Problème du sac à dos

TP de Licence Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université Fédérale Toulouse Midi-Pyrénées

Exposé du problème

On veut écrire un programme parallèle pour résoudre le problème du sac-à-dos :

- on dispose de **N** objets caractérisés par leur masse **`M[i]`** et leur utilité **`U[i]`** (`0≤i<N`) et d'un sac à dos de capacité **`C`** (masse totale maximale).
- on veut choisir un sous-ensemble d'objets dont la somme des masses est inférieure ou égale à C et qui maximise la somme des utilités.

Exemple

objet	`M[i]`	`U[i]`
0	4	3
1	2	4
2	3	2
3	1	1
4	3	3

Configurations possibles :

- objets 0 et 1 ➛ masse=6 et utilité=7
- objets 1, 2 et 3 ➛ masse=6 et utilité=7
- objets 2 et 4 ➛ masse=6 et utilité=5
- objets 1 et 2 ➛ masse=5 et utilité=6
- objets 0, 1 et 2 ➛ masse=9 et utilité=9
- objets 1, 3 et 4 ➛ masse=6 et utilité=8
- …

Exemple

objet	`M[i]`	`U[i]`
0	4	3
1	2	4
2	3	2
3	1	1
4	3	3

$\boxed{C = 6}$

Configurations possibles :

- objets 0 et 1 ➛ masse=6 et utilité=7 ✓
- objets 1, 2 et 3 ➛ masse=6 et utilité=7 ✓
- objets 2 et 4 ➛ masse=6 et utilité=5 ✓
- objets 1 et 2 ➛ masse=5 et utilité=6 ✓
- objets 0, 1 et 2 ➛ masse=9 et utilité=9 ✗
- objets 1, 3 et 4 ➛ masse=6 et utilité=8 ✓ 🥇
- …

Résolution du problème

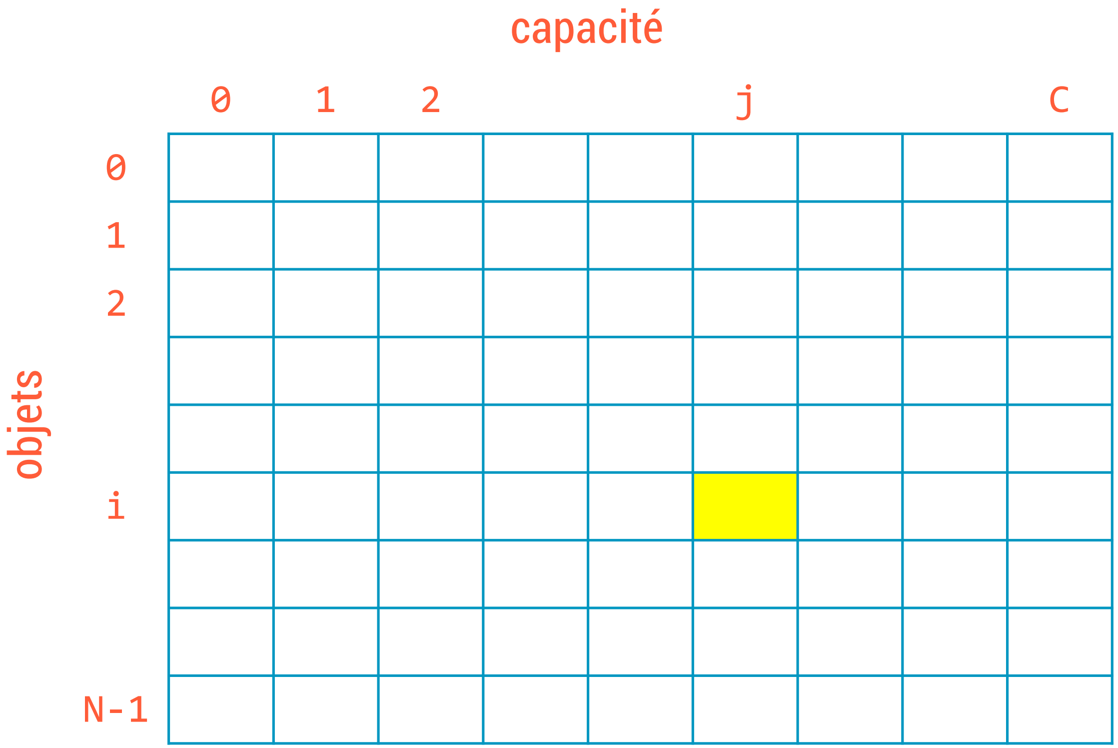
Il s'agit de construire un tableau **`S`** de taille `N×(C+1)`, tel que **`S[i][j]`** représente *l'utilité maximale que l'on peut obtenir choisissant des objets parmi ceux numérotés de 0 à `i`, et en limitant la somme des masses à `j`.*

Une fois le tableau `S` calculé, l'utilité maximale est donnée par **`S[N-1][C]`**.

On peut alors construire le tableau **`E`** de dimension **`N`** tel que **`E[i]=1`** si l'objet **`i`** doit être emporté et **`E[i]=0`** sinon. Pour cela, il faut remonter dans le tableau pour retrouver les décisions qui ont permis d'atteindre **`S[N-1][C]`**.

