Programiranje 2:
Koncepti programskih jezikov

3. domača naloga

Junij 2022

Vse naloge so enakovredne. Naloge se rešuje samostojno, v primeru prepisovanja se upošteva univerzitetni pravilnik. Oddajte eno TXT datoteko z OCaml kodo na `http://e.famnit.upr.si`. Pred oddajo nujno testirajte svojo kodo, ki se mora pravilno prevesti[1]. Če oddate kodo, ki se ne prevede, bo vaš izdelek točkovan z 0 točkami. Rešitev naj vsebuje vaše ime, priimek, smer, vpisno številko, ter priimek vašega asistenta (Hajdu/Krnc/Muršič) – ti podatki naj bodo vsebovani kot komentar. Poskusite rešiti vsako oštevilčeno nalogo kot eno samo funkcijo, razred ali metodo. Vaša funkcija se mora v vseh pogledih (ime funkcije, vrstni red vhodnih podatkov, izhod) skladati s primeri, podanimi v tem dokumentu. Priložena je tudi datoteka z nekaj primeri za testiranje. **Veliko sreče!**

1 Konzervatino Egipčansko družinsko drevo

Podane imate slednje tipe

```
# type gender = M | F;;
# type fTree = {n:string; s:string; g:gender; c:fTree
   array};;
```

Tip `fTree` predstavlja ukorenjeno družinsko drevo, kjer je n ime korena družinskega drevesa, g je njegov spol, s je ime soproga/e (ki je nasprotenega spola), in c predstavlja (potencialno prazno) polje z njunimi otroki (ki imajo lahko rekurzivno svoje otroke).

(a) Napišite funkcijo `countfm : fTree -> int * int = <fun>` ki za vhoden fTree izračuna število potomcev ženskega in število potomcev moškega spola (vključno s korenom, soprogi ne štejejo, ker niso v krvnem sorodstvu).

(b) Napišite funkcijo `childless : fTree -> fTree list = <fun>` ki za vhodno družinsko drevo vrne seznam potomcev, ki nimajo svojih otrok.

[1] Uspešno prevajanje lahko preverite s pomočjo orodja dosegljivega na `https://try.ocamlpro.com/`.

(c) Napišite funkcijo `childrenAtDetph : int -> fTree -> string list = <fun>` ki za vhodno naravno število n in fTree vrne polno ime vseh potomcev na globini n kot seznam ($n = 1$ so otroci, $n = 2$ vnuki, ...). Po tradicionalnem Egipčanskem poimenovanju dobi otrok polno ime kot združek lastnega imena, imena očeta in imena očeta od očeta (če obstaja).

```
# let tree1={n="Edrice";s="Nkosi";g=F;c=[|{n="Waaiz";s
   ="Dendera";g=M;c=[|{n="Bithiah";s="Ife";g=M;c=[|{n=
   "Baahir";s="Nephthys";g=M;c=[||]};{n="Nane";s="
   Abayomi";g=F;c=[||]};|]};{n="Eshe";s="Uthman";g=F;c
   =[|{n="Bahiti";s="Amr";g=F;c=[||]};{n="Habibah";s="
   Djoser";g=F;c=[||]};|]};|]};{n="Nour";s="Sef";g=F;c
   =[|{n="Khepri";s="Hager";g=F;c=[|{n="Thema";s="Apep
   ";g=F;c=[||]};{n="Femi";s="Eboney";g=M;c=[||]};{n="
   Saadah";s="Chione";g=M;c=[||]};|]};{n="Bastet";s="
   Nephi";g=F;c=[|{n="Nkosi";s="Zuberi";g=M;c
   =[||]};|]};|]};|]};;
# countfm tree1;;
- : int * int = (9, 6)
# childless tree1;;
- : fTree list =
[{n = "Baahir"; s = "Nephthys"; g = M; c = [||]};
{n = "Nane"; s = "Abayomi"; g = F; c = [||]};
{n = "Bahiti"; s = "Amr"; g = F; c = [||]};
{n = "Habibah"; s = "Djoser"; g = F; c = [||]};
{n = "Thema"; s = "Apep"; g = F; c = [||]};
{n = "Femi"; s = "Eboney"; g = M; c = [||]};
{n = "Saadah"; s = "Chione"; g = M; c = [||]};
{n = "Nkosi"; s = "Zuberi"; g = M; c = [||]}]
# childrenAtDetph 1 tree1 ;;
- : string list = ["Nour Nkosi"; "Waaiz Nkosi"]
# childrenAtDetph 2 tree1 ;;
- : string list =
["Bastet Sef"; "Khepri Sef"; "Eshe Waaiz Nkosi"; "
   Bithiah Waaiz Nkosi"]
# childrenAtDetph 3 tree1 ;;
- : string list =
["Nkosi Nephi"; "Saadah Hager"; "Femi Hager"; "Thema
   Hager"; "Habibah Uthman"; "Bahiti Uthman"; "Nane
   Bithiah Waaiz"; "Baahir Bithiah Waaiz"]
# childrenAtDetph 4 tree1 ;;
- : string list = []
```

2 Grafovski modul

V tej vaji bomo ustvarili modul za delo z grafi[2]. Na voljo sta naslednja tipa:

```
# type node = int;;
# type graph = (node*node list) list;;
```

Za namene te vaje predpostavimo, da so vozlišča vedno označena kot zaporedna pozitivna cela števila, začenši s številko 1. Vse funkcije, ustvarjene v tej vaji, se morajo ujemati s spodnjim podpisom in uporabiti zgoraj navedeni vrsti.

