SOLUCIÓ MODEL DE L'EXAMEN FINAL
PROGRAMACIÓ 2, FIB
JUNY DE 2014

Problema 1:

Una solució usant push_back:

```
void trams(vector<Llista<T> >& v) {
/* Pre: v és buit */
/* Post: el p.i. és buit i v conté els trams de v */

  node *n = primer;
  node *nant;
  int i = -1;

  while (n != NULL) {
    // Inv: n apunta a un node del p.i.;
    // si n != primer, nant apunta a l'element anterior a
    // l'apuntat per n; v té mida i+1; v[0..i] conté llistes
    // ben formades corresponents als trams del p.i. original
    // fins al node anterior a l'apuntat per n
    if (n == primer or nant ->info > n->info) {
       // tancar llista actual
       if (n != primer) n->ant->seg = NULL;
       // obrir llista nova
       v.push_back(Llista<T>());
       ++i;
       v[i].primer = v[i].ultim = v[i].act = n;
       v[i].longitud = 1;
       n->ant = NULL;
    }
    else {
      v[i].ultim = n;
      ++v[i].longitud;
    }
    nant = n;
    n = n->seg;
  }

  primer = ultim = act = NULL;
  longitud = 0;

}
```

Sense push_back(), es pot calcular primer el nombre de trams:

```
  // un primer recorregut per calcular el nombre de trams
  int ntrams = 0;
  if (n != NULL) {
    ++ntrams;
    while (n->seg != NULL) {
      if (n->info > n->seg->info) ++ntrams;
      n = n->seg;
    }
  }
  v = vector<Llista<T> >(ntrams,Llista<T>());
```

però aquesta solució farà el doble de comparacions entre T's, una a cada recorregut.

Altres solucions acceptades:

- En una primera passada, crear un vector d'int's, booleans o
  apuntadors per recordar els inicis de cada tram. En la segona,
  es fragmenta la llista sense comparacions de T's.

- (més difícil) En un recorregut, es fragmenta la llista i se
  n'encadenen els trams utilitzant algun apuntador que queda sense
usar,
  com ara l'ant de cada inici de tram. En un segon recorregut, sense
  comparacions entre T's, es transfereixen aquests trams encadenats a
  les entrades del vector. No usa memòria auxiliar i només fa tantes
  comparacions de T com elements a la llista.

Problema 2:

Una solució possible és:

```
void arbre_max_min(ArbreGen<T> &amax, ArbreGen<T> &amin) const
  /* Pre: el p.i. no es buit, amax i amin son buits */
  /* Post: amax i amin són els arbres de màxims i de mínims del p.i.
respectivament */
{
  T vmax,vmin;

arbre_max_min_aux(primer_node,amax.primer_node,amin.primer_node,vmax,v
min);
}

static void arbre_max_min_aux(node_arbreGen* m, node_arbreGen* &
n_max, node_arbreGen* & n_min, T & vmax, T & vmin)
  /* Pre: m no es NULL */
  /* Post: n_max apunta a una jerarquia de nodes que conte els valors
     maxims de la jerarquia de nodes a la que apunta m,
     n_min apunta a una jerarquia de nodes que conte els valors minims
de
     la jerarquia de nodes a la que apunta m,
     v_max es el valor maxim a la jerarquia de nodes a la que apunta
m,
     v_min es el valor minim a la jerarquia de nodes a la que apunta m
*/
{
  n_max=new node_arbreGen;
  n_min=new node_arbreGen;
  int n=m->seg.size();
  if(n==0){
    vmax=vmin=m->info;
    n_max->info=vmax;
    n_min->info=vmin;
  }
  else{
    T vmax_aux,vmin_aux;
    vmax=vmin=m->info;
    n_max->seg=vector<node_arbreGen*>(n);
    n_min->seg=vector<node_arbreGen*>(n);
    for(int i=0;i<n;++i){
      arbre_max_min_aux(m->seg[i],
              n_max->seg[i],n_min->seg[i],vmax_aux,vmin_aux);
      if(vmax_aux>vmax) vmax=vmax_aux;
```

```
      if(vmin_aux<vmin) vmin=vmin_aux;
    }
    n_max->info=vmax;
    n_min->info=vmin;
  }
}
```

Una altra solució (sense immersió del max i el min: fa servir que els nodes dels resultats contenen el max i el min de la corresponent jerarquia
original):

```
void arbre_max_min_bis(ArbreGen<T> &amax, ArbreGen<T> &amin) const
/* Pre: el p.i. no es buit, amax i amin son buits */
/* Post: amax es l'arbre de maxims del p.i, amin es l'arbre de minims
del p.i. */
{
        arbre_max_min_aux(primer_node,amax.primer_node,amin.primer_node
);
}

static void arbre_max_min_aux(node_arbreGen* m, node_arbreGen* &
n_max, node_arbreGen* & n_min)
  /* Pre: m no es NULL */
  /* Post: n_max apunta a una jerarquia de nodes que conte els valors
     maxims de la jerarquia de nodes a la que apunta m,
     n_min apunta a una jerarquia de nodes que conte els valors minims
de
     la jerarquia de nodes a la que apunta m */

{
        n_max=new node_arbreGen;
        n_min=new node_arbreGen;
        int n=m->seg.size();
        if(n==0){
                 n_max->info=n_min->info=m->info;
        }
        else{
                n_max->info=n_min->info=m->info;
                n_max->seg=vector<node_arbreGen*>(n);
                n_min->seg=vector<node_arbreGen*>(n);
                for(int i=0;i<n;++i){
                        arbre_max_min_aux(m->seg[i],n_max-
>seg[i],n_min->seg[i]);
                        if(n_max->seg[i]->info>n_max->info) n_max-
>info=n_max->seg[i]->info;
                        if(n_min->seg[i]->info<n_min->info) n_min-
>info=n_min->seg[i]->info;
                }
        }
}
```

Altres variants:

- En comptes de crear els nous nodes al principi de la funció, es poden crear abans de cada crida (2 crides a new en arbre_max_min, i 2 crides a new dins del for). És un xic més llarga però igual d'eficient. En aquesta solució, era correcte passar per valor (sense &) els paràmetres n_max i n_min, però s'ha de dir a la pre que no són
  NULL. No és tan fàcilment aplicable a la resta d'arbres i el codi resulta més difícil d'entendre.