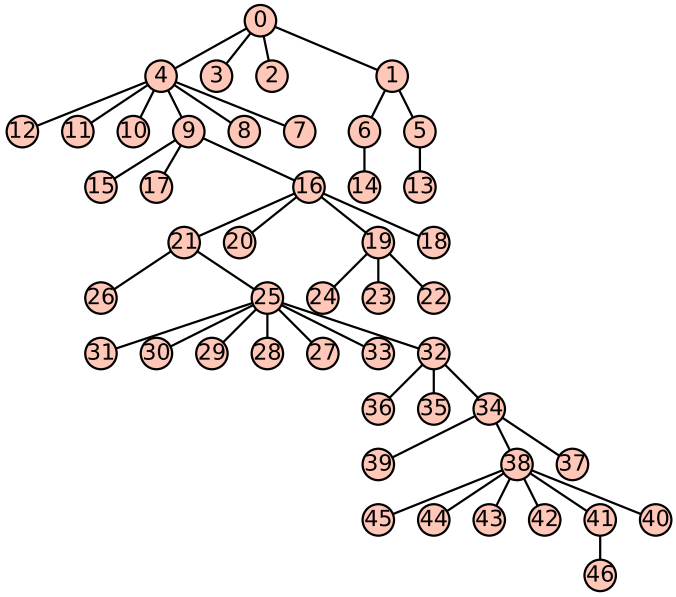
Matemàtica computacional i analítica de dades
Algorítmia i combinatòria en grafs ...
Curs 2020–21

6 Codificació i recorregut d'un arbre.

El codi que ve a continuació permet carregar la informació corresponent a un *arbre genealògic* guardada en el fitxer `ArbreG.csv`. Si visualitzeu el contingut d'aquest fitxer veureu que consisteix en una llista, on en cada línia hi apareix un número d'ordre al principi (que, finalment, ignorarem totalment però que és útil en l'escriptura del fitxer), després el nom d'un individu i, finalment, una llista de números (que són els corresponents als seus *fills*). L'esquema[1] d'aquest arbre és el següent:



Es pot veure que les característiques (nom, nombre de fills i *punters* cap a cada un dels fills) de cada individu (node de l'arbre) queden guardades en un `struct` de tipus `Persona`. Per tal de tenir un lloc on guardar tota la informació que es va llegint del fitxer, es crea el *vector* `llista`, es compta quants nodes apareixen (línies 23–26) i s'assigna dinàmicament la memòria necessària tant per al vector `llista` com per a cada un dels elements `llista[i]`. A continuació es procedeix a llegir la informació de cada node i a guardar-la en cada una de les posicions reservades pels punters `llista[i]` corresponents. El programa acaba donant un llistat de la informació que s'ha guardat seguint el contingut del vector `llista`.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct Persona{
5      char nom[20];
6      int nf;
7      struct Persona *fills[10];
8  }Persona;
9
10 int main()
11 {
```

[1] SageMath fecit.

