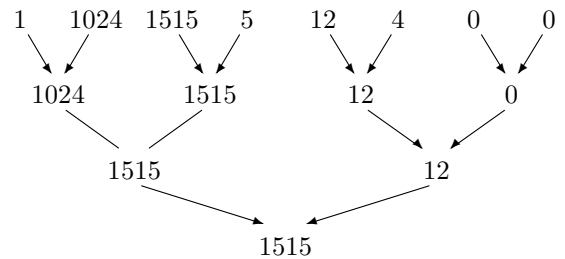
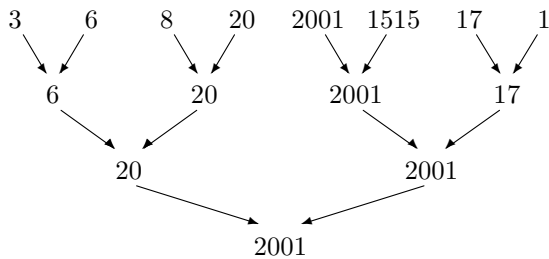


Partie I - Algorithmes pour la sélection

I.A - Recherche des deux plus grands éléments

I.A.1) Utilisation d'un arbre pour réaliser un tournoi.

- a) Il s'agit de calculer les étiquettes de tous les nœuds de hauteur $h \in \llbracket 0; p-1 \rrbracket$, chacun nécessitant une comparaison. Il y a donc $1 + 2 + \dots + 2^{p-1} = 2^p - 1 = \boxed{n - 1 \text{ comparaisons.}}$
- b) Le maximum m se trouve bien sûr à la racine et son calcul nécessite $n - 1$ comparaisons. Pour trouver le second plus grand élément, on cherche le plus grand des fils autres que m de tous les nœuds étiquetés par m (ce qu'on peut démontrer par récurrence en raisonnant sur la hauteur de l'arbre). Pour cela, on doit donc descendre dans l'arbre jusqu'à une feuille et on effectue $p - 1$ comparaisons. Or $p = \ln_2 n$, donc au total, on a effectué moins de $n + \ln_2 n$ comparaisons.
- c) - d) Dans le cas du deuxième tableau, on rajoute des 0 pour avoir un tableau ayant pour taille $2^3 = 8$, ce qui ne modifie pas les deux plus grands éléments, les e_i étant supposés > 0 . En cherchant le plus grand fils autre que le maximum de chaque nœud ayant même étiquette que la racine, on trouve 1515 et 1024 comme deuxième plus grand élément respectif de chaque tableau.



I.A.2) Nombre minimum de comparaisons pour déterminer le maximum.

- a) Raisonnons par récurrence sur le nombre p d'arêtes du graphe.
- . Si $p = 0$, comme (S, A) est connexe, on a nécessairement $|S| = 1$, donc $|A| \geq |S| - 1$.
 - . Supposons que tout graphe connexe ayant au plus $p - 1$ arêtes a un nombre de sommets inférieur ou égal au nombre d'arêtes plus une.
- Soit $G = (S, A)$ un graphe connexe ayant p arêtes. Choisissons une arête a joignant deux sommets i et j et considérons le graphe $G' = (S, A \setminus \{a\})$.
- . S'il est connexe, alors d'après l'hypothèse de récurrence, $|A'| \geq |S| - 1$, donc $|A| = |A'| + 1 \geq |S| > |S| - 1$.
 - . Sinon, G' a deux composantes connexes, une (S_1, A_1) contenant i , l'autre (S_2, A_2) contenant j . D'après l'hypothèse de récurrence, $|A_1| \geq |S_1| - 1$ et $|A_2| \geq |S_2| - 1$, d'où $|A'| \geq |S_1| + |S_2| - 2 = |S| - 2$ et comme $|A'| = |A| - 1$, on obtient $|A| \geq |S| - 1$.

Autre solution : montrons la propriété par récurrence sur le nombre n de sommets.

