Korespondenční Seminář z Programování

33. ročník | **KSP** | **Červen 2021**

Vzorová řešení páté série třicátého třetího ročníku KSP

33-5-1 Šotek a obrazy

Šotek nám zpřeházel obrazy očíslované od 1 do N podle nějakého polynomu pátého stupně, který máme zadaný na vstupu. Jak je seřadit zpátky? Zkoušet invertovat polynom by bylo složité, ale to naštěstí nepotřebujeme. Nám stačí *jenom* napárovat N funkčních hodnot polynomu na čísla od 1 do N.

Vyhodnotit polynom pátého stupně zvládneme v konstantním čase (je to jen několik násobení a sčítání), takže si můžeme v čase $\mathcal{O}(N)$ zkonstruovat následující posloupnost dvojic:

$$(1, f(1)), (2, f(2)), (3, f(3)), \ldots, (N, f(N)).$$

Pokud by se nám povedlo tuto posloupnost seřadit podle funkčních hodnot od nejmenší po největší, tak pak budou dvojice přesně odpovídat pořadí obrazů (protože šotek je také seřadil podle funkčních hodnot). Na rozdíl od přeházených obrazů však budeme mít jednu informaci navíc – číslo z rozsahu 1 až N udávající původní pořadí obrazu. Takže pokud se nám podaří dvojice seřadit podle funkčních hodnot, tak pak stačí jen projít přes všechny dvojice a vydat informaci o původním pořadí obrazu.

Jak ale seřadit funkční hodnoty rychle? Běžné třídící algoritmy využívající jen vzájemné porovnávání prvků nemohou být rychlejší než $\mathcal{O}(N \log N)$, za což dle zadání plné body nebudou, co s tím? Budeme potřebovat využít nějaké speciální vlastnosti čísel a použít algoritmus, který nedělá jen porovnávání dvojic.

Přihrádkové třídění

První, co by nás mohlo napadnout, je přihrádkové třídění neboli Radix sort (ten je speciální případ Bucket sortu). To využívá toho, že jsou čísla ke třídění jen z nějakého omezeného rozsahu a dá se jimi indexovat pole. Vyrobíme si přihrádky pro každé číslo z tohoto rozsahu a pak čísla procházíme a rozhazujeme je do jednotlivých přihrádek (což mohou být třeba spojové seznamy). Na konci jen projdeme všechny přihrádky a „slepíme“ je za sebe.

Jednoúrovňové použití pro nás nedává úplně smysl, ale pravá síla přihrádkového třídění přichází tehdy, když ho použijeme víceúrovňově. V prvním průchodu můžeme třídit čísla podle jejich poslední cifry, pak podle předposlední a tak dále až k první cifře. Díky tomu, že je přihrádkové třídění stabilní (nemění pořadí prvků se stejným třídícím klíčem), tak nám na konci vyjde seřazená posloupnost.

V našem případě by bylo nejlepší pořídit si pole N přihrádek a rozházet do nich čísla podle zbytku po dělení N. Pak je můžeme celočíselně vydělit N a celý postup opakovat, dokud nebudou čísla seřazená (vlastně jsme tím převedli čísla do soustavy o základu N). Jeden průchod přihrádkového třídění nám zabere čas $\mathcal{O}(N)$, kolik průchodů ale potřebujeme udělat?

Maximální hodnotu, která nám může vylézt z polynomu pátého stupně, lze odhadnout jako $\mathcal{O}(N^5)$. S úplně čistým svědomím však nelze prohlásit, že nám vždy stačí pět průchodů přihrádkového třídění, záleží totiž ještě na konstantách u samotného polynomu. Správný časový odhad přihrádkové třídění by tedy ještě musel počítat s maximem M, které nám může z polynomu vyjít, a výsledná časová složitost by tak byla $\mathcal{O}(N \log_N M)$.

Monotónnost

Pojďme využít ještě jiné vlastnosti polynomu pátého stupně, která nám již umožní dosáhnout skutečně lineární časové složitosti vzhledem k počtu obrazů.