```
12      FILE * dades;
13      int nfills=0,id,ll,npersones=0,i;
14      int fill;
15      Persona **llista;
16
17      dades=fopen("ArbreG.csv","r");
18      if(dades==NULL)
19      {
20          printf("\nNo s'ha accedit al fitxer de dades\n");
21          return 1;
22      }
23      while((ll=fgetc(dades)) != EOF)
24      {
25          if(ll=='\n'){npersones++;}
26      }
27      rewind(dades);
28      if((llista = (Persona **) malloc(npersones * sizeof(Persona*))) == NULL
        )
29      {
30          printf ("\nNo es possible assignar la memoria necessaria...\n\n");
31          return 1;
32      }
33      for(i=0;i<npersones;i++)
34      {
35          if((llista[i] = (Persona *) malloc(sizeof(Persona))) == NULL)
36          {
37              printf ("\nNo es possible assignar la memoria necessaria...\n\n
        ");
38              return 1;
39          }
40      }
41      for(i=0;i<npersones;i++)
42      {
43          nfills=0;
44          fscanf(dades,"%i;",&id);
45          fscanf(dades,"%[a-zA-Z]",llista[i]->nom);
46          while(fgetc(dades)!='\n')
47          {
48              fscanf(dades,"%i",&fill);
49              llista[i]->fills[nfills]=llista[fill];
50              nfills++;
51          }
52          llista[i]->nf=nfills;
53      }
54      fclose(dades);
55      for(i=0;i<npersones;i++)
56      {
57          printf("(%d) %s\nNFills= %d\n",i,llista[i]->nom,llista[i]->nf);
58          for(fill=0;fill<llista[i]->nf;fill++)
59          {
60              printf("Fill %d, %s. ",fill+1,llista[i]->fills[fill]->nom);
61          }
62          if(llista[i]->nf>0) printf("\n");
63      }
64      return 0;
65  }
```

A les seccions següents canviarem l'estructura de les dades del format de vector d'apuntadors que ens dóna la variable `llista` a dues llistes enllaçades:

- Moure's dins l'arbre en ordre *breath-first search*: això equivaldrà a programar-ho com una cua, i obtindrem una llista enllaçada que començarà al node `0` i anirà afegint a continuació els fills de cada node que anem afegint, per tant, quedarà en el mateix

ordre que el vector `llista`.

- Moure's dins l'arbre en ordre *depth-first search*: això equivaldrà a programar-ho com una pila, i obtindrem una altra llista enllaçada que també començarà al node `0` i anirà afegint els fills abans de tractar els germans. L'ordre serà `0`, `4`, `12`, `11`, `10`, `9`, `17`, ...

6.1 Recorregut com una cua: BFS

Afegiu com a les dues primeres línies del programa i en format comentari els vostres `Nom`, `Cognom` i `NIU`.

El nom dels programes que contenen el codi ha de ser de la forma `Pr6ExY.c` on `Y` fa referència a l'exercici. Per exemple, el nom del programa de l'apartat següent hauria de ser `Pr6Ex611.c`.

Exercici 6.1.1: L'objectiu d'aquest exercici és modificar el codi per a que ho tracti com una cua. Recordeu que programar una cua de forma estructurada consta de vàries parts, i aquí en proposem una forma:

- Una estructura per a cada element amb l'adreça a la persona i al següent element, que podem codificar com:

```
typedef struct Element{
   Persona * prsn;
   struct Element * seg;
}ElementCua;
```

- Una estructura de cua amb el primer i últim elements de la cua:

```
typedef struct{
   ElementCua * inici, * final;
}UnaCua;
```

- Una funció que posi un element nou a la cua (sempre al final) i un altre que tregui el primer, que podrien tenir els prototipus:

```
void posarencua(UnaCua * , Persona * );
void treuelprimer(UnaCua * );
```

El nou codi, a més de programar les funcions `posarencua` i `treuelprimer`, necessitarà altres variables:

```
Persona * arrel; // serà el llista[0]
UnaCua Pendents={NULL,NULL}; // de moment no hi ha ningú a la cua
```

Modifiqueu el codi per tal que, al final, es guardi l'adreça del node *arrel* de l'arbre (corresponent al valor de `llista[0]`), seguidament *s'oblidi* el vector `llista` per tal d'accedir a la informació de l'arbre a través del node arrel, i de les relacions *pare/-fill*, i s'acabi fent un llistat *per nivells* del mateix tipus que es produeix en l'exemple (s'obtenen tots els individus que apareixen a l'arbre amb els seus fills). Es tracta de programar una iteració que, mentre hi hagi persones pendents a la cua:

- Mostri la informació del primer de la cua i el tregui de la cua.
- Afegeixi a la cua tots els seus fills (si en té).

6.2 Recorregut com una pila: DFS

El recorregut dels elements d'un arbre *per nivells* permet assignar a cada node, de forma simple, la seva *profunditat* o *distància* respecte el node que s'hagi triat com arrel (que té profunditat 0). L'únic que cal tenir en compte, en aquest cas, és que la profunditat d'un *fill* és una unitat més gran que la del seu *pare*.

Per exemple, al graf següent tenim que el node 0 és l'arrel (i té profunditat 0), els nodes 1, 2, 3 i 4 tenen profunditat 1, ..., i el node 46 té profunditat 10.

Tenint en compte aquest fet, una manera natural de mantenir aquesta informació consisteix a actualitzar-la, mentre s'està fent el recorregut de l'arbre, en el moment d'incorporar cada node a la **cua de nodes pendents** quan s'està llegint la llista de fills d'un node concret.

A aquest apartat suposarem que heu completat l'exercici anterior i és important fer-ne una còpia: haureu de modificar els codi que ja heu fet.

Exercici 6.2.1: Modifiqueu la part del codi inicial que permet llegir i recórrer l'arbre codificat al fitxer `ArbreG.csv` per tal que calculi (i mantingui guardada) la profunditat de cada node de forma que, en el moment de mostrar la informació del recorregut, es pugui mostrar també aquesta dada.

Per tal que la informació no es perdi, i es pugui recuperar sense haver de tornar-ho a calcular, convé modificar l'estructura que guarda les característiques de cada *persona* afegit al `struct` corresponent un camp `unsigned prof` (que d'entrada convé inicialitzar a un valor que es pugui marcar com *infinit*).

