Els grafs i les xarxes socials

PART 1 de 2 del projecte LP curs 2019/20

Introducció Representació de Grafs

Les xarxes socials generen grafs de connexió entre les persones: qui és amic de qui, qui confia en qui etc. Volem fer la implementació d'un graf amb milers de nodes que estan connectats entre si, i poder fer consultes sobre ell. La característica principal d'aquest graf és que és un graf dispers, per això volem utilitzar estructures de dades que no utilitzin una gran quantitat de memòria innecessària.

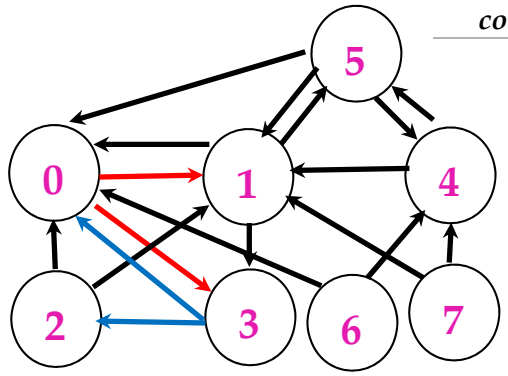
1. Introducció

Com escollir les opinions més importants dels usuaris d'una xarxa? Basant-nos en un article dels creadors de Google per mostrar les pàgines més importants associades a un concepte de l'any 1998 http://infolab.stanford.edu/~backrub/google.html, veiem com podem associar una importància a la opinió d'un usuari a una xarxa. La idea bàsica és que quantes més persones confiïn en un usuari més important serà la seva opinió. Però la importància dels usuaris que hi confien, també l'hem de tenir en compte.

1.2 Representació de Grafs

La majoria sabeu que un graf es pot representar amb una matriu d'adjacència. Una matriu d'adjacència pot representar les connexions entre nodes d'un graf de la següent manera:

Suposem que tenim un graf que representa la confiança que tenen els usuaris en la opinió dels altres. Així l'usuari 0 confiaria en l'usuari 3, i 1, mentre l'usuari 3 confiaria en l'usuari 0 i el 2. Això ho veiem representat en el graf següent i a la matriu d'adjacència que el representa.



confiança	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0		X		X				
1	X			X		X		
2	X	X						
3	X		X					
4		X				X	X	X
5	X	X			X			
6	X				X			
7		X			X			

Podem observar que a l'exemple, que anomenarem ***Xarxa1,*** de les 64 possibles relacions entre nodes només tenim 20. Això fa que representar aquest tipus de matrius amb una matriu de n files per n columnes pot desaprofitar molt d'espai. Penseu que aquí tenim un exemple molt petit però podem arribar a treballar amb matrius de milers de files i columnes.

2. Primer lliurament(de 2)

2.1 Objectiu

Volem tenir una xarxa social representada en forma de graf mitjançant una matriu d'adjacència que ens permeti desar grafs dispersos. Sobre aquesta xarxa volem fer una sèrie de càlculs dels quals parlarem a la segona fase del projecte. Per poder-los realitzar necessitarem tenir implementada una matriu que permeti guardar aquest tipus de grafs dispersos i permeti fer una sèrie d'operacions sobre ella.

Per fer-ho tenim dos fitxers que representen les xarxes:

- El fitxer **Xarxa1.txt** que és una xarxa de test amb nomes 8 nodes.
- El fitxer **Epinions.txt** que és una xarxa real amb 75877.

Exemple fitxer: Xarxa1.txt per veure la seva estructura:

```
0	3
1	0
1	3
1	5
2	0
2	1
3	0
3	2
4	1
4	5
4	6
4	7
5	0
5	1
5	4
6	0
6	4
7	1
7	4
```

Aquest fitxer ens indica que a la fila 0 columna 3 hi va un 1; a la fila 1 columna 0 va un 1 i així successivament. Les coordenades de la matriu que no apareixen al fitxer valen 0.

