CCP 2016 - OPTION INFORMATIQUE - UN CORRIGÉ

Corrigé proposé par Alain Schauber : alain.schauber@prepas.org

Partie I. Logique et calcul des propositions

I.1. Soit $A \in \mathcal{V}$ et f une tri-valuation définie sur $\mathcal{V}$ telle que $f(A) =?$, donc $\hat{f}(\neg A) =?$ et par suite $\hat{f}(A \vee \neg A) =? \neq \top$.

$$\boxed{\not\models_3 (A \vee \neg A)}$$

I.2. Telle que définie dans l'énoncé, la logique tri-valuée ne reconnaît pas les formules constantes $\bot$ et $\top$, sinon cette dernière aurait pu fournir une réponse à la question.
On peut cependant proposer : $A \Rightarrow A$. En effet, si A est une variable propositionnelle alors quelle que soit la tri-valuation f, on a $\hat{f}(A \Rightarrow A) = \top$, car les lignes de la table de vérité de $A \Rightarrow B$ correspondant à deux tri-valuations identiques pour A et B fournissent une tri-valuation égale à $\top$ pour $A \Rightarrow B$. Par induction structurelle (ou à l'aide du théorème de substitution), ce résultat reste vrai si on prend pour A une formule logique quelconque.

$$\boxed{\models_3 (A \Rightarrow A)}$$

I.3. On peut proposer pour tout couple $(A, B) \in \mathcal{V}^2$ et toute tri-valuation f :

$$\boxed{\hat{f}(A \wedge B) = \min(f(A), f(B)) \text{ et } \hat{f}(A \vee B) = \max(f(A), f(B))}$$

I.4. Il est clair en observant la table de vérité de $A \vee B$ que les propositions $A \vee B$ et $B \vee A$ sont équivalentes. Par suite, si on avait $\neg A \vee B$ équivalente à $A \Rightarrow B$ on aurait (par transitivité et spécialisation) équivalence entre les propositions $A \vee \neg A$ et $A \Rightarrow A$. Mais on a vu en question **I.1.** que $\not\models_3 (A \vee \neg A)$ alors que dans la question **I.2.** on a établi que $\models_3 (A \Rightarrow A)$. En conclusion :

$$\boxed{\text{Les propositions } \neg A \vee B \text{ et } A \Rightarrow B \text{ ne sont pas équivalentes en logique tri-valuée}}$$

I.5. Comme demandé, on construit la table de vérité conjointe des deux propositions pour les comparer :

A	B	$A \Rightarrow B$	$\neg B$	$\neg A$	$\neg B \Rightarrow \neg A$
$\top$	$\top$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	?	?	?	$\bot$	?
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$
$\bot$	?	$\top$	?	$\top$	$\top$
?	$\top$	$\top$	$\bot$	?	$\top$
?	$\bot$	?	$\top$	?	?
?	?	$\top$	?	?	$\top$

On en déduit que : $\boxed{\text{Les propositions } \neg B \Rightarrow \neg A \text{ et } A \Rightarrow B \text{ sont équivalentes}}$

I.6. Soit $\psi = ((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)) \Rightarrow B$. On en construit la table de vérité :

A	B	$A \Rightarrow B$	$\neg A \Rightarrow B$	$((A \Rightarrow B) \wedge ((\neg A) \Rightarrow B)$	B	ψ
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\top$	?	?	$\top$	?	?	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	?	$\top$	?	?	?	$\top$
?	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
?	$\bot$	?	?	?	$\bot$	?
?	?	$\top$	$\top$	$\top$	?	?

On constate en observant les deux dernières lignes que : $\boxed{\not\models_3 \psi}$

I.7. La dernière ligne de la table de vérité montre que $\hat{q}(A \to A) =? \neq \top$, d'où : $\boxed{A \to A \text{ n'est pas une tautologie}}$

I.8. Dans cette question semble se confirmer la remarque de la question **I.2.** concernant l'absence de la constante $\top$ dans l'ensemble des formules de la logique tri-valuée, car sinon elle représenterait une tautologie.
Notons q la tri-valuation de $\mathcal{V}$ dans $\{\top, \bot, ?\}$ définie par : $\forall A \in \mathcal{V}, q(A) =?$. Pour montrer qu'il n'existe pas de tautologie dans la logique tri-valuée associée aux opérateurs $\{\neg, \wedge, \vee, \to\}$, il suffit d'établir que : $\forall \phi \in \mathcal{F}, \hat{q}(\phi) =?$. Montrons cela par induction structurelle. Soit $\phi \in \mathcal{F}$.

- si $\phi \in \mathcal{V}$ alors $\hat{q}(\phi) = q(\phi) =?$
- supposons $\phi = \neg\phi_0$ et $\hat{q}(\phi_0) =?$. La table de vérité de l'opérateur $\neg$ montre que dans ce cas $\hat{q}(\phi) = \hat{q}(\neg\phi_0) =?$
- supposons $\phi = \phi_1 \Diamond \phi_2$, où ϕ_1, ϕ_2 vérifient l'hypothèse d'induction et $\Diamond$ représente l'un quelconque des opérateurs binaires $\wedge, \vee$ ou $\to$. En remarquant qu'une tri-valuation indéterminée à la fois de ϕ_1 et ϕ_2 correspond à la dernière ligne du tableau de vérité de tous ces opérateurs binaires, et que le résultat est dans les trois cas égal à ?, on en déduit que $\hat{q}(\phi) = \hat{q}\,(\phi_1 \Diamond \phi_2) =?$

$\boxed{\text{Il n'existe pas de tautologie dans la logique tri-valuée associée aux opérateurs } \{\neg, \wedge, \vee, \to\}}$

I.9. Cette question est inintelligible ! Et cela pour plusieurs raisons, dont :

- la proposition à étudier n'est pas une formule logique tri-valuée, donc le mot « tautologie » est inadapté.
- l'expression « est équivalent à » n'est pas définie dans le sujet : cela signifie-t-il $\Leftrightarrow$ au sens de la logique bi-valuée ? en un sens analogue non défini de la logique tri-valuée ? ou au sens de « même valeur de vérité » ?
- la présence de $(\models_3 A \to B)$ est pour le moins étrange au regard de ce qu'on a montré dans la question **I.8.**...

