És més ràpid Intel C++ o GCC?

Àlex Carruesco Llorens, Oriol Corcoll Andreu, Cristina Hurtado Obiols
Probabilitat i estadística, grup 11
Desembre 2011

Resum

Objectiu: Comparar l'eficiència entre els compiladors *Intel C++ Compiler* i *GNU Compiler* per a veure quin dels dos realitza la compilació òptima.

Mètodes: Hem generat 100 matrius amb un dibuix format per els caràcters '.' i 'X' i hem executat el programa de fractals compilat amb els dos compiladors. Un cop realitzades les proves, hem aplicat el logaritme natural a les dades per a poder estudiar-les amb la Distribució Normal. Amb la nova transformació aplicada, hem comprovat les premisses adients per a poder utilitzar els mètodes estadístics de la D. Normal i hem posat a prova la diferència del logaritme de temps de les dades per a veure si existien diferències entre els temps d'execució dels compiladors.

Resultats: El temps mig d'execució per al programa compilat amb GNU Compiler ha estat de 1,706633 segons, i el del compilat amb Intel C++ Compiler ha estat de 1,280915 segons. Rebutgem H_0 (que diu que no hi ha diferència entre els temps d'execució) i, per tant, sí que observem una diferència entre l'eficiència dels compiladors.

Conclusió: Segons el nostre estudi, hem vist que el compilador Intel C++ Compiler és un 24,64% millor que el GNU Compiler, amb un IC =[23.65, 25.61]. Però aquest resultat pot ser degut a que hem recopilat les dades d'execució sobre una màquina amb arquitectura Intel. Caldria fer la prova sobre diferents computadors si volguéssim generalitzar la conclusió i tenir en compte altres factors.

Introducció

En l'àmbit de la informàtica s'utilitzen compiladors per a poder crear executables a partir de codis de programes i una característica molt important que es valora abans d'escollir un compilador és l'eficiència d'aquest, és a dir, les optimitzacions que aportarà el compilador al codi que faran que el programa s'executi més ràpidament. Això fa que un compilador sigui un punt crític per a l'eficiència d'un programa.

Objectiu

Aquest motiu és el que ens ha impulsat a realitzar un estudi per a comparar dos dels compiladors de C++ més utilitzats: el *GNU Compiler* i el *Intel C++ Compiler*. Hem decidit provar-los sobre una màquina amb arquitectura Intel per veure si el compilador GCC de GNU (que és el més utilitzat) realment genera codi més eficient que el compilador de Intel o al revés.

Material i mètodes

Per a estudiar les diferències de la compilació entre els 2 compiladors hem utilitzat un programa que dibuixa fractals, elaborat per nosaltres, i per a poder generar les dades hem utilitzat un script (també programat per nosaltres) amb el qual hem generat un fitxer que conté 100 matrius (formades per els caràcters '.' i 'X') per a executar amb els programes compilats. Aquestes proves han estat generades amb funcions de les llibreries estàndard de python que proporcionen nombres aleatoris. A més, l'script de generació també s'encarrega d'executar les 100 proves amb cada programa i, aleatòriament, n'alterna l'ordre d'execució per a evitar que la càrrega del Sistema Operatiu afecti als resultat obtinguts.

Per a la recollida de dades hem utilitzat els compiladors Intel C++ Compiler i GNU Compiler per compilar el programa de fractals. Un cop obtinguts els dos executables, hem executat l'script generador sobre un computador amb un IntelCore i7-960 de 4 nuclis a 3,2 GHz, 8GB de RAM DDR3 i Ubuntu 10.10. La recollida de dades es va dur a terme el dia 22/11/2011.

Per a l'estudi de les dades hem utilitzat el software estadístic R v.2.13.1, de la companyia *The R Foundation for Statistical Computing*.

Al recollir el temps d'execució de cada executable hem vist amb les gràfiques de dispersió de punts que les dades aparentaven algun tipus de funció exponencial. Així que hem decidit treballar amb el logaritme de temps per linealitzar les mostres y poder estudiar les dades amb la Distribució Normal.

Per a referir-nos al programa compilat amb el GNU Compiler, parlarem de a1, i per a referir-nos al compilat amb l'Intel C++ Compiler, parlarem de a2. Les variables recullen el temps d'execució dels programes per a les 100 matrius mesurat en segons.

