Gina Escofet González, grup 13

JUSTIFICACIÓ DE CODI

1. Mètode iteratiu per pujar bicis

```
1  void Cjt_estaciones::completar_aux(const BinTree<string>& t,
2                                     Cjt_bicicletas& c_bici) {
3    map<string, Estacion>::iterator it = estaciones.find(t.value());
4    int missing = it->second.consultar_plazas();
5    if (not t.empty()) {
6      if (not t.left().empty()) {
7        map<string, Estacion>::iterator itleft =
8            estaciones.find(t.left().value());
9        int nleft = itleft->second.consultar_nbicis();
10       map<string, Estacion>::iterator itright =
11           estaciones.find(t.right().value());
12       int nright = itright->second.consultar_nbicis();
13
14       int i = 0;
15       while ((nleft != 0 or nright != 0) and i < missing) {
16         if (nleft > nright) {
17           autocompletar_bicis(it->second, itleft->second, c_bici);
18           --nleft;
19         } else if (nleft < nright) {
20           autocompletar_bicis(it->second, itright->second, c_bici);
21           --nright;
22         } else if (nleft == nright) {
23           string id_esq = itleft->second.menor_id();
24           string id_dre = itright->second.menor_id();
25
26           if (id_esq < id_dre) {
27             autocompletar_bicis(it->second, itleft->second, c_bici);
28             --nleft;
29           } else {
30             autocompletar_bicis(it->second, itright->second, c_bici);
31             --nright;
32           }
33         }
34         ++i;
35       }
36       completar_aux(t.left(), c_bici);
37       completar_aux(t.right(), c_bici);
38     }
39   }
40 }
41
42 void Cjt_estaciones::completar(Cjt_bicicletas& c_bici) {
43   completar_aux(arbolID, c_bici);
44 }
```

Imatge 1: Funcions per pujar bicis

Implementació:
Per a la funció de pujar bicis, en el meu codi anomenat “completar”, utilitzarem una funció auxiliar, “completar_aux”, ja que necessitem l’arbre d’identificadors per utilitzar els strings com a “key” per buscar fàcilment en el map<string, Estacion> i trobar el nombre de desocupacions en aquella estació. També necessitem el conjunt de bicicletes, perquè en moure bicis d’una estació a una altra, també s’ha de modificar la seva ubicació, però sense afegir un viatge.

Per tant, en aquesta funció auxiliar, per a cada node, anirem consultant el nombre de bicis a la dreta i a l’esquerra, i si és possible, pujarem tantes bicis al node. El node no té per què quedar ple. En resum, sigui “t” l’arbre d’identificadors, l’invariant seria: A cada node, la quantitat de bicicletes al “t.value()”, a “t.left()” i “t.right()” i la quantitat de desocupacions totals a l’arbre és constant des de l’inici fins al final.

Abans del bucle apuntem a l’estació corresponent al valor del node amb un iterador “it”. També guardem a un enter, “missing”, que és la desocupació d’aquesta estació.

Instruccions al bucle:

- *Inicialitzacions:* “nleft” i “nright” s’inicialitzen amb el nombre de bicicletes a “t.left()” i “t.right()”. “i = 0” indica el nombre d’iteracions del bucle.

- *Condició de sortida:* El bucle “while” seguirà mentre hi hagi bicicletes a les dues subestacions inferiors, és a dir, “nleft” i “nright” siguin diferents de 0, i mentre encara hi hagi places disponibles al node, és a dir, “i < missing”.

- *Cos del bucle:* Com volem que el nombre de bicicletes de les subestacions sigui el més equilibrat possible, tindrem 3 casos. Si “nleft > nright”, pugem una bici de t.left() i en restem 1. Si “nleft < nright”, pugem una bici de t.right() i en restem 1. I finalment, si “nleft” i “nright” són iguals, pujarem la bicicleta de menor identificador.

- *Acabament:* Sortim del bucle quan una de les condicions de sortida es compleixin.

- *Instruccions finals:* Finalment, “i” serà el nombre de bicicletes mogudes a aquell node, i les crides recursives a “t.left()” i “t.right()” completaran els nodes esquerra i dreta.

2. Mètode recursiu per assignar estació

```
1  void Cjt_estaciones::asignar_est_aux(const BinTree<string>& t, double& pl,
2                                       int& c, pair<double, string>& est) {
3    if (t.left().empty() and t.right().empty()) {
4      double plazas = estaciones[t.value()].consultar_plazas();
5      pl += plazas;
6      ++c;
7      if (plazas > est.first) {
8        est.first = plazas;
9        est.second = t.value();
10     } else if (plazas == est.first) {
11       if (t.value() < est.second) est.second = t.value();
12     }
13   } else {
14     double pl_left, pl_right;
15     int c_left, c_right;
16     pl_left = pl_right = c_left = c_right = 0;
17     asignar_est_aux(t.left(), pl_left, c_left, est);
18     asignar_est_aux(t.right(), pl_right, c_right, est);
19
20     c = c_left + c_right + 1;
21     pl = estaciones[t.value()].consultar_plazas() + pl_left + pl_right;
22     double plazas = pl / c;
23
24     if (plazas > est.first) {
25       est.first = plazas;
26       est.second = t.value();
27     } else if (plazas == est.first) {
28       if (t.value() < est.second) est.second = t.value();
29     }
30   }
31 }
32
33 string Cjt_estaciones::asignar_est() {
34   pair<double, string> est;
35   double pl = 0;
36   int c = 0;
37   asignar_est_aux(arbolID, pl, c, est);
38
39   return est.second;
40 }
```

Imatge 2: Funcions per assignar estació

- *Cas base:* Si estem a una fulla, “t.left().empty() and t.right().empty()”. En el cas base, actualitzem les variables “pl” i “c” i després comparem si en aquesta fulla hi ha més places que el màxim “est.first”. Si així és, actualitzem el pair<double, string> amb els valors de l’estació actual. Si hi ha les mateixes places que el màxim, es compararan els strings. El més petit serà el definitiu.

- *Cas inductiu:* Si no estem a una fulla, per tant, té 2 fills. En el cas inductiu, cridem a la funció recursiva “asignar_est_aux” pels nodes esquerre i dret. Després de les crides, actualitzem les variables “pl” sumant les places actuals, les de t.left() i les de t.right(), i “c” sumant els comptadors de l’esquerre, de la dreta i 1 de l’actual.
  La funció “asignar_est_aux” realitza crides recursives a t.left() i t.right() fins a arribar al cas base. L’actualització de les variables “pl” i “c” i la comparació amb el màxim de places es fan i/s’actualitzen a cada nivell, així que els resultats sempre estaran correctes.

- *Decreixement:* En aquesta funció recursiva, l’arbre original es divideix en subarbre esquerre i dret, així que el problema es redueix a una estructura molt menor.