Výsledková listina osmnáctého ročníku KSP po páté sérii

		škola	*ročník*	*sérií*	*1851*	*1852*	*1853*	*1854*	*1855*	*1856*	*suma*	*celkem*
1.	Josef Pihera	G Strakon	3	10		7	10	9	11		37,3	208,1
2.	Miroslav Klimoš	G Bílovec	1	14	10	7	9	10	9		36,5	190,0
3.	Pavel Klavík	G Chrudim	3	14	9	7	10	9	2		34,6	183,5
4.	Jakub Kaplan	GJKTyla	2	10	7	6		10	11		34,3	161,9
5.	Zbyněk Konečný	GKptJaroš	3	12	8	7		8	13	8	36,6	154,2
6.	Jiří Maršík	GJKTyla	2	5	2	4		8			17,6	138,1
7.	Peter Perešíni	GJGTajov	4	11							0,0	119,4
8.	Michal Pavelčík	G UBrod	3	8		7	10	8			25,6	115,9
9.	Adam Zivner	G UBrod	4	10	7		3	8	9		27,9	113,0
10.	Petr Kratochvíl	G SvětláNS	3	14	10	7					17,0	106,2
11. – 12.	Lukáš Lánský	GJKTyla	2	9		5	4	8			18,5	98,2
	Petr Onderka	G VKlobou	3	5		7					7,0	98,2
13.	Roman Smrž	GOhradní	2	8							0,0	88,1
14.	Jan Kohout	G Roudnice	3	5	2		1	2			8,6	82,2
15.	Tomáš Herceg	G Třebíč	3	11				7	3		10,0	77,3
16.	Michal Vaner	G Turnov	4	5							0,0	72,7
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							0,0	61,6
20.	Kristýna Krejčová	G Tišnov	3	3				6			8,2	56,6
21.	Richard Jedlička	G Vlašim	2	5				4			6,0	55,0
22.	Ondřej Bouda	GKptJaroš	3	6	2	7					10,0	48,1
23.	Daniel Marek	GZborov	4	7							0,0	47,0
24.	Ondřej Bílka	G Zlín	4	12							0,0	44,1
25.	Josef Špak	GJírovco	3	4							0,0	41,5
26.	Radim Pechal	SPŠ Rožnov	3	2							0,0	39,9
27.	Jan Hrnčíř	GFXŠaldy	4	12							0,0	39,3
28.	Jan Dvořák	GZborov	3	2	5	7			10		25,4	38,3
29.	Pavel Veselý	G Strakon	1	2							0,0	37,4
30.	Kateřina Böhmová	G Rožnov	4	2							0,0	36,2
31.	Cyril Hrubiš	G Bílovec	4	9							0,0	36,0
32.	Jiří Machálek	G Holešov	4	4							0,0	35,5
33.	Roman Říha	G Prachat	2	1	2	1	3	6	11		31,1	31,1
34.	Tereza Klimošová	G Lanškr	4	3							0,0	31,0
35.	Radim Cajzl	G NMnMor	0	5	2		3				8,0	30,4
36.	Jakub Pavlík jn.	G Kladno	3	3							0,0	28,1
37.	Vojtěch Molda	G Vsetín	4	1							0,0	26,9
38.	Martin Kahoun	GJNerudy	3	5							0,0	26,1
39.	Ondrej Mikuláš	G Lučenec	3	2							0,0	25,7
40.	Petr Trňák	G UHradi	3	3							0,0	24,3
41.	Martin Majer	SPŠÚžlabin	1	2							0,0	23,9
42.	Tomáš Sýkora	G VKlobou	2	3	2				7		13,8	22,8
43.	Ján Mikuláš	G Lučenec	4	1							0,0	21,7
44.	Adam Ráž	GBudějo	3	5							0,0	21,4
45.	Jan Krajdl	SPŠÚžlabin	1	2							0,0	17,3
46.	Jiří Cabal	SPŠ DvKrál	3	2							0,0	15,1
47.	Rudolf Rosa	G Kladno	3	2							0,0	15,0
48.	Matej Kollár	G PBystric	4	2							0,0	14,8
49.	Jiří Lekeš	G UBrod	2	1			2		6		14,4	14,4
50.	Miroslav Jančařík	G UBrod	2	2		4	2				10,0	13,5
51.	Lukáš Moravec	GSRandyJN	2	1							0,0	12,7
52.	David Škorvaga	G Kralupy	3	1							0,0	12,4
53.	Tomáš Ehrlich	G Holešov	3	3							0,0	9,8
54.	Jakub Balhar	GJNerudy	3	2							0,0	8,9
55.	Marián Bazálik	G Košice	4	1							0,0	8,3
56.	Jan Musílek	G NBydžov	2	1							0,0	7,3
57.	Jan Tichý	GDašická	1	1							0,0	6,6
58.	Jiří Václavík	G Dobříš	4	2							0,0	6,5
59.	Vladimír Munzar	SPŠ Rožnov	1	1							0,0	5,8
60. – 62.	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
63.	Robert Brunetto	SPŠMasaryk	2	1							0,0	4,3
64.	Jiří Keresteš	ZŠKostelní	0	2							0,0	4,2
65.	Jakub Loucký	G Písek	3	1							0,0	3,5

