Examen machine

SUJET

Remarques :

Les consignes

Lisez les consignes : assurez-vous d'avoir bien tout compris !

Lisez l'énoncé jusqu'au bout avant de commencer !

Caml

Vos fonctions doivent **impérativement** respecter les exemples d'application donnés : noms et modes de passage des paramètres doivent être les mêmes.

Il va de soi que votre code doit être indenté !

En dehors d'indication dans l'énoncé, vous ne pouvez utiliser aucune fonction prédéfinie de CAML , à l'exception de `failwith` et `invalid_arg`. **Tout manquement à cette règle entrainera une note nulle.**

Vos fonctions doivent prendre en compte tous les cas : y compris ceux où les paramètres ne respectent pas les spécifications.

L'optimisation est notée, il est donc très fortement déconseillé d'utiliser l'opérateur `@`, voir pas du tout, et même interdit !
Il n'est pas demandé de rendre vos fonctions terminales (sauf dans d'éventuels bonus) : si vous le faites, attention au résultat.

Fin de l'épreuve

Assurez-vous d'avoir bien enregistré vos fichiers, selon les consignes.

1 Trajectoire

On appelle trajectoire d'un entier naturel `x` la suite d'entiers suivante : le premier élément est `x`, puis chaque élément est obtenu en ajoutant l'élément précédent à son inverse.

Inverse

Écrire la fonction `reverse` qui donne le miroir d'un entier.

```
val reverse : int -> int = <fun>

# reverse 13579 ;;
- : int = 97531
```

Trajectoire

Écrire la fonction `trajectory x n` qui retourne la trajectoire d'un nombre `x` sur les `n-1` prochaines valeurs.

```
val trajectory : int -> int -> int list = <fun>

# trajectory 11 10 ;;
- : int list = [11; 22; 44; 88; 176; 847; 1595; 7546; 14003; 44044]

# trajectory 666 7;;
- : int list = [666; 1332; 3663; 7326; 13563; 50094; 99099]
```

(n + reverse n) :: l

2 Goldbach

La conjecture de Goldbach est une assertion mathématique non démontrée qui s'énonce comme suit : Tout nombre entier pair supérieur à 3 peut s'écrire comme la somme de deux nombres premiers. La conjecture de Goldbach est vérifiée pour tous les entiers pairs jusquà 4×10^{18} (on ne testera pas au delà de cette valeur). Par exemple l'entier 12 est décomposé en `12 = 5 + 7`. Donc en généralisant pour un entier pair `n` : si pour un entier `i` premier (compris entre 2 et `n-2`), l'entier `n-i` est premier également, nous avons donc la décomposition de `n` en deux premiers.

Premier

Écrire la fonction `prime` qui teste si un nombre est premier. Rappel : un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même)

```
val prime : int -> bool = <fun>

# prime 89 ;;
- : bool = true
```

Goldbach

Écrire la fonction `goldbach` qui pour tout entier pair donne sa décomposition en deux nombres premiers, une erreur si la valeur à décomposer est incorrecte.

```
val goldbach : int -> int * int = <fun>

# goldbach 50 ;;
- : int * int = (3, 47)

# goldbach 13 ;;
Exception: Invalid_argument "Goldbach failed: input must be even".
```

Liste de Goldbach

Écrire la fonction `goldbach_list` qui retourne la liste des entiers pairs, compris entre deux bornes (inférieure et supérieure), accompagnés de leur décomposition en somme de deux premiers.

```
val goldbach_list : int -> int -> (int * (int * int)) list = <fun>

# goldbach_list 20 25;;
- : (int * (int * int)) list = [(20, (3, 17)); (22, (3, 19)); (24, (5, 19))]

# goldbach_list 13 8 ;;
- : (int * (int * int)) list = []
```

3 Représentation des polynômes par listes

Un polynôme $a_n.x^n + ... + a_1.x + a_0$ peut être représenté par la liste $[a_0; a_1; ...; a_n]$ de ses coefficients [1].
La représentation proposée ci-dessus a l'inconvénient d'avoir une taille proportionnelle à l'ordre du polynôme et non pas au nombre de monômes non nuls.
Par exemple, le polynôme $4x^5 + x^2 - 3x + 1$ serait représenté par la liste :

```
[1; -3; 1; 0; 0; 4]
```

Une autre solution (utilisée ici) consiste donc à représenter un polynôme par la liste des monômes. Un monôme $c.X^d$ est représenté par le couple d'entiers (c, d). Les monômes sont triés par degrés décroissants et chaque monôme a un coefficient non nul ($c \neq 0$).

La liste vide représentera le polynôme nul.

Par exemple, le polynôme $4x^5 + x^2 - 3x + 1$ sera représenté par la liste :

```
# let poly1 = [(4,5); (1,2); (-3,1); (1,0)] ;;
val poly1 : (int * int) list = [(4, 5); (1, 2); (-3, 1); (1, 0)]
```

$2x^5 + 2x^4 - x^2 + x$:

```
# let poly2 = [(2,5) ; (2,4) ; (-1,2) ; (1,1)] ;;
val poly2 : (int * int) list = [(2, 5); (2, 4); (-1, 2); (1, 1)]
```

Les polynômes résultats devront être bien formés, c'est à dire :
- il n'y a qu'un seul monôme par degré ;
- il n'y a aucun coefficient nul dans la liste ;
- les monômes sont triés par degrés décroissants.

