Tópicos Avançados em Algoritmos

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

18 de fevereiro de 2019

Containers

Classes e Containers

- Vamos revisitar o tema Containers. Algumas classes, containers e containers adaptors interessantes são:
  - Classes: `bitset`, `complex`, `initializer_list`, `pair`, `tuple`
  - Classes e Containers de sequência: `array`, `deque`, `forward_list`, `list`, `queue`, `stack`, `valarray`, `vector`
  - Containers relacionados com ordem: `set`, `multiset`, `map`, `multimap`, `unordered_set`, `unordered_multiset`, `unordered_map`, `unordered_multimap`.

Já vistos

- Não vamos retomar os containers apresentados anteriormente.
- Vale a pena, no entanto, um exemplo deles:

Exemplo: URI 1975: Pérolas

VI é uma professora de cálculo muito excêntrica, sempre que corrige as provas dos alunos (Que por sinal são provas dificílimas), ela anota todas as pérolas que encontra enquanto corrige, para que no dia da entrega ela possa escrever todas no quadro, para deixar os alunos envergonhados e que eles nunca mais errem as mesmas coisas.
Sempre que a bronca termina e as provas são entregues, os alunos tentam descobrir quem foi que teve mais pérolas no quadro. Como a cada prova os números de pérolas aumentam e os alunos tem que estudar muito pois a cada semana acontece uma nova prova de cálculo, eles não tem tempo para verificar todas as provas e ver quem apareceu mais vezes no quadro.
Sabendo que você é programador eles pediram sua ajuda para mostrar qual foi o aluno que teve mais pérolas escritas no quadro naquele dia.

Entrada:

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém três inteiros **P**, **A** e **R** ($1 \leqslant \mathbf{P}, \mathbf{A}, \mathbf{R} \leqslant 10^4$), indicando respectivamente, o número de pérolas, número de alunos e a quantidade de respostas dadas por cada aluno. Segue **P** linhas com as pérolas escritas no quadro que terão no máximo 1000 caracteres. Em seguida terão A alunos, para cada aluno a primeira linha será seu nome com no máximo 100 caracteres minúsculos de 'a' até 'z', seguindo as R linhas mostrando suas respostas. A entrada termina quando $\mathbf{P} = \mathbf{A} = \mathbf{R} = 0$, e não deve ser processada.

Saída:

Para cada saída, você deverá imprimir o nome do aluno que teve mais aparições no quadro, em caso de empate seu programa deverá mostrar todos os alunos com mais aparições separados por vírgulas em ordem alfabética.

Exemplo de Entrada:

```
4 3 3
1 + 1 = 3
((x - 2) + (x - 2)) / (x - 2) = 1
4 / 0 = 0
n!  = n^2
gabriel
5 + 5 = 10
1 + 1 = 3
4 / 0 = ERRO
alexandre
((x - 2) + (x - 2)) / (x - 2) != 1
1 + 1 = 2
2 + 3 = 5
anne
1 + 1 = 3
4 / 0 = ERRO
2 + 3 = 5
0 0 0
```

Saída do Exemplo de Entrada:

```
anne, gabriel
```

Solução: parte 1 - função de comparação

```
#include <iostream>
#include <set>
#include <utility>
#include <string>
using namespace std;

class comp {
public:
 bool operator() (const pair<int,string>& a,
                  const pair<int,string>& b) const {
   bool res;
   res = a.first > b.first;
   if(a.first==b.first) {
    res = a.second < b.second;
   }
   return res;
 }
};
```

Solução: parte 2 - Construindo as pérolas

```
int main() {
 int p, a, r, prim;

 cin >> p >> a >> r;
 while(p) { //Cada caso de teste
  set<string> perolas;
  set<pair<int,string>,comp> lousa;
  cin.ignore();
  while(p--) { // Anotando as perolas num set
   string buff;
   getline(cin,buff);
   perolas.insert(buff);
  }
```

