

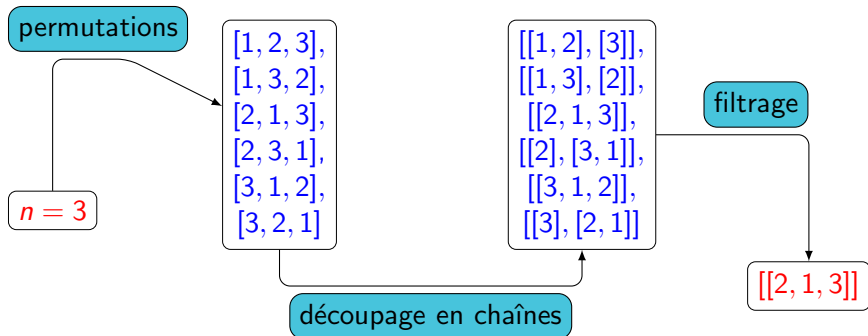
Recherche d'algorithmes...

Objectifs

- 👉 Détermination d'un **recouvrement optimal** en chaînes du graphe divisoriel **{algorithme 1}**,
- 👉 Détermination de la **plus longue chaîne** divisorielle dans une liste quelconque d'entier **{algorithme 2}**.

Algorithme 1 : Recouvrement optimal

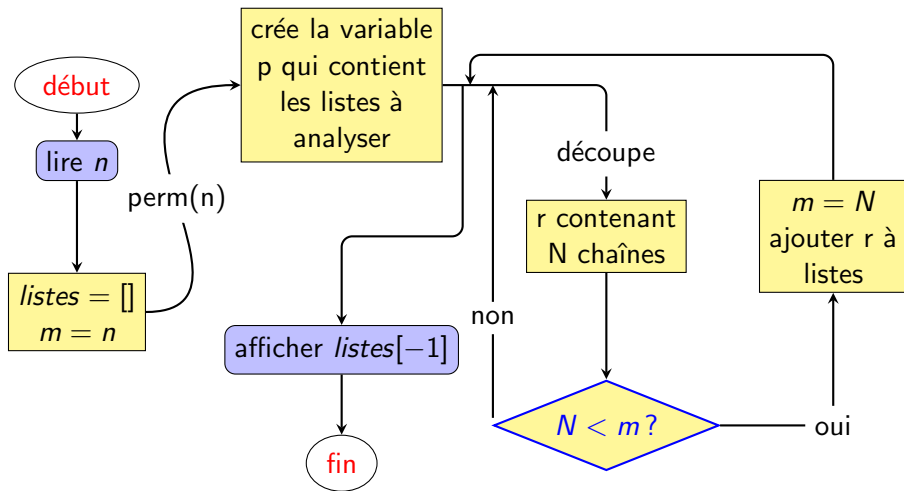
Un exemple pour comprendre



Algorithme 1 pour $n = 3$

Algorithme 1 : Recouvrement optimal

Organigramme



Algorithme 1 : Recouvrement optimal

Complexité & Optimisations

Complexité temporelle (algorithme non optimisé)

- 👉 Calcul des **permutations** en $\mathcal{O}(n!)$
- 👉 Coût d'un appel à la fonction **découpe** en $\mathcal{O}(n)$
- 👉 Complexité total en $\mathcal{O}(n \cdot n!)$

Optimisations **spatiale** et **temporelle**

- 👉 **paramètre m**, filtre les recouvrements non optimaux
- 👉 **initialisation** de m
- 👉 ne pas conserver les permutations en mémoire
- 👉 utiliser une seule grande boucle
- 👉 nombre premiers $p \leq n < 2p$
- 👉 ne pas analyser les recouvrements avec 1 en tête ou en queue de liste