Addition

Écrire la fonction `add_poly` calculant la somme de deux polynômes.
Exemple : $(4x^5 + x^2 - 3x + 1) + (2x^5 + 2x^4 - x^2 + x) = 6x^5 + 2x^4 - 2x + 1$

```
val add_poly : (int * 'a) list -> (int * 'a) list -> (int * 'a) list = <fun>

# add_poly poly1 poly2 ;;
- : (int * int) list = [(6, 5); (2, 4); (-2, 1); (1, 0)]
```

Multiplication monôme

Écrire la fonction `mult_poly poly mono` qui multiplie un polynôme `poly` par un monôme `mono`.
Exemple : $(4x^5 + x^2 - 3x + 1) \times (2x) = 8x^6 + 2x^3 - 6x^2 + 2x$

```
val mult_poly : (int * int) list -> (int * int) -> (int * int) list = <fun>

# mult_poly poly1 (2, 1) ;;
- : (int * int) list = [(8, 6); (2, 3); (-6, 2); (2, 1)]
```

Multiplication

Écrire la fonction `product_poly` calculant le produit de deux polynômes.
Exemple : $(4x^5 + x^2 - 3x + 1) \times (4x^5 + x^2 - 3x + 1) = 8x^{10} + 8x^9 - 2x^7 - 4x^5 + x^4 + 4x^3 - 4x^2 + x$

```
val product_poly : (int * int) list -> (int * int) list -> (int * int) list = <fun>

# product_poly poly1 poly2 ;;
- : (int * int) list = [(8, 10); (8, 9); (-2, 7); (-4, 5); (1, 4); (4, 3); (-4, 2); (1, 1)]
```

1. Il peut sembler plus logique d'inverser l'ordre des coefficients, mais la manipulation sera beaucoup trop compliquée !

4 Tri fusion

"Le tri fusion est un algorithme de tri stable par comparaison. Ce tri est basé sur la technique algorithmique diviser pour régner. L'opération principale de l'algorithme est la fusion, qui consiste à réunir deux listes triées en une seule. L'efficacité de l'algorithme vient du fait que deux listes triées peuvent être fusionnées en temps linéaire."
Wikipédia.

Algorithme

L'algorithme est naturellement décrit de façon récursive :
- Si la liste n'a qu'un élément, elle est déjà triée.
- Sinon, on sépare la liste en deux parties à quasi égales.
- On trie récursivement les deux parties avec l'algorithme du tri fusion.
- On fusionne les deux listes triées en une liste triée.

Split

Écrire la fonction `split` qui sépare une liste en deux listes de longueurs quasi identiques (à 1 élément près) : une moitié dans chaque liste. L'ordre des éléments dans le résultat n'a pas d'importance.

```
val split : 'a list -> 'a list * 'a list = <fun>

# split [1;2;3;4;5;6;7;8;9] ;;
- : int list * int list = ([9; 7; 5; 3; 1], [8; 6; 4; 2])
```

Fusion

Écrire la fonction `merge ord l1 l2` qui fusionne deux listes triées selon une relation d'ordre `ord` en une seule liste triée.

```
val merge : ('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list = <fun>

# let greater e1 e2 = e1 > e2 ;;
val greater : 'a -> 'a -> bool = <fun>
# merge greater [9;7;5;3;1] [8;6;4;2] ;;
- : int list = [9; 8; 7; 6; 5; 4; 3; 2; 1]

# merge (<=) ['a'; 'd'; 'f'] ['c'; 'e'; 'z'] ;;
- : char list = ['a'; 'c'; 'd'; 'e'; 'f'; 'z']
```

Tri

Écrire la fonction `merge_sort ord l` qui effectue un tri fusion sur la liste `l` passée en paramètre en utilisant la fonction de comparaison `ord`.

```
val merge_sort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>

# merge_sort (<=) [5;8;2;5;7;9;0;3;1;2;4] ;;
- : int list = [0; 1; 2; 2; 3; 4; 5; 5; 7; 8; 9]

# let grp x y =
    String.length x < String.length y || (String.length x = String.length y && x < y) ;;
val grp : string -> string -> bool = <fun>
# merge_sort grp ["python"; "ocaml"; "C++"; "ada"; "fortran"; "R"; "C#"; "K"; "F#"] ;;
- : string list = ["K"; "R"; "C#"; "F#"; "C++"; "ada"; "ocaml"; "python"; "fortran"]

# merge_sort
  (function x -> function y ->  x mod 2 < y mod 2 || (x mod 2 = y mod 2 && x < y))
  [5;8;2;6;7;9;0;3;1;10;4] ;;
- : int list = [0; 2; 4; 6; 8; 10; 1; 3; 5; 7; 9]
```