1. Ustvari modul `Graph` ki uporabi predstavitev seznamov sosešcin, ter implementiraj funkcije `order` (tj. število vozlišč), ter `degree`, ki pri podanem grafu G ter vozlišču $v \in V(G)$ vrne ustrezno stopnjo $\deg_G(v)$. Sorodno s tem implementiraj funkciji `max_degree` ter `min_degree`, ki pri podanem grafu G vrneta vrednosti $\delta(G)$, ter $\Delta(G)$ (kot zgled glej t.i. Claw graf spodaj).

```
# let claw:Graph.graph =
    [1,[2;3;4];2,[1];3,[1];4,[1]] ;;
val claw : Graph.graph = [(1, [2; 3; 4]); (2, [1])
    ; (3, [1]); (4, [1])]
# Graph.order claw ;;
- : int = 4
# Graph.degree claw 1 ;;
- : int = 3
# Graph.max_degree claw ;;
- : int = 3
# Graph.min_degree claw ;;
- : int = 1
```

2. Implementirajte funkcijo `matrix_of_graph`, ki vzame graf kot vhod in izračuna njegovo matriko sosednosti.

```
# Graph.matrix_of_graph claw ;;
- : int array array =
[|[|0; 1; 1; 1|]; [|1; 0; 0; 0|]; [|1; 0; 0; 0|];
    [|1; 0; 0; 0|]|]
```

3. Implementirajte funkcijo `add_edge`, ki podanemu grafu doda vhodno povezavo (ki je predstavljena z ustreznim parom vozlišč). Ne pozabite ustrezno posodobiti seznamov sosešcin.

```
# let paw = Graph.add_edge claw (3,4) ;;
val paw : Graph.graph = [(1, [2; 3; 4]); (2, [1]);
    (3, [4; 1]); (4, [3; 1])]
```

[2]Za standardne definicije iz teorije grafov glej `https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)`.

4. Implementirajte funkcijo `delete_last`, ki iz danega grafa izbriše vozlišče, označeno z najvišjim celim številom. Ne pozabite ustrezno posodobiti drugih seznamov soseščin.

```
# Graph.delete_last claw ;;
- : (Graph.node * Graph.node list) list = [(1, [2;
    3]); (2, [1]); (3, [1])]
```

5. Implementirajte funkcijo `add`, ki prejme kot vhod eno vozlišče (predstavljeno s svojim seznamom sosednosti) ter ga doda podanemu grafu na n vozliščih (označenih med 1 in n). Takšno oglišče mora seveda imeti oznako $n + 1$. Ne pozabite ustrezno posodobiti drugih seznamov soseščin.

```
# let bowtie = Graph.add paw [1;2] ;;
val bowtie : (Graph.node * Graph.node list) list =
 [(5, [1; 2]) ; (1, [5; 2; 3; 4]) ; (2, [5; 1]) ;
   (3, [4; 1]) ; (4, [3; 1]) ]
```

Pri implementaciji funkcij se je potrebno držati spodnjih podpisov.

```
module Graph :
  sig
    type node = int
    type graph = (node * node list) list
    val order : graph -> int
    val degree : graph -> node -> int
    val max_degree : graph -> int
    val min_degree : graph -> int
    val matrix_of_graph : graph -> int array array
    val add_edge : graph -> node * node -> graph
    val delete_last : graph -> graph
    val add : graph -> node list -> graph
  end
```

3 Učitelji, študentje, in asistenti

V univerzitetnem informacijskem sistemu so ljudje klasificirani v razred učiteljev in razred študentov. Asistenti so poseben pod-razred učiteljev in študentov, ki imajo lastnosti obeh.

1. Definiraj razred `oseba` ki vsebuje podatke o imenu, priimku in identifikatorju osebe. Razred `oseba` implementira metodo `predstavi`.

```
class oseba :
  string ->
  string ->
  int ->
  object
    val mutable id : int
    val mutable ime : string
    val mutable priimek : string
    method predstavi : string
  end
```

2. Definiraj razreda `ucitelj` in `student` kot pod-razreda razreda `oseba` ter razred `asistent`, ki deduje od razredov `ucitelj` in `student`. Razred `ucitelj` vsebuje podatkovni element `placa` in razred `student` vsebuje podatkovni element `mentor`. Implementiraj metodo `predstavi` v vseh treh razredih. Uporabi po potrebi metode `predstavi` iz nad-razredov!

```
class ucitelj :
  string ->
  string ->
  int ->
  int ->
  object
    val mutable id : int
    val mutable ime : string
    val placa : int
    val mutable priimek : string
    method predstavi : string
  end
```

```
class student :
  string ->
  string ->
  int ->
  string ->
  object
    val mutable id : int
    val mutable ime : string
```

```
    val mentor : string
    val mutable priimek : string
    method predstavi : string
  end
```

3. Kreiraj instanco razreda `asistent`. Napiši kodo, ki sproži metode `predstavi` iz razredov `ucitelj`, `student` in `asistent`![3]

```
class asistent :
  string ->
  string ->
  int ->
  int ->
  string ->
  object
    val mutable id : int
    val mutable ime : string
    val mentor : string
    val placa : int
    val mutable priimek : string
    method predstavi : string
    method predstavi_vse : unit
  end
```

[3] Takšno kodo se lahko zapiše na različne načine, zato ni nujno da se podpis vaše implementacije razreda `asistent` ujema s spodnjim podpisom.