# Desafios de Programação Matemática

**Wladimir Araújo Tavares** [1]

[1]Universidade Federal do Ceará - Campus de Quixadá

16 de março de 2017

# Exponenciação Rápida

- Computação recursiva de $a^n$

$$a^n = \begin{cases} 1 & , n = 0 \\ a & , n = 1 \\ (a^{n/2}) & , n \text{ é par} \\ a(a^{n-1/2})2 & , n \text{ é ímpar} \end{cases}$$

# Implementação recursiva

```
double pow(double a, int n){
  if(n==0) return 1;
  if(n==1) return a;
  double t = pow(a,n/2);
  return t*t*pow(a,n%2);
}
```

- Complexidade de tempo : $O(logn)$

# Implementação não recursiva

```
double pow(double a,  int n){
  double ret = 1;
  while(n){
    if(n%2==1) ret *= a;
    a *= a;
    n /= 2;
  }
  return ret;
}
```

# Algebra Linear

- Resolve sistemas de equações lineares
- Inverter uma matriz
- Encontrar o posto de uma matriz
- Computar o determinante de uma matriz
- Todas essas tarefas podem ser realizadas com o método de eliminação de Gauss.

# Eliminação de Gauss

```
#include <stdio.h>
#include <math.h>
#include <vector>

using namespace std;

typedef long double LD;
LD EPS = 1e-8;

typedef struct MATRIX
{
   int n,m;
   vector< vector<LD> > a;
   void resize(int x, int y, LD v=0.0);
   LD Gauss();
   int inverse();
   vector<LD> operator*(vector<LD> v);
   MATRIX operator*(MATRIX M1);
   void show();
}MATRIX;
```

# Eliminação de Gauss

```
void MATRIX::resize(int x, int y, LD v)
{
 n=x;
 m=y;
 a.resize(n);
 for(int i=0; i<n; i++) a[i].resize(m, v);
}
```

# Eliminação de Gauss

```
LD MATRIX::Gauss()
{
  int i,j,k;
  LD det=1.0, r;
  for(i=0;i<n;i++)
  {
   for(j=i, k=-1; j<n; j++) if(fabs(a[j][i])>EPS)
   { k=j; j=n+1; }
   if(k<0) {
     n=0; return 0.0;
   }
   if(k != i) { swap(a[i], a[k]); det=-det; }
   r=a[i][i]; det*=r;
   for(j=i; j<m; j++) a[i][j]/=r;
   for(j=i+1; j<n; j++)
   {
    r=a[j][i];
    for(k=i; k<m; k++) a[j][k]-=a[i][k]*r;
   }
  }
```

# Eliminação de Gauss

```
  for(i=n-2; i>=0; i--)
  for(j=i+1; j<n; j++)
  {
    r=a[i][j];
    for(k=j; k<m; k++) a[i][k]-=r*a[j][k];
  }
  return det;
}
```

# Eliminação de Gauss

```
int MATRIX::inverse()
 // assume n=m. returns 0 if not invertible
 {
   int i, j, ii;
   MATRIX T; T.resize(n, 2*n);
   for(i=0;i<n;i++) for(j=0;j<n;j++) T.a[i][j]=a[i][j];
   for(i=0;i<n;i++) T.a[i][i+n]=1.0;
   T.Gauss();
   if(T.n==0) return 0;
   for(i=0;i<n;i++) for(j=0;j<n;j++) a[i][j]=T.a[i][j+n];
   return 1;
}
```

# Eliminação de Gauss

```
vector<LD> MATRIX::operator*(vector<LD> v)
// assume v is of size m
{
 vector<LD> rv(n, 0.0);
 int i,j;
 for(i=0;i<n;i++)
 for(j=0;j<m;j++)
  rv[i]+=a[i][j]*v[j];
 return rv;
}
```

# Eliminação de Gauss

```
vector<LD> solve(MATRIX& M, vector<LD> v)
// return the vector x such that Mx = v; x is empty if M is not
{
 vector<LD> x;
 MATRIX M1=M;
 if(!M1.inverse()) return x;
 return M1*v;
}
```

# Eliminação de Gauss

```
LD det(MATRIX &M)
// compute the determinant of M
{
 MATRIX M1=M;
 LD r=M1.Gauss();
 if(M1.n==0) return 0.0;
 return r;
}
```

# Teste de primalidade

```
bool is_prime(int n) {
  if (n < 0) return is_prime(-n);
  if (n < 5 || n % 2 == 0 || n % 3 == 0)
    return (n == 2 || n == 3);
  int maxP = sqrt(n) + 2;
  for (int p = 5; p < maxP; p += 6)
    if (n % p == 0 || n % (p+2) == 0)
    return false;
 return true;
}
```