- . Si $n = 1$, alors $A = \emptyset$ et donc $|A| = 0 \geq |S| - 1$.
 - . Supposons la propriété vraie pour un graphe ayant $n - 1$ sommets.
- Appelons degré d'un sommet d'un graphe G le nombre d'arêtes ayant pour extrémité ce sommet.
- Lorsque G est connexe, chaque sommet a un degré ≥ 1 .
- S'il existe un sommet s de degré 1 et si a est l'arête arrivant en s , alors $G' = (S \setminus \{s\}, A \setminus \{a\})$ est connexe et a $n - 1$ sommets. D'après l'hypothèse de récurrence, $|A| - 1 \geq (|S| - 1) - 1$, donc $|A| \geq |S| - 1$.
 - si tous les sommets ont un degré ≥ 2 , alors on a directement $2|A| \geq 2|S|$, donc $|A| \geq |S| - 1$.
- b) D'après l'hypothèse faite sur e , A trouve le plus grand élément e_{i_0} avec au plus $n - 2$ comparaisons, ce qui signifie que le graphe $(\llbracket 1; n \rrbracket, C)$ a au plus $n - 2$ arêtes, donc il n'est pas connexe d'après le a). Cependant e_{i_0} a pu être comparé avec tous les autres pour savoir que c'est le plus grand. Donc tous sommets e_i autre que e_{i_0} est relié e_{i_0} par un chemin. Il en résulte que deux sommets distincts e_i et e_j peuvent être reliés par un chemin passant par e_{i_0} , ce qui contredit la non-connexité du graphe.

- c) Tout algorithme prenant en entrée n entiers et retournant le plus grand de ces entiers exécute au minimum $n - 1$ comparaisons (ce qui semble être une évidence a priori).

I.B - Recherche systématique dans un tableau

I.B.1) La recherche du minimum nécessite $n - 1$ comparaisons, celle du second plus grand élément en nécessite $n - 2$, etc..., celle du i -ème en nécessite $n - i$.

Le nombre $C(i)$ de comparaisons effectuées est donc égal à $\sum_{k=1}^i (n - k) = ni - \frac{i(i+1)}{2} = \frac{2ni - i^2 - i}{2}$.

$\frac{dC(i)}{di} = n - i - \frac{1}{2}$, donc $C(i)$ est une fonction croissante de i pour $i \leq n/2$.

Si n est pair, le maximum est obtenu pour $i = n/2$ et vaut $\frac{3n^2 - 2n}{8}$.

Si n est impair, le maximum est obtenu pour $i = (n - 1)/2$ et vaut $\frac{3n^2 - 4n + 1}{8}$.

Remarque : si $i > n/2$, alors il est moins coûteux de chercher le $n - i + 1$ -ième plus grand élément.

I.B.2) Si on effectue un tri dichotomique du tableau T , dont le nombre de comparaisons est en $O(n \ln n)$, alors on obtient immédiatement le i -ème élément pour i quelconque dans la i -ème position du tableau T .

I.C - Tri rapide dans une liste chaînée

I.C.1) Fonction partition

```
let partition liste pivot =
  let rec aux l avant apres nb = match l with
    | [] -> (avant, apres, nb)
    | t::q -> if t <= pivot then aux q (t::avant) apres (nb+1)
              else aux q avant (t::apres) nb
  in aux liste [] [] 0 ;;
```

I.C.2) Fonction recherche

```
let rec recherche i liste =
  if liste = [] then failwith "Erreur"
  else
    let pivot = hd liste in
    let (avant, apres, p) = partition (tl liste) pivot in
    if p = i - 1 then pivot
    else if p < i then recherche (i - p - 1) apres
    else recherche i avant ;;
```

- b) La terminaison de cette fonction est immédiate car les tailles des listes des appels successifs diminuent strictement. Après l'appel de la fonction **partition**, le **pivot** (qui était le premier élément de la liste) a exactement p éléments qui lui sont inférieurs, donc a pour rang $p + 1$.

. Si $i = p + 1$, alors le i -ième plus petit élément de la liste initiale est égal au **pivot**.

. Si $p < i - 1$, alors i -ième plus petit élément de la liste initiale est égal au $i - (p + 1)$ -ième élément situé dans la liste **apres** (on supprime les p éléments de la liste **avant** ainsi que le **pivot**).