La xarxa guardada a Xarxa1.txt serà la nostra base de dades de treball per tal de programar i depurar els nostres algorismes, però el nostre objectiu és que el projecte funcioni sobre la xarxa real, és a dir Epinions. La xarxa Epinions.com que utilitzarem és pública i anònima i representa una xarxa social a la qual els usuaris presenten opinions de productes. Els membres de la xarxa poden decidir en quins usuaris confiar. Totes aquestes relacions de confiança formen un graf i a partir d'elles i les qualificacions de les revisions de productes es determina quines revisions es mostren a un usuari. El graf te: 75879 nodes i 508837 relacions. Veiem doncs que el grau de dispersió és de $\frac{508837}{5757622641}$

2.2 Matriu exercici 6

Donada la matriu de l'exercici 6 avaluable del tema0 què passaria si afegim un nou constructor que creï una matriu a partir d'un fitxer d'entrada. El constructor rebria el nom del fitxer i la mida de la matriu en files i columnes. El constructor crearia la matriu posant tots els valors a 0 i modificant les posicions indicades en el fitxer per un 1.

- Matriu(string nomFitxer, int midaFila, int midaCol);
  Per exemple amb el fitxer Xarxa1.txt aquest constructor es cridaria de la següent forma:
  Matriu m("Xarxa1.txt",8,8);

Amb la classe Matriu definida d'aquesta manera si ara definíssim una matriu a partir del fitxer `"Xarxa1.txt"`. Quant ocuparia? I si obríssim `"Epinions.txt"`? Quin espai ocuparia la matriu?

Veiem doncs la necessitat de crear una altre estructura de dades per aquest tipus de matrius.

2.3 Matriu Dispersa

Una matriu dispersa és una matriu a on la majoria dels seus elements valen 0. El grau de dispersió es mesura en funció del nombre de valors diferents de zero de la matriu dividit pel nombre total de posicions a la matriu. En el nostre exemple $\frac{20}{64} = 0.2968$.

Hi ha diverses maneres de representar aquestes matrius per tal de no desaprofitar massa memòria, entre elles trobem el diccionari de claus i la CSR (Compressed Storage Row).

2.3.1 Diccionari de claus

Al diccionari de claus per cada valor no zero de la taula guardem (fila,columna,valor), així tenim un vector de triplets de (posició fila, posició columna, valor). En l'exemple guardaríem: 20X3=60 enters en comptes de 64. Com podeu veure l'estalvi no és molt gran perquè és un exemple d'un graf molt petit que ens servirà com a exemple per després tractar un cas real amb molts més nodes i a on aquesta diferència serà molt més gran.

Donada la matriu de l'exemple ***Xarxa1*** que trobarem al fitxer **Xarxa1.txt** a Caronte.:

confiança	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0		X		X				
1	X			X		X		
2	X	X						
3	X		X					
4		X				X	X	X
5	X	X			X			
6	X				X			
7		X			X			

Tindrem el fitxer **Xarxa1.txt** que la representa (veure apartat 2.1 el fitxer Xarxa1.txt)

La representació en forma de diccionari de claus seria:

	fila	col	valor
	0	1	1
	0	3	1
	1	0	1
	1	3	1
	1	5	1
	2	0	1
	2	1	1
	3	0	1
	3	2	1
	4	1	1
	4	5	1
	4	6	1
	4	7	1
	5	0	1
	5	1	1
	5	4	1
	6	0	1
	6	4	1
	7	1	1
	7	4	1

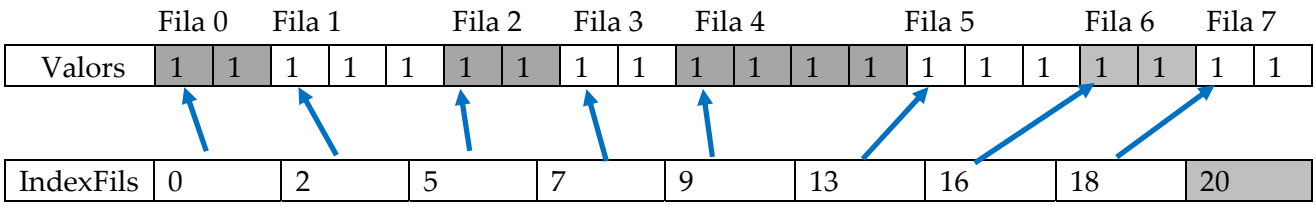
2.3.2 CSR (Compressed Storage Row)

Al CSR ens guardem 3 vectors: Valors, Columnes, IndexFiles.