Je pense que la question posée porte sur le problème suivant : on sait qu'en logique bi-valuée, pour démontrer un théorème de la forme $H \Rightarrow C$ il suffit de supposer que H est satisfaite et d'en déduire que C est satisfaite. Cette méthode correspond au *lemme de déduction*, qu'on peut énoncer ainsi (on note $\mathcal{T}_2$ l'ensemble des bi-valuations) :

$$\forall A, B \in \mathcal{F}, \forall f \in \mathcal{T}_2,\ \texttt{si}\ \hat{f} \vdash_2 (A)\ \texttt{entraîne}\ \hat{f} \vdash_2 (B)\ \texttt{alors}\ \hat{f} \vdash_2 (A \Rightarrow B)$$

A contrario, la *règle du modus ponens* exprime le fait que si un théorème du type $H \Rightarrow C$ est démontré, il suffit de vérifier H (l'hypothèse) pour pouvoir affirmer que C (la conclusion) est vérifiée, ce qui s'exprime par :

$$\forall A, B \in \mathcal{F}, \forall f \in \mathcal{T}_2,\ \texttt{si}\ \hat{f} \vdash_2 (A \Rightarrow B)\ \texttt{alors}\ \hat{f} \vdash_2 (A)\ \texttt{entraîne}\ \hat{f} \vdash_2 (B)$$

On peut donc exprimer une équivalence (au sens où chacune entraîne l'autre) entre les propositions :

$$\left(\hat{f} \vdash_2 (A)\ \texttt{entraîne}\ \hat{f} \vdash_2 (B)\right) \text{ et } \left(\hat{f} \vdash_2 (A \Rightarrow B)\right)$$

Le but de cette question est donc d'étudier si cette équivalence reste valable en tri-valuation et avec le nouvel opérateur d'implication. Autrement dit, le problème est d'examiner l'énoncé suivant :

$$\forall A, B \in \mathcal{F}, \forall f \in \mathcal{T}_3, \left(\hat{f} \vdash_3 (A)\ \texttt{entraîne}\ \hat{f} \vdash_3 (B)\right)\ \texttt{si et seulement si}\ \left(\hat{f} \vdash_3 (A \to B)\right)$$

qui peut être reformulée à l'aide des opérateurs classiques $\Rightarrow$ et $\Leftrightarrow$ de la logique bi-valuée « ordinaire » :

$$\forall A, B \in \mathcal{F}, \forall f \in \mathcal{T}_3, \left(\hat{f} \vdash_3 (A) \Rightarrow \hat{f} \vdash_3 (B)\right) \Leftrightarrow \left(\hat{f} \vdash_3 (A \to B)\right)$$

Or il est clair en considérant $A, B \in \mathcal{V}$ et $f \in \mathcal{T}_3$ telle que $f(A) =?$ et $f(B) = \bot$ que la proposition $\left(\hat{f} \vdash_3 (A)\ \texttt{entraîne}\ \hat{f} \vdash_3 (B)\right)$ est vraie (car $\hat{f} \vdash_3 (A)$ est fausse), tandis que $\left(\hat{f} \vdash_3 (A \to B)\right)$ est fausse (car $\hat{f}(A \to B) =? \neq \top$: voir avant-dernière ligne de la table de vérité de l'opérateur $\to$).

$\boxed{\text{La proposition } \left(\hat{f} \vdash_3 (A)\ \text{entraîne}\ \hat{f} \vdash_3 (B)\right) \text{ n'est pas équivalente à } \left(\hat{f} \vdash_3 (A \to B)\right)}$

Remarque : c'est donc le lemme de déduction qui est mis en défaut en logique tri-valuée (et ceci avec les deux définitions de l'implication proposées dans le sujet). On peut toutefois vérifier que le modus ponens reste valable.

I.10. Apparition tardive des constantes propositionnelles... On définit la formule ψ de la question **I.6.**

```
let psi = Implique (Et (Implique (Var "A", Var "B"), Implique (Non (Var "A"), Var "B")), Var "B");;
```

I.11. L'exemple fourni dans le sujet étant assez court, je choisis d'écrire une fonction a minima, c'est-à-dire sans parenthésage des constantes, variables et négations de constantes ou de variables mais en conservant tous les autres parenthésages. En particulier, les règles de priorité usuelles entre opérateurs ne sont pas implémentées.

```
let rec lectureFormule f =
  (* identification des formules logiques qui ne sont pas parenthésées *)
  let est_simple = function
    |Vrai |Faux |Indetermine |Var _
    |Non Vrai |Non Faux |Non Indetermine |Non (Var _) -> true
    |_ -> false
  in match f with
     Vrai -> "vrai"
    |Faux -> "faux"
    |Indetermine -> "Indéterminé"
    |Var s -> s
    |Non g when est_simple g -> "non "^(lectureFormule g)
    |Non g -> "non ("^(lectureFormule g)^")"
    |Et (g,h) when est_simple g && est_simple h -> (lectureFormule g)^" et "^(lectureFormule h)
    |Et (g,h) when est_simple g -> (lectureFormule g)^" et ("^(lectureFormule h)^")"
    |Et (g,h) when est_simple h -> "("^(lectureFormule g)^") et "^(lectureFormule h)
    |Et (g,h) -> "("^(lectureFormule g)^") et ("^(lectureFormule h)^")"
    |Ou (g,h) when est_simple g && est_simple h -> (lectureFormule g)^" ou "^(lectureFormule h)
    |Ou (g,h) when est_simple g -> (lectureFormule g)^" ou ("^(lectureFormule h)^")"
    |Ou (g,h) when est_simple h -> "("^(lectureFormule g)^") ou "^(lectureFormule h)
    |Ou (g,h) -> "("^(lectureFormule g)^") ou ("^(lectureFormule h)^")"
    |Implique (g,h) when est_simple g && est_simple h ->
                  (lectureFormule g)^" implique "^(lectureFormule h)
    |Implique (g,h) when est_simple g -> (lectureFormule g)^" implique ("^(lectureFormule h)^")"
    |Implique (g,h) when est_simple h -> "("^(lectureFormule g)^") implique "^(lectureFormule h)
    |Implique (g,h) -> "("^(lectureFormule g)^") implique ("^(lectureFormule h)^")"
;;
```

