Un corrigé : X-ENS INFO C 2025
Couplages maximaux et parfaits

Pour toutes les remarques ou corrections, vous pouvez m'envoyer un mail à
galatee.hemery@gmail.com

Partie I : Algorithme d'Edmonds

Question 1.

```
let rec del lst1 lst2 = match lst1 with
    |[] -> []
    |t::q when List.mem t lst2 -> del q lst2
    |t::q -> t::(del q lst2)
```

Question 2.

```
let rec keys lst = match lst with
    |[] -> []
    |(a,b)::q -> a::(keys q)
```

Question 3. (1) $\{\{1,2\},\{0,5\}\}$ n'est pas un couplage du graphe car 1 et 2 ne sont pas relié pas une arête dans le graphe représenté.

(2) $\{\{1,3\},\{0,5\}\}$ est un couplage.

(3) $\{\{0,1\},\{2,3\},\{0,5\}\}$ n'est pas un couplage car 0 est l'extrémité de deux arêtes de l'ensemble.

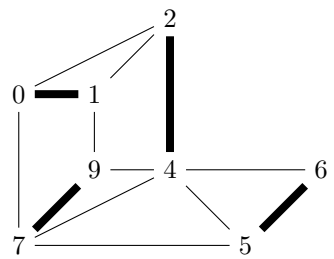
Question 4.

```
let rec couverts c = match c with
    |[] -> []
    |(a,b)::q -> a::b::(couverts q)
```

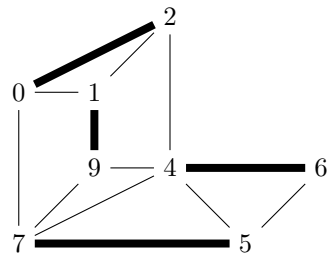
Question 5. *On note que la définition de couplage maximal du sujet correspond à la définition de couplage maximum enseigné en MPI.*
On peut proposer $\{\{0,5\},\{2,1\},\{3,4\}\}$ comme couplage maximal alternatif.

Question 6. *On note que la définition de chemin d'augmentation du sujet correspond à la définition de chemin alternant augmentant enseigné en MPI.*
On peut proposer $1,2,4,3$ comme chemin d'augmentation.

Question 7. On peut proposer le chemin d'augmentation $1,0,2,4,6,5,7,9$, qui donne le couplage augmenté :



On peut aussi proposé plus simplement $1,9$, qui donne le couplage augmenté :



Question 8. Soit $P = v_0, \ldots, v_n$ un chemin d'augmentation pour un couplage C.
On sait que v_0 et v_n ne sont pas couverts par C donc les arêtes $\{v_0, v_1\}$ et $\{v_{n-1}, v_n\}$ sont ajoutées car elles ne sont pas dans C.
Le caractère alternant implique que la longueur du chemin est impair $n = 2p+1$: on commence et on finit par des arêtes hors de C et une arête sur deux est dans C.
On enlève p arêtes et on en ajoute $p+1$ donc $C(P)$ a exactement une arête de plus que C.
D'autre part, v_0 et v_n deviennent couverts par une unique arête de $C(P)$ et les v_i pour $1 \leq i < n$ sont toujours couverts par une unique arête car on enlève une arête incidente à v_i et on en ajoute une autre.
Au total, $C(P)$ est un couplage contenant strictement plus d'arêtes que C.

Question 9.

```
let rec separer chm =
    |[] -> failwith "pas un chemin d'augmentation"
    |[a] ->  failwith "pas un chemin d'augmentation"
    |[a;b] -> [], [(a,b)]
    |a::b::c::q ->
        let lin, lout = separer (c::q) in
        (min b c, max b c)::lin, (min a b, max a b)::lout
```

Question 10.

```
let augmente cpl chm =
    let lstin, lstout = separer chm in
    let cpl1 = del cpl lstin in
    join cpl1 lstout (* ou bien cpl1@lstout *)
```

