Solució de l'Examen Final EDA 12/06/2015

Proposta de solució al problema 1

a) Aquest ja va sortir al parcial i la solució és la mateixa.
b) L'algorisme serveix per comprimir dades i per tant la correcta és la tercera.
c) El problema és en efecte NP-complet i per tant no pot existir cap algorisme de cost polinòmic que el resolgui a menys que P = NP; per tant la resposta correcta és la segona.

Proposta de solució al problema 2

a)

```
Arbre interseccio (Arbre a1, Arbre a2) {
    if (not a1 or not a2) return nullptr;
    Node* p = new Node;
    p→esq = interseccio (a1→esq, a2→esq);
    p→dre = interseccio (a1→dre, a2→dre);
    return p;
}
```

b) El cas pitjor és $\Theta(n1 + n2)$ (i es dóna quan els dos arbres són iguals, p.ex.).

Proposta de solució al problema 3

```
void backtrack(int k) {
  if (k == n) { solution_found = true; return; }
  for (int j = 0; j < n and not solution_found; ++j) {
    int cell_col = board[k][j].color;
    if (not onion_in_col[j] and not onion_in_region[cell_col]
      and not onion_in_neighborhood(k, j)) {
      col[k] = j;
      board[k][j].has_onion = true;
      onion_in_col[j] = true;
      onion_in_region[cell_col] = true;
      backtrack(k+1);
      if (not solution_found) {
        onion_in_region[cell_col] = false;
        onion_in_col[j] = false;
        board[k][j].has_onion = false;
      }
} } }
```

Proposta de solució al problema 4

Les EDs quedarien així:

```
struct Usuari {
    ...                                  // dades personals
    string nom;                          // no hi ha dos usuaris amb el mateix nom
    unordered_set<string> seguits;       // conjunt dels noms dels usuaris seguits
    list<Piulada> piulades;              // piulades de l'usuari en ordre invers
```

```
};

struct Xarxa {
    unordered_map<string, Usuari> usuaris;
};
```

El tipus *Usuari* és una extensió del donat; també podríem haver creat un tipus nou que es digui *UsuariExtes*. Aquesta diferència és menor.

Mantindrem tots els usuaris en un diccionari on les claus són els *nom*s d'usuari i les dades són les tuples dels *Usuari*s associats. D'aquesta forma, podrem accedir eficientment a totes les dades dels usuaris a través del seu nom. Com que no ens importa l'ordre, podem utilitzar una taula de hash per implementar el diccionari.

Per guardar les relacions de seguiment, extendrem la tupla *Usuari* amb un conjunt de strings que identifiquen els usuaris seguits. Com que no ens importa l'ordre, podem utilitzar una taula de hash per implementar el conjunt. Les operacions de *afegir_seguiment* $(x,\ u1,\ u2)$ i *treure_seguiment* $(x,\ u1,\ u2)$ s'implementen cercant *u1* al diccionaris *usuaris* i inserint/esborrant *u2* al conjunt *seguits* corresponent. El temps d'aquestes operacions és doncs constant en mitjana.

Per guardar les piulades, també extendrem la tupla *Usuari* amb una llista de piulades, aquesta llista guardarà les piulades en ordre cronològic invers. L'operació *piular_missatge* $(x,\ u,\ m)$ ha d'afegir el missatge m al davant de la llista de piulades de l'usuari u trobat al diccionari. El temps d'aquesta operació és doncs constant en mitjana.

Per implementar *mes_recents* (x,u,k) cal trobar primer la llista L de seguits de l'usuari u amb el diccionari. Després, conceptualment, cal concatenar totes les llistes de piulades dels usuaris en L, ordenar-les per hora d'emissió inversa i quedar-se amb les k primeres.

Però això es pot fer més eficient amb una cua de prioritats: Partint d'una cua de prioritats buida, s'insereix el primer missatge (el més recent doncs) de cada usuari seguit en L. Després, es repeteix k vegades el procés següent:

- S'extreu la piulada més recent de la cua de prioritats (i s'afegeix al final de la llista de piulades retornades),
- s'afegeix a la cua de prioritats el següent missatge del mateix usuari que el que s'ha tret.

Si u segueix m usuaris i $n = \max\{k, m\}$, el cost d'aquest procés és $O(n \log n)$, que sembla prou eficient donat que k és petit i n no és massa gran.