Exemples :

```
#lectureFormule (Et(Var "A",Non (Var "B")));;
- : string = "A et non B"

#lectureFormule psi;;
- : string = "((A implique B) et (non A implique B)) implique B"
```

Partie II. Algorithmique et programmation

II.1. Dans le calcul de la fonction bilan $\mathcal{B}(y)$, la somme de gauche représente la somme de la quantité de flot entrante (de diverses origines) vers le sommet y tandis que la somme de droite représente la somme de la quantité de flot sortante (vers diverses destinations). Il en résulte que cette fonction bilan représente le solde (de type crédit/débit) au sommet y.
Par suite, l'équation (1) signifie que le solde est nul (autant de flot entrant que de flot sortant) en tout sommet différent de la source et du puits, tandis que les soldes à la source et au puits sont opposés en signe et égaux en valeur absolue (l'énoncé ne précisant pas le signe de ν, on ne peut pas en dire plus, même si les exemples introductifs laissent suggérer que $\nu \geq 0$).
L'équation (2) signifie que la quantité de flot sur chaque arc est positive (le flux se déplace dans le sens de l'orientation de chaque arc) mais ne peut excéder la capacité de l'arc (ce qui semble raisonnable).

II.2. $\displaystyle\sum_{u\in S}\mathcal{B}(u) \underset{t_G\notin S}{=} \sum_{u\in S\setminus\{t_G\}}\mathcal{B}(u) \underset{s_G\in S}{=} \mathcal{B}(s_G) + \sum_{u\in S\setminus\{s_G;t_G\}}\mathcal{B}(u) = \mathcal{B}(s_G) = \nu.$

D'autre part on peut écrire :

$$\sum_{u\in S}\mathcal{B}(u) = \sum_{u\in S}\left(\sum_{\{v\in V_G,(u,v)\in E_G\}} x(u,v) - \sum_{\{v\in V_G,(v,u)\in E_G\}} x(v,u)\right)$$

$$= \sum_{u\in S}\sum_{\{v\in V_G,(u,v)\in E_G\}} x(u,v) - \sum_{u\in S}\sum_{\{v\in V_G,(v,u)\in E_G\}} x(v,u) = \sum_{\{(u,v)\in(S\times V_G)\cap E_G\}} x(u,v) - \sum_{\{(v,u)\in(V_G\times S)\cap E_G\}} x(v,u)$$

$$= \sum_{\{(u,v)\in(S\times T)\cap E_G\}} x(u,v) + \sum_{\{(u,v)\in(S\times S)\cap E_G\}} x(u,v) - \sum_{\{(v,u)\in(S\times S)\cap E_G\}} x(v,u) - \sum_{\{(v,u)\in(T\times S)\cap E_G\}} x(v,u)$$

0r, en faisant le changement de variable $(u,v)\to(v,u)$, on voit que les deux sommes centrales sont égales et par conséquent leur différence est nulle. Compte tenu des inégalités $x(v,u)\geq 0$ et $x(u,v)\leq c(u,v)$ on obtient alors :

$$\sum_{\{(u,v)\in(S\times T)\cap E_G\}} x(u,v) - \sum_{\{(v,u)\in(T\times S)\cap E_G\}} x(v,u) \leq \sum_{\{(u,v)\in(S\times T)\cap E_G\}} x(u,v) \leq \sum_{\{(u,v)\in(S\times T)\cap E_G\}} x(u,v) = c(S,T)$$

Ainsi : $\nu = \displaystyle\sum_{u\in S}\mathcal{B}(u) \leq c(S,T)$.

Pour tout flot de valeur ν dans G et pour toute s,t-coupe (S,T) de G on a $\nu \leq c(S,T)$

.

II.3. L'énoncé propose de représenter un réseau de transport G par une matrice d'adjacence. Dans ce contexte, il semble raisonnable de représenter V_G par l'ensemble des entiers $[\![0;|V_G|-1]\!]$ et un réseau de transport par une matrice de flottants de taille $|V_G|-1$ dont l'élément (i,j) contient la capacité $c(i,j)$ lorsque $(i,j)\in E_G$. Comme le signale l'énoncé, on peut indiquer l'absence d'arc d'un sommet à un autre par une capacité nulle. Enfin, une capacité infinie peut être modélisée par la création d'un type somme contenant l'infini et tous les flottants, ou par une capacité négative, ou encore par un nombre constant MAX « très grand ». Pour simplifier la présentation des programmes, c'est cette dernière solution que j'adopte.
Les flots seront également modélisés par une matrice de flottants.
J'associe la source à l'entier 0 et le puits à l'entier $|V_G|-1$.

```
type reseau_transport == float vect vect;;
type flot == float vect vect;;

let G =
  let M = make_matrix 6 6 0. in
    M.(0).(1) <- 2. ; M.(1).(2) <- 3. ; M.(1).(3) <- MAX ;
    M.(2).(3) <- 8. ; M.(2).(4) <- 1. ; M.(2).(5) <- 4. ;
    M.(3).(1) <- 2. ; M.(3).(2) <- 5. ; M.(4).(2) <- 5. ;
    M
;;
```