Question 11.

```
let contracteG grph lst w =
    let rec modifie_voisins lv nv_vu acc = match lv with
        |[] when nv_vu -> w::acc, true
        |[] -> acc, false
        |t::q when List.mem t lst -> modifie_voisins q true acc
        |t::q -> modifie_voisins q nv_vu (t::acc) in
    let rec parcours_listes l vois_w = match l with
        |[] -> [w,vois_w]
        |(u,lv)::q when List.mem u lst -> parcours_listes q vois_w
        |(u,lv)::q -> let nv_lv, voisin_w = modifie_voisins lv false [] in
                      if voisin_w then (u,nv_lv)::(parcours_listes q (u::vois_w))
                      else (u,nv_lv)::(parcours_listes q vois_w) in
    parcours_listes grph []
```

Question 12. Soit C un couplage dans un graphe G et B un bourgeon.
Montrons que C/B est un couplage dans G/B.
Considérons deux arêtes de C/B.

- Soit elles sont toutes les deux dans C et les extrémités hors du bourgeon B. Or C est un couplage donc elles ne partagent par d'extrémités.
- Soit on a une arête $\{a, b\}$ issu de C avec $a \notin B$ et $b \notin B$ et une arête $\{a', w\}$ avec $a' \notin B$ et l'existence d'un sommet $b' \in B$ tel que $\{a', b'\} \in C$.
  Or C est un couplage donc $\{a', b'\}$ et $\{a, b\}$ ne partagent par d'extrémités. Comme w est un nouveau sommet, $\{a', w\}$ et $\{a, b\}$ ne partagent par d'extrémités.
- Soit on a deux arêtes de la deuxième forme donc une arête $\{a, w\}$ avec $a \notin B$ et l'existence d'un sommet $b \in B$ tel que $\{a, b\} \in C$ et une arête $\{a', w\}$ avec $a' \notin B$ et l'existence d'un sommet $b' \in B$ tel que $\{a', b'\} \in C$.
  Or C est un couplage donc $\{a', b'\}$ et $\{a, b\}$ ne partagent par d'extrémités.
  En particulier, $b \neq b'$: il existe deux arêtes dans C ayant une extrémité dans le bourgeon et une extrémité hors du bourgeon. C'est impossible car dans un bourgeon, on a une unique base et c'est le seul sommet qui n'est pas couvert par une arête de C ayant ses deux extrémités dans le bourgeon.
  Ainsi, la troisième situation est impossible.

Dans tous les cas, deux arêtes de C/B ne partagent aucune extrémités donc chaque sommet est l'extrémité d'au plus une arête. C'est bien un couplage dans G/B.

Question 13.

```
let rec contracteC cpl brg w = match cpl with
    |[] -> []
    |(a,b)::q when List.mem a brg && List.mem b brg -> contracteC q brg w
    |(a,b)::q when List.mem a brg || List.mem b brg ->
        (min a w, max a w)::(contracteC q brg w)
    |(a,b)::q -> (a,b)::(contracteC q brg w)
```

Question 14.

```
let rec find foret v = match foret with
    |[] -> None
    |(N (e,f))::q when e = v -> Some [e]
    |(N (e,f))::q ->
        let test = find f v in
        if test = None then find q v
        else Some (e::(Option.get test))
```

Question 15.

```
let rec extend foret u v = match foret with
    |[] -> []
    |(N (e,f))::q when e = u -> (N (u,(N (v,[]))::(extend f u v)))::(extend q u v)
    |(N (e,f))::q -> (N (e,extend f u v))::(extend q u v)
```