I. Diferència dels temps d'execució

Per a poder veure si l'eficiència dels dos compiladors és diferent, hem utilitzat la diferència entre els temps d'execució dels dos programes compilats amb cadascun dels compiladors. Per aquest motiu, hem creat una variable a3, que correspon a la diferència dels logaritmes naturals del temps (a3 = log(a1)-log(a2)).

II. Premisses convenients

1. Normalitat per a la variable diferència.
2. Aleatorietat a les mostres, és a dir, proves generades amb valors a l'atzar. Com generem les mostres amb un script que utilitza números aleatoris, assegurem aquesta premissa.

III. Plantejament de la prova d'hipòtesi (bilateral):

$$\begin{cases} H_0: \mu_{a3} = 0 \\ H_1: \mu_{a3} \neq 0 \end{cases}$$

IV. Càlcul de l'estadístic:

Les dades no són independents i per tant realitzem un test aparellat. Utilitzem l'estadístic z ja que es compleix la premissa n>=100.

$$z_{a3} = \frac{\overline{a3}}{s/\sqrt{n}} \sim N(0,1)$$

On *s* és l'estimació de la desviació estàndard de la diferència (a3) i *n* és la grandària mostral.

V. Estudi del p-valor (α = 0.01)

Si p-valor < α, llavors podem rebutjar la hipòtesi nul·la, amb una confiança del (1-α)%.

VI. Interval de confiança per a la diferència

Amb aquesta fórmula, podem donar un interval que contindrà la mitjana de a3 amb una confiança del (1-α)%.

$$IC(a3, 1-\alpha) = \overline{a3} \pm z_{0,995} \cdot \frac{s}{\sqrt{n}}$$

Resultats

Descriptiva

La taula 1 conté les mitjanes i desviacions típiques del temps d'execució de cada programa, en segons.

	Mitjana	**Desviació típica**
G++ (a1)	1,706633	3,690189
Intel C++ (a2)	1,280915	2,788935
Diferència (a3)	0,4257181	0,9077492

Taula 1. Estadístics mostrals de les dades

Les dades obtingudes estan representades en aquests gràfics:

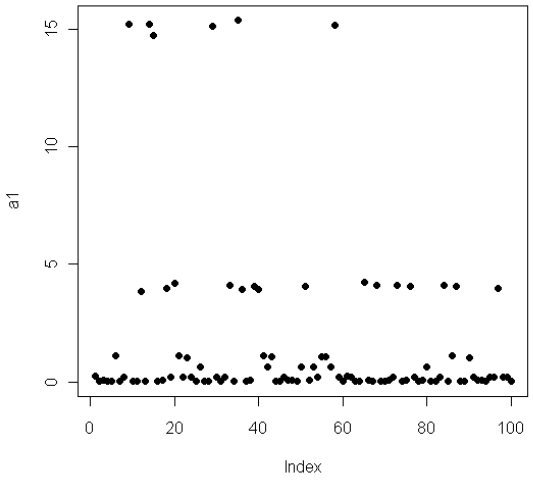


Figura 1. Temps d'execució de les proves de a1

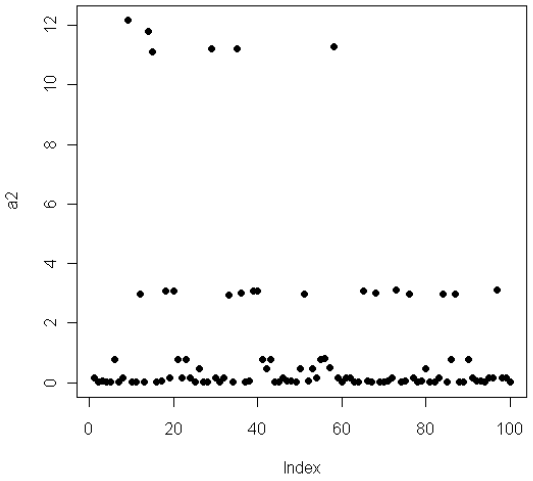


Figura 2. Temps d'execució de les proves de a2

Com es pot veure a les gràfiques, hi ha una mena de "nivells" on s'acumulen bastant els punts. Això es deu a que el cost del programa és exponencial, així que per a matrius petites triga poc, però a la que augmenta una mica la mida de la matriu o els "passos" en la fractal, el temps es dispara.

