Tópicos Avançados em Algoritmos

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

25 de março de 2019

Introdução à Teoria dos Números

Conjuntos

- Vamos lembrar os conjuntos numéricos:
  - Naturais: $\mathcal{N}$: $\{1, 2, 3, 4, ...\}$
  - Inteiros: $\mathcal{Z}$: $\{..., -4, -3, -2, -1, 0, 1, 2, 3, 4,...\}$
  - Racionais: $\mathcal{Q}$: $\{..., -1,..., -\frac{2}{3},..., -\frac{1}{2},..., -\frac{1}{5},..., 0,..., \frac{1}{7}, ..., \frac{1}{3},..., \frac{2}{4},..., \frac{5}{7},..., 1, ...\}$
  - Reais: $\mathcal{R}$: $\{..., -2,..., -\sqrt{2}, ..., -1,..., -\frac{1}{5},..., 0,..., 1,..., e, ..., \pi, ...\}$
  - Imaginários: $\mathcal{I}$, o conjunto dos números imaginários são representados pelo conjunto de números reais, onde a unidade representativa é: $i = \sqrt{-1}$: $\{..., -2i, ...-i\sqrt{2}, ..., 0, ..., i, ...\}$
  - Complexos: $\mathcal{C}$: $\mathcal{R} \otimes \mathcal{I}$.

Conjuntos

- Os conjuntos $\mathcal{N}$, $\mathcal{Z}$ e $\mathcal{Q}$ são ditos contáveis, ou enumeráveis, é possível associar um número natural para cada elemento de cada um dos conjuntos.O conjunto $\mathcal{R}$ não é contável.
- O tipo `unsigned int` pode representar os números naturais (incluindo 0 no conjunto).
- O tipo `int` pode representar os números inteiros.
- Existe o tipo `ratio` mas não serve para operar com os números racionais.
- O tipo `double` pode representar os números reais e imaginários.
- O tipo `complex` pode representar os números complexos.
- para os tipos `int` pode-se usar: `long` e `long long` e para o double o `long double`, isto inclui `ratio` em seus argumentos.
- O tipo ratio armazena em `num` e `den` valores primos entre si, mas ele define um tipo constante e não uma variável.

- Vamos tratar dos números inteiros, principalmente.
- Divisibilidade é fundamental na teoria dos números, para dois inteiros d e a, a notação $d|a$ representa d divide a, ou seja, existe um k inteiro tal que $a = kd$.
  - Todo inteiro divide 0.
  - Se $d|a$ e $d \geqslant 0$, então d é um *divisor* de a. Note que $d|a \leftrightarrow -d|a$.
  - Todo inteiro a é divisível pelos *divisores triviais* 1 e a. Os divisores não triviais de a são também chamados de *fatores* de a.
  - Um inteiro a que somente possui os divisores triviais é chamado de *número primo*
  - Um inteiro $a > 1$ que não é primo é chamado de *número múltiplo*
  - O número 1 não é nem primo nem múltiplo, é chamado de *unidade*.

Números primos

- A busca por números primos é um problema NP-difícil, pois é necessário testar todos os valores menores que n para saber se é primo, logo ele é $O(b^d)$ onde b é a base do sistema de numeração e d o expoente maior da representação numérica de n na base b.
- Problemas deste tipo são ditos *pseudo-polinomiais*.
- Uma abreviação na busca se dá ao reduzirmos a busca para no máximo $\sqrt{n}$, pois se $x > \sqrt{n}$ é divisor de n, então $y = n/x < \sqrt{n}$ também é divisor e já o consultamos.
- Podemos também jogar fora metade dos casos de teste, eliminando a busca em números pares.

URI 1221 - Primo Rápido

Mariazinha sabe que um Número Primo é aquele que pode ser dividido somente por 1 (um) e por ele mesmo. Por exemplo, o número 7 é primo, pois pode ser dividido apenas pelo número 1 e pelo número 7 sem que haja resto. Então ela pediu para você fazer um programa que aceite diversos valores e diga se cada um destes valores é primo ou não. Acontece que a paciência não é uma das virtudes de Mariazinha, portanto ela quer que a execução de todos os casos de teste que ela selecionar (instâncias) aconteçam no tempo máximo de um segundo, pois ela odeia esperar.

Entrada:

A primeira linha da entrada contém um inteiro **N** ($1 \leqslant \mathbf{N} \leqslant 200$), correspondente ao número de casos de teste. Seguem **N** linhas, cada uma contendo um valor inteiro **X** ($1 < X < 231$) que pode ser ou não, um número primo.

