Solució de l'Examen Final EDA 10/01/2022

Proposta de solució al problema 1

(a) El codi escriurà les 5 paraules següents:

pep, pe, ep, e, p

Pel que fa al cost, sabem que el cost de *mystery* ve donat per la recurrència $T(n) = 2T(n-1) + \Theta(n)$. Això és així perquè es fan dues crides recursives de mida $n-1$ i les altres operacions tenen cost constant, excepte les crides a *substr*, que tenen cost $\Theta(n-1) = \Theta(n)$. Aquesta recurrència té solució $T(n) \in 2^n$.

(b) Sigui $n = s.size()$. Anem a comptar quantes vegades s'executa l'*insert* de dins el bucle. Per cada i, el bucle intern dona exactament $n-i$ voltes. Com que i varia entre 0 i $n-1$, això ens dóna $n + (n-1) + \cdots + 1 = \Theta(n^2)$ crides a *substr*. Però amb això no en fem prou perquè cada crida a *substr* té cost $j - i + 1$.

Anem a comptar el cost de totes aquestes crides. Recordem que per unes i, j concretes el cost de la crida a *substr* és $j - i + 1$. Per una i concreta, j es mou entre i i $n-1$ i per tant, el cost de les crides a *substr* amb aquesta i és $1 + 2 + \cdots + (n-i) = \Theta((n-i)^2)$. Si sumem sobre totes les i's el cost total és $\sum_{i=0}^{n} \Theta((n-i)^2) = \Theta(n^3)$.

(c) Una possible solució és:

```
void mystery(const string & s, unordered_set<string>& res){
  for (int k = 1; k <= s.size (); ++k)
    for (int i = 0; i + k <= s.size (); ++i)
      res . insert (s.substr(i,k));
}
```

Proposta de solució al problema 2

(a) Una possible solució és:

```
bool find_ranking (vector<int>& ranking, vector<int>& pos_in_ranking,
                   vector<bool>& used, int idx){
  if (idx == n) return good_ranking( pos_in_ranking );
  else {
    for (int i = 0; i < n; ++i) {
      if (not used[i]) {
        ranking[idx] = i;
        pos_in_ranking[i] = idx;
        used[i] = true;
        if (find_ranking(ranking, pos_in_ranking ,used,idx+1)) return true;
        used[i] = false;
      } } }
  return false;
}

bool good_ranking (const vector<int>& pos_in_ranking) {
  for (Match& g : matches) {
    if (pos_in_ranking[g. first ] > pos_in_ranking[g.second]) return false;
```

```
        if (pos_in_ranking[g.second] > pos_in_ranking[g.third]) return false;
      }
      return true;
    }
```

(b) Essencialment el codi genera totes les permutacions dels n jugadors i comprova si n'hi ha alguna de correcta. En cas pitjor no n'hi haurà cap de correcta i per tant haurà de generar i comprovar totes les permutacions. Així doncs es faran $n!$ crides a *good_ranking*.

Com que cada crida a *good_ranking* té cost en cas pitjor $\Theta(m)$, això ens dona una fita inferior del codi de $\Theta(m \cdot n!)$.

(c) Es pot solucionar el problema en temps polinòmic en n i m. Per a fer-ho construïm un graf dirigit on els nodes són els jugadors. Per cada partida amb resultat (j_1, j_2, j_3) afegim dos arcs $j_1 \to j_2$ i $j_2 \to j_3$. Per tant afegirem com a molt $\Theta(m)$ arcs. Un cop hem construït el graf buscarem una ordenació topològica en temps $O(n + m)$. Si l'ordenació topològica acaba sense haver afegit tots els jugadors voldrà dir que no hi ha un rànquing possible.

Nota: si no anem amb compte podríem afegir arcs repetits, però això no és un problema per a la correcció o el cost de l'algorisme de cerca topològica explicat a classe.

Proposta de solució al problema 3

(a) Anomenem X al problema d'aquest apartat. No és raonable pensar que podem solucionar X en temps polinòmic. Vegem per què. Considerem `5-COL` el problema del 5-colorejat de grafs, que sabem que és NP-complet. Existeix una reducció de `5-COL` cap a X: donat un graf G amb vèrtexs V i arestes E, considerem el conjunt de col·laboradors $\{c_u | u \in V\}$, i per cada aresta $\{u, v\} \in E$ imposem que el col·laborador c_u vol evitar el col·laborador c_v. Com que hem reduït polinòmicament `5-COL` a X, si $X \in P$ també tindrem `5-COL` $\in P$, i això és un problema obert a dia d'avui.

(b) Anomenem Y al problema d'aquest apartat. Podem afirmar que $Y \in P$. Vegem per què. Considerem `2-COL` el problema del 2-colorejat de grafs, que sabem que pertany a P. Existeix una reducció de Y cap a `2-COL`: donat un conjunt de col·laboradors $\mathcal{C}$ i una llista d'incompatibilitats I_c per cada $c \in \mathcal{C}$, construïm una instància de `2-COL` que consisteix en el graf G amb vèrtexs $\{v_c \mid c \in \mathcal{C}\}$ i arestes $\{\{v_c, v_d\} \mid d \in \mathcal{C}, c \in I_d\}$. Com que hem trobat una reducció de Y cap a `2-COL` i aquest últim pertany a P, aleshores $Y \in P$.

(c) Anomenem Z al problema d'aquest apartat. No és raonable pensar que podem solucionar Z en temps polinòmic. Vegem per què. Considerem `PARTICIÓ`, el problema de determinar si podem partir un conjunt d'enters en dues parts que sumin igual. Sabem que aquest problema és NP-complet. Existeix una reducció de `PARTICIÓ` cap a Z: donat un conjunt d'enters S, considerem el multiconjunt de col·laboradors $\{c_s \mid s \in S\}$, tal que el patrimoni de c_s és precisament s. Com que hem reduït polinòmicament `PARTICIÓ` a Z, si $Z \in P$ també tindrem `PARTICIÓ` $\in P$, i això és un problema obert a dia d'avui.

Proposta de solució al problema 4

(a) Una possible solució és:

```
int search(int x, const vector<int>& v) {
  int n = v.size();
  if (n == 0) return -1;
  int b = 1;
  while (b < n and v[b] < x) b *= 2;
  return bin_search(x,v,b/2,min(n-1,b));
}
```

(b) El primer que cal fer és observar el comportament del bucle. Després de k voltes, el valor de b és 2^k. Fixem-nos que s'atura tan bon punt troba un valor b tal que $v[b] \geq x$. Per tant, s'atura amb un nombre de voltes k tal que $v[2^k] \geq x$ però tal que $v[2^{k-1}] < x$ i això ens indica que $k = \lceil \log i \rceil$. Per tant, el cost del bucle és $\Theta(\log i)$. Remarquem també que, si el bucle s'atura perquè $b \geq n$, el mateix raonament és vàlid.

Finalment, vegem el cost de la crida a *bin_search*. En el cas pitjor $b \geq n - 1$ i, per tant, el cost de la crida és $\Theta(\log(b - b/2 + 1))$. Com que després de k voltes, b val 2^k, si el bucle ha fet k voltes el cost de la crida a *bin_search* serà $\Theta(\log(2^k - 2^{k-1} + 1)) = \Theta(\log(2^{k-1} + 1))$. Sabem que el nombre de voltes és $k = \lceil \log i \rceil$, i per tant el cost de la crida a *bin_search* és $\Theta(\log i)$.

Resumint, tot plegat té un cost en cas pitjor de $\Theta(\log i)$.