Solução: parte 3 - Contando os erros

```
for(int i=0; i<a; i++) {
  int erros = 0;
  string nome;
  cin >> nome;
  cin.ignore();
  for(int j=0; j<r; j++) {
    string buff;
    getline(cin,buff);
    if(perolas.insert(buff).second) {
      perolas.erase(buff);
    } else erros++;
  }
  lousa.insert(make_pair(erros,nome));
}
```

Solução: parte 4 - Gerando a saída

```
    set<pair<int,string> >::iterator it = lousa.begin();
    prim = it->first;
    cout << it->second;
    ++it;
    while(it->first == prim) {
      cout << ', ' << it->second;
      ++it;
    }
    cout << endl;
    cin >> p >> a >> r;
  }
  return 0;
}
```

bitset

- As definições estão na referência <bitset>
- Armazena bits, elementos de valores 0 ou 1 (falso ou verdade)
- A classe emula um vetor de elementos booleanos (ou bits)
- Cada posição de bit pode ser acessada pelo operador `[]`. Para um bitset de nome `foo`, `foo[3]` retorna o bit que ocupa a quarta posição.
- Bitset pode ser contruído de/convertido para valores int ou string binárias. Também pode ter valores individuais inseridos ou removidos por streams.
- Estão definidos os operadores: `&=`, `|=`, `^=`, `<<=`, `>>=`, `<<`, `>>`, `==`, `!=`, `&`, `|`, `^`

bitset

```
template <size_t N> class bitset;
```

Métodos

`operator[]`	Acessa o bit
`count`	Conta os bits
`size`	Retorna o tamanho
`test`	Retorna um bit
`any`	Verifica se algum bit é 1
`none`	Verifica se nenhum bit é 1
`all`	Verifica se todos bits são 1
`set`	Set bits
`reset`	Zera bits
`flip`	Inverte bits
`to_string`	converte em string
`to_ulong`	converte em uns long
`to_ullong`	converte em uns long long

bitset - URI 2507 - Código de Hamming

O código de Hamming permite detecção e correção de erro na leitura de dados. Sejam palavras de 16 bits. Na tabela abaixo está a palavra $4ac5_{\text{hex}}$ em sua forma binária, cada bit possui sua numeração de posição indicada:

Posições	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Palavra	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	1

Na codificação de Hamming, a numeração das posições dos bits utiliza uma contagem que é feita pulando os valores que são potência de 2: 1, 2, 4, ... Estes bits serão utilizados para a paridade para a codificação de Hamming. No caso acima ficaria:

Posições	21	20	19	18	17	15	14	13	12	11	10	9	7	6	5	3	16	8	4	2	1
Palavra	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	1					

Para calcular a paridade de cada bit de hamming usamos os bits de acordo com sua posição. Por exemplo, o bit da posição 11 irá influenciar os bits de hamming: 8, 2 e 1, pois 11 = 8+2+1. Para o bit de paridade 2, vemos qual é o bit de cada posição que recebe influência, neste caso 3 (2+1), 6 (4+2), 7 (4+2+1), 10, 11, 14, 15, 18 e 19. Neste caso, se a soma destes bits for par, a paridade é par e o bit de hamming é 0, caso contrário o bit de hamming é 1. Veja a análise completa:

bitset - URI 2507 - Código de Hamming

Posições	21	20	19	18	17	15	14	13	12	11	10	9	7	6	5	3	16	8	4	2	1
Palavra	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	1					
Bit 1	0		0		1	0		0		1		0	0		0	1					1
Bit 2			0	0		0	1			1	0		0	1		1				0	
Bit 4	0	1				0	1	0	1				0	1	0				0		
Bit 8						0	1	0	1	1	0	0						1			
Bit 16	0	1	0	0	1												0				
Final	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	1	0	1	0	0	1

Uma vez assim codificada, a palavra resultante de 21 bits pode ser apresentada na forma hexadecimal como: 0958a9. A vantagem disto é que se na leitura houver erro de no máximo 1 bit, este erro poderá ser corrigido. Veja por exemplo se ao ler a palavra, tivessemos lido o valor 0978a9. Construindo a tabela dos bits teremos:

Posições	21	20	19	18	17	15	14	13	12	11	10	9	7	6	5	3	16	8	4	2	1
Palavra	0	1	0	0	1	0	1	1	1	1	0	0	0	1	0	1	0	1	0	0	1
Bit 1	0		0		1	0		1		1		0	0		0	1					0
Bit 2			0	0		0	1			1	0		0	1		1				0	
Bit 4	0	1				0	1	1	1				0	1	0				1		
Bit 8						0	1	1	1	1	0	0						0			
Bit 16	0	1	0	0	1												0				

`bitset` - URI 2507 - Código de Hamming

Olhando para as paridades dos bits de Hamming, vemos que a paridade não bate para os bits 1, 4 e 8, que representa que houve um erro de leitura no bit (1+4+8) 13 e prontamente corrigimos. A você foi pedido um programa que analize as palavras de 21 bits lidas e imprima as palavras corretas de 16 bits.

Entrada:

A entrada contém vários casos de teste. Cada caso contém em uma única linha um valor hexadecimal de 21 bits: H, $0 \leqslant H \leqslant$ `1fffff`, os dígitos hexadecimais alfabéticos estão grafados em letras minúsculas.

Saída:

Para cada caso de teste da entrada seu programa deve gerar uma única linha de saída uma palavra hexadecimal de 16 bits. Contendo a informação lida corrigida pela técnica de Hamming e livre dos bits de paridade. Os dígitos hexadecimais em letras minúsculas.

URI 2507 - Solução - Cálculo da paridade

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

int paridade(const bitset<21>& x, int p) {
 int sum;

 switch(p) {
  case 1: sum =x[5]+x[6]+x[8]+x[9]+x[11]+x[13]+x[15]+x[16]+x[18]+x[20];
          break;
  case 2: sum =x[5]+x[7]+x[8]+x[10]+x[11]+x[14]+x[15]+x[17]+x[18];
          break;
  case 4: sum =x[6]+x[7]+x[8]+x[12]+x[13]+x[14]+x[15]+x[19]+x[20];
          break;
  case 8: sum =x[9]+x[10]+x[11]+x[12]+x[13]+x[14]+x[15];
          break;
  case 16:sum =x[16]+x[17]+x[18]+x[19]+x[20];
 }
 return (sum%2);
}
```

URI 2507 - Solução