Milí řešitelé!

Touto opravenou pátou sérií jsme zakončili osmnnáctý ročník našeho semináře. Ale přeci ne úplně, na podzim b soustředění pro naše nejlepší řešitele. Soustředení se letos bude konat nejspíš v termínu 14. až 21. října a pozven přibližně třicet řešitelů z první čtyřicítky až padesátky.

Ještě bychom Vás chtěli poprosit: myslíme si, že je škoda, že se našeho semináře účastní tak málo řešitelů. Byl rádi, pokud byste nám na začátku příštího roku pomohli s „reklamou“ – například pověšením zadání první série nástěnku, ...

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

Vzorová řešení páté série osmnáctého ročníku KSP

18-5-1 Účetní

Řešení došlo opravdu hodně a většina jich byla i správně. Finančnímu úřadu tedy nezbývá než doufat, že se z vás stanou programátoři, a ne třeba ... účetní. Pro ty z vás, kterým se úlohu náhodou vyřešit nepodařilo, posíláme řešení vzorové, abyste měli v kriminále o čem přemýšlet :)

Vězte, že k získání maxima bodů nestačil pouze správný popis řešení. Nezbytnou součástí musel být i důkaz, aby vám účetní věřili, že je nechcete podfouknout.

Situace pro dva účetní je velice jednoduchá. První rozdělí majetek na dvě poloviny podle svého mínění. Druhý si vybere polovinu, která se mu zdá větší. První je spokojený, protože dostal přesně polovinu majetku. Druhý je taktéž spokojený, protože si vybral větší díl ze dvou, takže má alespoň polovinu majetku (podle jeho mínění).