O polynomu prvního stupně (třeba $f(x) = 4x$) můžeme říci, že je v celém svém rozsahu monotónní (tedy buď klesá nebo stoupá, ale nemění se to). U polynomu druhého stupně podobně platí, že má nejvýše dva monotónní úseky (třeba $f(x) = -x^2 + 2x$ od minus nekonečna do jedničky roste a pak od jedničky do nekonečna zase klesá). Podobná vlastnost existuje i u polynomů vyšších stupňů a lze prohlásit, že polynom k-tého stupně má nejvýše k monotónních úseků (může jich mít ale i méně, třeba $f(x) = x^5$ má jen jeden monotónní úsek, ačkoli má stupeň 5).

Díky této vlastnosti víme, že funkční hodnoty v našem seznamu dvojic jsou v některých úsecích seřazené. Úloha se nám tak zjednodušuje na nalezení těchto úseků a jejich slití dohromady.

Pojďme nejprve vyřešit jejich nalezení – budeme si vytvářet seznamy prvků jednotlivých úseků, u kterých si navíc budeme pamatovat jejich *směr* (jestli daný úsek stoupá nebo klesá). Na začátku do prvního úseku vložíme první dva prvky a podle nich určíme, jestli první úsek stoupá nebo klesá. Pak budeme procházet další prvky, a dokud bude další prvek vyhovovat směru, tak ho přidáme do aktuálního úseku, jinak založíme nový úsek s opačným směrem a vložíme prvek do něj. To zvládneme v $\mathcal{O}(N)$. Pro jednoduchost slévání si ještě můžeme na konci otočit klesající úseky na rostoucí, což také zabere maximálně čas $\mathcal{O}(N)$.

Nyní máme (nejvýše) pět seznamů, ve kterých jsou uspořádané funkční hodnoty, zbývá nám udělat jejich slití. To bude podobné slévání, které se používá v Mergesortu – v každém kroku se podíváme na první prvek všech seznamů, nalezneme ten nejmenší z nich, odebereme ho a vložíme ho do finálního pole. Tento postup opakujeme N-krát a na konci dostaneme seřazené pole. Každý krok nám zabere konstantně mnoho času (porovnání pěti hodnot), takže celkový čas tohoto slévání, a tedy i celého třídění využívajícího monotónnost je $\mathcal{O}(N)$.

Pokud si takto seřadíme dvojice z úvodu podle funkčních hodnot, tak už lehce vydáme správné pořadí obrazů.

Úlohu připravili: David Klement, Jirka Setnička

33-5-2 Zakázaná písmena

První, čeho je dobré si všimnout (a trochu to naznačovaly i limity uvedené v zadání), je, že při nahrazování písmen může délka řetězce velmi rychle narůstat (konkrétně exponenciálně). Představme si například nahrazovací pravidla `a` → `bbbb`, `b` → `cccc`, ..., `y` → `zzzz`, `z` → `0000`. Potom si

snadno rozmyslíte, že z jednoho `a` se stane řetězec nul dlouhý $4^{26} = 2^{52}$. Takovýto řetězec by zabral 4.5 *petabajtů* paměti, nemluvě o tom, jak dlouho by trvalo jej sestrojit. Tedy určitě si nemůžeme dovolit konstruovat celý výsledný řetězec.

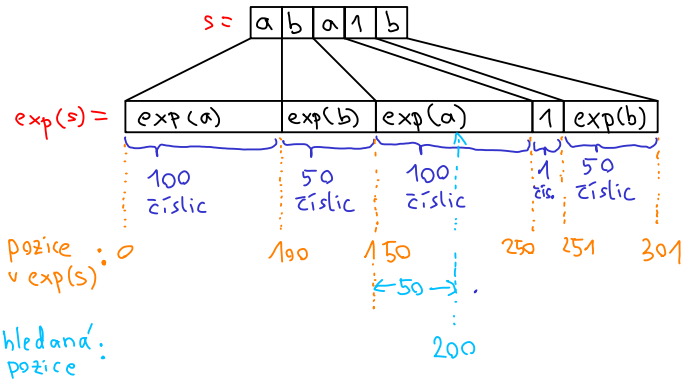
Nyní trochu terminologie. *Náhrada* nějakého znaku c (budeme značit sub(c)) je prostě pravá strana jeho nahrazovacího pravidla (může obsahovat písmena i jeho číslice). Oproti tomu *expanze* nějakého řetězce s (značíme exp(s)) je řetězec, který vznikne, pokud začneme s s a nahrazujeme podle pravidel, dokud to jde. Expanze vždy obsahuje pouze číslice.