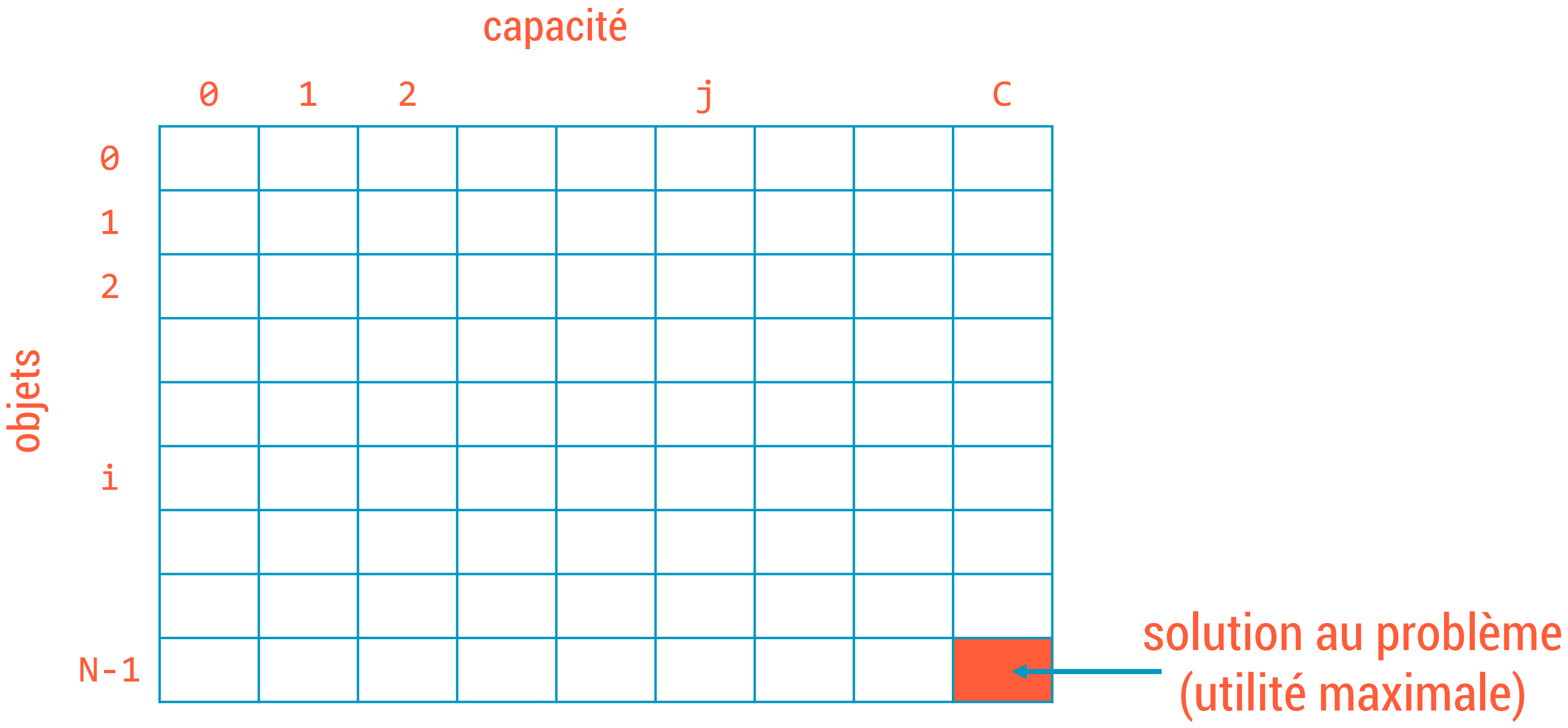
Résolution du problème

Il s'agit de construire un tableau `S` de taille `N×(C+1)`, tel que `S[i][j]` représente *l'utilité maximale que l'on peut obtenir choisissant des objets parmi ceux numérotés de 0 à `i`, et en limitant la somme des masses à `j`.*



Résolution du problème

Une fois le tableau `S` calculé, l'utilité maximale est donnée par **`S[N-1][C]`**.



Résolution du problème

La ligne 0 de ce tableau est facile à calculer :

- `S[0][j]=0` si `M[0]>j` (la masse de l'objet 0 est supérieure à la capacité j, donc on ne peut pas le prendre)
- `S[0][j]=U[0]` si `M[0]≤j` (on peut prendre l'objet 0)

capacité

objets

	0	1	2		M[0]				C
0	0	0	0	0	U[0]	U[0]	U[0]	U[0]	U[0]
1									
2									
N-1									

Résolution du problème

Lorsque l'on passe d'une ligne à la suivante, on élargit l'ensemble des objets qu'on peut sélectionner. Lorsque l'on calcule `S[i][j]`, la question qui se pose est : faut-il inclure l'objet `i` ?

