FINAL 2016-2017 Q2

PROBLEMA 2

```
// Existen soluciones mÃ¡s eficientes, pero esta soluciÃ³n o una soluciÃ³n
// correcta de coste temporal y espacial similar es suficiente para
// obtener 5 puntos.

/* Pre: ruta = RUTA y RUTA es una pila vacÃa.

   Post: El primer componente del resultado es cierto si existe un
   nodo en la jerarquÃa de nodos a cuya raÃz apunta 'primer' con campo
   info.ciudad igual a omega; en otro caso, el primer componente del
   resultado es falso. Si el primer componente del resultado es
   cierto, el segundo componente del resultado contiene el valor del
   campo info.kms_recorridos del nodo 'n_min' de la jerarquÃa de nodos
   a cuya raÃz apunta 'primer' con campo info.ciudad igual a omega y
   valor del campo info.kms_recorridos mÃnimo. En caso de empate,
   'n_min' es el primer nodo con estas caracterÃsticas de la jerarquÃa
   de nodos a cuya raÃz apunta 'primer' recorrida en pre-orden. AdemÃ¡s,
   si el primer componente del resultado es cierto, el parÃ¡metro 'ruta'
   contiene la informaciÃ³n de los nodos que forman el camino que
   conecta el nodo al que apunta 'primer' con el nodo 'n_min' de
   manera que el orden de extracciÃ³n de los elementos de la pila
   'ruta' coincida con el orden de los nodos de dicho camino recorrido
   desde el nodo al que apunta 'primer' hasta 'n_min'.
*/

static pair<bool,int> iti_aux(Node* primer, string omega, stack<Etapa>& ruta) {
  pair<bool,int> res;
  if (primer == NULL) res.first = false;
  else if (primer->info.ciudad == omega) {
     res.first = true;
     res.second = primer->info.kms_recorridos;
     ruta.push(primer->info);
  }
  else {
     res.first = false;
     int ari = primer->seg.size();
     for (int i = 0; i < ari; ++i) {
        stack<Etapa> ruta_i;
        pair<bool,int> res_i = iti_aux(primer->seg[i],omega,ruta_i);
        if (res_i.first) {
           if (not res.first or res.second > res_i.second) {
              res = res_i;
              ruta.swap(ruta_i);
           }
        }
     }
     if (res.first) ruta.push(primer->info);
  }
  return res;
}




pair<bool,int> itinerario(string omega, stack<Etapa>& ruta) const {
  return iti_aux(primer_node,omega,ruta);
}


// La soluciÃ³n al apartado 2.1 se obtiene eliminado las llamadas a
```

```
// ruta.push(primer->info), la llamada a ruta.swap(ruta_i), y la
// declaraciÃ³n stack<Etapa> ruta_i.

// En el apartado 2.1 la cabecera de la operaciÃ³n auxiliar es
// static pair<bool,int> iti_aux(Node* primer, string omega)
// y las llamadas recursivas iti_aux(primer->seg[i],omega). AdemÃ¡s,
// la definiciÃ³n de la operaciÃ³n principal es

// pair<bool,int> distancia(string omega) const {
//  return iti_aux(primer_node,omega);
// }
```