Question 16. Initialement, tous les sommets non couverts pas C sont racines d'arbres de F donc à profondeur 0 paire et aucun n'appartient à T.
L'algorithme n'ajoute aucun arbre à la forêt et ne modifie pas les racines donc on garde comme ensemble de racines un ensemble de sommets non couvert par T.
Le $(3)(a)$ est la seule étape ajoutant des arêtes à la forêt : elles sont ajoutées par 2, une hors du couplage $\{u, v\}$ et une dans le couplage de $\{v, z\}$ de sorte à ce que les chemins de la racine à une feuille dans chaque arbre soit alternant.
Ainsi, lorsque l'on se trouve dans la situation $(3)(b)(i)$, on a une arête $\{u, v\}$, ainsi que deux chemins alternant :

$$r_u \text{ — } u_1 \text{ ▬ } u_2 \text{ - - } u_k \text{ ▬ } u \text{ — } v \text{ ▬ } v_p \text{ - - } v_2 \text{ ▬ } v_1 \text{ — } r_v$$

Les sommets r_u et r_v sont non couverts par C et dans le cas où l'arête $\{u, v\}$ n'appartient pas au couplage C, on a bien un chemin d'augmentation.
Dans le cas où $\{u, v\} \in C$, on aurait $u_k = v$ et $v_p = u$:

$$r_u \text{ — } u_1 \text{ ▬ } u_2 \text{ - - - - - - } u_{k-2} \text{ ▬▬ } u_{k-1} \text{ — } v \text{ ▬ } u \text{ ▬ } v \text{ ▬ } u \text{ — } v_{p-1} \text{ ▬▬ } v_{p-2} \text{ - - } v_2 \text{ ▬ } v_1 \text{ — } r_v$$

Pour ajouter les arêtes $\{u_{k-1}, v\}$ et $\{v_{p-1}, u\}$, les sommets v_{k-1} et v_{p-1} ont été traités avant u.
Supposons sans perte de généralités que u_{k-1} a été traité en dernier. Ainsi, v était voisin et comme v_{p-1} a été traité, on a ajouté $\{v_{p-1}, u\}$ et $\{u, v\}$ donc v apparaissait déjà dans un autre arbre à profondeur paire donc on aurait du s'arrêter lors du traitement de u_{k-1}.
Ainsi, on et toujours dans le premier cas et l'algorithme renvoie bien un chemin d'augmentation.

Question 17. On est dans le cas où v est dans la forêt mais pas à profondeur paire donc on passe au voisin suivant.

```
    -> aux_voisins u tl
```

Question 18. On est dans le cas $(3)(b)(i)$, u et v apparaissent à profondeurs paires dans deux arbres distincts, donc on renvoie un chemin d'augmentation.

```
    -> let fin = List.rev cv in
       Some (cu@fin)
```

Question 19.

```
let rec extrait cu cv = match (cu,cv) with
    |tu::tu1::qu, tv::tv1::qv when tu = tv && tu1 = tu2 -> extrait qu qv
    |tu::tu1::qu, tv::tv1::qv when tu = tv && tu1 <> tu2 -> tu1::(qu@cv)
    |_ -> assert false (*situation impossible*)
```

On note que `extrait` est appelé sur deux chemins de longueur paire alternant du même arbre. A profondeur impaire, on utilise les arêtes du couplage C donc les sommets n'ont qu'un seul fils donc on opère les vérifications aux profondeurs paires uniquement.

Question 20. On recherche un chemin d'augmentation dans `ng` pour `nc`, si on n'en trouve pas, on passe à la suite, sinon, on renvoie le chemin d'augmentation correspondant dans G.

```
    ( match (recherche ng nc) with
        | None -> aux_voisins u tl
        | Some ch -> Some (gonfle graphe bourgeon ch nv) )
```

Question 21.

```
let edmonds gph =
    let couplage_vide = [] in
    let rec ameliore cpl = match (recherche gph cpl) with
        | None -> cpl
        | Some ch -> ameliore (augmente cpl ch) in
    ameliore couplage_vide
```

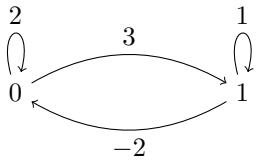
Partie II : Calculs de déterminants

Question 22.

```
int read_sqmatrix (int n, int *A, int i, int j){
    return A[i*n+j];
}

void write_sqmatrice (int n, int *A, int i, int j, int val){
    A[i*n+j] = val;
}
```

Question 23. On obtient le graphe suivant.