II.4. Je construis la fonction de bilan $\mathcal{B}$ et je teste les deux équations en deux temps. A noter que le signe de ν n'étant pas spécifié dans l'énoncé je me dispense de vérifier s'il est positif. Le programme est commenté :

```
let flot_admissible g x =
  let n = vect_length g in
    (* calcul de la fonction bilan B *)
    let bilan u =
      let b = ref 0. in for v = 0 to n-1 do b := !b +. x.(u).(v) -. x.(v).(u) done;
      !b
    in try
      (* test de la valeur des bilans à la source et au puits *)
      if bilan 0 +. bilan (n-1) != 0. then raise Exit;
      (* test des bilans aux sommets intermédiaires *)
      for u = 1 to n-2 do if bilan u != 0. then raise Exit done;
      (* test de l'équation (2) *)
      for u = 0 to n-1 do
        for v = 0 to n-1 do
          if x.(u).(v) < 0. || x.(u).(v) > g.(u).(v) then raise Exit
        done
      done;
      true
    with Exit -> false
;;
```

II.5. Il ne me semble pas possible de construire un programme purement récursif, je vais donc me contenter d'un programme récursif auxiliaire encapsulé. Je propose un programme de type DFS sur le réseau de transport, qui s'arrête dès que l'on rencontre le sommet cible et qui privilégie un éventuel arc direct du sommet source au sommet cible. Classiquement, on tient à jour durant le parcours un tableau de booléens dont la cellule n° i indique si le sommet i a déjà été rencontré.
La fonction récursive `aux i j` se place au sommet `i` et examine la possibilité de passer par le sommet `j` pour parvenir jusqu'à `v`. Si le processus est à son début (sommet `j` $= 0$), il y a d'abord un test d'existence d'un arc direct vers `v`. Sinon, on examine dans l'ordre croissant tous les sommets de `g` et lorsqu'on trouve un voisin de `i` pour lequel un chemin vers `v` existe on complète ce chemin en insérant le sommet `i` en tête et le processus s'arrête. Si tous les voisins de `i` sont sollicités en vain (cas `j = n`) on retourne le chemin vide.
Le programme évite par le procédé de marquage les chemins comportant un cycle. Enfin, c'est la fonction `aux` qui construit la liste des sommets du chemin sous la forme d'une pile. Le programme est commenté ci-dessous :

```
let chemin(g,u,v) =
  let n = vect_length g in
  (* création du tableau indiquant les sommets déjà rencontrés *)
  let marquage = make_vect n false in
    let rec aux i j =
      marquage.(i) <- true;
      match j with
        (* on teste d'abord l'existence d'un arc menant à v *)
         0 when g.(i).(v) <> 0. -> [i;v]
        (* recherche infructueuse : on a essayé en vain de passer par chaque voisin *)
        |j when j = n -> []
        (* sommet non voisin ou déjà exploré en vain : on passe au prochain *)
        |j when g.(i).(j) = 0. || marquage.(j) -> aux i (j+1)
        (* sommet voisin non exploré : on cherche s'il y a un chemin de ce voisin à v *)
        (* et dans ce cas on lui ajoute le sommet courant, sinon on passe au prochain *)
        |j -> let l = aux j 0 in if l = [] then aux i (j+1) else i::l
    in
    if u = v then [] else aux u 0
;;
```

II.6. La **Définition 11** a dû laisser perplexe plus d'un candidat ! Il faut probablement comprendre qu'un chemin améliorant est un chemin P de s_G vers t_G tel que pour tout arc (u, v) de P on peut *ou bien* augmenter $x(u, v)$ de α unités *ou bien* diminuer $x(v, u)$ de α unités (ou non exclusif) mais qu'*on ne fait que l'une de ces deux opérations*

quand elles sont toutes les deux possibles (par exemple on augmente $x(u,v)$ par défaut et sinon on diminue $x(v,u)$). Dès lors, la remarque fondamentale pour montrer que x' est encore un flot est de remarquer que pour un arc $(u,v) \in P$, chacune de ces deux opérations a exactement le même effet sur les bilans de u et de v : cela fait augmenter le bilan de u de α et diminuer celui de v de α. En effet, dans la fonction bilan, les flots entrants sont soustraits et les flots sortants sont ajoutés, donc ajouter α à un flot sortant équivaut à soustraire α à un flot entrant.
On peut maintenant vérifier l'équation (1) en calculant la fonction de bilan $\mathcal{B}'$ associée au flot x' sur V_G.
Soit $y \in V_G$: quatre cas peuvent se présenter :

- si y n'est pas traversé par le chemin P, alors pour tout arc (u,v) de E_G ayant y pour origine ou pour extrémité, on a $x'(u,v) = x(u,v)$. Il en résulte que $\mathcal{B}'(y) = \mathcal{B}(y) = 0$.
- si $y = s_G$, point de départ du chemin P, alors il existe un unique sommet v_0 tel que $(s_G, v_0) \in P$ et l'arc (s_G, v_0) est le seul arc sur lequel le flot est modifié, donc d'après la remarque on a : $\mathcal{B}'(s_G) = \mathcal{B}(s_G) + \alpha = \nu + \alpha$.
- si $y = t_G$, point d'arrivée du chemin P, alors il existe un unique sommet u_0 tel que $(u_0, t_G) \in P$ et l'arc (t_G, u_0) est le seul arc sur lequel le flot est modifié, donc d'après la remarque on a : $\mathcal{B}'(t_G) = \mathcal{B}(t_G) - \alpha = -\nu - \alpha$.
- enfin, si y est traversé par P, alors il existe un unique sommet u_0 et un unique sommet v_0 tels que $(u_0, y) \in P$ et $(y, v_0) \in P$, et on obtient comme ci-dessus : $\mathcal{B}'(y) = \mathcal{B}(y) - \alpha + \alpha = 0 - \alpha + \alpha = 0$.

