INFORMATIQUE - Concours Polytechnique 2000

Première partie. Parties d'un ensemble

Question 1.

```
let rec card = function [] -> 0 |i::q -> 1 + card q;;
```

Question 2.

```
let rec delta p1 p2 =
  match p1,p2 with
    [],_ -> p2
   |_,[] -> p1
   |i1::q1,i2::q2 when i1 = i2 -> delta q1 q2
   |i1::q1,i2::_ when i1 < i2 -> i1::delta q1 p2
   |_,i2::q2 -> i2::delta p1 q2;;
```

La fonction delta parcourt simultanément les deux parties et s'arrête dès que l'une des deux est entièrement parcourue. Dans le pire des cas, les deux parties sont parcourues entièrement, et on a donc bien une complexité bornée par la somme des cardinaux de ces parties.
A chaque étape, les deux éléments les plus petits (en tête des listes) sont comparés. S'ils sont égaux, ils n'apparaissent pas dans la différence symétrique des parties. Sinon, le plus petit ne peut pas apparaître dans l'autre partie, et doit donc être ajouté à la différence symétrique.
Dans tous les cas, on enlève un élément à au moins l'une des parties, et on l'ajoute à la différence symétrique des reliquats.

On peut remarquer à propos de la loi Δ qu'elle est associative, commutative, d'élément neutre $\emptyset$, et que chaque ensemble est son propre symétrique, de telle sorte qu'elle munit l'ensemble des parties d'un ensemble d'une structure de groupe abélien. Ces propriétés seront utilisées dans les questions suivantes.

Question 3.

```
let test p1 p2 =
  let rec imprime_partie = function
    [] -> print_newline()
   |i::q -> printf__printf ''%d '' i;  imprime_partie q
  in imprime_partie (delta p1 p2);;
```

Il s'agit d'un banal parcours de la partie, en imprimant tour à tour tous ses éléments, appliqué ensuite à la différence symétrique des parties en argument. On aurait aussi pu utiliser directement la fonction de bibliothèque do_list.

Deuxième partie. Enumération des parties par incrément

Question 4.

```
let succ p =
  let rec insere j = function
    [] -> [j] |i::q when j < i -> j::i::q |i::q -> insere (j+1) q
  in insere 0 p;;
```

On utilise le principe de l'addition d'une puissance de 2 à un nombre écrit en base 2.
Si l'exposant de cette puissance ne figure pas dans la liste des exposants non nuls de la décomposition du nombre, il suffit de l'insérer dans la liste sans modifier les autres termes. Sinon, on la supprime, et on insère la puissance d'exposant immédiatement supérieur. Au départ, on ajoute 1, donc on insère 0, puis on procède récursivement en fonction de la présence ou non d'une retenue.

Question 5.a)

```
let test_incr N =
  let p1 = ref [] and p2 = ref []
  in
  while card !p1 <> N do
    p2 := succ !p1;
    test !p1 !p2;
    print_newline ();
    p1 := !p2
  done;
  test !p1 [];;
```

On réalise une boucle dont chaque tour traite le passage au successeur de la partie courante.
On commence évidemment avec la partie vide, mais il faut s'arrêter lorqu'on a atteint I_N, ce qu'on peut tester en calculant le cardinal de la partie courante, qui vaut alors N pour la première fois.

Question 5.b)
Appelons x_n le nombre d'interrupteurs à commuter pour réaliser le test sur n interrupteurs.
Pour réaliser ce test sur $n+1$ interrupteurs, on laisse levé le $(n+1)$-ième d'entre eux, et on réalise le test sur les n premiers dans cette configuration (soit x_n commutations), puis on abaisse le $(n+1)$-ième (1 commutation), et on remet ça avec les n premiers (x_n commutations). Enfin on relève le dernier (1 commutation). En définitive, on a donc $x_{n+1} = 2x_n + 2$, ce qui se résout en $x_n + 2 = 2^n (x_0 + 2)$.
Comme $x_0 = 0$, on obtient finalement:

Le nombre d'interrupteurs à commuter pour réaliser ce test sur N interrupteurs est $2^{N+1} - 2$

