Dredd - Juiz Online

Principal | Perfil | Minhas Provas | Sair

Minutos Restantes: 13056

Usuário: Lucas Gomes Colombo

Notas:
Q1: ?
Q2: 3
Q3: ?
Q4: ?
Q5: ?
Q6: ?
Q7: 75.8
Q8: ?
Q9: ?
Q10: ?
Q11: ?
Q12: ?
Q13: ?
Q14: ?
Q15: ?
Q16: ?
Total: 5

REO 3 - Lista, Deque, Sequence Set e Hash

Prova Aberta Até: 09/08/2022 23:59:59

Número Máximo de Tentativas: 3

Atenuação da Nota por Tentativa: 0%

Instruções para a prova: A lista é individual. Você pode trocar ideias com colegas, desde que não copie o código.
Em todas as questões:
=> **Não é permitido** o uso de classes da STL ou bibliotecas similares para as estruturas sendo utilizadas.
=> **Não é permitida** a "quebra da estrutura", devendo-se respeitar as propriedades e características de cada estrutura.

Questão 1: Criar lista duplamente encadeada com método inverte()

Você deve implementar uma lista duplamente encadeada adicionando o método **inverte()**, que inverte a ordem dos elementos da lista sem modificar o conteúdo de cada nó. O método não pode fazer alteração do *valor* armazenado nos nós, podendo alterar somente as referências aos atributos *próximo* e *anterior* dos mesmos. Ou seja, não serão aceitas soluções que façam as trocas de valores dos nós, os nós devem mudar de posição, por meio da manipulação de ponteiros.

Obs: não devem ser utilizadas estruturas de dados adicionais para a solução do problema

Entradas:

Uma sequência de caracteres e palavras tal que:

- I: Insere uma palavra na lista original
- X: Executa a função inverte
- P: Imprime a lista em ordem direta
- R: Imprime a lista em ordem reversa
- Q: Encerra os comandos

Saídas:

1. Impressões da lista pelos comandos
2. Impressão da lista em ordem direta e reversa após o comando Q

Exemplo de Entrada:

```
I Horacio I Sales I Beltrao P X P I GCC216 Q
```

Exemplo de Saída:

```
Horacio Sales Beltrao
Beltrao Sales Horacio
Beltrao Sales Horacio GCC216
GCC216 Horacio Sales Beltrao
```

Exemplo de Entrada:

```
I 1 I 2 I 3 I 4 I 5 I 6
X
P
R
X
P
R
I 7 I 8 I 9
X
R
I 10 I 11 I 12
X
P
I 13
Q
```

Exemplo de Saída:

```
6 5 4 3 2 1
1 2 3 4 5 6
1 2 3 4 5 6
6 5 4 3 2 1
1 2 3 4 5 6 7 8 9
12 11 10 1 2 3 4 5 6 7 8 9
```

```
12 11 10 1 2 3 4 5 6 7 8 9 13
13 9 8 7 6 5 4 3 2 1 10 11 12
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 2: Deque - controle de undo

Vários programas tem um comando para desfazer a operação anterior, conhecido como *undo*. Usar este comando seguidamente faz com que a última ação seja desfeita, depois a penúltima e assim por diante.

Para não gastar muita memória é comum que haja um limite de ações disponíveis para o *undo*. Por exemplo, pode ser possível desfazer apenas as últimas 20 operações realizadas.

Este tipo de funcionalidade é mais fácil de controlar usando uma estrutura do tipo *deque* (*double ended queue*). Use alguma implementação de fila ou de lista para criar uma classe `Deque`. Depois, faça um programa para guardar e retirar linhas de texto, que representam comandos.

Entradas:

Inicialmente seu programa deve ler um número natural que representa a quantidade máxima de instruções que a estrutura de dados vai guardar. Depois várias linhas de texto serão lidas. Sempre que a linha for "undo", o programa deve dizer que desfez a última instrução. Se a instrução não for "undo", o programa deve guardar a linha na lista de coisas a desfazer. Se a capacidade máxima for atingida, o programa deve "esquecer" a instrução mais antiga antes de inserir a nova instrução na estrutura. O comando especial "sair" faz com que a execução termine.

1. Capacidade da fila de comandos;
2. comandos, cada um numa linha.

Saídas:

O comando "undo" deve produzir a saída: `desfazer: comando que foi desfeito`. Toda vez que a capacidade máxima for atingida, o programa deve produzir a saída: esqueci: comando que foi removido. No caso de tentarem desfazer ação que não existe, o programa deve escrever ERRO (letras maiúsculas) na saída padrão.

Exemplo de Entrada:

```
3
comando um
comando dois
undo
comando três
comando quatro
comando cinco
sair
```

Exemplo de Saída:

```
desfazer: comando dois
esqueci: comando um
```

Peso: 1

Última tentativa realizada em: 27/07/2022 09:28:44

Tentativas: 1 de 3

Nota (0 a 100): 3

Status ou Justificativa de Nota: A quantidade de dados escritos pelo programa é diferente da quantidade de dados esperados.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 3: Lista encadeada - Divisão de listas

Implemente um método que recebe como parâmetro um valor inteiro ***N*** e divide a lista encadeada em duas. A lista atual será mantida até a posição ***N*** , e o retorno do método será uma segunda lista, que começa no primeiro nó logo após a posição ***N*** na lista original. No caso do valor inteiro ser maior que a última posição da lista, o resultado deve ser a impressão da lista original e o valor -1; se o valor inteiro corresponder exatamente à posição do último elemento da lista, são impressas a lista original e uma linha em branco.

Obs: a inserção é sempre no fim da lista.

Entradas:

1. Número de elementos a serem inseridos na lista.
2. Elementos a serem inseridos na lista.
3. Valor inteiro indicando a posição na qual a lista deve ser separada.

Saídas:

1. Elementos da lista original até a posição ***N***.
2. Elementos da lista criada após a divisão.

Exemplo de Entrada:

