hrany a zorientuje je tak, aby vedly vždy z vrcholu s vyšším pořadovým číslem do vrcholu s číslem nižším. Podmínka (i) je tak splněna, neboť každá cesta je posloupnost vrcholů s klesajícím pořadím a platí, že libovolný vrchol kromě cíle má souseda s nižším číslem. To je zřejmé z toho, jak jsme grafem procházeli. Cíl je vrchol s nejnižším pořadovým číslem, a proto v něm všechny cesty musí končit. Podmínka (ii) je rovněž splněna, neboť každá cesta obsahuje vrcholy v klesajícím pořadí a taková cesta nemůže být uzavřená.

A jak vypadá implementace? Ve skutečnosti si nemusíme žádná čísla pamatovat, stačí, když pro dva vrcholy dokážeme určit, který z nich má menší pořadové číslo. Když jsme dospěli do nějakého vrcholu a procházíme všechny hrany z něj vedoucí, pak vrchol na druhém konci buď již navštíven byl a má tak pořadové číslo jistě nižší, nebo ještě navštíven nebyl. Potom bude jeho pořadové číslo zákonitě vyšší.

Je tedy možné hrany orientovat již během prvního průchodu a dokonce jejich orientaci ihned vypisovat tak, že vypisujeme pouze nově zorientované hrany. Přičemž nově zorientované hrany vždy směřují do zpracovávaného vrcholu. Hrany z něj vycházející totiž byly zorientovány již před jeho zpracováním. A to je celé.

Časová i paměťová složitost algoritmu je zřejmě $\mathcal{O}(N+M)$, kde N je počet křižovatek a M je počet ulic. Program implementuje prohledávání do šířky a pro zpestření jsou seznamy následníků vrcholu reprezentovány spojovými seznamy, aby paměťová složitost byla opravdu $\mathcal{O}(N+M)$.

Zbyněk Falt

18-4-4 Metro pro krtky

Ukončete nástup, dveře se zavírají.

Vítejte v krtčím metru. Naši krtečci byli velmi potěšeni, protože všichni poslali funkční řešení. Bohužel u velké části z vás by krtečci už pro několik desítek tisíc krtin (což je při třiceticentimetrových krtincích těsně vedle sebe krtčí Stonehenge o průměru pouhopouhý kilometřík) značně zešedivěli. I přesto vaše řešení v $\mathcal{O}(N^2)$ získala až 6 bodů, protože krtčí správa v městečku Jablonec nad Krysou s nimi byla spokojena. Ovšem naše cíle jsou vyšší. Ano, míříme až do krtkopolis Krtečkov. A tady se s ničím menším (větším) než $\mathcal{O}(N \log N)$ nespokojí.

Ale jak jen docílit této mety nejvyšší (nejnižší)? Představme si tunely jako intervaly omezené úhlem krtin. Abychom si situaci ulehčili, budeme za levý okraj považovat menší z obou hodnot. Díky tomu si můžeme kružnici v nule přestřihnout, aniž bychom tak rozdělili nějaký interval na dva (to si rozmyslete!), a nahradit ji úsečkou od 0° do 360°. Úlohu jsme si tím změnili na nalezení počtu průniku intervalů na úsečce.

Vezměme si interval s nejmenším levým okrajem. V tomto intervalu mohou ostatní tunely začínat nebo začínat a končit, ale hlavně v něm nemohou pouze končit. Proč? Kdyby v něm nějaký interval pouze končil, tak by musel mít počáteční krtinu ještě před tímto intervalem, ale to je spor, protože ten byl vybrán právě proto, že je jeho levý okraj nejmenší.

Našim cílem je tedy spočítat, kolik tunelů v tomto intervalu začíná a zároveň nekončí. To je možné třeba tak, že spočteme, kolik jich tam začíná, a od tohoto počtu odečteme, kolik jich tam končí. To vede na pěkné kvadratické řešení. Nebo ne?

Setřiďme si krtiny podle úhlu od nejmenší. Nyní v takto setříděném poli prohlašme krtinec tunelu s menším úhlem za počáteční, naopak s větším úhlem za koncový. Pokud si počátky označíme 1 a konce -1 tak si povšimněte, že součtem čísel na určitém úseku získám, o kolik je v daném úseku více nebo méně počátků než konců. Například pokud zde začne 9 tunelů a jen 5 skončí je součet čísel na daném úseku 4. Doplňme si počet krtin na nejbližší mocninu dvojky (nulami, tím nic nezkazím) a vystavme nad nimi binární strom a to tak, že v každém uzlu bude součet jeho levého a pravého syna. Jen tak mimochodem si uvědomte, že v kořeni tohoto stromu musí být 0. Navíc si pro každý začátek tunelu pamatujme, kde je umístěn jeho konec. V takovémto stromu je jednoduché jedním průchodem z listu, ve kterém je umístěn konec tunelu, do kořenu najít součet prvků nalevo nebo napravo od něj.

Tím jsme vyřešili první tunel, ale co ten druhý? Třetí? Chtělo by to první odebrat. Ale to je snadné, na umístění jeho krtin dáme nuly a strom opět, tentokrát dvěma průchody (za každou krtinu jeden) opravím. Obě tyto operace mají složitost $\mathcal{O}(\log N)$. Tím se druhý tunel stane prvním a vše může pokračovat.

