Cognoms **Nom** **DNI**

Examen Final EDA **Duració: 3h** **16/01/2023**

- *L'enunciat té 4 fulls, 8 cares, i 4 problemes.*
- *Poseu el vostre nom complet i número de DNI a cada full.*
- *Contesteu tots els problemes en el propi full de l'enunciat a l'espai reservat.*
- *Llevat que es digui el contrari, sempre que parlem de cost ens referim a cost asimptòtic en temps.*
- *Llevat que es digui el contrari,* **cal justificar les respostes.**

Problema 1 **(2 pts.)**

Responeu les preguntes següents:

(a) (0.75 pts.) Considereu el procediment següent:

```
void f (int x) {
  if (x ≠ 0) {
    f(x/2);
    cout ≪ x%2;
  }
}
```

Sigui x un nombre natural i sigui n el nombre de bits de x. Quin és el cost de f en funció d'n?

(b) (1.25 pts.) Considereu la funció següent, on assumirem que la mida de v és sempre una potència de 2:

```
double mystery (const vector<double>& v) {
  if (v.size() == 1) return v[0];
  else {
    vector<double> aux;
    for (int i = 0; i < v.size(); i+=2)
      aux.push_back((v[i] + v[i+1])/2); // assumim cost Theta(1)
    return mystery(aux);
  }
}
```

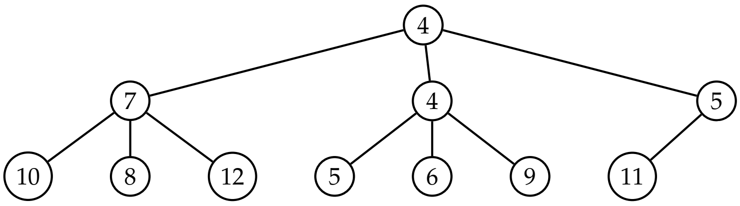
Què calcula aquesta funció? Justifica formalment la teva resposta.

Si $n = v.size()$, quin és el cost d'aquest programa en funció d'n?

Cognoms **Nom** **DNI**

Problema 2 **(3 pts.)**

Diem que un arbre ternari d'alçada h és *complet* si els seus h primers nivells estan plens i l'últim nivell té totes les fulles el màxim a l'esquerra possible. Un *min-heap ternari* és un arbre ternari complet tal que el valor de tot node és menor o igual que el valor dels seus fills. Un exemple de min-heap ternari d'alçada 2 és el següent:



(a) (0.75 pts.) Demostra que per a tot $h \geq 0$ tenim $1 + 3 + 3^2 + \ldots + 3^h = \frac{3^{h+1}-1}{2}$.

(b) (0.75 pts.) Quin és el mínim nombre de nodes que un min-heap ternari d'alçada h pot tenir? Utilizeu aquesta quantitat per demostrar que l'alçada d'un min-heap ternari amb n nodes és $O(\log n)$.

(c) (0.5 pts). De la mateixa manera que ho fem amb els min-heaps binaris, guardarem un min-heap ternari amb n nodes en un vector de mida $n+1$, on la primera posició no la utilitzarem. Per exemple, el min-heap de la figura anterior el guardarem en el vector

0	1	2	3	4	5	6	7	8	9	10	11
X	4	7	4	5	10	8	12	5	6	9	11

Donat un node que es guarda a la posició i del vector, en quines posicions es guarden els seus fills? I el seu pare? No cal que justifiqueu la resposta.

(d) (1 pt.) Us donem a continuació una implementació parcial d'un min-heap ternari per a guardar enters. Completeu-la per tal que, donat un min-heap ternari amb n elements, la funció *remove_min* tingui cost $\Theta(\log n)$ en cas pitjor.

```
class THeap {
  vector<int> v;
  void sink (int i);
  public:
  THeap () {v.push_back(0);}
  int size ( ) const;
  int min ( ) const;
  void add (int x);
  int remove_min ( );
};
```

```
int THeap::remove_min ( ){
  int x = v[1];
  v[1] = v.back();
  v.pop_back();
  sink(1);
  return x; }
```

```
  void THeap::sink (int i) {
  if ( ________ < v.size()) {
    int pos_min = ________ ;

    ________

    if (v[pos_min] < v[i]) {

      ________

  } } }
```

Cognoms Nom DNI

Problema 3 **(2 pts.)**

Com ja sabem, donat un conjunt finit de variables $\{x_1, x_2, \ldots, x_n\}$, diem que un **literal** és una variable (x_i) o bé la negació d'una variable ($\neg x_i$). Una **clàusula** és una disjunció de literals, per exemple, $x_3 \vee \neg x_1 \vee \neg x_2$. Una fórmula en **CNF** és una conjunció de clàusules.

El conegut problema **CNF-SAT** consisteix en, donada una fórmula F en CNF, determinar si F té almenys un model. És a dir, decidir si existeix una funció α que assigna cert o fals a cada variable i satisfà F.

Per resoldre aquest problema assumirem que les fórmules venen donades en el format DIMACS, on la primera línia indica el nombre de variables i clàusules, i les variables són nombres $\{1, 2, \ldots, n\}$.

