**Ministério da Educação**
**Universidade Tecnológica Federal do Paraná**
**Câmpus Pato Branco**


# 3ªAvaliação – AE22CP – 2017/1

**Instruções para a avaliação:** **prof.: Jean P. Martins (`jeanmartins@utfpr.edu.br`)**

- Leia a prova com atenção e coloque o seu nome completo na mesma.
- A avaliação é individual.
- A interpretação da prova faz parte da avaliação.
- É PROIBIDA a consulta à Internet.
- O peso de cada exercício está indicado ao final do enunciado.
- Todos os códigos deverão ser submetidos em: **https://www.hackerrank.com/ae22cp-prova3**

**NOME :** _____

**Definição de conjuntos**

Um conjunto é uma coleção de elementos distintos, ou seja, não pode existir em um mesmo conjunto dois elementos que representem o mesmo objeto. Portanto, considerando-se números inteiros, `A={2,2}` ***não é um conjunto válido.***

Uma implementação eficiente da estrutura de dados conjunto (`Set`) pode ser feita utilizando-se *Tabelas Hash*. Considere uma tabela hash implementada com **tratamento de colisões por encadeamento separado**. Uma estrutura `Set` pode ser definida da seguinte forma:

```
#define CAPACIDADE 1021 // Utilize esta capacidade
typedef struct {
      Lista* tabela_hash[CAPACIDADE];
} Set;
```

## 1. TABELAS HASH

### a) Tabelas hash: conjuntos (inserir) _____ **(Peso 2,0)**

Implemente uma função para inserção de elementos em um conjunto de inteiros.

```
int inserir(Set* s, long elemento);
```

***Formato de entrada***

O programa receberá como entrada, primeiramente o número de elementos a serem inseridos. Seguido pelos respectivos elementos. Exemplo:

```
7
13 8 6 5 1 4 4
```

***Restrições***

- Elementos repetidos não podem ser inseridos no conjunto.
- Elementos cujo hash colidam, devem ser inseridos **no início da lista correspondente**.

***Output Format***

```
1 4 5 6 8 13
```

- Assumindo que as colisões foram tratadas inserindo-se os elementos **no início da lista** e `CAPACIDADE=1021`, as sequências de saída estarão na mesma ordem que as minhas.
- **Há um espaço `' '` após cada número impresso, inclusive no final**

### b) Tabelas hash: conjuntos (união) _____ **(Peso 2,5)**

Implemente uma função que efetue a união de dois conjuntos e retorne o conjunto união.

```
Set* uniao(Set* A, Set* B);
```

***Formato de entrada***

O programa receberá como entrada, primeiramente o número de elementos a serem inseridos no primeiro conjunto. Seguido pelos respectivos elementos. O mesmo ocorrendo para o segundo conjunto. Exemplo:

```
7
13 8 6 5 1 4 4
8
195 41 45 101 191 11 102 4
```

***Restrições***

- Utilizar a função `inserir` para inserir os elementos lidos nos conjuntos.
- Utilizar a função `uniao` para gerar um terceiro conjunto, contendo a união de `A` e `B` e imprimí-lo.

***Output Format***

```
1 4 5 6 8 11 13 41 45 101 102 191 195
```

**c) Justificativa** **(Peso 1,5)**

Descreva um benefício das tabelas hash em relação à listas encadeadas no contexto da implementação das funções acima. Ou seja, por que tabelas hash nos possibilitam uma implementação mais eficiente da estrutura `Set`?

**2. PILHAS: Decimal para binário** **(Peso 2,0)**

Dado um número inteiro positivo, transforme este número em binário, utilizando uma pilha como estrutura auxiliar.

```
void decimalParaBinario(long num);
```

***Formato de entrada***

O programa receberá como entrada um inteiro positivo, que deve ser considerado como **long**

```
11
```

***Output Format***

```
1011
```

**2. MATRIZES ESPARSAS** **(Peso 2,0)**

Matrizes esparsas se caracterizam por possuírem grande parte de suas posições inutilizadas. Portanto, o uso de estruturas de dados específicas para sua armazenagem, possibilita considerável economia de memória. Descreva, brevemente, os principais elementos envolvidos na implementação de matrizes esparsas por **listas encadeadas**.