Saída:

Para cada caso de teste imprima a mensagem “Prime” (Primo) ou “Not Prime” (Não Primo), de acordo com o exemplo abaixo.

Solução

```
#include <iostream>
using namespace std;

int main() {
 unsigned int n, x, i;
 bool prime;
 cin >> n;
 while(n--) {
  cin >> x;
  prime=true;
  if(x==1) cout << "Not Prime" << endl;
  else if(x==2 || x==3 || x==5 || x==7) cout << "Prime" << endl;
  else if(!(x%2)) cout << "Not Prime" << endl;
  else {
   for(i=3; i*i <=x && prime; i+=2)
    if(x%i==0) prime=false;
   if(prime) cout << "Prime" << endl;
   else cout << "Not Prime" << endl;
  }
 }
 return 0;
}
```

Números primos entre si

- Números primos entre si são números que não possuem divisores em comum, exceto o 1.
- Por exemplo, 9 e 16 são primos entre si, os divisores de 9 são: $\{1, 3, 9\}$ e os divisores de 16 são: $\{1, 2, 4, 8, 16\}$, a interseção dos conjuntos de divisores é $\{1\}$.
- Por outro lado, 12 e 18 não são primos entre si. Os divisores de 12 são: $\{1, 2, 3, 4, 6, 12\}$ e de 18 são $\{1, 2, 3, 6, 9, 18\}$. A interseção dos divisores (divisores comuns) é: $\{1, 2, 3, 6\}$.
- Um valor importante é o Máximo Divisor Comum: MDC. MDC(12,18) $= 6$.
- Dois números x e y são primos entre si quando MDC(x,y) $= 1$.

Algoritmo de Euclides: MDC

- Vamos considerar o mínimo entre dois números a e b, sendo $a \geqslant b$
- Tomamos o resto da divisão de a por b: r.
- Se o resto é 0, então o MDC é b.
- Se o resto é maior que 0, então vamos verificar o resto entre b e r.
- Isto se processa de forma iterativa até obter um resto 0.
- Neste caso, o MDC é o menor número.
- Com certeza, o algoritmo acaba com o menor número 1.
- O MDC de vários números se obtém 2 a 2, sempre usando o MDC da operação anterior com o próximo número

URI 1307 - Tudo o que Você Precisa é Amor

"All you need is love. All you need is love.
All you need is love, love... love is all you need." The Beatles

Foi inventado um novo dispositivo poderoso pela Beautifull Internacional Machines Corporation chamado de "Máquina do amor!". Dada uma string feita de dígitos binários, a máquina do amor responde se isto é feito somente de amor, ou seja, se tudo o que você irá precisar para construir aquela string for somente amor. A definição de amor para a Máquina do amor é outra string de dígitos binários, fornecida por um operador humano. Vamos supor que nós temos uma string L que representa "love"e forneçamos uma string S para a máquina do amor. Diremos então que tudo o que você precisa é amor para construir S se pudermos repetidamente subtrair L de S até que sobre apenas L. A subtração definida aqui é a mesma subtração aritmética binária na base 2. Por definição é fácil de ver que L>S (em binário), então S não é feito de amor. Se S=L então S é obviamente feito de amor.
Por exemplo, suponha S = "11011"e L = "11". Se repetidamente subtrairmos L de S, obteremos: 11011, 11000, 10101, 10010, 1111, 1100, 1001, 110, 11. Portanto, dado este L, tudo o que você necessita é amor para construir S. Devido a algumas limitações da Máquina do Amor, não será possível lidar com strings com zero à esquerda. Por exemplo "0010101", "01110101", "011111"etc. são string Inválidas. Strings que contenham apenas um dígito também são strings inválidas (isto é outra limitação).

URI 1307 - Tudo o que Você Precisa é Amor

Sua tarefa para este problema é: dadas duas strings binárias válidas, S1 e S2, veja se é possível ter uma string L válida tal que ambas, S1 e S2 possam ser feitas apenas de L (i.e. dadas duas strings válidas S1 e S2, indique se existe pelo menos uma string L válida tal que ambas S1 e S2 sejam feitas apenas de L). Por exemplo, para S1=11011 e S2=11000, nós podemos ter L=11 tal que S1 e S2 são feitas ambas somente de L (como pode ser visto no exemplo abaixo).

