UN EXEMPLE D’ALGORITHME

PSEUDO-POLYNOMIAL

Le problème SSP

Etant donné l'ensemble de nombres naturels $S=\{x_1, x_2, \ldots, x_n\}$ et un nombre naturel t, on pose la question s'il existe un sous-ensemble de S, dont la sommme des éléments soit t.

Notation

- on notera "L + *x*", la liste triée obtenue par addition de *x* à chaque élément de la liste triée L

- on notera "merge(L,L')" la liste triée et sans doublons obtenue à partir des listes triées L et L'

- on notera "supr(L,*k*)" la liste triée obtenue à partir de la liste triée L par suppression des éléments supérieurs à *k*

Algo exponentiel pour SSP

$n \leftarrow |S|$

$L_0 \leftarrow \langle 0 \rangle$

pour i = 1 à n faire

  $L_i \leftarrow$ merge($L_{i\text{-}1}$, $L_{i\text{-}1} + x_i$)

  $L_i \leftarrow$ supr(L_i, t)

if max(L_n) = t

  then return(OUI)

  else returrn(NON)

Exemple

S={1, 4, 7, 10}

t=18

L_0={0}

L_1={0, 1}

L_2={0, 1, 4, 5}

L_3={0, 1, 4, 5, 7, 8, 11, 12}

L_4={0, 1, 4, 5, 7, 8, 10, 11, 12, 14, 15, 17, 18}

Résultat : 18, donc OUI

Le même exemple encore

S={10, 7, 4, 1}

t=18

L_0={0}

L_1={0, 10}

L_2={0, 7, 10, 17}

L_3={0, 4, 7, 10, 11, 14, 17}

L_4={0, 1, 4, 5, 7, 8, 10, 11, 12, 14, 15, 17, 18}

Résultat : 18, donc OUI

Exemple de l'exponentialité

$S=\{1, 2, 4, 8, 16, \ldots\}=\{2^{i-1} \mid i=1, \ldots, n\}$

t assez grand $\Theta(2^n)$

$L_1=\{0, 1\}$

$L_2=\{0, 1, 2, 3\}$

…

$L_i = \{0, 1, \ldots, 2^i-1\}$

Comme la taille des L_i est exponentielle, l'algorithme est exponentiel.

Oui mais ….

Si on considère V la plus grande valeur des données.

Alors la taille des L_i est bornée par t, donc est bornée par V.

Le nombre d'étapes de calcul – le nombre de L_i à calculer est borné par n, la taille de l'ensemble S et donc par N la taille des données.

Ainsi

Dans l'algorithme on fait trois types de calcul

- **L + *x* – se fait en temps O(V)**
- **merge(L,L') – se fait en temps O(V)**
- **supr(L,*k*) – se fait en temps O(V)**

Ainsi la complexité de l'algorithme est en O(NV).

C'est exactement la définition d'un algorithme pseudo-polynomial.

Méthodes de preuve de NP-complétude

- **transformations identité**

  (*restrictions*, *cas particuliers*)

  CLIQUE, SSP, PM, ACDB, …
- **transformations locales**

  3-SAT, Partition, PPET, …
- **transformations globales**

  SAT, VC, CIRCUITHAM, …

Une dernière remarque concernant la NP-complétude

Problèmes de reconnaissance vs optimisation

Si on sait reconnaître en temps polynomial, alors en un nombre O(log *Rés*) de reconnaissances on peut trouver l'optimum.

Si on sait trouver l'optimum alors on sait répondre au problème de reconnaissance.

Remarque : pour les problèmes d'optimisation on parle de NP-difficulté seulement.

Conclusion

Que peut-on faire quand-même ?

- **heuristiques**
- **algorithmes d'approximation**

Algorithmes d'approximation

L'idée

Ayant un problème NP-complete, que fait-on ?

ABANDONNER ?

- **Si possible – algo exponentiel**
- **Sinon, algorithme polynomial d'approximation.**
- **S'il existe, algorithme probabiliste**

Approximation ?

Qu'est-ce ?

Soit Π un pb de minimisation (optimisation)

- **C^* - la valeur d'une solution optimale**
- **C - la valeur d'une solution approchée**

$$C > C^*$$

Ratio bound $\max\{C/C^*, C^*/C\} \leq \rho(n)$

Ratio bound

$$\max\{C/C^*, C^*/C\} \leq \rho(n)$$

Erreur relative

$$|C - C^*|/C^* \leq \varepsilon(n)$$

Relation

$$\varepsilon(n) \leq \rho(n) - 1$$

Le problème Bin Packing

NOM : **BIN PACKING**

DONNEES : **ensemble fini d'éléments U (taille $\in$N) et une taille B $\in$N.**

QUESTION : **Quel est le plus petit *k* tel qu'on peut partitionner U en U_1, U_2, …, U_k sans que la somme des tailles des éléments des U_i dépasse B ?**

Le problème de décision Bin Packing

NOM : BIN PACKING

DONNEES : ensemble fini d'éléments U (taille $\in$N), une taille B $\in$N et un nombre k $\in$N.

QUESTION : Peut-on partitionner U en $U_1, U_2, \ldots, U_k$ sans que la somme des tailles des éléments des U_i dépasse B ?

La NP-complétude

Théorème :

BIN PACKING est NP-complet.

