maratona de programação

UNIFEI

2 – **Estruturas de Dados Parte I**

Prof. **João Paulo** R. R. Leite
joaopaulo@**unifei**.edu.br
ECOE44 & ECOE45 – Maratona de Programação
sites.google.com/site/unifeimaratona/

Tipos Básicos de Dados

É necessário conhecer bem cada tipo primitivo de dado, para que tenhamos uma noção real de capacidade de armazenamento de cada um deles e saibamos utilizá-los da melhor maneira em nossos programas. São eles:

bool: é um valor lógico, ou booleano (true/false).
char: inteiro sinalizado de 8 bits, normalmente utilizado para representar caracteres com ASCII.
short: inteiro de 16 bits.
Int: inteiro de 32 bits.
long long: inteiro de 64 bits.
float: número com ponto flutuante de 32 bits.
double: número com ponto flutuante de 64 bits.
long double: número com ponto flutuante de 128 bits.
string: uma sequência de caracteres.

Veja suas capacidades:

Type	Bytes	Min value	Max value
bool	1		
char	1	-128	127
short	2	-32768	32767
int	4	-2148364748	2147483647
long long	8	-92233720368547758O8	9223372036854775807
	n	-2^{8n-1}	$2^{8n-1}-1$

Type	Bytes	Min value	Max value
unsigned char	1	0	255
unsigned short	2	0	65535
unsigned int	4	0	4294967295
unsigned long long	8	0	18446744073709551615
	n	0	$2^{8n}-1$

Type	Bytes	Min value	Max value	Precision
float	4	$\approx -3.4 \times 10^{-38}$	$\approx 3.4 \times 10^{+38}$	≈ 7 digits
double	8	$\approx -1.7 \times 10^{-308}$	$\approx 1.7 \times 10^{+308}$	≈ 14 digits

Fonte: Guðmundsson & Magnússon
Reykjavík University

Big Integers

O que fazer quando mesmo um **long long** não for suficiente para comportar uma entrada muito grande? Além da alternativa apresentada na semana passada, da classe **BigInteger** de Java, temos outra alternativa:

Ler o número e guardá-lo como uma string!

E como realizamos operações aritméticas?
Utilizando os mesmos algoritmos que aprendemos no ensino fundamental... Dígito por dígito.

Exercício A: Grandes Cálculos

Há um novíssimo supercomputador na UNIFEI e há um aluno realizando estudos de somas de potências, com números que chegam a ter dezenas de casas decimais. Precisamos conferir se seus resultados estão corretos!

Entrada

A entrada consiste um único caso de teste. O caso de teste possui N linhas (N ≤ 100), contendo um número muito grande X cada. Todos os números são inteiros e não-negativos. O final da entrada é marcado por X = 0.

Saída

Para cada caso de teste, imprima na saída uma linha contendo a soma dos N números que compõem aquele caso.

Exemplo de Entrada	Exemplo de Saída
12345678901234567890123456789012345678901234567890123456789012345678901234567890 0	37037036703703703670370370367037037036703703703670

Estruturas de Dados

Estruturas que já coinhecemos:

1) Arrays estáticos: int vet[1000];
2) Arrays dinâmicos: vector<int>
3) Listas encadeadas: list<int>
4) Pilhas: stack<int>
5) Filas: queue <int>
6) Filas de prioridade: priorirty_queue<int>
7) Sets: set<int>
8) Maps: map<int, int>

Caso ainda não conheça bem algum desses ou tenha com pouca fluência, estude fortemente STL, começando pela apresentação **02-extra-stl-bibliotecas.pdf** disponível em nosso site sites.google.com/site/unifeimaratona/aulas.

Estruturas de Dados

Normalmente, é melhor que utilizemos as implementações padrão da linguagem, disponíveis através de suas bibliotecas.

- Estão livre de bugs e são implementações otimizadas
- Economizamos o tempo de escrita e depuração de códigos que são, muitas vezes, bem complexos

Em alguns casos específicos, será necessário escrever nossa própria implementação.

- Problema pede maior flexibilidade
- Quando precisarmos modificar alguma regra básica, personalizar a estrutura

Estruturas de Dados

As duas principais operações que podem ser feitas com estruturas de dados são:

- Ordenação de um vetor: sort(vet.begin(), vet.end())
- Busca em array desordenado: find(vet.begin(), vet.end(), x)
- Busca em array ordenado: lower_bound(vet.begin(), vet.end(), x)
  binary_search(vet.begin(), vet.end(), x)

Também disponíveis na **biblioteca padrão**.

Ordenação

Definição:

Dada uma coleção de objetos fora de ordem, ordene-os.

Alguns algoritmos são bastante populares

$O(n^2)$: Bubble, Selection, Insertion sort.

$O(n \log n)$: Merge, Quick, Heap sort.

