Estructures de la Informació. Control parcial

Dept. de Ciències de la Computació
E.P.S.E.V.G., 28 de novembre de 2016, 12:30-14:30

IMPORTANT: Resol els problemes en fulls separats.

1. (2 punts. 20 minuts) **Eficiència algorísmica**

   La següent funció permet pintar un arbre de *n* nodes:

```
void pinta_arbre (int k, int n, bool hor, int x0, int y0) {
  if (n >= 1) {
    pinta_node (x0 + n /2, y0 + n /2);
    if (n != 1) {
      if ( hor ) {
        pinta_linia (x0 + n /2, y0 + n /2, x0 + n /4, y0 + n /2);
        pinta_linia (x0 + n /2, y0 + n /2, x0 + 3* n /4, y0 + n /2);
        pinta_linia (x0 + n /4, y0 + n /2, x0 + n /4, y0 + 3* n /4);
        pinta_linia (x0 + n /4, y0 + n /2, x0 + n /4, y0 + n /4);
        pinta_linia (x0 + 3* n /4, y0 + n /2, x0 + 3* n /4, y0 + 3* n /4);
        pinta_linia (x0 + 3* n /4, y0 + n /2, x0 + n /4, y0 + n /4);
      }
      else {
        // sis crides més a pinta_linia (...)
      }
      pinta_arbre (k - 1, (n -1) / 2, not hor, x0, y0);
      pinta_arbre (k - 1, (n -1) / 2, not hor, x0 + n /2, y0);
      pinta_arbre (k - 1, (n -1) / 2, not hor, x0, y0 + n /2);
      pinta_arbre (k - 1, (n -1) / 2, not hor, x0 + n /2, y0 + n /2);
    }
  }
}
```

   El cost de les accions `pinta_node` i `pinta_linia` és constant. Quin és el cost asimptòtic temporal de l'acció `pinta_arbre` en funció de *n*? 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. (4 punts. 55 minuts) **Estructures Lineals**

   Donada una classe llista d'enters implementada amb nodes doblement encadenats sense element fantasma i circular. Fes el següent:

   a) Escriu la representació d'aquesta classe.

   b) Implementa el mètode `duplica_o_esborra`, que fa que cada nombre primer aparegui dues vegades i esborra els nombres compostos.

      Un *nombre primer* és un nombre enter superior a 1 que només té dos divisors: ell mateix i l'1. Un *nombre compost* és un nombre natural que té més de dos divisors.

      Per exemple, si:

      L = [11, 32, -27, 1, 0, 60, 55, 11, 23, 11]

      llavors la crida a aquest mètode generaria:

      L = [11, 11, -27, 1, 0, 23, 23]

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

3. (4 punts. 45 minuts) **Diccionaris**

   Donada aquesta classe `dicc` implementada com un arbre binari de cerca, la representació de la qual és la següent:

```
template <typename C, typename V>
class dicc {
  ...
  private:
    struct node {
      node *fesq;
      node *fdret;
      C clau;
      V valor;
    };
    node* _arrel;
    ...
};
```

   fes un mètode d'aquesta classe que donats dos dicc indiqui si els dos diccionaris tenen exactament les mateixes claus.

   NOTA 1: Suposa que el tipus C disposa dels operadors de comparació.

   NOTA 2: Cal que implementis els mètodes addicionals que utilitzis explícitament.

Solució problema 1

Càlcul del cost temporal de l'acció `pinta_arbre()`:

Aquesta acció és una acció recursiva l'equació de la recurrència de la qual és la següent:

$$T(n)=\begin{cases} ctt & si\ 0\leqslant n<1 \\ a\cdot T(n/b)+\Theta(n^k) & si\ 1\leqslant n \end{cases}$$

on *a*=4 ja que hi ha quatre crides recursives dins la mateixa acció; *b*=2 ja que la disminució de les dades és per la meitat; *k*=0 ja que els cost de les altres operacions que hi ha dins d'aquesta acció són constants. Per tant, l'equació de la recurrència és:

$$T(n)=\begin{cases} ctt & si\ 0\leqslant n<1 \\ 4\cdot T(n/2)+\Theta(n^0) & si\ 1\leqslant n \end{cases}$$

Per resoldre el cost de `pinta_arbre()` es pot usar el teorema mestre de decreixement geomètric (ja que l'equació de recurrència s'adapta correctament a aquest teorema mestre).

