Solució de l'Examen Parcial EDA 15/04/2021

Proposta de solució al problema 1

(a) Calculem primer el límit

$$\lim_{x\to\infty} \frac{2^{2n}}{2^n} = \lim_{x\to\infty} 2^n = \infty$$

.

Per tant, l'única afirmació certa és que $2^{2n} \in \Omega(2^n)$.

A continuació, calculem el límit

$$\lim_{x\to\infty} \frac{\log(2n)}{\log(n)} = \lim_{x\to\infty} \frac{\log(2)+\log(n)}{\log(n)} = \lim_{x\to\infty} \frac{\log(2)}{\log(n)} + \lim_{x\to\infty} \frac{\log(n)}{\log(n)} = 0+1 = 1$$

.

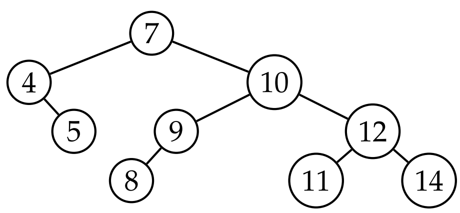
Així doncs, $\log(2n) \in \Theta(\log(n))$. Per tant també $\log(2n) \in O(\log(n))$ i $\log(2n) \in \Omega(\log(n))$.

(b) El cos de cada iteració del bucle és $\Theta(1)$ i per tant només cal comptar quantes iteracions es fan. Si y_t denota el valor de y al final de la t-èsima iteració, el que busquem és el mínim $t \geq 0$ tal que $y_t > n$. Sabem que $y_t = 1+2+3+4+\cdots+(t+1) = \Theta(t^2)$ i per tant el mínim t és $\Theta(\sqrt{n})$.

(c) El codi llegeix dues seqüències d'n enters i calcula si les dues seqüències tenen intersecció no buida.

El cas pitjor es dona quan tots els elements de la primera seqüència són diferents. Això farà que hi hagi n elements al *map*. Sabem que en C++ buscar un element en un *map* té cost logarítmic en el nombre d'elements. Per tant, en el cas pitjor es donen n voltes al bucle, i cada volta té cost logarítmic, pel que el cost total en cas pitjor és $\Theta(n \log n)$.

(d) L'arbre AVL resultant és:



Proposta de solució al problema 2

(a) Ho demostrarem per inducció sobre n.

Cas base: $(n = 0)$. En aquest cas, la graella té mida 1×1 i per tant només té una casella, que ha de ser necessàriament la casella bloquejada. Així doncs, no hi ha caselles restants per omplir i el resultat es compleix trivialment.

Pas d'induccció: sigui $n > 0$ i assumim que el resultat és cert per graelles de mida $2^{n-1} \times 2^{n-1}$. Aleshores podem partir la graella en 4 parts iguals, que seran subgraelles de mida $2^{n-1} \times 2^{n-1}$. Si, tal com es mostra a la figura de l'enunciat, la casella bloquejada cau a la subgraella de baix a l'esquerra, aleshores podem col·locar una peça a la part central de manera que bloqueja exactament una casella a les altres 3 subgraelles. Si la casella bloquejada cau a una altra subgraella, sempre podrem escollir una peça que bloquegi exactament una casella a les altres 3 subgraelles. Per tant, després d'haver col·locat aquest peça central sempre tindrem 4 subgraelles de mida $2^{n-1} \times 2^{n-1}$ amb exactament una casella bloquejada a cadascuna d'elles. Gràcies a la hipòtesi d'inducció sabem que totes elles es poden omplir amb les peces disponibles, i per tant la graella original de mida $2^n \times 2^n$ també.

(b)

```
int quadrant(Coord pos, int i_l, int i_r, int j_l, int j_r) {
  int size = j_r - j_l + 1;
  int i_m = i_l + size/2;
  int j_m = j_l + size/2;
  if (pos.first < i_m and pos.second < j_m) return 0;
  if (pos.first < i_m) return 1;
  if (pos.second < j_m) return 2;
  return 3;
}

void fill (vector<vector<int>>& M, int i_l, int i_r, int j_l, int j_r,
           Coord c_blocked, int& num){
  if (i_l == i_r) return; // 1x1
  int size = j_r - j_l + 1;
  int i_m = i_l + size/2; // Midpoints
  int j_m = j_l + size/2;

  vector<Coord> coords_blocked(4); // Blocked cell in each quadrant
  coords_blocked[0] = {i_m - 1, j_m - 1};
  coords_blocked[1] = {i_m - 1, j_m};
  coords_blocked[2] = {i_m, j_m - 1};
  coords_blocked[3] = {i_m, j_m};

  int q = quadrant(c_blocked, i_l, i_r, j_l, j_r);
  coords_blocked[q] = c_blocked;

  for (int k = 0; k < 4; ++k)
    if (M[coords_blocked[k].first][coords_blocked[k].second] == -1)
```

```
        M[coords_blocked[k].first][coords_blocked[k].second] = num;

    ++num;

    fill (M, i_l, i_m - 1, j_l, j_m - 1, coords_blocked[0], num); // Q0
    fill (M, i_l, i_m - 1, j_m, j_r,     coords_blocked[1], num); // Q1
    fill (M, i_m, i_r,     j_l, j_m - 1, coords_blocked[2], num); // Q2
    fill (M, i_m, i_r,     j_m, j_r,     coords_blocked[3], num); // Q3
}
```

(c) Per analitzar el cost del codi, ens hem de fixar en la funció $fill$. Fixem-nos que el seu codi essencialment no té bucles. Només n'hi ha un, però sempre dóna 4 voltes (independentment de la mida del problema). Per tant, fa un nombre constant de voltes. Com que totes les altres instruccions triguen temps $\Theta(1)$, si no consideréssim les crides recursives la funció trigaria temps constant. No obstant, per un problema de mida $2^n \times 2^n$ es fan 4 crides recursives amb mida $2^{n-1} \times 2^{n-1}$. Per tant, el cost en funció de n ve donat per la recurrència:

$$T(n) = 4T(n-1) + \Theta(1)$$

que té solució $T(n) \in \Theta(4^n)$.

Com que el nombre de caselles és $2^n \cdot 2^n = 4^n$, podem afirmar que el codi és lineal en el nombre de caselles.