- si on n'inclut pas l'objet `i`, l'utilité optimale est `S[i-1][j]`.
- si on inclut l'objet `i`, il faut ensuite compléter le sac à dos avec la meilleure combinaison parmi les objets `0` à `i-1`. L'utilité obtenue est alors : `S[i-1][j-M[i]] + U[i]`
- on choisit la solution qui apporte l'utilité **maximale** tout en respectant la contrainte de capacité

capacité

objets

	0	1	2	j - M[i]		j			C
0	0	0	0	0	U[0]	U[0]	U[0]	U[0]	U[0]
1	..	..	..	..	..	..	..	..	..
2	..	..	..	..	..	..	..	..	..
	..	..	..	..	..	..	..	..	..
i-1	..	..	..	..	..	..	..	..	..
i					+U[i]				
N-1									

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1							
3	2	2							
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4			
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	4		
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	+2 4	2 | 4		
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	4		
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	4		
1	1	3							
3	3	4							

Exemple

`M[i]`	`U[i]`	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	4	4	
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	+2 4	6 | 4	
1	1	3							
3	3	4							

Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	4	6	
1	1	3							
3	3	4							

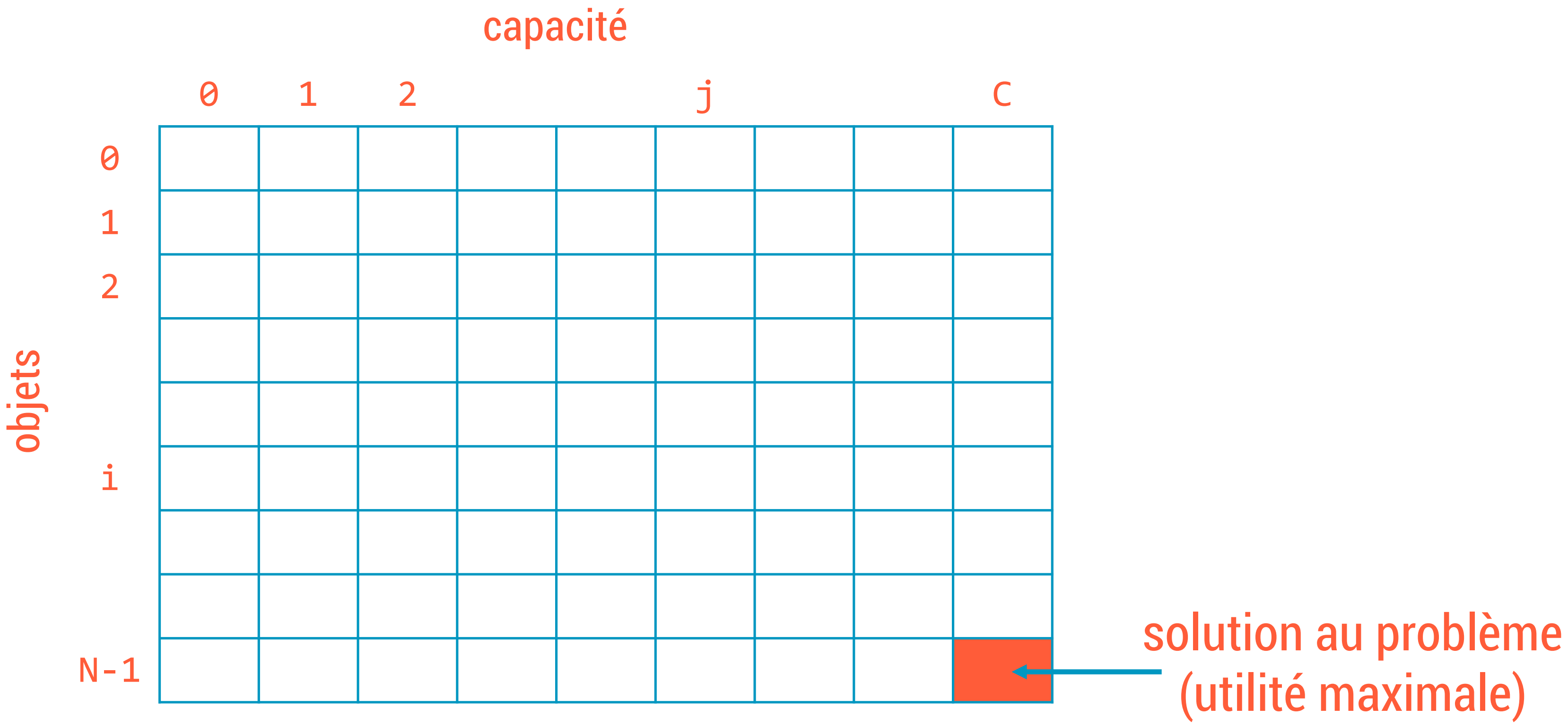
Exemple

M[i]	U[i]	objet	0	1	2	3	4	5	6
4	3	0	0	0	0	0	3	3	3
2	4	1	0	0	4	4	4	4	7
3	2	2	0	0	4	4	4	6	7
1	1	3	0	1	4	5	5	6	7
3	3	4	0	1	4	5	5	7	**8**

