Korespondenční Seminář z Programování

37. ročník **KSP** **Březen 2024**

Vzorová řešení třetí série třicátého sedmého ročníku KSP

37-3-1 Robot

Najdôležitejšie pri riešení tejto úlohy je uvedomiť si, aké prípady môžu nastať. Keď im budeme rozumieť, môžeme už úlohu vyriešiť pomocu jednoduchého prehľadávania grafu.

Zaujíma nás, ktoré políčka skladu sú dosiahnuteľné, teda na ktoré vieme robota prinútiť ísť. Určite to platí pre políčko, na ktorom začína. Ďalej teda musíme ísť z neho. Pokiaľ má toto políčko jedno alebo dve voľné susedné políčka, môžeme robotovi prikázať ísť na opačné pole, než chceme aby šiel, a prinútiť ho tak spraviť krok požadovaným smerom.

Pokiaľ má však susedných políčok viac, má robot po každom príkaze aspoň dve možnosti kam sa pohnúť, a teda nevieme ho dostať ani na jedno susedné políčko, pretože sa vždy pohne inak, než chceme.

Stačí nám teda spustiť BFS alebo DFS z počiatočného políčka a pokračovať len po políčkach, ktoré majú najviac dvoch susedov. Vždy keď takto nájdeme pascu, určite je dosiahnuteľná, takže všetky pasce, ktoré nájdeme, budú označené správne. Funguje takéto riešenie? Bohužiaľ nie.

Problémom je, že stále môžu existovať pasce, ktoré sme týmto procesom neoznačili. V nasledujúcom príklade môžeme vysávač naviesť do každej pasce napriek tomu, že doterajším algoritmom by sme neoznačili ani jednu:

```
##.P.
V..#P
##.P.
```

Ale v tomto podobnom príklade môžeme vysávač naviesť iba do prvej pasce v strednom riadku, nevieme ho naviesť do žiadnej zo zvyšných troch.

```
##.P.##
V..#P.P
##.P.##
```

Predošlým algoritmom by sme označili len prvé tri políčka cesty k pasciam (vrátane začiatočného). Využijeme teda tieto príklady a pridáme pozorovanie: okrem predošlých označených pascí dokážeme vysávač naviesť aj do takých, ktoré

1. ležia na ceste, ktorá spája oba východy z križovatky.
2. ležia na križovatke, kde sa stretávajú všetky vetvy, na ktoré sa nejaká cesta delí (ale nie do takých, ktoré ležia na ceste do tejto križovatky, pretože vysávač si vždy vie vybrať inú z ciest, aby sa im vyhol).

Tieto pasce vieme nájsť tak, že z každej pasce spustíme BFS prehľadávanie, ktorým sa pokúsime dostať na štartovné políčko. Pokiaľ navštívime križovatku (čo vieme zistiť z počtu susedov daného políčka), označíme si, že na to, aby sme z nej mohli pokračovať, musíme ju navštíviť ešte raz za každú ďalšiu cestu, ktorá do nej vedie, okrem tej, ktorou chceme pokračovať (teda o dva menej, ako je počet jej susedných políčok, pretože z jedného sme práve prišli a jedným odídeme).

Vždy, keď danú križovatku navštívime znova, jej počítadlo znížime. Ak sa počítadlo križovatky dostane na nulu, pridáme ju do fronty pokračujeme z nej pri prehľadávaní. Ak narazíme na štartovné políčko, označíme pascu, z ktorej sme začali prehľadávanie, ako dosiahnuteľnú.

Toto prehľadávanie má časovú zložitosť $\mathcal{O}(R \times S \times P)$, kde P je počet pastí, pretože teoreticky môže byť potrebné prejsť pre každú pascu celý sklad.

Program (Python):
`http://ksp.mff.cuni.cz/viz/37-3-1.py`

Úlohu připravili: Daniel Cullíver, Ján „Jančí“ Plachý

37-3-2 KSP-Nim-Vim

Moc se omlouváme, ale řešení Nim-Vimu nám ještě chvilku zabere sepsat.

37-3-3 Hledání podmatice

Kdybychom tuto úlohu řešili v 1D světě, šlo by o klasický problém hledání v textu. Na ten existuje například algoritmus KMP, o kterém se dočtete v kuchařce.[1] My ale potřebujeme hledat ve 2D. Celkem přímočarý způsob, jak převést algoritmus do více dimenzí, je zpracovávat jednotlivé osy postupně.