Algorithme 1 : Recouvrement optimal

Résultats en Python

```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ----- RESTART -----
>>>
>>> recouvrement(8)
[[5], [7, 1, 2], [3], [4], [6], [8]]
[[5], [7, 1, 2], [3], [4, 8], [6]]
[[5], [7, 1, 2], [3, 6], [4, 8]]
[[5], [7, 1, 2, 4, 8], [3, 6]]
[[5], [7, 1, 3, 6, 2, 4, 8]]
>>> recouvrement(10)
[[7, 1, 2], [3], [4], [5], [6], [8], [9], [10]]
[[7, 1, 2], [3], [4], [5, 10], [6], [8], [9]]
[[7, 1, 2], [3], [4, 8], [5, 10], [6], [9]]
[[7, 1, 2], [3, 6], [4, 8], [5, 10], [9]]
[[7, 1, 2, 4], [5, 10], [6, 3, 9], [8]]
[[7, 1, 2, 4, 8], [5, 10], [6, 3, 9]]
[[7, 1, 4, 8, 2, 6, 3, 9], [5, 10]]
>>> recouvrement(12)
[[7], [11, 1, 2], [3], [4], [5], [6], [8], [9], [10], [12]]
[[7], [11, 1, 2], [3], [4], [5], [6, 12], [8], [9], [10]]
[[7], [11, 1, 2], [3], [4], [5, 10], [6, 12], [8], [9]]
[[7], [11, 1, 2], [3], [4, 8], [5, 10], [6, 12], [9]]
[[7], [11, 1, 2], [3], [5, 10], [6, 12, 4, 8], [9]]
[[7], [11, 1, 2], [3, 6, 12, 4, 8], [5, 10], [9]]
[[7], [11, 1, 2, 4, 8], [5, 10], [6, 12, 3, 9]]
[[7], [11, 1, 2, 8, 4, 12, 6, 3, 9], [5, 10]]
[[7], [11, 1, 5, 10, 2, 8, 4, 12, 6, 3, 9]]
>>> ----- RESTART -----
>>>
>>> arange_liste(4)
[[3, 1, 2, 4], [3, 2, 1, 4], [3, 1, 4, 2], [3, 4, 1, 2]]
>>> len(arange_liste(6))
96
>>> len(arange_liste(8))
600
>>> 720*7*8
40320
>>>
```

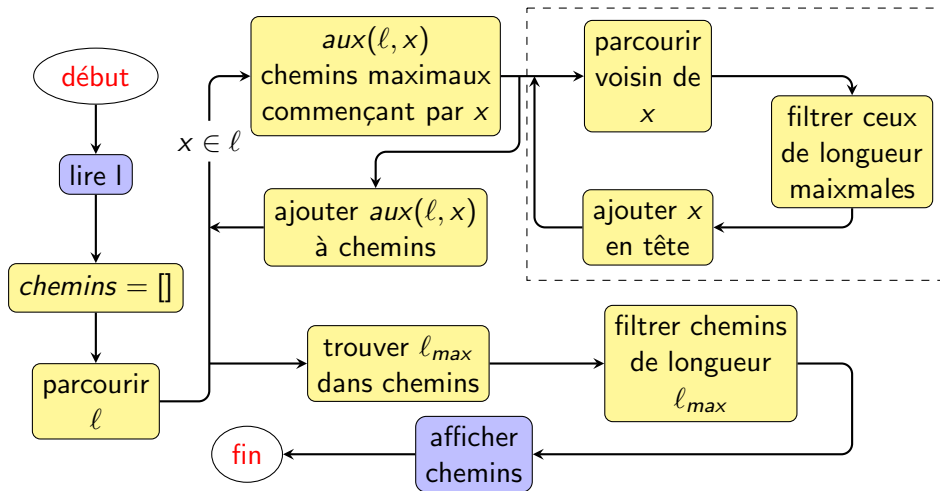
Algorithme 1 : Recouvrement optimal

Tableau comparatif

n	$n!$	permutations à analyser	temps d'exécution
4	24	4	≤ 1 s
6	720	96	≤ 1 s
8	40 320	600	≤ 1 s
10	3 628 800	322 560	3 s
11	39 916 800	322 560	3 s
12	479 001 600	3 265 920	23 s
13	6 227 020 800	3 265 920	24 s
14	87 178 291 200	(-)	≥ 1 h
15	1 307 674 368 000	(-)	(-)

Algorithme 2 : Plus longue chaîne

Organigramme : Ici règne le récursif ...