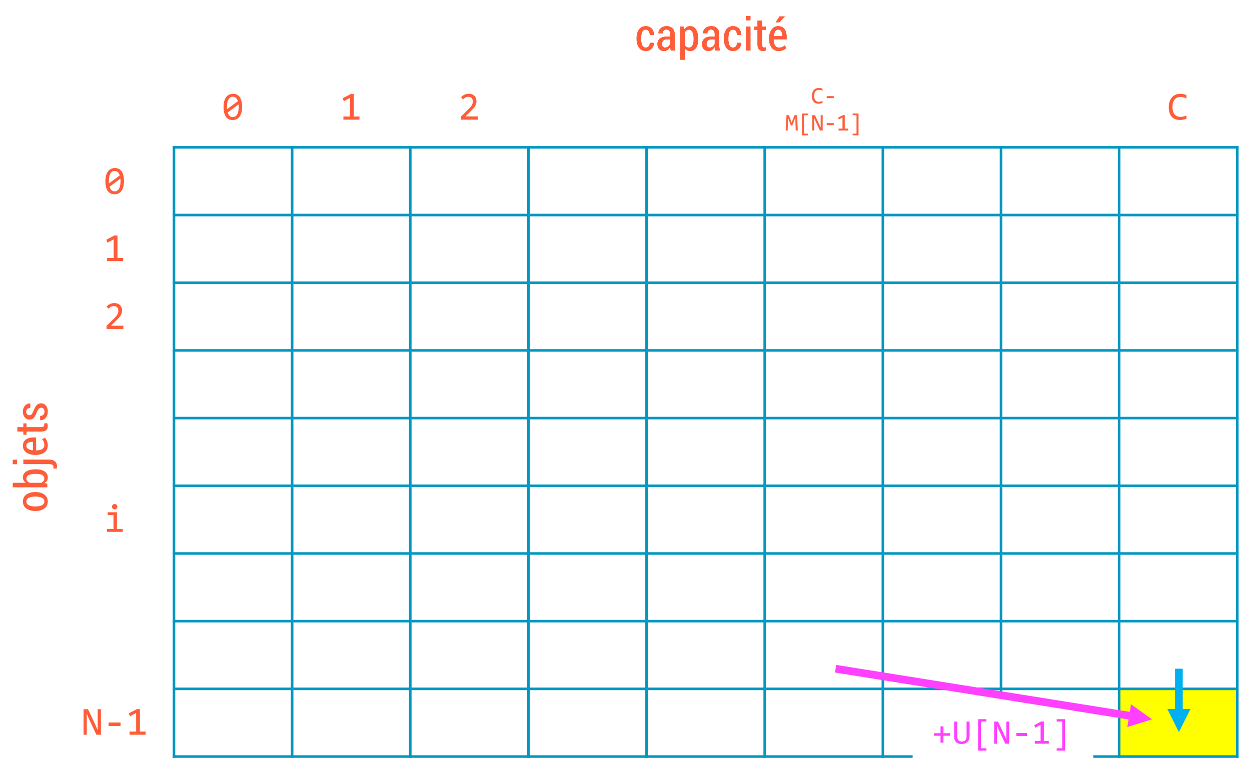
l'utilité maximale
que l'on peut obtenir

Résolution du problème

On peut alors construire le tableau **`E`** de dimension **`N`** tel que **`E[i]=1`** si l'objet **`i`** doit être emporté et **`E[i]=0`** sinon. Pour cela, il faut remonter dans le tableau pour retrouver les décisions qui ont permis d'atteindre **`S[N-1][C]`**.

