Estructures de la Informació. Control parcial

Dept. de Ciències de la Computació

E.P.S.E.V.G., 19 de abril de 2013, 15:00-17:00

IMPORTANT: Resol els problemes en fulls separats.

1. (2 punts. 20 minuts) Eficiència algorísmica

   L'algorisme d'ordenació OBK (ordenació boja d'en Kim) ordena un array T[i..j] recursivament de la següent manera:

```
void obk (int T[], int i, int j) {
  if (i >= j) return; // ordenar 0 ó 1 element no cal fer res
  if (i + 1 == j) {   // ordenar 2 elements
    if (T[i] > T[j]) {
      intercanviar(T[i], T[j]); // cost constant
    }
  }
  else {  // ordenar tres o més elements
    int n = j - i + 1;
    int m1 = i + (n / 3);
    int m2 = j - (n / 3);
    obk(T, i, m2);
    obk(T, m1, j);
    obk(T, i, m2);
  }
}
```

   a) Calcula el cost en funció del nombre *n* d'elements a ordenar. Si és necessari, pots utilitzar els teoremes adjunts.

   b) Convé fer servir aquest algorisme en comptes de l'algorisme d'ordenació per selecció (que té cost $\Theta(n^2)$ en el cas pitjor)?

   IMPORTANT: Raona la teva resposta.

Decreixement aritmètic

Teorema. *Sigui T(n) el cost d'un algorisme recursiu descrit per la recurrència*

$$T(n)=\begin{cases} f(n) & si\ 0\leqslant n<n_0 \\ a\cdot T(n-c)+g(n) & si\ n_0\leqslant n \end{cases}$$

on n_0 és una constant, $c\geqslant 1$, $f(n)$ és una funció arbitrària i $g(n)=\Theta\left(n^k\right)$ amb *k* constant. Llavors,

$$T(n)=\begin{cases} \Theta\left(n^k\right) & si\ a<1 \\ \Theta\left(n^{k+1}\right) & si\ a=1 \\ \Theta\left(a^{n/c}\right) & si\ a>1 \end{cases}$$

Decreixement geomètric

Teorema. *Sigui T(n) el cost d'un algorisme recursiu descrit per la recurrència*

$$T(n)=\begin{cases} f(n) & si\ 0\leqslant n<n_0 \\ a\cdot T(n/b)+g(n) & si\ n_0\leqslant n \end{cases}$$

on n_0 és una constant, $b>1$, $f(n)$ és una funció arbitrària i $g(n)=\Theta\left(n^k\right)$ amb *k* constant. Llavors,

$$T(n)=\begin{cases} \Theta\left(n^k\right) & si\ a<b^k \\ \Theta\left(n^k\log n\right) & si\ a=b^k \\ \Theta\left(n^{\log_b a}\right) & si\ a>b^k \end{cases}$$

2. (5 punts. 60 minuts) **Estructures Lineals**

Donada la següent classe llista:

```
template <typename T>
class llista {
   ...
   void stutter(int pas, int reps) throw(error);
   ...
 private:
   // Llista simplement enllaçada, no circular i sense element fantasma.
   struct node {
     T info;
     node *seg;
   };
   node* _head;
   int _size;
};
```

Implementa en C++ el mètode `stutter` de l'anterior classe llista. Aquest mètode modifica la llista sobre la qual s'aplica de manera que els elements a les posicions *pas*, *2 x pas*, *3 x pas*, *4 x pas*, ..., es repeteixen *reps cops*.

Per exemple, donada la llista d'strings:

`L={"digues", "que", "vols", "fer", "demà", "a", "la", "tarda"}`

llavors després de cridar al mètode `L.stutter(3,2)` la llista passaria a ser com

`L={"digues", "que", "vols", "vols", "fer", "demà", "a", "a", "la", "tarda"}`

NOTA 1: Cal que implementis els mètodes addicionals que utilitzis explícitament.

NOTA 2: Pots suposar que la classe T disposa del constructor per còpia i de l'operador d'assignació.

3. (3 punts. 40 minuts) **Arbres**

Implementa en C++ una funció que donat un arbre binari retorni un vector amb el número de nodes a cada nivell:

```
template <typename T>
void nodes_per_nivell (const Abin<T> &a, vector<int> &L) throw(error);
```

Disposes de les classes `Abin<T>` i `Abin<T>::iterador` amb els mètodes públics vistos a classe i llistats a continuació.