Nejprve zkusíme zodpovědět otázku, která s úlohou na první pohled tolik nesouvisí. Rádi bychom pro každé písmeno spočítali délku jeho expanze.

To snadno vyřešíme pomocí dynamického programování.[1] Délku expanze jednoho písmene spočítáme tak, že se podíváme na jeho náhradu, rekurzivně spočteme délky expanzí všech písmen v této náhradě a sečteme je. Navíc si pořídíme cache na už spočítané výsledky, takže nikdy nebudeme počítat expanzi stejného písmene dvakrát.

Výpočet pro jedno písmeno nás stojí (nepočítaje rekuzivní volání) $\mathcal{O}(\ell)$, kde ℓ je délka jeho náhrady. Protože pro každé písmeno spouštíme výpočet nejvýše jednou, celkem spotřebujeme čas $\mathcal{O}(L)$, kde L je součet délek všech nahrazovacích pravidel.

Nyní si představme, že máme nějaký řetězec s a navíc známe délku expanzí všech znaků. Pak si pro každý znak c v s můžeme spočítat, na jaké pozici v exp(s) začíná expanze tohoto znaku. To je prostě součet délek expanzí všech předcházejících znaků. Například pro řetězec `aba1b` s délkami expanzí `a`: 100, `b`: 50:



Nyní například, kdybychom chtěli najít číslici na pozici 200 v exp(s), víme že bude ležet uvnitř expanze druhého a, konkrétně na pozici 50. Jinými slovy potřebujeme najít znak na pozici 50 v exp(`a`), což můžeme udělat rekurzivní aplikací stejného postupu na sub(`a`).

Z toho plyne přímočarý rekurzivní algoritmus Cislice(s, i) na nalezení i-té číslice v exp(s) pro nějaký řetězec s:

1. Pro každý znak urči pozici v exp(s), na které začíná jeho expanze. Pro k-tý znak s si tuto pozici označíme pos(k). *Např. v příkladu výše* pos(2) = 150.
2. Urči, do expanze kterého znaku s patří pozice i. Říkejme tomuto znaku c, jeho pozici v s pak k. Víme, že expanze c začíná na pozici pos(k) v exp(s). *Např. v příkladu výše c=`a`, k=2.*
3. Pokud c je číslice, vrať c.
4. Pokud c je písmeno, vrať Cislice(sub(c), i − pos(k)).

[1] `http://ksp.mff.cuni.cz/viz/kucharky/dynamika`

Výpočet Cislice(s, i) trvá čas $\mathcal{O}(|s|)$, nepočítaje rekurzivní volání. Označme si K délku výchozího řetězce na vstupu. Pak v našem programu pro zjištění jedné číslice proběhnou následující volání funkce Cislice:

- Jedno volání s řetězcem délky K.
- Pro každé písmeno c nejvýše jedno volání s řetězcem sub(c). Všimněme si, že jelikož v pravidlech nejsou cykly, nikdy se nezavoláme dvakrát na sub(c) pro stejné c.

Pokud si označíme L celkovou délku všech nahrazovacích pravidel, bude časová složitost celého algoritmu $\mathcal{O}(K + L)$ – včetně předpočítání délek expanzí.

A protože zadání požaduje vypsání 10 číslic, prostě celý algoritmus spustíme desetkrát pro pozice P až $P + 9$. To nám časovou složitost asymptoticky nezhorší.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-5-2.py`

Úlohu připravili: Jirka Sejkora, Filip Štědronský

33-5-3 Těžký vozík

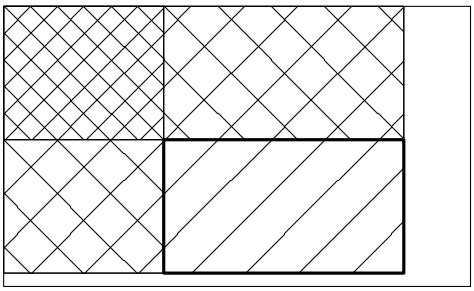
Uvážíme všechny možné kombinace polohy a rychlosti jako vrcholy grafu. Každý vrchol tedy reprezentuje možný stav vozíku před začátkem tahu.