Algorithme 2 : Plus longue chaîne

Complexité & Optimisations

Complexité temporelle

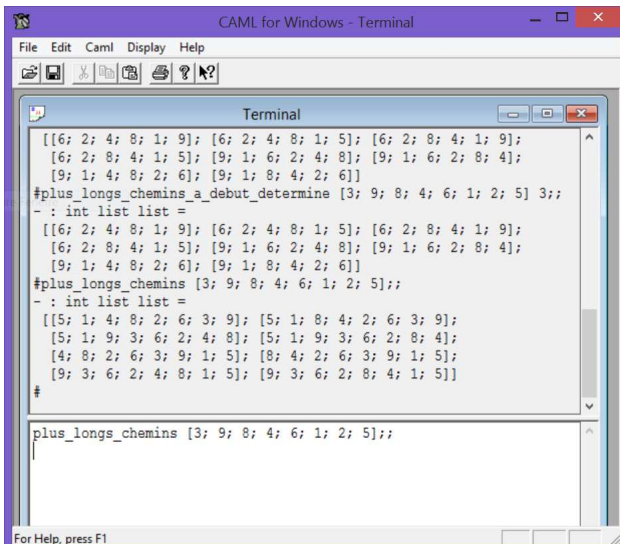
- 👉 calcul des plus long chemins à **début fixé** selon la relation $T_n = n \cdot T_{n-1}$ d'où $T_n = \mathcal{O}(n!)$
- 👉 Recherche de la **longueur maximale** d'un chemin dans la liste des chemins, *linéaire en la taille de la liste*
- 👉 **Filtrage** des chemins de longueur maximale, *linéaire en la taille de la liste*
- 👉 Complexité total $\mathcal{O}(n!)$

Optimisations

- 👉 nombre premiers $p \leq n < 2p$ dans le cas du graphe divisoriel
- 👉 **paramètre m**, filtrage de chemins non maximaux
- 👉 plus **difficile d'optimiser** (récursif ...)

Algorithme 2 : Plus longue chaîne

Résultats en Caml



```
[[6; 2; 4; 8; 1; 9]; [6; 2; 4; 8; 1; 5]; [6; 2; 8; 4; 1; 9];  
[6; 2; 8; 4; 1; 5]; [9; 1; 6; 2; 4; 8]; [9; 1; 6; 2; 8; 4];  
[9; 1; 4; 8; 2; 6]; [9; 1; 8; 4; 2; 6]]  
#plus_longes_chemin_a_debut_determine [3; 9; 8; 4; 6; 1; 2; 5] 3;;  
- : int list list =  
[[6; 2; 4; 8; 1; 9]; [6; 2; 4; 8; 1; 5]; [6; 2; 8; 4; 1; 9];  
[6; 2; 8; 4; 1; 5]; [9; 1; 6; 2; 4; 8]; [9; 1; 6; 2; 8; 4];  
[9; 1; 4; 8; 2; 6]; [9; 1; 8; 4; 2; 6]]  
#plus_longes_chemin [3; 9; 8; 4; 6; 1; 2; 5];;  
- : int list list =  
[[5; 1; 4; 8; 2; 6; 3; 9]; [5; 1; 8; 4; 2; 6; 3; 9];  
[5; 1; 9; 3; 6; 2; 4; 8]; [5; 1; 9; 3; 6; 2; 8; 4];  
[4; 8; 2; 6; 3; 9; 1; 5]; [8; 4; 2; 6; 3; 9; 1; 5];  
[9; 3; 6; 2; 4; 8; 1; 5]; [9; 3; 6; 2; 8; 4; 1; 5]]  
#  
  
plus_longes_chemin [3; 9; 8; 4; 6; 1; 2; 5];;  
|
```

For Help, press F1