Question 24. On propose la marche fermée de longueur 3 et de tête 1 suivante :

$$1 \to 3 \to 2 \to 1$$

Question 25.

(1) Dans le graphe donné, les marches fermées de longueur au plus 3 sont :

$$(C_1) \quad 0 \to 1 \to 2 \to 0 \text{ de tête 0 et de poids } 2 \times 1 \times 2 = 4$$

$$(C_2) \quad 1 \to 2 \to 1 \text{ de tête 1 et de poids } 1 \times 3 = 3$$

$$(C_3) \quad 1 \to 2 \to 2 \to 1 \text{ de tête 1 et de poids } 1 \times 5 \times 3 = 15$$

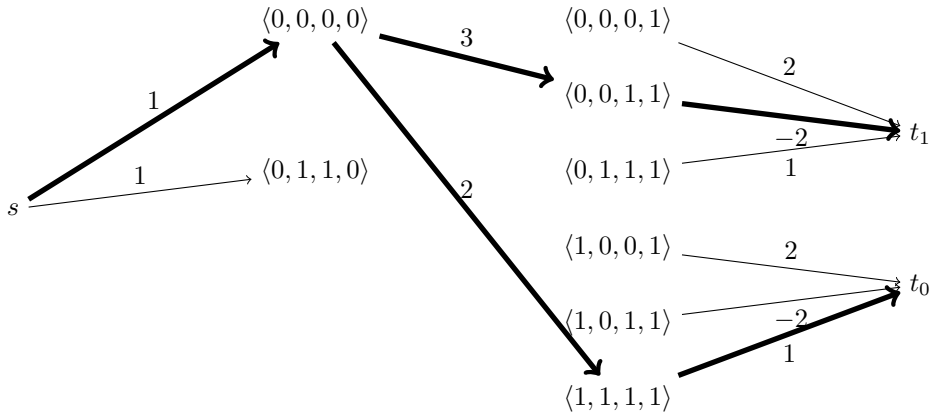
$$(C_4) \quad 2 \to 2 \text{ de tête 2 et de poids } 5$$

```
let rec recherche graphe couplage =
    let nc = del (keys graphe) (couverts couplage) in
    let foret = ref (List.map (fun x -> N(x, [])) nc) in
    let rec aux_voisins u liste_voisins =
        let cu = Option.get (find !foret u) in
        match liste_voisins with
            | [] -> None
            | v :: tl ->
                      match find !foret v with
                      | None -> let z = apparie couplage v in
                                foret := extend (extend !foret u v) v z;
                                aux_voisins u tl
                      | Some cv when (List.length cv) mod 2 = 0
                      (* nombre de sommets pair = profondeur impaire *)
                             -> aux_voisins u tl
                      | Some cv when (List.hd cu) <> (List.hd cv)
                             -> let fin = List.rev cv in
                                Some (cu@fin)
                      | Some cv
                             -> let bourgeon = extrait cu cv in
                                let nv = frais (keys graphe) in
                                let ng = contracteG graphe bourgeon nv in
                                let nc = contracteC couplage bourgeon nv in
                                ( match (recherche ng nc) with
                                    | None -> aux_voisins u tl
                                    | Some ch -> Some (gonfle graphe bourgeon ch nv) )
    in
    let rec aux_sommets traites =
        match prochain !foret traites with
        | None -> None
        | Some u -> let liste_voisins = del (List.assoc u graphe) traites in
                    match aux_voisins u liste_voisins with
                    | None -> aux_sommets (u::traites)
                    | Some ch -> Some ch
    in
    aux_sommets []
```

FIGURE 1 – Fonction `recherche` complétée.

(2) Les suites de marches fermées n'utilisant que les marches fermées de la question précédentes car le nombre total d'arcs doit être égal à 3 le nombre de sommets.
Ici, on a les suites :
— C_1 de poids 4 ;
— C_2, C_4 de poids $3 \times 5 = 15$;
— C_3 de poids 15.