```
8
3 9 4 0 11 56 21 73
4
```

Exemplo de Saída:

```
3 9 4 0 11
56 21 73
```

Exemplo de Entrada:

```
5
3 9 4 0 11
4
```

Exemplo de Saída:

```
3 9 4 0 11
```

Exemplo de Entrada:

```
3
3 9 4
3
```

Exemplo de Saída:

```
3 9 4
-1
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 4: Lista - Controle de um voo

Você como um grande programador, foi contratado por uma empresa de aviação para construir um programa que controla a aquisição e espera por vagas em um determinado voo, que tem capacidade para 200 passageiros.

O programa deve operar de forma que:

- Cada assento é identificado por um número único (1, 200).
- Quando uma passagem é comprada, são armazenados os dados do cliente (CPF, nome e telefone de contato) e ele é vinculado ao assento.
- Cada assento tem sua própria lista de espera, e devem ser armazenados os dados do cliente que opta por entrar nessa lista.

As operações previstas são:

1. Aquisição de uma passagem: uma passagem pode ser diretamente adquirida caso o assento escolhido pelo passageiro esteja disponível. Caso contrário, o cliente deve ser alocado na lista de espera do assento.
2. Cancelamento de uma passagem: uma passagem pode ser cancelada segundo a vontade do cliente. Sempre que uma passagem é cancelada, o primeiro passageiro da lista de espera deve ser alocado como ocupante do assento em questão.
3. Consultar a disponibilidade de um assento: é informado se esse está disponível ou não, caso não, é impresso o tamanho da lista de espera.

Restrições:

- Um passageiro não pode figurar em mais de um assento, assim caso esteja em uma lista de espera, ele não poderá estar em outra.

Formato das entradas:

```
1 {identificador do assento} {CPF do cliente} {Nome} {Telefone}
2 {CPF do cliente}
3 {identificador do assento}
```

Observações:

1. Devem ser aceita entradas até que a opção seja diferente das permitidas (1, 2 ou 3).

Exemplo de Entrada:

```
1 10 123.345.678-90 Joao da Silva 99123456
1 36 321.654.987-09 Maria Loca 88214365
1 10 987.654.321-12 Carlos Mendes 98456789
1 154 456.123.789-34 Jose Rodrigues 91236589
3 36
3 10
3 191
2 456.123.789-34
2 123.345.678-90
3 10
1 76 111.222.333-45 Pedro Paulo 92326589
1 76 333.222.111-12 Ana Paula 87452365
2 333.222.111-12
-1
```

Exemplo de Saída:

```
Ocupado - Espera: 0
Ocupado - Espera: 1
Disponível
Ocupado - Espera: 0
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 5: Lista encadeada - remove repetidos

Implemente um método *removeRepetidos()* em listas dinamicamente duplamente encadeadas de modo a percorrer seus elementos e remove aqueles que estão repetidos. Em caso de repetição deverá ser mantida a primeira ocorrência do elemento.

Todos os valores devem ser inseridos numa lista **antes** da ativação do método `RemoverRepetidos`. **Não transfira a remoção de repetidos para a inserção. Não utilize estruturas auxiliares para a tarefa.**

Entradas:

1. Número de elementos a serem inseridos em L1
2. Elementos de L1

Saídas:

1. Elementos restantes em L1 (impressão normal e reversa)

Exemplo de Entrada:

```
10
1 3 5 7 1 9 3 8 2 9
```

Exemplo de Saída:

```
1 3 5 7 9 8 2
2 8 9 7 5 3 1
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 6: Lista dinamicamente encadeada

O arquivo disponível possui o esqueleto básico de uma lista dinamicamente encadeada, contudo diversas funções e procedimentos não estão implementadas. Sua tarefa é implementar os métodos faltando (`insere`, `remove`, `procura` e `vazia`).

Não altere a função main do código pois ele será utilizado pelo Dredd para validar seu código. Não altere a característica de encadeamento simples da lista. As posições na lista começam em zero.

Entradas:

Uma sequência de caracteres e números tal que:

- I: Significa a inserção de um valor
- W: Significa a inserção de um valor em uma posição
- P: Procura por um valor específico e imprime a posição
- R: Remove o valor de uma determinada posição
- V: Verifica se a lista é vazia
- Q: Encerra os comandos

Saídas:

1. A sequência resultante dos comandos de entrada
2. A lista resultante

Exemplo de Entrada:

```
I 10 I 5 I 2 R 1 V P 10 Q
```

Exemplo de Saída:

```
0
0
10 2
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 7: Remover interseção em lista

Faça uma implementação para a manipulação de listas dinamicamente duplamente encadeadas de modo a implementar um método *remove()*, que recebe, como parâmetro de entrada, outro objeto do tipo *lista*, ou seja, dada L1.remove(L2), o método deverá percorrer L1 removendo os elementos identificados em L2. Se todos os elementos de L1 forem removidos, o programa deve imprimir o valor -1 apenas uma vez.

Sugestão: utilize referências na implementação do método, para evitar necessidade de implementação do construtor de cópia: `void remove(listadup& L2)`.

Obs: os dados são inseridos no fim da lista.

Entradas:

1. Número de elementos a serem inseridos em L1.
2. Elementos a serem inseridos em L1.
3. Número de elementos a serem inseridos em L2.
4. Elementos a serem inseridos em L2.

Saídas:

1. Elementos de L1 (impressão normal e reversa).

Minutos Restantes: 13056

Usuário: Lucas Gomes Colombo

Notas:
Q1: ?
Q2: 3
Q3: ?
Q4: ?
Q5: ?
Q6: ?
Q7: 75.8
Q8: ?
Q9: ?
Q10: ?
Q11: ?
Q12: ?
Q13: ?
Q14: ?
Q15: ?
Q16: ?
Total: 5

Exemplo de Entrada:

```
8

1 8 3 6 2 7 4 5

3

3 5 8
```

Exemplo de Saída:

```
1 6 2 7 4

4 7 2 6 1
```

Peso: 1

Última tentativa realizada em: 26/07/2022 21:59:04

Tentativas: 1 de 3

Nota (0 a 100): 75.8

Status ou Justificativa de Nota: A quantidade de dados escritos pelo programa é diferente da quantidade de dados esperados.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 8: Lista Encadeada em Arranjo - Implementação

O código fornecido implementa uma lista duplamente encadeada armazenando os dados em arranjo, faltando os métodos de inserção e remoção apenas. Você deverá justamente implementá-los.

A implementação disponibilizada controla as posições vazias disponíveis no arranjo por meio de uma lista encadeada de posições vazias. Para obter a próxima posição vazia, basta utilizar o método `alocaPosicao()`.

A função principal encontra-se implementada e não pode ser alterada. Você poderá modificar e adicionar métodos na lista, a seu critério.

Entradas:

O programa lê códigos de comandos a executar. Sempre um caractere identificando o comando, seguido dos parâmetros necessários para executar o comando, se houverem. Os códigos de comandos são:

1. `I`: para inserir um elemento no início da lista;
2. `i`: para inserir um elemento no fim da lista;
3. `R`: para remover um elemento no início da lista;
4. `r`: para remover um elemento no fim da lista;
5. `p`: para imprimir os elementos da lista, seguindo o encadeamento do primeiro ao último;
6. `d`: para imprimir os vetor de dados da lista, para depuração;
7. `s`: para sair da aplicação.

Saídas:

Apenas os comandos de impressão e depuração possuem saída, sendo que já foram implementados.

Exemplo de Entrada:

```
4
i 2
i 4
r
i 5
R
I 0
p
d
i 6
I 7
p
d
```

```
r
R
r
R
I 8
p
d
s
```

Exemplo de Saída:

```
0 5
5 0
2/{0-1}/2:(-1,[0],1)(0,[5],-1)(-1,[-1],3)(-1,[-1],-1)
7 0 5 6
6 5 0 7
4/{3-2}/-1:(3,[0],1)(0,[5],2)(1,[6],-1)(-1,[7],0)
8
8
1/{0-0}/1:(-1,[8],-1)(0,[5],3)(1,[6],-1)(-1,[7],2)
```

Exemplo de Entrada:

```
10
i 20 i 30 i 50 i 70 i 90 i 100 i 1 i 2 i 3 i 4
p
d
r r R R
I 5 I 6
i 7 i 8
p
d
r r R R
r r R R
i 200 i 300
I 9 I 10
p
d
s
```

Exemplo de Saída:

```
20 30 50 70 90 100 1 2 3 4
4 3 2 1 100 90 70 50 30 20
10/{0-9}/-1:(-1,[20],1)(0,[30],2)(1,[50],3)(2,[70],4)(3,[90],5)(4,[100],6)(5,[1],7)(6,[2],8)(7,[3],9)(8,[4],
6 5 50 70 90 100 1 2 7 8
8 7 2 1 100 90 70 50 5 6
10/{0-9}/-1:(-1,[6],1)(0,[5],2)(1,[50],3)(2,[70],4)(3,[90],5)(4,[100],6)(5,[1],7)(6,[2],8)(7,[7],9)(8,[8],-1
10 9 90 100 200 300
300 200 100 90 9 10
6/{7-2}/1:(-1,[6],8)(-1,[5],0)(3,[300],-1)(5,[200],2)(6,[90],5)(4,[100],3)(7,[9],4)(-1,[10],6)(7,[7],9)(8,[8
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 9: Lista – Remover repetidos de lista duplamente encadeada

Utilizando o código fornecido, implemente o método `RemoverRepetidos` da classe `Lista` que remove todos os elementos repetidos de uma lista duplamente encadeada, mantendo somente a primeira ocorrência de cada valor.

Para testar seu método, já está disponível um programa que lê vários números inteiros positivos, insere-os numa lista, remove os repetidos e depois escreve as saídas do problema. Todos os valores devem ser inseridos numa lista antes da ativação do método `RemoverRepetidos`. Não transfira a remoção de repetidos para a inserção.

Entradas:

1. Elementos da lista (números inteiros positivos).

Saídas:

1. Quantidade de elementos da lista após a remoção dos repetidos.
2. Os elementos da lista, após a remoção dos repetidos.

