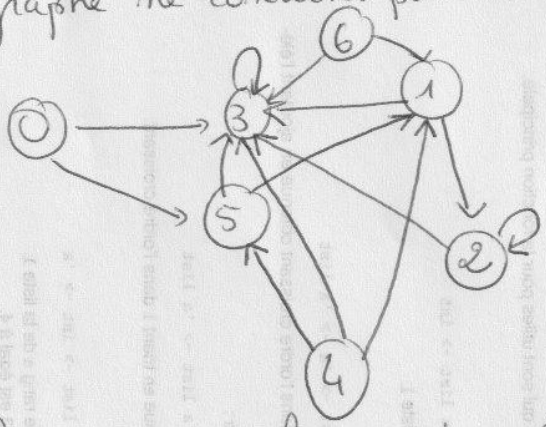
Partie 2

① 1. Ce graphe ne contient pas de boucle donc pas de clique.

2.



$\{3\}$ forme une clique de célébrités.

② Soient C_1 et C_2 deux cliques de célébrité.
Fixons $c_1 \in C_1$ et $c_2 \in C_2$ quelconques alors comme C_2 est une clique de célébrités, $(c_1, c_2) \in A$ mais comme C_1 est une clique de célébrités, ceci implique que $c_2 \in C_1$ et ainsi on a montré que $C_2 \subset C_1$.

Par symétrie des rôles, on obtient aussi $C_1 \subset C_2$ et donc $\boxed{C_1 = C_2}$.

③ On va tout d'abord montrer que $C_G \setminus \{p\}$ est une clique de célébrités de G'.

Soit $x \in C_G \setminus \{p\}$ et $y \in S \setminus \{p\}$ alors $(x, y) \in A$ et donc à $A \cap (S \setminus \{p\} \times S \setminus \{p\})$
de plus si $(y, x) \in A$ alors $y \in C_G$ et donc $(y \neq p)$ à $C_G \setminus \{p\}$

1. Si $C_{G'} = \phi$ alors d'après le résultat précédent, pour la clause C_G, on a $C_G \setminus \{p\} = \phi$ donc

$$\boxed{C_G \in \{\phi, \{p\}\}}$$

2. Si $C_G \setminus \{p\} \neq \emptyset$ alors c'est une clique de célébrités $\neq \emptyset$ de G' et par unicité c'est bien $C_{G'}$.

3. (a) $(p, c') \notin A$ implique $c' \notin C_G$.
Donc $C_G \setminus \{p\} \neq C_{G'}$ donc par contraposée de 2, on a
$C_G \setminus \{p\} = \emptyset$ et donc $C_G \in \{\emptyset, \{p\}\}$

(b) Si $(c', p) \notin A$ alors $p \notin C_G$ et donc
$C_G = C_G \setminus \{p\}$ ainsi C_G est vide ou
$C_G = C_G \setminus \{p\}$ est une clique de célébrités pour G' non vide, c'est $C_{G'}$ ainsi
$$C_G \in \{\emptyset, C_{G'}\}$$

(c) Si (p, c') et (c', p) sont dans A alors
↳ si $p \in C_G$ on a $c' \neq p \in C_G$
ainsi $\boxed{C_G \neq \{p\}}$
Or, on sait par ② que $C_G \setminus \{p\} \in \{\emptyset, C_{G'}\}$
et donc $C_G \in \{\emptyset, \{p\}, C_{G'} \cup \{p\}\}$.
On a exclu la deuxième possibilité donc
$C_G \in \{\emptyset, C_{G'} \cup \{p\}\}$.

④

```
bool** graphe-vide (int n) {
    bool** res = malloc (n * sizeof (bool*));
    for (int i=0; i<n; i++) {
        res[i]= malloc (n * sizeof (bool));
        for (int j=0; j<n; j++) {
            res[i][j] = false;
        }
    }
    return res;
}
```

⑤

```
bool est-clique (bool** g, int n, bool* ens) {
    for (int i=0; i<n; i++) {
        if (ens[i]) {
            for (int j=0; j<n; j++) {
                if (ens[j]) {
                    if (!g[i][j]) return false;
                }
            }
        }
    }
    return true;
}
```

⑥ C = [ ]

1e tour: C = [0]

2e tour: r = 1 , c = 0 , t = false
(1,0)? oui (0,1) oui
on ajoute 1

3e tour: C = [0,1] r = 2 c = 0 t = false.
(2,0)? non (0,2)?
donc C devient [2]

4e tour: C = [2] r = 3 c = 2 t = false.
(3,2)? oui (2,3)? non.
t = true

5e tour: C = [2] r = 4 c = 2 t = false.
(4,2)? oui (2,4) oui
donc C devient [2,4]

6e tour: C = [2,4] r = 5 c = 2 t = false.
(5,2)? oui (2,5)? non
t = true

7e tour: C = [2,4] r = 6 c = 2 t = false.
(6,2)? oui (2,6)? oui
on ajoute 6.

A la fin: C = [2,4,6]

(7)

- Si G a un seul sommet x et que $C_G \neq \emptyset$ alors $C_G = \{x\}$ et l'algo renvoie effectivement $C = \{x\}$ (un seul tour de boucle où on est ds le 1e cas)

- <u>hérédité :</u> Soit la propriété vraie pour tout graphe à n sommets.

Soit $S = \{0, \dots, n\}$ et $G = (S, A)$ à $n+1$ sommets.

