ÉCOLE POLYTECHNIQUE - PSI/PT - ÉPREUVE FACULTATIVE D'INFORMATIQUE 2003

Corrigé rédigé par Alain Schauber - alain.schauber@prepas.org

NB : dans ce problème, on manipule des tableaux unidimensionnels indexés de 0 à n - 1. Pour définir un tel tableau, on ne peut pas utiliser ici la fonction vector car pour cette fonction l'indexation doit commencer à 1. On va donc se tourner vers la fonction array . Mais il faut alors préciser l'indexation à la définition.

Partie I - Bit de parité

Question 1

```
> ou_exclusif:=(x,y)-> if x=y then 0 else 1 fi;
```

Question 2

On fait un parcours du mot **b** en évaluant la parité du nombre de bits à 1 parmi les cellules déjà visitées à l'aide d'une variable **bp** initialisée à 0.
A chaque nouvelle cellule visitée, la mise à jour de la variable **bp** peut se faire à l'aide de la fonction précédente **ou_exclusif**.

```
> bit_parite:=proc(b) local bp,i;
    bp:=0;
    for i from 0 to n-1 do bp := ou_exclusif(b[i],bp) od;
    bp
  end;
```

Question 3

Notons d'abord que le bit de parité lui-même peut être altéré lors de la transmission.
Si on modifie exactement 1 des **n**+1 bits du mot, le nombre total de bits à 1 change de parité. Par récurrence immédiate, on voit donc que le nombre de bits à 1 change de parité si et seulement si on en modifie un nombre impair de bits.
Il en résulte que la technique du bit de parité permet de détecter tout nombre impair d'erreurs de transmission.

Partie II - Codage CRC

Question 4

Soit **m** la longueur du tableau (l'énoncé semble suggérer que cette fonction doit pouvoir s'appliquer à un polynôme de degré quelconque - et pas seulement **n**-1 -, ce qui se confirme à la question 6). On effectue un parcours du tableau **b** tant qu'on n'a pas trouvé une cellule non nulle. Dès que cela se produit, on interrompt la boucle et on retourne le complémentaire à **m**-1 de **i**, où i est l'indice minimal d'une cellule non nulle de **b**. La minimalité de i équivaut à la maximalité de **d** = **m**-1- **i**. Si toutes les cellules sont nulles (cas du polynôme nul), on retourne -1.

```
> degre:=proc(b) local m,i;
    m:=nops(op(3,eval(b)));# op(3,eval(b)) retourne une liste de
  même longueur que b
    for i from 0 to m-1 do if b[i] <> 0 then RETURN(m-1-i) fi od;
    -1
  end;
```

Question 5

Une simple boucle de longueur **l** permet de répondre à la question. La fonction ne semble avoir qu'un effet de bord (modification en place du tableau **b**), donc elle ne retourne rien.

```
> plus:=proc(b,c,i,j,l) local k;
    for k from 0 to l-1 do b[i+k] := ou_exclusif(b[i+k],c[j+k])
  od;
    RETURN()
  end;
```

Question 6

On peut proposer un algorithme itératif. On commence par calculer **k** et par créer un tableau **c** de longueur **n** + **k**.

Tant que le degré de **c** est strictement inférieur à **k** on ajoute **g** à **c** (grâce à la fonction **plus**), en prenant **l** = **k**+1, et pour **i** et **j** les indices des cellules à 1 les plus à gauche dans **c** et **g**. Quand la boucle est terminée, on retourne un tableau de longueur **k**, constitué uniquement des **k** cellules les plus à droite dans **c**.

On peut remarquer que **j** et **k** ne varient pas et dépendent du degré de **g**. On peut donc les calculer une fois pour toutes.

```
> crc:=proc(b,g) local k,j,c,u,r;
    k:=degre(g);
    j:=nops(op(3,eval(g)))-1-k; # indice du premier élément non
  nul dans g
    c:=array(0..(n+k-1),[]); # crée un tableau vide
    for u from 0 to n+k-1 do c[u]:=0 od; # initialisation à 0
    for u from 0 to n-1 do c[u]:=b[u] od; # copie de b en début de
  c
    while degre(c) >= k do plus(c,g,n+k-1-degre(c),j,k+1) od;
    r:=array(0..k-1,[]); # crée un tableau vide pour destiner à
  recevoir le reste
    for u from 0 to k-1 do r[u]:=c[n+u] od; # remplissage de r à
  partir de c
    eval(r) # puis retour du résultat
  end;
```