[1] `http://ksp.mff.cuni.cz/viz/kucharky/hledani-v-textu`

Nejdříve budeme hledat po řádcích. Jelikož hledáme výskyty několika různých řádků malé matice, použijeme algoritmus Aho-Corasicková. Ten má navíc tu výhodu, že pokud jsou některé řádky malé matice identické, tak je najdeme, protože jim bude odpovídat stejný vrchol v trii. Projdeme každý řádek velké matice a na všech políčkách si zaznamenáme, v jakém stavu (vrcholu trie) jsme byli. Tím dostaneme novou matici, která nám říká, jestli a jaký malý řádek na daném místě končí.

Poté budeme hledat po sloupcích. Sloupec obsahuje pravý kraj výskytu malé matice, právě když jsou v něm ve správném pořadí konce řádků malé matice. To je opět hledání v textu, jen s jinou abecedou tvořenou vrcholy trie. Jelikož je tentokrát jenom jedna jehla, použijeme algoritmus KMP. Projdeme každý sloupec zpracované velké matice. Každý výskyt jehly pak odpovídá pravému dolnímu rohu výskytu malé matice.

Celková časová složitost algoritmu je $\mathcal{O}(N^2 + K^2)$. Trii si připravíme v $\mathcal{O}(K^2)$, KMP tabulku v $\mathcal{O}(K)$ a obě prohledávání trvají $\mathcal{O}(N^2)$. Pohyb v trii sice závisí na abecedě, ale ta je v prvním prohledávání konstantní. Ve druhém hledání máme abecedu velikosti až K^2, ale KMP naštěstí umí pracovat bez zpomalení s libovolně velkou abecedou. V druhém prohledávaní můžeme vynechat prvních $K-1$ sloupců, protože nemůžou obsahovat konec malého řádku, ale asymptoticky to nic nezmění.

Program (C++):
`http://ksp.mff.cuni.cz/viz/37-3-3.cpp`

Program (Python):
`http://ksp.mff.cuni.cz/viz/37-3-3.py`

Úlohu připravili: Jan Adámek, Ján „Jančí“ Plachý

37-3-4 Výhybky

(Ne)obvyklé hradlo

Nejdřív si ukažme, že z výhybkového hradla jdou sestrojit hradla, která všichni známe. `NOT` sestrojíme jednoduše – vybereme nulu, pokud je vstupní hodnota jedna, a opačně:

```
NOT: X 0 1
```

U `AND`u si všimneme, že pokud první vstup je nula, výsledek je určitě nula. Jinak stačí vrátit druhý vstup:

```
AND: X Y 0
```

`OR` vyřešíme obdobně:

```
OR: X 1 Y
```

Zbývá vyřešit `XOR`. Pokud oba vstupy jsou jedna, vrátíme nula. Jinak vrátíme jejich `OR`:

```
AND: X Y 0
OR: X 1 Y
XOR: AND 0 OR
```

Čas řešit

Nyní pojďme řešit jednotlivé podúlohy. V první podúloze stačí udělat `AND` všech vstupů. Abychom dosáhli logaritmické hloubky, tak stačí udělat `AND`y dvou sousedních hodnot. (Pokud je počet vstupů lichý, tak nám jedna zbude.) A to stačí dělat opakovaně, čímž skončíme s jednou hodnotou. Protože v každé vrstvě vydělíme počet hodnot dvěma (a zaokrouhlíme nahoru), hloubka bude logaritmická.

Druhou podúlohu vyřešíme obdobně – stačí vyměnit `AND` za `OR`.

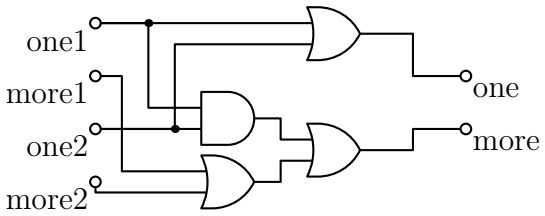
V třetí podúloze chceme zjistit, jestli máme právě jednu jedničku. Na to zobecněme náš postup. Máme nějaké hodnoty reprezentující úseky. (V předchozích podúlohách to byl `AND` nebo `OR` daného úseku.) Poté v jedné vrstvě zkombinujme sousední dvojice do jedné, čímž dané hodnoty budou odpovídat úseku o dvojnásobné délky. (Až na případy, kdy počet hodnot je lichý.)