On pose $G' = (S \setminus \{n\}, A \cap (S \setminus \{n\} \times S \setminus \{n\}))$ ĉ en q° 3.

Les n premières itérations de l'algo construisent C correspondant à l'execution de l'algo sur G' ainsi :

<u>cas 1 :</u> Si $C_{G'} \neq \emptyset$ alors par HR C correspond à l'ensemble $C_{G'}$ et de plus en posant c le 1e elt de C

* si $(n, c) \notin A$ alors par 3.3.a, $C_G = \{n\}$ et c'est exactement ce que fait la condition : on pose $C = [n]$ et ceci ne changera pas quelle que soit le résultat de la condition suivante
* si $(c, n) \notin A$ $\wedge$ (mais $(n, c) \in A$) alors par 3.3.b, $C_G = C_{G'}$ et c'est ce qui se passe dans la 2e condition $C = C_{G'}$ en ne changeant pas, on récapète le C du tour précédent.
* sinon, l'algo ajoute n à $C_{G'}$ et on obtient $C = \{n\} \cup C_{G'}$ ce qui est exactement C_G par 3.3.c.

<u>cas 2 :</u> Si $C_{G'} = \emptyset$ alors on ne sait pas trop ce que vaut C après les n ler tours de boucles.

et d'après 3.1, on sait que $C_G = \{n\}$.
Ainsi, on sait que soit C est vide et on ajoute n à l'ensemble vide ce qui convient soit $C \neq \phi$ et son 1e elt $c \neq n$, $c \notin C_G$ ainsi on est sûr que $(n, c) \notin A$ et $(c, n) \in A$ donc l'algo va passer la première condition $(n, c) \notin A$ et poser $C = [n]$ ce qui convient.

(8) Si $G_C = \phi$ comme c'est le cas dans l'exemple proposé par l'énoncé alors la liste renvoyée peut très bien être non vide comme c'est le cas dans l'exemple étudié.

(9)
```
bool est_vide (liste l) {
    return (l == NULL); }
```

(10)
```
int premier (liste l) {
    assert (! est_vide (l));
    return (l -> value);
}
```

(11)
```
liste append (liste l, int v) {
if (est_vide (l)) {
    struct noeud* new = malloc (sizeof (struct noeud));
    new -> value = v;
    new -> next = NULL;
    return new;
} else { liste lp = l;
    while (lp -> next != NULL) {
        lp = lp -> next; }
    struct noeud* new = malloc (sizeof (struct noeud));
    new -> value = v; new -> next = NULL;
```

```
    lp -> next = neco;
    return l;
  }
}
```

(12)

```
liste clique_possible_c (bool** g, int n) {
    liste l = NULL;
    for (int i = 0; i<n; i++) {
        if (est_vide(l)) {
            l = append(l, i);
        }
        else {
            int c = premier(l);
            bool t = false;
            if (!g[i][c]) {
                t = true;
                l = append(NULL, i);
            }
            if (!g[c][i]) {
                t = true;
            }
            if (!t) {
                l = append(l, i);
            }
        }
    }
    return l;
}
```

(13) Etant donnés un graphe G, un entier k et la description d'un ensemble de sommets de G de taille k, on peut vérifier en temps polynomial que cet ensemble est une clique, c'est d'ailleurs ce que fait la fonction de la q° 5.

De plus $\langle G, k\rangle \in$ Clique ssi il existe une telle description d'un ensemble de sommets de taille k qui sera accepté par cette fonction avec G et k.

Enfin la description d'un tel ensemble est bien polynomiale en la taille de $\langle G, k\rangle$ car on peut utiliser un tableau de booléens indexé par les sommets de G. Ainsi Clique $\in$ NP.

(14) Soit $\varphi = C_1 \wedge \dots \wedge C_k$ une formule sous forme 3SAT.

On construit G_φ de la façon suivante:

- pour chaque clause C_i, on crée trois sommets, un pour chaque littéral et on ne met pas d'arêtes entre ces trois sommets (il y a au total $3k$ sommets dans G_φ).
- pour deux sommets l et l' appartenant à deux clauses $\neq$ alors on les relie sauf s'ils sont une variable et sa négation.

Montrons $\varphi \in$ 3SAT ssi $\langle G_\varphi, k\rangle \in$ Clique.

↳ Si φ est satisfiable, soit $v \models \varphi$

On considère $V = \{ l \in G_\varphi : v(l) = \text{true} \}$

Montrons que V une clique de G_φ de taille k.

Tout d'abad, comme $v \models \varphi$, il y a un littéral de chaque clause qui est dans V au moins. On va en choisir un dans chaque clause et on obtient ainsi un ensemble $V' \subset V$ avec $|V'| = k$.

Soit l et soit l' dans V' puis que $v(l) = v(l') = \text{true}$ alors l et l' ne sont pas une variable et sa négat° et sont donc reliés dans G_φ par construction.

Ainsi V' est bien clique de taille k de G_φ.

↳ Réciproquement, soit C une clique de taille k dans G_φ, on pose v qui rend chaque littéral de C vrai et met les variables non déterminées à false. v est bien définie puisque deux sommets de C ne peuvent pas être opposés. v met à vrai au moins un littéral de chaque clause car une clique de taille k contient un sommet de chaque clause qui ne contient pas d'arête et donc v est un modèle de φ qui est bien satisfiable.