Ve třech je situace o trochu komplikovanější, protože musíme rozebrat několik případů. Dokonce existuje několik řešení, jak účetní spravedlivě podělit. Nyní si ukážeme jedno z nich: První rozdělí majetek na tři díly, o kterých tvrdí, že jsou to přesně třetiny (označím je A, B a C). Nyní druhý a třetí ukáží na hromádku, která se jim zdá největší. Zde mohou nastat dva případy. Buď oba ukázali na různé hromádky (řekněme, že 2. ukázal na B a 3. ukázal na C). Pak si tyto hromádky nechají a prvnímu zbude poslední hromádka (tj. A). Všichni budou spokojeni, protože 2. a 3. si myslí, že mají největší hromádku ze tří (což je určitě víc než 1/3 a 1. dostane právě třetinu (sám si to tak rozdělil). Komplikace nastane, pokud 2. a 3. ukáží na stejnou hromádku (řekněme na A). V takovém případě si tuto hromádku rozdělí půl na půl (předchozím algoritmem) a znovu ukážou na hromádku, která se jim zdá největší (ze zbývajících dvou - B a C). I zde mohou nastat dva případy. Pokud oba ukáží na stejnou hromádku (řekněme B), tak si ji rozdělí „fifty-fifty“ a poslední hromádku C nechají prvnímu. První bude spokojen určitě, protože je mu jedno, kterou hromádku dostane. Druhý a třetí budou spokojeni rovněž, neboť si mezi sebe spravedlivě rozdělili dvě „největší“ hromádky (lze jednoduše nahlédnout, že tyto hromádky obsahují alespoň 2/3 majetku). Druhá možnost je, že při druhém výběru si každý vybere jinou hromádku (řekněme 2. vybere B a 3. vybere C). V takovém případě se oba podělí o zvolenou hromádku s prvním účetním (algoritmem dělení pro dva). Opět budou všichni spokojeni. První dostal dvakrát alespoň 1/2 z 1/3, takže má nejméně $2 \cdot 1/2 \cdot 1/3 = 1/3$. Druhý a třetí dostali každý alespoň polovinu z dvou (podle nich) největších hromádek, takže mají každý alespoň 1/3.

Abychom si ukázali i jiný algoritmus na dělení, budeme řešit úlohu N účetních trochu jinak. Řešení i důkaz bude založené na rekurzi a matematické indukci. Předpokládejme, že umíme vyřešit úlohu pro $N-1$ účetních ji rozšířit na N účetních. Vezmeme prvního účetn chvíli jej postavíme stranou. Teď necháme zbývajíc účetních, aby si rozdělili spravedlivě všechen majo sebe. Každý účetní kromě prvního má tedy svoji h (označme je $H_2, H_3 ... H_N$) o které si myslí, že obsa spoň $1/(N-1)$ majetku. Nyní poprosíme jednotliv aby každý rozdělil svoji hromádku na N stejných mádek. První účetní pak obejde všechny své kol každého si vezme jednu podhromádku. Všichni by spokojeni. První dostal z každé hromádky alesp tinu. O jednotlivých hromádkách toho z pohledu účetního moc tvrdit nemůžeme, ale víme, že jeji je 1 (dohromady tvoří celý majetek):

$$1/N \cdot H_2 + 1/N \cdot H_3 + \ldots + H_N = 1/N \cdot (H_2 + H_3 + \ldots$$
$$= 1/N \cdot 1 = 1/N$$

První má tedy alespoň $1/N$-tinu majetku. Ostatn být také spokojeni. Při počátečním dělení dostal k mádku, na které bylo (podle jeho mínění) alespoň majetku. Následně jej první obral přesně o $1/N$- hromádky, takže každému zbyla přesně $(N-1)/N$ ho, co už dostal. Ve výsledku každému zbylo $1/$ $(N-1)/N = 1/N$ majetku.

Všichni jsou spokojeni (snad až na finanční poli hurá na Seychely...

Martin „Bobřík“

18-5-2 Permutovat se musí legálně!

Pro „výzkumné“ účely si úlohu pozměňme: Chcen všechny permutace dané množiny lexikograficky Jak na to? Zafixujeme nejmenší prvek z množiny ku a za něj postupně připojíme všechny lexikografi dělené permutace zbývajících prvků. Poté zafixuje nejmenší prvek a postup zopakujeme, atd. Takt možné napsat rekurzivní funkci, která by vždy fix ky od nejnižšího k nejvyššímu (zafixovaný prvek jako prvek přilehlý) a pomocí sama sebe by vyg všechny permutace zbylých prvků (jejich množinu jako zbytek).

Jak nyní vyřešíme původní úlohu? Stačí najít zbyt v zadané permutaci právě dopermutoval, a k němu prvek. Nyní můžeme permutovat zbytek opět od s tím rozdílem, že pouze vyměníme přilehlý prvek ším vyšším prvkem z nalezeného zbytku (tedy to v modifikované úloze).