```
typedef struct Persona{
  char nom[20];
  int nf;
  struct Persona * fills[10];
  unsigned prof;
}Persona;
```

D'aquesta manera es podrà actualitzar el valor del *camp* `prof` de cada node en el moment que passa a la cua de nodes pendents de processar provinent de la llista de fills d'un node. Per tal de comprovar-ne el funcionament, afegiu la profunditat a la impressió dels resultats.

Recordeu que l'estratègia per a recórrer els nodes d'un arbre seguint el camí de la profunditat en comptes de anant per nivells consisteix a *anar posant en una pila* els fills de cada node que es processa. Segons en quin moment es retira de la pila el seu primer element i es processa es parla d'algoritmes de *pre-ordre* o *post-ordre*. En aquesta sessió considerarem tota l'estona que fem un recorregut de tipus pre-ordre: es retira el primer element de la pila i a continuació es processa mentre s'inclouen a la pila (a dalt) els seus fills.

Exercici 6.2.2: Canvieu el programa de recorregut de l'Exercici 6.1.1 de l'arbre codificat a `ArbreG.csv` per tal que el resultat sigui un recorregut en profunditat. Òbviament, caldrà programar les funcions de manipulació d'una pila adaptant els tipus de les variables (i `struct`) a la situació que es té en aquest exemple.
Indicació: Una manera de procedir seria com l'aconsellada a l'apartat de cues. Primer necessitem les estructures per manipular les piles. A més de l'estructura `Persona` i `ElementCua` necessitem:

```
typedef struct{
  ElementCua *inici;
}UnaPila;
```

A una pila, tant sols necessitem saber quin és el primer element, ja que quan n'afegim un de nou el posem per davant del primer (i no ens preocupa de qui és l'últim).
Hem de substituir les funcions `posarencua` i `treuelprimer` per les anàloges per a piles, que tindrien els prototipus:

```
void apila(UnaPila *, Persona * );
Persona * treuelprimerpila(UnaPila * );
```

on, `treuelprimerpila` retorna l'adreça de la persona que traiem de la pila (que és la primera) per a poder-la processar (l'hem de treure abans per a evitar que els seus fills no es posin a sobre).

6.3 Optimització de la memòria i adaptació a altres grafs

Aquest apartat és independent dels anteriors, pel que podeu partir del codi inicial.

Exercici 6.3.1: L'estructura `Persona` conté un vector de 10 posicions on es guarden els punters dels seus *fills*. Quan un individu no té fills (el camp `nf` val 0) l'espai de memòria corresponent es desaprofita, cal estar segurs que cap individu tindrà més de 10 fills i, a més, no hi ha control sobre quin és el seu contingut. Aquests inconvenients es poden evitar substituint aquest camp per un punter que es pugui inicialitzar a `NULL` (mentre no hi ha constància que l'individu té fills) i *apunti* a un vector fabricat amb assignació de memòria dinàmica quan es llegeix en el fitxer que l'individu corresponent té fills.

Instruccions finals

Quan acabeu la pràctica, feu el lliurament dels fitxers de codi (que tenen els noms de la forma `Pr6ExY.c` segons el que hem indicat anteriorment) a través del Campus Virtual des de l'apartat de lliuraments de l'assignatura. Recordeu que al principi de cada fitxer hi ha d'haver el vostre nom i NIU.