. Si $p > i$, alors i -ième plus petit élément de la liste initiale est égal au i -ième élément situé dans la liste **avant**.

- c) Le meilleur des cas est celui où le i -ième plus grand élément est placé au départ en tête de la liste : il n'y a alors qu'un seul appel de la fonction **partition** qui utilise $n - 1$ comparaisons.

Les cas les plus coûteux sont ceux pour lesquels les tailles successives des sous-listes ne diminuent que d'une unité à la fois, c'est à dire lorsque qu'après chaque appel de la fonction **partition**, l'une des sous-listes **avant** ou **apres** est vide, ce qui se produit en particulier si la liste initiale est déjà triée dans l'ordre croissant ou dans l'ordre décroissant. Si par exemple $i = 1$ et la liste est triée dans l'ordre décroissant, alors chaque liste **apres** est vide et les tailles successives

des listes **avant** sont $n - 1, n - 2, \dots, 1, 0$, ce qui conduit à $\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$ comparaisons.

- d) Si la liste est formée de tous les entiers d'un intervalle $\llbracket p; q \rrbracket$, on peut prendre comme pivot la partie entière de la moyenne de tous les éléments de la liste.

La fonction partition renvoie alors des listes **avant** et **après** elles-mêmes uniformes.

Le coût du calcul de cette moyenne est de même ordre de grandeur que celui de la fonction **partition**.

Ainsi chaque appel récursif s'applique sur une liste dont la taille est diminuée de moitié.

Mais cela ne fournit pas une amélioration dans tous les cas, seulement en moyenne vraisemblablement.

I-D - Recherche de l'élément médian

I.D.1) En utilisant une fonction de fusion.

- a) On détermine d'abord les m premiers éléments du tableau fusionné qui à la sortie de la boucle sont (en désordre) : $T_0, \dots, T_{i-1}, T_m, \dots, T_{j-1}$. L'élément médian μ est alors le suivant, donc le minimum de t_i et t_j .

```
let median t =
  let m = vect_length t / 2 in
  let i = ref 0 and j = ref m in
  while !i + !j < 2 * m do
    if t.(!i) < t.(!j) then i := !i + 1 else j := !j + 1
  done;
  min t.(!i) t.(!j) ;;
```

- b) Le nombre de comparaisons est égal à $m + 1$.

I.D.2) Recherche dichotomique.

Cherchons le médian μ des éléments de T dont l'indice $i \in \llbracket g_1; d_1 \rrbracket \cup \llbracket g_2; d_2 \rrbracket$ avec $d_1 - g_1 = d_2 - g_2 = 2^k - 1$, g_1 et g_2 pairs, les deux sous-tableaux $T[g_1..d_1]$ et $T[g_2..d_2]$ étant supposés rangés dans l'ordre croissant.

On pose $m = 2^{k-1}$. Ainsi la partie de T dont on cherche l'élément médian a $4m$ éléments, le médian étant le $(2m + 1)$ -ième plus petit.

On remarque d'abord que si $g_1 = d_1$ (et $g_2 = d_2$), alors μ est le plus grand des deux éléments T_{g_1} et T_{g_2} .

On pose $m_1 = \lfloor (g_1 + d_1)/2 \rfloor$, $m_2 = \lfloor (g_2 + d_2)/2 \rfloor$, $x = T_{m_1+1}$ et $y = T_{m_2+1}$.

Ainsi les 4 intervalles $\llbracket g_1; m_1 \rrbracket$, $\llbracket m_1 + 1; d_1 \rrbracket$, $\llbracket g_2; m_2 \rrbracket$, $\llbracket m_2 + 1; d_2 \rrbracket$ ont tous le même nombre d'éléments égal à m .

- Si $x < y$, alors :



* les m éléments d'indice $i \in \llbracket d_1; m_1 \rrbracket$ sont tous $< x < y$.

les m éléments d'indice $i \in \llbracket d_2; m_2 \rrbracket$ sont tous $< y$.

Il y a donc au moins $2m + 1$ éléments inférieurs à y .

