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

10. 9. 2019 - Dvořák

Odeslat odpověď

Hledat v tomto tématu Hledat

Příspěvek: 1 • Stránka **1** z **1**

- Ohlásit tento příspěvek
- Odpovědět s citací

10. 9. 2019 - Dvořák

od **Quake** » 11. 9. 2019 14:58

Ahoj, tady je zadání písemné části zkoušky, nějaké postřehy k písemné i ústní části a má řešení s tím, jak byla hodnocena.
Malé příklady:
1. Prolog: Generování hodnot výrokových proměnných (5 bodů)
Definujte binární predikát aspon2, který
• obdrží seznam výrokových proměnných (reprezentovaných atomy), v němž je každá proměnná ohodnocena hodnotou true nebo false
• vrátí seznam všech takových ohodnocení týchž proměnných, v němž se každé ohodnocení bude od vstupního lišit v hodnotách alespoň 2 proměnných.
Příklad:

Kód: Vybrat vše

```
?- aspon2([x1-true, x2-false, y-true], V).
  V =  [  [x1-false, x2-true, y-true],
          [x1-false, x2-false, y-false],
          [x1-true, x2-true, y-false],
          [x1-false, x2-true, y-false] ]
```

2. Prolog: Trojúhelníky v grafu (5 bodů)
Graf je zadán jako seznam svých vrcholů se seznamy sousedů (viz příklad). Definujte binární predikát troj(+Graf, -SeznamTrojuhelniku)
který k takovému grafu vrátí seznam všech jeho trojúhelníků. Ve výsledném seznamu by se každý trojúhelník měl vyskytovat právě jednou (t(a,b,c), t(b,c,a) a t(c,a,b) jsou stejné trojúhelníky).
Příklad:

Kód: Vybrat vše
```
?- troj([a-[b,c,d],b-[a,c],c-[a,b,d],d-[a,c],e-[]], S).
     S = [t(a,b,c), t(a,c,d)]
```

3. Haskell: Převody mezi číselnými soustavami (5 bodů)
Definujte funkce:

Kód: Vybrat vše
```
prevod1 cislo puvodni
```

pro převod čísla z z číselné soustavy o základu puvodni do dekadické číselné soustavy, a

Kód: Vybrat vše
```
prevod2 cislo nova
```

pro převod čísla z dekadické do číselné soustavy o základu nova.
Příklad:

Kód: Vybrat vše
```
> prevod1 [1,1,1,0] 2    -- převede binární 1110 do desítkové soustavy
 14
> prevod2 33 16    -- převede dekadické číslo 33 do hexadecimální soustavy
[2,1]
```

(a) Doplňte typové signatury definovaných funkcí

Kód: Vybrat vše
```
prevod1 ::
prevod2 ::
```

(b) Definujte funkci prevod1 s využitím rekurze.
(c) Sestavte alternativní definici funkce prevod1 s využitím alespoň jedné z funkcí map, filter, foldr či foldl, ale bez (explicitního) použití rekurze.
(d) Definujte funkci prevod2 s využitím funkce unfold definované následovně:

Kód: Vybrat vše
```
unfold :: (t -> Bool) -> (t -> (a, t)) -> t -> [a]
unfold done step x =  if done x then []
                                else let (y,ys) = step x
                                     in y: unfold done step ys
```

4. Haskell: Řády prvků grupy (5 bodů)
Definujte unární funkci rady, která

obdrží multiplikativní tabulku grupy jako matici prvků. První řádek matice obsahuje násobení grupovou jednotkou e a pořadí prvků odpovídající řádkům a sloupcům je stejné.
vydá seznam všech prvků spolu s jejich řády.
Řád prvku p je nejmenší přirozené číslo n takové, že n-tá mocnina p je rovna e.
(a) Definujte typovou signaturu funkce rady.
(b) Funkci rady definujte.
Příklad:

Kód: Vybrat vše

```
> rady [["e","a","b"], ["a","b","e"], ["b","e","a"]]
[("e",1), ("a",3), ("b",3)]
```