Question 26. On obtient le graphe complété suivant avec les nouveaux arcs en gras.



Question 27. Si $C = h, u_1, \ldots, u_m, h$ est une marche fermée, on a en particulier $A_{h,u_1} \neq 0$, $A_{u_m,h} \neq 0$ et pour tout $1 \leq j < m, A_{u_j,u_{j+1}} \neq 0$.
De plus $h < u_j$ pour tout $1 \leq j \leq m$.
Ainsi, d'après le point (2) de la définition des arcs de H_A, on a l'existence des arêtes de $\langle p, h, u_j, k\rangle$ vers $\langle p, h, u_{j+1}, k+1\rangle$ pour $1 \leq j < m$ et de l'arête de $\langle p, h, h, k\rangle$ vers $\langle p, h, u_1, k+1\rangle$ pour tout $k < n-1$ et pour tout $p \in \{0, 1\}$.
Ainsi, en prenant des valeurs de k partant d'un $i < n - m$ et croissant, on obtient bien le chemin attendu.

$$\langle p, h, h, i\rangle \to \langle p, h, u_1, i+1\rangle \to \cdots \to \langle p, h, u_m, i+m\rangle$$

Pour une suite de marches fermées $S = C_1 \ldots C_k$, on a pour chaque marche fermée de la suite un morceau de chemin en notant u_{C_i} l'avant dernier sommet de la marche C_i :

$$(C_1) \quad \langle n \mod 2, t(C_1), t(C_1), 0\rangle \to \cdots \to \langle n \mod 2, t(C_1), u_{C_1}, |C_1| - 1\rangle$$

$$(C_2) \quad \langle 1 - (n \mod 2), t(C_2), t(C_2), |C_1|\rangle \to \cdots \to \langle 1 - (n \mod 2), t(C_2), u_{C_2}, |C_1| + |C_2| - 1\rangle$$

$$\cdots$$

$$(C_k) \quad \langle p_k, t(C_k), t(C_k), \sum_{i=1}^{k-1} |C_i|\rangle \to \cdots \to \langle p_k, t(C_k), u_{C_k}, \left(\sum_{i=1}^{k} |C_i|\right) - 1\rangle$$

avec $p_k = n \mod 2$ si k est impair et $p_k = 1 - (n \mod 2)$ si k est pair.
Le point (1) de la définition des arcs de H_A nous donne un arc de poids 1 de s vers $\langle n \mod 2, t(C_1), t(C_1), 0\rangle$ qui commence le chemin.
Le point (3) de la définition des arcs de H_A nous donne l'existence des arcs de $\langle p_j, t(C_j), u_{C_j}, \left(\sum_{i=1}^{j} |C_i|\right) - 1\rangle$ vers $\langle 1 - p_j, t(C_{j+1}), t(C_{j+1}), \sum_{i=1}^{j} |C_i|\rangle$ pour $1 \leq j < k$ qui permettent de concaténer les chemins bout à bout.
Le point (4) de la définition des arcs de H_A nous donne l'existence de l'arc entre $\langle p_k, t(C_k), u_{C_k}, \left(\sum_{i=1}^{k} |C_i|\right) - 1\rangle$ et t_{1-p_k} qui permet de terminer le chemin car $\sum_{i=1}^{k} |C_i| = n$.
Au total, on a bien un chemin de s à t_0 ou t_1 correspondant à toute marche fermée. En particulier, si elle est positive, on a $1 - p_k = 0$ car n et k ont même parité donc le chemin aboutit à t_0.