```
int main() {
 int n, err;
 while(cin >> hex >> n) {
  bitset<21> x(n);
  err=(paridade(x,1)==x[0] ? 0 : 1)+(paridade(x,2)==x[1] ? 0 : 2) +
      (paridade(x,4)==x[2] ? 0 : 4) + (paridade(x,8)==x[3] ? 0 : 8) +
      (paridade(x,16)==x[4] ? 0 : 16);
  if(err != 0 && err != 1 && err != 2 && err != 4
     && err != 8 && err != 16) {
   if(err < 4) x.flip(err+2);
   else if(err < 8) x.flip(err+1);
   else if(err < 16) x.flip(err);
   else x.flip(err-1);
  }
  string s=x.to_string();
  s.erase(16,5);
  bitset<16> out(s);
  cout << hex << out.to_ulong() << endl;
 }
 return 0;
}
```

`complex`

- Um objeto da classe `complex` representa um número complexo $z = x + y * i$, onde i é a unidade base dos números imaginários: $i^2 = -1$.
- Esta classe possui duas compontentes: `real`, representado por x e `imag` representado por y.
- Por ser uma classe que representa um número ela possui características numéricas, inclusive a sobreposição de operadores: =, +=, -=, *=, /=, +, -, *, /, além dos unários +, -, os relacionais ==, !=, e os de stream de E/S: >> e <<.

complex

- Os métodos da classe são apenas: `real()` e `imag()` que retorna separadamente as partes real e imaginária do número.
- Mas estão definidas para o número complexo várias funções da biblioteca <cmath>:
  - `real` e `imag` (coordenadas cartesianas), `abs` e `arg` (coordenadas polares), `norm` (norma), `conj` (complexo conjugado) e `polar` (constrói um complexo a partir de suas coordenadas polares).
- Também são aplicáveis todas funções trigonométricas e hiperbólicas diretas e inversas, além do `exp`, `log`, `log10`, e `pow` e `sqrt`.

initializer_list

- Normalmente podemos inicializar um vetor a partir de uma lista de inicialização:
  `int a[] = {10, 20, 30, 40, 50};`
- Um `initializer_list` neste caso é representado por {10, 20, 30, 40, 50}.
- Muitas classes podem ser iniciadas a partir de um initializer_list.
- Um initializer list provê iterators para acessar os elementos.
- initializer_list é provido por `<initializer_list>`

pair

- Permite definir uma dupla de objetos unidos.
- Possui os atributos `first` e `second` para acesso direto aos elmentos do pair.
- Pode ser atribuído valores pelo operador = e possui a função `swap` para troca dos valores.
- Possui já pré-definido operadores relacionais, mas é interessante que cada uso do pair atribua as comparações de relação.
- É possível construir pair a partir de uma função externa: `make_pair` ou gerar um tuple a partir de um pair.

`tuple`

- Tuple são objetos que encapsulam elementos de (possivelmente) tipos diferentes juntos em um único objeto, semelhante a `pair`.
- Conceitualmente são semelhantes às antigas estruturas do C, mas ao invés dos elementos serem acessados pelo nome, são acessados por sua posição na tupla.
- A classe Tupla e Pair estão proximamente relacionadas, ambas estão definidas na biblioteca `<utility>`.

tuple

Métodos:
`tuple_size`: Retorna a quantidade de elementos.
`tuple_element`: Define uma classe que dá acesso a um elemento da tupla
`make_tuple`: Constrói tuplas
`forwared_as_tuple`: Permite criar uma lista de argumentos a ser encaminhada para funções.
`tie`: Liga variáveis a elementos da tupla.
`tuple_cat`: Concatena tuplas
`get`: Retorna elemento da tupla

`valarray`

- É um array de valores numéricos sobre o qual se realiza operações matemáticas.
- A maioria das operações matemáticas pode ser aplicada diretamente sobre o objeto `valarray`
- Somar dois `valarray` faz com que cada elemento some com o seu correspondente.
- Somar um `valarray` com uma constante faz com que a constante seja adicionada a cada elemento do vetor.

`array`

- Constrói um vetor de tamanho fixo.
- Um classe que representa a sintaxe de vetor [] declarado pelo tamanho e do tipo.
- Permite acesso aos elementos através de um random access iterator.
- Oferece reverse iterators.
- É uma classe que acrescenta à sintaxe de vetores [] a funcionalidade de métodos associados ao acesso e manipulação dos valores. Semelhante à classe `vector`
- A classe `vector` difere da `array` por ter seu tamanho dinâmico.

Container Adaptors

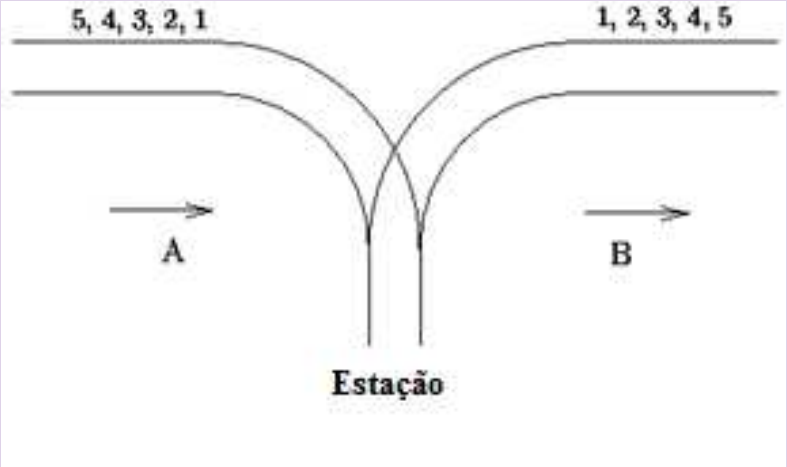
- São 3 os container adaptors, eles são construídos por sobre outros containers que ofereçam algumas funções específicas, principalmente variando entre `push_back`, `pop_back` ou `pop_front`, dependendo do estilo.
- Se nenhum container original for especificado, para `stack` e queue será usado deque, e para `priority_queue` o `vector`:

