Universidade Federal do Ceará

Campus de Quixadá

QXD0153 - Desafios de Programação

Lista 2 - Algoritmos Matemáticos

1. O número de divisores positivos de n é denotado por $d(n)$. Seja a fatoração em primos de $n = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \ldots \times p_k^{\alpha_k}$ então o número de divisores é $d(n) = (\alpha_1 + 1) \times (\alpha_2 + 1) \times \ldots \times (\alpha_k + 1)$. Modifique o algoritmo de fatoração em primos para calcular $d(n)$.

```
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;
}

int divisores(int n){
        prime_map aux;
```

```
        aux = factor(n);
        int d=1;
        for (auto it=aux.begin(); it != aux.end(); it++ ){
                d *= (it->second + 1);
        }
        return d;
}
```

https://oeis.org/A000005

2. O número de fatores primos diferentes de n é denotado por $numDiff(n)$. Modifique o algoritmo de fatoração em primos para calcular $numDiff(n)$.

```
int numDiff(int n){
  prime_map aux;
  aux = factor(n);
  return aux.size();
}
```

3. A soma de divisores positivos de n é denotado por $\sigma(n)$. Desenvolva um algoritmo para calcular $\sigma(n)$.

Seja uma fatoração em primos de n, $n = \Pi_{i=1}^{k} p_i^{m_i}$, então a soma dos divisores de n é:

$$\sigma(n) = \Pi_{i=1}^{k} \frac{p_i^{m_i+1} - 1}{p_i - 1} \tag{1}$$

Solução João Vitor:

```
int sigma(int n){
  prime_map M = factor(n);
  int ac = 1;
  for (prime_map::iterator it = M.begin(); it!=M.end(); it++){
    ac*= (pow(it->first,it->second+1)-1)/((it->first)-1);
  }
  return ac;
}
```

4. O número de inteiros positivos menores que n que são primos entre si com n é denotado $\phi(n)$. Seja a fatoração em primos de $n = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \ldots \times p_k^{\alpha_k}$ então o número de divisores é

$$\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2}) \ldots (1 - \frac{1}{p_k}) \tag{2}$$

Modifique o algoritmo de fatoração para calcular $\phi(n)$.

```
int phi(int n){
  int ac = n;
  prime_map M = factor(n);

  for (prime_map::iterator it = M.begin(); it!=M.end(); it++){
        ac /= it->first;
        ac *= (it->first)-1;
  }

  return ac;
}
```

5. O número de fatores primos diferentes de n pode ser determinado para intervalo de inteiros $[0..MAXN]$ modificando o algoritmo de Crivo de Eratóstenes da seguinte maneira:

```
vector<int> sieve_numdiff(int MAXN){
 vector<int> numDiff;
 numDiff.resize(MAXN+1,0);
 for(int i=2;i<=MAXN;i++){
  if(numDiff[i]==0){
   for(int j=i;j<=MAXN;j+=i)
    numDiff[j]++;
  }
 }
 return numDiff;
}
```

Modifique o algoritmo do Crivo de Eratóstenes para calcular $\phi(n)$ para todos os números no intervalo $[0 \dots MAXN]$

Solução Pedro Olímpio:

```
vector<int> phiIntervalo(int maxn){
        vector<int> numDiff;
        for (int i = 0; i <= maxn; i++){
                numDiff.push_back(i);
        }
        for (int i = 2; i <= maxn; i++){
                if (numDiff[i] == i){
                        for (int j = i; j <= maxn; j += i){
                                numDiff[j] /= i;
                                numDiff[j] *= i - 1;
                        }
                }
        }
        return numDiff;
}
```

6. O soma dos divisores de um número n é denotado por $sumDiv(n)$. Modifique o algoritmo do Crivo de Eratóstenes para calcular $sumDiv(n)$ para todos os números no intervalor $[0 \dots MAXN]$

```
vector<int> sumDivIntervalo(int maxn){
        vector<int> numDiff;
        numDiff.resize(maxn + 1);
        for (int i = 1; i <= maxn; i++){
                for (int j = i; j <= maxn; j += i){
                        numDiff[j] += i;
                }
        }
        return numDiff;
}
```

7. O método de Horner consiste em reescrever um polinômio de forma a obter o valor de $p(x)$ tal

que $p(x) = a_0 + a_1 x + a_2 * x^2 + \ldots + a_n x^n$ em que $a_0, a_1, \ldots, a_n$ são os coeficientes do polinômio. Observe que que $p(x)$ pode ser reescrito da seguinte forma:

$$p(x) = a_0 + x(a_1 + x(a_2 * + \ldots + x(a_{n-1} + x a_n))) \tag{3}$$

(a) Implemente o método de Horner para a avaliação de um polinômio de grau n.

```
int horner(int n, int vet[], int x){
        int acumulador = vet[n];
        for (int i = n - 1; i >= 0; i--){
                acumulador *= x;
                acumulador += vet[i];
        }
        return acumulador;
}
```

(b) Utilize o método de Horner para descobrir o resto da divisão de número N de até 100 dígitos por um inteiro M.

```
int char2int(char c){
        return c - '1' + 1;
}


int resto(char *n, int m){
        int size = strlen(n);
        int cont = char2int(n[0]) % m;
        for (int i = 1; i < size; i++){
                cont *= 10;
                cont += char2int(n[i]);
                cont = cont % m;
        }
        return cont;
}
```

(c) Implemente um método que recebe um vetor binário representando um número na base 2 e devolve sua representação na base 10.

```c
int horner_bin(int n, int vet[]){
        int acumulador = vet[0];
        for (int i = 1; i < n; i++){
                acumulador *= 2;
                acumulador += vet[i];
        }
        return acumulador;
}
```