Solució de l'Examen Final EDA 16/01/2023

Proposta de solució al problema 1

(a) Observem que f és un procediment recursiu, pel que escriurem la recurrència que descriu el seu cost i la solucionarem. És fàcil observar que totes les operacions que fa el procediment són $\Theta(1)$ (comparacions entre enters, divisions per 2 i residus entre 2), excepte la crida recursiva. Sabem que, si x té n bits, $x/2$ tindrà $n-1$ bits, pel que la recurrència que descriu el cost és $C(n) = C(n-1) + \Theta(1)$. Si apliquem el teorema mestre per recurrències substractores del tipus $C(n) = a \cdot C(n-c) + \Theta(n^k)$, podem identificar $a = 1$, $c = 1$ i $k = 0$, que sabem que té solució $C(n) = \Theta(n^{k+1}) = \Theta(n)$.

(b) La funció retorna la mitjana aritmètica dels nombres del vector v. Per a veure-ho, si la mida de v és 2^k, ho podem demostrar per inducció sobre k.

Per $k = 0$, el vector té un sol element i per tant, la mitjana coincideix amb l'únic element del vector, tal com fa el codi.

Sigui ara $k > 0$ i assumim la hipòtesis d'inducció: per a tot vector de mida 2^{k-1}, la funció retorna la mitjana dels seus nombres. Aleshores, donat un vector $v = (x_1, x_2, \ldots, x_{2^k})$ la funció construeix aux, un vector de mida 2^{k-1} amb els elements $(\ (x_1 + x_2)/2,\ (x_3 + x_4)/2,\ \ldots,\ (x_{2^k-1} + x_{2^k})/2\ )$. Per tant, la hipòtesis d'inducció ens garanteix que la crida recursiva retornarà la mitjana d'aquest conjunt:

$$\frac{\frac{x_1+x_2}{2} + \frac{x_3+x_4}{2} + \ldots + \frac{x_{2^k-1}+x_{2^k}}{2}}{2^{k-1}}$$

que és igual a

$$\frac{x_1 + x_2 + x_3 + x_4 + \ldots + x_{2^k-1} + x_{2^k}}{2^k}$$

és a dir, la mitjana aritmètica dels elements de v.

Pel que fa al seu cost, veiem que és una funció recursiva. El vector es passa per referència, i això té cost $\Theta(1)$. Crear el vector buit aux també té cost $\Theta(1)$. El bucle fa $n/2$ voltes, i a cada volta es fa un treball $\Theta(1)$. Per tant, el cost del bucle és $\Theta(n)$. Finalment, és fàcil veure que el vector aux té mida $n/2$. Així doncs, la recurrència que descriu el cost de la funció és $C(n) = C(n/2) + \Theta(n)$. Si apliquem el teorema mestre per recurrències divisores del tipus $C(n) = a \cdot C(n/b) + \Theta(n^k)$, podem identificar $a = 1$, $b = 2$, $k = 1$ i calcular $\alpha = \log_2(1) = 0$. Com que $k > \alpha$, sabem que la solució és $C(n) = \Theta(n^k) = \Theta(n)$.

Proposta de solució al problema 2

(a) Ho demostrarem per inducció sobre h. El cas base ($h = 0$) és fàcil perquè és obvi que $1 = \frac{3-1}{2}$.

Sigui $h > 0$ i assumim la hipòtesis d'inducció: $1 + 3 + \ldots + 3^{h-1} = \frac{3^h-1}{2}$. Aleshores

$$1 + 3 + \ldots + 3^{h-1} + 3^h = \frac{3^h - 1}{2} + 3^h = \frac{3^h - 1 + 2 \cdot 3^h}{2} = \frac{3^{h+1} - 1}{2}$$

(b) Per construir un min-heap ternari d'alçada h amb el menor nombre de nodes, haurem d'omplir els h primers nivells i situar un únic node en el nivell $h+1$.

És fàcil veure que en el primer nivell tenim 1 node, en el segon nivell 3 nodes, en el tercer 3^2 nodes, i en general, en el nivell i tenim 3^{i-1} nodes.

Feta aquesta observació, el nombre mínim de nodes d'un min-heap ternari és $1 + 3 + \ldots + 3^{h-1} + 1$, que gràcies a l'apartat anterior sabem que equival a $\frac{3^h-1}{2} + 1 = \frac{3^h+1}{2}$.

Per tant, si n és el nombre de nodes d'un min-heap ternari qualsevol d'alçada h, sabem que

$$n \geq \frac{3^h + 1}{2}$$

que equival a afirmar que

$$h \leq \log_3(2n - 1)$$

.

Per tant, podem concloure que $h \in O(\log n)$.