D'autre part, il paraît sous-entendu dans la **Définition 11** que $x'(u,v) = x(u,v)$ et $x'(v,u) = x(v,u)$ pour tous les arcs (u,v) qui n'appartiennent pas au chemin P. Il en résulte donc que l'équation (2) de la **Définition 6** reste vérifiée pour ces arcs-là. De plus, elle reste vérifiée pour les arcs de P par construction de x'.

x' est toujours un flot de G et sa valeur est $\nu + \alpha$

II.7. D'après les conditions imposées au flot x' dans la **Définition 11**, il faut que $x(u,v) + \alpha \leq c(u,v)$, donc $\alpha \leq c(u,v) - x(u,v)$ si (u,v) est parcouru dans le sens direct, et aussi que $x(u,v) - \alpha \geq 0$, donc $\alpha \leq x(u,v)$ si (u,v) est parcouru dans le sens rétrograde. On a donc : $\forall (u,v) \in P, \alpha \leq \min(c(u,v) - x(u,v); x(v,u))$ et par suite :

Pour tout chemin améliorant P dans G, l'augmentation maximale est : $\min_{(u,v)\in P} \{\min(c(u,v) - x(u,v); x(v,u))\}$

II.8. L'énoncé est un peu maladroit. En effet, la **Définition 11** parle bien d'*unités* d'augmentation mais n'insiste pas suffisamment sur le fait que l'on considère ici des chemins améliorants à augmentations entières, ce qui implique, compte tenu de l'initialisation du flot à 0 dans l'algorithme de Ford et Fulkerson, que les valeurs des flots obtenus dans l'algorithme sont toujours des entiers. D'autre part, on parle bien d'*augmentation*, donc $\alpha > 0$, ce qui sous-entend que les valeurs de flot obtenus à l'issue de chaque tour de la boucle **tant que** dans l'algorithme forment une suite strictement croissante à valeurs dans $\mathbb{N}$.
Dire que l'algorithme ne s'arrête pas signifie que cette suite possède une infinité de termes distincts, ce qui n'est possible que si elle n'est pas majorée, car une suite de réels majorée converge et une suite d'entiers qui converge est stationnaire à partir d'un certain rang. Par conséquent, pour prouver la terminaison de l'algorithme de Ford et Fulkerson, il suffit de montrer que l'ensemble des valeurs des flots de G est majoré.

Raisonnons par l'absurde en supposant que l'ensemble des flots de G n'est pas majoré. Alors, d'après la **Question II.2**, toutes les s,t-coupes de G sont de capacité infinie. Montrons que cela implique l'existence d'un chemin orienté de s_G vers t_G composé uniquement d'arcs de capacité infinie, ce qui est contraire à une hypothèse de l'énoncé.

Pour cela, construisons par récurrence finie une suite strictement croissante $S_1 \subsetneq S_2 \subsetneq \cdots \subsetneq S_r$ de sous-ensembles S_k de V_G contenant s_G et tel que pour tout sommet v de S_k différent de s_G il existe un chemin orienté de s_G vers v composé uniquement d'arcs de capacité infinie et dont les sommets sont tous dans S_k.
On arrête le processus dès que $t_G \in S_k$, ce qui se produit au plus tard pour $k = |V_G|$, et on obtient alors l'existence du chemin à capacité infinie cherché.

- Pour $k = 1$, l'ensemble $S_1 = \{s_G\}$ convient mais ne contient pas t_G.
- Pour $k < |V_G|$, supposons construit S_k et supposons que $t_G \notin S_k$. Alors en posant $T_k = V_G \setminus S_k$, on peut réaliser la (S_k, T_k)-coupe de G. Cette coupe étant de capacité infinie, l'un au moins des arcs (u,v) tel que $u \in S_k$, $v \in T_k$ et $(u,v) \in G$ est de capacité infinie. Soit (u_0, v_0) un tel arc. Alors en posant $S_{k+1} = S_k \cup \{v_0\}$, on obtient bien un ensemble contenant S_k strictement, et dont tout élément est l'extrémité d'un chemin à arcs de capacité infinie issu de s_G et ne passant que par des sommets de S_{k+1} : c'était en effet déjà vrai pour tous les éléments de S_k, en particulier pour u_0, et c'est vrai aussi pour v_0 puisque $c(u_0, v_0) = \infty$.

L'algorithme de Ford et Fulkerson s'arrête effectivement

Remarque :
L'hypothèse de non-existence d'un chemin à arcs de capacité infinie depuis la source jusqu'au puits est évidemment

une condition nécessaire d'arrêt de l'algorithme, puisque la valeur ν d'un flot est définie comme un réel (image de la source par la fonction de bilan à valeurs réelles). Il est moins évident que l'hypothèse d'augmentation par unités soit nécessaire. On trouvera cependant sur la page Wikipedia en anglais consacrée à l'algorithme de Ford et Fulkerson un exemple où cette hypothèse n'est pas vérifiée et où l'algorithme ne s'arrête pas.