- A **Valors** guardem tots els valors diferents de zero de la matriu per ordre d'esquerra a dreta i de dalt a baix.
- A **Columnes** guardem l'índex de la columna corresponent al valor guardat a la mateixa posició del vector Valors.
- A **IndexFiles** guardem tants valors com files te la matriu més una. Cada valor indica la posició a on comencen els valors de la fila corresponent al vector Valors. Si una fila no te cap valor tindrà el mateix índex que la fila següent. La ultima posició indica el nombre total de valors diferents de zero que hi ha a la matriu.

Donada la matriu de l'exemple ***Xarxa1*** La representació en forma de CSR seria:

Columnes	1	3	0	3	5	0	1	0	2	1	5	6	7	0	1	4	0	4	1	4



2.4 Primera part del projecte

Aquesta primera part del projecte consisteix en crear una classe MatriuDispersa que permeti guardar una matriu com la presentada al fitxer Epinions.txt. ens basarem en la matriu implementada a l'exercici11 però haurem de modificar la seva estructura. Per això haureu de triar quin tipus de representació voleu utilitzar tenint en compte les dues opcions que us presentem als apartats 2.3.1 i 2.3.2.

Un cop triada la representació volem tenir:

- ~~Tots els atributs necessaris per guardar la matriu dispersa de nombres reals, que haurà de poder augmentar a mida que afegim nous elements i que només guardarà els elements diferents de 0.~~
- ~~Un constructor per defecte que inicialitza la matriu buida amb dimensions (0,0).~~
- ~~Un destructor.~~
- ~~Un constructor amb dos paràmetres que siguin el nº de files i el nº de columnes de la matriu, que suposarà la matriu inicialitzada tota a zeros.~~
- ~~Un constructor de còpia que inicialitzi la matriu de forma que sigui exactament igual a la matriu que es passa com a paràmetre (tant en dimensions com en contingut).~~
- ~~Un operador d'assignació que permeti assignar a la matriu el valor d'una altra matriu (m1 = m2) de forma que passin a ser iguals (tant en dimensions com en contingut). El contingut previ de la matriu es perd (i per tant, també s'ha d'alliberar la memòria dinàmica que tingués reservada).~~
- ~~Un mètode init que li pasem el nombre de files i columnes i inicialitza la matriu a 0. Igual que el constructor explicat al punt 4.~~
- ~~Mètodes getNFiles i getNColumnes per recuperar el nº de files i de columnes de la matriu, respectivament.~~
- Mètode getValor(fila,col,valor) que retornarà un booleà indicant si la posició (fila,col) és una posició de la matriu. En cas de ser una posició de la matriu, si troba el valor el retornarà a un paràmetre per referència "valor" i si no hi ha cap valor emmagazemat a aquella posició retornarà a un paràmetre per referència "valor" un 0.
- ~~Mètode setValor(fila,col,valor) que afegirà el valor (si no és 0) a la posició (fila,col) de l'estructura. Aquest mètode afegirà tantes files i columnes com sigui necessari si la posició (fila,col) és més gran que els marges de la matriu actual, i modificarà la mida de la matriu en aquest cas.~~
- ~~Una sobrecàrrega de l'operador * de multiplicació perquè rebi com a paràmetre un nº real i retorni el resultat de multiplicar tots els valors de la matriu per aquest valor que es passa com a paràmetre.~~
- Una sobrecàrrega de l'operador * de multiplicació perquè rebi com a paràmetre un vector de floats de la stl i retorni un altre vector de la mateixa mida, que serà el resultat de multiplicar la matriu per aquest vector que es passa com a paràmetre.
- ~~Una sobrecàrrega de l'operador / de divisió perquè rebi com a paràmetre un nº real i retorni el resultat de dividir tots els valors de la matriu per aquest valor que es passa com a paràmetre.~~
- Una sobrecàrrega de l'operador << per pintar per pantalla de les files de la matriu que siguin diferents de 0 les columnes que també siguin diferents de 0. A més hauria d'indicar quina mida en files i columnes te la matriu. Aquest operador ens retornarà la matriu de la següent manera en funció de si és format CSR o Coordenades. Tingueu

en compte que això s'utilitzarà per comparar els resultats obtinguts de la formació de la matriu amb els esperats i és important que respecteu: Majúscules/minúscules, espais en blanc i salts de línia.

a) Per matriu de COORDENADES

MATRIU DE (FILES: **numero de files** COLUMNES: **numero columnes**)

VALORS (FILA::COL::VALOR)

(**fila** :: **col** :: **valor**)

I aquí tots els triplets : fila col valor que tingui la matriu. Per exemple per Xarxa1.txt tindríem:

