


IFT2015 :: autumn 2017

Duty 2: Connectivity in the Erdős-Rényi model

Introduction

It has been 60 years since Pál Erdős and Alfréd Rényi discovered an exciting fact about the connectivity of random graphs ["On random graphs I." *published in Mathematicae* . 6 : 290-297, 1959.  PDF (http://www.renyi.hu/~p_erdos/1959-11.pdf)]. They consider the following problem: if we add random one-to-one edges between n nodes, when does the graph become connected? If we denote this number by $v(n)$, we find that $(v(n) - \frac{1}{2}n \ln n)/n \rightarrow 0$ (almost surely). In other words, the average degree in the connected graph will be $\ln(n)$.

This lab examines the theoretical prediction by measuring $v(n)$ in simulations. We need software that tracks connectivity in a random graph with the addition of edges. It also gives us the perfect opportunity to compare different implementations of union-find.

The TP targets the following skills:

- implementation and use of an advanced data structure (union-find)
- experimental design for empirical validation of a theoretical prediction
- Graph manipulation (related components)



Modalities

Work in teams of 2.

Use freely available code on the Internet or other medium, but credit the source!

Deadline for submission: October 31, 23:59.

Description

A. Simulation

We want to implement the following test using a membership-union data structure (input: number of nodes n):

1. initialize union-find for n elements, n empty adjacency lists, and $N = 0$
2. repeat while the graph does not become connected:
 1. choose x and y at random ($x \neq y$)
`// index: java.util.Random.nextInt(n)`
 2. if xy is not yet an edge in the graph (check on adjacency list), then add it to the graph, increment N , and call `union(x, y)`.
3. return N

Hint: I sketched the implementation on [github \(https://github.com/csurosm/IFT2015-A17/tree/master/src/unionfind\)](https://github.com/csurosm/IFT2015-A17/tree/master/src/unionfind) .

One can check if the graph becomes connected in the operation `union`:

(1) either one maintains the size of the groups with the identical elements (a table size $[0 .. n-1]$, v. [Sedgewick book](#) (<http://algs4.cs.princeton.edu/15uf/>)), and then the graph becomes connected if the size reaches n ,

(2) either we maintain the rank and the number of connected components (a table rank [0 .. n -1], v. [course notes](https://ift2015a17.files.wordpress.com/2017/10/ift2015-handout08-unionfind.pdf) (<https://ift2015a17.files.wordpress.com/2017/10/ift2015-handout08-unionfind.pdf>) + a counter), and then the graph becomes connected when the number of components drops to 1.

B. Path compression efficiency

We would also like to examine how path compression changes performance. In order to characterize the calculation time, we count the accesses to the cells of the array (of parents in union-find). "Access" includes reading (`y=parents[x]`) and writing (`parent[x]=y`).

Hint: the easiest way is to integrate the counters in the union-find code:

```

1  package unionfind;
2  class UnionFind
3  {
4      private final parent[]; // tableau de parents
5
6      public UnionFind(int n){ this.parent = new int[n]; ... }
7
8      private static final boolean DEBUG_COUNT_OPERATIONS=true;
9      private long cnt_get=0L; // long car on veut compter après 2^31
10     private long cnt_set=0L;
11     long getCountGet(){ return cnt_get;}
12     long getCountSet(){ return cnt_set;}
13
14     private int getParent(int x)
15     {
16         if (DEBUG_COUNT_OPERATIONS) cnt_get++;
17         return parent[x];
18     }
19     private void setParent(int x, int p)
20     {
21         if (DEBUG_COUNT_OPERATIONS) cnt_set++;
22         parent[x]=p;
23     }
24     public int find(int x)
25     {
26         int y = getParent(x);
27         while (y != x)
28         { x=y; y=getParent(x); } // pas de compression de chemin
29         return y;
30     }
31     ...

```

C. Work to be done

1. Code for simulation

Implement the code for the simulation sketched in A.

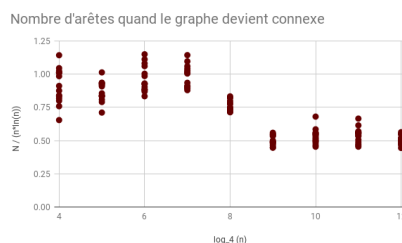
2. Path compression

Implement two or three versions of find: without path compression, and at least one other with path compression (full or half), with a mechanism to count access operations to the parent array until that the graph becomes connected.

3. Empirical Study

Performez a series of experiments with versions implemented on $n = 2^k$, $k = 8, 10, 12, \dots, K$ max Repeat each combination of k and release the same number of times (at least 5). Choose a $K_{max} \geq 24$. Show the result of the experiments on two graphs:

1. n and $v(n)$: abscissa = n on logarithmic scale; ordinate = $\frac{v(n)}{n \ln n}$



2. n and cost of operations: abscissa = n on logarithmic scale; ordinate = number of access to the array / number of union calls = $\frac{n_{acc}}{v(n)}$.



(Illustrations for a different random connectivity model.)

Hint : write text table + transfer to spreadsheet software (Microsoft Excel, Google Spreadsheet), add cells with appropriate formulas + create and save graphs.

Discount of work


Submit a single JAR file `unionfind.jar` that contains the source code and the classes of C.1 and C.2), as well as the two graphics of C.3 (2 files, PDF or PNG format).

Reflections (do not submit)

- Study the difference between the performances, and draw your own conclusions about the structures. (Above, Method2 and Method1 perform well, almost independently of n (M1 is 50% slower), while Method3 becomes worse with n increasing.)
- Est-ce qu'on a besoin de maintenir des listes des adjacences? Supposons qu'on appelle `union(x,y)` avec x,y tirés au hasard sans tester si c'est une arête répétée. Est-ce que le comportement serait très différent?