A implementace? Prvním úkolem je najít zbyt v zadané permutaci právě dopermutoval. To je né, neboť víme, že se nachází na konci permuta

dopermutovanost znamená, že zbytková posloupnost je sestupná a nejdelší možná. Začít permutovat zbytek od začátku je také jednoduché. Stačí si uvědomit, že jediné co je třeba udělat, je setřídit ji vzestupně. To bylo ale kamenem úrazu některých řešení, neboť posloupnost třídili některým z třídících algoritmů místo aby si všimli, že ze sestupné posloupnosti vytvoříme vzestupnou tak, že obrátíme pořadí jejích prvků.

Záměna přilehlého prvku je již triviální. Problém může nastat pouze tehdy, když žádný přilehlý prvek neexistuje. To se ale stane jenom tehdy, když jsme na vstupu dostali lexikograficky nejvyšší možnou permutaci.

Paměťová i časová složitost je $\mathcal{O}(N)$.

Jan Bulánek & Zbyněk Falt

18-5-3 Číňanské volby

Představme si následující „paralelní“ algoritmus na vyhodnocení voleb: každý Číňan si najde do dvojice nějakého Číňana, který chce volit někoho jiného. Pokud má některý kandidát nadpoloviční většinu hlasů, někteří z jeho příznivců zůstanou nespárovaní; u takto nalezeného potenciálního vítěze si ještě ověříme, zda skutečně vyhrál (spočítáme si počet jeho hlasů).

Druhou část tohoto algoritmu jistě zvládneme v lineárním čase a s konstantní pamětí, zbývá si rozmyslet, jak realizovat tu první. Zjevně nás nezajímá, jak jsou Číňané spárováni, stačí nám vědět, kolik jich zůstane nespárovaných a pro koho nespárovaní hlasují. Budeme si tedy udržovat dvě čísla – K (číslo kandidáta, pro nějž hlasují nespárovaní Číňané), a C (počet nespárovaných Číňanů). Postupně procházíme všechny Číňany. Pokud aktuální Číňan hlasuje pro kandidáta K, nejde ho spárovat, a proto zvýšíme C o jedna. Pokud hlasuje pro někoho jiného (kandidáta X), rozlišíme dva případy: jestliže je $C > 0$, pak tohoto voliče spárujeme s jedním z voličů kandidáta K, tedy snížíme C o jedna. Jestliže je $C = 0$, nelze aktuálního Číňana spárovat, a proto dosadíme $K = X$ a $C = 1$.

Tento algoritmus používá konstantní množství paměti a seznam hlasů projde dvakrát, časová složitost je tedy $\mathcal{O}(N)$. Pokud bychom chtěli být přesnější, je třeba vzít do úvahy to, že na uložení čísel potřebujeme $\mathcal{O}(\log N)$ bitů, což pro velká N nelze zanedbat tedy paměťová složitost je $\mathcal{O}(\log N)$ a časová $\mathcal{O}(N \log N)$.

Zdeněk Dvořák

18-5-4 Detektýv

S důkladností takřka šerlokovskou prozkoumáme několik možných řešení, až usvědčíme to nejrychlejší. Označme si (věrni písmenkům ze zadání) N délku stopovaného řetězce, k počet podezřelých sekvencí, $p_1, \ldots, p_k$ délky těchto sekvencí a $P = p_1 + \ldots + p_k$ jejich celkovou délku.

0. pokus (jak by ho vymyslel strážník Vopička): Budeme hledat každou sekvenci zvlášť, a to tak, že si po vstupu „pojedeme okénkem“ délky p_i a vždy porovnáme, jestli se okénko rovná i-té sekvenci. Kdybychom si okénko ukládali jako cyklické pole, zvládli bychom ho posunout v konstantním čase, ale stejně nás nemine čas $O(p_i)$ na porovnání. Celkově trvá $O(Np_1 + \ldots + Np_k) = O(NP)$ a navíc potřebujeme k-krát volat rewind.

1. pokus (inspektor Neverley): Damned, na hledání výskytů jednoho řetězce přeci můžeme použít algoritmus KMP z té vaší cookbook, takže jeden průchod zvládneme v timu $O(N + p_i)$, celkově tedy $O(Nk + P)$ s k rewindy. That's it.