Fórmula:

$$
\begin{array}{l}
(x_1 \vee \neg x_2) \wedge \\
(x_2 \vee \neg x_3 \vee x_1) \wedge \\
(x_3) \wedge \\
(x_2 \vee x_3)
\end{array}
$$

Format DIMACS:

```
p cnf 3 4
1 -2 0
2 -3 1 0
3 0
2 3 0
```

(a) (1.5 pts.) Omple el següent codi per tal de determinar si una fórmula en CNF és satisfactible. Dins la funció *SAT* no pots utilitzar cap *if*:

```
int main ( ){
  vector<vector<int>> F;
  int n, m; // n variables, m clauses
  string aux;
  cin >> aux >> aux >> n >> m;
  for (int i = 0; i < m; ++i) {
    F.push_back({});
    int lit ;
    while (cin >> lit and lit != 0) F.back().push_back( lit );
  }
  vector<bool> alpha(1); // alpha[0] not used because var 0 does not exist
  cout << SAT(n, F, alpha) << endl;
}


bool evaluate_lit (int lit , const vector<bool>& alpha) {
  if ( lit > 0) return alpha[ lit ];
  else return not alpha[- lit ];
}
```

```
bool evaluate (const vector<vector<int>>& F, const vector<bool>& alpha) {
  for (int i = 0; i < F.size(); ++i) {

  }
  return true;   }

bool SAT (int n, const vector<vector<int>>& F, vector<bool>& alpha) {
  if (alpha.size() == n+1)
    return evaluate(F,alpha);
  else {

    bool b1 = SAT(n,F,alpha);

    bool b2 = SAT(n,F,alpha);

    return
  } }
```

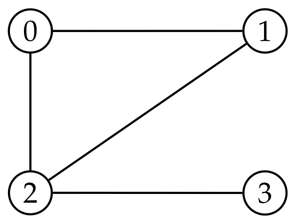
(b) (0.5 pts.) En funció d'n, quantes vegades es crida la funció *evaluate* en el cas pitjor? I en el cas millor?

Cognoms   Nom   DNI

Problema 4 **(3 pts.)**

Per a tot enter $k \geq 1$, donat un graf $G = (V, E)$ no dirigit, el problema **k-COL** consisteix en determinar si existeix una funció $c : V \rightarrow \{1, 2, \ldots, k\}$ de manera que per a tota aresta $\{u, v\} \in E$ es compleixi $c(u) \neq c(v)$.

(a) (0.7 pts.). Ens asseguren que un procediment *reduccio* és una reducció polinòmica de **k-COL** cap a **CNF-SAT**. Donat el graf de l'esquerra i $k = 3$, aquest procediment escriu la fórmula en CNF de la dreta, on cada línia és una clàusula, `v` indica una disjunció, les negacions de variables es representen amb "`-`" i, intuïtivament, una variable `x(i,j)` serà certa si i només si el vèrtex `i` té el color `j`. Hem afegit línies en blanc per millorar la llegibilitat.




```
-x(0,1) v -x(1,1)
-x(0,1) v -x(2,1)
-x(1,1) v -x(2,1)
-x(2,1) v -x(3,1)

-x(0,2) v -x(1,2)
-x(0,2) v -x(2,2)
-x(1,2) v -x(2,2)
-x(2,2) v -x(3,2)

-x(0,3) v -x(1,3)
-x(0,3) v -x(2,3)
-x(1,3) v -x(2,3)
-x(2,3) v -x(3,3)
```

De forma més precisa, si G és un graf amb n vèrtexs $\{0, 1, 2, \ldots, n-1\}$ representat com a llistes d'adjacència, i $1 \leq k \leq n$, el procediment *reduccio* és el següent:

```
string x (int u, int k) { // retorna l'string "x(u,k)"
  return "x(" + to_string (u) + "," + to_string (k) + ")";
}

void reduccio (const vector<vector<int>>& G, int k) {
  int n = G.size ();
  for (int c = 1; c ≤ k; ++c)
    for (int u = 0; u < n; ++u)
      for (int v : G[u])
        if (v > u) cout << "−" << x(u,c) << " v −" << x(v,c) << endl;
}
```

Expliqueu per què el prodeciment anterior no compleix totes les propietats d'una reducció polinòmica. *Pista:* si necessiteu un contraexemple, n'hi ha prou amb considerar un cert graf amb 3 vértexs i $k = 2$.

(b) (0.7 pts.) Expliqueu com modificaríeu el procediment anterior per a què sigui una reducció polinòmica correcta. No és necessari escriure codi en C++.

(c) (1.6 pts.) Considereu les següents afirmacions sobre **k-COL**:

(1) Si G és una instància positiva de **2-COL**, aleshores també és una instància positiva de **3-COL**.

(2) Si G és una instància positiva de **4-COL**, aleshores també és una instància positiva de **3-COL**.

(3) Si trobéssim un algorisme polinòmic per **3-COL**, també existiria un algorisme polinòmic per **4-COL**.

(4) Si trobéssim un algorisme polinòmic per **4-COL**, també existiria un algorisme polinòmic per **3-COL**.

Ompliu la següent taula amb una C o una F depenent de si l'afirmació corresponent és **C**erta o **F**alsa. Cada resposta correcta suma 0.4 punts. Cada resposta incorrecta resta 0.4. Les respostes en blanc no compten.

1	2	3	4