Exemplo de Entrada:

```
12 2 3 4 8 3 6 4 7 1  -1
```

Exemplo de Saída:

```
8
(12) (2) (3) (4) (8) (6) (7) (1)
```


Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 10: Criar hash com método para simular conjuntos

Uma estrutura de dados do tipo conjunto (*set*) é uma coleção de elementos. Dado os conjuntos *S*, *A* e *B*, e um elemento *x*, um conjunto geralmente implementa as seguintes operações básicas:

- `membro(S,x)` – Retorna true se o item *x* for membro do conjunto *S*.
- `adiciona(S,x)` – Adiciona o elemento *x* no conjunto *S*, descartando duplicatas. Se *x* já fizer parte de *S*, não é gerado erro, apenas não se modifica o conteúdo de *S*.
- `remove(S,x)` – Remove o elemento *x* do conjunto *S*. Se *x* não fizer parte de *S*, não é gerado erro, apenas não se modifica o conteúdo de *S*.
- `tamanho(S)` – Retorna o número de elementos de S.
- `imprime(S)` – Imprime os elementos de S. Caso o conjunto seja vazio, imprime {}
- `uniao(A,B)` – Retorna um novo conjunto $A \cup B$, formado pela união dos elementos de *A* e *B*.
- `intersecao(A,B)` – Retorna um novo conjunto $A \cap B$, formado pela interseção dos elementos de *A* e *B*.
- `diferenca(A,B)` – Retorna um novo conjunto *A* - *B*, formado pelos elementos de *A* que não fazem parte de *B*.

Como pode-se perceber, um conjunto é muito similar a uma lista, com algumas particularidades (a impossibilidade de duplicatas, por exemplo). Conjuntos podem ser implementados de várias formas, incluindo: i) dicionários (tabelas hash); ii) árvores ou iii) listas dinâmicas com ponteiros. No caso de listas dinâmicas, os dados podem ainda ser armazenados por meio de ponteiros ou arranjos.

Neste exercício, você deverá implementar a hash e os métodos para simular as operações básicas de conjuntos listadas anteriormente. Perceba que as operações de `uniao(A,B)`, `intersecao(A,B)` e `diferenca(A,B)` não precisam ser necessariamente implementadas, uma vez que estamos utilizando dicionários (**tabelas hash**).

Para este problema em específico, o conjunto estará limitado a 100 elementos inteiros, portanto, utilize a função de espelhamento dada por `hash(k) = k mod 101`. O vetor de armazenamento do hash terá obviamente 101 posições. O valor associado à chave (o elemento) a ser armazenado é 1 (para indicar pertinência do elemento ao conjunto) ou -1 (quando o elemento não pertence ao conjunto). Obviamente, como o conjunto inicialmente encontra-se vazio, todas os valores nas posições do vetor de armazenamento devem ser inicializadas em -1. Obviamente também, por usar um vetor para armazenamento, **o tratamento de colisões deverá ser por endereçamento aberto.**

Além da estrutura de dados, você deverá implementar uma aplicação básica para teste, que irá criar um conjunto e executará uma série de operações na seguinte ordem:

1. Constrói o conjunto a partir de uma lista de 10 elementos digitada pelo usuário.
2. Remove três elementos indicados pelo usuário.
3. Verifica se um dado valor informado pelo usuário é membro do conjunto (imprima 1 para pertinência e -1 para não-pertinência.
4. Repete a operação de verificação de pertinência para outro valor informado pelo usuário.
5. Imprime o conteúdo do conjunto, na ordem em que estão armazenados.
6. Adiciona três novos elementos indicados pelo usuário
7. Imprime novamente o conteúdo do conjunto, na ordem em que estão armazenados.

Entradas:

1. 10 elementos a serem inseridos no conjunto
2. 3 elementos a serem removidos do conjunto
3. 2 valores para verificação de pertinência
4. 3 elementos a serem inseridos no conjunto

Saídas:

1. Informação sobre pertinência de um valor fornecido pelo usuário
2. Informação sobre pertinência de outro valor fornecido pelo usuário
3. Elementos do conjunto
4. Elementos do conjunto, após inserção de novos elementos

Exemplo de Entrada:

```
1 2 3 4 5 6 7 8 9 10
9 10 11
7 10
1 9 10
```

Exemplo de Saída:

```
1
-1
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9 10
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 11: Tabela hash - Dicionário

Utilize uma tabela hash para implementar um dicionário de capacidade para 23 palavras, sua tabela terá o tratamento de colisões por encadeamento. Seu programa deve ser capaz de armazenar e buscar por uma palavra e seu significado. Caso seja feita uma busca por uma palavra inexistente em seu dicionário, o programa imprimirá NULL no lugar do significado da palavra. Seu programa deve permitir quantas buscas forem desejadas pelo usuário, sendo -1 a palavra que representa a intenção de termino do programa. **OBS:** Sua função que faz o hash será da seguinte forma: tamanho_da_palavra mod 23.

Entradas:

1. Quantidade de palavras a serem inseridas.
2. Palavra.
3. Significado da Palavra.
4. Palavras buscadas.

Saídas:

1. [Palavra_buscada] => Significado da palavra buscada.

Exemplo de Entrada:

```
5
Casa Edifício de formatos e tamanhos variados, ger. de um ou dois andares, quase sempre destinado à habitaçã
Aniversário Diz-se de ou dia em que se completa um ou mais anos em que se deu determinado acontecimento.
Carta Mensagem, manuscrita ou impressa, a uma pessoa ou a uma organização, para comunicar-lhe algo.
Exonerado  Libertar ou libertar-se de uma obrigação ou de um dever.
Concomitantemente Que acompanha ou coexiste. Que acontece ou se faz ao mesmo tempo.
Carta
Casa
Boneca
-1
```

Exemplo de Saída:

```
[Carta] => Mensagem, manuscrita ou impressa, a uma pessoa ou a uma organização, para comunicar-lhe algo.
[Casa] => Edifício de formatos e tamanhos variados, ger. de um ou dois andares, quase sempre destinado à hab
[Boneca] => NULL
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 12: Hash com Vetor de Quatro Posições (v2)

A implementação de Hash com encadeamento é uma das mais utilizadas. Para o encadeamento, a implementação tradicional faz uso de uma lista encadeada. Entretanto, várias implementações utilizam estruturas diferenciadas para o encadeamento das colisões. Por exemplo, a classe HashMap no Java 8 utiliza uma árvore binária para essa tarefa.

Implemente uma tabela hash com um número genérico `n` de posições, informado durante a construção da tabela, com resolução de colisões utilizando um vetor de quatro posições. Inclusive, o primeiro elemento na posição deverá ser armazenado também nesse vetor, na posição 0. Utilize como hash a função `h(k) = k mod n`, em que `k`, um inteiro positivo, é a chave do registro sendo armazenado e `n` é o número de posições da tabela. Não é necessário implementar métodos para alteração ou remoção de dados, apenas inserção e recuperação. Chaves não utilizadas devem ser mantidas como -1, tendo a string vazia como valor associado.

Entradas:

1. Tamanho da tabela hash
2. Quantidade de dados a serem inseridos na tabela hash
3. Conjunto de dados a serem inseridos, cada um possuindo uma chave e uma palavra (como valor)
4. Quatro chaves para recuperação

Saídas:

1. Texto "ITEM JÁ ESTÁ NA TABELA!" quando da tentativa de inserção de novo elemento com uma chave já existente
2. Texto "NÚMERO MÁXIMO DE COLISÕES PARA CHAVE!" quando da tentativa de inserção de um quinto elemento na mesma posição do hash
3. Valores recuperados ou o texto "NÃO ENCONTRADO!"
4. Impressão da estrutura da tabela hash, como exemplificado a seguir

Exemplo de Entrada:

```
5
5
5 primeiro
10 segundo
15 terceiro
20 quarto
25 ih
3
2
5
15
```

Exemplo de Saída:

```
NÚMERO MÁXIMO DE COLISÕES PARA CHAVE!
NÃO ENCONTRADO!
NÃO ENCONTRADO!
primeiro
terceiro
0:[5/primeiro][10/segundo][15/terceiro][20/quarto]
1:[-1/][-1/][-1/][-1/]
2:[-1/][-1/][-1/][-1/]
3:[-1/][-1/][-1/][-1/]
4:[-1/][-1/][-1/][-1/]
```

Exemplo de Entrada:

```
5
10
5 primeiro
10 segundo
15 terceiro
20 quarto
6 xxxx
7 yyyy
11 aaaa
12 bbbb
16 aleluia
13 perae
3
13
5
15
```

Exemplo de Saída:

```
NÃO ENCONTRADO!
perae
primeiro
terceiro
0:[5/primeiro][10/segundo][15/terceiro][20/quarto]
1:[6/xxxx][11/aaaa][16/aleluia][-1/]
2:[7/yyyy][12/bbbb][-1/][-1/]
3:[13/perae][-1/][-1/][-1/]
4:[-1/][-1/][-1/][-1/]
```

Exemplo de Entrada:

```
5
9
5 primeiro
10 segundo
15 terceiro
20 quarto
6 xxxx
15 111
20 300
301 301
399 309
111
15
1511
20
```

Exemplo de Saída:

```
ITEM JÁ ESTÁ NA TABELA!
ITEM JÁ ESTÁ NA TABELA!
NÃO ENCONTRADO!
terceiro
NÃO ENCONTRADO!
quarto
0:[5/primeiro][10/segundo][15/terceiro][20/quarto]
1:[6/xxxx][301/301][-1/][-1/]
2:[-1/][-1/][-1/][-1/]
3:[-1/][-1/][-1/][-1/]
4:[399/309][-1/][-1/][-1/]
```


Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 13: Hash - Redimensionamento

O objetivo das tabelas hash de ser uma estrutura de dados com tempo de acesso constante aos elementos é prejudicado pela existência de colisões. Quanto mais colisão houver, mais custoso é o acesso a alguns elementos da tabela. Uma forma de mitigar esse problema é expandir o espaço de endereços da tabela sempre que houver muitos valores num mesmo endereço. No caso em que há encadeamento de valores, a expansão pode ser feita recalculando a posição de cada valor e movendo o nó (chave/valor) para um novo encadeamento.

Implemente uma tabela hash com tratamento de colisão por encadeamento - novos elementos devem ser inseridos no fim da lista. A função para cálculo da posição em que um elemento deve ser inserido é ***chave % capacidade*** e a operação de redimensionamento deve ser realizada quando a inserção de um elemento fizer com que 70% das posições da tabela sejam ocupadas. A capacidade da tabela deve ser dobrada a cada redimensionamento.