Teď ještě pár slov k těmto dvěma operacím. Začneme opravením. Nejprve změním daný list na nulu. Pak skočím do jeho otce a přiřadím mu součet jeho synů. Tím se opět změnila jeho hodnota a tak opět skočím o úroveň výš. To budu opakovat dokud nedojdu do kořene. Trochu složitější se mi zdá zjištění počtu počátečních a koncových krtin. Představte si nyní nějakou (pokud možno hodně klikatou) cestu z listu ke kořeni. To co my chceme získat, je součet všech listů nalevo od našeho. Teď se posuňme o úroveň výš. Mohly nastat dva případy

i) do otce jsme přišli zleva. Tak nic neřešíme, protože pravý list obsahuje jistě součet (nějakých) listů napravo od našeho.

ii) do otce jsme přišli zprava. To je mnohem zajímavější, v tom případě navýšíme počítadlo o hodnotu levého syna, protože je to součet úseku vlevo od našeho listu. Pokud si navíc nakreslíte obrázek, tak krásně uvidíte, že tento postup skutečně pokryje celý interval vlevo od našeho vrcholu.

Výsledek v počitadle získáme v $\mathcal{O}(\log N)$, protože v každém kroku skočíme o úroveň výš, ve stejném čase také opravíme strom. Protože tyto operace provedeme $N/2$-krát (tolik je

počátečních krtin), je celková složitost $\mathcal{O}(N \log N)$. Paměti potřebujeme na stromeček a umístění konců pouze lineárně mnoho, takže $\mathcal{O}(N)$. Samozřejmě časová složitost je vzhledem k použití QuickSortu jen průměrná, ale použitím jiného sortu bychom si ji zajistili.

Ukončete výstup, dveře se zavírají.

Jan Bulánek

18-4-5 Datakopci

Nejprve bylo dobré povšimnout si, že protože každá zpráva bude doručena (to plyne přímo ze zadání – každý dopravník buď vede přímo do redakce nebo pokračuje pásem na sousedním políčku, který vede stejným směrem), stačí nám pamatovat si rozdíl počtu dobrých a špatných zpráv s tím, že znaménka množství doručených na severní okraj obrátíme. Také nám hodně pomůže fakt, že hranice mezi S-J a Z-V pásy bude lomená čára začínající v severozápadním (levém horním) rohu, vedoucí pouze na jih a východ a končící v jihovýchodním (pravém dolním) rohu. Jižně a západně od ní leží dopravníky ženoucí se na západ do redakce pohádek, na sever a západ ty vedoucí na sever do redakce zpravodajství. Proč nemůže mít i nějaký ten záhyb na sever či západ? Zprávy z něj by se už určitě nedostaly a to je zakázáno. Zkoušet všechny tyto dělící cesty není dobrý nápad, je jich totiž $n^n/n!$, ačkoliv by to byla teoretická možnost.

My úlohu vyřešíme lépe a to dynamickým programováním. Jakkoli hrozně vám některým může tento pojem znít, jde vlastně jen o to, že si pro každý řádek pro všechna $k = 0 \ldots n$ postupně spočtu, jaký nejlepší výsledek mohu dostat, pokud v něm vede právě k dopravníků na západ a zbylé na sever. Pro první řádek to zjistíme snadno a pro $j+1$-ní to závisí pouze na optimech pro j-tý řádek – když potřebuji zjistit optimum pro situaci, kdy k dopravníků vede na západ, vyzkouším to zkombinovat se situacemi, kdy dělící cesta pokračuje o řádek výše v pozici $0, 1, 2, \ldots, k$. Toto stihnu mnohem rychleji – kombinací k_{j+1} s možnými k_j je $O(n^2)$, každý z přepočtů mi bude trvat $O(n)$ a toto udělám pro $O(m)$ řádků, celkem tedy $O(n^3 m)$ nebo $O(m^3 n)$ pokud bych se místo po řádcích rozhodl jít po sloupcích, což by byla stejná jen zrotovaná úvaha.

Zkusme to ale zlepšit. Toho, že toto spočteme pro $O(m)$ řádků se nezbavíme, zrychlíme ale jednotlivé řádky. První zrychlení dosáhneme předpočítáním částečných součtů zleva na jednotlivých řádcích (při jeho rozdělení v jednotlivých místech by nám řádek vynesl $\sum_{i=1}^{k} d_i - \sum_{i=k+1}^{n} d_i = 2\sum_{i=1}^{k} d_i - \sum_{i=1}^{n} d_i = 2s_k - konst.$, konstantu nezávisející na místě rozdělení mohu ignorovat a počítat jen s $2s_k$ a to klidně polovičním. Tímto si trochu zjednoduším počítání s_k), když už si totiž vybereme konkrétní k_j a k_{j+1}, je nové optimum jen součtem tohoto částečného součtu $s_{k_{j+1}}$ a optima na k_j v minulém řádku. Tímto jsme dosáhli $O(n^2 m)$ (předpočítat částečný součet umíme v čase $O(n)$ na řádek).