2. pokus (policejní rada Žák): V kuchařce je přeci i algoritmus A-McC na hledání výskytů více slov najednou. Stačí, když hlášení výskytu nahradíme připočtením jedničky k počítadlu. (Na to praktikant Hlaváček:) Dobrý plán, pane rado, ale má jedno háčisko jak na sumce: jelikož se sekvence mohou překrývat, může jich v jednom místě končit až k, takže jsme opět na $O(Nk + P)$, i když tentokrát bez rewindů.

3. pokus (Šérlok osobně): Postavíme si vyhledávací automat jako v minulém pokusu, ale místo abychom počítali rovnou výskyty, budeme si pamatovat jen to, kolikrát jsme prošli kterým stavem, a pak z toho výskyty dopočítáme. Well, ale jak?

Pokud máme nějaký stav α (o kterém víme, že je prefixem některého z vyhledávaných slov, takže mimo jiné mezi stavy najdeme všechny sekvence stop, které počítáme) a chceme zjistit, kolikrát se slovo α v textu vyskytlo, stačí sečíst počet průchodů tímto stavem a všemi dalšími stavy, které končí na α, což jsou přesně ty, ze kterých se do α lze dostat pomocí zpětné funkce (případně zavolané vícekrát).

Stačí tedy projít automat v opačném pořadí, než ve kterém jsme vytvářeli zpětnou funkci (nejlepší bude si během konstrukce automatu toto pořadí zapamatovat, třeba v poli, v němž jsme měli uloženu frontu). Pro každý stav α pak přičteme počítadlo odpovídající tomuto stavu k počítadlu stavu, do nějž vede z α zpětná funkce. (To se pak přičte podle další zpětné funkce atd., takže počítadlo stavu α se opravdu postupně popřičítá ke všem rozšířením stavu α.)

To vše zvládneme v čase $O(P + N + P)$ (konstrukce automatu + průchod textem + dopočítání), čili $O(P + N)$, a v paměti $O(P + N)$, bez jediného zavolání rewindu.

Program obnáší připsání cca čtyř řádků ke zdrojáku z kuchařky. Abychom z toho nevybruslili tak snadno, ukážeme si trochu jinou implementaci v Céčku.

It's a lemon tree, my dear Watson!

Martin Mareš

P.S.: V kuchařce si, nečekán, nezván, opět zařádil šotek a trochu pomíchal vypisování nalezených slov. Opravenou verzi naleznete na webu, rdícího se kuchaře opodál.

18-5-5 Do vysokých kruhů

Nejprve bylo potřeba oblasti převést na objekty, se kterými umíme manipulovat rozumněji než s obecnými množinami bodů v rovině. Velmi užitečné je představit si protínající se kružnice jako graf s průsečíky a dotyky kružnic jako vrcholy. Hrany budou oblouky mezi sousedními vrcholy. Tento graf je vlastně multigraf, což je graf, ve kterém může mezi dvěma vrcholy vést více než jedna hrana a z jednoho vrcholu do toho samého může vést více než jedna smyčka. Takový graf je určitě jednoznačně zadán polohami a poloměry kružnic a je rovinný (původní rozmístění kružnic je jeho rovinné nakreslení). Bohužel se nám do něj nijak nepromítnou izolované kružnice, ty je třeba ošetřit jinak.

