Chapitre 1.2 Blocking
– Partie B -
Programmation des formules récursives

Analyse de dépendances,
Redondance et Mémoïsation,
Cache : blocking iteratif (aware) et récursif (oblivious)

Formulations récursives

- Une définition d’un objet est récursive si elle fait appel à la définition d’un objet du même type.

- Intéressant pour l’optimisation discrète:
  - Énumération récursive des solutions à envisager
  - Exemple

- Gloire… et déboires!
  - Algorithmes de bonne qualité (ex. D&C)
  - Ou complètement inefficaces si programmation naïve…

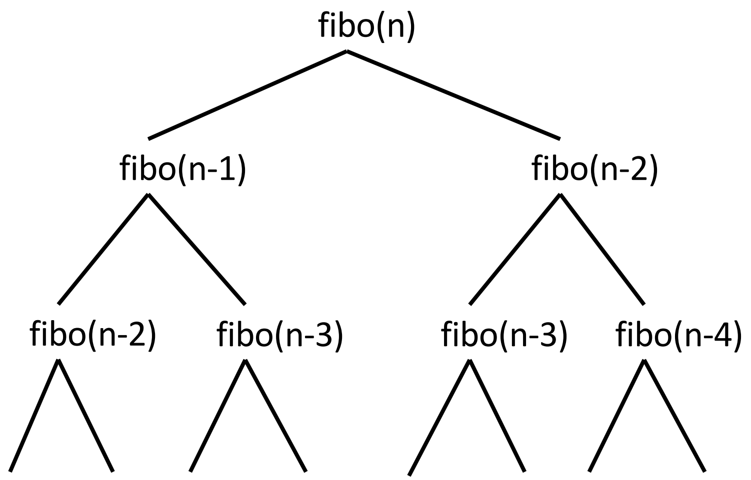
Exemple: suite de Fibonacci

fibo(n) = fibo(n-1) + fibo(n-2)

- Programme récursif naïf:

```
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
        return fib(n-1) + fib(n-2)
```

  - énormément de calculs redondants!
  - > $2^{n/2}$!!!

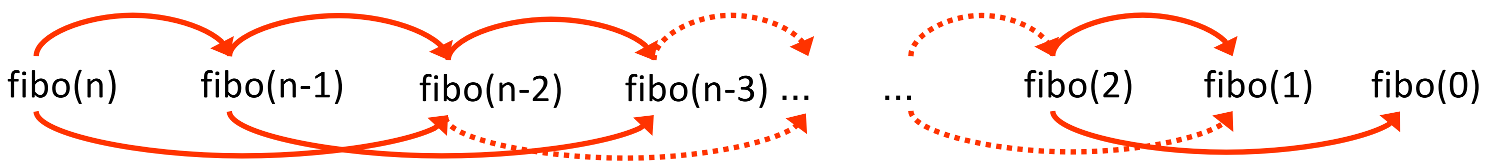


arbre d'appels

- Fib(20) =
- Fib(30) =
- Fib(40) =

Elimination des appels redondants

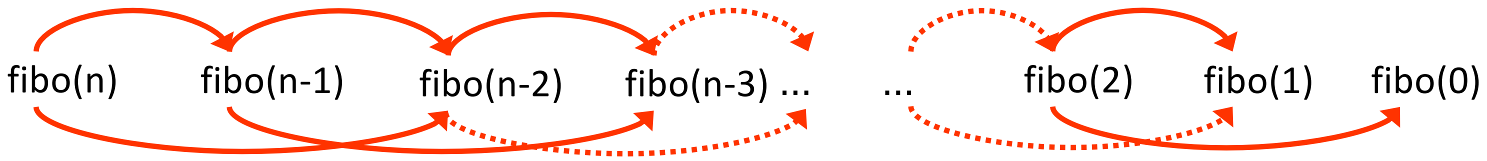
- Au lieu de représenter un arbre d'appel, on représente un **graphe d'appels**, en confondant les sommets correspondant à un même appel avec les mêmes valeurs



- Redondances bien visibles!
- Deux techniques d'élimination:
  - soit **ordonnancement** selon ordre topologique
  - soit « **mémoïsation** » (marquage)

Ordonnancement des calculs
- Exemple: Fibonacci

- Recherche d'un ordre topologique dans le graphe des appels



- Principe de l'algorithme?
  - cours graphes: trouver un ordre de traitement des sommets (tri topologique des tâches à accomplir)

Exemple: Fibonacci

```
integer fibo (n: integer) is
    k, fib_k, fib_k_1: integer
    k := 1
    fib_k_1 := 0
    fib_k := 1
    Pour k de 2 à n faire
        aux := fib_k_1 + fib_k
        fib_k_1 := fib_k
        fib_k := aux
    Retouner fib_k
```

- On retrouve l'algorithme itératif classique!
- En fait ce raisonnement amène d'un problème récursif à une écriture itérative!
- Q(n,L,Z) = O(1) !!!!