Llavors es compleix que $a>b^k$ ($4 > 2^0$) i, per tant, el cost temporal de `pinta_arbre()` és:

$$\Theta(n^{\log_b a})=\Theta(n^{\log_2 4})=\Theta(n^2)$$

Solució problema 2

a)

```
struct node {
  int info;
  node *seg;
  node *ant;
};

node *_head;   // apunta al primer element de la llista
```

b)

```
// mètode privat de classe
bool llista::es_primer(int n) throw() {
  unsigned int i = 0;
  bool trobat = n < 2;
  while (i*i <= n and not trobat) {
    if (n % i == 0) trobat = true;
    else ++i;
  }
  return not trobat;
}

// mètode privat de classe
bool llista::es_compost(int n) throw() {
  return n>3 and not es_primer(n);
}

// mètode privat
void llista::esborra(node &*n) throw() {
  if (n != _head) {
    node *aux = n;
    n->ant->seg = n->seg;
    n->seg->ant = n->ant;
    n = n->seg;
    delete aux;
  }
  else {
    if (_head->seg == _head) {
      delete n;
      _head = NULL;
      n = NULL;
    }
    else {
      node *aux = n;
      n->ant->seg = n->seg;
      n->seg->ant = n->ant;
      n = n->seg;
      delete aux;
      _head = n;
    }
  }
}
```

```
void llista::duplica_o_esborra() throw(error) {
  node *fi = (_head != NULL) ? _head->ant : NULL;
  node *n = _head;
  while (n != NULL and n != fi) {
    if (es_primer(n->info)) {
      duplica(n);
      n = n->seg;
    }
    else if (es_compost(n->info)) {
      esborra(n);
    }
    else {
      n = n->seg;
    }
  }
  // tractament del primer element de la llista
  if (n != NULL) {
    if (es_primer(n->info)) {
      duplica(n);
    }
    else if (es_compost(n->info)) {
      esborra(n);
    }
  }
}


// mètode privat de classe
void llista::duplica(node* n) throw(error) {
  // inserim el node duplicat abans de n
  node *nou = new node;
  nou->info = n->info;
  nou->ant = n->ant;
  nou->seg = n;
  n->ant->seg = nou;

  // esborrem la resta de nodes que tenen la mateixa info que n
  node *p = n->seg;
  while (p != nou) {
    if (p->info == n->info) {
      esborra(p);
    }
    else {
      p = p->seg;
    }
  }
}
```

Solució problema 3

```
template <typename C, typename V>
bool dicc<C,V>::mateixes_claus (const dicc &d) const throw(error) {
  return mateixes_claus(_arrel, d._arrel) and mida() == d.mida();
}

template <typename C, typename V>
bool dicc<C,V>::mateixes_claus (node *p, node* q) const throw(error) {
  bool res = (p == NULL) and (q == NULL);
  if (p != NULL) {
    res = conte(p->clau, q) and mateixes_claus(p->fesq, q) and
          mateixes_claus(p->fdret, q);
  }
  return res;
}

template <typename C, typename V>
bool dicc<C,V>::conte (const C &clau, node *on) const throw(error) {
  bool res = false;
  if (on != NULL) {
    if (clau == on->clau) {
      res = true;
    }
    else if (clau > on->clau) {
      res = conte(clau, on->fdret);
    }
    else {
      res = conte(clau, on->fesq);
    }
  }
  return res;
}


// Per tal de fer aquesta solució més eficient, es podria afegir un atribut
// _mida en la representació de la classe que s'incrementés cada cop que inserim
// un nou parell <clau, valor> en el diccionari (i es decrementés quan se
// n'esborri un). D'aquesta manera el nombre d'elements del diccionari estaria
// calculat en tot moment.
template <typename C, typename V>
bool dicc<C,V>::mida () const throw(error) {
  return mida(_arrel);
}

template <typename C, typename V>
bool dicc<C,V>::mida (node *n) throw(error) {
  int res = 0;
  if (n != NULL) {
    res = mida(n->fesq) + mida(n->fdret) + 1;
  }
  return res;
}
```