Velký příklad:
Zadání: Problém zamčených truhel
V našem problému máme otevřít N zamčených truhel, přičemž máme k dispozici N+1 klíčů.
Každá truhla má nějakou barvu, přitom různé truhly mohou být obarveny stejnou barvou. Klíče jsou také obarveny barvami, přitom různé klíče mohou mít stejnou barvu.
Háček je v tom, že truhlu vždy odemyká jen klíč téže barvy, jakou je truhla obarvena. Po odemčení truhly zůstane klíč vězet v zámku, a nelze ho tedy použít na odemčení jiné truhly téže barvy.
V každé truhle je - kromě části pokladu - uložen také jeden klíč, který lze - po odemčení této truhly - použít na odemčení další truhly příslušné barvy.
Na začátku obdržíte jeden klíč.
Lze odemčít všechny truhly? Pokud ano, tak jak?
Problém: Na vstupu je
- barva klíče, který má hledač na začátku u sebe
- množina truhel, pro každou její barva a barva klíče v ní uloženého
Je možné odemčít všechny truhly?
- pokud ano, vypište pořadí, v jakém je lze odemčít
- existuje-li více řešení, stačí nalézt jedno
Poznámka: Problém má efektivní řešení. Ale Dvořák říkal, že to není zkouška z algoritmů, takže pokud nás efektivní řešení nenapadne, tak to nevadí, a můžeme implementovat brute force řešení s nějakou heuristikou. Dále nám doporučoval začít odshora, prostě napsat nějakou kostru, podle které bude jasné, jakým způsobem to chceme řešit a implementovat kritické sekce.

Postřehy ze zkoušky:

> 11 bodů z malých příkladů stačí na dvojku
>
> U velkého příkladu toho není třeba implementovat až tak moc, já měla necelých padesát řádků včetně slovních komentářů, přišla jsem na optimální řešení a Dvořák s tím byl spokojen. Při ústní mu stačilo jen popsat jak přesně to funguje a chtěl po mě, abych napsala reprezentaci grafu pomocí Data a Type, což v mém řešení nebylo. Finální známka byla dva, ale nabízel mi i jedničku, s tím, že se mě ještě bude vyptávat.
>
> Spolužáka se ptal na rozdíly mezi data, type a newtype, generičnost a brute force v Haskellu. A na lazy evaluation a druhy řezů v Prologu.
>
> Dvořák je u ústní velmi hodný, do ničeho moc nerýpe, je to spíš příjemné povídání si o odevzdaných řešení, převážně o velké úloze, takže doporučuji si ji před ústní projít.

Řešení:
1. Prolog - toto stačilo na dva body - vypisovalo to do jednoho velkého seznamu a ne do seznamu seznamů a byly tam duplicity.

Kód: Vybrat vše

```
aspon2(L, R) :- length(L, N), generate(L, N, R).

generate(L, 2, R) :- !, swapN(L, 2, [], R).
generate(L, N, R) :- swapN(L, N, [], R1), NN is N -1, generate(L, NN, R2),
    append(R1, R2, R).

swapN([], _, A, A).
swapN([H-V|T], N, A, R) :- length([H-V|T], L), L = N, !,
    NN is N-1, switch(V, NV), swapN(T, NN, [H-NV|A], R).
swapN([H-V|T], N, A, R) :- NN is N-1, switch(V, NV),
    swapN(T, NN, [H-V|A], R1), swapN(T, NN, [H-NV|A], R2), append(R1, R2, R).

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

2. Prolog - toto řešení bylo za čtyři body, ale stačilo opravit jen pár chybek a bylo by za plný počet (i přesto že je neefektivní)

Kód: Vybrat vše

```
% graf je reprezentovany jako seznam hran: [a-b, a-c, b-c, a-d, c-d]

troj(G, R) :- vertices(G, [], V), comb(V, 3, [], R). %find_triangles(C, G, [], R).

find_triangles([], _, A, A).
find_triangles([H|T], G, A, R) :- is_triangle(H, G), !, find_triangles(T, G, [H|A], R).
find_triangles([_|T], G, A, R) :- find_triangles(T, G, A, R).

comb(_, 0, A, [A]) :-!.
comb([H|T], N, A, [R]) :- length([H|T], L), N = L, !,
    append([H|T], A, R).
comb([H|T], N, A, R) :- N1 is N-1, comb(T, N1, [H|A], R1),
    comb(T, N, A, R2), append(R1, R2, R).

vertices([], A, A).
vertices([V1-V2|T] ,A, R) :- member(V1, A), member(V2, A), !, vertices(T, A, R).
vertices([V1-V2|T], A, R) :- member(V2, A), !, vertices(T, [V1|A], R).
vertices([V1-V2|T], A, R) :- member(V1, A), !, vertices(T, [V2|A], R).
vertices([V1-V2|T], A, R) :- vertices(T, [V1, V2|A], R).