As funcionalidades disponíveis são:

1. I - Inserir item na tabela
2. R - Remover item - caso seja solicitada a exclusão de um item inexistente, o programa deve exibir a mensagem "ERRO".
3. B - Buscar item
4. P - Imprimir tabela
5. S - Sair

Entradas:

1. Capacidade da tabela
2. Funcionalidades a serem executadas.

Saídas:

1. Resposta das funcionalidades selecionadas.

Exemplo de Entrada:

```
10
I 32 TRINTADOIS
I 23 VINTETRES
I 2 DOIS
I 21 VINTEUM
P
I 14 QUATORZE
I 69 SESSENTANOVE
I 42 QUARENTADOIS
I 11 ONZE
I 51 CIQUENTAUM
P
I 13 TREZE
I 7 SETE
I 88 OITENTAOITO
P
S
```

Exemplo de Saída:

```
Posicao 0:
Posicao 1: 21 VINTEUM
Posicao 2: 32 TRINTADOIS
Lista de colisao: 2 DOIS
Posicao 3: 23 VINTETRES
Posicao 4:
Posicao 5:
Posicao 6:
```

```
Posicao 7:
Posicao 8:
Posicao 9:
-----------------------
Posicao 0:
Posicao 1: 21 VINTEUM
Lista de Colisao: 11 ONZE, 51 CINQUENTAUM
Posicao 2: 32 TRINTADOIS
Lista de colisao: 2 DOIS, 42 QUARENTADOIS
Posicao 3: 23 VINTETRES
Posicao 4: 14 QUATORZE
Posicao 5:
Posicao 6:
Posicao 7:
Posicao 8:
Posicao 9: 69 SESSENTANOVE
-----------------------
Posicao 0:
Posicao 1: 21 VINTEUM
Lista de Colisao: 11 ONZE, 51 CINQUENTAUM
Posicao 2: 32 TRINTADOIS
Lista de colisao: 2 DOIS, 42 QUARENTADOIS
Posicao 3: 23 VINTETRES
Lista de colisao: 13 TREZE
Posicao 4: 14 QUATORZE
Posicao 5:
Posicao 6:
Posicao 7: 7 SETE
Posicao 8: 88 OITENTAOITO
Posicao 9: 69 SESSENTANOVE
Posicao 10:
Posicao 11:
Posicao 12: 32 TRINTADOIS
Posicao 13: 13 TREZE
Posicao 14: 14 QUATORZE
Posicao 15:
Posicao 16:
Posicao 17:
Posicao 18:
Posicao 19:
-----------------------
```

Exemplo de Entrada:

```
10
I 10 DEZ
I 2 DOIS
I 22 VINTEDOIS
I 15 QUINZE
I 32 TRINTADOIS
R 25
P
I 18 DEZOITO
I 9 NOVE
R 32
P
S
```

Exemplo de Saída:

```
ERRO
Posicao 0: 10 DEZ
Posicao 1:
Posicao 2: 2 DOIS
Lista de colisao: 22 VINTEDOIS, 32 TRINTADOIS
Posicao 3:
Posicao 4:
Posicao 5: 15 QUINZE
Posicao 6:
Posicao 7:
Posicao 8:
Posicao 9:
-----------------------
Posicao 0: 10 DEZ
Posicao 1:
Posicao 2: 2 DOIS
Lista de colisao: 22 VINTEDOIS
Posicao 3:
Posicao 4:
Posicao 5: 15 QUINZE
Posicao 6:
Posicao 7:
Posicao 8: 18 DEZOITO
Posicao 9: 9 NOVE
-----------------------
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 14: Tabela Hash – Aniversários

Sua empresa passará a presentear seus funcionários com um presente aleatório no dia de seus respectivos aniversários. Para isso ela necessita de um sistema que seja possível cadastrar os funcionários, sendo necessários armazenar seu nome e dia, mês e ano de seu aniversario, e então dada uma data saber quantos presentes necessitam ser preparados. Sua tarefa é desenvolver esse programa utilizando uma tabela hash, de forma que são cadastrados vários funcionários e então é fornecida uma data (dia/mês), com isso o programa deve exibir a quantidade de presentes necessários naquele dia.

A tabela hash deve ser criada com uma quantidade de posições fornecida no início da execução. Após escrever a quantidade de presentes necessários, escreva a porcentagem de posições da tabela que estão usadas na forma de um número entre 0 e 1. Use, a seu critério, o código fornecido, que já tem pronto o tipo funcionário e outras coisas. Não é necessário usar os mesmos métodos que estão no programa, mas os métodos que não forem usados devem ser apagados.

A mesma função hash que está no programa deve ser usada: (dia * mes - 1) *mod* limite (onde *mod* é a operação de resto da divisão).

Entradas:

1. Quantidade de posições que a tabela hash deve ter.
2. Quantidade de funcionários a serem cadastrados.
3. Nome de cada funcionário (uma palavra só) com sua data de nascimento (3 inteiros, não é necessário verificar se a data é válida).
4. Uma data qualquer (dois inteiros para o dia e mês).

Saída:

1. Quantidade de aniversariantes naquele dia.
2. Porcentagem de posições usadas na tabela (número real entre 0 e 1).

Exemplo de Entrada:

```
4
5
João 10 05 1985
Maria 25 09 1976
Marcos 03 01 1992
Paulo 06 07 1993
Pedro 25 09 1996
25 09
```

Exemplo de Saída:

```
2 0.75
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta

Questão 15: Hash com encadeamento (lista implícita) e chaves repetidas

Implemente as funções de inserir e recuperar em uma tabela hash, com chaves e valores em formato de string, utilizando como base o código fornecido pelos professores, com um número genérico *n* de posições, informado durante a construção da tabela, com resolução de colisões por encadeamento. Utilize a função de hash disponibilizada no código.