Question 28. *Pour être cohérent avec les poids des marches fermées, on considère que le poids d'un chemin est le produit des poids des arêtes et non la somme comme c'est le cas habituellement.*
On a pour tout $p \in \{0, 1\}, 1 \leq i \leq n-1, 0 \leq h < v \leq n-1$ (poursuite d'une marche fermée) :

$$F_A(\langle p, h, v, i\rangle) = \sum_{h \leq u \leq n-1, A_{u,v} \neq 0} A_{u,v} \times F_A(\langle p, h, u, i-1\rangle) = \sum_{u=h}^{n-1} A_{u,v} \times F_A(\langle p, h, u, i-1\rangle)$$

On a pour tout $p \in \{0,1\}, 1 \leq i \leq n-1, 0 \leq h = v \leq n-1$ (changement de marche fermée) :

$$F_A(\langle p,h,h,i\rangle) = \sum_{0\leq h'<h, h'\leq u\leq n-1, A_{u,h'}\neq 0} A_{u,h'} \times F_A(\langle 1-p,h',u,i-1\rangle) = \sum_{h'=0}^{h-1}\sum_{u=h'}^{n-1} A_{u,h'} \times F_A(\langle 1-p,h',u,i-1\rangle)$$

Le cas $v < h$ n'est pas à discuter car le sommet n'existe pas dans H_A.

Question 29.

```
1  Fourtable *initialise (int n) {
2      Fourtable* t = create(n);
3      for (int h = 0; h < n; h++) {
4          write(t,1,n % 2,h,h,0); // Sommet accessible pas une arête de poids 1
5      }
6      return t;
7  }
```

Les autres sommets de la couche 0 sont inaccessibles depuis s dont 0 convient (c'est la valeur par défaut utilisée par `create`).

Question 30.

```
1  Fourtable *remplit (int n, int *A) {
2      Fourtable* t = initialise(n);
3      for (int i = 1; i < n; i++) {
4          for (int p = 0; p < 2; p++) {
5              for (int h = 0; h < n; h++) {
6                  for (int v = h+1; v < n; v++) { //premier cas v>h
7                      for (int u = h; u < n; u++) {
8                          int tmp1 = read(t,p,h,v,i);
9                          int tmp2 = read(t,p,h,u,i-1);
10                         int coeff = read_sqmatrix(n,A,u,v);
11                         write(t,tmp1+coeff*tmp2,p,h,v,i);
12                     }
13                 }
14                 for (int hprime = 0; hprime < h; h++) { // second cas v=h
15                     for (int u = hprime; u < n; u++) {
16                         int tmp1 = read(t,p,h,v,i);
17                         int tmp2 = read(t,1-p,hprime,u,i-1);
18                         int coeff = read_sqmatrix(n,A,u,hprime);
19                         write(t,tmp1+coeff*tmp2,p,h,v,i);
20                     }
21                 }
22             }
23         }
24     }
25     return t;
26 }
```

On utilise les relations données à la question 28.
La complexité de `initialise` est en $O(\mathtt{n}^3)$ donc inférieure à $O(\mathtt{n}^4)$.
Dans chaque bloc lignes 8 à 11 ou ligne 16 à 19, on observe une complexité en $O(1)$ car toutes les fonctions de lecture et écriture sont de complexité en $O(1)$.
Il faut donc compter le nombre de passages dans ces blocs. On a deux $\mathtt{n}-1$ itérations de la boucle sur `i`, puis 2 itérations de la boucles sur `p` et enfin `n` itérations de la boucle sur `h`.
Pour une itérations de la boucle sur `h`, on a :

- d'une part deux boucles imbriquées concernant le premier cas, donnant au total $(\mathtt{n}-1-\mathtt{h}) \times (\mathtt{n}-\mathtt{h}) \leq \mathtt{n}^2$ itérations ;
- d'autre part deux boucles imbriquées concernant le second cas, donnant au total $\mathtt{h} \times (\mathtt{n}-\mathtt{hprime}) \leq \mathtt{n}^2$ itérations.

Ainsi, au total, on a au plus $(\mathtt{n}-1) \times 2 \times \mathtt{n} \times 2\mathtt{n}^2 \leq 4\mathtt{n}^4$ passages dans les blocs de complexité en $O(1)$ et on atteint bien une complexité en $O(\mathtt{n}^4)$ pour la fonction.