Advertisements



Men's Helcor Leather 6-Inch Pre-Work Boot

Our Helcor leather boots have the recognizable styling of our original waterproof boots, but in...
\$190
SHOP NOW

[Men's Custom 6-Inch Pre-Work Boot](#)



Men's Helcor Leather 6-Inch Pre-Work Boot

Our Helcor leather boots have the recognizable styling of our original waterproof boots, but in...
\$190
SHOP NOW

[Men's Custom 6-Inch Pre-Work Boot](#)

☐ octobre 19, 2017 ☐ csurosm ☐ devoir

10 réflexions sur “Devoir 2: Connexité dans le modèle Erdős-Rényi”

1. Mathieu Matos dit :

octobre 24, 2017 à 3:38

Est-ce normal que nous sommes pas capables de faire une simulation pour des k approchant 24? Ça nous donne « OutOfMemoryError ». La liste d'adjacence prend trop de mémoire dans ces cas, y-a-t'il une autre façon de s'y prendre?

Répondre

◦ *amir.sh* dit :

octobre 24, 2017 à 9:33

when your conclusion from memory analysis or from reading the Plumb report are that memory use is legal and there is nothing to change in the source code, you need to allow your JVM more Java heap space to run properly. In this case, alter your JVM launch configuration and add (or increase the value if present) the following:

`-Xmx1024m`

The above configuration would give the application 1024MB of Java heap space. You can use `g` or `G` for GB, `m` or `M` for MB, `k` or `K` for KB. For example all of the following are equivalent to saying that the maximum Java heap space is 1GB:

```
java -Xmx1073741824 com.mycompany.MyClass
java -Xmx1048576k com.mycompany.MyClass
java -Xmx1024m com.mycompany.MyClass
java -Xmx1g com.mycompany.MyClass
```

Répondre

◦ csurosm dit :

octobre 25, 2017 à 2:18

Le switch -Xms est également utile ici : c'est le minimum du heap space. Typiquement, quand le Java Virtual Machine se lance, elle commence à allouer plus et plus d'espace comme nécessaire jusqu'à -Xmx, mais cela peut toujours donner une OutOfMemoryError si Xmx est proche du RAM sur l'ordinateur (pensez allocation dynamique: au moment de copier n -> capacité $2n$, on occupe $3n$ cases). Dans des applications voraces de mémoire, je fais simplement -Xms et -Xmx avec la même quantité de mémoire: java -Xms4096M -Xmx4096M ...

Comme ça, JVM utilise exactement autant de mémoire dès le départ.

Répondre

2. levent1012 dit :

octobre 25, 2017 à 12:01

Bonjour,

J'ai aussi une question par rapport de la liste d'adjacence. C'est nécessaire d'utiliser la liste d'adjacence pour ce devoir? En fait, c'est très clair qu'on n'en a pas besoin si on utilise le tableau parent[0...n-1].

Répondre

◦ csurosm dit :

octobre 25, 2017 à 2:27

C'est une des questions que je vous propose sous «Réflexions».

Sans listes d'adjacence, on peut appeler union(x,y) ou union(y,x) deux fois: dans le modèle ER, on n'a pas le droit de choisir la même arête (entre x et y) plus qu'une fois. Donc oui, c'est nécessaire pour implémenter le modèle. Ce qui se passe dans le modèle non-ER, où on fait des union() sans être gêné par répétitions, c'est la question. Tirez votre conclusion, on pourra en discuter au cours avec tout le monde.

Répondre

◦ jcamirand dit :

octobre 29, 2017 à 9:10

Êtes-vous certain qu'il est nécessaire d'avoir une liste d'adjacence pour vérifier qu'une arête n'existe pas déjà? J'ai l'impression avoir une solution fonctionnelle sans une telle liste.

Répondre

3. walid dit :

octobre 26, 2017 à 9:33

salut , est ce que quelqu'un cherche un binôme pour ce devoir ?

Répondre

4. Abdramane Diasso dit :

octobre 29, 2017 à 12:52

Bonjour

Je comprend pas l'intérêt d'avoir une nouvelle instance de UnionFind dans cette methode getTransitionGiantComponent. Aussi pour deux des versions de mes methodes find bien souvent j'ai des boucles infinies.

Répondre

5. Nathan Renaud dit :

octobre 30, 2017 à 5:53

Est-il normal que le temps nécessaire à la construction d'un graphe convexe avec 2^{24} noeuds soit très élevé? Quel serait un temps et un nombre d'instructions raisonnable de résolution selon vous? Je ne vois pas de différence notable entre les trois modèles que j'ai implémenté. Merci.

Répondre

6. csurosm dit :

octobre 31, 2017 à 9:52

Il est probable que la maintenance d'une grande structure pour les listes d'adjacence prend bcp plus de temps (réel = wallclock) que la maintenance de la structure UF. Faites attention, s'il y a un ou deux Objects (incluant Integer) créés par arête, c'est 24-48 octets. Mais autrement on attend que le temps se multiplie proche à : $2^{24} \ln(2^{24}) / 2^{22} \ln(2^{22}) = 4 \cdot (24/22)$ donc juste un peu plus que 4 quand on va de 2^{22} à 2^{24} . Si c'est plus, il y a trop de swapping entre le mémoire vive et le disque (voir commentaire en haut pour -Xmx -Xms) ou une erreur dans le code.

Répondre