Entrada:

A primeira linha de entrada contém um valor inteiro positivo **N** (**N**$<$ 10000) que indica o número de casos de teste. Então, 2$*$**N** linhas vem a seguir. Cada par de linhas consiste de um caso de teste. Cada par de linhas contém respectivamente **S1** e **S2** que serão inseridas como entrada para a máquina do amor. Nenhuma string conterá menos do que 2 ou mais do que 30 caracteres. Você pode assumir que as strings de entrada serão válidas e estarão de acordo com as regras acima.

Saída:

Para cada par de strings, seu programa deve imprimir uma das seguintes mensagens:
`Pair #p: All you need is love!`
`Pair #p: Love is not all you need!`
Onde p representa o número do par de entrada (que inicia em 1). Seu programa deve imprimir a primeira mensagem no caso de existir pelo menos uma string L válida tal que ambas strings S1 e S2 possam ser feitas somente de L. Caso contrário, imprima a segunda linha.

Solução

```
#include <iostream>
#include <algorithm>
#include <bitset>
using namespace std;

int mdc(int a, int b) {
  int r;
  int x = max(a,b);
  int y = min(a,b);
  r = x%y;
  while(r) {
    x = y;
    y = r;
    r = x%y;
  }
  return y;
}
```

Solução: Main

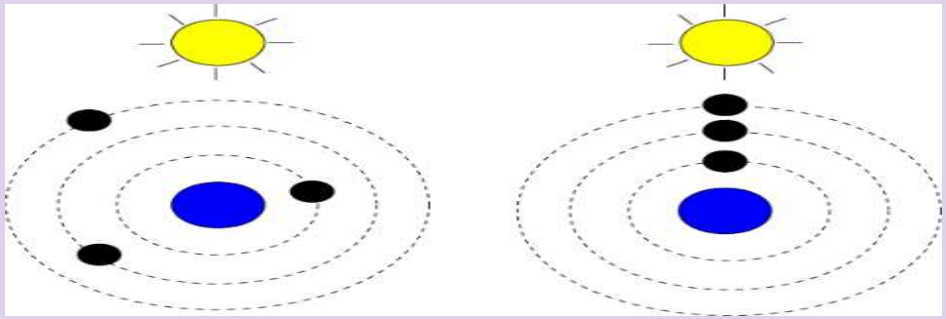
```
int main() {
  int n, a, b;
  string sa, sb;
  cin >> n;
  for(int i=1; i<=n; i++) {
    cout << "Pair #"<< i << ":  ";
    cin >> sa >> sb;
    bitset<30> ba(sa);
    bitset<30> bb(sb);
    a = ba.to_ulong();
    b = bb.to_ulong();
    if(mdc(a,b) > 1) cout << "All you need is love!" << endl;
    else cout << "Love is not all you need!" << endl;
  }
  return 0;
}
```

MMC - Mínimo Múltiplo Comum

- Outro problema comum, é: dados os múltiplos de 2 números diferentes, encontrar o mínimo que seja comum aos dois.
- Um múltiplo comum é dado por um número que seja múltiplo dos divisores de cada número.
- Se os números são primos entre si, então o MMC é o conjunto dos divisores de cada um, ou seja o produto entre eles.
- Se os números não são primos entre si, então o MMC é o conjunto de divisores em comum.
- Em suma: $MMC(a, b) = \frac{a * b}{MDC(a, b)}$
- O MMC de vários números se faz 2 a 2, sempre usando o MMC anterior com o próximo número.

URI 2514 - Alinhamento Lunar

Em uma galáxia muito, muito distante, existe o planeta Nlôguérrà, habitado predominantemente por dinossauros. Nlôguérrà é orbitado por três luas. A órbita de cada lua tem a forma de uma circunferência cujo centro é Nlôguérrà, como indica a figura abaixo, à esquerda.



Sempre que as três luas se alinham e ficam entre o planeta e o sol, como mostra a figura acima, à direita, uma catástrofe terrível acontece! Na última vez que isto ocorreu, há exatamente **M** anos, uma grande seca se instaurou em todo o planeta, reduzindo sua população de dinossauros pela metade. A primeira lua leva $\mathbf{L}_1$ anos terrestres para completar uma volta ao redor do planeta, enquanto a segunda leva $\mathbf{L}_2$ anos e a terceira leva $\mathbf{L}_3$ anos. Determine quantos anos irão se passar até o próximo alinhamento lunar entre o planeta e o sol.
Considere que tanto o planeta quanto o sol são estacionários.