*Question subsidiaire: comment calculer fib(n) en O(*log *n) ops sur des entiers ?*
(une piste: dépendances algébriques…)

Mémoisation

- Mémoriser les appels effectués
  - Dans une table de hachage :
    - Clef = paramètres d’appel de la fonction
    - Valeur = valeur de l’appel avec ces paramètres
- Plus systématique
  - Automatisable dans les langages fonctionnels

Exemple: Fibonacci

- Mémoïsation:
  Algorithme récursif avec marquage

```
integer fibo (n: integer) is
    Si (tab[n] = -1) Alors  // pas encore calculé
        tab[n] := fibo(n-1) + fibo(n-2)
    Retourner tab[n]
```

tab[0]:=0
tab[1]:=1

Inconvénient?

O(n) en mémoire au lieu de O(1)

Q(n,L,Z) = n/L au lieu de O(1)

```
def fib_tab(n, table):
    if n not in table:
        if n == 0:
            table[ 0 ] =  0
        elif n == 1:
            table [ 1] = 1
        else:
            table [ n ] = fib_tab(n-1, table) + fib_tab(n-2, table )
    return table[n]


def fib_efficace (n):
    tab = {}
    return fib_tab(n , tab)
```

Automatisation/Généralisation:
Un peu de fonctionnel

- def map(f, liste):

  return [f(x) for x in liste]

- def square(x) :
          return x*x
  print map(square, [0,1,2,3])
  [0, 1, 4, 9]

- map(lambda x: x*x , [0,1,2,3])

Map et ordre supérieur

- def my_map(f):
  def map_f (liste):
  return [f(x) for x in liste]
  return map_f

```
def my_map( f ):
    def map_f ( liste ):
        return [ f(x)  for x in liste ]
    return map_f
```

- mapsquare = my_map(square)
- Print mapsquare([0,1,2,3])

Exercice: reduce, filter

Memoïsation: ordre supérieur

- def memoize(f):
    memo = {}

    def memo_appel(x):

        if x not in memo:

            memo_appel[x] = f(x)

        return memo[x]

    return memo_appel # retourne la fn!
- fibKO = memoize(fib)
- fibKO(40)

fib = memoize(fib)

fib(40)

Décorateur **@** en Python

- **@**memoize
```
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else: return fib(n-1) + fib(n-2)
```

Exemple 2 : Coefficients binômiaux

Formule du triangle de Pascal :

$$\begin{pmatrix} n \\ p \end{pmatrix} = \begin{pmatrix} n-1 \\ p \end{pmatrix} + \begin{pmatrix} n-1 \\ p-1 \end{pmatrix}$$

avec les conditions d'arrêt

$$\begin{pmatrix} n \\ n \end{pmatrix} = 1;$$

$$\begin{pmatrix} n \\ 0 \end{pmatrix} = 1.$$

Exemple: combinatoire

$$\begin{cases} C_n^p = C_{n-1}^p + C_{n-1}^{p-1} & 0 < p < n \\ C_n^0 = C_n^n = 1 & \end{cases}$$