(c) Donat un node en la posició i, els seus tres fills (en cas que existeixin tots tres) estaran en les posicions $(3i-1, 3i, 3i+1)$. El seu pare estarà a la posició $\lfloor \frac{i+1}{3} \rfloor$. També és correcte l'expressió $\lceil \frac{i-1}{3} \rceil$.

(d) Una possible solució és:

```
void Heap::sink (int i) {
  if (3*i - 1 < v.size ()) {
    int pos_min = 3*i - 1;
    if (3*i < v.size () and v[3*i] < v[pos_min]) pos_min = 3*i;
    if (3*i + 1 < v.size () and v[3*i + 1] < v[pos_min]) pos_min = 3*i + 1;
    if (v[pos_min] < v[i]) {
      swap(v[i], v[pos_min]);
      sink(pos_min);
    }
  }
}
```

Proposta de solució al problema 3

(a) Una possible solució és:

```
bool evaluate (const vector<vector<int>>& F, const vector<bool>& alpha) {
  for (int i = 0; i < F.size (); ++i) {
    bool some_true = false;
    for (int j = 0; not some_true and j < F[i]. size (); ++j)
      some_true = evaluate_lit (F[i][ j ], alpha );
```

```
    if (not some_true) return false;
  }
  return true;
}

bool SAT (int n, const vector<vector<int>>& F, vector<bool>& alpha) {
  if (alpha.size() == n+1)
     return evaluate (F,alpha );
  else {
    alpha.push_back( false );
    bool b1 = SAT(n,F,alpha );
    alpha.back() = true;
    bool b2 = SAT(n,F,alpha );
    alpha.pop_back ();
    return b1 or b2;
  }
}
```

(b) La clau és observar que, essencialment, aquest programa prova totes les possibles *alpha* i, per cada una d'elles crida a la funció *evaluate*. Com que hi ha 2^n possibles *alpha*, aquest és el nombre de crides. En aquest programa, el cas millor i pitjor coincideixen.

Si ho volguéssim justificar més formalment, podríem calcular $C(k)$, el nombre de crides a *evaluate* que fa la funció SAT quan *alpha* té mida $(n+1)-k$.

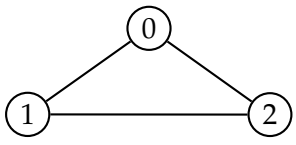
És fàcil veure que $C(0) = 1$, perquè en aquest cas *alpha* tindrà mida $n+1$ i estarem en el cas base de SAT, que només fa una crida a *evaluate*.

Per a $k > 0$ tenim que $C(k) = 2C(k-1)$, perquè es fan dues crides recursives on s'ha afegit un element a *alpha*.

És fàcil veure que aquesta recurrència té solució $C(k) = 2^k$. Quan fem la crida a SAT des del main, *alpha* té mida 1, i així doncs el nombre de crides a *evaluate* que es faran serà $C(n) = 2^n$.

Proposta de solució al problema 4

(a) Una de les propietats de les reduccions és que transformen instàncies negatives en instàncies negatives. Considerem, per exemple, el graf



Clarament aquesta és una instància negativa de $\mathbf{2-COL}$. No obstant, la funció *reduccio* ens construeix una fórmula on totes les variables apareixes només de manera negada. Per tant, és obvi que fent totes les variables falses podem satisfer la fórmula. Així doncs, $reduccio(G, 2)$ és una instància positiva de $\mathbf{2-COL}$ i aquesta reducció no compleix la propietat que hem esmentat.

(b) La clau està en adonar-se que, essencialment, la fórmula que construeix *reduccio* assegura que dos vèrtexs units per una aresta no poden tenir el mateix color. No obstant, en cap cas assegura que cada vèrtex té almenys un color. Això ho podem assegurar afegint, per a cada vèrtex i (amb $0 \leq i < n$) una clàusula:

```
x(i,1) v x(i,2) v x(i,3) v ... v x(i,k)
```

Tot i que no era necessari, mostrem el codi C++ corresponent. Caldria afegir el següent bucle al final de *reduccio*:

```
for (int u = 0; u < n; ++u) {
  for (int c = 1; c ≤ k; ++c) cout ≪ (c==1 ? "" : " v ") ≪ x(u,c);
  cout ≪ endl;
}
```

(c)
- Si G és una instància positiva de **2-COL**, aleshores també és una instància positiva de **3-COL**: Cert
- Si G és una instància positiva de **4-COL**, aleshores també és una instància positiva de **3-COL**: Fals
- Si trobéssim un algorisme polinòmic per **3-COL**, també existiria un algorisme polinòmic per **4-COL**: Cert
- Si trobéssim un algorisme polinòmic per **4-COL**, també existiria un algorisme polinòmic per **3-COL**: Cert