DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 2

16 de Mayo de 2016 — Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Ciruelos Rodríguez, Gonzalo	063/14	gonzalo.ciruelos@gmail.com
Costa, Manuel José Joaquín	035/14	manucos94@gmail.com
Gatti, Mathias Nicolás	477/14	mathigatti@gmail.com
Maddonni, Axel Ezequiel	200/14	axel.maddonni@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Ejercicio 1

2. Ejercicio 2

3. Ejercicio 3

4. Ejercicio 4

5. Ejercicio 5

6. Ejercicio 6

7. Ejercicio 7

8. Apéndice

8.1. Generación de grafos conexos aleatorios

Algorithm 1 Pseudocódigo del procedimiento para generar grafos conexos al azar

```
1:  procedure GRAFO_RANDOM(int n, int m)→ Grafo
2:      k_n ← {(0,1),(0,2),...,(0,n),(1,2),(1,3),...,(n−2,n−1)}
3:      vertices ← {random.range(0,n)}                ▷ Empiezo con un vértice al azar
4:      agm ← {}
5:      while vertices.size() < n do
6:          aristas ← "aristas (u,v) de k_n tal que u ∈ vertices y v ∉ vertices o viceversa"
7:          arista_nueva ← random.choice(aristas)
8:          agm.add(arista_nueva)
9:          k_n.remove(arista_nueva)
10:         vertices.add("extremo de arista_nueva que no estaba en vertices")
11:             ▷ Cuando termina este ciclo tenemos un árbol de n vertices y n − 1 aristas
12:     grafo ← agm
13:     while grafo.size() < m do
14:         arista ← random.choice(k_n)
15:         grafo.add(arista)
16:         k_n.remove(arista)
17:     for arista ∈ grafo do
18:         peso(arista) ← random.random()
        return grafo
```

El algoritmo, se basa en generar un grafo conexo minimal (es decir, un árbol) de n vértices. Para lograr esto, técnicamente lo que hacemos es empezar con K_n, es decir, el grafo completo de n vértices, con todos sus aristas de igual peso, y le encontramos un árbol generador mínimo utilizando Prim. Todo esto es obviamente trivial en este caso, dado que todas las aristas tienen igual peso, así que básicamente lo que hacemos es elegir una arista al azar en cada paso.

Luego, una vez que tenemos el árbol terminado, lo completamos con aristas al azar, hasta llegar al objetivo de m aristas.

Finalmente, se eligen pesos al azar para cada arista.

8.2. Partes relevantes del código