- MATRIU DE (FILES: 8 COLUMNES: 8)
- VALORS (FILA::COL::VALOR)
- (0 :: 1 :: 1)
- (0 :: 3 :: 1)
- (1 :: 0 :: 1)
- (1 :: 3 :: 1)
- (1 :: 5 :: 1)
- (2 :: 0 :: 1)
- (2 :: 1 :: 1)
- (3 :: 0 :: 1)
- (3 :: 2 :: 1)
- (4 :: 1 :: 1)
- (4 :: 5 :: 1)
- (4 :: 6 :: 1)
- (4 :: 7 :: 1)
- (5 :: 0 :: 1)
- (5 :: 1 :: 1)
- (5 :: 4 :: 1)
- (6 :: 0 :: 1)
- (6 :: 4 :: 1)
- (7 :: 1 :: 1)
- (7 :: 4 :: 1)

b) Per la matriu CSR:

- MATRIU DE FILES: **numfiles** : COLUMNES: **numcolumnes**
- VALORS FILA: **numFila** (COL:VALOR)
- (**numcol** : **valor**)
- ...
- "MATRIUS"
- "VALORS"
- "("valors separats per espai en blanc ")"
- "COLS"

- “(“valors de les columnes corresponents”)”
- “INIFILA”
- “([“posició” : “valor”][“posició” : “valor0 : 0] [5 : 1] [8 : 2] [18 : 3] [4365 : 8] [4393 : 11] [4396 : 12] [9455 : 14] [10317 : 16] [11734 : 19] [34891 : 21] [34930 : 22] [34954 : 23] [34957 : 25] [34962 : 26] [34966 : 27] [34968 : 29] [34969 : 30] [Num Elems:31])

Per exemple el fitxer de Xarxa1.txt quedaria així:

- MATRIU DE FILES: 8 : COLUMNES: 8
- VALORS FILA:0(COL:VALOR)
- (1 : 1) (3 : 1)
- VALORS FILA:1(COL:VALOR)
- (0 : 1) (3 : 1) (5 : 1)
- VALORS FILA:2(COL:VALOR)
- (0 : 1) (1 : 1)
- VALORS FILA:3(COL:VALOR)
- (0 : 1) (2 : 1)
- VALORS FILA:4(COL:VALOR)
- (1 : 1) (5 : 1) (6 : 1) (7 : 1)
- VALORS FILA:5(COL:VALOR)
- (0 : 1) (1 : 1) (4 : 1)
- VALORS FILA:6(COL:VALOR)
- (0 : 1) (4 : 1)
- VALORS FILA:7(COL:VALOR)
- (1 : 1) (4 : 1)
- MATRIUS
- VALORS
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
- COLS
- 0 3 5 0 1 0 2 1 5 6 7 0 1 4 0 4 1 4)
- INIFILA
- ([0 : 0] [1 : 2] [2 : 5] [3 : 7] [4 : 9] [5 : 13] [6 : 16] [7 : 18] [Num Elems:20])

- Un constructor que creï una matriu a partir d’un fitxer d’entrada. El constructor rebrà el nom del fitxer. El constructor crearà la matriu posant un 1 a les coordenades indicades pel fitxer i deixant a 0 les altres posicions.

  MatriuDispersa(string nomFitxer);
  Per exemple amb el fitxer Xarxa1.txt aquest constructor es cridaria de la següent forma: MatriuDispersa m(“Xarxa1.txt”);

Es valorarà la creació de l’estructura i els algorismes de forma eficient.

Alguns dels mètodes es poden trobar amb paràmetres que no son correctes. Per exemple si volem multiplicar una matriu per un vector el vector ha de tenir el mateix nombre de files que

la matriu de columnes. Per evitar que això generi problemes treballarem amb excepcions.(Veure Annex d'excepcions).

3. Segon lliurament(de 2)

El segon lliurament el definirem després de lliurar el primer i consistirà en definir un graf dispers utilitzant aquesta matriu dispersa com a matriu d'adjacència i calcular les comunitats que apareixen seguint l'algorisme (que explicarem més endavant) de Claudet i Newman: Aaron Clauset, MEJ Newman and Cristopher Moore. Finding community structure in a very large networks. **Phys. Rev. E** 70, **066111. December 2004.** DOI: https://doi.org/10.1103/PhysRevE.70.066111.