Do tohoto grafu pak přidáme orientované hrany. Každá hrana bude reprezentovat přechod mezi stavy, který lze učinit pomocí jednoho tahu.

Z každého vrcholu tedy povede maximálně devět hran, protože v každém směru máme tři možnosti, jak rychlost vozíku změnit.

Pro každý vrchol zvládneme hrany z něho odcházející poměrně snadno spočítat. Pro každou možnou změnu rychlosti nejprve rychlost přepočítáme a novou polohu pak určíme jako součet rychlosti a polohy. Poté je nutné zkontrolovat, zdali nová poloha a rychlost skutečně odpovídají nějakému vrcholu, tedy jestli poloha leží v plánku a velikost rychlosti není příliš veliká (bude vysvětleno později). Zbývá tedy jen určit, zdali je tento čtverec volný, a tedy tato hrana je validní.

Toto můžeme kontrolovat pomocí dvojrozměrných prefixových součtů. To je datová struktura, která si pamatuje počet překážek pro všechny obdélníky s jedním rohem v levém horním rohu plánku a druhým rohem na jednom z políček. Pomocí nich lze pak zjišťovat počet překážek v libovolném obdélníku v konstantním čase. Stačí vhodně sčítat a odečítat počty překážek v předpočítaných obdélnících:



Když už máme graf vygenerovaný, požadovanou odpověď již zvládneme jednoduše zjistit pomocí průchodu do šířky. Jedná se totiž o vzdálenost vrcholů odpovídající políčkům začátku a konce s nulovou rychlostí.

Nyní pojďme omezit velikost grafu.

Spočítáme, kolik minimálně políček vozík projede, než se rozjedeme na rychlost X v nějakém směru a pak zase zabrzdí. Jedná se o součet řady: $1+2+\cdots+(X-1)+X+(X-1)+\cdots+2+1 = \frac{(X-1)X}{2}+X+\frac{(X-1)X}{2} = X+X(X-1) = X^2$. V kolmém směru se přitom může pohybovat libovolně. Tedy maximální možná velikost rychlosti v každém směru je odmocninou z délky příslušného směru.

Počet vrcholů grafu tedy je $\mathcal{O}(NM\sqrt{NM})$. Z toho nám vyplyne odhad paměťové i časové složitosti $\mathcal{O}(NM\sqrt{NM})$.

Úlohu připravil: Jirka Kalvoda

33-5-4 Nevěřící Bob

Různost prvků

Pokud chceme Boba přesvědčit o tom, že čísla $x_1, \ldots, x_n$ jsou navzájem různá, stačí mu ukázat, jak je setřídit. Pošleme mu posloupnost indexů $i_1, \ldots, i_n$ takovou, že

$$x_{i_1} < x_{i_2} < \ldots < x_{i_n}.$$

Tento důkaz je velký $\mathcal{O}(n)$. V čase $\mathcal{O}(n)$ umíme zkontrolovat, že se v něm každý index od 1 do n vyskytuje právě jednou. Pak projdeme čísla v tomto pořadí indexů a ověříme, že jsou ostře rostoucí.

Nejkratší cesta

Nejprve předpokládejme, že žádná hrana grafu nemá nulovou délku. Pak jako důkaz stačí předložit vzdálenosti (délky nejkratších cest) od startu do všech ostatních vrcholů. Ukážeme, jak tento důkaz ověřit.

Označme $d(v)$ vzdálenost od startu s do v uvedenou v důkazu a $\ell(u,v)$ délku hrany mezi u a v.

Nejprve zkontrolujeme, že důkaz je konzistentní s grafem. K tomu stačí ověřit následující podmínky:

1. $d(s) = 0$
2. Pro každý vrchol $v \neq s$ existuje jeho soused u takový, že $d(v) = d(u) + \ell(u,v)$. Tomuto sousedovi říkáme *předchůdce* vrcholu v.
3. Pro každou hranu $\{u,v\}$ platí $d(v) \leq d(u) + \ell(u,v)$. *(Vlastně říkáme, že si žádnou hranou nelze zkrátit cestu. To je trojúhelníková nerovnost.)*

Proč to stačí? Z prvních dvou podmínek plyne, že pokud z nějakého vrcholu v chodíme po předchůdcích, $d(\ldots)$ stále klesá, takže po konečném počtu kroků musíme dojít do s. Tím jsme sestrojili cestu z v do s, jejíž délky hran se sečtou na $d(v)$. A je-li $d(v)$ délkou nějaké cesty, nemůže být menší než délka nejkratší cesty. Jenže $d(v)$ nemůže být ani větší – kdyby existovala nějaká kratší cesta než ta, kterou jsme sestrojili, na aspoň jedné její hraně by musela být porušena podmínka 3.