Další a poslední zlepšení: při zkoušení optimálního pokračování dělící cesty o řádek výše vždy vybereme maximum z možných mezivýsledků. Toto nalezené maximum pro k_{j+1} si ale můžeme schovat pro $k_{j+1}+1$ (další políčko na řádce) a jen ho třeba zlepšit na $s_{k_{j+1}}$, pokud je větší. Tento malý trik nám zlepší složitost až na $O(mn)$. Nejen, že už nezáleží na tom, zda jdeme po sloupcích či řádcích, navíc jsme určitě dosáhli optima – na každou hodnotu musíme v libovolném fungujícím algoritmu sáhnout alespoň jednou, mohlo by se tam ukrývat nějaké velmi velké množství pohádek, o které přece nechceme přijít...

Paměťová složitost je $O(mn)$.

Tomáš Gavenčiak

18-4-6 Kompilátorový φgl

Jedním z možných řešení této úlohy bylo použít algoritmus popsaný v řešení úlohy 18-3-6 a modifikovat ho pro práci nad SSA formou. Existuje ovšem přímočařejší řešení – vyjdeme z definice živého kódu.

Výraz je živý, pokud má nějaké vedlejší efekty, nebo pokud je jeho hodnota použita v živém výrazu. Představme si následující orientovaný graf: jeho vrcholy budou příkazy v programu. V grafu bude hrana z příkazu p_1 do příkazu p_2, pokud p_1 používá hodnotu vypočtenou v příkazu p_2, tedy pokud p_2 je buď přiřazení `x = expr`, nebo `x = phi(...)`, a příkaz p_2 používá proměnnou x. Vrcholy odpovídající příkazům s vedlejšími efekty si označíme. Pak příkaz je živý právě tehdy, pokud do jemu odpovídajícímu vrcholu vede cesta z některého z označených vrcholů. Toto je ovšem dobře známá úloha na dosažitelnost v grafu, kterou můžeme vyřešit například procházením grafu do hloubky (viz třeba kuchařku z druhé série). Algoritmus tedy může vypadat zhruba takto:

pro každý příkaz p:
  if p má vedlejší efekty **then**
    označ p a ulož ho na zásobník

while zásobník není prázdný **do**
  odeber ze zásobníku příkaz (p)
  pro všechny proměnné x použité v p:
    buď q příkaz, který definuje x
    if q není označený **then**
      označ q a přidej ho do zásobníku

pro každý příkaz p:
  if p není označený **then**
    smaž p

V algoritmu využíváme toho, že x je definováno jen jedním příkazem, a tedy nemusíme zjišťovat, které z definic odpovídají danému použití. Pro každý příkaz v programu si musíme pamatovat značku a mít pro něj místo na zásobníku, tedy paměťová složitost je lineární. Každý příkaz se dostane do zásobníku nanejvýš jednou, proto je časová složitost také lineární.