Or l'élément médian μ a exactement $2m$ éléments qui lui sont inférieurs, donc $\mu < y$.

* les m éléments d'indice $i \in \llbracket m_1 + 1; d_1 \rrbracket$ sont tous $\geq x$.

les m éléments d'indice $i \in \llbracket m_2 + 1; d_2 \rrbracket$ sont tous $\geq y > x$.

Il y a donc au moins $2m$ éléments supérieurs ou égaux à x .

Or l'élément médian μ a exactement $2m$ éléments qui lui sont supérieur ou égal, donc $x \leq \mu$.

Ainsi, si on supprime les éléments d'indice $i \in \llbracket g_1; m_1 \rrbracket \cup \llbracket m_2 + 1; d_2 \rrbracket$, alors on supprime autant d'éléments situés avant μ que d'éléments situés après μ . Donc μ est encore égal au médian des éléments d'indices $i \in \llbracket m_1 + 1; d_1 \rrbracket \cup \llbracket g_2; m_2 \rrbracket$.

- Si $x > y$, alors, de même μ est le médian des éléments d'indice $i \in \llbracket g_1; m_1 \rrbracket \cup \llbracket m_2 + 1; d_2 \rrbracket$.

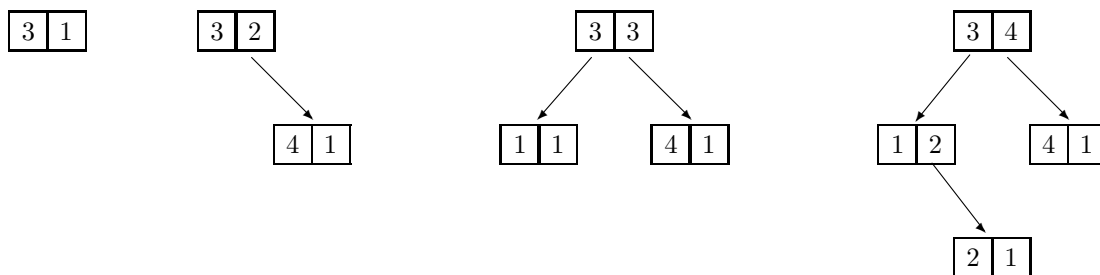
```
let median t =
  let n = vect_length t in
  let m = n / 2 in
  let rec recherche_dicho g1 d1 g2 d2 =
    if g1=d1 then max t.(g1) t.(g2)
    else
      let m1 = (g1+d1)/2 and m2 = (g2+d2)/2 in
      let x=t.(m1+1) and y=t.(m2+1) in
      if x < y then recherche_dicho (m1+1) d1 g2 m2
      else recherche_dicho g1 m1 (m2+1) d2
  in recherche_dicho 0 (m-1) m (n-1) ;;
```

c) La validité du programme précédent a été prouvée au a).

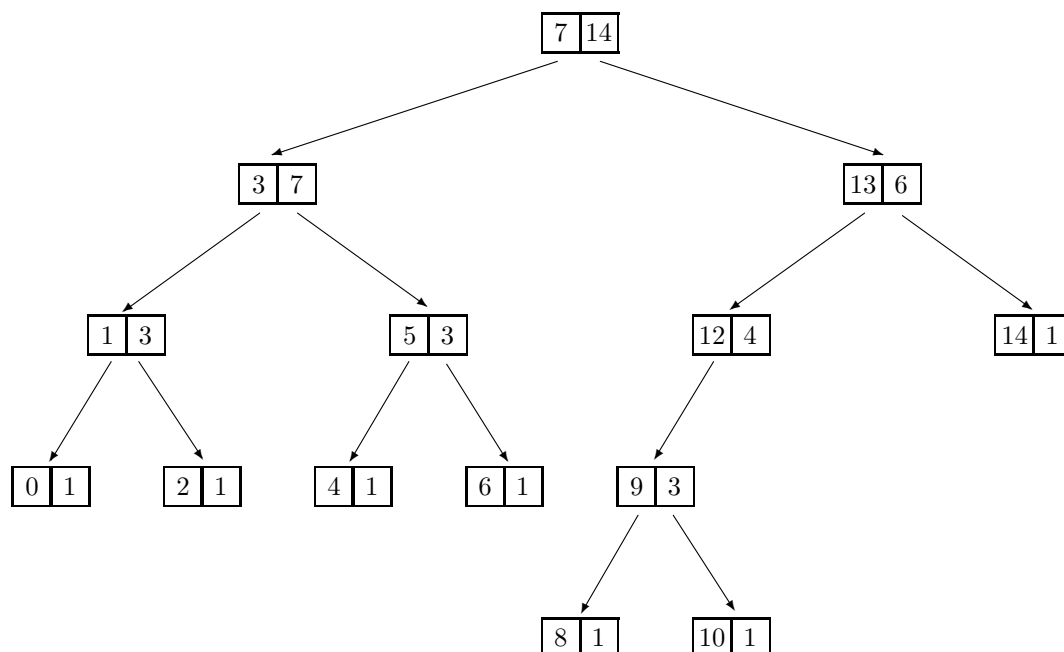
d) Chaque étape coûte une comparaison et diminue la taille par 2. Il y a donc $p = \ln n$ comparaisons.