Annexe 0. Excepcions

Introducció al concepte

El terme **excepció** és un forma curta de la frase "succes excepcional". Una **excepció** és un event que es dona durant l'execució del programa que interromp el fluxe normal de les sentencies. Són el mecanisme per gestionar els errors d'execució que es produeixen en un programa, i serveixen per:

- Donar un avis de que es pot produir un error.
- Transferir la gestió dels errors a fragments de codi específicament destinats a això.
- Quan apareix una condició excepcional es crea un objecte *Throwable que s'envia al mètode que l'ha generat*

La gestió de les excepcions permet la detecció i correcció d'errors en execució:

- Simplifiquen els programes donat que es diferencia entre el codi normal i el de tractament d'errors
- Es creen programes més robustos donat que (en molts casos) el programa no compila sense codi de tractament d'excepcions.
    - Només s'han d'utilitzar quan no es poden resoldre les situacions anòmales directament en el context, s'ha de seguir fent el control d'errors habitual

Podem trobar dos tipus de situacions:

- **Excepcions**: Situacions més o menys habituals que impedeixen completar l'execució correcta del codi. Generalment el programador ha de proporcionar el codi que les tracti o gestioni. Alguns exemples: Error en el codi o a les dades, Us inadequat d'un mètode, etc.
- **Errors:** Representen situacions d'error normalment no recuperables. El programador normalment no ha de proporcionar un tractament per a elles. Exemples:No es pot localitzar i carregar una classe, S'esgota la memòria, etc.

Utilització de les excepcions a C++

Les excepcions ens donen un mecanisme per reaccionar davant de circumstàncies excepcionals transferint el control a funcions especials anomenades handlers.

Diem que les excepcions es llencen (**throw**) i es recullen (**catch**). Per poder recollir una excepció d'un codi hem de posar aquest codi sota inspecció d'excepcions, per fer-ho posarem aquest codi a un bloc anomenat **try-catch**.

Exemple divisió nombres double:

```
#include <iostream>
using namespace std;

double division(int a, int b)
{   if( b == 0 )
    {throw "Division by zero condition!"; }
    return (a/b);
}

int main ()
{  int x = 50;
   int y = 0;
   double z = 0;
   try
   {   z = division(x, y);
       cout << z << endl;
   }
   catch (const char* msg)
   {   cout << msg << endl;
   }
```

A la part del **try** posem el codi que volem executar i que eventualment pot llençar una excepció, a la part del catch posarem el codi que recollirà l'excepció i eventualment realitzarà una acció. Quan passa una excepció mentre s'executa el codi del try aquest llença una excepció (la podem llençar nosaltres al propi codi amb un **throw** o el sistema a l'hora d'allocatar memòria etc.). Un cop es llença l'excepció el control es transfereix al handler de l'excepció. Si no es produeix cap excepció el codi segueix executant-se i s'ignora la part dels handlers. Si l'excepció no te un handler associat es pararà l'execució del programa com si no estiguéssim fent control d'excepcions.

A l'exemple al try cridem a una funció de divisió, si tot va correctament només s'executarà aquesta part del codi i donarà el resultat. Però si li passem un 0 ens retornarà una excepció en forma de missatge de text. En aquest moment el control es pasarà al handler del missatge que està a la part del catch i escriurà el missatge rebut.

Exemple alocatació de memòria

```
int main ()
{ ...
  try {  ...
        //crides a diferents mètodes que alocaten memòria
         //i a mètode que llença una excepció en forma de texte.
  }
  catch (std::bad_alloc) {
        cout << "Error allocatar memòria"<< endl;
  }
  catch (const char* msg) {
        cout << msg << endl;
  }
```

• • •

A l'exemple de l'alocatació de memòria veiem que podem concatenar diversos handlers que recullin diferents excepcions. A l'exemple una excepció d'alocatació de memòria i un missatge.

Volem que programeu utilitzant excepcions simples.

La idea es que si es produeix una excepció dins d'un codi retornem un char* amb un missatge que ens indiqui que ha passat de manera que qui ha cridat al mètode pugui recuperar-lo. A més recollirem altres excepcions que es poden produir com per exemple que hi hagi un problema a l'hora d'allocatar memòria (std::bad_alloc) o accedir a una posició d'un vector (std::out_of_range).

Trobareu més informació a: http://www.cplusplus.com/doc/tutorial/exceptions/