

CALCULUM
STRATÉGIES DE RÉOLUTION

Mattéo Delabre

Université de Montréal
2 octobre 2023

OBJECTIFS

- ▶ La plupart des problèmes demandent de **rechercher une solution** parmi un ensemble d'éléments possibles.
- ▶ Différentes stratégies existent :
 - Recherche exhaustive (« *bruteforce* »)
 - Programmation dynamique
 - Diviser pour régner
 - Vorace (« *greedy* »)
- ▶ Notre programme d'aujourd'hui :
 - En quoi consistent ces stratégies ?
 - Comment choisir la bonne ?

PLAN

- 1 Recherche exhaustive**
- 2 Programmation dynamique**
- 3 Diviser pour régner**
- 4 Vorace**
- 5 Choisir la bonne stratégie**

PRINCIPE

- ▶ *Pour trouver la solution, il suffit de tester toutes les possibilités !*
- ▶ Problèmes-types : satisfiabilité et contraintes.
- ▶ Heuristiques : Orienter la recherche dans la bonne direction.
- ▶ Élagage (*pruning*) : Certaines possibilités n'ont pas à être explorées.

ASTUCES D'IMPLÉMENTATION

1 Itération sur toutes les possibilités.

- `itertools.product("ABC", repeat=2)`
`>>> (AA, AB, AC, BA, BB, BC, CA, CB, CC)`
- `itertools.permutations("ABC", 2)`
`>>> (AB, AC, BA, BC, CA, CB)`
- `itertools.combinations("ABC", 2)`
`>>> (AB, AC, BC)`
- `itertools.combinations_with_replacement("ABC", 2)`
`>>> (AA, AB, AC, BB, BC, CC)`

ASTUCES D'IMPLÉMENTATION

- 2 Retour sur trace (*backtracking*) : faire des choix et se donner la possibilité de revenir en arrière.
- Utiliser la récursivité pour gérer le retour à notre place.
 - Attention à la limite de récursion en Python (dans l'ordre de 1 000)
 - `sys.setrecursionlimit(n)`

RÉFÉRENCES ET EXERCICES

► Exercices :

- Advent of Code 2020, « *Jurassic Jigsaw* ».
<https://adventofcode.com/2020/day/20>
- Catégorie sur Codeforces :
<https://codeforces.com/problemset?tags=brute+force>

► Dans les livres de référence :

- Dürr et Vie, §15.
- Laaksonen, §15.5.
- Halim, §3.2.

PLAN

- 1 Recherche exhaustive
- 2 **Programmation dynamique**
- 3 Diviser pour régner
- 4 Vorace
- 5 Choisir la bonne stratégie

PRINCIPE

- ▶ Deux propriétés principales nécessaires :
 - 1 *Une solution optimale est une combinaison de sous-solutions optimales.*
 - 2 *On calcule plusieurs fois les mêmes valeurs lors du calcul d'une solution.*

- ▶ Problèmes classiques :
 - plus courts chemins ;
 - sac à dos ;
 - rendu de monnaie ;
 - distance d'édition ;
 - ...*et bien d'autres!*

ASTUCES D'IMPLÉMENTATION

- ▶ `collections.defaultdict(int)`
Dictionnaire dont la valeur par défaut est 0.
- ▶ Initialisation virtuelle :
<https://eli.thegreenplace.net/2008/08/23/initializing-an-array-in-constant-time>

RÉFÉRENCES ET EXERCICES

► Exercices :

- Advent of Code 2020, « *Adapter Array* ».
<https://adventofcode.com/2020/day/10>
- Catégorie sur Codeforces :
<https://codeforces.com/problemset?tags=dp>

► Dans les livres de référence :

- Dürr et Vie, §1.6.5.
- Laaksonen, §6.
- Halim, §3.5.

PLAN

- 1 Recherche exhaustive
- 2 Programmation dynamique
- 3 **Diviser pour régner**
- 4 Vorace
- 5 Choisir la bonne stratégie

PRINCIPE

- ▶ *Une solution optimale est une combinaison **d'un nombre constant** de sous-solutions optimales.*
- ▶ Techniques classiques :
 - recherche dichotomique ;
 - tri par fusion, tri rapide ;
 - arbre binaire de recherche.

RÉFÉRENCES ET EXERCICES

► Exercices :

- Catégorie sur Codeforces : <https://codeforces.com/problemset?tags=divide+and+conquer>

► Dans les livres de référence :

- Dürr et Vie, §1.6.7.
- Laaksonen, §4.3 et §15.4.2.
- Halim, §3.3.

PLAN

- 1 Recherche exhaustive
- 2 Programmation dynamique
- 3 Diviser pour régner
- 4 **Vorace**
- 5 Choisir la bonne stratégie

PRINCIPE

- ▶ *Faire un choix optimal à chaque étape amène toujours à une solution optimale.*
- ▶ Problème classique : *scheduling* d'événements dont l'heure de début et de fin est connue.

RÉFÉRENCES ET EXERCICES

► Exercices :

- Advent of Code 2020, « *Report Repair* ».
<https://adventofcode.com/2020/day/1>
- Catégorie sur Codeforces :
<https://codeforces.com/problemset?tags=greedy>

► Dans les livres de référence :

- Dürr et Vie, §1.6.4.
- Laaksonen, §4.2.2.
- Halim, §3.4.

PLAN

- 1 Recherche exhaustive
- 2 Programmation dynamique
- 3 Diviser pour régner
- 4 Vorace
- 5 Choisir la bonne stratégie**

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	 4 WEEKS	 3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	 8 WEEKS	 6 DAYS	 1 DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	 4 WEEKS	 6 DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	 5 WEEKS	 5 DAYS	 1 DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	 10 DAYS	 2 DAYS	5 HOURS
	6 HOURS				2 MONTHS	 2 WEEKS	 1 DAY
	 1 DAY				 8 WEEKS	 5 DAYS	

STRATÉGIES DE RECHERCHE ET COMPLEXITÉS ATTENDUES

- ▶ Recherche exhaustive (« *bruteforce* ») : $O(2^n)$, $O(n!)$
- ▶ Programmation dynamique : $O(n^2)$, $O(n^3)$, ...
- ▶ Diviser pour régner : $O(\log n)$, $O(n \log n)$, ...
- ▶ Vorace (« *greedy* ») : $O(n)$

TAILLE DE L'ENTRÉE ET COMPLEXITÉ ACCEPTABLE

n	Pire complexité	Exemples d'algorithmes
≤ 10	$O(n!), O(n^6)$	Permutations
≤ 15	$O(2^n \times n^2)$	Voyageur de commerce en programmation dynamique
≤ 18	$O(2^n \times n)$	
≤ 100	$O(n^4)$	Prog. dyn. à trois variables
≤ 400	$O(n^3)$	Floyd-Warshall
≤ 2000	$O(n^2 \log n)$	
$\leq 10^4$	$O(n^2)$	Tris quadratiques
$\leq 10^6$	$O(n \log n)$	Tris par comparaisons optimaux
$\leq 10^8$	$O(n), O(\log n), O(1)$	<i>Note : Assez rare</i>

En supposant une limite de temps de 3 s sur une machine exécutant 30 millions d'opérations par seconde. Tiré de Halim (Table 1.4).