ESTRUTURAS DE DADOS

Tabela Hash (implementação)

Roteiro

- **Aplicação da Estrutura**
- **Tipo Abstrato de Dados**
- **Detalhes de Implementação**
- **Uso da Implementação**

Aplicação da Estrutura

Podemos utilizar as tabelas hash sempre que queremos armazenar uma coleção de dados para depois obter os registros de maneira eficiente.

Se a função de hash foi bem estruturada, conseguiremos inserir e obter os valores em tempo constante.

Vamos supor que queremos organizar alunos em uma estrutura e, posteriormente, fazer buscas pelo registro acadêmico (ra).

- **A única informação que queremos é o nome.**

```
class Aluno{
private :
  int         ra;
  std::string nome;
public:
  Aluno();
  Aluno(int ra, std::string nome);
  string getNome() const;
  int getRa() const;
};
```

```
Aluno::Aluno(){
  this->ra   = -1;
  this->nome = "";
};
Aluno::Aluno(int ra, std::string nome){
  this->ra    = ra;
  this->nome  = nome;
}
string Aluno::getNome() const {
  return nome;
}
int Aluno::getRa() const{
  return ra;
}
```

Tipo Abstrato de Dados

```
class Hash {
 public:
  Hash(int max_items = 100);
  bool isFull() const;
  int getLength() const;

  void retrieveItem(Aluno& aluno, bool& found);
  void insertItem(Aluno aluno);
  void deleteItem(Aluno aluno);
  void print();
 private:
  int getHash(Aluno aluno);
  int   max_items;
  int   length;
  Aluno* structure;
};
```

Criaremos uma estrutura minimalista, apenas com busca, inserção e remoção.

Detalhes de Implementação

```
Hash::Hash(int max) {
  length = 0;
  max_items = max;
  structure = new Aluno[max_items];
}

bool Hash::isFull() const {
  return (length == max_items);
}

int Hash::getLength() const {
  return length;
}
```

Alocamos a estrutura dinamicamente na inicialização.

Em isFull, basta comparar o número de elementos inseridos com o tamanho máximo permitido.

Em retrieveItem, como não estamos tratando colisões, vamos apenas verificar se na localização indicada pela função de hash existe um elemento com a chave de busca requerida.

```
void Hash::retrieveItem(Aluno& aluno, bool& found) {
  int location = getHash(aluno);
  Aluno aux    = structure[location];
  if (aluno.getRa() == -1) {
    found      = false;
  } else {
    found      = true;
    aluno      = aux;
  }
}
```

Um RA igual a –1 indica uma posição vazia.

Tanto insertItem quanto deleteItem possuem implementações provisórias, em que sequer verificamos se a posição está ocupada, simplesmente inutilizamos a posição em ambos os casos.

```
void Hash::insertItem(Aluno aluno) {
  int location = getHash(aluno);
  structure[location] = aluno;
  length++;
}
void Hash::deleteItem(Aluno aluno) {
  int location = getHash(aluno);
  structure[location] = Aluno();
  length--;
}
```

O construtor vazio da classe aluno atribui –1 para o RA.

Para imprimir a estrutura na saída padrão, percorremos o vetor do início ao fim, imprimindo todos os valores presentes, incluindo os nulos.

```
void Hash::print() {
  for (int i = 0; i < max_items; i++) {
    cout << i <<":"<<
      structure[i].getRa()<<", "<<
      structure[i].getNome()<<endl;
  }
}
```

Finalmente, a função de hash. Nesta implementação, estamos simplesmente garantindo que não colocaremos um registro fora dos limites do vetor.

```
int Hash::getHash(Aluno aluno){
   return aluno.getRa() % max_items;
}
```

Uso da Implementação

Para utilizar, basta colocar alunos em uma variável da classe Hash

- **Vamos colocar 7 alunos em um hash com capacidade para 10.**

```
int main(){
  Hash alunosHash(10);
   int   ras[7]   = {
     12704, 31300, 1234,
     49001, 52202, 65606,
     91234};
  string nomes[7] = {
    "Pedro", "Raul", "Paulo",
    "Carlos", "Lucas", "Maria",
    "Samanta"};
```

Primeiramente, vamos alocar os elementos e olhar o resultado.

```
for (int i = 0; i < 7; i++) {
  Aluno aluno = Aluno(ras[i], nomes[i]);
  alunosHash.insertItem(aluno);
}
alunosHash.print();
cout << "------------------------------" <<  endl;
```

```
0:31300, Raul
1:49001, Carlos
2:52202, Lucas
3:-1,
4:91234, Samanta
5:-1,
6:65606, Maria
7:-1,
8:-1,
9:-1,
```

Onde estão Pedro e Paulo?

- Foram apagados para dar lugar a outros alunos por causa das colisões.

Podemos testar o método de busca. Por exemplo, vamos verificar quem é o elemento com RA 12704.

```
Aluno aluno(12704,"");
bool      found = false;
alunosHash.retrieveItem(aluno, found);
cout << aluno.getNome() << " -> " << found << endl;
```

```
Samanta -> 1
```

Por fim, vamos remover a Samanta:

```
alunosHash.deleteItem(aluno);
alunosHash.print();
```

```
0:31300, Raul
1:49001, Carlos
2:52202, Lucas
3:-1,
4:-1,
5:-1,
6:65606, Maria
7:-1,
8:-1,
9:-1,
```

ESTRUTURAS DE DADOS

Tabela Hash (implementação)