Fórum studentů MFF UK

Fórum pro všechny studenty matematicko-fyzikální fakulty UK, informatiky, fyziky i matematiky

Přejít na obsah

Hledej… Hledat
Pokročilé hledání

- Obsah fóra ‹ Informatika LS ‹ Výuka LS 2. ročník ‹ PRG005 Neprocedurální programování
- Změnit velikost textu
- Napsat e-mail
- Verze pro tisk

- FAQ
- Registrovat
- Přihlásit se

Zkouška 6. 6. 2017

Odeslat odpověď

Hledat v tomto tématu Hledat

Příspěvek: 1 • Stránka **1** z **1**

- Ohlásit tento příspěvek
- Odpovědět s citací

Zkouška 6. 6. 2017

od **pienieczek** » 9. 6. 2017 12:31

Malé příklady:

1. (Prolog) Na vstupu máme graf reprezentovaný jako

Kód: Vybrat vše
```
graf(SeznamVrcholu, SeznamHran)
```

(bylo ale dovoleno si reprezentaci grafu změnit) a číslo N. Máme určit, jestli v grafu existuje cyklus délky alespoň N. Pokud ano, program alespoň jeden takový cyklus vypíše, pokud ne, vrátí fail.
Pozn.: Problém je NP-úplný, tzn. očekává se řešení typu hrubá síla.

2. (Prolog) Na vstupu je seznam binárních proměnných a jejich hodnot (true,false). Máme vytvořit všechny takové seznamy, že se od původního liší v hodnotě alespoň 2 proměnných, a vrátit je všechny najednou v jednom seznamu.
Příklad:

Kód: Vybrat vše

```
aspon2([x1-true,x2-false,x3-true],X).
X = [[x1-false, x2-true, x3-true], [x1-false, x2-false, x3-false], [x1-true, x2-true, x3-
false], [x1-false, x2-true, x3-false]]
```

3. (Haskell) Na vstupu máme text a číslo n. Úkolem je vybrat z textu všechna slova delší rovna n, lexikograficky je setřídit a vrátit jako výsledek nějakou datovou strukturu, ve které bude pro každé takové slovo uložený počet jeho výskytů.
a) Navrhněte datovou strukturu pro reprezentaci oboru hodnot. (Očekávalo se, že napíšete definici s type, tzn. např.

Kód: Vybrat vše
```
type Stats = [(String,Int)]
```

b) Napište typovou definici funkce.
c) Napište funkci.
Slova jsou oddělená neprázdnou posloupností znaků ' ', '\t', '
'.

4. (Haskell) Máme dva druhy stromů - obecný n-ární:

Kód: Vybrat vše
```
data NTree a = N2 a [NTree a]
```

a n-ární, ve kterém je řečeno, které podstromy jsou vlevo a které vpravo:

Kód: Vybrat vše
```
data UspTree a = UT [UspTree a] a [UspTree a]
```

Máme napsat funkci, která obecný n-ární strom převede na uspořádaný strom. V každém uzlu obecného n-árního stromu na vstupu je kromě hodnoty uložený taky počet synů, kteří jsou vlevo.

U všech příkladů jsme mohli předpokládat, že vstup je zadán korektně.

Moje řešení:
U ústní mi pan Dvořák vytýkal, že je kód místy neefektivní, ale celkově byl spokojený a dal mi za toto (resp. o něco horší, opravila jsem ještě potom pár chyb, aby byl kód vskutku funkční) řešení plný počet bodů.

Kód: Vybrat vše
```
% 1
cyklus(g([],_),_,_) :- fail.
cyklus(g([V|_],H),N,C) :- hrana(V,X,H), najdicyklus(X,V,H,[X-V],C), length(C,L), L>=N.
cyklus(g([V|Rcholy],H),N,C) :- delete(H,V-_,H1), delete(H1,_-V,H2),
cyklus(g(Rcholy,H2),N,C).

hrana(X,Y,H) :- member(X-Y,H).
hrana(X,Y,H) :- member(Y-X,H).

najdicyklus(O,K,H,A,C) :- hrana(O,X,H), X\=K, \+member(X-_,A), najdicyklus(X,K,H,[X-O|A],C).
najdicyklus(O,K,H,C,[K-O|C]) :- hrana(O,K,H), \+member(O-K,C).
```

Kód: Vybrat vše

```
% 2
aspon2(P,V) :- aspon(2,P,V).

aspon(N,P,V) :- length(P,M), M>=N,!, presne(N,P,V1), N1 is N+1, aspon(N1,P,V2),
append(V1,V2,V).
aspon(_,_,[]).

presne(N,[K-P|Ps],[[K-R|V1]]) :- length([K-P|Ps],M), M=N, !,switch(P,R), N1 is N-1,
presne(N1,Ps,[V1]).
presne(N,[K-P|Ps],V) :- N>0,switch(P,R), N1 is N-1, presne(N1,Ps,V1), pridej(K-R,V1,Vr1),
    presne(N,Ps,V2), pridej(K-P,V2,Vr2), append(Vr1,Vr2,V).
presne(0,X,[X]).

switch(true,false).
switch(false,true).

pridej(R,[S|Ss],[[R|S]|Xs]):-pridej(R,Ss,Xs).
pridej(_,[],[]).
```