Inferència

Com hem vist que les mostres no semblaven seguir la Distribució Normal (es pot veure a les gràfiques afegides a l'Annex 1), hem decidit aplicar una transformació amb el logaritme natural del temps. Amb la nova transformació podem observar un millor ajustament a la Distribució Normal perquè, com es pot veure a la QQ-Plot (Figura 3), els quantils teòrics s'ajusten bastant als quantils observats a les mostres. Tot i que també podem veure alguns valors que s'allunyen de la distribució, que en cas de no ser errors, poden ser valors que aportin una informació bastant important sobre l'experiment.

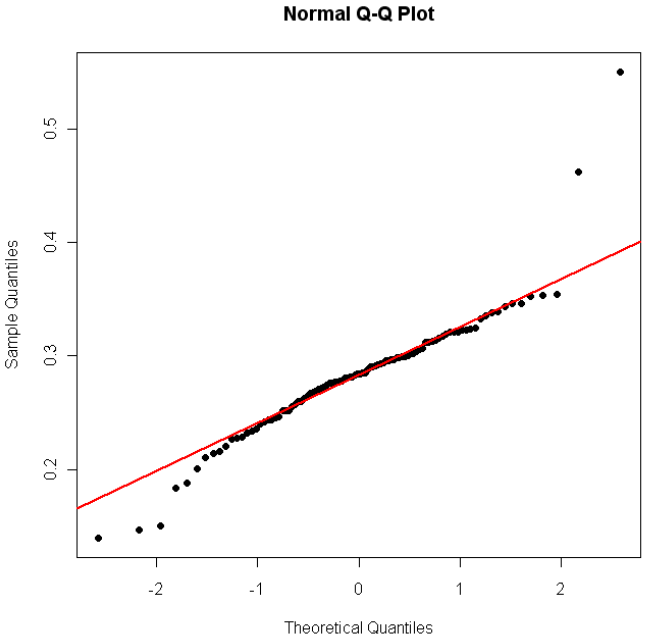


Figura 3. Gràfica QQnorm

Hem calculat la variable diferència a3 per treballar amb la diferència de logaritmes naturals. A partir d'aquesta variable, hem trobat l'estadístic z de la Distribució Normal, amb un valor de 51.4667, que ens ha permès posar a prova la hipòtesi nul·la. Calculant el p-valor de l'estadístic amb R hem obtingut un valor molt pròxim a 0. Això vol dir que podem rebutjar la hipòtesi nul·la amb molta seguretat i, per tant, podem afirmar que hi ha diferència en l'eficiència dels compiladors.

Després, hem calculat un interval de confiança per a ajustar el valor de la mitjana de a3. Com que hem trobat que a2 triga menys que a1 en mitjana i hem vist que la diferència no és 0, podem afirmar amb una confiança del 99% que, en mitjana, a2 triga 0,2828944 log(s) menys que a1, amb un IC=[0.269897, 0.2958919].

Si desfem els logaritmes naturals, podem trobar l'eficiència relativa entre els 2 compiladors:

$$\mathrm{a3} = \ln(a1) - \ln(a2) = 0{,}2828944$$

$$\ln\left(\frac{a1}{a2}\right) = 0{,}2828944$$

$$\frac{a1}{a2} = e^{0{,}2828944} = 1{,}32697$$

$$Eficiència\ relativa = \left(1 - \frac{a2}{a1}\right) \cdot 100 = 24{,}64\%$$

Per tant, el compilador a2 és, en mitjana, un 24,64% més eficient que a1 i, amb una confiança del 99%, podem donar un IC per a aquest percentatge transformant el IC anterior amb la mateixa transformació que acabem de fer per desfer els logaritmes naturals: IC =[23.65, 25.61].

Discussió

Després de realitzar el nostre estudi estadístic hem pogut observar que el compilador Intel C++ Compiler és millor que el GNU Compiler, basant-nos en les dades recollides de les proves de l'experiment. Però aquestes dades poden ser degudes a que hem executat els programes compilats sobre una màquina amb processador Intel amb Sistema Operatiu Ubuntu.