% tady budou vsechny kombinace
is_triangle((V1, V2, V3), G) :- member(V1-V2, G), member(V2-V3,G), member(V3-V1,G).
```

4. Haskell - tohle bylo za plný počet

Kód: Vybrat vše

```
rady :: Eq a => [[a]] -> [(a, Int)]
rady ((h:hh):t) = help h hh ((h:hh):t) [(h, 1)]

help _ [] _ r = r
help a (h:t) l r = help a t l (r++ (compute a 1 (h, a) l))

compute e n (a, a1) l
 | (e == (eval a a1 l)) = [(a, n)]
 | otherwise = compute e (n+1) (a, (eval a a1 l)) l
```

```
eval a b (h:t) = evaluate b h (row a (h:t))

evaluate b (h1:t1) (h2:t2)
 | (b == h1) = h2
 | otherwise = evaluate b t1 t2

row a ((h:hh):t)
 | (a == h) = (h:hh)
 | otherwise = column a t
```

Velký příklad - algoritmus je optimální, měl k tomu jen pár připomínek, ale ty drobné chyby mu vůbec nevadily, protože ví, že je na to málo času. Bylo to hodnoceno mezi jedničkou a dvojkou

Kód: Vybrat vše

```
-- Ulohu prevedeme na orientovany graf, kde vrcholy budou barvy. Z vrcholu b1 vede hrana do
vrcholu b2, pokud v aspon jedne truhlici barvy b1 je klic barvy b2, na hrane b1 - b2 bude
seznam takovych truhel.
-- Vsechny truhly je mozne otevrit jen v pripade, ze takto vytvoreny graf je souvisly a
vsechny vstupni stupne vrcholu jsou rovny vystupnim stupnu, krome jednoho vrcholu (mame
jeden klic navic), v tomto vrcholu vsak nesmime zacinat

-- Reprezentace vstupu: seznam truhel je seznam trojic - cislo truhly, barva truhly, barva
klice

-- Ze seznamu truhel chceme vyrobit orientovany graf, graf je reprezentovany jako dvojice -
seznam vrcholu a seznam hran, vrchol je jmeno barvy, hrana je trojice - pocatecni vrchol,
koncovy vrchol, seznam truhel:

solve s (v, e)
 | existSol = findPath s (v, e) []
 | otherwise = []

findPath _ ([], _) path = path
findPath s (v, e) path = findPath (newStart (snd (selectEdge s (v, e)))) (fst (selectEdge s
(v, e))) (path ++ [snd (selectEdge s (v, e))])


selectEdge s (v, e) = helpSelectEdge s (v, e) (findEdges s e)

-- Zkusim vybrat nejakou hranu, pokud po jejim odebrani ma uloha stale reseni, tak ji
odeberu, jinak si vyberu jinou
helpSelectEdge s (v, e) ((from, to, l):t)
 | existSol (removeEdge e (from, to) []) = (v, (removeEdge e (from, to)) [], (from, to, l))
 | otherwise = helpSelectEdge s (v, e) t

newStart (_, to, _) = to

-- Odebere hranu ze seznamu - zkusi jit prvni truhlou, pokud je to jedina truhla, tak smaze
celou hranu. Nasledne to na graf pusti bfs z noveho startovniho vrcholu, aby se zjistilo,
zda je graf stale souvisly
removeEdge ((from, to, (h:t)):tt) (f, t) r
 | ((from != f) || (to != t)) = removeEdge tt (f, t) (r ++ [(from, to, (h:t))])
 | (length t) > 0 = ((r ++ (f, t, t)), (f, t, h))
 | otherwise = (r, (f, t, h))
```

```
-- Overeni souvislosti udelame pomoci bfs pusteneho ze zdroje
bfs v e = bfs_ e (q e v []) [v]

bfs_ _ [] r = r
bfs_ e ((v1, v2):t) r
 | ((member r v1) && (member r v2)) = bfs_ e t r
 | (member r v1) = bfs_ e (t ++ (q e v2 [])) (r ++ [v2])
 | otherwise = bfs_ e (t ++ (q e v1 [])) (r ++ [v1])

member [] _ = False
member ((h,_):t) e
 | (h == e) = True
 |  otherwise = member t e

q [] _ r = r
q ((v1, v2):t) v r
 | ((v == v1) || (v == v2)) = q t v (r ++ [(v1, v2)])
 | otherwise = q t v r
```

Quake
Matfyz(ák|ačka) level I

Příspěvky: 9
Registrován: 25. 5. 2018 21:28
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

POWERED_BY
Český překlad – phpBB.cz