Question 31. On utilise le théorème admis de Mahajan et Vinay et les observations de la question 27 concernant les suites de marches fermées.

```
int determinant (int n, int *A){
    Fourtable* t = remplit(n,A);
    int det = 0;
    for (int h = 0; h < n; h++){
        for (int u = h; u < n; h++) {
            int pos = read(t,1,h,u,n-1);
            int neg = read(t,0,h,u,n-1);
            int coeff = read_sqmatrix(n,A,u,h);
            det = det + pos*coeff - neg*coeff;
        }
    }
    free(t);
    return det;
}
```

La fonction `remplit` a une complexité en $O(\mathtt{n}^4)$, la double boucle a une complexité en $O(\mathtt{n}^2)$ et `free` en $O(1)$ (admis). Ainsi, au total, la fonction a bien une complexité en $O(\mathtt{n}^4)$.

Partie III : Méthode algébrique et probabiliste pour les couplages parfaits

Question 32.

```
int tirage_tutte(int n, int *A){
    int X = malloc(n*n*sizeof(int)); assert(X!=NULL);
    for (int k = 0; k < n*n; k++) {
        X = rand_range(n*n);
    }
    int TG = malloc(n*n*sizeof(int)); assert(TG!=NULL);
    for (int i = 0; i < n; i++){
        for (int j = 0; j < i; j++){
            if (read_sqmatrix(n,A,i,j) == 1) { // {i,j} arête, j<i
                write_sqmatrix(n,TG,i,j,-read_sqmatrix(n,X,i,j));
            }
            else { // {i,j} pas une arête
                write_sqmatrix(n,TG,i,j,0);
            }
        }
        write_sqmatrix(n,TG,i,i,0); // i=j
        for (int j = i+1; j < n; j++){
            if (read_sqmatrix(n,A,i,j) == 1) { // {i,j} arête, i<j
                write_sqmatrix(n,TG,i,j,read_sqmatrix(n,X,i,j));
            }
            else { // {i,j} pas une arête
                write_sqmatrix(n,TG,i,j,0);
            }
        }

    }
    free(X);
    int det = determinant (n, TG);
    free(TG);
    return det;
}
```

Question 33. Le théorème de Tutte affirme que G possède un couplage parfait si et seulement si le déterminant de $T(G)$ n'est pas le polynôme nul.
Ainsi, dès lors que `tirage_tutte` renvoie une valeur non nulle, on sait que le polynôme n'est pas nul et que le graphe G admet un couplage parfait.
Néanmoins, lorsque l'on obtient une valeur nulle, on n'a aucune certitude quant à G.
Le polynôme correspondant au déterminant de $T(G)$ est de degré total au plus n d'après la formule du déterminant (dans le cas d'un graphe complet par exemple).
Le théorème de Schwartz-Zippel donne pour les tirage de Tutte de la question précédente :

$$Pr[\det(T(G))(x_{0,0}, x_{0,1}, \ldots, x_{n,n})] \leq \frac{n}{n^2+1}$$

car $S = \{0, 1, \ldots, n^2\}$. Si on répète k tirages de Tutte, la probabilité d'obtenir 0 k fois alors que le polynôme n'est pas nul est majorée par $\left(\frac{n}{n^2+1}\right)^k$.
On souhaite choisir k à partir d'un entier p de sorte à ce que

$$\left(\frac{n}{n^2+1}\right)^k \leq 2^{-p}$$

Soit :

$$k \log_2\left(\frac{n}{n^2+1}\right) \leq -p$$

$$k \geq \frac{p}{\log_2\left(n+\frac{1}{n}\right)} = O\left(\frac{p}{\log_2(n)}\right)$$

Ainsi, on obtient la complexité et la probabilité d'erreur attendues.

```
bool parfait (int n, int *A, int p){
    double k = p/(log2(n+1/n));
    for (int i = 0; i < k+1; i++) {
        if (tirage_tutte(n,A) != 0) {
            return true;
        }
    }
    return false;
}
```