Troisième partie. Enumération des parties par un code de Gray

Question 6.a)

$T(4) = < 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0 >$

Question 6.b)
$S_0 = \{\}$ $S_1 = \{0\}$ $S_2 = \{0, 1\}$ $S_3 = \{1\}$ $S_4 = \{1, 2\}$ $S_5 = \{0, 1, 2\}$
$S_6 = \{0, 2\}$ $S_7 = \{2\}$ $S_8 = \{2, 3\}$ $S_9 = \{0, 2, 3\}$ $S_{10} = \{0, 1, 2, 3\}$
$S_{11} = \{1, 2, 3\}$ $S_{12} = \{1, 3\}$ $S_{13} = \{0, 1, 3\}$ $S_{14} = \{0, 3\}$ $S_{15} = \{3\}$

Question 7.a)
On peut facilement montrer par récurrence sur $n \geq 1$ que la suite $T(n)$ contient $2^n - 1$ termes, dont chaque valeur figure un nombre pair de fois, sauf la valeur centrale $n - 1$ qui ne figure qu'une fois.
Mais par associativité et commutativité, $S_{2^n-1} = S_0 \triangle \{t_0\} \triangle .. \triangle \{t_{2^n-1}\}$ peut se réordonner en regroupant les parties égales entre elles. Comme la différence symétrique de deux parties égales est vide, il ne reste plus que la partie $\{n - 1\}$ à ne pas être simplifiée. En conclusion:

$$\underline{S_{2^n-1} = \{n - 1\}}$$

Question 7.b)
Si $0 \leq i < 2^n$, $S_{2^n+i} = S_{2^n-1} \Delta \{t_{2^n-1}\} \Delta \{t_{2^n}\} \Delta \{t_{2^n+1}\} \Delta ... \Delta \{t_{2^n+i}\}$.
Or il est clair que $t_{2^n+i} = t_i$. Donc $\{t_{2^n}\} \Delta \{t_{2^n+1}\} \Delta ... \Delta \{t_{2^n+i}\} = S_i$. De plus $t_{2^n-1} = n$.
Enfin, d'après 7.a), on a $S_{2^n-1} = \{n - 1\}$.
En définitive, on a donc $S_{2^n+i} = \{n - 1\} \Delta \{n\} \Delta S_i = \{n - 1, n\} \Delta S_i$.

$$\underline{S_{2^n+i} = \{n - 1, n\} \Delta S_i}$$

Question 7.c)
Montrons par récurrence sur n que $\{S_0, ..., S_{2^n-1}\}$ est l'ensemble P_n des parties de I_n.

- Pour $n = 0$, c'est évident car $\{S_0\} = \{\emptyset\} = P_0$.
- D'après 7.b), $\{S_0, ..., S_{2^{n+1}-1}\} = \{S_0, ..., S_{2^n-1}\} \cup \{\{n-1, n\} \Delta S_0, ..., \{n-1, n\} \Delta S_{2^n-1}\}$.
  Cette réunion est disjointe, car $\forall i < 2^n, n \notin S_i \iff \forall i < 2^n, n \in \{n-1, n\} \Delta S_i$.
  Par hypothèse de récurrence, $\{S_0, ..., S_{2^n-1}\} = P_n = \{q \in P_{n+1}, n \notin q\}$.
  D'autre part, la loi Δ étant une loi de groupe sur P_n, l'application $\Phi : p \mapsto \{n-1\} \Delta p$ est une bijection de P_n sur lui-même.
  Il en résulte que $\{\{n-1, n\} \Delta S_0, ..., \{n-1, n\} \Delta S_{2^n-1}\} = \{\{n\} \Delta p, p \in P_n\} = \{q \in P_{n+1}, n \in q\}$.
  En effectuant la réunion de $\{S_0, ..., S_{2^n-1}\}$ et de $\{\{n-1, n\} \Delta S_0, ..., \{n-1, n\} \Delta S_{2^n-1}\}$, on trouve bien P_{n+1}.