Pokud nám Alice kromě důkazu pošle i nejkratší cestu, umíme snadno ověřit, že je skutečně nejkratší. Stačí pro každou hranu $\{u,v\}$ na cestě zkontrolovat, že $d(v) = d(u)+\ell(u,v)$.

Máme tedy důkaz velký $\mathcal{O}(n)$, který umíme zkontrolovat v čase $\mathcal{O}(n+m)$.

Zbývá dořešit, co si počít s hranami nulové délky. Předchozí důkazový systém pro ně nefunguje – chození po předchůdcích by mohlo skončit na nějakém cyklu z nulových hran a nikdy nedojít do s. Stačí ale provést jednoduchou transformaci: všechny délky hran vynásobíme n a přičteme k nim 1. Tím pádem se délky cest vynásobí n a přičte se počet hran na cestě. Nejkratší cesty stále zůstanou nejkratšími cestami (počet hran nikdy nedosáhne n, pročež se nejkratší a druhá nejkratší cesta nemohou prohodit). Tak jsme se zbavili nulových hran, a přesto stále umíme ověřit nejkratší cestu v původním grafu.

Editační vzdálenost

Editační vzdálenost převedeme na nejkratší cestu v grafu pomocí myšlenky z kuchařky o dynamickém programování.

Hledáme-li vzdálenost řetězce α délky a od řetězce β délky b, vytvoříme ohodnocený orientovaný graf, jehož vrcholy budou v ležet mřížce $(a+1)\times(b+1)$. Řádky budou indexované znaky prvního řetězce, sloupce znaky druhého řetězce.

Hrany budou odpovídat editačním operacím. Speciálně z vrcholu (i,j) povedou hrany:

- do $(i+1,j)$ s délkou 1 – tato hrana odpovídá smazání znaku $\alpha[i]$
- do $(i,j+1)$ s délkou 1 – tato odpovídá vložení znaku $\beta[i]$
- do $(i+1,j+1)$ s délkou 1 – změna znaku $\alpha[i]$ na $\beta[j]$
- do $(i+1,j+1)$ s délkou 0, pokud $\alpha[i] = \beta[j]$ – znak necháme na pokoji

Stejně jako v kuchařce nahlédneme, že cesty z $(0,0)$ do (n,m) odpovídají posloupnostem editačních operací, které z α udělají β. Navíc délka cesty je rovna počtu operací. Vzdálenost po nejkratší cestě tedy odpovídá editační vzdálenosti α od β.

Teď by stačilo použít náš důkazový systém pro nejkratší cesty (a rozmyslet si, že funguje i pro orientované grafy). Jenže ouha – graf je moc velký: má $\Theta(ab)$ vrcholů.

Zadání nám ale slibuje, že editační vzdálenost bude řádově menší než a a b. Nechť e je editační vzdálenost. V grafu tedy existuje cesta délky e z $(0,0)$ do (a,b). Tato cesta ovšem nikde nemůže opustit pás okolo „diagonály“ (vrcholy (i,i)) široký e na každou stranu od diagonály. Všechny hrany, které se vzdalují od diagonály, mají délku 1, takže všechny vrcholy mimo pás jsou od $(0,0)$ vzdálené víc než e.

Stačí tedy důkazový systém pro nejkratší cesty použít na tento pás, čímž jsme velikost důkazu snížili na $\Theta(ae)$. Čas potřebný na ověření důkazu bude také $\Theta(ae)$. (Divíte-li se, kam se podělo b, pak vězte, že a a b se mohou lišit nejvýš o e, takže $\Theta(ae)$ je totéž co $\Theta(be)$.)

Úlohu připravil: Martin „Medvěd“ Mareš

33-5-X1

Právě jste vynalezli stroj času. Potřebujete se dostat do minulosti. Máte neodkladný úkol. Musíte říct svému mladšímu já něco důležitého – zadání této úlohy, aby ji mohlo řešit.

