e3a concours

Concours ESTP - ENSAM - ECRIN - ARCHIMEDE

Epreuve de INFORMATIQUE

Filière MP

durée 3 heures

Les quatre exercices et le problème sont indépendants. Indiquez le nom du logiciel ou du langage que vous utilisez.

1. Calcul parallèle

Un processeur permet de faire une addition de 2 entiers en 1 unité de temps. Une addition de 9 entiers nécessite alors 8 unités de temps. Plusieurs processeurs pourraient coopérer pour faire cette addition en moins de temps.

- **Donner un exemple d'une coopération de 3 processeurs permettant d'additionner 9 entiers en 4 unités de temps.** (on négligera le temps nécessaire à gérer cette coopération)

- **Écrire la procédure récursive suivante :**

```
paral : paramètres     p, n : entiers ;
        retour         u : entier;
```

qui calcule avec u égal au nombre minimum d'unités de temps nécessaire à p processeurs coopérant au mieux pour calculer la somme de n entiers.

Note : on remarquera qu'il est impossible d'utiliser plus de n/2 processeurs en même temps et que l'utilisation de p processeurs diminue de p le nombre d'entiers à additionner lors de l'unité de temps suivante.

2. Le calcul de π : la méthode de Salamin

Dans les années 1970, Salamin a proposé une méthode très rapide pour calculer une approximation de π, basée sur la théorie des intégrales elliptiques. Dans cet algorithme, la n^e valeur approchée de π se calcule par : $\pi_n = \dfrac{4a_n^2}{1 - 2\sum_{i=1}^{n} 2^i\left(a_i^2 - b_i^2\right)}$

avec $a_0 = 1;\ b_0 = \dfrac{1}{\sqrt{2}};\ a_{n+1} = \dfrac{1}{2}\left(a_n + b_n\right);\ b_{n+1} = \sqrt{a_n b_n}$

- **Écrire la procédure suivante**

```
salamin : paramètre n : entier ; retour p : réel;
```

avec p égal à la n^e approximation de π.

3. Multiplication : la méthode des quarts de carrés

La méthode des quarts de carrés est une méthode de multiplication utilisée au Moyen-Âge, reposant sur l'identité $ab = \frac{(a+b)^2}{4} - \frac{(a-b)^2}{4}$.

Les mathématiciens d'alors disposaient de tables des quarts de carrés (qu'ils appelaient logarithmes) leur permettant de faire n'importe quelle multiplication, en ne faisant que des additions et des soustractions. Par exemple, pour faire 13x29, ils soustrayaient $\frac{(29-13)^2}{4} = 64$ de $\frac{(29+13)^2}{4} = 441$ pour obtenir 377.

- **Montrer que, pour cette méthode, la division entière (iquo(49,4)→12) peut être utilisée au lieu de la division réelle (49/4→12.25) ,quels que soient a et b.**

- **Écrire, en utilisant le principe des quarts de carré, la procédure suivante :**

```
produit :    paramètres a, b : entiers positifs ou nuls;
             retour m : entier
```

avec m = a.b

Notes : - on n'utilisera pas l'opérateur de multiplication
- on suppose existante la procédure sqr qui calcule le carré de son argument

4. Permutations

Une liste est une permutation d'une autre liste si elle est composée des mêmes éléments, pas forcément dans le même ordre. [3, 1, 2, 3, 4, 1] est une permutation de [1, 4, 1, 2, 3, 3], mais pas de [1, 4, 2, 4, 3, 1]. La façon de chercher si une liste est une permutation d'une autre dépend beaucoup des données.

Algorithme A : Si l'alphabet (l'ensemble des valeurs possibles) est restreint, on pourra créer un compteur pour chaque valeur possible. On parcourra la première liste, en incrémentant de 1 le compteur de chaque valeur rencontrée, puis on parcourra la deuxième liste, en décrémentant de 1 le compteur de chaque valeur rencontrée. Finalement, les listes seront des permutations l'une de l'autre si tous les compteurs sont nuls.

Algorithme B : Si l'alphabet est trop important (ou inconnu), on pourra créer une copie de la seconde liste, puis pour chaque valeur de la première liste, on testera si elle est présente dans la copie de la seconde liste. Si oui, on la supprimera de la copie, et on passera à la valeur suivante de la première liste. Si non, les deux listes ne sont pas des permutations. Lorsque la première liste sera parcourue entièrement, la copie devra être vide pour que les deux listes soient des permutations.

- **Écrire les procédures suivantes :**

```
perm1 :      paramètres     liste1, liste2 : listes ;
             retour         p : booléen;
```

basée sur l'algorithme A, avec p Vrai si liste1 et liste2 sont des permutations, Faux sinon

```
perm2 :      paramètres     liste1, liste2 : listes ;
             retour         p : booléen;
```

basée sur l'algorithme B, avec p Vrai si liste1 et liste2 sont des permutations, Faux sinon

5. Problème : travail sur les polynômes

Le but de ce problème est d'écrire les utilitaires permettant de travailler sur des polynômes à coefficients réels.

Un polynôme sera représenté par la liste de ses coefficients à partir de l'ordre 0.
Par exemple, [-2, 3, 5, -1] représentera le polynôme $-X^3+5X^2+3X-2$.
Le polynôme nul sera représenté par la liste vide.

Degré et coefficients d'un polynôme

- **Écrire les procédures suivantes :**

```
degre : paramètre      p : polynôme ;
        retour         n : entier;
```

avec `n` égal au degré du polynôme `p`

Note : **degre** retournera -1 pour le polynôme nul (pour indiquer $-\infty$)

```
lit_coef :paramètres   p : polynôme, i entier ;
          retour       c : réel
```

avec `c` égal au coefficient d'ordre `i` de `p`

Note : la procédure retournera 0 pour les coefficients d'ordre supérieur au degré.

```
ecr_coef: paramètres   p : polynôme, i : entier, c : réel ;
          retour       q : polynôme
```

avec `q` égal à `p`, sauf le coefficient d'ordre `i` égal à `c`

Note : si `i` est supérieur au degré de `p`, la procédure insérera des 0 pour les coefficients compris entre le degré de `p` et `i`

Valeur d'un polynôme en un point

- **Écrire la procédure suivante :**

```
eval : paramètres p : polynôme, x réel ; retour y : réel;
```

avec `y` égal à la valeur de `p` au point `x`

Note : on utilisera le schéma de calcul de Hörner, basé sur l'ordre d'évaluation

$$P(x) = (((...((a_n x + a_{n-1})x + a_{n-2})x+...+a_3)x + a_2)x + a_1)x + a_0$$

Opérations sur les polynômes

- **Écrire les procédures suivantes :**

```
add : paramètres p1, p2 : polynômes ; retour q : polynôme;
```

avec `q` égal à `p1 + p2`

Note : on veillera à ajuster la taille de la liste représentant le polynôme résultat

```
mult : paramètres p1, p2 : polynômes ; retour q : polynôme;
```

avec `q` égal au produit `p1.p2`