Limitacions

Hi ha factors que molt possiblement influeixen en els resultats del nostre estudi, com els components del computador on s'han fet les proves, o el Sistema Operatiu utilitzat. Però com no sabem estudiar aquests factors amb els coneixements que tenim sobre estadística, ens hem centrat en comparar les mostres fixant aquestes variables.

Un altre factor important ha estat que les nostres matrius de les proves han tingut unes dimensions molt limitades, ja que el programa de fractals té un cost exponencial i trigaria massa temps en acabar les execucions si fossin de mides grans.

Treball futur

Per a complementar aquest estudi es podrien realitzar les proves sobre diversos PCs amb diferents arquitectures, i així poder generalitzar i comprovar quin compilador és millor, evitant la influència que pot tenir l'arquitectura sobre les optimitzacions dels compiladors. També es podria variar el SO per a evitar les influències que aquest pugui causar sobre els resultats.

Annex 1 – Estudi de les variables

Hem pogut generar aquestes gràfiques a partir de les dades:

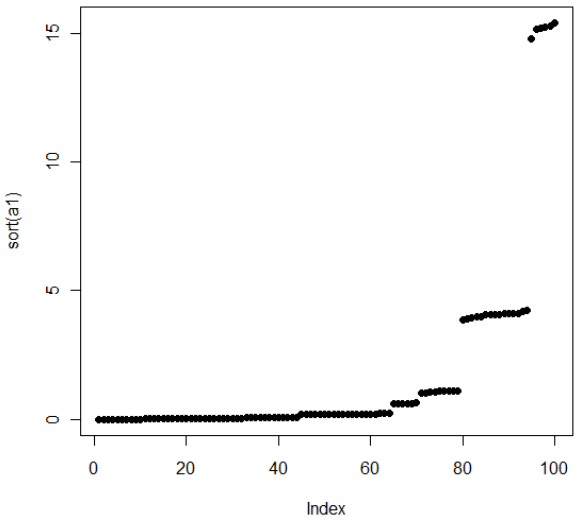


Figura 4. Gràfica de a1

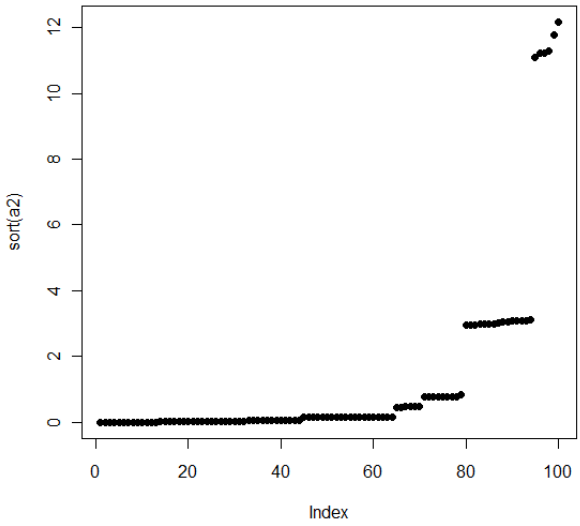


Figura 5. Gràfica de a2

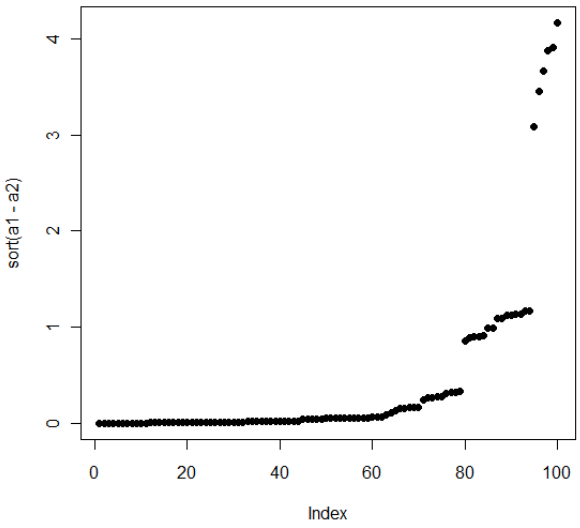


Figura 6. Gràfica de a3

Com es pot observar, les dades formen una gràfica que sembla aproximar-se a una funció exponencial. Com no sabem estudiar distribucions d'aquest tipus, hem decidit aplicar el logaritme natural a les variables, ja que llavors aconseguim unes gràfiques bastant lineals, les quals podem estudiar-les amb la Distribució Normal.