Na competição, podemos esquecê-los, e utilizar o algoritmo pronto na biblioteca padrão, que executará em $O(n \log n)$.

Ordenação será sempre um passo preliminar para um algoritmo mais complexo ou um passo final para organização da saída.

Nunca será um fim em si mesma na competição.

Na STL,

Temos três algoritmos prontos (biblioteca **<algorithm>**):

- **sort:** O algoritmo específico não é fixo e pode variar dependendo da implementação. No entanto, a complexidade no pior caso é, obrigatoriamente, O(n logn).
  - Bastante **rápido;**
  - Ordena tanto dados **básicos** quanto tipos **definidos pelo usuário**.

- **partial_sort**: Implementa a heap sort e pode ser utilizado para ordenar apenas uma parte da estrutura. Se for necessário ordenar k itens, sua complexidade no tempo será de O(k logn).

- **stable_sort**: Preserva ordem e elementos com o mesmo valor, se necessário.

Busca de Dados

Definição:
Dado um conjunto de dados, encontre um valor.

Existem duas variantes:

- Quando os dados estão **fora de ordem**, é necessário fazer a busca linear que é trivial e roda em O(n).
- Quando os dados estão **ordenados**, utilize a busca binária, que executa em O(log n).
  - Pode ser complicada ou, no mínimo, perigosa de se implementar.
  - Utilize **lower_bound** da C++ STL, que retorna um iterador ou **binary_search** que retorna true/false <algorithm>.

Estruturas Lineares

1. **Lista Encadeada** (C++ STL list)
   Normalmente não será utilizada a list, simplesmente utilize vector, quando tiver que armazenar uma lista de dados.

2. **Pilhas** (C++ STL stack)
   Processa eventos na ordem "ultimo a entrar, primeiro a sair". Exemplos: Simulação de recursão, busca em profundidade em grafo, inversão de sequência, verificação de parênteses, etc.

Estruturas Lineares

3. **Filas** (C++ STL queue)

   Processa eventos na ordem "primeiro a entrar, primeiro a sair". Exemplos: Busca em largura no grafo, ordenação topológica, etc.

4. **Fila duplamente Encadeada** (C++ STL deque)

   Trata-se de uma fila com duas extremidades. Possui métodos para inserção e remoção eficiente no início e fim. Similar a vector e utilizada em algoritmos de "janela deslizante", etc.

Exercício B: I Can Guess

There is a bag-like data structure, supporting two operations:
1 x : Throw an element x into the bag.
2 : Take out an element from the bag.

Given a sequence of operations with return values, you're going to guess the data structure. It is a stack (Last-In, First-Out), a queue (First-In, First-Out), a priority-queue (Always take out larger elements first) or something else that you can hardly imagine!

Exercício B: I Can Guess

Input

There are several test cases. Each test case begins with a line containing a single integer n ($1 \leq n \leq 1000$). Each of the n next lines is either a type-1 command, or an integer 2 followed by an integer x. This means that executing the type-2 command returned the element x. The value of x is always a positive integer not larger than 100. The input is terminated by end-of-file (EOF).

Exercício B: I Can Guess

Output

For each test case, output one of the following:

`stack`

It's definitely a stack.

`queue`

It's definitely a queue.

`priority queue`

It's definitely a priority queue.

`impossible`

It can't be a stack, a queue or a priority queue.

`not sure`

It can be more than one of the three data structures mentioned above.

Exemplo de Entrada	Exemplo de Saída
6 1 1 1 2 1 3 2 1 2 2 2 3 6 1 1 1 2 1 3 2 3 2 2 2 1 2 1 1 2 2 4 1 2 1 1 2 1 2 2 7 1 2 1 5 1 1 1 3 2 5 1 4 2 4 1 2 1	queue not sure impossible stack priority queue impossible