Kód: Vybrat vše

```
-- 3
slova :: String -> Int -> [(String,Int)]
slova text n = sluc (mujSort (vseSlova text n))

vseSlova::String -> Int -> [String]
vseSlova [] _ = []
vseSlova text n | (length slovo)>=n = slovo:(vseSlova zbytek n)
                | otherwise = vseSlova zbytek n
    where (slovo,zbytek) = dejSlovo text

dejSlovo::String -> (String,String)
dejSlovo text = (takeWhile (not.white) (dropWhile white text), dropWhile (not.white)
(dropWhile white text))

white::Char -> Bool
white x | elem x [' ', '\t', '
'] = True
        | otherwise = False

mujSort::Ord a => [a] -> [a]
mujSort [] = []
mujSort (x:xs) = (mujSort mensi)++[x]++(mujSort vetsi)
    where mensi = filter (<=x) xs
          vetsi = filter (>x) xs

sluc::Eq a => [a] -> [(a,Int)]
sluc (x:xs) = sluc' x 1 xs

sluc'::Eq a => a -> Int -> [a] -> [(a,Int)]
sluc' x i [] = [(x,i)]
sluc' x i (y:ys) | x==y = sluc' x (i+1) ys
                 | otherwise = (x,i):(sluc' y 1 ys)
```

Kód: Vybrat vše

```
-- 4
data NTree a = N2 a [NTree a] deriving Show
data UspTree a = UT [UspTree a] a [UspTree a] deriving Show

preved:: NTree (a,Int) -> UspTree a
preved (N2 (x,_) []) = UT [] x []
preved (N2 (x,n) l) = UT (take n ll) x (drop n ll)
    where ll = map preved l
```

Pozn. U 3 jsem si nebyla jistá, jestli haskellí sort nevyhazuje duplicitní prvky (tak jako to dělá prologovský sort), proto jsem si raději napsala vlastní. Ale naštěstí se prý ta vestavěná funkce chová rozumně, tzn. stačilo by použít jenom sort.

Velký příklad - Ostrov pokladů
Na ostrově pokladů je N různě barevných truhel (několik truhel může mít stejnou barvu). V každé truhle je část pokladu a jeden klíč. Klíče také mají různé barvy (opět může být víc klíčů od jedné barvy) a platí, že truhlu můžete odemčít jenom klíčem, který má stejnou barvu. Navíc můžete každý klíč použít jenom jednou (poté, co s ním odemčete nějakou truhlu, zůstane vězet v zámku). Na začátku máme k dispozici jeden klíč. Na vstupu je barva počátečního klíče a seznam truhel - pro každou její barva a barva klíče, který se v ní nachází. Úkolem je zjistit, jestli existuje posloupnost odemykání truhel taková, že zvládneme odemknout všechny truhly, a pokud ano, nějakou takovou posloupnost vypsat.

Problém má efektivní (=polynomiální) řešení, na které jsem ovšem nepřišla. Popsala jsem tedy řešení exponenciální, které jsem stihla i celé nakódit, a stačilo to, pan Dvořák říkal, že nezkouší algoritmy, ale neprocedurální programování.

Nástin mého řešení:
Problém jsem si představila jako orientovaný graf, vrcholy jsou truhly, šipky vedou mezi vrcholem i a j, jestliže j-tou truhlu můžeme odemčít klíčem z i-té truhly, jeden další vrchol je počáteční klíč. Chceme najít cestu z počátečního vrcholu takovou, že každý vrchol navštívíme právě jednou (což je pochopitelně NP-úplné). Stačilo na to obyčejné DFS, psala jsem v Prologu.

Na ústní se mnou prošel malé příklady, podíval se, jak jsem řešila ten velký, zeptal se na pár věcí (jako třeba jak by se tahle funkce dala napsat efektivněji, nebo jak se dá v Prologu appendovat v konstantním čase - rozdílové seznamy).

pienieczek
Matfyz(ák|ačka) level I

Příspěvky: 2
Registrován: 9. 6. 2017 11:46
Typ studia: Informatika Bc.

Nahoru

Odeslat odpověď
Příspěvek: 1 • Stránka **1** z **1**

Zpět na PRG005 Neprocedurální programování

Přejít na: PRG005 Neprocedurální programování Přejít

Kdo je online

Uživatelé procházející toto fórum: Žádní registrovaní uživatelé a 1 návštěvník

- Obsah fóra
- Tým • Smazat všechny cookies z fóra • Všechny časy jsou v UTC + 1 hodina