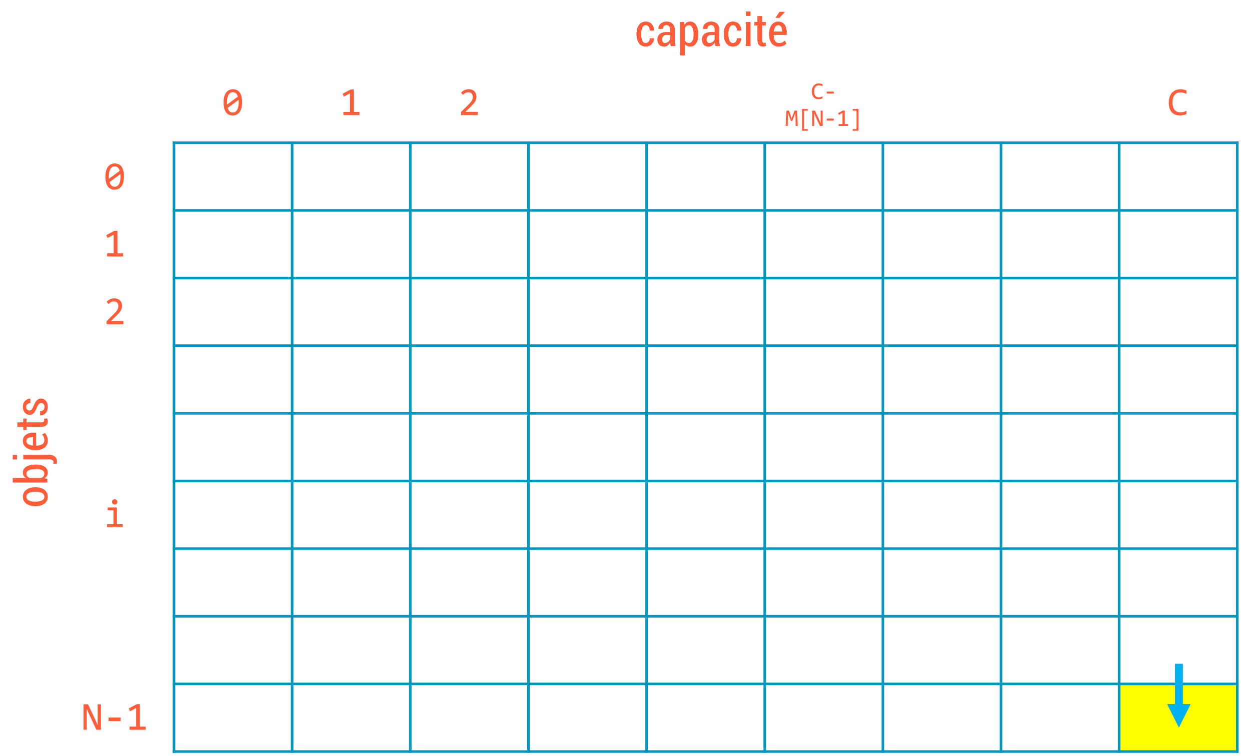


Résolution du problème



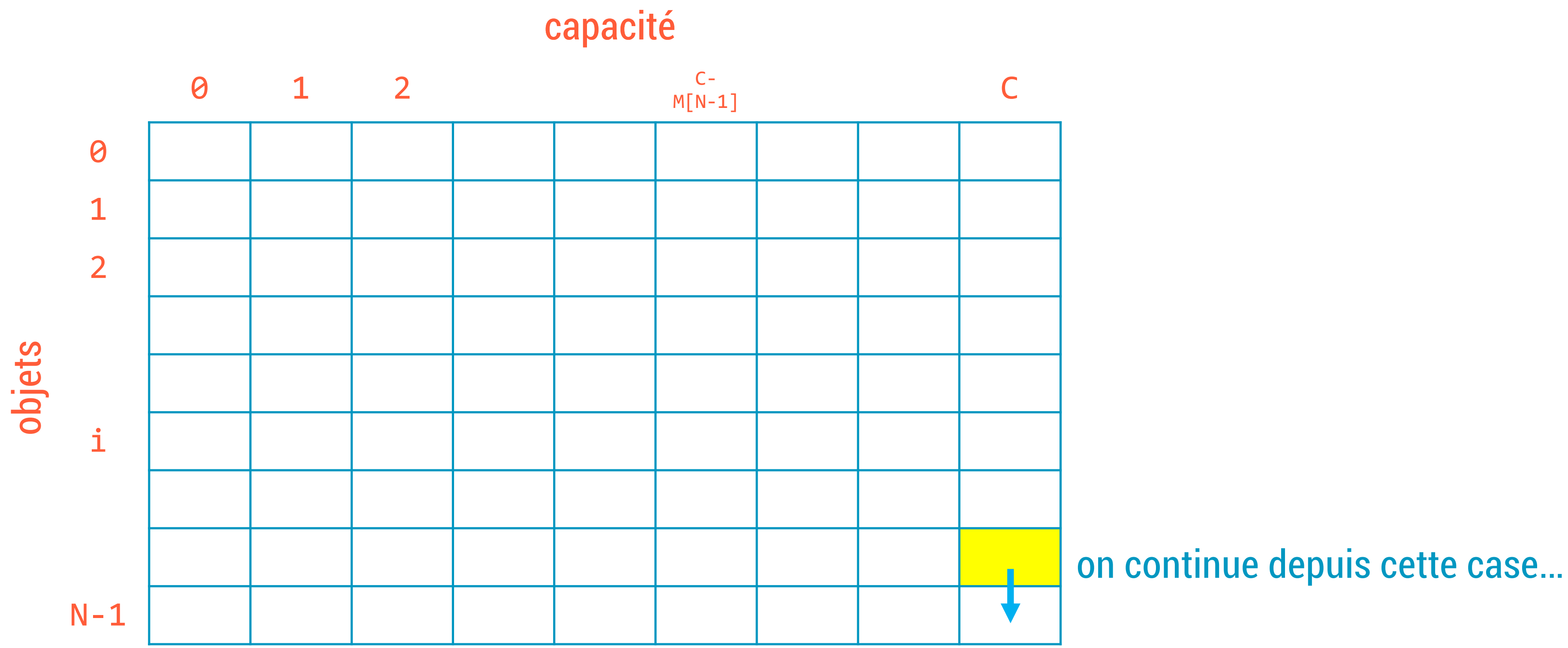
faut-il prendre l'objet `N-1`
pour atteindre cette valeur ?