Zdeněk Dvořák

Úloha 18-3-4 – Pochoutka pro prasátko – program

```
/* KSP 18-3-4 řešení dle Programátorské kuchařky -- Dijkstrův algoritmus s haldou */

#include <stdio.h>

#define MaxN 100
#define MaxM 100
#define MaxK 50

typedef struct policko_typ {
  int x,y,vzdal,je_docas,predx,predy; // souřadnice, vzdálenost, zda je dočasné, předchůdce
}policko;

typedef struct pravidlo_typ {         // možné pohyby čuněte
  int x,y,namaha;                     // posun o x, posun o y, vynaložená námaha
}pravidlo;

typedef struct uzel_typ {             // uzel v haldě
  int vzdal,x,y;                      // dočasná vzdálenost, souřadnice
}uzel;

int N,M,K;                            // rozměry lesa, počet pravidel
pravidlo pravidla[MaxK];              // všechny pohyby čuněte
uzel halda[MaxN*MaxM*MaxK];           // halda na políčka
int halda_len;                        // velikost haldy
policko les[MaxN][MaxM];              // dvojrozměrný prostor = les

void halda_vloz(int x, int y, int vzdal) {
  int i;
  uzel pom;

  printf("Davam na haldu [%d, %d]: %d\n",x,y,vzdal);
  i = halda_len;
  halda_len++;
  halda[i].x = x; halda[i].y = y; halda[i].vzdal = vzdal;
  while (i > 1 && halda[i/2].vzdal > halda[i].vzdal) {
    pom = halda[i/2];
    halda[i/2] = halda[i];
    halda[i] = pom;
    i = i/2;
  }
}

void halda_odeber_min() {
  int i, j;
  uzel pom;
  halda[1] = halda[halda_len];
  halda_len--;
  i = 1;
  while (2*i<=halda_len) {
    j = i;
    if (halda[j].vzdal > halda[2*i].vzdal) j = 2*i;
    if (2*i+1 <= halda_len && halda[j].vzdal > halda[2*i+1].vzdal) j = 2*i+1;
    if (i == j) break;
    pom = halda[i]; halda[i] = halda[j]; halda[j] = pom;
    i = j;
  }
}

void dijkstra(int start_x, int start_y, int cil_x, int cil_y) {
  int vx, vy, nx, ny, i, j;

  for (i = 0; i < N; i++) {           // inicializace
    for (j = 0; j < M; j++) {
      les[i][j].je_docas = 1;
      les[i][j].vzdal = -1;
      les[i][j].predx = les[i][j].predy = -1;
    }
  }
  les[start_x][start_y].vzdal = 0;    // vzdálenost start-start je 0
  halda_vloz(start_x, start_y, 0);    // start na haldu

  while (halda_len > 0) {             // dokud není halda prázdná
    vx = halda[0].x;                  // vybereme políčko s nejmenší dočasnou vzdáleností
    vy = halda[0].y;
    printf("Odebiram z haldy [%d, %d]: %d\n",vx, vy, halda[0].vzdal);
    halda_odeber_min();               // odebereme políčko z haldy
    if (vx == cil_x && vy == cil_y) break;    // bukvice nalezena, konec
    if (!les[vx][vy].je_docas) {
```

```
      printf("Tady uz jsme jednou byli.\n");
      continue;     // tady uz jsme byli, jdeme dál
    }
    les[vx][vy].je_docas = 0;                // políčko prohlásíme za trvalé

    for (i = 0; i < K; i++) {
      nx = pravidla[i].x + vx;               // kam se odsud umíme dostat s pravidlem i
      ny = pravidla[i].y + vy;
      printf("Zkousim nove policko [%d, %d]\n",nx,ny);
      if (nx < N && ny < M && nx >=0 && ny >= 0)  // nevyběhli jsme z lesa
        if ((les[nx][ny].vzdal == -1) ||
            (les[nx][ny].vzdal > les[vx][vy].vzdal + pravidla[i].namaha)) {
          les[nx][ny].vzdal = les[vx][vy].vzdal + pravidla[i].namaha;   // nová vzdálenost
          les[nx][ny].predx = vx;                                       // nový předek
          les[nx][ny].predy = vy;
          halda_vloz(nx,ny,les[nx][ny].vzdal);                          // vlož na haldu
        }
    }
  }
}

int main() {
  int prase_x, prase_y, bukv_x, bukv_y;
  int nx, ny, i;

  printf("Rozměry lesa: "); scanf ("%d %d", &N, &M);
  printf("Pozice prasete: "); scanf("%d %d", &prase_x, &prase_y);
  printf("Pozice bukvice: "); scanf("%d %d", &bukv_x, &bukv_y);
  printf("Počet pravidel pohybu prasete: "); scanf("%d", &K);
  for (i = 0; i < K; i++) scanf("%d %d %d",&pravidla[i].x,&pravidla[i].y,&pravidla[i].namaha);

  dijkstra(prase_x, prase_y, bukv_x, bukv_y);

  if (les[bukv_x][bukv_y].vzdal == -1) printf("Cesta neexistuje.\n");
  else {
    printf("Nejkratší cesta pozpátku: \n");
    nx = bukv_x; ny = bukv_y;
    while (nx != -1 && ny != -1) {
      printf("[%d, %d]\n",nx,ny);
      nx = les[nx][ny].predx; ny = les[nx][ny].predy;
    }
  }
  return 0;
}
```

Úloha 18-4-1 – HP – program

```
program hp;
var
  cislo, m, pocet, s, sp: longint;
  c: char;
begin
   cislo:= 0;    {načti číslo ukončené tečkou Hornerovým schématem}
   c:= '0';
   while c<>'.' do begin
      cislo:= 2*cislo + ord(c)-ord('0');
      read(c);
   end;

   m:= 10;     {zjisti nejbližší nižší 10^{2^i} které dělí číslo}
   if cislo mod 10 = 0 then pocet:= 1
   else pocet:= 0;
   while cislo mod (m*m) = 0 do begin
      m:= m * m;
      pocet:= 2*pocet;
   end;

   {"půlením intervalů" mezi 10^(2^(i+1)) a 10^(2^i) zjisti přesný počet nul}
   s:= round(sqrt(m));
   sp:= pocet div 2;
   while sp > 0 do begin
      if cislo mod (m*s) = 0 then begin
         m:= m*s;
         pocet:= pocet+sp;
      end;
      s:= round(sqrt(s));
      sp:= sp div 2;
   end;
   writeln('počet nul je ', pocet);
end.
```