I-E - Utilisation d'arbres binaires de recherche

I-E.1) On obtient successivement :



b) L'élément 1 figure deux fois, ce qui contredit l'énoncé qui annonce des listes d'éléments distincts. Je ne tiendrais donc pas compte du second 1 de la liste.



I.E.2) La fonction d'insertion suivante suppose que l'élément à insérer n'est pas déjà dans l'arbre.

```

let rec insere x a = match a with
|   vide -> Noeud {valeur = x; taille = 1; gauche = vide; droit = vide}
| Noeud e -> e.taille <- e.taille + 1;
              if x < e.valeur then e.gauche <- insere x e.gauche
              else e.droit  <- insere x e.droit;
a ;;
  
```

I.E.3)

- Si l'arbre est bien équilibré au sens où toutes les feuilles sont à la hauteur h ou $h - 1$, alors $h = \ln_2 n$ (où n désigne la taille de l'arbre) et la complexité de la fonction `insere` est alors en $O(h)$, donc en $O(\ln_2 n)$.
- Si l'arbre est un "peigne", alors sa hauteur est égale à $n - 1$ et la complexité de la fonction `insere` est alors en $O(n)$.

Ce cas se produit lorsqu'on construit l'arbre en insérant des entiers successifs rangés dans l'ordre croissant (peigne à droite) ou dans l'ordre décroissant (peigne à gauche).

I.E.4) La fonction de recherche dans l'arbre utilise le même principe que celle du I.C.2), la racine jouant le rôle de pivot.

```
let taille_arbre a = match a with
| vide -> 0
| Noeud b -> b.taille ;;

let rec recherche i a = match a with
| vide -> failwith "Erreur"
| Noeud e -> let g = taille_arbre e.gauche in
if g = i - 1 then e.valeur
else if g >= i
then recherche i e.gauche
else recherche (i-g-1) e.droit ;;
```

Partie II - Portes logiques universelles

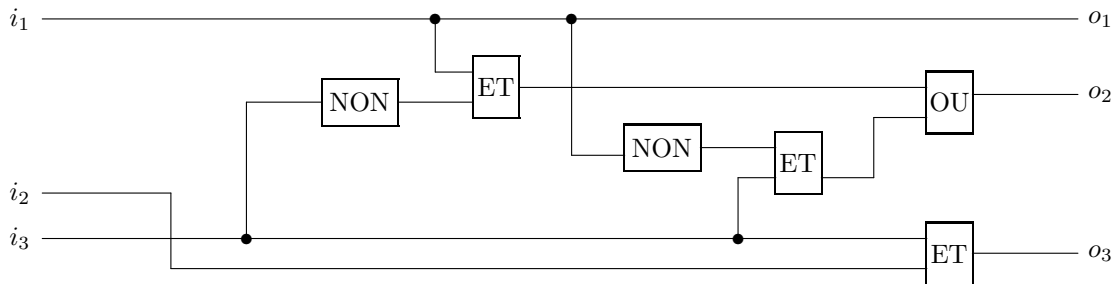
II.A $\underline{o_1 = i_1 \wedge i_2}$ et $\underline{o_2 = i_2 \wedge \neg(i_2 \vee i_3)}$.