II.9. L'algorithme initialise le flot à 0 puis l'augmente au minimum d'une unité à chaque tour de la boucle **tant que**. Il en résulte que le nombre N de tours de cette boucle est majoré par le flot maximum supporté par le réseau de transport, qui est lui-même majoré par la capacité de n'importe quelle coupe. En considérant la s,t-coupe définie par $S = \{s_G\}$, qui contient au plus $|V_G| - 1$ arcs, on constate donc que $N \leq C(|V_G| - 1)$.
Reste à estimer la complexité de la recherche d'un chemin améliorant. Pour trouver un tel chemin on peut imaginer un algorithme du même type que celui qui est utilisé dans le programme `chemin(g,u,v)` de la **Question II.5.**, modifié en tenant à jour une variable α (initialisée à MAX) et où tout test d'existence d'un arc serait remplacé par un test d'existence d'un arc dont le flot peut être augmenté (en mettant ensuite à jour l'augmentation α du chemin en cours de construction selon que l'augmentation de l'arc testé est inférieure ou supérieure à α). Un tel algorithme est clairement susceptible dans le pire des cas de tester tous les arcs du réseau, il est donc de complexité $O(E_G)$.
En définitive on obtient donc pour l'algorithme de Ford et Fulkerson une complexité dans le pire des cas :

$$\boxed{T = O(C.|V_G|.|E_G|)}$$

II.10. Un résidu est un graphe symétrique qui peut être assimilé à un réseau de transport. On peut donc déclarer :

```
type residu == reseau_transport;;
```

II.11. Il s'agit de mettre à jour une matrice initialisée à 0 à l'aide des égalités de l'énoncé à partir de la matrice du réseau de transport G et de celle du flot x. Cela se prête évidemment très bien à une programmation itérative. On peut donc s'étonner de l'obligation d'utiliser une méthode récursive, après avoir laissé au candidat le choix de la structure de données. Allons-y quand même, sans gaîté de coeur...

```
let graphe_residu g x =
  let n = vect_length g in let res = make_matrix n n 0.
  in
  let rec remplit i j =
    match (i,j) with
       (i,j) when i = n -> ()
      |(i,j) when i = j -> remplit (i+1) 0
      |(i,j) -> begin
                  res.(i).(j) <- g.(i).(j) -. x.(i).(j) +. x.(j).(i);
                  res.(j).(i) <- g.(j).(i) -. x.(j).(i) +. x.(i).(j);
                  remplit i (j+1)
                end
  in
  remplit 0 0;
  res
;;
```

II.12. On a pour tout arc (u, v) dans le résidu : $r(u, v) = \underbrace{c(u, v) - x(u, v)}_{\geq 0} + \underbrace{x(v, u)}_{\geq 0} \geq 0$.

$$\boxed{\text{On a toujours } r(u, v) \geq 0}$$

II.13. On considère le réseau de transport G tel que $V_G = \{s_G; t_G\}$ avec $c(s_G, t_G) = 1$ et $c(t_G, s_G) = 0$, et le flot admissible x défini par $x(s_G, t_G) = 1$ et $x(t_G, s_G) = 0$. Alors $r(t_G, s_G) = 1 > 0 = c(t_G, s_G)$.

$$\boxed{\text{Il est possible d'avoir } r(u, v) > c(u, v)}$$

II.14. Cette question manque de précision sur la définition d'un chemin améliorant dans $G[x]$, en supposant déjà acquise l'interprétation du résidu qu'elle est censée faire appréhender !
Sous la notion de « chemin améliorant P dans $G[x]$ » il faut comprendre, comme on le voit dans l'algorithme 2, un chemin dont tous les arcs possèdent un résidu (c'est-à-dire une réserve d'amélioration) strictement positif.

En revanche, comme on l'a vu à la question précédente, un résidu strictement positif sur un arc de P dans $G[x]$ ne fournit pas toujours une augmentation permise du flot x dans G.
Supposons qu'un arc $(u,v) \in P$ vérifie $r(u,v) \geq \alpha$ et notons $\beta = \min\{\alpha, c(u,v) - x(u,v)\}$.
Alors $x(v,u) = r(u,v) - (c(u,v) - x(u,v)) = r(u,v) - \beta \geq \alpha - \beta$. Il est donc possible d'augmenter le flot sur l'arc (u,v) de la quantité positive ou nulle β et de diminuer le flot sur l'arc (v,u) de la quantité positive ou nulle $\alpha - \beta$ en respectant les contraintes (2), et ceci indépendamment du fait que $c(u,v) \leq x(u,v) + \alpha$ ou pas.
La stratégie d'augmentation de la valeur du flot dans G est alors de poser $x'(u,v) = x(u,v)$ et $(x'(v,u) = x(v,u)$ si (u,v) n'est pas concerné par P et de poser pour tous les arcs (u,v) de P pris dans le sens direct :

$$x'(u,v) = x(u,v) + \beta \text{ et } x'(v,u) = x(v,u) - (\alpha - \beta)$$

Il reste à vérifier que x' est bien un flot vérifiant les contraintes (1), dont la valeur est augmentée de α.
Soit $y \in V_G$. On reprend le schéma de la **Question II.6.** et on examine quatre cas :

- si y n'est pas traversé par le chemin P, alors pour tout arc (u,v) de E_G ayant y pour origine ou pour extrémité, on a $x'(u,v) = x(u,v)$. Il en résulte que $\mathcal{B}'(y) = \mathcal{B}(y) = 0$.
- si $y = s_G$ alors il existe un unique sommet v_0 tel que $(s_G, v_0) \in P$ et les arcs $(s_G, v_0), (v_0, s_G)$ sont les seuls arcs pris en compte dans le calcul de $\mathcal{B}'(s_G)$ pour lequel $x'(u,v) \neq x(u,v)$, le premier étant pris dans le sens direct, donc on a :
  $\mathcal{B}'(s_G) = \mathcal{B}(s_G) + x'(s_G, v_0) - x(s_G, v_0) - (x'(v_0, s_G) - x(v_0, s_G)) = \mathcal{B}(s_G) + \beta + (\alpha - \beta) = \mathcal{B}(s_G) + \alpha$.
- si $y = t_G$ alors il existe un unique sommet u_0 tel que $(u_0, t_G) \in P$ et les arcs $(t_G, u_0), (u_0, t_G)$ sont les seuls arcs pris en compte dans le calcul de $\mathcal{B}'(t_G)$ pour lequel $x'(u,v) \neq x(u,v)$, le second étant pris dans le sens direct, donc on a :
  $\mathcal{B}'(t_G) = \mathcal{B}(t_G) + x'(t_G, u_0) - x(t_G, u_0) - (x'(u_0, t_G) - x(u_0, t_G)) = \mathcal{B}(s_G) - (\alpha - \beta) - \beta = \mathcal{B}(t_G) - \alpha$.
- enfin, si y est traversé par P, alors il existe un unique sommet u_0 et un unique sommet v_0 tels que $(u_0, y) \in P$ et $(y, v_0) \in P$, ces deux arcs étant simultanément pris en sens direct ou rétrograde et on obtient alors : $\mathcal{B}'(y) - \mathcal{B}(y) = -\alpha + \alpha$ (ou $+\alpha - \alpha) = 0$.

