Cognoms **Nom** **DNI**

Examen Parcial AP3 Duració: 2.5 hores 30/10/2019

- *L'enunciat té 5 fulls, 10 cares i 3 problemes.*
- *Poseu el vostre nom complet i número de DNI a cada full.*
- *Contesteu tots els problemes en el propi full de l'enunciat i a l'espai reservat.*
- *A no ser que es digui el contrari, cal justificar les respostes.*

Problema 1 (4 punts)

Una *variable booleana* només pot prendre el valor 0 (fals) o el valor 1 (cert). Un *literal* és una variable booleana o la seva negació. Una *clàusula* és una disjunció de literals. Una *CNF* és una conjunció de clàusules. El problema de **CNF-SAT** consisteix en, donada una CNF F, determinar si és satisfactible, és a dir, si existeix una assignació de valors a les variables booleanes de F que faci F certa.
Navegant per Internet us trobeu una pàgina web on es dóna un programa que s'afirma que resol **CNF-SAT** en temps polinòmic.

(a) (1 pt.) Hi ha cap motiu que faci pensar que probablement hi hagi algun problema amb el programa? Si es que sí, per què?

(b) (0.75 pts.) El programa representa les dades de la forma següent. Les variables booleanes es representen amb nombres enters positius, i els literals amb nombres enters diferents de 0. La negació de la variable booleana x és $-x$. Les clàusules es representen com conjunts de literals (la disjunció s'entén implícita), i les CNF com conjunts de clàusules (la conjunció s'entén implícita):

```
typedef int          variable ;
typedef int          literal  ;
typedef set< literal >  clause ;
typedef set<clause>     cnf;
```

Així doncs, per exemple es representen:

- les variables booleanes x_1, x_2 i x_3 amb 1, 2 i 3 respectivament;
- els literals x_1, $\neg x_2$ i x_3 amb 1, -2 i 3 respectivament;
- les clàusules $x_1 \vee \neg x_2 \vee x_3$ i $\neg x_1 \vee \neg x_3$ amb $\{1, -2, 3\}$ i $\{-1, -3\}$ respectivament;
- la CNF $(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$ amb $\{\{1, -2, 3\}, \{-1, -3\}\}$.

La funció principal del programa és la següent funció *sat*:

```
bool propagate(clause& C, literal l) {
  return C.erase(-l); // Retorna true quan -l era a C i per tant s'ha esborrat
}

bool sat(cnf& F) {
  set <literal> propagated;
  stack<literal> pending;

  for (const clause& C : F) {
    if (C.size() == 0) return false;
    else if (C.size() == 1) {
      literal l = *(C.begin());
      if (propagated.find(l) == propagated.end()) {
        propagated.insert(l);
        pending.push(l);
  } } }

  while (not pending.empty()) {
    literal l = pending.top();
    pending.pop();
    cnf nF;
    for (const clause& C : F) {
      clause nC = C;
      if (propagate(nC, l)) {
        if (nC.size() == 0) return false;
        else if (nC.size() == 1) {
          literal l = *(nC.begin());
          if (propagated.find(l) == propagated.end()) {
            propagated.insert(l);
            pending.push(l);
      } } }
      nF.insert(nC);
    }
    F = nF;
  }
  return true;
}
```

Demostreu que tota assignació que satisfaci la fórmula ha de fer certs els literals continguts a la variable *propagated*.

(c) (0.75 pts.) És el cost de la funció *sat* polinòmic? Justifiqueu la vostra resposta.

(d) (0.75 pts.) Si *sat* (*F*) retorna **false**, podem afirmar llavors que *F* és una fórmula insatisfactible? Justifiqueu la vostra resposta.

Pista: podeu usar l'apartat (b)

(e) (0.75 pts.) Si *sat* (*F*) retorna **true**, podem afirmar aleshores que *F* és una fórmula satisfactible? Justifiqueu la vostra resposta.

Cognoms **Nom** **DNI**

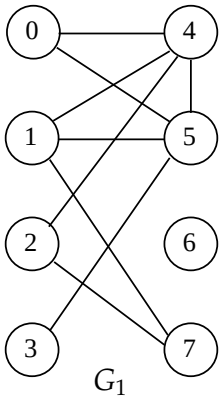
Problema 2 **(4 punts)**

Dos grafs (no dirigits) $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ es diuen *isomorfs* si existeix un *isomorfisme* entre ells, això és, una bijecció $\rho : V_1 \to V_2$ tal que $\{u, v\} \in E_1$ si i només si $\{\rho(u), \rho(v)\} \in E_2$.

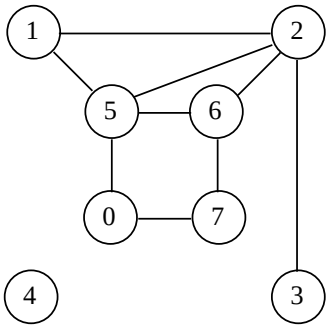
(a) (0.5 pts.) Demostreu que, si hi ha un isomorfisme $\rho : V_1 \to V_2$ entre $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$, aleshores per tot $u \in V_1$ es té que $\text{grau}(u) = \text{grau}(\rho(u))$.