Nyní se nám z na první pohled neuchopitelného problému stal problém mnohem jednodušší – spočítat stěny rovinného grafu. K tomu se ideálně hodí Eulerova věta:

$$V + F = K + E + 1$$

Toto je vztah mezi počtem vrcholů (V), stěn (včetně té vnější) (F), komponent souvislosti (K) a hran (E). Tato

Úloha 18-5-5 – Do vysokých kruhů – program

```
#include <stdio.h>
#include <math.h>

#define SQR(x) ((x)*(x))                                        // Druhá mocnina
#define EPSILON (4.2e-10)                                       // Podovnání s tolerancí
#define EQ(x,y) (((x)+EPSILON>(y))&&((x)-EPSILON<(y)))

#define MAX_N 10000
int N;                                                          // Počet kružnic

double x[N], y[N], r[N];                                        // Kružnice
int byl[N];                                                     // Navštíveno?
int v=0;                                                        // Vrcholy (průsečíky)
int k=0;                                                        // Komponenty

int pruseciku(int a, int b)                                     // Kolik je mezi a a b průsečíků?
{
  double d=sqrt(SQR(x[a]-x[b])+SQR(y[a]-y[b]));                 // Vzdálenost středů
  if (d>r[a]+r[b]) return 0;                                    // Úplně mimo
  if ((d+r[a]<r[b])||(d+r[b]<r[a])) return 0;                   // Jedna ve druhé
  if (EQ(d,r[a]+r[b])) return 1;                                // Vnější dotyk
  if ((EQ(d+r[a],r[b]))||(EQ(d+r[b],r[a]))) return 1;           // Vnitřní dotyk
  return 2;                                                     // Jinak mají právě dva průsečíky
}

void navstiv(int koho)                                          // Rekurze pro průchod komponent
{
  int i,p;

  byl[koho]=1;

  for (i=0;i<N;i++) {                                           // Teď projdi všechny sousedy
    p=pruseciku(i,koho);
    if ((i!=koho) && (p>0)) {
      if (i<koho) v+=p;                                         // Připočti průsečíky (ale ne dvakr
      if (!byl[i]) navstiv(i);                                  // Navštiv
    }
  }
}

int main()
{
  int i;
  scanf("%d",&N);                                               // Načteme
  for (i=0;i<N;i++) {
    scanf("%lf %lf %lf",&(x[i]),&(y[i]),&(r[i]));
    byl[i]=0;
  }
  for (i=0;i<N;i++)
    if (!byl[i]) {                                              // V této komponentě jsme ještě neby
      k++;
      navstiv(i);
    }

  printf("Oblastí: %d\n",k+v+1);
  return 0;
}
```

```
void count(void)                                // propagování počtů po zpětných hranách
{
  for (struct state *s=tail; s != &root; s=s->qprev) {
    s->back->count += s->count;
    if (s->out)                                 // ... a vypisování četností slov
      printf("%6d %s\n", s->count, s->out);
  }
}

int main(int argc, char **argv)                 // main sweet main :)
{
  for (int i=1; i<argc; i++)
    insert(argv[i]);
  build();
  run();
  count();
  return 0;
}
```


věta platí pro rovinné grafy a platí i pro multigrafy, pokud si zvolím, že mezi „rovnoběžnými“ násobnými hranami jsou také stěny a že smyčka přidává jednu stěnu. Toto rozšíření přesně odpovídá naší představě toho, jak kružnice dělí rovinu na oblasti.

Věta se dokazuje indukcí podle složitosti grafu. Pro prázdný graf určitě platí $(0 + 1 = 0 + 0 + 1)$. Přidáme-li nový vrchol, stoupnou V i K o jedna a rovnost zůstane zachována. Přidáme-li hranu a zvýšíme tak E o jedna, pak jsme buď spojili dvě komponenty souvislosti a snížili K o jedna, nebo přidali jednu stěnu rozdělením nějaké existující na právě dvě. V obou případech zůstane rovnost zachována a druhý případ navíc zahrnuje přidávání násobných hran a smyček. Každý rovinný (multi)graf lze postavit z prázdného přidáváním vrcholů a hran, takže pro něj věta musí platit.

Stačilo by tedy spočítat počet komponent, hran a průsečíků. Víme, že na každé kružnici je stejně vrcholů a hran. Vrchol je ale sdílen mezi dvěma kružnicemi, zatímco hrana patří právě jedné. Jinak řečeno je stupeň každého vrcholu 4. Z toho plyne, že $E = 2V$. Tedy:

$$F = K + 2V + 1 - V = K + V + 1.$$

Tento vzorec nám navíc zahrne i izolované kružnice, počítáme-li je jako jednu komponentu bez průsečíků. To nám trochu zjednoduší algoritmus.