$$\underline{\forall n \in \mathbb{N}, \{S_0, ..., S_{2^n-1}\} = P_n}$$

Question 8.a)

```
let test_gray N =
  let rec T n =
    match n with
       0 -> []
      |_ -> let l = T (n-1) in l@((n-1)::l)
  in
  let f i = printf__printf ''%d '' i; print_newline()
  in
  do_list f (T N);
  printf__printf ''%d '' (N-1);print_newline();;
```

Le principe de cet algorithme est de parcourir les sous-ensembles de I_N dans l'ordre des S_i.
Mais dans ce cas, $S_i \Delta S_{i+1} = S_i \Delta S_i \Delta \{t_i\} = \{t_i\}$. Il suffit donc d'afficher la liste des $(t_i)_{0 \leq i \leq 2^N-2}$, puis d'afficher le dernier commutateur à relever pour revenir à la situation initiale, soit $N-1$.
Etant donné N, on construit d'abord $T(N)$ à l'aide d'une fonction récursive, puis on en affiche les éléments.

Question 8.b)

L'explication de la question précédente montre qu'il y a exactement $2^N - 1 + 1$ interrupteurs à commuter.

$$\underline{\text{Il y a exactement } 2^N \text{ interrupteurs à commuter par cette méthode}}$$

On ne peut pas faire mieux, car il y a exactement 2^N configurations à tester, correspondant toutes à une configuration distincte des interrupteurs, et exigeant par conséquent la commutation d'au moins 1 interrupteur pour passer de l'une à l'autre.

Question 9.a)

Montrons par récurrence sur n que $\forall i \in \left[2^n, 2^{n+1} - 1\right], i$ impair $\Longrightarrow t_i = 1 + \min(S_i)$.

- Pour $n = 0$, on a bien $t_1 = 1 = 1 + 0$ et $S_1 = \{0\}$.
- Supposons la propriété vraie jusqu'au rang n. Soit i impair élément de $\left[2^{n+1}, 2^{n+2} - 1\right]$.
  - Si $i = 2^{n+2} - 1$, alors $S_i = \{n+1\}$ (question 7.a)), tandis que $t_i = n + 2 = 1 + \min(S_i)$.
  - Supposons maintenant $i < 2^{n+2} - 1$. Alors $\forall j \in S_i, j \leq n$.
  D'après la question 7.b), on a $S_i = S_{i-2^{n+1}} \Delta \{n, n+1\}$.
  Or $i - 2^{n+1}$ est impair, et appartient à $\left[0, 2^{n+1} - 2\right]$.
  Par hypothèse de récurrence, on a $t_{i-2^{n+1}} = 1 + \min(S_{i-2^{n+1}})$. Mais on sait que $t_{i-2^{n+1}} = t_i$ (voir question 7.b)).
  Il reste donc à prouver que $\min(S_i) = \min(S_{i-2^{n+1}})$. Or $S_i = S_{i-2^{n+1}} \Delta \{n, n+1\}$.
  Comme $S_{i-2^{n+1}} \neq \{n\}$ (puisque $i \neq 2^{n+2} - 1 \Longrightarrow i - 2^{n+1} \neq 2^{n+1} - 1$), on a d'abord $S_{i-2^{n+1}} \Delta \{n\} \neq \emptyset$.
  Ensuite, $\forall j \in S_{i-2^{n+1}}, j \leq n \Longrightarrow \min(S_{i-2^{n+1}} \Delta \{n\}) = \min(S_{i-2^{n+1}})$.
  Enfin $\min(S_{i-2^{n+1}} \Delta \{n, n+1\}) = \min(S_{i-2^{n+1}} \Delta \{n\})$, ce qui achève la preuve.

$$\underline{\forall i > 0, t_i \text{ impair} \Longrightarrow t_i = 1 + \min(S_i)}$$

Question 9.b)

```
let rec gray = function
   [] -> [0]
  |p -> if (card p) mod 2 = 0 then delta p [0] else delta p [1 + hd p];;
```

On calcule la parité de l'indice correspondant à la partie S_i en argument grâce à son cardinal. En effet, comme le passage d'une partie à la suivante se fait par différence symétrique avec un singleton, il y a ou adjonction, ou suppression de l'élément correspondant, et donc inversion de la parité.
Ensuite, si i est pair, c'est que t_i est nul, et alors on sait calculer le successeur par différence symétrique avec $\{0\}$. Sinon, la question 9.a) permet de calculer t_i à partir de S_i, ce qui permet aussi de calculer S_{i+1}.