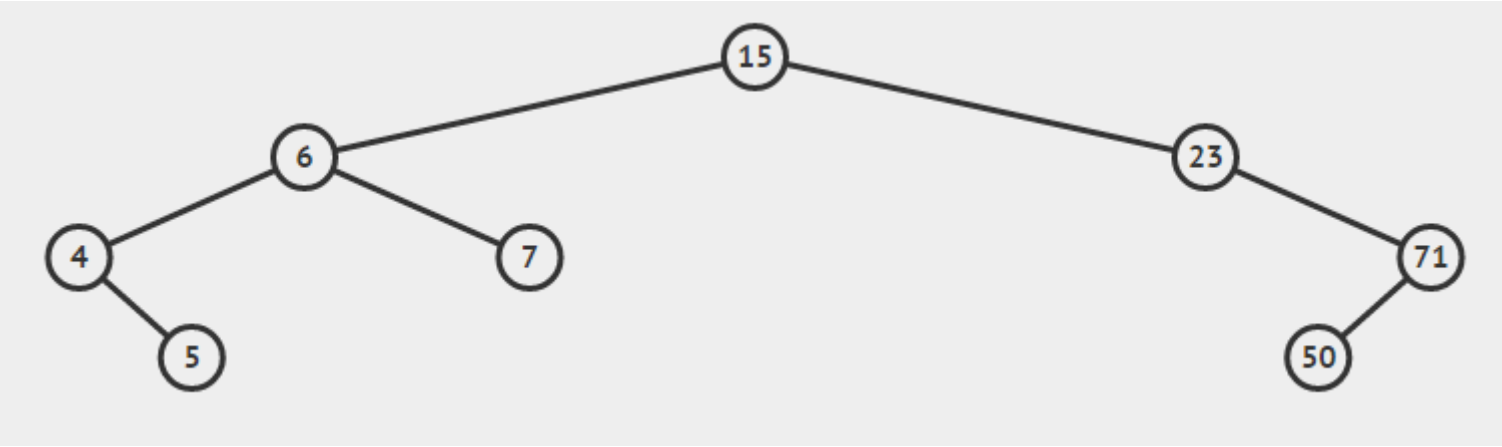
Estruturas Não-Lineares - BST

Árvore Binária

- Maneira de se organizar dados em forma de árvore.
  - Em cada sub-árvore com raiz em x, os itens na sub-árvore à **esquerda** de x são **menores x** e os itens na sub-árvore à **direita** de x são **maiores ou iguais a x**.
- Este tipo de organização permite inserção, busca e remoção em O(log n), mas somente funciona se a árvore for balanceada (AVL, RB-Tree).
  - C++ STL <**map**> e <**set**> são implementações de um tipo de árvore binária balanceada chamada **Red-Black Tree**. Nelas, todas as operações são realizadas em O(log n).
- Qual a diferença?
  - <**map**> armazena pares (key, data)
  - <**set**> armazena apenas (key)

Estruturas Não-Lineares

- Algumas propriedades interessantes são:
  - Não são guardados valores com chaves repetidas e, portanto, são estruturas que nos tornam capazes de remover redundância de dados.
  - Ao remover os dados da estrutura, eles são retirados "em ordem". Caso seja necessário ler uma sequência de dados e imprimi-los em ordem, set/map é uma boa pedida.



Exercício C: Word Index

Encoding schemes are often used in situations requiring encryption or information storage/transmission economy. Here, we develop a simple encoding scheme that encodes particular types of words with five or fewer (lower case) letters as integers.

Consider the English alphabet {a, b, c, ..., z}. Using this alphabet, a set of valid words are to be formed that are in a strict lexicographic order. In this set of valid words, the successive letters of a word are in a strictly ascending order; that is, later letters in a valid word are always after previous letters with respect to their positions in the alphabet list {a, b, c, ..., z}. For example, “abc aep gwz” are all valid three-letter words, Whereas “aab are cat” are not.

For each valid word, associate an integer which gives the position of the word in the alphabetized list of words. That is: a -> 1 b -> 2 ... z -> 26 ab -> 27 ac -> 28 ... az -> 51 bc -> 52 ... vwxyz -> 83681

Your program is to read a series of input lines. For each word read, if the word is invalid give the number 0. If the word read is valid, give the word's position index in the above alphabetical list.

Exercício C: Word Index

Input

The input consists of a series of single words, one per line. The words are at least one letter long and no more that five letters. Only the lower case alphabetic {a, b, ...,z} characters will be used as input. The first letter of a word will appear as the first character on an input line. The input will be terminated by end-of-file.

Output

The output is a single integer, greater than or equal to zero (0) and less than or equal 83681. The first digit of an output value should be the first character on a line. There is one line of output for each input line.

Input Samples	**Correspondent Output**
z a cat vwxyz	26 1 0 83681

Gere todas as palavras, adicione a um *map* para auto ordenação.

Estruturas Não-Lineares - Heap

- Heap é outra maneira de organizar dados em forma de árvore. No entanto, possui propriedades diferentes da BST.
  - Para cada sub-árvore com raiz em x, itens das sub-árvores esquerda e direita são menores do que x. Esta propriedade garante que o topo da heap sempre terá o elemento de valor máximo.
  - A árvore precisa ser completa pelo menos até seu penúltimo nível. No último nível, os itens estão "justificados" à esquerda.
- Normalmente não trabalhamos com busca na heap, mas inserção e remoção podem ser feitas em O(log n).
- É importante no Processamento de eventos em ordem de prioridade. Busca de caminho mínimo em grafo (Dijkstra), alguns algoritmos gulosos (Kruskal), etc.