Váš stroj času je ovšem jenom prototyp a proto má nějaká omezení. Umí vás přenést pouze do určitého času – do února tohoto roku. Navíc vás nedoveze do vašeho domu, ale skončíte na druhé straně svého města. Z důvodu rušení je tu ještě jeden problém, můžete cestovat pouze v noci. V minulosti se samozřejmě nemůžete zdržovat moc dlouho, aby nevznikl nějaký zásadní paradox.

Z místa, kam vás stroj času přenese, se tedy budete muset dostat k sobě domů. Ovšem je tu problém. V době, kam se

přenesete, platí přísná protiepidemická opatření.[2] Je tedy zakázáno pohybovat se v noci jen tak po městě. Na toto opatření samozřejmě dohlíží policie, se kterou se nechcete dostat do křížku, protože vysvětlit policistům, že cestujete v čase, není jednoduché.

Bohužel se vám nepodařilo zjistit, kde se v daný den nacházely policejní hlídky. Ovšem víte, že policajti nedělají žádnou práci zbytečně, takže ve městě bude rozmístěno co nejméně hlídek tak, aby každá křižovatka byla *v dohledu* alespoň jedné hlídky. Město si můžeme představit jako graf – křižovatky odpovídají vrcholům a ulice mezi nimi jsou hrany. Být v dohledu znamená, že se jedná o vrchol, v jehož sousedním vrcholu je hlídka.

Vašim úkolem tedy je zjistit, kde byly hlídky (můžete předpokládat, že existuje jediné minimální rozestavění), a pak naplánovat nejbezpečnější cestu domů. Chodit přes křižovatky, kde je hlídka, samozřejmě nemůžete. Také ale můžete být spatřeni hlídkou na sousední křižovatce. *Nebezpečnost* průchodu každou křižovatkou je tedy počet hlídek, které s ní sousedí. Vy pak chcete minimalizovat součet nebezpečností na křižovatkách po cestě.

Na vstupu dostanete popis města. Pro každý vrchol je tu jeden řádek začínající znakem označujícím daný vrchol následovaný dvojtečkou. Za ní jsou pak vypsány všechny sousední vrcholy dle pořadí, jak jdou ulice na dané křižovatce po sobě. Začínáte v prvním vrcholu a chcete se dostat do posledního.

Na výstup je potřeba vypsat několik řetězců oddělených bílými znaky. Nejprve jako jedno slovo identifikátory všech vrcholů, kde budou hlídky, poté nebezpečnost vaší trasy, dále pak vrchol, do kterého budete muset vyrazit ze startu, a nakonec popis trasy. Pro každý vrchol trasy kromě startu a cíle vypište řetězec znaků `+` nebo `-` reprezentující, jestli máte dále vyrazit [počet mínusů]-tou ulicí doleva, nebo [počet plusů]-tou ulicí doprava vůči ulici, kterou jste přišli. Na každé křižovatce je nutné použít nejkratší možný zápis.

Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš

33-5-S Globální iluminace a path tracing

Zde naleznete odkazy na referenční implementace úkolů ze všech dílů seriálu. Slovní popis řešení zde není, protože tím je vlastně samotný text seriálu.

První díl – Z hlubin fraktálů

- Úkoly 1, 2, 3 (pro zrcadlení viz řádek 28): `https://www.shadertoy.com/view/WstyRX`
- Úkol 4: `https://www.shadertoy.com/view/3dtcRX`
- Úkol 5: `https://www.shadertoy.com/view/ws3BzH`

Druhý díl – Šumy

- Úkol 1: `https://www.shadertoy.com/view/WtdcDB`
- Úkol 2: `https://www.shadertoy.com/view/tttyWS`
- Úkoly 3, 4: `https://www.shadertoy.com/view/wtdyWS`
- Úkol 5: `https://www.shadertoy.com/view/tttcWS`

Třetí díl – Světlo a stín

Úkol 2: `https://www.shadertoy.com/view/WtyBDV`

Vzorové řešení prvního úkolu je vlastně první půlka textu zadání, proto zde není.

