Funções Finitas I: árvores binárias

Métodos de Programação II

2004/2005

1 Objectivos

Construir uma pequena biblioteca em C para *Árvores Binárias de Procura*. Sobre essa biblioteca deverá ser construída uma biblioteca de *Funções Finitas* (também designadas por *Dicionários* ou *Mappings*).

2 Motivação

> Pretende-se desenvolver um programa que produza um histograma para os tamanhos das palavras de um texto (lido do stdin).

O desenvolvimento do programa proposto leva-nos rapidamente à utilização do ADT de *Funções Finitas*. Esse tipo de dados permite armazanar/aceder a um número finito de pares `(chave, valor)`, sendo que sobre estes pares existe uma dependência funcional: uma chave está associada a, quanto muito, um valor. As operações que caracterizam este ADT são:

put – que adiciona um novo par (chave, valor) à função finita (ou actualiza o valor associado a uma chave, no caso de esta já existir);

get – que retorna o valor associado a uma chave numa função finita.

Uma implementação óbvia de funções finitas é por intermédio de listas de "pares"(i.e. listas em que a informação dos nós consiste na chave e no valor respectivo). No entanto, é importante garantir a eficiência das operações de "inserção"e de "consulta", pelo que se é levado a procurar alternativas de representação que tornem essas operações mais eficientes.

As "Árvores Binárias de Procura"são aqui uma alternativa interessante já que permitem definir operações de inserção e de remoção eficientes (pelo menos quando a forma da árvore não é degenerada... um assunto para a próxima aula).

3 Árvores Binárias de Procura

As árvores binárias de procura são caracterizadas pelo seguinte invariante:

Uma árvore binára é uma *Árvore Binára de Procura* quando:

- for uma árvore vazia;
- for uma árvore não vazia em que o valor armazenado na raíz é superior a todos os valores armazenados na sub-árvore esquerda e inferior a todos os valor armazenados na sub-árvore direita e ambas as sub-árvores são elas próprias árvores binárias de procura.

Nas árvores binárias de procura podemos definir a operação pesquisa de forma eficiente: a comparação do elemento a pesquisar com o valor armazenado na raíz dá-nos informação sobre em que sub-árvore poderemos encontrar esse elemento.

Uma possibilidade para a *header file* do módulo:

```
/*
   BinTree.h
   =========

   Modulo de árvores binárias (com chaves inteiras)
*/

typedef int KeyType;

typedef struct treeEntry{
  KeyType key;
  void *value;
} TreeEntry;

typedef struct treeNode{
  TreeEntry entry;
  struct treeNode *left;
  struct treeNode *right;
} TreeNode;

typedef TreeNode *Tree;

// cria uma árvore vazia
Tree treeCreate(void);

// insere um novo elemento numa árvore binária de procura.
// (obs.: substitui valor se chave já existir)
Tree treeInsert(Tree, KeyType, void *);

// procura um elemento numa árvore binária de procura
void *treeSearch(Tree, KeyType);

// travessias
void treePreorder(Tree,void (*fun)(KeyType, void*));
```

```
void treeInorder(Tree,void (*fun)(KeyType, void*));
void treePostorder(Tree,void (*fun)(KeyType, void*));
```

3.1 Outras funções úteis

Faz sentido incluir na biblioteca algumas funções adicionais:

```
// testa se a árvore é vazia
int treeIsEmpty(Tree);

// conta o número de nodos da árvore
int treeSize(Tree);

// calcula a altura da árvore
int treeHeight(Tree);

// testa se a árvore binária está completa
int treeIsComplete(Tree);

// apaga toda árvore
void treeClear(Tree *);

// procedimento que cria uma cópia da árvore binária
void treeCopy(Tree,Tree *);

// remove um elemento (com uma dada chave) de uma árvore binária de procura.
void treeDeleteKey(Tree *, KeyType);
```

3.2 A fazer...

Implementar as funções declaradas nas *header-files* apresentadas.

4 Funções Finitas

A implementação de um módulo de funções finitas sobre o de árvores bináras é trivial: é suficiente instanciar as funções do primeiro módulo nas funções apropriadas do segundo. A título de exemplo, poderemos definir as funções básicas como:

```
/*
   TreeMapInt.h
   ============

   Módulo de "Funções Finitas" (maps) implementados sobre árvores binárias
   de procura com chaves inteiras.
*/
```

```
#include "BinTree.h"

typedef Tree TreeMapInt;

/*
  Cria um novo Map
*/
TreeMapInt mapCreate(void)
{
  return treeCreate();
}


/*
  Adiciona uma associação "(chave, valor)" a um Map.
  (obs.: substitui associação se a chave já pertencer ao domínio do Map)
*/
void mapPut(TreeMapInt* map, KeyType key, void *value)
{
  *map = treeInsert(*map, key, value);
}


/*
  Retorna valor associado a uma chave (NULL se a chave não pertencer ao
  domínio).
*/
void *mapGet(TreeMapInt map, KeyType key)
{
  return treeSearch(map, key);
}



/*
  Itera a aplicação da função "procEntry" por todos os pares "(chave,valor)"
  contidos no Map.
*/
void mapIter(TreeMapInt map, void (*procEntry)(KeyType, void*))
{
  treeInorder(map, procEntry);
}
```

5 O Programa Principal

O programa proposto inicialmente pode agora ser codificado sem dificuldades de maior:

```
/*
```

```
   hist.c
   ======

*/

#include <stdio.h>
#include <ctype.h>

#include "TreeMapInt.h"

#define TRUE 1
#define FALSE 0

void showEntry(KeyType key, void* value);

int main()
{
  TreeMapInt map;
  char c;       // carácter lido

  int fOnWord;  // flag "sobre palavra"
  int count;    // contador de caracteres na palavra

  int *valPtr;  // valor armazenado no Map

  // Inicializações
  map = mapCreate();
  fOnWord = FALSE;
  count = 0;

  // Leitura do ficheiro de entrada
  while ( (c=getchar()) != EOF) {
    if ( isspace(c) ) {
      if (fOnWord) {
        // Consulta valor associado à chave
        valPtr = mapGet(map, count);

        // Se entrada para a chave não existe, cria-se nova...
        if (!valPtr) {
          valPtr = (int*) malloc(sizeof(int));
          *valPtr = 0;
          mapPut(&map, count, valPtr);
        }

        // Actualiza valor associado à chave
        (*valPtr)++;

        fOnWord = FALSE;
        count = 0;
      }
```

```
    } else {
      count ++;
      fOnWord = TRUE;
    }
  }

  printf("APRESENTAÇÃO DOS RESULTADOS:\n============================\n");
  mapIter(map, showEntry);
}

void showEntry(KeyType key, void* value)
{
  printf("%d -> %d\n",key, *(int*)value);
}
```