Úloha 18-4-2 – Elektronické hrátky – program

```
const MAX = 1000;       { maximalní počet hradel ... to není důležité }

type
        THradlo = record
                in1, in2: Integer;              { indexy na vstupní hradla (nebo bity) }
                done: Boolean;                  { je true, pokud bylo hradlo již spočteno }
                val: Boolean;                   { spočtená hodnota (pouze pokud done = true) }
        end;

var
        PocetHradel, PocetVstupu, PocetVystupu: Integer;
        Hradla: array[1..MAX] of THradlo;       { pole všech hradel }
        Vystupy: array[0..MAX] of Integer;      { indexy hradel, zapojených na výstupy }
        Vstupy: array[0..MAX] of Boolean;       { hodnoty vstupních bitů }


{ rekurzivní funkce, která určí výstupní hodnotu hradla }
{ Pozn: Budu zde sahat na glob. proměnné (Hradla, Vstupy a Pocty) }
function spocitejHradlo(index: Integer): Boolean;
var
        val_tmp: Boolean;
begin
        if (index < 1) then begin       { index neukazuje na hradlo, ale na vstup. bit }
                spocitejHradlo := Vstupy[-index];
                Exit;
        end;

        { jen pro jistotu - jestli se ptáme na existující hradlo }
        spocitejHradlo := false;
        if ((index < 1) or (index > PocetHradel)) then Exit;

        if (not Hradla[index].done) then begin  { hradlo ještě nebylo spočteno }
                Hradla[index].done = true;                      { tak ho spočítáme }
                val_tmp := spocitejHradlo(in1);
                if (val_tmp) then
                        Hradla[index].val := not (val1 and spocitejHradlo(in2))
                else            { optimalizace - nemusíme počítat dál}
                        Hradla[index].val = true;       { 0 NAND s čímkoli je vždy true }
        end;

        spocitejHradlo := Hradla[index].val;    { sáhnem pro výsledek do cache }
end;


{ Připraví hradla pro další dotaz }
procedure vycistiHradla;
var
        i: Integer;
begin
        for i := 1 to PocetHradel do Hradla[i].done := False;
end;


{ Spočítá všechna výstupní hradla a vytiskne spočtené hodnoty }
procedure spocitejHradla;
var
        i: Integer;
begin
        for i := 0 to PocetVystupu-1 do
                Writeln("bit ", i, ".: ", spocitejHradlo(Vystupy[i]) );
end;


procedure nactiVstup;
{ načte ze vstupu hodnoty do pole Vstupy - implentaci vynechávám }

procedure nactiHradla;
{ načte ze vstupu hodnoty do pole Hradla a Vystupy a konstanty Pocet... }


begin
        nactiHradla;
        ...
        { při každém dotazu se zavolá posloupnost }
        vycistiHradla;
        nactiVstup;
        spocitejHradla;
end.
```

Úloha 18-4-3 – Běh městem – program

```
program BehMestem;
const MAXN = 512;

type Uk = ^tUzel;
     tUzel = record
       info : integer;
       dalsi : Uk;
     end;

     tVrchol = record
       navstiven : boolean;
       naslednici : Uk;
       stupen : integer;
     end;

var FIFO : array [0..MAXN] of integer;
    zac,kon : integer;
    vrchol : array [1..MAXN] of tVrchol;
    v1,v2,i,j,x,N,M,cil : integer;
    P : Uk;

begin
  readln(N,M,cil);

  for x:=1 to N do begin
    vrchol[x].naslednici:=nil;
    vrchol[x].stupen:=0;
    vrchol[x].navstiven:=false;
  end;

  for x:=1 to M do begin
    readln(i,j);

    new(P);                         { vytváření seznamu následníků }
    P^.info:=i;
    P^.dalsi:=vrchol[j].naslednici;
    vrchol[j].naslednici:=P;

    new(P);
    P^.info:=j;
    P^.dalsi:=vrchol[i].naslednici;
    vrchol[i].naslednici:=P;
  end;

  FIFO[0]:=cil;                     { začínáme od cíle }
  zac:=0;
  kon:=1;

  while zac<kon do begin            { dokud je co zpracovávat }
    v1:=FIFO[zac];
    inc(zac);

    vrchol[v1].navstiven:=true;

    P:=vrchol[v1].naslednici;
    while P<>nil do begin           { projdeme všechny sousedy }
      v2:=P^.info;
      P:=P^.dalsi;

      if not vrchol[v2].navstiven then begin
        FIFO[kon]:=v2;
        inc(kon);
        writeln(v2,' -> ',v1);      { nově zorientované hrany vypisujeme }
      end;
    end;
  end;
end.
```