Estruturas Não-Lineares - Heap

Priority queue (STL)

Filas de prioridade são um tipo de adaptador de container projetado especificamente para que seu **primeiro elemento seja sempre o maior entre todos os elementos**, de acordo com um critério de ordenação.

O contexto é, portanto, similar ao de uma heap, onde elementos podem ser inseridos a qualquer momento, e somente o elemento máximo da heap pode ser obtido (aquele no topo da fila de prioridade, pop_back).

Funções membro:

- **empty**
- **size**
- **top**
- **push** (push_back)
- **pop** (pop_back)

Array de bits

C++ STL nos provê com outro tipo de dado, chamado **bitset**.

Essa classe emula um vetor de elementos do tipo bool, mas é otimizado para alocação de espaço: geralmente, cada elemento ocupa apenas um bit.

Nele, cada posição pode ser acessada individualmente. Por exemplo, para um bitset chamado foo, foo[3] acessa seu quarto bit, da mesma maneira que um array normal acessaria seus dados.

Possui funções set (1), reset (0) e test (?).
Veja a documentação em
http://www.cplusplus.com/reference/bitset/bitset/

Bitmasks

Podemos representar conjuntos também de uma maneira alternativa muito interessante e eficaz.

- Se tivermos um número pequeno de itens ($n \leq 30$), podemos rotular cada item com um inteiro no intervalo 0, 1, ..., n-1.
- Conseguimos representar conjuntos destes itens através de um inteiro de 32 bits, onde a presença ou ausência do item i de nossa coleção pode ser representada através do i-ésimo bit do inteiro.
- Exemplo: Imagine o conjunto {0, 3, 4}
  - int x = (1<<0) | (1 << 3) | (1 << 4);
  - 00000000000000000000000000011001

Por exemplo, se deslocar uma variável `x` a esquerda:

```
int x = 1; // 0000 0001

int x0 = (x << 0); // 0000 0001 Não deslocado
int x1 = (x << 1); // 0000 0010
int x2 = (x << 2); // 0000 0100
int x3 = (x << 3); // 0000 1000
int x4 = (x << 4); // 0001 0000
int x5 = (x << 5); // 0010 0000
int x6 = (x << 6); // 0100 0000
int x7 = (x << 7); // 1000 0000
```

Agora se deslocar a direita:

```
int x = 128; // 1000 0000

int x0 = (x >> 0); // 1000 0000 Não deslocado
int x1 = (x >> 1); // 0100 0000
int x2 = (x >> 2); // 0010 0000
int x3 = (x >> 3); // 0001 0000
int x4 = (x >> 4); // 0000 1000
int x5 = (x >> 5); // 0000 0100
int x6 = (x >> 6); // 0000 0010
int x7 = (x >> 7); // 0000 0001
```

Fonte:
http://pt.stackoverflow.com/questions/10174/como-funciona-o-deslocamento-de-bits-em-c-c

Bitmasks

Conseguimos realizar operações bastante interessantes:

- Conjunto vazio: `x = 0;`
- Inserir um elemento i: `x |= (1 << i)`
- Remover um elemento i: `x &= ~(1 << i)`
- União de conjuntos: `x | y`
- Interseção de conjuntos: `x & y`
- Checar se um elemento i do conjunto está presente:

```
if(x & (1 << i)) {
        // sim
} else {
        // não
}
```

Bitmasks

Mas porque devemos complicar as coisas?
Porque não utilizamos simplesmente um set<int>?

A representação com máscara de bits é muito leve.
Todas as operações são realizadas através de simples manipulações de bits, o que o deixa **muito mais rápido** que um vector<bool>, bitset ou set<int>.

Eficiência e velocidade são muito importantes na competição, então bitmasks podem ser ótimas alternativas!

Exercício D: Grey Codes

We are going to generate a sequence of integers in binary. Start with the sequence

0
1
—

Reflect it in the horizontal line, prepend a zero to the numbers in the top half and a one to the numbers on the bottom and you will get

00
01
11
10

Repeat this again, and you will have 8 numbers

000 0
001 1
011 3
010 2
110 6
111 7
101 5
100 4

The corresponding decimal values are shown on the right.
These sequences are called Reflected Gray Codes for 1, 2 and 3 bits respectively. A Gray Code for n bits is a sequence of 2^n different n-bit integers with the property that every two neighbouring integers differ in exactly one bit. A Reflected Gray Code is a Gray Code constructed in the way shown above.

Input

The first line of input gives the number of cases, N (at most 250000). N test cases follow. Each one is a line with 2 integers: n ($1 \leq n \leq 30$) and k ($0 \leq k < 2^n$).

Output

For each test case, output the integer that appears in position k of the n-bit Reflected Gray Code.

Exemplo de Entrada	Exemplo de Saída
7 1 0 1 1 2 0 2 1 2 2 2 3 3 0	0 1 0 1 3 2 0