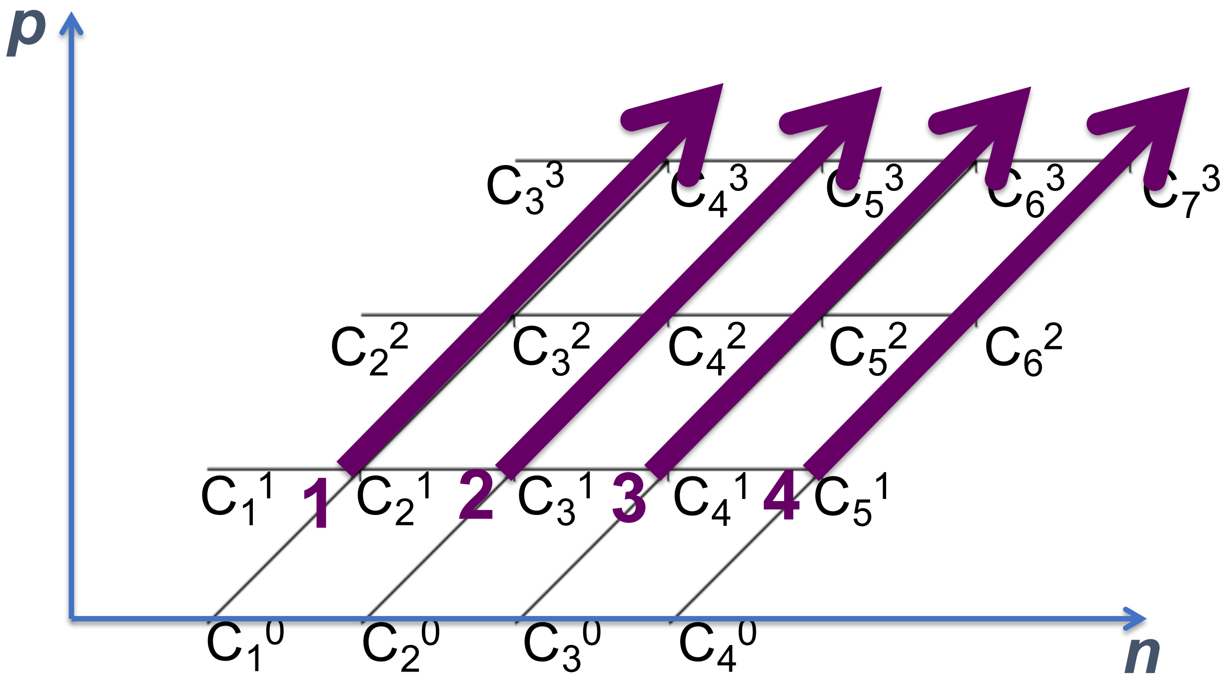
- Arrangements de p valeurs parmi n

- Exercice: écrire un programme qui calcule C(n,p) avec mémoïsation.
  - Donner le coût

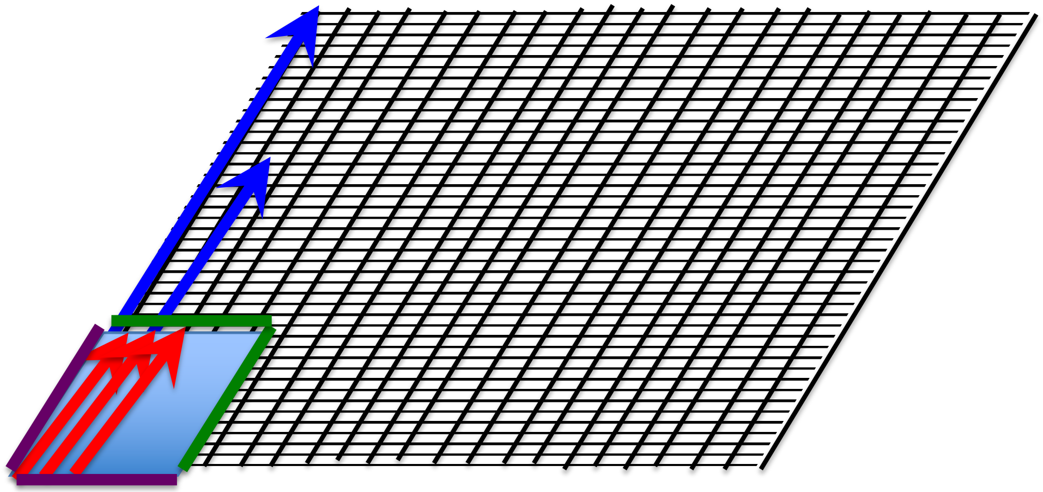
C_n^p Cache aware (1)

