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.

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.

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!”

Stabilní párování

- "Pánové, zadejte se!"
- Pánové vyrazí za dámami na "prvním" místě.

Stabilní párování

- "Pánové, zadejte se!"
- Pánové vyrazí za dámami na "prvním" místě.
- Dáma si vybere mezi současným a nově příchozími nejlepšího na seznamu.

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.

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?

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!?

□

Prohledávání grafu

- Motivace: Mínótaurus se v bludišti živí Athéňany.
  Mezi těmi se objevil princ Théseus, který Mínótaura zabil.

Prohledávání grafu

- Motivace: Mínótaurus se v bludišti živí Athéňany.
  Mezi těmi se objevil princ Théseus, který Mínótaura zabil.
- Algoritmus:

Prohledávání grafu

- Motivace: Mínótaurus se v bludišti živí Athéňany.
  Mezi těmi se objevil princ Théseus, který Mínótaura zabil.
- Algoritmus:
  1. Najdi Mínótaura,

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.

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 niť), obarvi tuto chodbu (začátek a konec) a projdi jí.
  - Jinak namotej niť (vrať se na předchozí křižovatku).

  Namotávej niť, dokud nevidíš Ariadne.

Proč je algoritmus správný? Konečnost? [trik s čísly]
Parciální správnost? Invarianty? [sled k Ariadne]

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 niť), obarvi tuto chodbu (začátek a konec) a projdi jí.
  - Jinak namotej niť (vrať se na předchozí křižovatku).

  Namotávej niť, 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;
    c:string;
begin
      write('Napis cislo:');
      readln(a);
      write('Napis dalsi cislo:');
      readln(b);
      writeln('Soucet je ',a+b);
      writeln(x,' je ',text);
end.
```

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

Následuje sekce definice konstant uvedená slovem `const`.
Konstanty přiřazujeme:
`konstanta = hodnota`

Vzhled programu v Pascalu

Příklad:

```
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.
```

Následuje definice proměnných uvedená slovem `var`. Definujeme celočíselné proměnné `a` a `b`.

Za definicí (globálních) proměnných následují definice funkcí a procedur (prázdné) a tělo hlavního programu `begin ... end`. Indentace zlepšuje estetický dojem.

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.
```

Proměnné a jejich typy:

Proměnná má stanovený typ. Datové typy (v Pascalu) jsou:

- **byte:** 0 .. 255 (celá čísla),
- **integer:** $-32\,768$ .. $32\,768$,
- **longint:** -2^31 .. 2^31,
- **real:** -10^{38} .. 10^{38} (necelá čísla),
- **word:** 0 .. 65 535 (celá čísla),
- **char:** znaky (jeden znak 8-bitového ASCII),
- **string:** řetězec znaků (text) o délce až 255 znaků,
- **boolean:** pravda nebo lež (nabývá hodnot `true` a `false`).

Výrazy aritmetické:

- $+$ Sčítání,
- $-$ odčítání,
- $*$ násobení,
- $/$ dělení,
- závorky,
- `div` celočíselné dělení,
- `mod` zbytek po dělení.

Pozor na priority!
Pozor, `div` a `mod` má prioritu mezi sčítáním a násobením!

Pozor na sčítání stringů!

Příklad:
$(a + 5) * 17 + (b \bmod c)$

Přiřazovací příkaz: `:=`
Příklad: `x:= 2*y;`

Relační operátory

- $<$ je menší než (kupř. a¡b),
- $>$ je větší než,
- $>=$ je větší nebo rovno,
- $<=$ je menší nebo rovno,
- $<>$ nerovná se,
- $=$ rovná se (porovnání na rovnost).

Podmínky

Syntax (a sémantika):

- `if` podminka `then` prikaz;
- `if` podminka `then` `begin` blok prikazu `end`;
- `if` podminka `then` prikaz `else` prikaz;
  Pozor, před `else` se středník nepíše!
- `if` podminka `then` `begin` blok `end` `else` `begin` blok `end`;

Příklad:

```
if teplota>25 then
          writeln('Jdu do hostince!');
```

Příklad

```
if teplota>25 then writeln('Jdu do hostince!');
```

```
if teplota>25 then
begin
        writeln('Jdu do hostince!');
end
else
begin
        writeln('Nejdu nikam!');
end;
```

Cykly

- `while` podminka `do` prikaz nebo blok;
  Opakuj, dokud je `podminka` splněna.
- `for i:=1 to 10 do` prikaz nebo blok;
  Opakuj pro každou hodnotu proměnné od první do druhé meze.
- `for i:=100 downto 1 do` prikaz nebo blok;
- `repeat` prikazy; `until podminka;`
  Opakuj, dokud je `podminka` **nesplněna!**

Příklad:

```
program dvojkova;
var a:integer;
begin
        readln(a);
        while a > 0 do
        begin
                if a mod 2 = 1 then
                        write(1)
                else    write(0);
                a:=a div 2;
        end;
end.
```

Příklad vylepšený

Při programování je hlavní **myslet, jinak si** často zbytečně **přidělámе práci!**

```
program dvojk;
var a:integer;
begin
        readln(a);
        while a > 0 do
        begin
                write(a mod 2);
                a:=a div 2;
        end;
end.
```

Příklad, hledání prvočíselného rozkladu:

```
program rozklad;
var a,i:integer;
begin
        i:=2;
        readln(a);
        while i <= a do
        begin   if (a div i)*i = a then
                begin
                        write(i);
                        a:=a div i;
                end
                else    i:=i+1;
        end;
end.
```

Příklad, hledání prvočíselného rozkladu vylepšený:

```
program rozklad;
var a,i:integer;    opakujeme:boolean;
begin   i:=2;   opakujeme:=false;
        readln(a);
        while i <= a do
        begin   if (a div i)*i = a then
                begin   if opakujeme then
                                write('*')
                        else   opakujeme:=true;
                        write(i);
                        a:=a div i;
                end  else  i:=i+1;
        end;
end.
```