Entrada:

A entrada contém vários casos de teste. A primeira linha de cada caso contém o inteiro **M** ($1 \leqslant \mathbf{M} \leqslant 10^9$), indicando há quantos anos ocorreu o último alinhamento. A segunda linha contém três inteiros $\mathbf{L}_1$, $\mathbf{L}_2$ e $\mathbf{L}_3$ ($1 \leqslant \mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3 \leqslant 10^3$), o tempo levado, em anos, para as luas completarem uma volta. É garantido que não houve alinhamentos como especificado nos últimos **M**-1 anos e que não há alinhamento este ano.
A entrada termina com fim-de-arquivo (EOF).

Saída:

Para cada caso de teste, imprima uma linha contendo um número **X**, indicando que o próximo alinhamento lunar entre o planeta e o sol ocorrerá daqui a **X** anos.

Solução: Main

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
  int l1, l2, l3;
  unsigned long int m;
  while (cin >> m) {
    cin >> l1 >> l2 >> l3;
    cout << (mmc(l1,l2,l3)-m) << endl;
  }
  return 0;
}
```

Função mmc

```
unsigned long int mmc(int l1, int l2, int l3) {
 unsigned long int x, y, r, mdc, m;
 x = max(l1,l2); y = min(l1,l2);
 r = x%y;
 while(r) {
  x = y; y = r;
  r = x%y;
 }
 mdc = y;
 m = (l1*l2)/mdc;
 x = max(m,(unsigned long int) l3);
 y = min(m,(unsigned long int) l3);
 r = x%y;
 while(r) {
  x = y; y = r;
  r = x%y;
 }
 mdc = y;
 m = (m*l3)/mdc;
 return m;
}
```

Números racionais

- Apesar de ser uma representação comum, não existe uma biblioteca ou classe específica para representar números racionais no STL.
- A biblioteca `<ratio>` trabalha somente com template, você somente consegue criar um tipo e não variáveis.
- Uma solução é criar sua própria classe, quando necessário, com as operações mais relevantes, para cada caso.
- Lembrando sempre de, ao armazenar o numerador e o denominador, simplificar os valores para números primos entre si (basta dividi-los pelo MDC entre eles).

Permutação

- Permutação de uma sequência é o número de formas que pode alterar os elementos da sequência criando uma sequência diferente (cada elemento é único).
- Por exemplo, se temos n pessoas em uma fila, a quantidade de formas distintas de organizar a fila é dada pela permutação de n.
  - Dada uma sequência de tamanho n, na primeira posição eu tenho n elementos distintos que posso escolher.
  - Para cada elemento da primeira posição, na segunda posição eu tenho $n-1$ elementos distintos.
  - Para formação das duas primeiras posições, na terceira são $n-2$ elementos.
  - Até a última posição, para cada formação das $n-1$ posições anteriores somente vai sobrar um elemento.
- $P_n = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 = n!$

Arranjo

- Um arranjo de n elementos dipostos p a p representa de quantas maneiras eu consigo formar p dos n elementos, sendo que cada elemento é único.
- Por exemplo em uma competição, com n times quantas formas eu consigo formar o pódio com p competidores é definido pelo arrando de n elementos dispostos p a p.
  - Na primeira posição eu posso colocar n elementos.
  - Para todas formações da primeira posição, na segunda posição eu posso colocar $n-1$ elementos.
  - Até a p-ésima posição, onde para cada formação das $p-1$ posições anteriores, na p-ésima posição eu posso colocar $n-(p-1)$ elementos.
