




Soutenance

Projet Ouverture - STL 2024

ARNOULD Yann, AYED Fatima, KONÉ Daba

10/12/2024

Fonctions



poly_add

poly_add_canonique

poly_prod

-> monome_prod

->List.map

->List.fold_left

poly_prod_canonique

abr2poly

->testPow

->mult2poly

->plus2poly

->aux

Expérimentations

expressions arborescentes



`exper_gen_abrs (n) (taille)` : génère `n` abr de longueur `taille` qui vont être transformé en expression arborescente

`exper_gen_abr_20 (n) (pas) (max)` : version de `exper_gen_abrs` qui a été conçu pour répondre à l'énoncé du projet en amenant dans ce cas 10 listes d'expressions arborescentes de longueur 100 (`n`) à 1000 (`max`) avec un pas (`pas`) de 100 et une taille de 20

`exper_gen_abr_20 (pow_max)` : autre version de `exper_gen_abrs` qui elle va créer `pow_max + 2` listes d'expressions de la taille (somme des puissances de 2 de 0 à `pow_max + 1`), ici `pow_max` vaut 13

Les expressions contenus dans ces listes seront transformées en polynôme pour la somme et le produit

Résultat dans un fichier

exper_gen_abr_20 et exper_gen_abr_15 ajoute dans un fichier leur temps de calculs dans des fichiers

time_execution fct arg f is_end : fonction générale utilisé par addition et le produit pour calculer le temps des stratégies et de le mettre dans un fichier f

```
100:0.054788
200:0.110397
300:0.168864
400:0.225229
500:0.276265
600:0.332835
700:0.389009
800:0.446508
900:0.521277
1000:0.557138
```

```
100:0.005177,152;0.004908,152;0.004807,152
200:0.013352,190;0.013777,190;0.013776,190
300:0.025752,237;0.026123,237;0.025804,237
400:0.030793,214;0.030872,214;0.032545,214
500:0.041720,219;0.041108,219;0.041605,219
600:0.066529,265;0.065637,265;0.066533,265
700:0.063874,242;0.062948,242;0.062928,242
800:0.083936,252;0.083575,252;0.084088,252
900:0.091894,254;0.091264,254;0.090548,254
1000:0.096619,252;0.099224,252;0.097833,252
```

Somme



`exp_somme` va être la fonction qui va utiliser les 3 stratégies d'additions et la fonction afin de comparer leurs temps d'exécutions et leurs résultats pour vérifier qu'elle calcule la même valeur .

Stratégie 1 : fonction naïve récursive qui prend une liste de polynôme et qui va effectuer l'addition dans un paramètre de la fonction

Stratégie 2 : fonction qui va cette fois ci utiliser `List.fold_left` pour réaliser l'addition des polynômes de la liste

Stratégie 3 : fonction itérative en utilisant le `while ... do` et la référence d'objet pour accumuler le résultat de l'addition des polynômes

Produit



Même chose que `exp_somme` sauf qu'on emploie ici 4 stratégies de produit de polynôme

Stratégie 1 et 2 : fonction naïve récursive qui va calculer le produit de la liste de polynôme dans un accumulateur sauf que la ou la stratégie 2 va canoniser le polynôme résultat du produit la stratégie 1 va quand à elle canoniser à chaque appel le résultat de la multiplication de 2 polynômes

Stratégie 3 : fonction itérative avec `while ... do` comme avec la somme

Stratégie 4 : fonction qui va s'appuyer sur l'algorithme diviser pour régner afin de calculer le produit de la liste de polynôme qui lui a été fourni

Interprétation des résultats



Produit des n arbres

- Utilisation de `exper_produit`
 - `exper_produit1` : stratégie naïve récursive 1 (application de canonique à chaque étape)
 - `exper_produit2` : stratégie naïve récursive 2 (application de canonique à la fin)
 - `exper_produit3` : stratégie naïve itérative
 - `exper_produit4` : stratégie diviser pour régner

Produit des n arbres

n	Réursive 1	Réursive 2	Itérative	Diviser pour régner
1	100:0.606063,9095;0.267387,9095;0.595225,9095;3.672771,9095			
2	200:3.143859,18287;1.650727,18287;3.219671,18287;22.794479,18287			
3	300:7.863855,26448;4.575657,26448;8.080054,26448;59.429376,26448			
4	400:12.768406,32495;7.340388,32495;13.265589,32495;102.898884,32495			
5	500:6.435261,0;7.265534,0;6.717634,0;110.258347,0			
1	100:0.595107,8923;0.270813,8923;0.615772,8923;3.592731,8923			
2	200:3.307896,17861;1.673665,17861;3.238502,17861;21.576438,17861			
3	300:6.410314,15392;4.356062,15392;6.286881,15392;46.553336,15392			
4	400:10.634828,0;6.285231,0;13.487114,0;91.972173,0			
1	100:0.552137,8860;0.239669,8860;0.563123,8860;3.453716,8860			
2	200:2.444042,15359;1.262531,15359;2.548323,15359;17.264858,15359			
3	300:6.949957,16815;4.033621,16815;8.035043,16815;53.717908,16815			
4	400:12.684859,29805;7.555160,29805;13.303919,29805;103.301186,29805			
5	500:18.654284,35057;10.755853,35057;19.530229,35057;155.992964,35057			
6	600:7.567712,0;10.652148,0;7.656665,0;140.346333,0			