II.B - Ensembles de portes universelles

II.B.1) La fonction φ de l'énoncé n'a pas de sens.

Faut-il comprendre que $\varphi : (i_1, i_2, i_3) \mapsto (i_1, i_1 \oplus i_3, i_2 \wedge i_3)$?

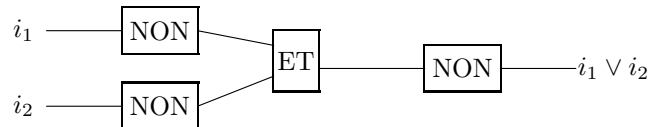
On obtient alors, sachant que $i_1 \oplus i_3 = (i_1 \wedge \neg i_3) \vee (\neg i_1 \wedge i_3)$:



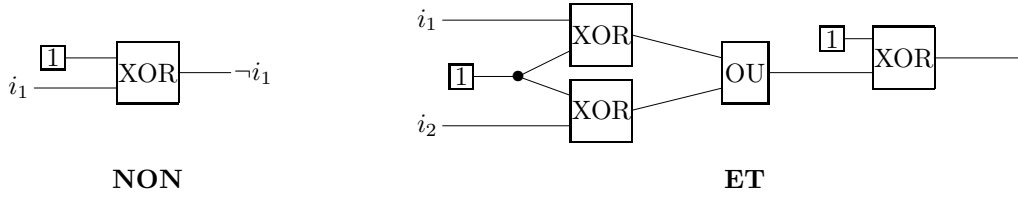
II.B.2) Sachant que l'ensemble $\{\wedge, \vee, \neg\}$ est complet, l'ensemble des portes $\{\text{ET}, \text{NON}, \text{OU}\}$ est universel.

Mais toute porte OU peut être remplacée par un petit circuit faisant intervenir des portes ET et NON seulement, selon la loi de Morgan : $i_1 \vee i_2 \equiv \neg(\neg i_1 \wedge \neg i_2)$. Ainsi le système des portes $\{\text{ET}, \text{NON}\}$ est universel.

Il suffit de remplacer dans C_2 la porte OU par :

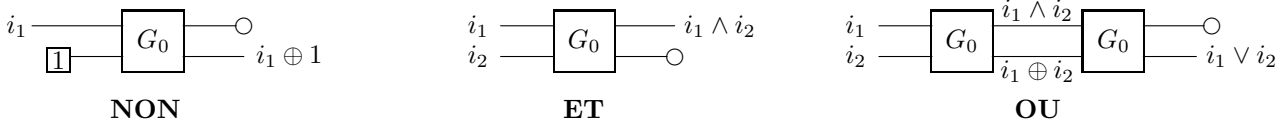


II.B.3) $\underline{\neg i_1 = i_1 \oplus 1}$ et $\underline{i_1 \wedge i_1 = \neg(\neg i_1 \vee \neg i_2)}$. On remplace donc dans C_0 les portes NON et ET par :

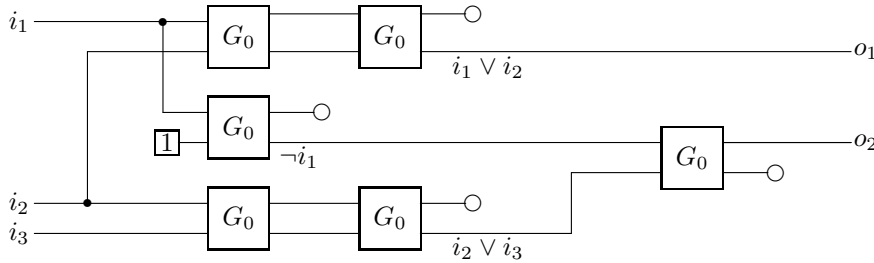


II.B.4) Étude de la porte G_0 .

