Random math stuff

Quimey Vivas

Training camp - FCEN-UBA

6 de agosto de 2014

Exponenciacion rapida para un tipo *tipo* que admite producto y neutro para el producto (o sea el 1).

```
tipo exp(tipo a,int b)
{
  if(b==0)return (tipo)(1);
  if(b%2)return b*exp(a,b-1);
  tipo c=exp(a,b/2);
  return c*c;
}
```

Algoritmo de primalidad de Rabin-Miller (supone n impar), la complejidad es $O(\log^3 n)$

```
bool es_primo_probable(int n,int a)
{
  int d=n-1,s=0;
  while(d%2==0)
  {
    d/=2;
    s++;
  }
  int x = expmod(a,d,n);
  if(x==1 or x+1 == n)return true;
  for(int i=0;i<s-1;i++)
  {
      x=(x*x)%n;
      if(x == 1) return false;
      if(x+1 == n)return true;
  }
  return false;
}
```

Listas de testigos: Si $n < 2.152.302.898.747$, alcanza con probar $a = 2$, 3, 5, 7, y 11.

Algoritmo de factorización Rho de Pollard.

```c
int buscar_factor(int n)
{
  int x=2,y=2,d=1;
  while(d==1)
  {
    x=f(x);
    y=f(f(y));
    d=gcd(x-y,n)
  }
  return d;
}
```

Suma

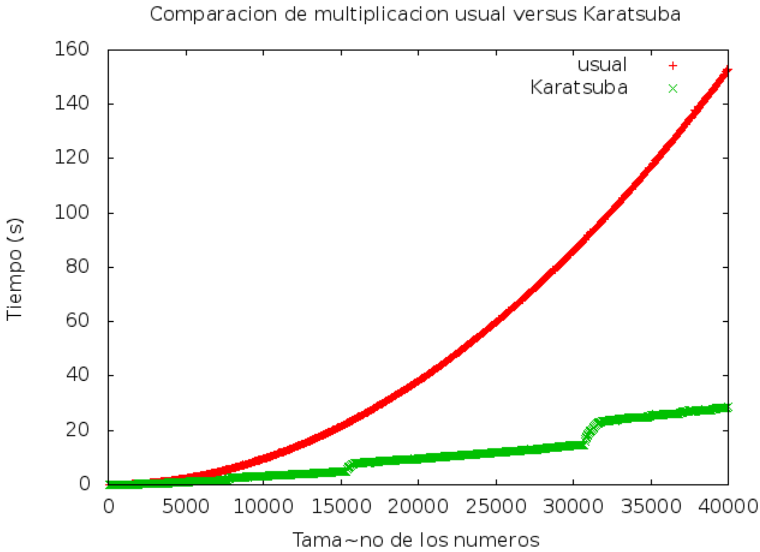
```
vector<int> suma(vector<int> const &x,vector<int> const &y)
{
    int n=x.size(),m=y.size();
    vector<int>z(max(n,m));
    forn(i,n)z[i]+=x[i];
    forn(j,m)z[j]+=y[j];
    //while(not z.back())z.pop_back();
    return z;
}
```

Multiplicacion

```
vector<int> naive(vector<int> const &x,vector<int> const &y)
{
    int n=x.size(),m=y.size();
    vector<int>z(n+m-1);
    forn(i,n)forn(j,m)z[i+j]+=x[i]*y[j];
    return z;
}
```

Multiplicación de Karatsuba

```
//suponemos que xx e yy tienen el mismo tama o
 vector<int> krec(vector<int> const &xx,vector<int> const &yy)
{
    int t=(xx.size()+1)/2,s=xx.size()-t;
    if(t<16)return naive(xx,yy);
    vector<int>ax(t),bx(s),ay(t),by(s);
    ax.assign(xx.begin(),xx.begin()+t);
    bx.assign(xx.begin()+t,xx.end());
    ay.assign(yy.begin(),yy.begin()+t);
    by.assign(yy.begin()+t,yy.end());
    vector<int> aux1 = krec(ax,ay);
    vector<int> aux2 = krec(resta(bx,ax),resta(by,ay));
    vector<int> aux3 = krec(bx,by);
    vector<int> ans(2*(t+s)-1);
    forn(i,2*t-1)ans[i]+=aux1[i];
    forn(i,2*t-1)ans[i+t]+=aux1[i]-aux2[i];
    forn(i,2*s-1)ans[i+t]+=aux3[i];
    forn(i,2*s-1)ans[i+2*t]+=aux3[i];
    return ans;
}
```


Comparacion de multiplicacion usual versus Karatsuba
usual
Karatsuba
Tiempo (s)
0
20
40
60
80
100
120
140
160
0
5000
10000
15000
20000
25000
30000
35000
40000
Tama~no de los numeros

```
unsigned int aa[300000],bx[300000],by[300000];
unsigned long long wl[300000];
const unsigned int m=(3<<18)+1;
const unsigned int e=1<<18;
void fft(unsigned int *a,unsigned int *b,const int d,const int l)
{
    if(l==1)
    {
        b[0]=a[0]%m;
        return;
    }
    unsigned int f[l];
    unsigned int u[l];
    forn(i,l/2)
    {
        f[i]=a[2*i];
        f[l/2+i]=a[2*i+1];
    }
    fft(f,u,d+1,l/2);
    fft(f+l/2,u+l/2,d+1,l/2);
    forn(i,l/2)
    {
        b[i]=(u[i]+wl[((e/l)*i)%e]*u[i+l/2])%m;
        b[i+l/2]=(u[i]+(m-wl[((e/l)*i)%e])*u[i+l/2])%m;
    }
}
```

```
void mfft(int n,int *x,int *y,int *z)
{
    if(n<30000){krec(n,x,y,z);return;}
    int w=5;// ra z primitiva
    wl[0]=1;
    forn(i,e)wl[i+1]=(wl[i]*w)%m;
    memset(aa,0,sizeof(unsigned int)*300000);
    forn(i,n)aa[i]=x[i];
    fft(aa,bx,0,e);
    memset(aa,0,sizeof(unsigned int)*300000);
    forn(i,n)aa[i]=y[i];
    fft(aa,by,0,e);
    forn(i,e)aa[i]=((long long)bx[i]*by[i])%m;
    reverse(wl+1,wl+e);
    fft(aa,bx,0,e);
    forn(i,2*n-1)z[i]=(m-(3LL*bx[i])%m)%m;
}
```

Algoritmo de Strassen

$$
\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}
$$

Evolución de la complejidad de multiplicar matrices