Stačí tedy spočítat počet komponent a průsečíků, obojí zvládneme v čase $\mathcal{O}(N^2)$ průchodem do hloubky (s hledáním sousedů vyzkoušením všech) a vyzkoušením všech dvojic. Zkoušení dvojic navíc zahrneme do toho průchodu. V programu je průchod do hloubky realizován rekurzivní funkcí `navstiv()`. Ta bude pro každý vrchol spuštěna určitě právě jednou, určitě projde celou komponentu a zároveň správně napočítá počet průsečíků. Jen je si třeba dát pozor, abychom nezapočítávali průsečíky dvakrát (za páry kružnic (k_i, k_j) a (k_j, k_i)).

Toto řešení má časovou složitost $\mathcal{O}(N^2)$, paměťovou $\mathcal{O}(N)$. Existuje ještě jiné o dost složitější řešení používající *zametací čáru* k dosažení složitosti $\mathcal{O}((N+V)\log N)$, což je lepší než naše $\mathcal{O}(N^2)$, pokud je počet průsečíků $V < N^2/\log N$, tedy pro dost „řídké“ konfigurace kružnic. Pro $V = \mathcal{O}(N^2)$ má ale časovou složitost až $\mathcal{O}(N^2 \log N)$. Paměťová složitost tohoto algoritmu je $\mathcal{O}(N)$. Jeho popis by ale byl dost komplikovaný a proto ho neuvádím.

Tomáš Gavenčiak

18-5-6 Proφlování

Úloha 1: Buď G graf, který vznikne z CFG tak, že
neme na orientaci hran, přidáme hranu mezi vstup
stupním blokem CFG, a zahodíme hrany, na kter
čítač. Graf G nemůže obsahovat cyklus – počty
hran na odpovídajícím cyklu (nebo cestě) v CFG
možné zvyšovat a snižovat bez toho, že bychom
libovolný čítač, tedy by nebylo možné pouze z čí
profil. G je tedy les, a má nejvýše $N - 1$ hran,
počet bloků v programu. Proto v původním CFG
vynechat čítač na nejvýše $N - 2$ hranách.

Naopak, pokud vynecháme čítač na libovolných N
nách takových, aby G byl strom, dokážeme počty
zbývajících hran dopočítat: protože G je strom, n
stupně jedna (vede do něj jen jedna hrana e). To
bloku, u nějž neznáme počty provedení pouze jedn
které do něj vstupují nebo z něj vystupují. Souč
provedení hran vstupujících do bloku je roven souč
provedení hran, které z něj vystupují, tedy dokáž
počítat počet provedení hrany e. Hranu e vyhodí
tento postup opakujeme, dokud neurčíme počty
všech hran. Časová i paměťová složitost tohoto a
je lineární.

Úloha 2: Zjevně stačí spočítat počty provedení bl
kud hrana vychází z bloku, který se provede n-kr
vedeme ji s pravděpodobností p, pak výpočet tou
projde pn-krát. Mějme blok b, do nějž vstupují
$e_2, \ldots, e_k$, které vycházejí z bloků $b_1, b_2, \ldots, b_k$ s p
dobnostmi $p_1, p_2, \ldots, p_k$. Počet provedení bloku b
součtem počtu provedení hran, které do něj vstup
jestliže je blok b proveden n-krát a bloky b_i jsou p
n_i-krát, pak platí

$$n = \sum_{i=1}^{k} p_i n_i.$$

Pokud navíc položíme počet provedení vstupního
ven 1, dostáváme soustavu lineárních rovnic, jejím
je hledaný profil (samozřejmě, kdybychom chtěli
la přesní, je třeba ještě dokázat, že tato soustava
noznačné řešení). Časová složitost závisí na tom,
soustavu budeme řešit. Pokud použijeme Gausso
naci, dostáváme kubickou časovou složitost, což j
příliš mnoho. Proto se většinou používají algori
ré využívají toho, že CFG má speciální struktur
se nepoužije příkaz skoku `goto`, každý cyklus má
den vstup. V případě, že `goto` použijeme a tuto p
porušíme, tyto algoritmy vrátí pouze přibližné řeš
však fungují v lineárním čase.