T(**degre**) est un "*O*" du nombre des cellules nulles les plus à gauche du tableau considéré et la boucle conditionnelle **while ...** comporte un nombre de tours en *O*(**n**).
Parmi ces tours de boucle, il n'y a que dans la dernière que la fonction **degre** est susceptible de parcourir tout le tableau **c** avant de rendre son verdict, sinon elle ne parcourt au plus que les n premières cellules du tableau. Il en résulte que cette boucle a une complexité temporelle en $O(\mathbf{n}^2)+O(\mathbf{n}+\mathbf{k})$. Les autres instructions ou boucles dans la fonction **crc** sont en *O*(**n**), en *O*(**k**) ou en *O*(1). On en déduit que $\underline{T(\mathbf{crc}) = O(\mathbf{n}^2)+O(\mathbf{k})}$.
Quant à la complexité spatiale de **crc**, elle est entièrement déterminée par l'allocation en mémoire du tableau **c**. Donc $\underline{E(\mathbf{crc}) = O(\mathbf{n}+\mathbf{k})}$.

Question 7
Dans la boucle conditionnelle de la fonction **crc** ci-dessus, seuls nous sont utiles dans **c** les **k** + 1 cellules de **c** qui sont à modifier à chaque tour de boucle. On se munit donc d'un tableau **r** de longueur **k** + 1 organisé en registre circulaire contenant les **k** + 1 cellules en question, et d'une variable-curseur **d** indiquant l'indice de la cellule contenant le bit le plus significatif, c'est-à-dire celui sous lequel dans **c** on « aligne » **g** (ou plutôt sa partie significative, comme dans **crc**).
On programme ensuite une boucle impérative qui « aligne » **g** tour à tour sous chacun des bits de **r** selon un mode circulaire. Si le bit courant de **r** vaut 1 on réalise
l'addition circulairement vers la droite à l'aide de la fonction **ou_exclusif**, s'il vaut o on ne fait rien. Dans tous les cas la cellule indexée par **d** sera à 0 et on affecte à la cellule ainsi libérée le premier bit non encore utilisé dans **b** (ou 0 si **b** est épuisé).
Lorsque cette boucle est terminée, **r** contient le reste cherché sur **k**+1 bits, mais ceux-ci sont permutés circulairement, le premier bit significatif étant toujours signalé par **d**. Il ne reste donc plus qu'à réordonner le tableau **r** par une permutation circulaire et à le retourner. Le reste étant de degré < **k**, le bit d'indice 0 de **r** vaut nécessairement 0, il suffit donc de l'actualiser à la fin. Le premier des **k** bits significatifs de **r** est donc trouvé à la cellule d'indice 1.

Comme la fonction **crc1** n'alloue dans la mémoire qu'un tableau de longueur **k**+1, on a clairement $E($**crc1**$) = O($**k**$)$.
On remarque que puisqu'on fait l'économie de la fonction **degre** dans la boucle indexée par **i**, la complexité temporelle est elle aussi améliorée : $T($**crc1**$) = O($**nk**$)$.

```
> crc1:=proc(b,g) local k,j,r,u,d,i;
    k:=degre(g);
    j:=nops(op(3,eval(g)))-1-k; # indice du premier élément non
  nul dans g
    r:=array(0..k,[]); # création du registre circulaire
    for u from 0 to k do r[u]:=b[u] od;  # initialisation du
  registre
    d:=0; # initialisation du curseur
    for i from 0 to n-1
    do
      if r[d] = 1 then # dans ce cas on peut ajouter g au reste
  courant
        for u from 0 to k
        do
        # utilisation de la fonction ou_exclusif plutôt que la
  fonction plus
        # pour tenir compte de l'indexation des cellules traitées
  modulo la longueur du registre
          r[(d+u) mod (k+1)] := ou_exclusif(r[(d+u) mod
  (k+1)],g[j+u]);
        od
      fi;
      # dans tous les cas le bit r[d] est nul, on le remplace
      # soit par le premier bit de b non encore utilisé, soit par
  0 si b est entièrement exploité
      if i+k+1 <= n-1 then r[d] := b[i+k+1] else r[d] := 0 fi;
      d:=(d+1) mod (k+1); # et on déplace le curseur
  circulairement vers l'avant
    od;
    while d <> 1
    do # rangement des k bits du reste final dans les k cellules
  de droite de r
      for u from 1 to k do r[(d-u) mod (k+1)]:=r[(d-1-u) mod
  (k+1)] od;
      d:=(d+1) mod (k+1);
    od;
    r[0]:=0; # le reste étant de degré < k le bit le plus à gauche
  de r est nul
    eval(r)  # retour du résultat en un tableau de k+1 cellules
  end;
```