# Fatoração em primos

```
#include <map>
typedef map<int, int> prime_map;
void squeeze(prime_map& M, int& n, int p) {
  for (; n % p == 0; n /= p) M[p]++;
}
prime_map factor(int n) {
  prime_map M;
  if (n < 0) return factor(-n);
  if (n < 2) return M;
  squeeze(M, n, 2);
  squeeze(M, n, 3);
  int maxP = sqrt(n) + 2;
  for (int p = 5; p < maxP; p += 6) {
      squeeze(M, n, p);
      squeeze(M, n, p+2);
  }
  if (n > 1) M[n]++;
  return M;
}
```

# Crivo TrialFactorization

```
//Complexity: O(n^2/(log n)^2)
vector <int> SieveTrialFactorization(int n){
  vector <int> primes;
  primes.push_back(2);
  primes.push_back(3);
  for(int p = 5; p <= n; p += 2)
  {
    bool is_prime = true;
    for(int i = 1; i < primes.size(); i++)
    {
      int k = primes[i];
      if(k*k > p) break;
      if( p%k == 0)
      {
        is_prime = false;
        break;
      }
    }
    if( is_prime)
    {
      primes.push_back(p);
    }
  }
  return primes;
}
```

# Crivo de Erathostenes

```
//Complexity: O(n log log n)
vector<int> SieveErathostenes(int n){
  vector <bool> S;
  vector <int> primes;

  for(int p = 0; p <= n; p++) S.push_back(true);

  S[1] = false;
  for(int p = 2; p*p <= n; p++){
    if(S[p]){
      for(int q = p*p; q <= n; q += p)
      {
        S[q] = false;
      }
    }
  }
  for(int p = 2; p <= n; p++){
    if(S[p]) primes.push_back(p);
  }
  return primes;
}
```

# Crivo Linear de Pritchard

```
//Complexity: O(n)
vector <int> LinearSievePritchard(int n){
  vector <bool> S;
  for(int p = 0; p <= n; p++) S.push_back(true);
  S[1] = false;
  vector <int> primes = SieveErathostenes(ceil(sqrt(n)));

  for(int f = 2; f <= n/2; f=f+1){
    for(int i = 0; i < primes.size(); i++)
    {
      int p = primes[i];
      if(p > n/f) break;
      S[p*f] = false;
      if(f%p==0) break;
    }
  }
  for(int k = primes[primes.size()-1] + 2; k <= n; k += 2)
  {
    if(S[k]) primes.push_back(k);
  }
  return primes;
}
```

N = 100000000

| SieveErathostenes | SieveTrialFactorization | Linear Sieve Pritchard |
|---|---|---|
| 18.49700 | 81.54500 | 12.46000 |

# Máximo Divisor Comum

- $gcd(a, b)$ : maior inteiro que divide $a$ e $b$.
- $gcd(a, b) = gcd(a, b - a)$
- $gcd(a, b) = gcd(b, a\%b)$
- $gcd(a, 0) = a$
- $gcd(a, b)$ é o menor inteiro positivo em $\{ax + by | x, y \in \mathbb{Z}\}$

# Algoritmo de Euclides

- Example:

$$
\begin{aligned}
gcd(15,12) &= gcd(12,15\%12)\\
&= gcd(12, 3)\\
&= gcd(3, 12\ \%3)\\
&= gcd(3, 0)\\
&= 3
\end{aligned}
$$

# Implementação

```
int gcd(int a, int b){
    while(b){
        int r = a%b;
        a = b;
        b = r;
    }
    return a;
}
```

- Complexidade de tempo: O( log (a+b) )

# Algoritmo de Euclides Estendido

- $gcd(a, b) = ax + by$ para algum inteiro $x$ e $y$
- Os inteiros $x$ e $y$ pode ser encontrados usando o algoritmo de euclides estendido.
- Example: $a = 15, b = 12$

| r | q | x | y |
|---|---|---|---|
| 7 | * | 1 | 0 |
| 5 | * | 0 | 1 |
| 2 | 1 | 1 | -1 |
| 1 | 2 | -2 | 3 |

- gcd(7,5) = 7(-2)+5(3) = 1

# Algoritmo de Euclides Estendido

```
int gcd(int a, int b, int & x, int & y)
{
  int q, t;
  int x1, y1;
  x = 1, y = 0;
  x1 = 0, y1 = 1;
  while(b)
  {
    q = a/b;
    t = b;
    b = a - q*b;
    a = t;
    t = x1;
    x1 = x - q*x1;
    x = t;
    t = y1;
    y1 = y - q*y1;
    y = t;
  }
  return a;
}
```

# Inverso Multiplicativo Modular

- $a^{-1}$ é o inverso multiplicativo de a módulo n se somente se $aa^{-1} = 1(mod\ \ n)$
- Se $gcd(a, n) = 1$ então $ax + ny = 1$ para algum $x, y \in \mathbb{Z}$
- Tomando módulo $n$, temos $ax = 1(mod\ \ n)$
- $x$ é o inverso de $a(mod\ \ n)$.
- Example: Qual é o inverso de $55^{-1}(mod\ 89)$?
  - gcd(55,89) = 1 = 55(34) + 89(-21)
  - 55(34) = 1 (mod 89)
  - 34 é inverso de 55 módulo 89.