`[(7,0); (1,1); (8,2); (12,3)]`

=> 4

- Longueur de la liste
- Nombre de monômes
- Nombre de degrés différents dans le polynôme

Produit des n arbres

n	Réursive 1	Réursive 2	Itérative	Diviser pour régner
1	100:0.606063,9095;0.267387,9095;0.595225,9095;3.672771,9095			
2	200:3.143859,18287;1.650727,18287;3.219671,18287;22.794479,18287			
3	300:7.863855,26448;4.575657,26448;8.080054,26448;59.429376,26448			
4	400:12.768406,32495;7.340388,32495;13.265589,32495;102.898884,32495			
5	500:6.435261,0;7.265534,0;6.717634,0;110.258347,0			

=> Fonctionnel jusqu'à n=400

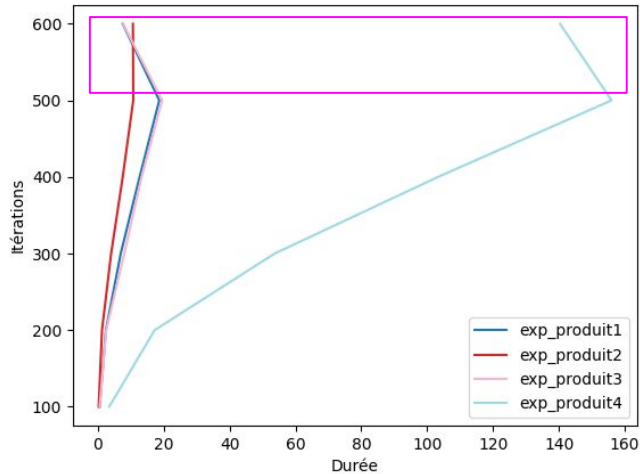
1	100:0.595107,8923;0.270813,8923;0.615772,8923;3.592731,8923
2	200:3.307896,17861;1.673665,17861;3.238502,17861;21.576438,17861
3	300:6.410314,15392;4.356062,15392;6.286881,15392;46.553336,15392
4	400:10.634828,0;6.285231,0;13.487114,0;91.972173,0

=> Fonctionnel jusqu'à n=300

1	100:0.552137,8860;0.239669,8860;0.563123,8860;3.453716,8860
2	200:2.444042,15359;1.262531,15359;2.548323,15359;17.264858,15359
3	300:6.949957,16815;4.033621,16815;8.035043,16815;53.717908,16815
4	400:12.684859,29805;7.555160,29805;13.303919,29805;103.301186,29805
5	500:18.654284,35057;10.755853,35057;19.530229,35057;155.992964,35057
6	600:7.567712,0;10.652148,0;7.656665,0;140.346333,0

=> Fonctionnel jusqu'à n=500

Comparaison des stratégies



```
1 100:0.552137,8860;0.239669,8860;0.563123,8860;3.453716,8860
2 200:2.444042,15359;1.262531,15359;2.548323,15359;17.264858,15359
3 300:6.949957,16815;4.033621,16815;8.035043,16815;53.717908,16815
4 400:12.684859,29805;7.555160,29805;13.303919,29805;103.301186,29805
5 500:18.654284,35057;10.755853,35057;19.530229,35057;155.992964,35057
6 600:7.567712,0;10.652148,0;7.656665,0;140.346333,0
```

- Stratégie diviser pour régner : stratégie la moins rapide
- Les trois stratégies naïves : plus rapides
 - En particulier la stratégie récursive 2 (appel de canonique une seule fois, à la fin)
- Idée d'amélioration pour poly_prod :
algorithme de Karatsuba



Cause des problèmes (1)

Résultat du produit de deux polynômes $p1$ et $p2$ est vide
=> **$p1$** est vide ou **$p2$** est vide ou **$p1$ et $p2$** sont vides

Ajout d'un log dans `poly_prod` :

```
1  p2 = []
2  p1 = (22,0) ; (5,3) ;
3
4  p2 = []
5  p1 = (1,1) ; (-124,0) ; (-25,1) ; (1,1) ; (1,66) ; (-145,0) ;
6
7  p2 = []
8  p1 = (1,1) ; (171,65) ;
9
10 p2 = []
11 p1 = (198,0) ; (1,1) ; (1,83) ;
```

=> Génération d'un polynôme vide,
potentiellement dans la chaîne des
transformations de `gen_permutations` :

- etiquetage
- `gen_arb`
- `arb2poly`

Resultat de la multiplicacion :

009213693952,5173) (-4611686018427387904,5174) (-4611686018427387904,5175) (-4611686018427387904,5176)

- -4611686018427387904
- 4611686018427387903

Ajouter des Big Int en OCaml :

- Module Num

Merci de votre attention.