Zdeně

Úloha 18-5-2 – Permutovat se musí legálně! – program

```
program Permutace;

procedure swap(var a, b : char);
var
   p : char;
begin
   p := a; a := b; b := p;
end;

var
   perm : string;
   N, i, j : integer;
begin
   readln(perm);
   N := length(perm);

   i := N;
   while (i > 1) and (perm[i-1] > perm[i]) do
      dec(i);                                  { Hledáme zbytek }
   if i > 1 then begin
      for j := 0 to (N-i-1) div 2 do
         swap(perm[j+i], perm[N-j]);           { Setřídíme zbytek }

      j := i;
      dec(i);
      while (perm[i] > perm[j]) do             { Najdeme nejbližší vyšší prvek než prvek přilehlý }
         inc(j);
      swap(perm[i], perm[j]);                  { Zaměníme je }
      writeln(perm);
   end else                                    { Pokud takový neexistuje, permutace byla poslední }
      writeln('Dopermutovali jsme');
end.
```

Úloha 18-5-3 – Čínanské volby – program

```
program Cina;

const N = 100;
var hlasy : array[1..N] of integer;
    i, x, k, c : integer;

begin
  k := 0;                                      { Najdeme kandidáta. }
  c := 0;
  for i := 1 to N do
    begin
      readln (x);
      hlasy[i] := x;
      if c = 0 then k := x;
      if k = x then inc (c) else dec (c);
    end;

  c := 0;                                      { Ověříme, zda vyhrál. }
  for i := 1 to N do
    begin
      x := hlasy[i];
      if k = x then inc (c);
    end;

  if 2 * c > N then
    writeln ('Vyhrál kandidát číslo ', k, '.')
  else
    writeln ('Nikdo nevyhrál.')
end.
```


Úloha 18-5-4 – Detektýv – program

```
/* Counting words. MM scribebat me per III Id. Mai. MMDCCLIX AUC. */

#include <stdio.h>
#include <stdlib.h>

struct state {                                 // jeden stav automatu
  struct state *fwd[256], *back;               // hrany dopředné a zpětná
  struct state *qnext, *qprev;                 // předchůdce a následník ve frontě
  char *out;                                   // které slovo v tomto stavu končí
  int count;                                   // počet průchodů stavem
};

struct state root;                             // počáteční stav (kořen stromu)
struct state *head, *tail;                     // první a poslední ve frontě

struct state *step(struct state *s, int c)     // jeden krok automatu
{                                              //   (včetně vracení se po zpětných hranách)
  while (s) {
    if (s->fwd[c])                             // hrr na ně!
      return s->fwd[c];
    s = s->back;                               // a když to nejde, tož cófnem
  }
  return &root;
}

void insert(char *x)                           // vložení jednoho slova do stromu
{
  struct state *s = &root;
  int c;
  for (char *y = x; c=*y++; ) {
    if (!s->fwd[c])
      s->fwd[c] = calloc(1, sizeof(struct state));
    s = s->fwd[c];
  }
  s->out = x;
}

void build(void)                               // konstrukce zpětných hran přesně podle kuchař
{
  struct state *s;
  head = tail = &root;
  while (head) {
    for (int c=0; c<256; c++)
      if (s = head->fwd[c]) {
        s->back = step(head->back, c);
        tail->qnext = s;
        s->qprev = tail;
        tail = s;
      }
    head = head->qnext;
  }
}

void run(void)                                 // projití celého vstupu s počítáním průchodů s
{
  struct state *s = &root;
  for (int c; (c = getchar()) != EOF;) {
    s = step(s, c);
    s->count++;
  }
}
```