A v této podúloze si budeme posílat následující dvojici:

– Máme alespoň jednu jedničku (`one`)
– Máme alespoň dvě jedničky (`more`)

Na začátku v každém úseku o máme dvojici (i-tý vstup, konstantní nula). Kombinovat dvojice s již zkonstruovanými hradly zvládneme následovně:

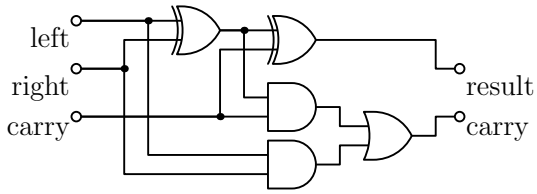
```
one = one1 OR one2
more = (one1 AND one2) OR more1 OR more2
```



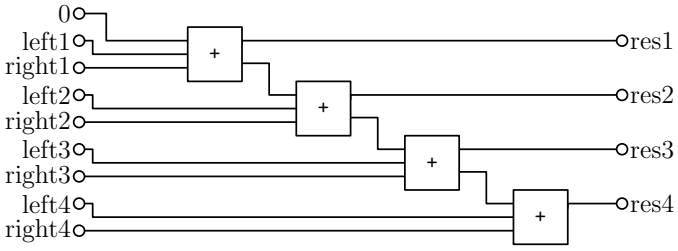
Na konci stačí pouze zkontrolovat, že výsledná dvojice je $(1, 0)$. To můžeme udělat následovně: Ze všech hodnot, co mají být 0, uděláme `NOT`, a následovně všechny hodnoty z`AND`ujeme.

Vzhůru na čtvrtou úlohu – zde je už doopravdy zapotřebí počítat počty jedniček. Naše úseky budou reprezentovány $\log n$ bity, udávajícími počet jedniček v daném úseku. (Na začátku jen vstupní hodnoty zleva doplníme konstantními nulami.) A zkombinovaní dvou úseku odpovídání sečtení dvou $\log n$ bitových čísel.

Na to si nejdříve sestrojme primitivní obvod, který sečte tři bity – bit z levého úseku, bit z pravého úseku a bit za přenos. Výsledek bude tedy dvoubitové číslo (`carry`, `result`):



Když už máme sčítačku, tak jsme je schopni zapojit za sebe pro libovolně dlouhá čísla (příklad pro 4 bity):



Kde poslední `carry` bit prostě zahodíme – víme, že tolik jedniček v našem vstupu prostě nemůže být. Na závěr stačí opět znegovat bity, co mají být nulové, a udělat `AND`.

Nicméně tohle použije moc hradel. Uděláme proto ještě jednu optimalizaci – všimneme si, že v i-té vrstvě jedničkových může být pouze nejnižších i bitů. Když tyto nepotřebné části obvodu odstraníme, už se do limitu počtu hradel vlezeme.

Program (Python):
`http://ksp.mff.cuni.cz/viz/37-3-4.py`

Magnum opus

Na závěr zmiňme, že řešení čtvrté podúlohy jde použít na řešení všech ostatních. – Stačí spočítat počet jedniček a porovnat ho s požadovaným počtem. Pouze u podúlohy na alespoň jednu jedničku stačí porovnat s nulou a udělat negaci.

Program (Python):
`http://ksp.mff.cuni.cz/viz/37-3-4-mo.py`

Úlohu připravili: Dan Skýpala, Ben Swart

37-2-X1 Intervalové většiny

Moc se omlouváme, ale řešení intervalových většin nám také ještě chvilku zabere sepsat.

37-3-S Parsery vrací úder

Seriál lze odevzdávat za snížený počet bodů až do konce školního roku. Řešitelům, kteří už třetí sérii odevzdali, rozešleme vzorové řešení napřímo. Pokud chceš tuto sérii přeskočit a začít řešit další díl, můžeš si o vzorové řešení napsat na ksp@mff.cuni.cz.

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*