```
template <class T, class Container = deque<T> > class ...;
```

- Todos oferecem as seguintes funções: `empty`, `size`, `push`, `emplace`, `pop` e `swap`.
  - `stack`: Uma pilha LIFO, com a função específica: `top`
  - `queue`: Uma fila FIFO, com as funções específicas: `front` e `back`.
  - `priority_queue`: Uma fila de prioridade, se não especificado, o maior elemento está no início da fila. Sua contrução está baseada em um heap. As função específica é: `top`.

Exemplo: Problema URI 1062: Trilhos

Há uma famosa estação de trem na cidade PopPush. Esta cidade fica em um país incrivelmente acidentado e a estação foi criada no último século. Infelizmente os fundos eram extremamente limitados naquela época. Foi possível construir somente uma pista. Além disso, devido a problemas de espaço, foi feita uma pista apenas até a estação (veja figura abaixo).



Exemplo: Problema URI 1062: Trilhos

A tradição local é que todos os comboios que chegam vindo da direção A continuam na direção B com os vagões reorganizados, de alguma forma. Suponha que o trem que está chegando da direção A tem N ¡= 1000 vagões numerados **sempre** em ordem crescente **1,2, ..., N**. O primeiro que chega é o **1** e o último que chega é o **N**. Existe um chefe de reorganizações de trens que quer saber se é possível reorganizar os vagões para que os mesmos saiam na direção B na ordem a_1, a_2, a_n..
O chefe pode utilizar qualquer estratégia para obter a saída desejada. No caso do desenho ilustrado acima, por exemplo, basta o chefe deixar todos os vagões entrarem na estação (do 1 ao 5) e depois retirar um a um: retira o 5, retira o 4, retira o 3, retira o 2 e por último retira o 1. Desta forma, se o chefe quer saber se a saída 5,4,3,2,1 é possível em **B**, a resposta seria **Yes**. Vagão que entra na estação **só pode sair para a direção B** e é possível incluir quantos forem necessários para retirar o primeiro vagão desejado.

Entrada:

O arquivo de entrada consiste de um bloco de linhas, cada bloco, com exceção do último, descreve um trem e possivelmente mais do que uma requisição de reorganização. Na primeira linha de cada bloco há um inteiro N que é a quantidade de vagões. Em cada uma das próximas linhas de entrada haverá uma permutação dos valores **1,2, . . . , N**. A última linha de cada bloco contém apenas 0. Um bloco iniciando com zero (0) indica o final da entrada.

Saída:

O arquivo de saída contém a quantidade de linhas correspondente às linhas com permutações no arquivo de entrada. Cada linha de saída deve ser **Yes** se for possível organizar os vagões da forma solicitada e **No**, caso contrário. Há também uma linha em branco após cada bloco de entrada. No exemplo abaixo, O primeiro caso de teste tem 3 permutações para 5 vagões. O ultimo zero dos testes de entrada não devem ser processados.

Solução:

```
cin >> n;
while(n) {
 cin >> sai;
 while(sai) {
  stack<int> trem;
  cont=n-1;
  for(int entra=1; entra<= n; entra++) {
   trem.push(entra);
   while(!trem.empty() && sai==trem.top()) {
    trem.pop();
    cin >> sai; cont--;
   }
  }
  if(trem.empty()) cout << ''Yes''<< endl;
  else cout << ''No''<< endl;
  while(cont>=0) {
   cin >> sai; cont--;
  }
 }
 cout << endl;
 cin >> n;
}
```