Stabilní párování

- Instance: N pánů a N dam. Každá postava má seznam "přijatelnosti"(všech) příslušníků opačného pohlaví. Problém: Vytvořte N koedukovaných dvojic, aby vzniklé párování bylo stabilní.
- Párování je stabilní, pokud neexistuje dvojice $I\iota$ tak, že $I\kappa$ a $K\iota$, ale kdybychom je "přepojili"na $I\iota$ a $K\kappa$, jak I, tak ι by si polepšili.
- Ne zcela triviální algoritmus minule velmi vágně naznačený, konečnost lze dokázat snadno.

Stabilní párování

- "Pánové, zadejte se!"
- Pánové vyrazí za dámami na "prvním" místě.
- Dáma si vybere mezi současným a nově příchozími nejlepšího na seznamu.
- "Odmítnutí" pokračují postupně k dalším a dalším dle seznamu.
- Proč je algoritmus konečný?
- Proč je nalezené párování stabilní?

Nelze dokázat silnější tvrzení?

Lze: *Párování nalezené algoritmem pánské volenky je mezi všemi stabilními párováními pro pány nejvýhodnější.*
To znamená: V žádném stabilním párování žádný z pánů nemůže mít "lepší" partnerku, než tu, kterou získá algoritmem pánské volenky.

Definition

Hříchem nazveme situaci, kdy ve volenkovém algoritmu *dáma odmítne pána*, kterého by mohla mít v *nějakém* stabilním párování.

Dokážeme, že v algoritmu pánské volenky nenastane hřích.

Lemma

V algoritmu pánské volenky nenastane hřích.

Důkaz.

Sporem: Nenastane první hřích – tedy nechť nastane.

- První tedy zhřešila *Eva.* Odmítla *Adama*, kterého mohla mít a pojala jistého *Žibřida*, který v párování získaném nevolenkovým algoritmem má jistou *Kunhutu.*
- Jenže v jakém vztahu jsou zúčastnění?
- A co na to volenkový algoritmus?
- Je Žibřid lepší nebo horší!?

□

Rozklad na prvočinitele

- Nápady
- Naivní algoritmus: Hledej postupně prvočísla až do n a zkoušej jimi dělit.
- Pokus o méně naivní algoritmus: `Hledej prvočísla do` $\sqrt{n_i}$ `(kde` n_i `je hodnota, kterou zbývá faktorizovat).` nebude fungovat. Proč?
- Ještě méně naivní algoritmus: Zkoušej všechna čísla až do $n/2$. Proč toto funguje?

K Eratosthenovu sítu

Jak nagenerovat všechna prvočísla menší než n?

- Naivní algoritmus (generuj a testuj),
- Méně naivní algoritmus generuj a zkoušej dělit jen již nagenerovanými prvočísly.
- Eratosthenes: Nageneruj čísla $2 \dots n$. Pro i od 2 do $\sqrt{n}$ pokud je i prvočíslo, proškrtej všechny jeho násobky.

Prohledávání grafu

- Motivace: Mínótaurus se v bludišti živí Athéňany.
  Mezi těmi se objevil princ Théseus, který Mínótaura zabil.
- Algoritmus:
  1. Najdi Mínótaura,
  2. Zabij Mínótaura.
- Druhou část mu ponecháme, zajímavá je část první.

Prohledávání grafu - hledání do hloubky

- Théseus dostal od Ariadny nit,
- ovšem buďto použil randomizovaný algoritmus, nebo dostal ještě kyblík s barvou.
- Algoritmus (zajímavé jen křižovatky):
  Pokud jsme ještě nenašli Mínótaura:
  - Pokud existuje neobarvená chodba (kterou nevede nit), obarvi tuto chodbu (začátek a konec) a projdi jí.
  - Jinak namotej nit (vrať se na předchozí křižovatku).

  Namotávej nit, dokud nevidíš Ariadne.

Proč je algoritmus správný? Konečnost? [trik s čísly]
Parciální správnost? Invarianty? [sled k Ariadne]
Každou chodbou projdeme nejvýše 2x.
Nexistuje vrchol, který navštívit můžeme a nenavštívíme.

Jiný algoritmus:

Dokud nejsme u Mínótaura:

- Pokud existují dvě obarvené chodby, kterými vede niť, namotej niť.
- jinak pokud existuje neobarvená chodba, obarvi ji a projdi jí.
- Jinak namotej niť.

Dokud nejsme u Ariadne:

- namotej niť.

Lemma

I tento algoritmus je správně, protože namotáváme-li, ačkoliv můžeme ještě kupředu, znamená to, že se do tohoto vrcholu ještě vrátíme. Navíc invariant o sledu k Ariadne lze posílit na cestu *k Ariadne.*

Poznámky

- Jedná se o algoritmus prohledávání do hloubky, kdy postupujeme dále, dokud to jde.
- K prohledávání grafů existuje i algoritmus prohledávání do šířky zvaný *algoritmus vlny.*
- Algoritmus vlny: Do bludiště vyrazí neomezený počet bojovníků, kteří se bludištěm šíří jako povodeň. Používáme kupříkladu, pokud chceme najít nejkratší cestu (příklady budou později).

Zápis programů

Při programování (zápisu algoritmů):

- pracujeme s proměnnými různých typů (a s konstantami),
- modifikujeme obsahy proměnných,
- voláme podprocedury,
- porovnáváme obsahy proměnných,
- rozhodujeme se podle toho,
- cyklíme,
- čteme vstup, vypisujeme výstup.

Vzhled programu v Pascalu

Příklad:

```
program nanic;
const   x=10;
        text='deset';
var a,b:integer;
```

Program začíná vždy klíčovým slovem `program`!
Jednotlivé příkazy oddělujeme středníkem!

Následuje sekce definice konstant uvedená slovem `const`.
Konstanty přiřazujeme: `konstanta = hodnota`
Následuje definice proměnných uvedená slovem `var`.
Definujeme celočíselné proměnné `a` a `b`.

Důležitost indentace:

```
program nanic; const x=10; text='deset'; var
a,b:integer; c:string;
begin write('Napis cislo: '); readln(a);
write('Napis dalsi cislo: '); readln(b);
writeln('Soucet je ',a+b); writeln(x,' je ',text);
end.
```