O tratamento de colisão deverá ser feito por meio de uma lista implícita (a tabela é um vetor, em que cada posição é um ponteiro para um nó). As colisões deverão ser sempre colocadas no final da lista. A inserção deverá permitir elementos com chaves repetidas, sendo que cada par chave/valor será armazenado em um nó diferente. Além disso, quando da recuperação de elementos com uma mesma chave, todos os valores associados a essa chave deverão ser concatenados. Quando a chave a ser recuperada não estiver no hash, deverá ser retornada a string vazia ("").

Lembrete: Em C++, valores do tipo `string` são concatenados usando o operador +.

A função principal já se encontra implementada e executa a entrada e saída de dados.

Entradas:

O programa (função principal) inicialmente lê o tamanho (capacidade do hash). Em seguida, ele lê uma série de comandos e executá-los até receber o comando para terminar a execução. Os comandos possíveis são:

- "I" para inserir um dado no hash; ele é seguido da chave e do valor a ser inserido (ex: `I sapo boi` insere o valor "boi" com a chave "sapo" no hash);
- "R" para recuperar os valores associados a uma dada chave no hash; ele é seguido da chave em questão (ex: `R nota` irá recuperar uma string resultante da concatenação de todas os valores associados à chave "nota");
- "P" para imprimir o conteúdo do hash (para depuração);
- "Q" para terminar a execução do programa

Saídas:

Os comandos de retiradas produzem saída, assim como o de impressão do hash.

Exemplo de Entrada:

```
5
I A A
I B B
I C C
I D D
I E E
I F F
I G G
P
I H H
I I I
I J J
I K K
P
R A
R C
I A a
I B b
I C c
R A
R B
P
I A VOGAL
I B CONSOANTE
R A
R B
P
Q
```

Exemplo de Saída:

```
0:[A/A]->[F/F]->NULL  1:[B/B]->[G/G]->NULL  2:[C/C]->NULL  3:[D/D]->NULL  4:[E/E]->NULL
0:[A/A]->[F/F]->[K/K]->NULL  1:[B/B]->[G/G]->NULL  2:[C/C]->[H/H]->NULL  3:[D/D]->[I/I]->NULL  4:[E/E]->
[J/J]->NULL
A
C
Aa
Bb
0:[A/A]->[F/F]->[K/K]->[A/a]->NULL  1:[B/B]->[G/G]->[B/b]->NULL  2:[C/C]->[H/H]->[C/c]->NULL  3:[D/D]
->[I/I]->NULL  4:[E/E]->[J/J]->NULL
AaVOGAL
BbCONSOANTE
0:[A/A]->[F/F]->[K/K]->[A/a]->[A/VOGAL]->NULL  1:[B/B]->[G/G]->[B/b]->[B/CONSOANTE]->NULL
2:[C/C]->[H/H]->[C/c]->NULL  3:[D/D]->[I/I]->NULL  4:[E/E]->[J/J]->NULL
```

Exemplo de Entrada:

```
10
I 13 TREZE
I 3 TRES
I 23 VINTETRES
I 2 DOIS
I 5 CINCO
P
I 8 OITO
I 3 OUTROTRES
I 3 MAISOUTROTRES
P
I 33 TRINTAETRES
I OITO 8
P
I A AGORAVAI
I B VAIMESMO
I X XIS
P
R X
R 3
R 2
I 3 DENOVO
R 3
P
Q
```

Exemplo de Saída:

```
0:[2/DOIS]->NULL  1:[3/TRES]->[23/VINTETRES]->NULL  2:NULL  3:[5/CINCO]->NULL  4:NULL  5:NULL
6:NULL  7:NULL  8:[13/TREZE]->NULL  9:NULL
0:[2/DOIS]->NULL  1:[3/TRES]->[23/VINTETRES]->[3/OUTROTRES]->[3/MAISOUTROTRES]->NULL  2:NULL
3:[5/CINCO]->NULL  4:NULL  5:NULL  6:[8/OITO]->NULL  7:NULL  8:[13/TREZE]->NULL  9:NULL
0:[2/DOIS]->NULL  1:[3/TRES]->[23/VINTETRES]->[3/OUTROTRES]->[3/MAISOUTROTRES]->[OITO/8]->NULL
2:NULL  3:[5/CINCO]->NULL  4:[33/TRINTAETRES]->NULL  5:NULL  6:[8/OITO]->NULL  7:NULL  8:
[13/TREZE]->NULL  9:NULL
0:[2/DOIS]->NULL  1:[3/TRES]->[23/VINTETRES]->[3/OUTROTRES]->[3/MAISOUTROTRES]->[OITO/8]->NULL
2:NULL  3:[5/CINCO]->NULL  4:[33/TRINTAETRES]->NULL  5:[A/AGORAVAI]->NULL  6:[8/OITO]->
[B/VAIMESMO]->NULL  7:NULL  8:[13/TREZE]->[X/XIS]->NULL  9:NULL
XIS
TRESOUTROTRESMAISOUTROTRES
DOIS
TRESOUTROTRESMAISOUTROTRESDENOVO
0:[2/DOIS]->NULL  1:[3/TRES]->[23/VINTETRES]->[3/OUTROTRES]->[3/MAISOUTROTRES]->[OITO/8]->
[3/DENOVO]->NULL  2:NULL  3:[5/CINCO]->NULL  4:[33/TRINTAETRES]->NULL  5:[A/AGORAVAI]->NULL  6:
[8/OITO]->[B/VAIMESMO]->NULL  7:NULL  8:[13/TREZE]->[X/XIS]->NULL  9:NULL
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo escolhido | Enviar Resposta

Questão 16: Hash com endereçamento aberto

Implemente uma tabela hash com tratamento de colisão por endereçamento aberto, utilizando código fornecido como pontapé inicial e a função principal não pode ser alterada. No mais, a implementação deve seguir a seguinte estratégia:

- A tabela hash é criada para uma determinada capacidade. A capacidade não pode ser alterada e tentativas de inserção além da capacidade falham.
- A tabela hash implementada (`hashEA`) não possui métodos de redimensionamento ou para alterar valores.
- Devem ser implementados métodos para busca, inserção, remoção. O método para depuração já encontra-se fornecido.
- Os dados armazenados são uma dupla [chave,valor], em que chave é uma string e valor um inteiro positivo. Para facilitar marcação, valor é representado internamente por `int`. Uma classe básica para os dados foi implementada, bem como operadores para comparação de igualdade e diferença. Sinta-se à vontade para alterar ou expandir a classe.
- A função para cálculo da posição original de um dado, sem ocorrência de colisão, é dada pelo método `posicao()` da classe hashEA e já encontra-se implementada. A posição real de um dado na tabela é dada pela função buscaChave(), que faz a busca considerando o tratamento de colisões por endereçamento aberto.
- Em caso de colisão, usa-se a posição consecutiva e assim sucessivamente. A posição que sucede a última é a primeira.
- As seguintes constantes (já declaradas) devem ser utilizadas para tornar o código mais claro:

```
const dado INVALIDO = {"",-1};
const dado REMOVIDO = {"",-2};
const int POSINVALIDA = -1;
```

- Em caso de colisão, usa-se a posição consecutiva e assim sucessivamente. A posição que sucede a última é a primeira.
  Elas representam, respectivamente, dados inválidos (posições não utilizadas na tabela), dados removidos e posições inválidas. Esta última é utilizada para indicar, por exemplo, que um determinado dado não se encontra na tabela.
- A busca de itens na tabela hash termina em caso de sucesso ou ao encontrar um espaço vazio (dado `INVALIDO`).

Com relação ao tratamento de erros, o código deve ser a prova de programadores e usuários descuidados. Mandar remover de estrutura vazia, por exemplo, não pode passar despercebido, mas a decisão sobre o que fazer não pode ser tomada na classe Hash. A função main é a responsável por determinar o que será feito. Em todos os casos de erro, a função main vai simplesmente escrever a mensagem de erro associada à exceção na saída padrão. Assim, a implementação dada deve usar manipulação de exceções, preferencialmente usando `throw runtime_error("`*`mensagem de erro`*`")`, gerando pelo menos os seguintes erros (recomendamos que você copie e cole as mensagens, para evitar erro de digitação):

- `"Chave inválida"`, em caso de ser passada uma chave vazia na inserção;
- `"Tabela hash cheia"`, em caso de não haver espaço disponível para nova inserção;
- `"Inserção de chave que já existe"`, em caso de tentativa de inserção de um novo dado com chave similar à uma já armazenada na tabela hash;
- `"Impossível remover de hash vazia"`, quando da tentativa de remoção em tabela hash vazia;
- `"Chave não encontrada para remoção"`, quando da tentativa de remoção de chave não existente na tabela hash;
- `"Chave não encontrada para consulta"`, quando da tentativa de consulta de dado não armazenado na tabela hash.

Entradas e Saídas:

A parte de interface do programa já está implementada. Inicialmente é lido um valor para a capacidade da tabela hash. Em seguida, o programa recebe comandos e os executa. Cada comando produz uma saída específica, se necessário. Os comandos são:

- A letra i, seguida de uma chave (palavra) e um valor (inteiro), para inserir uma chave/valor na estrutura.
- A letra r, seguida de uma chave (palavra) para remover uma chave/valor da estrutura.
- A letra c, seguida de uma chave (palavra) para consultar o valor de uma chave (o valor é escrito).
- A letra d, para depurar (escrever todas as informações armazenadas em formato específico.
- A letra f, para finalizar a execução do programa.

Exemplo de entrada:

```
10
r aaa
i aaa 0
i bbb 1
i ccc 2
d
i aaa 25
r ddd
r aaa
d
c cc
c ccc
i eee 3
i ggg 4
d
r bbb
i hhh 5
i a 1
i b 2
i c 3
i d 4
i e 5
i f 6
i g 7
d
f
```

Exemplo de saída:

```
Impossível remover de hash vazia
[0:] [1:bbb/1] [2:] [3:] [4:ccc/2] [5:] [6:] [7:] [8:aaa/0] [9:]
Inserção de chave que já existe
Chave não encontrada para remoção
[0:] [1:bbb/1] [2:] [3:] [4:ccc/2] [5:] [6:] [7:] [8:REMOVIDO] [9:]
Chave não encontrada para consulta
2
[0:eee/3] [1:bbb/1] [2:] [3:] [4:ccc/2] [5:] [6:ggg/4] [7:] [8:REMOVIDO] [9:]
Tabela hash cheia
[0:eee/3] [1:a/1] [2:b/2] [3:c/3] [4:ccc/2] [5:d/4] [6:ggg/4] [7:e/5] [8:f/6] [9:hhh/5]
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo escolhido Enviar Resposta