Résolution du problème

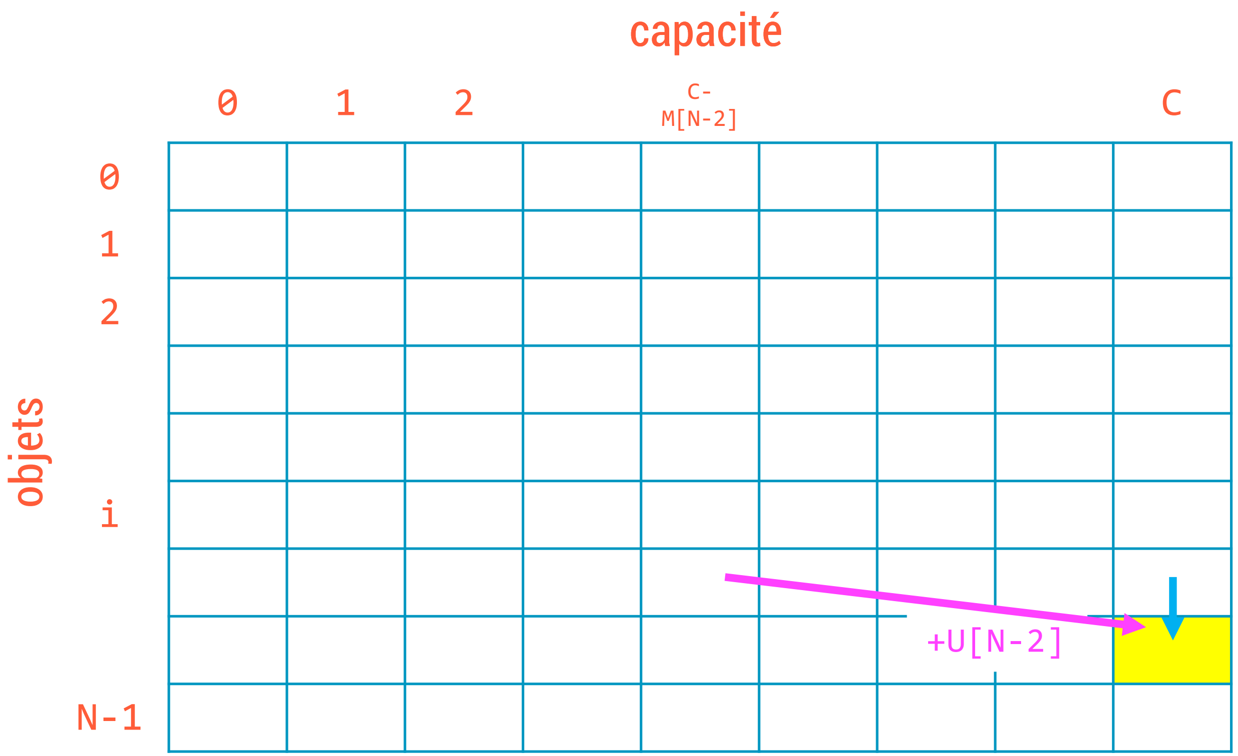


E[N-1] = 0

Résolution du problème



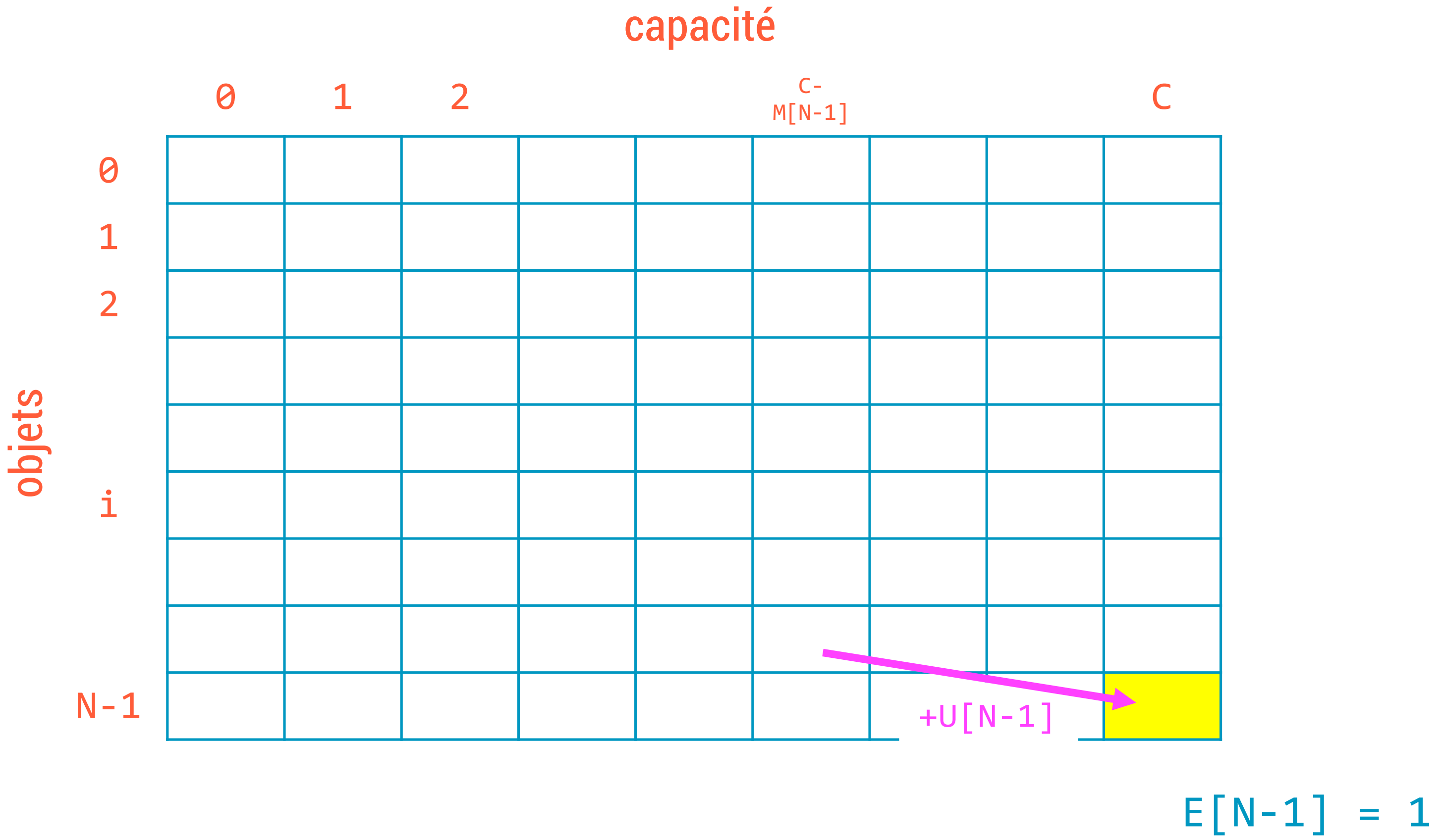
Résolution du problème