Úloha 18-4-4 – Metro pro krtky – program

```
program serie1644rel;
const MAX=1024;

type node=record { uzel ve stromě }
        id:integer;
        val:real;
     end;

var tree:array[1..MAX] of node;
    place:array[1..MAX div 2] of integer;{ umístění konců úseček }
    N,N2,i,poc_prusec:integer;

procedure sort(a,b:integer); { QuickSort }
var i,j:integer;
    pivot:real;
    pom:node;
begin
    i:=a;j:=b;pivot:=tree[(a+b) div 2].val;
    while(i<=j) do begin
        while(tree[i].val<pivot)do Inc(i);
        while(tree[j].val>pivot) do Dec(j);
        if (i<j) then begin
            pom:=tree[i];tree[i]:=tree[j];tree[j]:=pom;
            Inc(i);Dec(j);
        end else if (i=j) then begin
            Inc(i);Dec(j);
        end;
    end;
    if (a<j) then sort(a,j);
    if (b>i) then sort(i,b);
end;

procedure repair(i:integer); { opraví strom z daného listu }
begin
    while(i>1) do begin
        i:=i div 2;
        tree[i].val:=tree[2*i].val+tree[2*i+1].val;
    end;
end;

function value(i:integer):integer;{ vrací počet otevřených cest od začátku k i }
var pom:real;
begin
    pom:=tree[i].val;{ krajní bod tam patří také }
    while (i>1) do begin
        if (i mod 2 = 1) then pom:=pom+tree[i-1].val;
        { je-li pravý syn, tak hodnota levého značí kolik konců nebo
        začátků leží uvnitř intervalu, tím vlastně získáme počet začátků,
        které nejsou uzavřeny. Protože začátků bude alespoň tolik, kolik
        konců, vyjde kladné číslo }
        i:=i div 2;
    end;
    value:=round(pom);
end;

begin
    writeln('Zadej pocet krtin');readln(fin,N);
    N2:=1;
    while N2<N do N2:=2*N2; { nalezení nejbližší vyšší mocniny 2 }

    for i:= 1 to N div 2 do begin { načtení cest }
        writeln('Zadej 1. krtinu cesty');readln(tree[N2].val);
        { načítám rovnou do listu stromu }
        writeln('Zadej 2. krtinu cesty');readln(tree[N2+1].val);
        tree[N2].id:=i; { uložení indexu cesty }
        tree[N2+1].id:=i;
        Inc(N2,2);
    end;

    N2:=N2-N;{ návrat na první list }
    sort(N2,2*N2-1);

    for i:=2*N2-1 downto N2 do begin
        if (tree[i].id<>0) then begin
            if (place[tree[i].id]=0) then begin { je to začátek tunelu }
                tree[i].val:=-1;{ jeho hodnota je nám už k ničemu, teď značí konec }
                place[tree[i].id]:=i;{ kde je konec }
            end else
```

```
                tree[i].val:=1; { začátek tunelu }
        end;
    end;

    for i:= N2-1 downto 1 do { naplnění stromu }
        tree[i].val:=tree[2*i].val+tree[2*i+1].val;

    for i:=N2 to 2*N2-1 do begin
        if tree[i].id<>0 then begin
            poc_prusec:=poc_prusec+value(place[tree[i].id]);
            tree[place[tree[i].id]].id:=0;{  }
            tree[place[tree[i].id]].val:=0;{ neutralní hodnota nic neovlivní }
            repair(place[tree[i].id]);{ opravi strom v místě konce }
            tree[i].val:=0;
            repair(i);{ oprava v místě začátku tunelu }
        end;
    end;
    writeln('Bude treba ',poc_prusec,' zizal.');
end.
```

Úloha 18-4-5 – Datakopci – program

```
#include <stdio.h>

//VSTUP:
#define M 4
#define N 4
int dobre[M][N], spatne[M][N];

// nultý sloupec je vždy 0 (pro usnadnění)
int d[M][N+1]; // rozdíly počtu dobrých a špatných
int s[M][N+1]; // částečné součty
int o[M][N+1]; // optima

int k[M][N+1]; // pro zjištění tvaru výsledné dělící cesty si pamatuji,
               // kde pokračovala o řádek výše pro všechna její pokračování tady
int nazapad[M+1]; // kolik cest nakonec kde vede na Z

int main() {
        int i,j,max;
        //VSTUP - neřeším
        for(i=0;i<M;i++) {                              //spočtu rozdíly a částečné součty
                d[i][0]=s[i][0]=o[i][0]=k[i][0]=0;
                for(j=1;j<=N;j++) {
                        d[i][j]=dobre[i][j-1]-spatne[i][j-1];
                        s[i][j]=s[i][j-1]+d[i][j];
                }
        }

        for(j=0;j<=N;j++) o[0][j]=s[0][j];              // první řádek
        for(i=1;i<M;i++) {                              // a ty další
                max=0;                                  // pozice maxima
                for(j=1;j<=N;j++) {
                        if(o[i-1][max]<o[i-1][j]) max=j;
                        o[i][j]=s[i][j]+o[i-1][max];
                        k[i][j]=max;
                }
        }

        max=0;
        for(j=1;j<=N;j++) if(o[M-1][max]<o[M-1][j]) max=j;
        // nyní mám v max optimální konec cesty, z něj, zpětně zjistím, kudy optimálně vedla
        nazapad[M-1]=max;
        for(i=M-2;i>=0;i--) nazapad[i]=k[i+1][nazapad[i+1]];

        max=0;                                          // a ještě výpis
        for(i=0;i<M;i++) {
                for(j=0;j<nazapad[i];j++) {
                        max+=dobre[i][j];
                        printf("-");
                }
                for(;j<N;j++){
                        max+=spatne[i][j];
                        printf("|");
                }
                printf("\n");
        }
        printf("Vydoluje se %d zpráv\n",max);
        return 0;
}
```