Recordeu que el grau d'un vèrtex en un graf és el nombre d'arestes que hi incideixen.

(b) (0.5 pts.) Doneu un isomorfisme $\rho : V_1 \to V_2$ que certifiqui que els grafs G_1 i G_2 a continuació són isomorfs:



G_1



G_2

$\rho(0) =$ $\rho(1) =$ $\rho(2) =$ $\rho(3) =$

$\rho(4) =$ $\rho(5) =$ $\rho(6) =$ $\rho(7) =$

Pista: podeu usar l'apartat anterior

(c) (2 pts.) Completeu el codi següent per decidir si dos grafs donats són isomorfs.

Noteu que els grafs s'implementen amb matrius d'adjacència: si G és un graf, el booleà $G[x][y]$ és cert quan x i y són adjacents a G, i val el mateix que $G[y][x]$.

```
typedef vector<vector<bool>> Graph;

bool compatible(const Graph& G1, int x1, const Graph& G2, vector<int>& p) {
  for (int y1 = 0; y1 <= x1; ++y1)
    if ( [                                ] ) return false;
  return true;
}

bool rec(int x1, const Graph& G1, const Graph& G2, vector<int>& p, vector<bool>& used) {
  if (x1 == G1.size()) return [          ];
  for (int x2 = 0; x2 < G2.size(); ++x2) {
    if ( [          ] ) {
      [          ]
      p[x1] = x2;
      if (compatible(G1, x1, G2, p) and rec(x1+1, G1, G2, p, used)) return [          ];
      [          ]
  } }
  return [          ];
}

bool iso(int n1, int m1, const Graph& G1, int n2, int m2, const Graph& G2) {
  if (n1 != n2 or m1 != m2) return [          ];
  vector<int> p(n1);
  vector<bool> used(n1, false);
  return rec(0, G1, G2, p, used);
}

void read(int& n, int& m, Graph& G) {
  cin >> n >> m;
  G = Graph(n, vector<bool>(n, false));
  for (int k = 0; k < m; ++k) {
    int x, y;
    cin >> x >> y;
    G[x][y] = G[y][x] = true;
} }

int main() {
  int n1, m1, n2, m2;
  Graph G1, G2;
  read(n1, m1, G1);
  read(n2, m2, G2);
  if (iso(n1, m1, G1, n2, m2, G2)) cout << "true" << endl;
  else                             cout << "false" << endl;
}
```

Cognoms	**Nom**	**DNI**

(d) (1 pt.) Expliqueu quins canvis faríeu al codi de l'apartat (c) per fer-lo més eficient usant l'apartat (a).

Aquesta cara estaria en blanc intencionadament si no fos per aquesta nota.

Cognoms **Nom** **DNI**

Problema 3 **(2 punts)**

Donat un conjunt U i un conjunt $C = \{S_1, \ldots, S_n\}$ de subconjunts de U (és a dir, $S_i \subseteq U$ per tot $1 \leq i \leq n$), un *empaquetament* de C és un subconjunt $P \subseteq C$ tal que per tot $S, S' \in P$ es compleix $S \cap S' = \emptyset$.

Per exemple, si $U = \{1,2,3,4,5,6\}$ i $C = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{6\}\}$, llavors $\{\{1,2\},\{3,4\},\{6\}\}$ és un empaquetament, però en canvi $\{\{1,2\},\{2,3\},\{6\}\}$ no (perquè $\{1,2\} \cap \{2,3\} = \{2\} \neq \emptyset$).

El problema de **EMPAQUETAMENT** consisteix en, donats un conjunt U, un conjunt C de subconjunts de U i un natural k, determinar si existeix un empaquetament P de C amb almenys k elements, és a dir, $|P| \geq k$.

Nota: aquí $|P|$ representa el nombre d'elements de P, **no** el nombre de bits necessaris per a codificar P.

(a) (0.5 pt.) Demostreu que **EMPAQUETAMENT** pertany a la classe NP.

(b) (1 pt.) Donat un graf $G = (V, E)$ i un natural k, el problema de **INDEPENDENT** consisteix en determinar si existeix un subconjunt $S \subseteq V$ d'almenys k vèrtexs (és a dir, $|S| \geq k$) que sigui *independent*: per tot $u, v \in S$, els vèrtexs u i v **no** són adjacents, o sigui, $\{u, v\} \notin E$.

És ben sabut que **INDEPENDENT** és un problema NP-complet.

Doneu una reducció polinòmica de **INDEPENDENT** a **EMPAQUETAMENT** i demostreu que ho és.

(c) (0.5 pts.) Què podeu concloure, tenint en compte els dos apartats anteriors?