Els salts que es poden observar a les gràfiques són deguts a que les matrius per al programa de fractals són de mides de valors naturals. Per aquest motiu, com el programa té un cost exponencial, augmentant una unitat les dimensions de la matriu, augmenta molt el temps que triga el programa en processar les dades.

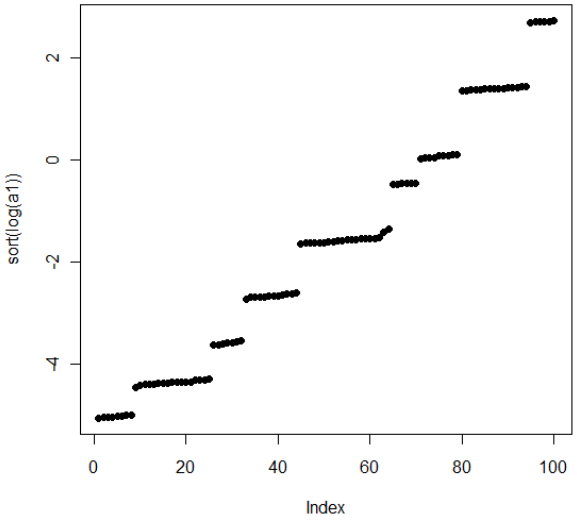


Figura 7. Gràfica de log(a1)

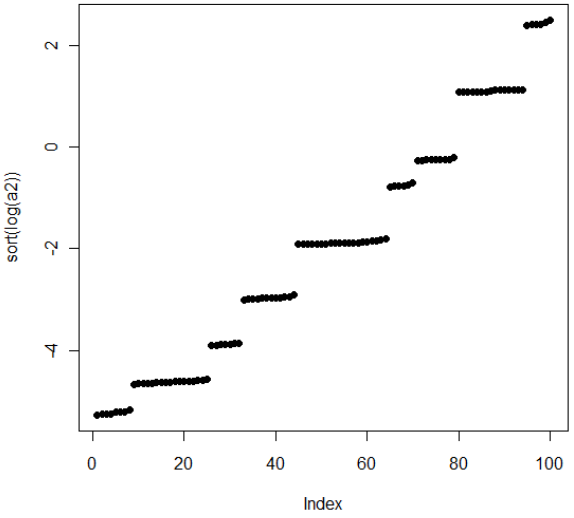


Figura 8. Gràfica de log(a2)

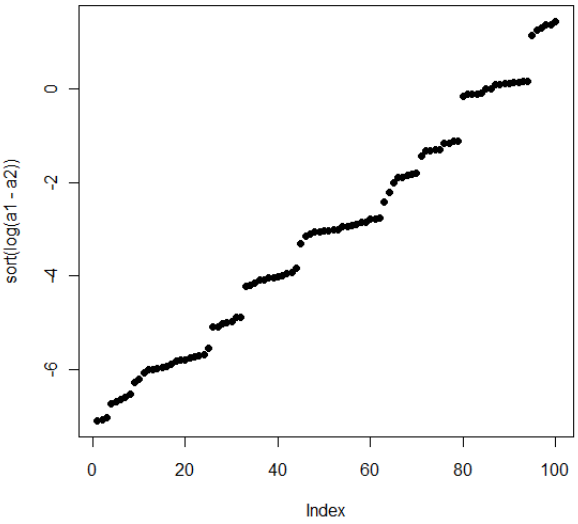


Figura 9. Gràfica de log(a3)

En aquestes gràfiques veiem que les dades formen una gràfica bastant lineal, tot i que podem observar salts deguts al motiu explicat anteriorment.