Čtvrtý díl – Raytracing

- Úkoly 1–5: `https://www.shadertoy.com/view/wdGBzV`
- Úkol 5, alternativa s vícenásobnými odrazy: `https://www.shadertoy.com/view/NsfSzX`
- Úkol 6: `https://www.shadertoy.com/view/fdfXRX`

Pátý díl – Globální iluminace a path tracing

Všechny úkoly:
`https://www.shadertoy.com/view/3ttfDr`

Všechny programy společně

Zip:
`http://ksp.mff.cuni.cz/viz/33-S-.zip`

Úlohu připravil: Kuba Pelc

[2] Nebo protiepidemiologická? Bylo to tehdy všechno nějaké zmatené…

Výsledková listina páté série třicátého třetího ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*5-1*	*5-2*	*5-3*	*5-4*	*5-S*	*5-X1*	*série*	*KSP-X*	*celkem*
1.	Kristýna Petrlíková	SPŠJičín	3	15	11	10	10	14	15	10	60,0	10,0	300,5
	max. std. počet bodů				11	10	10	14	15	10	60,0	50,0	300,0
2.	Jan Adámek	GKepleraPH	4	10	11	10	10	13	14,5	10	58,5	28,0	277,0
3.	Daniel Skýpala	GTomkovaOL	3	20	11	10	9	10	15		55,0	5,0	270,0
4.	Jiří Kvapil	GTomkovaOL	3	19	2	10	8	7	15		42,0	3,0	257,5
5.	Ondřej Skácel	GTomkovaOL	2	5	3	10	5,5	10	15		43,5	0,0	253,0
6.	Filip Hejsek	GPísnickáPH	4	7	11,5	10				10	21,5	10,0	217,0
7.	Robert Jaworski	GÚstavníPH	3	7	10	10		3			23,0	2,0	214,5
8.	Jan Kotovský	GPísnickáPH	2	6	4	10	1	10	15		40,0	0,0	210,5
9.	Lukáš Veškrna	GKepleraPH	3	5		10	5		15	1,8	30,0	5,8	199,0
10.	Vít Skalický	GPísnickáPH	3	19		10	7,5		10		27,5	0,0	188,0
11.	Jakub Ondroušek	GTomkovaOL	1	5		10	6		15		31,0	0,0	176,5
12.	Viktor Fukala	GKepleraPH	4	8							0,0	19,0	171,5
13.	Jakub Surga	ParkLane	3	5		2		0			2,0	0,0	156,5
14.	Janek Hlavatý	GJirsíkaČB	2	9		10		5	3	0	18,0	0,0	151,0
15.	Patrik Herman	GTomkovaOL	2	6	4	10		1	8		23,0	0,0	137,5
16.	Eliška Macáková	CENADA BA	1	3							0,0	20,0	133,0
17.	Matej Štencel	GPošKošice	4	6							0,0	0,0	114,0
18.	Dominik Farhan	GMikulášPL	4	8							0,0	0,0	113,0
19.	Vladimír Chudý	G Chrudim	4	19						9,8	0,0	14,3	110,0
20.	Adam Kolník	SSŠVTPraha	2	5		10				0,7	10,0	0,7	98,0
21.	Ondřej Sladký	GMikulášPL	4	11							0,0	22,0	91,0
22.	Václav Janáček	GJarošeBO	4	6							0,0	16,0	80,0
23.	Prokop Randáček	GFXŠaldyLI	2	6						0,5	0,0	0,5	78,0
24.	Šimon Genčur	GBBr	1	7		2					2,0	0,0	66,5
25.	Pavel Jordán	GPOA Znojmo	2	2							0,0	0,0	66,0
26.	Klára Grinerová	GZborovPH	4	5		5		3			8,0	0,0	42,5
27.	Kryštof Maxera	GJírovcČB	0	2	4	10	2			1	16,0	1,0	33,0
28.	David Holas	SPŠEMasLI	1	1							0,0	1,0	32,5
29.	Martin Havelka	Gym Třeboň	3	3							0,0	0,0	28,0
30.	Robert Gemrot	GKomHavíř	4	4							0,0	0,0	27,5
31.	Petr Šícho	GKepleraPH	3	2							0,0	0,0	24,0
32.	Jiří Bartošík	SUHr	3	2							0,0	0,0	22,0
33.	Albert Kučera	GNadŠtolPH	4	4							0,0	0,0	21,0
34.	Petr Hladík	GMikulášPL	3	3	10,5	10				4,8	20,5	4,8	20,5
35.	Kristýna Umlaufová	SPŠOstrov	4	2							0,0	0,0	20,0
36.	Jáchym Tuma	G FrýdlNOs	0	2							0,0	0,0	19,0
37.	Andrej Thomas Dobrev	GJHroncaBA	4	1							0,0	0,0	18,0
38.	Michal Žáček	MensaG	4	1							0,0	0,0	15,0
39.	Daniel Šoltýs	GTřeKošice	3	3							0,0	0,0	14,0
40.	Adam Húšťava	EupSchoolLux	3	4							0,0	0,0	13,0
41.	Tomáš Kašpárek	G FrýdlNOs	3	1							0,0	0,0	12,0
42.–48.	Vojtěch Březina	GCoubTábor	4	4							0,0	0,0	10,0
	Petr Filip	GLovosice	2	1							0,0	0,0	10,0
	Vojtech Gadurek	PORGPha	4	1							0,0	0,0	10,0
	Štěpán Kovář	GNadKavaPH	4	1							0,0	0,0	10,0
	Michal Pavlíček	MendelGOP	3	1							0,0	0,0	10,0
	Lukáš Tomoszek	GTři	3	1		10					10,0	0,0	10,0
	Filip Úradník	GyMimoň	4	1							0,0	0,0	10,0
49.	Jan Ráček	SPŠEMasLI	1	1							0,0	0,0	7,0
50.–51.	Bohumil Kulvejt	G Sokolov	3	1							0,0	0,0	6,0
	Josef Malý	GPísnickáPH	2	1							0,0	0,0	6,0
52.	Veronika Jůzková	MensaG	3	3		1					1,0	0,0	4,0
53.–54.	Kryštof Marek	SGPCE	1	1		2					2,0	0,0	2,0
	Matěj Strnad	ZŠRiegraSM	0	1							0,0	0,0	2,0

