1ª Chamada de Algoritmos e Estruturas de Dados I

Departamento de Informática

Universidade Nova de Lisboa

17 de Junho de 2003

Por favor, entregue as respostas em folhas distintas.

1. [2 valores] Suponha que se acrescenta a seguinte função à classe das árvores binárias de pesquisa (`BinarySearchTree`).

   int *height*( );
   // Retorna a altura da árvore.

   Implemente o método *height* e calcule a complexidade temporal do seu algoritmo, no melhor caso, no pior caso e no caso esperado, justificando-a.

2. [3 valores] Dada uma sequência de números inteiros, considere a permutação desses elementos que:

   - alterna, enquanto for possível, os números pares com os números ímpares;
   - começa com um número par (se existir algum); e
   - mantém sempre a ordem original, tanto dos números pares, como dos números ímpares.

   Para este efeito, assuma que zero é par. Por exemplo, se a sequência original for:

   - 2, 8, 5, 4, 0, 6, −3, 9, 4,
     a permutação pretendida é 2, 5, 8, −3, 4, 9, 0, 6, 4;
   - 1, −3, −7, −5, 0, a permutação pretendida é 0, 1, −3, −7, −5;
   - vazia, a permutação pretendida também é vazia.

   Suponha agora que, em termos de programação, ambas as sequências são geradas por iteradores (que devolvem objectos do tipo `Integer`). Programe a classe `AlternateBetweenEvenAndOddIterator` cujo construtor recebe um iterador da sequência original e devolve um iterador da permutação pretendida. Ignore o método *rewind*, ou seja, defina apenas as variáveis de instância e programe os três métodos seguintes.

   - *AlternateBetweenEvenAndOddIterator*(`Iterator` sequence);
   - **boolean** *hasNext*( );
   - `Object` *next*( );
     // O objecto retornado é do tipo `Integer`.

   Calcule a complexidade temporal dos seus três métodos, no melhor caso e no pior caso, justificando-a.

3. [3 valores] Pretende-se implementar uma fila com prioridade que permita decrementar de um dado valor a prioridade de todos os elementos que se encontram na fila.

   Assuma que os objectos do tipo `ObjectoComChaveInteira` respondem aos métodos

   - **int** *obtémChave*( );
     // Retorna a chave do objecto.
   - **void** *alteraChave*(**int** chave);
     // Altera a chave do objecto para a chave especificada.

   e que a prioridade do elemento na fila é a sua chave.

   Considere então o tipo abstracto de dados `FilaComPrioridade` de elementos do tipo `ObjectoComChaveInteira`, definido pelas seguintes operações.

   - `FilaComPrioridade` *cria*( );
     // Cria uma fila vazia.
   - **void** *destrói*( );
     // Destrói a fila.
   - **boolean** *vazia*( );
     // Retorna `true` sse a fila estiver vazia.
   - **boolean** *cheia*( );
     // Retorna `true` sse a fila estiver cheia.
   - `ObjectoComChaveInteira` *mínimo*( );
     // Retorna um elemento da fila com prioridade mínima.
     // **Pré-condição**: a fila não está vazia.
   - **void** *insere*(`ObjectoComChaveInteira` elemento);
     // Coloca o elemento especificado na fila.
     // **Pré-condição**: a fila não está cheia.
   - `ObjectoComChaveInteira` *removeMínimo*( );
     // Retira e retorna um elemento da fila com prioridade mínima.
     // **Pré-condição**: a fila não está vazia.
   - **void** *decrementaPrioridades*(**int** valor);
     // Decrementa a prioridade de todos os elementos da fila
     // do valor especificado.

   Indique as estruturas de dados mais adequadas para implementar este tipo abstracto de dados, descreva brevemente como implementaria as oito operações e calcule a sua complexidade temporal, no melhor caso, no pior caso e no caso esperado, justificando-a.

4. [3 valores] Suponha que se inserem sete elementos numa árvore AVL vazia, cujas chaves são os números inteiros 1, 2, 3, 4, 5, 6 e 7. Especifique uma ordem pela qual se podem inserir os elementos que minimize o número total de rotações efectuadas nas sete inserções. De acordo com essa ordem, desenhe o estado da árvore após cada inserção, assinalando onde são efectuadas as rotações.

5. [4 valores] Um *anagrama* de uma palavra P é uma palavra que tem exactamente as mesmas letras que P, por uma ordem diferente, ignorando as diferenças entre maiúsculas e minúsculas. Por exemplo, na Língua Portuguesa, `amor` é um anagrama de `ramo`. Se as palavras $P_1, P_2, \ldots, P_n$ forem todas anagramas umas das outras, diz-se que $\{P_1,\ P_2,\ \ldots,\ P_n\}$ é um *conjunto de anagramas de cardinalidade* n (com $n \geq 2$). Portanto, {`amor`, `mora`, `ramo`, `Roma`} é um conjunto de anagramas de cardinalidade 4.

   Por outro lado, relembre que um *gene* é um fragmento de ADN que pode ser caracterizado por uma sequência não vazia de nucleótidos básicos: adenina (que se representa por `A`), timina (`T`), guanina (`G`) e citosina (`C`). Ora, em alguns ramos da Engenharia Genética, é importante identificar os conjuntos de anagramas que se encontram em grandes sequências de genes.

   Mais precisamente, pretende-se um programa que, dada uma sequência de genes, possivelmente com repetições, identifique todos os conjuntos de anagramas de maior cardinalidade.

   Por exemplo, se a sequência de genes fosse:

   ```
   ATATA   TATA   CCCA   TATA   TTAA   TTAA   TATA
   CCCA   CCAC   ACCC   ATATA   ATG   TGA   AGT
   TATA   TATA   CCAC
   ```

   os conjuntos de anagramas de maior cardinalidade seriam (por qualquer ordem):

   - {`CCCA`, `CCAC`, `ACCC`} e
   - {`ATG`, `TGA`, `AGT`}.

   Sabe-se que o comprimento das sequências de nucleótidos básicos que descrevem os genes não excede 20 e que o comprimento da sequência de genes não ultrapassa 100 000. (No exemplo, o comprimento das sequências de nucleótidos básicos não excede 5 e o comprimento da sequência de genes é 17.)

   Explicite detalhadamente as estruturas de dados mais adequadas para resolver este problema, descreva sumariamente o algoritmo e calcule a sua complexidade, no caso esperado. Para simplificar, pode assumir que há sempre, pelo menos, um conjunto de anagramas na sequência de genes.