- Graphe de dépendance en stencils (en plusieurs dimensions): schémas de calcul fréquents :
  - Cnp (déjà vu)
  - Calcul numérique (Jacobi)
  - Exemples similaires en programmation dynamique
    - Patch optimal, distance de Fréchet, …
- **Ici** : calcul du Cnp par calcul itératif par diagonale en supposant p < n-p (sinon par colonne):
  - Si Z assez grand : Q(n,p,L,Z)= p/L défauts de cache ☺
    - Une fois la diagonale de taille p en cache, plus de défauts!
  - Si Z petit (Z<p): Q(n, p, L, Z)= (n-p).p/L défauts cache☹
    - En bas de la hiérarchie (niveau L1), Z est petit…

C_n^p Cache aware (2.a)

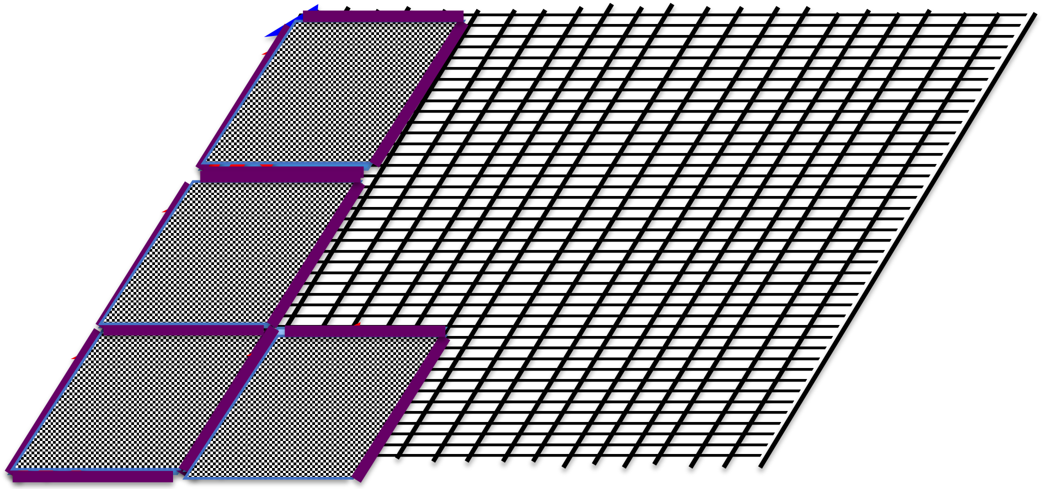


C_n^p Cache aware (2.b)



- En faisant les calculs par blocs tenant dans le cache :
  - Il suffit de stocker les frontières du blo:c **read E-S** **;** **write W-N**
  - Pour minimiser le nombre de points sur les frontières: prendre des bloc carrés K x K tel que 5.K rentre dans le cache de taille Z
    - Seulement O(1 + K/L) cache miss pour K^2 calculs
    - Il y a (n-p).p/K^2 blocs **=> (n-p).p/K.(1/L+1/K)** cache miss!

C_n^p Cache aware (2.c)



- Avec K = Z/5 : $Q(n, p, L, Z) = O\left((n-p).p / (L.Z) \right)$ ☺
- Le calcul par blocs permet le parallélisme sur la boucle externe! ☺

Cnp cache oblivious

- découpe récursive par blocs
  - On ne fait des calculs qu'au seuil d'arrêt !
    Le reste n'est que des appels récursifs sans accès aux tableaux (juste découpe=arithmétique de pointeurs)
  - On s'arrête a un seuil assez petit; on peut unroller la boucle sur le bloc qui concentre tout le coût du calcul et les accès mémoire !
- La découpe récursive (en 2 selon la plus grosse dimension) tire parti de la hiérarchie mémoire
  - Au bout d'un moment on fini par tomber dans le cache visé, à tout niveau de la hiérarchie!
  - Q= O((n-p).p / (L.Z)) sans connaître L et Z !

Conclusion

Technique de blocking : présentation et code

- Rappel : Hiérarchie mémoire et modèle CO
- Le modèle CO : cache de taille Z, ligne de taille L, police LRU
- Technique de blocking
- Programmation : exemple de la transposition de matrice
- Transposition de matrice

Exemples d'algorithmes cache aware/oblivious

- Produit de matrices
- Méthodolologie pour le blocking
- Exemple : double boucle imbriquée

Cas d'algo. récursifs : mémoïsation et localité

- Fibonacci
- Méthodologie
- Coefficients binômiaux

Conclusion - Méthodologie