- a) Montrons que la porte G_0 est universelle. Pour cela, il suffit de montrer que l'on peut construire les portes NON, ET et OU. On vérifie facilement (avec table de vérité) que : $i_1 \vee i_2 = i_1 \oplus i_2 \oplus (i_1 \wedge i_2)$.



c) C_3



- c) G_0 n'est pas inversible, car la fonction associée g n'est pas injective. En effet, les deux entrées $(0, 1)$ et $(1, 0)$ donnent la même sortie $(0, 1)$: en appliquant n'importe quelle porte Q derrière G_0 , l'entrée $(1, 0)$ ne donne qu'une seule sortie. L'assemblage des deux portes G_0 suivie de Q ne calcule donc pas la fonction identité de $\mathcal{B}^2$.

II.C - Portes réversibles

II.C.1) Soit P une porte logique (n, p) réversible et soit Q une porte logique de type (p, n) de sorte que les fonctions f et g respectivement associées vérifient : $g \circ f = Id_{\mathcal{B}^n}$.

Ainsi f est nécessairement injective et g surjective.

Comme f va de $\mathcal{B}^n$ dans $\mathcal{B}^p$, on a nécessairement $n \leq p$.

En outre, si $n = p$, alors f est nécessairement bijective (car la source et le but de f ont le même nombre d'éléments).

Réciproquement, si $n = p$ et si f est bijective, en considérant la porte Q associée à f^{-1} , on voit que P est réversible.

II.C.2) En particulier, si $n = p = 1$, alors $\mathcal{B}^n$ a deux éléments : il existe seulement deux bijections de $\mathcal{B}$ dans lui-même : l'identité et la "transposition", cette dernière étant associée à la porte NON.

II.C.3) Si $n = p = 2$, alors $\mathcal{B}^2$ a $2^2 = 4$ éléments. Il y a donc $4! = 24$ bijections de $\mathcal{B}^2$ dans lui-même, donc 24 portes $(2, 2)$ réversibles.

Parmi elles, il y a des involutions, comme par exemple :

- . $(i_1, i_2) \mapsto (i_2, i_1)$ (l'inverseur).
- . $(i_1, i_2) \mapsto (i_1, \neg i_2)$.
- . $(i_1, i_2) \mapsto (\neg i_1, \neg i_2)$.

II.C.4) Supposons $n \leq p$. Soit f une application injective de $\mathcal{B}^n$ dans $\mathcal{B}^p$. On peut construire $g : \mathcal{B}^p \rightarrow \mathcal{B}^n$ de sorte que $g \circ f = Id_{\mathcal{B}^n}$.

En effet, soit $y \in \mathcal{B}^p$. Si il existe $x \in \mathcal{B}^n$ tel que $y = f(x)$ (x nécessairement unique car f est injective), alors on pose $g(y) = x$. Sinon, on pose $g(y) = x_0$ où x_0 est un élément quelconque fixé de $\mathcal{B}^n$.

Le nombre d'injections de $\mathcal{B}^n$ dans $\mathcal{B}^p$ est $A_{2^p}^{2^n} = 2^p(2^p - 1) \dots (2^p - 2^n + 1)$.

C'est aussi le nombre de portes (n, p) réversibles.

II.D - Portes réversibles universelles

II.D.1) Supposons que G soit une porte réversible $(2, 2)$. Ainsi, la fonction g associée est une bijection de $\mathcal{B}^2$ dans lui-même.

a) Vérifions que $g(i + i') = g(i) + g(i') + g(0)$.

* C'est évident si $i = i'$, car $i + i' = 0$ et $g(i) + g(i') = 0$.

* Si i ou i' est nul, alors par exemple $i' = 0$, donc $g(i + i') = g(i) = g(i) + g(0) + g(0) = g(i) + g(i') + g(i)$.

* Si i et i' sont distincts et non nuls, alors leur somme $i + i'$ est non nulle et distincte de i et i'

(en effet $+$ est commutative et $(1, 0) + (1, 0) = (1, 1)$, $(1, 1) + (1, 0) = (0, 1)$ et $(1, 1) + (0, 1) = (1, 0)$.)