Annex 2 – Script en R

```
# Inserció de les dades en les variables a1 i a2

a1<-c(0.258479, 0.012425, 0.069353, 0.013320, 0.027892, 1.093978, 0.012855,
0.214034, 15.251585, 0.012533, 0.027185, 3.869372, 0.012339, 15.239102,
14.763268, 0.012346, 0.067549, 3.982882, 0.197413, 4.200502, 1.098165, 0.197653,
1.018726, 0.198593, 0.012121, 0.630553, 0.012797, 0.027821, 15.131950, 0.212077,
0.012706, 0.199264, 4.114807, 0.026681, 15.388718, 3.914925, 0.006312, 0.067808,
4.076630, 3.932327, 1.104757, 0.615704, 1.047787, 0.028686, 0.006450, 0.211778,
0.072756, 0.070202, 0.011435, 0.627741, 4.083280, 0.064434, 0.626246, 0.197785,
1.050355, 1.088686, 0.627465, 15.170997, 0.214262, 0.013436, 0.241160, 0.196718,
0.006571, 0.013243, 4.235897, 0.068713, 0.006497, 4.112308, 0.011861, 0.012624,
0.073216, 0.207094, 4.099445, 0.027361, 0.068064, 4.066480, 0.210966, 0.026593,
0.067712, 0.615798, 0.006648, 0.012836, 0.195706, 4.122786, 0.006369, 1.106221,
4.059722, 0.012572, 0.006425, 1.039898, 0.203941, 0.071964, 0.069267, 0.013226,
0.204227, 0.218997, 3.980866, 0.193057, 0.210336, 0.006644)
a2<-c(0.148996, 0.009851, 0.051837, 0.009422, 0.020233, 0.784605, 0.009799,
0.150244, 12.161522, 0.009837, 0.020528, 2.958996, 0.009815, 11.783324,
11.099272, 0.009355, 0.051637, 3.084388, 0.149014, 3.068605, 0.783217, 0.149169,
0.779632, 0.149276, 0.009772, 0.463875, 0.010327, 0.020943, 11.218142, 0.150418,
0.009933, 0.155832, 2.943305, 0.020511, 11.223972, 3.019693, 0.005450, 0.052716,
3.089611, 3.081525, 0.781271, 0.466328, 0.772802, 0.020992, 0.005137, 0.157921,
0.054407, 0.050740, 0.009521, 0.478835, 2.962284, 0.049679, 0.467489, 0.148991,
0.776287, 0.824489, 0.493566, 11.289255, 0.152626, 0.010111, 0.151928, 0.149786,
0.005379, 0.009786, 3.066712, 0.050614, 0.005653, 2.994789, 0.009827, 0.009992,
0.051390, 0.162319, 3.110687, 0.020335, 0.051048, 2.974716, 0.153294, 0.020412,
0.050621, 0.456212, 0.005170, 0.009517, 0.148762, 2.989271, 0.005480, 0.777680,
2.972065, 0.009493, 0.005155, 0.773254, 0.151249, 0.052118, 0.051467, 0.010034,
0.151497, 0.163184, 3.095341, 0.156463, 0.152225, 0.005251)

# Observació de les mostres

plot(sort(a1), pch=19)
plot(sort(a2), pch=19)
plot(sort(a1-a2), pch=19)
plot(sort(log(a1)), pch=19)
plot(sort(log(a2)), pch=19)
plot(sort(log(a1-a2)), pch=19)

# Obtenció de la variable diferència de logaritmes naturals de temps

a3 = log(a1)-log(a2)
```

```
# Obtenció de les mitjanes i les desviacions típiques mostrals de les variables

mean(a1)
sd(a1)
mean(a2)
sd(a2)
mean(a3)
sd(a3)

# Contrast de la hipòtesi nul·la utilitzant el p-valor

z = mean(a3)/(sd(a3)/sqrt(length(a3)))
pvalor=2*(1-pnorm(z,0,1))

# Comprovem la prova d'hipòtesi amb la funció que ens proporciona el software R

t.test(log(a1),log(a2),paired=TRUE)

# IC de la diferència de logaritmes de temps

zcritic=qt(0.99,length(a3))
ICinf=mean(a3)-zcritic*(sd(a3))/sqrt(length(a3))
ICsup=mean(a3)+zcritic*(sd(a3))/sqrt(length(a3))

# Gràfica per justificar la premissa de normalitat

qqnorm(a3,pch=19)
qqline(a3,lwd=2,col=2)
```