- $A_n^p = n \times (n-1) \times \ldots \times (n-p+2) \times (n-p+1) = \dfrac{n!}{(n-p)!}$

Combinação

- As combinações de n elementos tomados p a p são escolhas não ordenadas destes elementos.
- Por exemplo se tenho n pessoas e quero formar uma comissão de p pessoas, então, o número de formas que posso organizar a comição é a combinação de n, p a p. Observe que p pessoas em uma comissão, são sempre as mesmas p pessoas, não importa a ordem.
- O arranjo de n, p a p, me dá, para os mesmos p elementos, todas as formações possíveis de p elementos ou seja, a permutação de p elementos.
- Assim: $C_n^p = \dfrac{A_n^p}{P_p} = \dfrac{n!}{p!(n-p)!}$
- Em especial, $C_n^p = C_n^{n-p}$.

Permutação com repetição

- No caso de os elementos não serem únicos em uma sequência, a permutação dos elementos na sequência não pode repetir a mesma formação trocando de posição dois elementos iguais.
- Sendo n elementos onde k elementos são distintos, havendo repetições, o primeiro elemento repete r_1 vezes, o segundo r_2 vezes e o k-ésimo r_k vezes.
  - As possibilidades de se colocar o primeiro elemento na sequência é dado por $C_n^{r_1}$,
  - As possibilidades de colocar o segundo elemento na sequência é dada por $C_{n-r_1}^{r_2}$
  - O terceiro elemento: $C_{n-r_1-r_2}^{r_3}$
- $R_n^k = C_n^{r_1} \times C_{n-r_1}^{r_2} \times \ldots \times C_{n-r_1-\ldots-r_{k-1}}^{k} =$

$$C_n^{r_1} \times \prod_{i=2}^{k} \left\{ C_{\left(n-\sum_{j=1}^{i-1} r_j\right)}^{r_i} \right\}$$

Permutações circulares

- Permutações circulares consideram a sequência circular, o primeiro elemento vem antes do segundo, o segundo antes do terceiro, ..., o último antes do primeiro.
- Por exemplo, para um grupo de n pessoas sentadas ao redor de uma mesa redonda, de quantas maneiras distintas podemos arrumar estas pessoas.
- Se considerarmos a permutação de n pessoas, e que o resultado é idêntico se a primeira pessoa ocupasse qualquer uma das n posições, então:
- $Pc_n = \frac{P_n}{n} = (n-1)!$

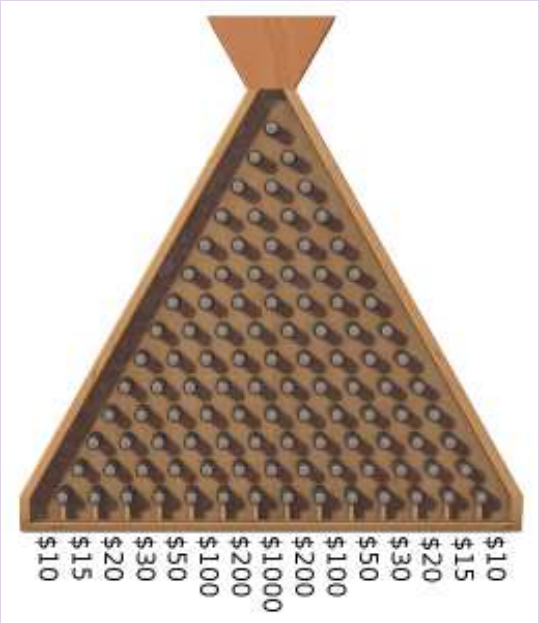
Arranjo com repetição

- Arranjo com repetição de n elementos tomados p a p considera que de n elementos, posso escolher p elementos sendo que eu posso tomar o mesmo elemento 1 ou até p vezes.
- Por exemplo, em um macarrão temperado de livre escolha, dos n temperos diferentes eu tenho direito de escolher p, se eu gostar do k-ésimo tempero, eu posso pedir p porçoes do elemento k.
  - Na primeira posição, eu posso escolher n elementos.
  - Para cada escolha da primeira posição, na segunda posição eu posso escolher n elementos...
  - Na p-ésima posição, para cada escolha das $p-1$ posições anteriores, eu posso escolher n elementos.
- $Ar_n^p = \underbrace{n \times n \times \ldots \times n}_{p} = n^p$

URI 1946 - Pirâmide da Sorte

Um grande show de TV distribui prêmios à platéia através da Pirâmide da Sorte. Um convidado joga uma bolinha no topo da pirâmide (que é um triângulo, na verdade) e ela vai descendo para a esquerda ou para a direita aleatoriamente até chegar em uma das caixinhas na base. O convidado ganha o prêmio que está associado àquela caixinha.

O grande prêmio sempre fica no meio da base da pirâmide, que sempre tem, portanto, um número ímpar de caixinhas na base. Veja uma pirâmide com 15 caixinhas na figura.