Conclusion : Pour un arc (u,v) dans G, le résidu $r(u,v)$ dans $G[x]$ représente la réserve d'augmentation de la valeur du flot que permet cet arc s'il est emprunté dans le sens direct dans G, en combinant une augmentation du flot $x(u,v)$ et une diminution du flot $x(v,u)$.
Cette statégie correspond dans l'algorithme 2 à l'étape `MiseAJour` de la Phase 2.

II.15. Si la condition `Marque[t]` est vérifiée, cela signifie qu'il a été trouvé dans le résidu $G[x]$ un chemin de la source au puits dont la valuation de chaque arc est strictement positive, ce qui correspond à un chemin améliorant dans $G[x]$, donc permettant de construire une augmentation de la valeur du flot dans G par la stratégie décrite à la **Question II.14.**
Plus précisément, P représente le chemin en question et α est l'augmentation maximale possible de la valeur du flot à l'aide du chemin P.

II.16. La boucle principale s'arrête lorsque la condition `Marque[t]` n'est plus vérifiée, donc $t_G \in T$. Par ailleurs, au début du dernier tour de cette boucle, la marque de s_G est mise à vrai et n'est plus modifiée, donc $s_G \in S$. Comme S et T sont par ailleurs complémentaires dans V_G, on peut conclure que :

$$\boxed{(S,T) \text{ est une s,t-coupe de } G}$$

D'autre part, on voit dans l'instruction conditionnelle de la phase 1 que tout élément marqué fait un passage dans l'ensemble S de l'algorithme 2. et est donc choisi à un certain tour de la boucle principale de la phase 1. Par conséquent, s'il existait un couple $(u_0, v_0) \in S \times T$ tel que $r(u_0, v_0) > 0$, u_0 aurait été choisi à un certain moment, et la condition de l'instruction conditionnelle de la Phase 1 aurait été vérifiée pour le couple (u_0, v_0), ce qui aurait eu pour effet de marquer v_0, ce qui n'est pas le cas. D'où :

$$\boxed{\forall (u,v) \in S \times T, r(u,v) = 0}$$

Par définition, $r(u,v) = c(u,v) - x(u,v) + x(v,u)$.
Par conséquent, si (u,v) est un arc de S vers T, on a $r(u,v) = 0 \Longrightarrow c(u,v) - x(u,v) = -x(v,u)$.
Or les contraintes (2) entraînent que le premier membre de cette égalité est positif tandis que le second est négatif. Ils sont donc tous deux nuls. Comme (u,v) est un arc de S vers T équivaut à (v,u) est un arc de T vers S on a bien :

$$\boxed{\text{Pour tout couple } (u,v) \in S \times T \text{ on a } x(u,v) = c(u,v) \text{ et } x(v,u) = 0}$$

II.17. Dans la **Question II.2.** on a vu que : $\nu = \sum_{u \in S} \mathcal{B}(u)$, puis que :

$$\sum_{u \in S} \mathcal{B}(u) = \sum_{\{(u,v) \in (S \times T) \cap E_G\}} x(u,v) - \sum_{\{(v,u) \in (T \times S) \cap E_G\}} x(v,u)$$

Or, d'après la question précédente, la deuxième des sommes ci-dessus est nulle et la première vaut $c(S,T)$.
On peut donc conclure :

$$\boxed{\nu = c(S,T)}$$

II.18. L'énoncé est maladroit en ne précisant pas clairement les données du problème. Je vais supposer qu'*on se donne préalablement* la partition (A,B) et qu'on cherche un couplage de cardinal maximum dont chaque arête relie un sommet de A à un sommet de B. Soit un graphe G non orienté (que l'on peut donc considérer comme un graphe orienté symétrique) et une partition (A,B) de V_G. A cette partition (A,B) on associe un réseau de transport G' de la façon suivante :

- $V_{G'} = V_G \cup \{s,t\}$: on ajoute aux sommets de G une source et un puits.
- $E_{G'} = E_G \cup \{(s,a), a \in A\} \cup \{(b,t), b \in B\}$: on ajoute aux arêtes (vues comme orientées) déjà existantes dans G une arête de la source à tout sommet dans A, et de tout sommet dans B vers le puits.
- $\forall (u,v) \in E_{G'}, ((u = s)$ ou $(v = t)$ ou $(u \in A$ et $v \in B)) \Longrightarrow c(u,v) = 1$ sinon $c(u,v) = 0$: on n'autorise le flot que dans les arêtes de la source vers un sommet de A ou d'un sommet de A vers un sommet de B ou d'un sommet de B vers le puits.