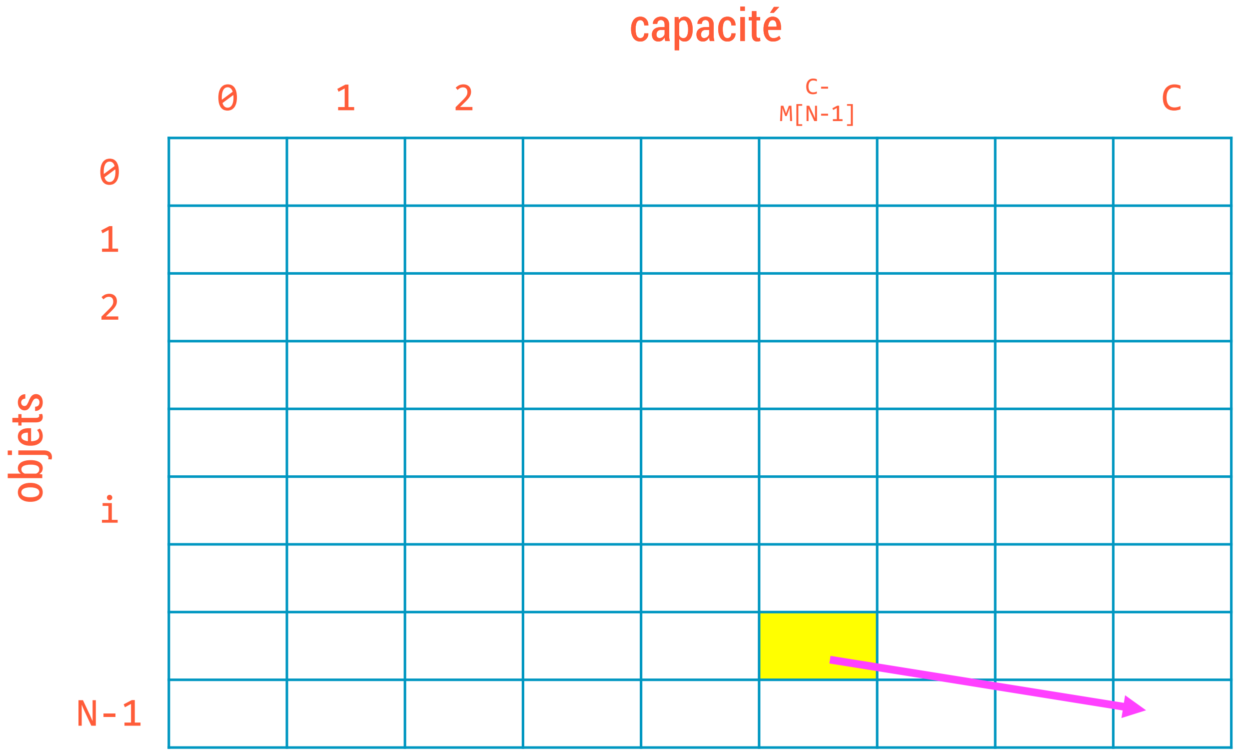
on continue depuis cette case…

E[N-2] = …

Résolution du problème

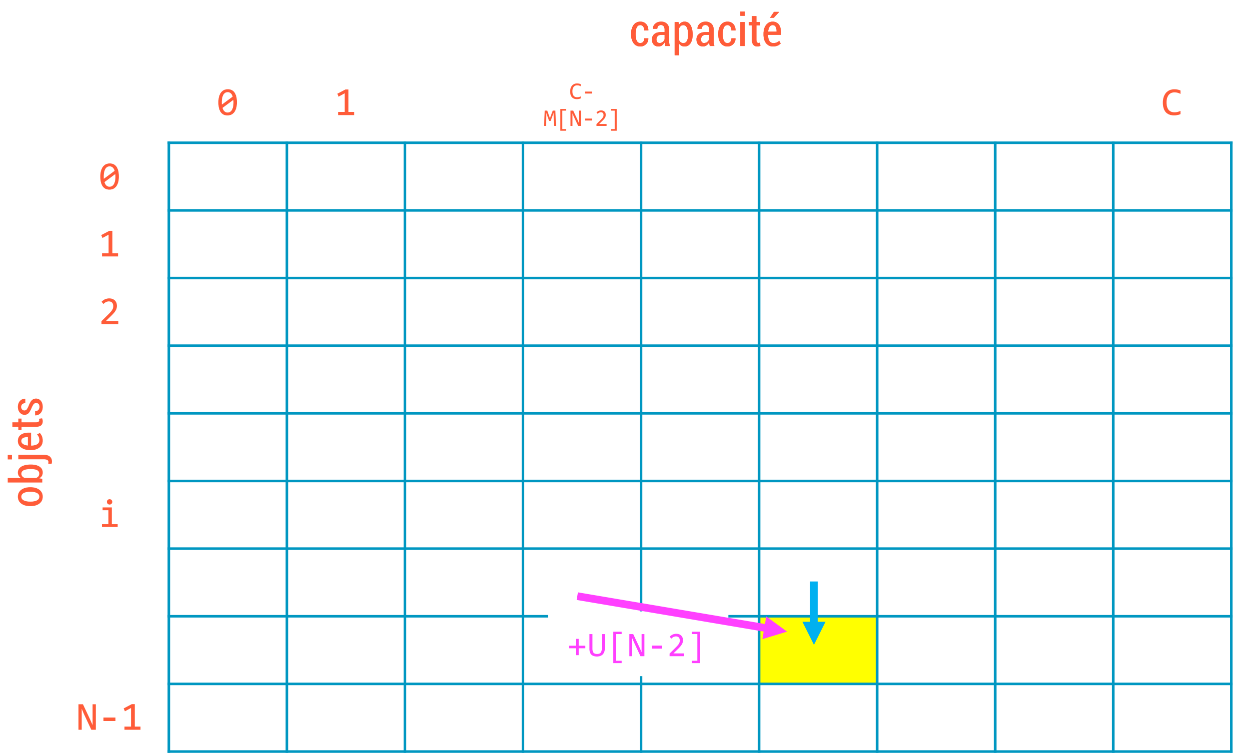


Résolution du problème