Bonusové úlohy označené „X“ mají svou vlastní výsledkovou listinu a nepočítají se do normálního bodování ročníku.

Výsledková listina KSP-X po páté sérii třicátého třetího ročníku

	řešitel	*škola*	*ročník*	*sérií*	*1-X1*	*2-X1*	*3-X1*	*4-X1*	*5-X1*	*celkem*
0.					10	10	10	10	10	50,0
1.	Jan Adámek	GKepleraPH	4	10	7	6	5	0	10	28,0
2.	Ondřej Sladký	GMikulášPL	4	11	9		10	3		22,0
3.	Eliška Macáková	CENADA BA	1	3	9	1	10			20,0
4.	Viktor Fukala	GKepleraPH	4	8	9	10				19,0
5.	Václav Janáček	GJarošeBO	4	6	8		8			16,0
6.	Vladimír Chudý	G Chrudim	4	19	4,5				9,8	14,3
7.–8.	Filip Hejsek	GPísnickáPH	4	7					10	10,0
	Kristýna Petrlíková	SPŠJičín	3	15					10	10,0
9.	Lukáš Veškrna	GKepleraPH	3	5	4				1,8	5,8
10.	Daniel Skýpala	GTomkovaOL	3	20	4	1	0			5,0
11.	Petr Hladík	GMikulášPL	3	3					4,8	4,8
12.	Jiří Kvapil	GTomkovaOL	3	19	2	1				3,0
13.	Robert Jaworski	GÚstavníPH	3	7	1	1				2,0
14.	Pavel Altmann	GMikulášPL	2	1					1,4	1,4
15.–16.	David Holas	SPŠEMasLI	1	1			1			1,0
	Kryštof Maxera	GJírovcČB	0	2					1	1,0
17.	Adam Kolník	SSŠVTPraha	2	5					0,7	0,7
18.	Prokop Randáček	GFXŠaldyLI	2	6					0,5	0,5
19.	Petr Budai	G JGJ PH	4	5					0,2	0,2

Bonusové úlohy z jednotlivých sérií se nepočítají do bodování ročníku. Mají svou vlastní výsledkovou listinu a za jejich úspěšné vyřešení (alespoň polovina bodů za úlohu) udělujeme speciální odměny.