MC202—Estruturas de Dados
Lista de Exercícios 3

Islene Calciolari Garcia

Primeiro Semestre de 2007

1 Árvores 2-3 e árvores 2-3-4

```
  typedef struct no23{
  int nch;                         /* Número de chaves (1 ou 2) no nó */
  struct no23* esq;                /* Apontador esquerdo */
  int chesq;                       /* Chave esquerda     */
  struct no23* cen;                /* Apontador central  */
  int chdir;                       /* Chave direita      */
  struct no23* dir;                /* Apontador direito  */
} No23;
```

- Escreva uma função que libera todos os nós da árvore.
- Escreva uma função que conta o número de folhas na árvore.
- Escreva uma função que verifica para todos os nós, se todas as suas sub-árvores têm a mesma altura.
- Escreva uma função que verifica se a inserção de uma nova chave com valor V na árvore acarretará a criação de novos nós.
- Escreva uma função que verifica se a inserção de uma nova chave com valor V na árvore acarretará o aumento da altura da árvore.
- Escreva uma função que calcula o número médio de chaves por nó (número de chaves armazenadas/ número de nós na árvore). Esta função deve retornar zero se a árvore estiver vazia.

2 Grafos

- Veja os arquivos `digrafo-lista.c`, `digrafo-matriz.c` e `grafo-matriz.c` disponíveis em `http://www.ic.unicamp.br/ islene/mc202/aula24`. Implemente as funções `imprime_grafo`.

- O objetivo de uma busca em um grafo a partir de um nó v é visitar todos os nós alcançáveis a partir de v. Na *busca em profundidade*, um nó v é visitado e inicia-se uma busca em profundidade em um nó adjacente w (ainda não visitado). Este procedimento termina quando não existirem mais vértices não visitados adjacentes aos visitados. Na *busca em largura*, o nó v é visitado, depois dos os seus vizinhos, depois os vizinhos dos seus vizinhos (ainda não visitados), e assim por diante. Implemente estes dois algoritmos de busca. Que estrutura adicional é necessária para implementar o percurso em largura?

3 Hashing

1. Na tabela de espalhamento representada na figura abaixo, a função de *hash* é dada pelo resto da divisão do valor da chave pelo tamanho da tabela e a estratégia para resolução de colisões é **linear**.

Posição	Chave	Estado
0		Livre
1	6	Ocupado
2	1	Ocupado
3		Livre
4		Removido

   (a) Explique a importância para a operação de busca dos três estados: livre, ocupado e removido.

   (b) Indique o estado da tabela após a inserção das chaves 2 e 3 e posterior remoção da chave 6.

2. Implemente as seguintes operações para *hashing* com tratamento de colisões/overflow via encadeamento externo (listas ligadas).

   - cria_tabela
   - destroi_tabela
   - insere
   - remove
   - busca