URI 1946 - Pirâmide da Sorte

Os produtores do programa querem economizar o máximo possível e pediram para você calcular qual a probabilidade de alguém ganhar o grande prêmio, dado o número de caixinhas na base da pirâmide. Considere que, em cada ponto da pirâmide, existe a mesma chance da bolinha ir para a esquerda ou para a direita.

Entrada:

A entrada é dada em uma única linha, que contem o número S de caixinhas na base da pirâmide ($3 \leqslant \mathbf{S} \leqslant 4999$). **S** é sempre ímpar.

Saída:

A saída deve ser dada em uma única linha, que contem a probabilidade da bolinha cair na caixinha com o grande prêmio. A probabilidade deve ser exibida com 2 casas decimais.

Buscando a solução

- A primeira coisa a fazer é entender a regra de formação.

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}
$$

$$
\begin{array}{ccccccccccc}
1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1
\end{array}
$$

- Com um pouco de imaginação, e brincando com números:

$$C_0^0$$

$$C_1^0 \quad C_1^1$$

$$C_2^0 \quad C_2^1 \quad C_2^2$$

$$C_3^0 \quad C_3^1 \quad C_3^2 \quad C_3^3$$

$$C_4^0 \quad C_4^1 \quad C_4^2 \quad C_4^3 \quad C_4^4$$

$$C_5^0 \quad C_5^1 \quad C_5^2 \quad C_5^3 \quad C_5^4 \quad C_5^5$$

$$C_6^0 \quad C_6^1 \quad C_6^2 \quad C_6^3 \quad C_6^4 \quad C_6^5 \quad C_6^6$$

Calculando a probabilidade

- Para $n = 3$, $p(3) = \dfrac{C_2^1}{C_2^0 + C_2^1 + C_2^2}$
- Para $n = 7$, $p(3) = \dfrac{C_6^3}{C_6^0 + C_6^1 + C_6^2 + C_6^3 + C_6^4 + C_6^5 + C_6^6}$
- Para facilitar vamos tomar $p^{-1}(n)$ para n ímpar, e $m = (n-1)/2$
- $p^{-1}(n) = \dfrac{C_{n-1}^0 + C_{n-1}^1 + C_{n-1}^2 + \ldots + C_{n-1}^m + \ldots + C_{n-1}^{n-3} + C_{n-1}^{n-2} + C_{n-1}^{n-1}}{C_{n-1}^m}$
- Sabemos que $C_{n-1}^0 = C_{n-1}^{n-1}$, $C_{n-1}^1 = C_{n-1}^{n-2}$, ..., logo:
- $p^{-1}(n) = 2\dfrac{C_{n-1}^0}{C_{n-1}^m} + 2\dfrac{C_{n-1}^1}{C_{n-1}^m} + 2\dfrac{C_{n-1}^2}{C_{n-1}^m} + \ldots + 1$

Calculando a probabilidade

- continuando...
- $p^{-1}(n) = 2\frac{C_{n-1}^0}{C_{n-1}^m} + 2\frac{C_{n-1}^1}{C_{n-1}^m} + 2\frac{C_{n-1}^2}{C_{n-1}^m} + \ldots + 1$
- $p^{-1}(n) = 2\frac{\frac{(n-1)!}{0!(n-1)!}}{\frac{(n-1)!}{m!m!}} + 2\frac{\frac{(n-1)!}{1!(n-2)!}}{\frac{(n-1)!}{m!m!}} + 2\frac{\frac{(n-1)!}{2!(n-3)!}}{\frac{(n-1)!}{m!m!}} + \ldots + 1$
- Simplificando:
- $p^{-1}(n) = 2\frac{m!m!}{0!(n-1)!} + 2\frac{m!m!}{1!(n-2)!} + 2\frac{m!m!}{2!(n-3)!} + \ldots + 1$
- $p^{-1}(n) = 2\frac{m!}{(n-1)(n-2)..(m+1)} + 2\frac{m!}{(n-2)(n-3)..(m+1).1} + 2\frac{m!}{(n-3)..(m+1).2.1} + \ldots + 1$
- Como no numerador e no denominador existe a mesma quantidade de termos, ao invés de fazer todo o produto de cada um, vamos fazer produtos de frações, pois queremos um número de ponto flutuante.

- Aqui, a rotina (parcel) recebe três parâmetros: n (que já é $n-1$), k (que representa o primeiro termo: $n-k$, ou seja, vai de 0 a $m-1$), e m.

```
double parcel(int n, int m, int k) {
 double prod=1;
 for(int i=0; i<m; i++) {
  prod=prod*(m-i)/(n-(k+i) <= m ?  m-i :  n-(k+i));
 }
 return prod;
}
```

- Calculando p

```
 double total = 1;
 for(int i=0; i<m; i++) {
  total+=parcel(n-1,m,i)*2;
 }
 cout << fixed << setprecision(2) << (100.0/total) << endl;
```