# Inverso Multiplicativo Modular

```
/* a pode ser negativo e n > 0*/
int mod(int a, int n){
  printf("mod(%d,%d) = %d\n", a, n, (a%n) + a < 0 ? n : 0);
  return (a%n + n)%n;
}

int invmod(int a, int n){
  int d,x,y;
  d = gcd(a, n, x, y);
  if(d==1)
    return mod(x, n);
}
```

# Congruência Linear

```
/*
Achar a menor solução negativa de ax = b (mod m)
Caso contrário, retorna -1
*/
int solve_mod(int a, int b, int m){
if(m < 0) return solve_mod(a,b,-m);
if(a<0||a>=m||b<0||b>=m)
solve_mod( mod(a,m), mod(b,m), m);
d = gcd(a,m,x,y);
if(b %d != 0) return -1;
else return mod( x*(b/d), m );
}
```

# Computando Coeficiente Binomial

- Calcule utilizando a fórmula:

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k!}$$

```
typedef long long int lli;
lli binom2(int n, int k){
    if(k > n) return 0;
    if(n == k) return 1;
    if(k > n-k) k = n-k;
    lli c = 1;
    for(int i = 1; i <= k; i++){
        c *= n--;
        c /= i;
    }
    return c;
}
```

# Computando Coeficiente Binomial

- Use o triângulo de Pascal

$$\binom{n}{k} = \begin{cases} 1 & , k = 0 \\ 1 & , k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & , \text{caso contrário} \end{cases}$$

```
#define MAX 100
typedef long long int lli;
lli C[MAX+1][MAX+1];
void prebinom(){
  C[0][0] = 1LL;
  for(int i = 1; i<= MAX; i++){
    for(int j = 0; j <= i; j++){
      if(j==0) C[i][j] = 1LL;
      else if(j == i) C[i][j] =1LL;
      else C[i][j] = C[i-1][j-1] + C[i-1][j];
    }
  }
}
```

# Sequência de Fibonacci

- Definição recursiva:

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + Fn - 2
\end{aligned}
$$

  Complexidade O(n)

- Fórmula Fechada

$$F_n = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n)$$

  - Não pode ser computado exatamente para valores grandes
  - Complexidade O(1)

# Sequência de Fibonacci

- Matriz de Recorrência

$$\begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \times \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} \quad (1)$$

- Pode ser calculado em $O(lgn)$ usando exponenciação rápida

# Exponenciação rápida de matriz

```
template <class T> class matrix {
  public:
  int m, n;
  vector < vector<T> > a;
  matrix(int m = 0, int n =0);
  vector<T>& operator [](int i) { return a[i]; }
  matrix operator*(matrix M1);
  matrix power(matrix<T> M, int n);
};
```

# Exponenciação rápida de matriz

```
template <class T>
matrix<T>::matrix(int m , int n): m(m), n(n) {
   a.resize(m);
   for (int i = 0; i < m; i++) a[i].resize(n);
}
template <class T>
matrix<T> matrix<T>::operator*(matrix<T> M1){
   matrix R(n,M1.m);
   int i,j,k;
   for(i=0;i<n;i++)
    for(j=0;j<M1.m;j++)
     for(k=0;k<m;k++)
      R.a[i][j]+=a[i][k]*M1.a[k][j];
   return R;
}
```

# Exponenciação rápida de matriz

```
template <class T>
matrix<T> matrix<T>::power(matrix<T> M, int n){
    if(n==0 || n==1)
      return M;
    matrix R = power(M,n/2);
    R = R*R;
    if(n&1){
     R = R*M;
    }
    return R;
 }
```

# Exponenciação rápida de matriz

```
int fibonacci(int n){

 matrix<int> M(2,2);
 M[0][0] = 1;
 M[0][1] = 1;
 M[1][0] = 1;
 M[1][1] = 0;

 if(n==0||n==1) return n;

 M = M.power(M, n-1);
 //[ F_{n}      F_{n-1} ]
 //[ F_{n-1}    F_{n-2} ]
 for(int i=0;i<2;i++){
  for(int j=0;j<2;j++) printf("%d ", M[i][j]);
  printf("\n");
 }

 return M[0][0];
}
```