Výsledková listina osmnáctého ročníku KSP po třetí sérii

		škola	*ročník*	*sérií*	*1831*	*1832*	*1833*	*1834*	*1835*	*1836*	*suma*	*celkem*
1.	Josef Pihera	G Strakon	3	8			9	12	15	10	46,3	126,6
2.	Peter Perešíni	GJGTajov	4	11		5	10		14		29,0	119,4
3.	Pavel Klavík	G Chrudim	3	12	5		10	10	12	10	41,9	112,5
4.	Miroslav Klimoš	G Bílovec	1	12	4		9	10	13		35,8	110,7
5.	Jakub Kaplan	GJKTyla	2	8	5		6	9	13		34,3	98,9
6.	Roman Smrž	GOhradní	2	8	4		10				14,3	88,1
7.	Zbyněk Konečný	GKptJaroš	3	10	5	5	10	2			22,2	85,5
8.	Jiří Maršík	GJKTyla	2	3	5		5	5	4		27,2	85,2
9.	Lukáš Lánský	GJKTyla	2	8	3		3	2	9		20,2	79,7
10.	Petr Onderka	G VKlobou	3	3	5		3	2			14,4	73,2
11.	Michal Pavelčík	G UBrod	3	6			8				8,9	71,9
12.	Petr Kratochvíl	G SvětláNS	3	12	4	5	10	4			22,6	66,2
13.	Michal Čudrnák	G Holešov	4	2							0,0	64,1
14.	Michal Vaner	G Turnov	4	4	5			6			13,0	63,7
15.	Tomáš Zámečník	GJKeplera	3	3	4		3				10,1	62,5
16.	Adam Zivner	G UBrod	4	8			5	6			13,0	61,0
17.	Jan Kohout	G Roudnice	3	3	4	2		2	8		22,8	52,8
18.	Tomáš Herceg	G Třebíč	3	9	4			3	10		18,4	51,6
19.	Kristýna Krejčová	G Tišnov	3	2	5		4		2		16,3	48,4
20.	Josef Špak	GJírovco	3	4	4	1	3				11,6	41,5
21.	Drahoslav Viktorýn	G UBrod	3	2	4	1		3			12,5	40,0
22.	Radim Pechal	SPŠ Rožnov	3	2							0,0	39,9
23.	Jan Hrnčíř	GFXŠaldy	4	12	5		7				11,7	39,3
24.	Daniel Marek	GZborov	4	6							0,0	38,0
25.	Pavel Veselý	G Strakon	1	2							0,0	37,4
26.	Kateřina Böhmová	G Rožnov	4	2							0,0	36,2
27.	Cyril Hrubiš	G Bílovec	4	9	3	2					5,6	36,0
28.	Jiří Machálek	G Holešov	4	4	5						5,0	35,5
29.	Ondřej Bílka	G Zlín	4	11	1	0	8	1	7		17,0	33,7
30. – 31.	Richard Jedlička	G Vlašim	2	3	3						4,1	31,0
	Tereza Klimošová	G Lanškr	4	3							0,0	31,0
32.	Jakub Pavlík jn.	G Kladno	3	3	3						4,1	28,1
33.	Vojtěch Molda	G Vsetín	4	1							0,0	26,9
34.	Ondrej Mikuláš	G Lučenec	3	2	5	5	6				18,4	25,7
35.	Ondřej Bouda	GKptJaroš	3	4	5		5				12,2	25,3
36.	Petr Trňák	G UHradi	3	3	2						3,3	24,3
37.	Ján Mikuláš	G Lučenec	4	1							0,0	21,7
38.	Martin Kahoun	GJNerudy	3	4	4			2	1		10,4	21,6
39.	Adam Ráž	GBudějo	3	5	5	5		2			13,5	21,4
40.	Martin Majer	SPŠÚžlabin	1	1	3		3	6			18,9	18,9
41.	Jiří Cabal	SPŠ DvKrál	3	2							0,0	15,1
42.	Rudolf Rosa	G Kladno	3	2							0,0	15,0
43.	Matej Kollár	G PBystric	4	2	4						4,7	14,8
44.	Lukáš Moravec	GSRandyJN	2	1							0,0	12,7
45.	David Škorvaga	G Kralupy	3	1							0,0	12,4
46.	Radim Cajzl	G NMnMor	0	3	4						4,7	10,4
47.	Tomáš Ehrlich	G Holešov	3	3							0,0	9,8
48.	Tomáš Sýkora	G VKlobou	2	2	2	1					5,5	9,0
49.	Marián Bazálik	G Košice	4	1							0,0	8,3
50.	Jan Musílek	G NBydžov	2	1							0,0	7,3
51.	Jan Tichý	GDašická	1	1							0,0	6,6
52.	Jiří Václavík	G Dobříš	4	2	3						4,2	6,5
53.	Vladimír Munzar	SPŠ Rožnov	1	1							0,0	5,8
54. – 58.	Jakub Balhar	GJNerudy	3	1							0,0	4,7
	Martin Fojtík	GSRandyJN	2	1							0,0	4,7
	Jan Krajdl	SPŠÚžlabin	1	1	4						4,7	4,7
	Dušan Rychnovský	G Hranice	2	1	4						4,7	4,7
	Radek Svoboda	G Roudnice	3	1	4						4,7	4,7
59.	Robert Brunetto	SPŠMasaryk	2	1	3						4,3	4,3
60.	Jiří Keresteš	ZŠKostelní	0	2	3						4,2	4,2
61. – 62.	Miroslav Jančařík	G UBrod	2	1							0,0	3,5
	Jakub Loucký	G Písek	3	1							0,0	3,5