on continue depuis cette case…

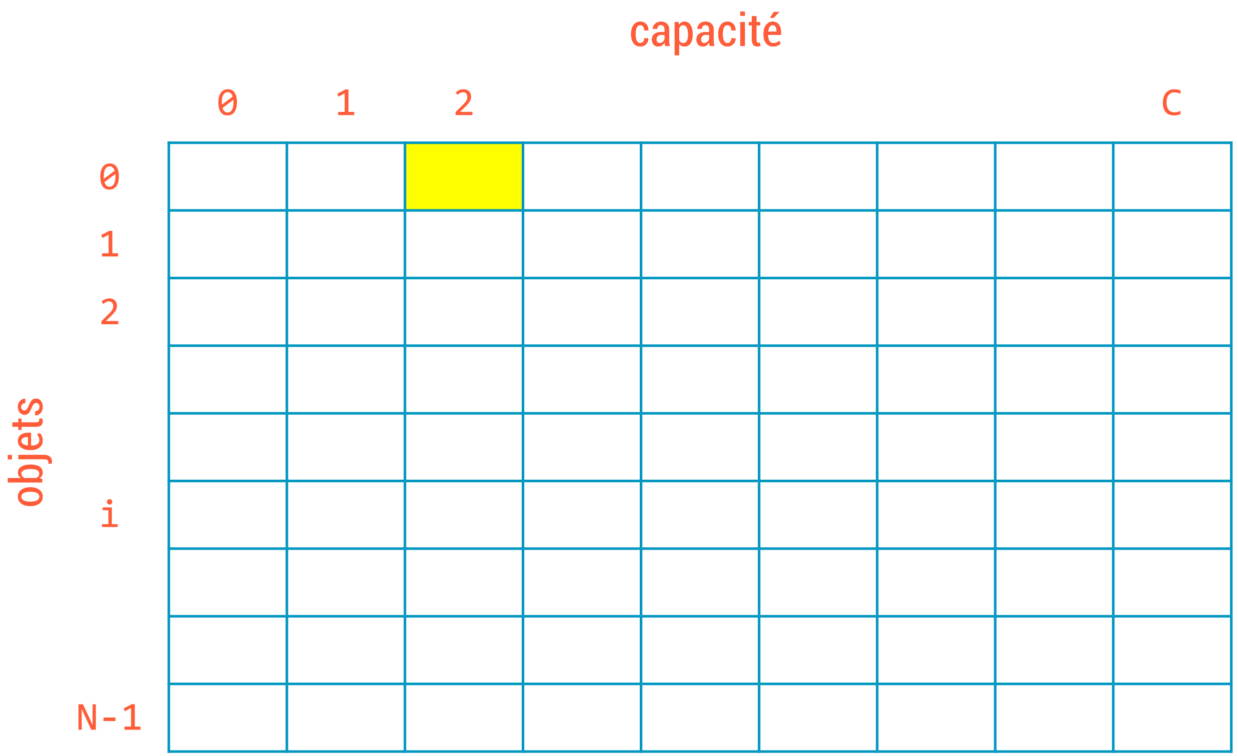
Résolution du problème



on continue depuis cette case…

E[N-2] = …

Résolution du problème



`E[0] = …`