Annex 3 – Programa de fractals en C++

Aquest programa dibuixa una fractal a partir d'un patró (llavor o *seed*) representat en una matriu de caràcters ('.' i 'X') i en realitza k passos, on k forma part de l'entrada del programa.

```
#include<iostream>
#include<vector>
using namespace std;
typedef vector<char> Fila;
typedef vector<Fila> Matriu;

voidllegirmatriu(Matriu& M){
	for (int i = 0; i <M.size(); ++i){
		for (int j = 0; j < M[i].size(); ++j) cin>> M[i][j];
	}
}

voidescriurematriu(const Matriu& M){
	for (int i = 0; i <M.size(); ++i){
		for (int j = 0; j < M[i].size(); ++j) cout<< M[i][j];
		cout<<endl;
	}
	cout<<endl;
}

int pot(int n, int k){
	if (k == 1) return n;
	else{
		int p = pot(n,k/2);
		if (k%2 == 0) return p*p;
		return p*p*n;
	}
}

void matrius(const Matriu&aux, Matriu& res, int i, int j){
	for (int k = 0; k <aux.size(); ++k){
		for (int l = 0; l <aux[k].size(); ++l){
			res[k+i][l+j] = aux[k][l];
		}
	}
}

Matriu fractal(const Matriu& M, int n, int m, int k, int pot1, int pot2){
	if (k == 1) return M;
	else{
```

```
                Matriu res(pot1,Fila(pot2,'.'));
                Matriu aux(pot1/n,Fila(pot2/m));
                pot1 = pot1/n;
                pot2 = pot2/m;
                aux = fractal(M,n,m,k-1,pot1,pot2);
                for (int i = 0; i <res.size(); i += pot1){
                        for (int j = 0; j < res[i].size(); j += pot2){
                                if (M[i/pot1][j/pot2] == 'X') matrius(aux,res,i,j);
                        }
                }
                return res;
        }
}

int main(){
        int n, m;
        while (cin>> n){
                cin>> m;
                Matriu M(n,Fila(m));
                llegirmatriu(M);
                int k;
                cin>> k;
                int pot1 = pot(n,k), pot2 = pot(m,k);
                Matriu res(pot1,Fila(pot2));
                res = fractal(M,n,m,k,pot1,pot2);
                escriurematriu(res);
        }
}
```

Annex 4 – Script en python per a la generació de proves i l'execució dels programes

Aquest script s'encarrega de generar les 100 proves, que estan formades per matrius de caràcters omplertes amb '.' i 'X' (on les X formen un dibuix d'una fractal) i per un valor k que determina el nombre de passos que ha de fer el programa per a construir la fractal desitjada. També s'encarrega d'executar els programes alternant l'ordre per a evitar que aquest pugui influir en els resultats. Finalment, crea un arxiu que conté el temps d'execució de cada programa per cada prova.

```
import os
import sys
import timeit
import random

defexec_prog(prog):
        #Ejecutamos el programa con los ejemploscreados
        os.system("./%s <exemp> /dev/null" % prog)

defcreate_ex(n, limit_min, limit_max, limit_depth):
        f = open("exemp", 'w+')
        for x in range(n):
                random.seed()
                m = random.randint(limit_min, limit_max)
                random.seed()
                m2 = random.randint(limit_min, limit_max)
                f.write('%d %d\n' % (m, m2))
                make = lambda : ''.join(['X' ifrandom.randint(1, 1000)%2 else '.' for y in
                range(0, m2)])
                f.writelines('\n'.join([make() for y in range(0, m)]))
                random.seed()
                depth = random.randint(4, limit_depth)
                f.write('\n%s\n\n' % depth)
        f.close()

if __name__ == '__main__':
        n_execs = 100    #Numero de ejecuciones de a1 y a2
        n_exemp = 1     #Numero de ejemplo por ejecucion
        limit_min = 4   #Limiteminimo de columnas y filas
        limit_max = 5   #Limitemaximo de columnas y filas
        limit_depth = 6 #Limite de la profundidad
        progs = ['a1.out', 'a2.out']
```

```
#Ejecutamos el programa en ordenaleatorio
for x in range(0, n_execs):
        #Creamos los ejemplos
        create_ex(n_exemp, limit_min, limit_max, limit_depth)
        random.shuffle(progs)
        aux_timeit = lambda p: timeit.Timer("exec_prog('%s')" % p, "from
        __main__ import exec_prog").timeit(1)
        print ['Time %s: %f' % (p, aux_timeit(p)) for p in progs]
```