Les 4 éléments $0, i, i', i + i'$ sont distincts deux à deux, donc leurs images par g aussi et forment les 4 éléments de $\mathcal{B}^2$, donc leur somme est nulle. Ainsi $g(i + i') + g(i) + g(i') + g(0) = 0$, donc $g(i + i') = g(i) + g(i') + g(0)$.

b) Remarquons d'abord qu'une fonction $f : (i_1, \dots, i_n) \mapsto (o_1, \dots, o_p)$ de $\mathcal{B}^n$ dans $\mathcal{B}^p$ est affine si et seulement si chaque fonction composante $f_k : (i_1, \dots, i_n) \mapsto o_k$ est affine de $\mathcal{B}^n$ dans $\mathcal{B}$.

D'autre part, les projections canoniques $p_j : (i_1, \dots, i_n) \mapsto i_k$ sont affines, donc toutes les entrées du circuit sont affines en $i = (i_1, \dots, i_n)$.

Montrons que pour toute porte G figurant dans le circuit dont les deux entrées e_1 et e_2 sont affines en $i = (i_1, \dots, i_n)$, alors les deux sorties sont aussi affines en $i = (i_1, \dots, i_n)$.

On pose $F(i) = (e_1(i), e_2(i))$. Ainsi F est affine de $\mathcal{B}^n$ vers $\mathcal{B}^2$ au sens de l'énoncé.

$\forall (i, i') \in \mathcal{B}^n \times \mathcal{B}^n$, $g(F(i + i')) = g(F(i) + F(i') + F(0)) = g(F(i) + F(i')) + g(F(0)) + F(0) = g(F(i)) + g(F(i')) + F(0) + g(F(0)) + F(0)$, donc $(g \circ F)(i + i') = (g \circ F)(i) + (g \circ F)(i') + (g \circ F)(0)$, ce qui montre que les deux sorties de chaque porte G sont des fonctions affines de l'entrée $i = (i_1, \dots, i_n)$ du circuit considéré.

Il en résulte que le circuit calcule une fonction affine de $\mathcal{B}^n$ dans $\mathcal{B}^p$.

c) Or il existe des fonctions non affines de $\mathcal{B}^n$ dans $\mathcal{B}^p$ dès que $n \geq 2$.

En effet, pour en construire une, il suffit de choisir deux éléments non nuls et distincts a et b de $\mathcal{B}^n$ et de s'arranger pour que $f(0) = f(a) = f(b) = (0, \dots, 0)$ et $f(a + b) = (1, \dots, 1)$, ce qui est possible puisque $a + b$ n'est égal ni à 0 , ni à a , ni à b .

Conclusion : il n'existe pas de porte $(2, 2)$ à la fois réversible et universelle.

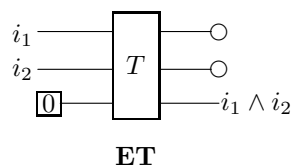
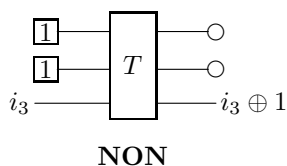
II.D.2) Étude de la porte de Toffoli.

a) Vérifions que T est réversible, c'est à dire que la fonction associée est bijective.

i_1	i_2	i_3	$i_1 \wedge i_2$	o_1	o_2	o_3
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	1	0	0
1	0	1	0	1	0	1
1	1	0	1	1	1	1
1	1	1	1	1	1	0

Les 8 triplets (o_1, o_2, o_3) sont distincts.

Pour montrer que T est universelle, il suffit de montrer, d'après II.B.2.a), que l'on peut obtenir des portes équivalentes à celles du NON et du ET.



b) La fonction f calculée par T n'est pas affine car la troisième composante o_3 ne l'est pas puisque $o_3(0, 0, 0) = o_3(1, 0, 0) = o_3(1, 1, 1) = 0$ et $o_3(0, 1, 1) = 1$, donc $o_3((0, 1, 1) + (1, 0, 0)) = o_3(1, 1, 1) = 0 \neq 1 = o_3(0, 1, 1) + o_3(1, 0, 0) + o_3(0, 0, 0)$.

... FIN ...