Výsledková listina osmnáctého ročníku KSP po čtvrté sérii

		škola	*ročník*	*sérií*	*1841*	*1842*	*1843*	*1844*	*1845*	*1846*	*suma*	*celkem*
1.	Josef Pihera	G Strakon	3	9			9	10	15	10	44,2	170,8
2.	Miroslav Klimoš	G Bílovec	1	13		9	9	10	15		42,8	153,5
3.	Pavel Klavík	G Chrudim	3	13	1	9	9	10		9	36,4	148,9
4.	Jakub Kaplan	GJKTyla	2	9	5	10	9	4			28,7	127,6
5.	Jiří Maršík	GJKTyla	2	4	5	10	9	6	5		35,3	120,5
6.	Peter Perešíni	GJGTajov	4	11							0,0	119,4
7.	Zbyněk Konečný	GKptJaroš	3	11	5	9	9	6	8		32,1	117,6
8.	Petr Onderka	G VKlobou	3	4		10		6			18,0	91,2
9.	Michal Pavelčík	G UBrod	3	7		10	8				18,4	90,3
10.	Petr Kratochvíl	G SvětláNS	3	13		9	7	8			23,0	89,2
11.	Roman Smrž	GOhradní	2	8							0,0	88,1
12.	Adam Zivner	G UBrod	4	9		9	8	6			24,1	85,1
13.	Lukáš Lánský	GJKTyla	2	8							0,0	79,7
14.	Jan Kohout	G Roudnice	3	4	4		7	6			20,8	73,6
15.	Michal Vaner	G Turnov	4	5			9				9,0	72,7
16.	Tomáš Herceg	G Třebíč	3	10	4		5	6			15,7	67,3
17.	Michal Čudrnák	G Holešov	4	2							0,0	64,1
18.	Tomáš Zámečník	GJKeplera	3	3							0,0	62,5
19.	Drahoslav Viktorýn	G UBrod	3	3	4		8	6			21,6	61,6
20.	Richard Jedlička	G Vlašim	2	4		10		6			18,0	49,0
21.	Kristýna Krejčová	G Tišnov	3	2							0,0	48,4
22.	Daniel Marek	GZborov	4	7			9				9,0	47,0
23.	Ondřej Bílka	G Zlín	4	12	0	1	3	7		0	10,4	44,1
24.	Josef Špak	GJírovco	3	4							0,0	41,5
25.	Radim Pechal	SPŠ Rožnov	3	2							0,0	39,9
26.	Jan Hrnčíř	GFXŠaldy	4	12							0,0	39,3
27.	Ondřej Bouda	GKptJaroš	3	5	5	6					12,8	38,1
28.	Pavel Veselý	G Strakon	1	2							0,0	37,4
29.	Kateřina Böhmová	G Rožnov	4	2							0,0	36,2
30.	Cyril Hrubiš	G Bílovec	4	9							0,0	36,0
31.	Jiří Machálek	G Holešov	4	4							0,0	35,5
32.	Tereza Klimošová	G Lanškr	4	3							0,0	31,0
33.	Jakub Pavlík jn.	G Kladno	3	3							0,0	28,1
34.	Vojtěch Molda	G Vsetín	4	1							0,0	26,9
35.	Martin Kahoun	GJNerudy	3	5	4						4,5	26,1
36.	Ondrej Mikuláš	G Lučenec	3	2							0,0	25,7
37.	Petr Trňák	G UHradi	3	3							0,0	24,3
38.	Martin Majer	SPŠÚžlabin	1	2	5						5,0	23,9
39.	Radim Cajzl	G NMnMor	0	4	3			6			12,0	22,4
40.	Ján Mikuláš	G Lučenec	4	1							0,0	21,7
41.	Adam Ráž	GBudějo	3	5							0,0	21,4
42.	Jan Krajdl	SPŠÚžlabin	1	2	3			6			12,6	17,3
43.	Jiří Cabal	SPŠ DvKrál	3	2							0,0	15,1
44.	Rudolf Rosa	G Kladno	3	2							0,0	15,0
45.	Matej Kollár	G PBystric	4	2							0,0	14,8
46.	Jan Dvořák	GZborov	3	1	5			5			12,9	12,9
47.	Lukáš Moravec	GSRandyJN	2	1							0,0	12,7
48.	David Škorvaga	G Kralupy	3	1							0,0	12,4
49.	Tomáš Ehrlich	G Holešov	3	3							0,0	9,8
50.	Tomáš Sýkora	G VKlobou	2	2							0,0	9,0
51.	Jakub Balhar	GJNerudy	3	2	3						4,2	8,9
52.	Marián Bazálik	G Košice	4	1							0,0	8,3
53.	Jan Musílek	G NBydžov	2	1							0,0	7,3
54.	Jan Tichý	GDašická	1	1							0,0	6,6
55.	Jiří Václavík	G Dobříš	4	2							0,0	6,5
56.	Vladimír Munzar	SPŠ Rožnov	1	1							0,0	5,8
57. – 59.	Martin Fojtík	GSRandyJN	2	1							0,0	4,7
	Dušan Rychnovský	G Hranice	2	1							0,0	4,7
	Radek Svoboda	G Roudnice	3	1							0,0	4,7
60.	Robert Brunetto	SPŠMasaryk	2	1							0,0	4,3
61.	Jiří Keresteš	ZŠKostelní	0	2							0,0	4,2
62. – 63.	Miroslav Jančařík	G UBrod	2	1							0,0	3,5
	Jakub Loucký	G Písek	3	1							0,0	3,5