Quatrième partie. Enumération des parties par un code de Gray

Question 10.

```
let rec test_sur p1 p2 =
  match p1,p2 with
     [],_ -> do_list (printf__printf ''%d '') p2
    |_,[] -> do_list (printf__printf ''%d '') p1
    |i1::q1,i2::_ when i1 < i2 -> printf__printf ''%d '' i1;
                                  test_sur q1 p2
    |i1::q1,i2::q2 when i1 = i2 -> test_sur q1 q2
    |i1::q1,i2::q2 -> test_sur p1 q2;$_{{}}$
                     printf__printf ''%d '' i2;;
```

On parcourt simultanément les deux parties dans le même esprit que pour le programme de la fonction delta, afin de n'imprimer que les interrupteurs présents dans l'une seulement des deux parties, mais en adoptant la stratégie suivante: dès qu'on rencontre un interrupteur à commuter dans la partie p1, on l'imprime (ce qui revient à le relever). Si par contre on rencontre un interrupteur à commuter dans la partie p2, on le laisse en attente et on examine les suivants (ce qui revient à différer son abaissement).
De ce fait tous les interrupteurs à relever dans la partie de départ seront relevés avant qu'on ne commence à abaisser ceux qui doivent l'être pour atteindre la partie cible.

Question 11.a)
Il est clair que la suite $l_{n,k}$ est définie par les relations suivantes:

$$\forall n,k\geq 1,\left\{\begin{array}{l}(1):l_{1,k}=1\\(2):l_{n+1,1}=l_{n,k}+2\\(3):l_{n+1,k+1}=l_{n,k+1}+l_{n,k}+1\end{array}\right.$$

Or, en posant $C_n^j=0$ si $n<j$, on a $\forall n,k\geq 1, s_{n,k}=\sum_{j=0}^{j=k}C_n^j$.
Et compte tenu de relation de Pascal $C_{n+1}^j=C_n^j+C_n^{j-1}$, la suite $s_{n,k}$ vérifie les relations:

$$\forall n,k\geq 1,\left\{\begin{array}{l}(1'):s_{1,k}=2\\(2'):s_{n+1,1}=s_{n,k}+1\\(3'):s_{n+1,k+1}=s_{n,k+1}+s_{n,k}\end{array}\right.$$

Il en découle que $s_{n,k}-1$ vérifie les relations (1) et (3) et la relation $(2')$. La suite obtenue en lui ajoutant encore C_{n-1}^k vérifie alors (1), (2) et (3). Finalement:

$$\underline{\forall n,k\geq 1, l_{n,k}=s_{n,k}+C_{n-1}^k-1}$$

Question 11.b)
Montrons par récurrence sur n que $\forall n\geq 1,\forall k\geq 1, P_{n,k}=\{A\subset I_n, Card\,(A)\leq k\}$ et $S_{k,l_{n,k}}=\{n-1\}$.

- $\underline{n=1}$
  $l_{1,k}=1$, donc $\forall k\geq 1, P_{1,k}=\{\emptyset,\{t_{k,0}\}\}=\{\emptyset,\{0\}\}=\{A\subset\{0\}, Card\,(A)\leq 1\}$.
  De plus, $S_{k,l_{1,k}}=\{0\}=\{1-1\}$.

- $\underline{n\rightsquigarrow n+1}$
  Soit $k\geq 1$.