Si on se limite comme précédemment à des flots à valeurs entières, la quantité de flot dans un arc de G' ne peut donc valoir que 0 ou 1. De plus, tout chemin P de s à t dans G' est de la forme $P = \{(s,a),(a,b),(b,t), a \in A, b \in B\}$ et est donc entièrement déterminé par le couple (a,b). Enfin, pour un sommet a de A il ne peut pas y avoir deux sommets distincts b et b' de B pour lesquels existent simultanément un flot non nul de a vers b et de a vers b', car sinon il y aurait deux unités de flot sortant de a et au plus une unité entrant en a (en provenance de s), ce qui enfreint la contrainte (1) de nullité de la fonction bilan en a. De même, sur un élément b de B ne peut arriver au plus qu'une unité de flot en provenance d'un élément de A. On en déduit que tout flot sur G' est bijectivement associé à un ensemble d'arêtes deux à deux non adjacentes à origine dans A et extrémité dans B, c'est-à-dire à un couplage de G. De plus, la valeur de ce flot est égal au nombre d'arêtes issues de s dans lesquelles passe un flot non nul, qui est égal au nombre d'éléments du couplage. On peut donc conclure :

Tout flot dans le réseau de transport G' défini ci-dessus est bijectivement associé à un couplage de G relatif à la partition (A,B), dont le cardinal est égal à la valeur du flot

Ainsi, pour trouver un couplage maximum de G, il suffit de chercher un flot maximum pour G' puis de récupérer les arêtes de type $(a,b), a \in A, b \in B$ dans lesquelles la quantité de flot est non nulle.

Le problème du couplage maximal de G relatif à (A,B) est équivalent au problème du flot maximal dans G'

II.19. D'après la question précédente, pour implémenter la recherche d'un couplage maximum dans un graphe non orienté G associé à une partition (A,B) de V_G, on peut construire le réseau de transport associé G' puis, en partant d'un flot initialement nul, appliquer l'algorithme 2. à la recherche d'un flot maximal pour G'.
Dans ce contexte, l'algorithme 2. est grandement simplifié par le fait que le flot et le résidu ne peuvent contenir que les valeurs 0 ou 1. Par suite, s'il existe un chemin améliorant dans le résidu, il conduit nécessairement à une augmentation du flot d'une unité, et pour un arc de ce chemin le flot sera soit augmenté d'une unité sur cet arc s'il n'est pas déjà égal à 1 ou diminué d'une unité sur l'arc rétrograde dans le cas contraire, comme on l'a vu à la **Question II.14.**.
Quant à la recherche d'un tel chemin améliorant, on peut l'effectuer à l'aide de la fonction `chemin(g,u,v)` de la **Question II.5.**, quitte à adapter un peu son exploitation (dans le cadre de la sous-fonction `MiseAJour` de l'algorithme 2.) car on obtient une liste de sommets alors que l'algorithme 2. manipule une liste d'arcs. Le programme est commenté ci-dessous, il comporte plusieurs sous-programmes récursifs.

```
let couplage_max G A B =
  (* A et B sont des vecteurs de sommets représentant la partition,
     k est le nombre de sommets de A, n-k celui de B et n celui de G*)
  let n = vect_length G and k = vect_length A in
    (* Construction du flot x et du réseau de transport G' : la source est affectée du n°0
       le puits du n° (n+2), les sommets de A des n° 1 à k et ceux de B des n° k+1 à n+1 *)
    let x = make_matrix (n+2) (n+2) 0. and Gprime = make_matrix (n+2) (n+2) 0. in
      for a = 1 to k do Gprime.(0).(a) <- 1. done;
      for b = 1 to n-k do Gprime.(k+b).(n+1) <- 1. done;
      for a = 1 to k do
        for b = 1 to n-k do
          if G.(A.(a-1)).(B.(b-1)) <> 0. then Gprime.(a).(k+b) <- 1.;
        done
      done;
    (* construction du résidu associé associé au flot x et au réseau de transport G' *)
    let residu = graphe_residu Gprime x in
      (* réalise la mise à jour du flot et du résidu de la phase 2 de l'algorithme 2 *)
      let MiseAJour P =
        (* mise à jour pour un arc du chemin *)
        let modifie_flot_residu u v =
            if x.(u).(v) = 0. then x.(u).(v) <- 1. else x.(v).(u) <- 0.;
            residu.(u).(v) <- Gprime.(u).(v) -. x.(u).(v) +. x.(v).(u);
            residu.(v).(u) <- Gprime.(v).(u) -. x.(v).(u) +. x.(u).(v);
            v
        (* puis mise à jour sur tout le chemin grâce à la fonction it_list *)
        in it_list modifie_flot_residu (hd P) (tl P)
      in
      (* Mise en oeuvre de l'Algorithme 2 par utilisation du residu *)
      let rec Ford_Fulkerson () =
        let P = chemin(residu,0,n+1) in
          if P = [] then ()
          else
            begin
              MiseAJour P;
              Ford_Fulkerson ()
            end
       in
       let rec couplage a b l =
       (* recherche des arêtes du couplage obtenu par recensement des arêtes de A vers B
          à flot non nul et retour à la numérotation des sommets dans le graphe G initial *)
         match (a,b) with
           (a,b) when a = k+1 -> l
           |(a,b) when b = n-k+1 -> couplage (a+1) 1 l
           |(a,b) when x.(a).(k+b) = 0. -> couplage a (b+1) l
           |(a,b) -> couplage a (b+1) ((A.(a-1),B.(b-1))::l)
       in
       Ford_Fulkerson ();
       couplage 1 1 []
;;
```

Par exemple :

```
let test =
  let t = make_matrix 5 5 0. in
    t.(0).(1) <- 1.; t.(1).(0) <- 1.;  t.(2).(1) <- 1.; t.(1).(2) <- 1.;
    t.(2).(3) <- 1.; t.(3).(2) <- 1.;  t.(2).(4) <- 1.; t.(4).(2) <- 1.;
    t.(3).(4) <- 1.; t.(4).(3) <- 1.;  t.(0).(2) <- 1.; t.(2).(0) <- 1.;
    t;;
couplage_max test [|1;2|] [|0;3;4|];;
- : (int * int) list = [2,3;1,0]
```