Question 8
Le circuit présenté modélise l'algorithme précédent dans le cas particulier du polynôme générateur *G*(*X*) de degré **k** = 5. Pour le voir, montrons que le tableau des 6 bits r0-r1-r2-r3-r4-e (où e désigne l'entrée) suit les mêmes modifications que le tableau **r** manipulé dans la fonction **crc1**, les noms des cellules étant les mêmes, à la différence de la cellule **r5** de **r**, qui est modélisée par e. De plus, dans le modèle présenté, les bits du reste sont toujours alignés dans le bon ordre.

Observons d'abord qu'à chaque séquence d'horloge le bit r0 est ajouté à r1, à r3 et à e, les autres restant invariants, puis chaque bit r1 r2, r3, r4, e est décalé vers la gauche (le nouveau r1 prenant la place de r0), tandis que r0 est perdu et que la position e est occupée par une nouvelle entrée.
Ainsi, lorsque le bit r0 est nul, aucun des bits r1 r2, r3, r4, e n'est modifié, il en résulte donc uniquement un décalage vers la gauche. Par conséquent, comme les ri et e sont initialement nuls, les 6 premières séquences d'horloge conduisent à l'initialisation du tableau **r**, à l'instar de ce qui a été fait dans **crc1**.

De même, lors de la lecture de tous les bits d'entrée, une valeur r0 = 0 ne modifie pas les cellules déjà présentes et il n'y a donc que le décalage séquentiel qui est opéré (comme c'est fait dans **crc1** par l'intermédiaire du déplacement du curseur **d**). Voyons maintenant ce qui se passe lorsque r0 = 1 : On ajoute (par **ou_exclusif**) 1 à r1, 0 à r2, 1 à r3, 0 à r4 et 1 à e = r5, avant de les décaler puis de récupérer l'entrée suivante. Compte tenu du fait que pour notre polynôme *G*(*X*) le tableau **g** est <1,1,0,1,0,1>, on s'aperçoit qu'il s'agit exactement de la boucle des **k**+1 = 6 additions ou_exclusives exécutées dans le cas **r[d] = 1** dans **crc1**, le bit 1 le plus à gauche de **g** étant virtuellement ajouté à r0 = **r[d]** avant que celui-ci soit perdu suite au décalage. Il y a ensuite décalage des cellules, ce qui modélise bien le déplacement du curseur **d**.

Enfin, la séquence des 4 = **k**-1 bits nuls envoyés à la fin en entrée correspond à l'opération similaire de mise à jour de **r[d]** dans **crc1** lorsque le tableau **b** est épuisé.
A l'issue du processus, la dernière valeur de e est nulle (correspondant au bit nul en première position du résultat **r** de **crc1**) et les bits significatifs du reste sont bien les **k**=5 bits ri (0 <= i <= 4).

Question 9
Si on admet que la technique du bit de parité est un cas particulier de la méthode CRC, il est clair que **k** = 1 (le résultat est codé sur un 1 bit) et par conséquent le polynôme générateur ne peut être que *X* ou *X*+1. Le premier cas est absurde car la division de *P*(*X*).*X* par *X* donne toujours un reste nul. Il en résulte que nécessairement on doit avoir *G*(*X*) = *X*+1.
Réciproquement, si **g** = <1,1>, l'ajout par **ou_exclusif** de **g** au reste courant bascule chacun des deux bits concernés de ce reste en son contraire, et par conséquent ne modifie pas la parité du nombre de bits à 1. A l'issue du processus, la parité du reste initial (c'est-à-dire **b**) est la même que celle du reste final (c'est-à-dire **r**, codé sur 1 bit car de degré < 1). Autrement dit, en adjoignant ce bit aux bits de **b** , on a bien au total un nombre pair de bits à 1.
Conclusion : la technique du bit de parité est un cas particulier de la méthode CRC avec *G*(*X*) = *X*+1.