  1. Si $k=1$, alors par hypothèse de récurrence on sait que $P_{n,1}=\{A\subset I_n, Card\,(A)\leq 1\}=\{\emptyset\}\cup\{\{j\}\}_{1\leq j\leq n-1}$.
     De plus, $P_{n+1,1}=P_{n,1}\cup\left\{S_{1,l_{n,1}}\vartriangle\{n-1\}, S_{1,l_{n,1}}\vartriangle\{n-1,n\}\right\}$.
     Or $S_{1,l_{n,1}}=\{n-1\}\Longrightarrow S_{1,l_{n,1}}\vartriangle\{n-1\}=\emptyset$ et $S_{1,l_{n,1}}\vartriangle\{n-1,n\}=\underline{S_{1,l_{n+1,1}}=\{n\}}$.
     Donc $P_{n+1,1}=\{\emptyset\}\cup\{\{j\}\}_{1\leq j\leq n}=\underline{\{A\subset I_{n+1}, Card\,(A)\leq 1\}}$.
  2. Sinon, on a $P_{n+1,k}=P_{n,k}\cup\left\{S_{k,l_{n,k}}\vartriangle\{n\}\right\}\cup\left\{S_{k,l_{n,k}}\vartriangle\{n\}\vartriangle\left\{t_{k,l_{n,k}}\right\}\vartriangle..\vartriangle\{t_{k,j}\}\right\}_{0\leq j\leq l_{n,k}}$.
     Mais $\left\{t_{k,l_{n,k}}\right\}\vartriangle..\vartriangle\{t_{k,j}\}=\left(\left\{t_{k,l_{n,k}}\right\}\vartriangle..\vartriangle\{t_{k,0}\}\right)\vartriangle(\{t_{k,0}\}\vartriangle..\vartriangle\{t_{k,j-1}\})=S_{k,l_{n,k}}\vartriangle S_{k,j}$, donc $S_{k,l_{n,k}}\vartriangle\{n\}\vartriangle\left\{t_{k,l_{n,k}}\right\}\vartriangle..\vartriangle\{t_{k,j}\}=S_{k,j}\vartriangle\{n\}$.
     Comme aucun $S_{k,j}$ ne contient n pour $j\leq l_{n,k}$, on en déduit que $S_{k,j}\vartriangle\{n\}=S_{k,j}\cup\{n\}$.
     Par suite, $P_{n+1,k}$ apparaît bien comme la réunion de $P_{n+1,k}$ (ensemble des sous-ensembles de I_{n+1} de cardinal inférieur à $k+1$ ne contenant pas n) et de $\{A\cup\{n\}, A\subset P_{n,k}\}$ (ensemble des sous-ensembles de I_{n+1} de cardinal inférieur à $k+1$ contenant n).
     En outre, $S_{k,l_{n+1,k}}=S_{k,0}\cup\{n\}=\{n\}$.

$$\underline{\forall n\geq 1,\forall k\geq 1, P_{n,k}=\{A\subset I_n, Card\,(A)\leq k\}\text{ et }S_{k,l_{n,k}}=\{n-1\}}$$

Question 12.

```
let test_panne N K =
  let rec T n k =
    match n,k with
       1,_ -> [0]
      |_,1 -> (T (n-1) 1) @ [n - 2;n - 1]
      |_ -> (T (n-1) k) @ [n - 1] @ (rev (T (n - 1) (k - 1)))
  in
  do_list (printf__printf ''%d '') (T N K);
  printf__printf ''%d '' (N - 1);;
```

Question 13.

La solution de cette question est de Gilbert Primet.
Le test des configurations d'au plus k interrupteurs baissés parmi n nécessite que toutes les configurations soient testées au moins une fois, soit au moins $s_{n,k}$ configurations correspondant à tous les sous-ensembles d'au plus k éléments dans un ensemble à n éléments.
De plus, comme on part de l'ensemble vide et qu'on aboutit à l'ensemble vide en modifiant à chaque nouvelle configuration un seul élément à la fois, il y a lors du test autant de configurations à nombre pair d'interrupteurs baissés que de configurations à nombre impair d'interrupteurs baissés. Il faut donc ajouter à $s_{n,k}$ la valeur absolue de la différence du nombre de parties de cardinal pair et du nombre de parties de cardinal impair parmi les parties à au plus k éléments dans un ensemble à n éléments, soit $\left|\sum_{j=0}^{j=k} (-1)^j C_n^j\right|$.

Mais on montre facilement, par récurrence sur k et à l'aide la relation de Pascal, que $\sum_{j=0}^{j=k} (-1)^j C_n^j = (-1)^k C_{n-1}^k$, d'où le nombre minimal de configurations à tester: $s_{n,k} + C_{n-1}^k = l_{n,k} + 1$.