NOTA 1: Cal que implementis les funcions addicionals que utilitzis.

NOTA 2: Pots suposar que el vector L està inicialment buit i pots fer servir les operacions tals com *push_back*, *push_front*, *size*, etc.

```
template <class T>
class Abin {
 public:
   Abin() throw(error);

   Abin(const Abin<T> &a) throw(error);
   Abin& operator=(const Abin<T> &a) throw(error);
   ~Abin() throw();

   Abin(Abin<T> &fesq, const T &x, Abin<T> &fdret) throw(error);

   bool es_buit() const throw();
```

Iterador sobre arbres binaris.

```
   friend class iterador {
   public:
     friend class Abin;

     iterador() throw();

     Abin<T> arbre() const throw(error);
```

Retorna l'element apuntat per l'iterador.

```
     const T& operator*() const throw(error);

     iterador fesq() const throw(error);
     iterador fdret() const throw(error);

     bool operator==(const iterator &it) const throw();
     bool operator!=(const iterator &it) const throw();

     static const int IteradorInvalid = 410;
   };

   iterador arrel() const throw();
   iterador final() const throw()
};
```

Solució problema 1

a) Sigui T(n) el cost d'aquesta funció. Si n>=3, es fan unes poques operacions de cost constant i tres crides recursives, cadascuna de les quals rep aproximadament 2n/3 d'elements.

Per tant

$$T(n)=\begin{cases} ctt & si\ 0\leqslant n<3 \\ a\cdot T(n/b)+\Theta(n^k) & si\ 3\leqslant n \end{cases}$$

on *a*=3, *b*=3/2, *k*=0.

Llavors es compleix que $a>b^k$ i, per tant, el cost temporal és
T(n) = $\Theta\left(n^{\log_b a}\right)=\Theta\left(n^{\log_{3/2}3}\right)$

b) Per altra banda $(3/2)^2 = 9/4 < 3$ i $(3/2)^3 = 27/8 > 3$, deduïm que $2<\log_{3/2}3<3$, i per tant l'algorisme `obk` és més ineficient que l'algorisme d'ordenació per selecció, que és quadràtic. De fet $\log_{3/2}3=2,7$.

Solució problema 2

```
template <typename T>
void llista<T>::stutter(int pas, int reps) throw(error) {
  if (pas > 0 and reps > 0) {
    rstutter(_head, 1, pas, reps);
  }
}

// Mètode privat de classe. Caldria declarar-lo com a static dins la part
// privada de l'especificació de la classe (.hpp).
template <typename T>
void llista<T>::rstutter(node *n, int act, int pas, int reps) throw(error) {
  // Pre: act > 0, pas > 0 i reps > 0
  if (n != NULL) {
    node *n2 = n;
    if (act %  pas == 0) {
      n2 = duplica(n, reps);
    }
    try {
      rstutter(n2->seg, act+1, pas, reps);
    }
    catch (error) {
      if (act % pas == 0) {
        while (n->seg != n2) {
          node *aux = n->seg;
          n->seg = n->seg->seg;
          delete aux;
        }
      }
      throw;
    }
  }
}

// Mètode privat de classe. Caldria declarar-lo com a static dins la part
// privada de l'especificació de la classe (.hpp).
template <typename T>
typename llista<T>::node* llista<T>::duplica(node* n, int reps) throw(error) {
  // Pre: n != NULL and reps>0
  // Post: retorna l'últim node duplicat
  node *ret;
  if (reps > 1) {
    node *aux = new node;
    try {
      aux->info = n->info;
      aux->seg = n->seg;
      ret = duplica(aux, reps-1);
      n->seg = aux;
    }
    catch (error) {
      delete aux;
      throw;
    }
  }
  else {
    ret = n;
  }
  return ret;
}
```

Solució problema 3

```
template <typename T>
void nodes_per_nivell (const Abin<T> &a, vector<int> &L) throw(error) {
  return rnodes_per_nivell(a.arrel(), a.final(), 1, L);
}


template <typename T>
void rnodes_per_nivell (const Abin<T>::iterador &it,
                        const Abin<T>::iterador &end,
                        int nivell,
                        vector<int> &L) throw(error) {
  if (it != end) {
    if (L.size() < nivell) {
      L.push_back(1);
    }
    else {
      ++L[nivell-1];
    }
    rnodes_per_nivell (it.fesq(), end, nivell+1, L);
    rnodes_per_nivell (it.fdret(), end, nivell+1, L);
  }
}
```