Preuve :

i) **BIN PACKING $\in$ NP**

ii) **BIN PACKING est NP-difficile**

nous le montrons par

PARTITION $\propto$ BIN PACKING

La transformation

BIN PACKING est une généralisation de **PARTITION** !

En effet, si B est la moitié de la somme des tailles des éléments et k=2, alors un **BIN PACKING** existe si et seulement si un **PARTITION** existe.

(ou … **PARTITION** est un cas particulier de **BIN PACKING** avec deux bins de taille la moitié de la somme des éléments)

Algorithme pseudo-polynomial

Pour un cas spécial :

- ***n*** **objets (mais seulement** ***k*** **tailles différentes)**
- **bins de taille** ***B***

Ainsi les données deviennent : $I = (i_1, i_2, \ldots, i_k)$**,**
avec i_j **le nombre d'objets de taille** ***j*****.**

On utilise la programmation dynamique.

suite

- BINS(i_1, i_2, …, i_k) est le nombre minimum de bins nécessaires pour $I = (i_1, i_2, \ldots, i_k)$.
- Soit (n_1, n_2, …, n_k) une donnée $\Sigma_i n_i = n$.
- On calcule Q, l'ensemble de tous les k-uplets tels que BINS(q_1, q_2, …, q_k) = 1.

  Au plus O(n^k), donc se calcule en temps O(n^k).

suite

On rempli la table BINS(i_1, i_2, …, i_k), avec $i_j \leq n_j$.

1. $\forall q \in Q$ BINS(q_1, q_2, …, q_k) = 1.
2. Si $\exists\, j$, t.q. $i_j < 0$, alors BINS(i_1, i_2, …, i_k) = ∞.
3. Pour tout autre q, utiliser la récurrence

BINS(i_1, i_2, …, i_k) =

$$1 + \min_{q \in Q} \text{BINS}(i_1 - q_1, i_2 - q_2, \ldots, i_k - q_k).$$

suite

Question : **dans quel ordre faut-il calculer les valeurs de BINS ?**

Complexité :

- **chaque entrée du tableau : O(n^k)**
- **le tableau est de taille : O(n^k)**
- **temps total : O(n^{2k})**

Un heuristique : FF (first fit)

- Pour chaque élément, on examine les bins, et on le place dans le premier où il y a de la place
- Complexité : $\Theta(nk)$ où k est le résultat.

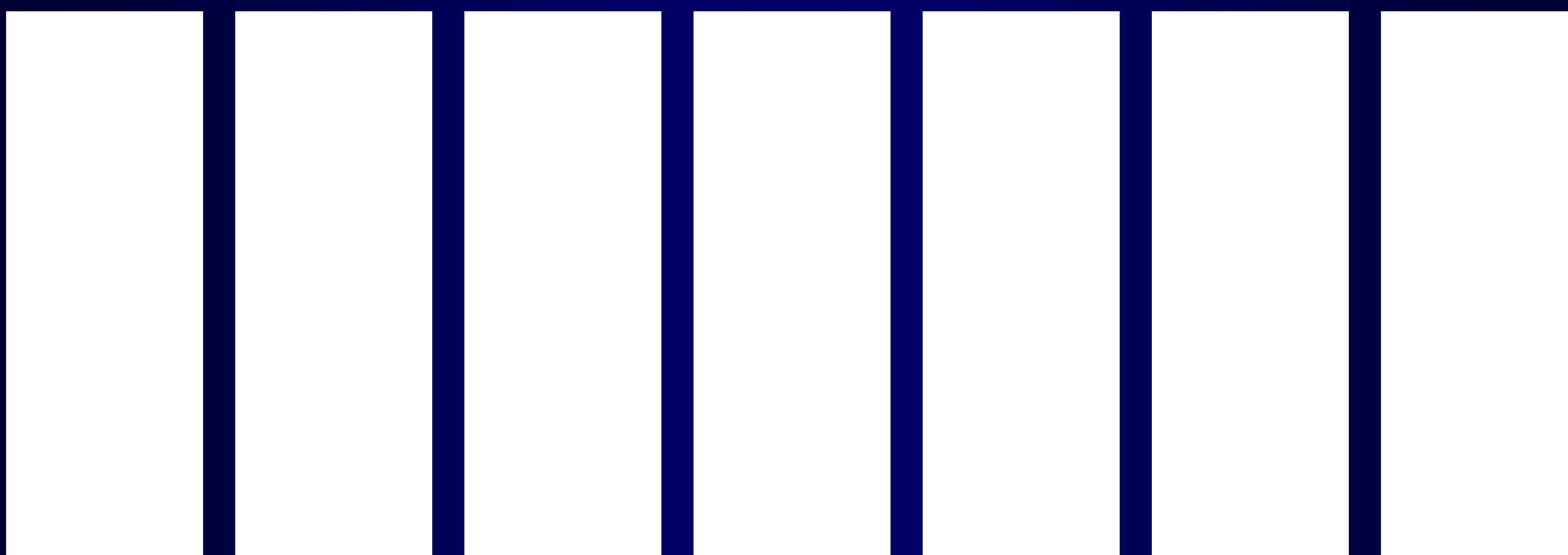
- Est-ce un "bon" heuristique ? Pour une donnée D, soit FF(D) le résultat obtenu par FF et OPT(D) la solution optimale.

FF (suite)

Pour se faire une idée de la "qualité" de l'heuristique, nous proposons deux exemples

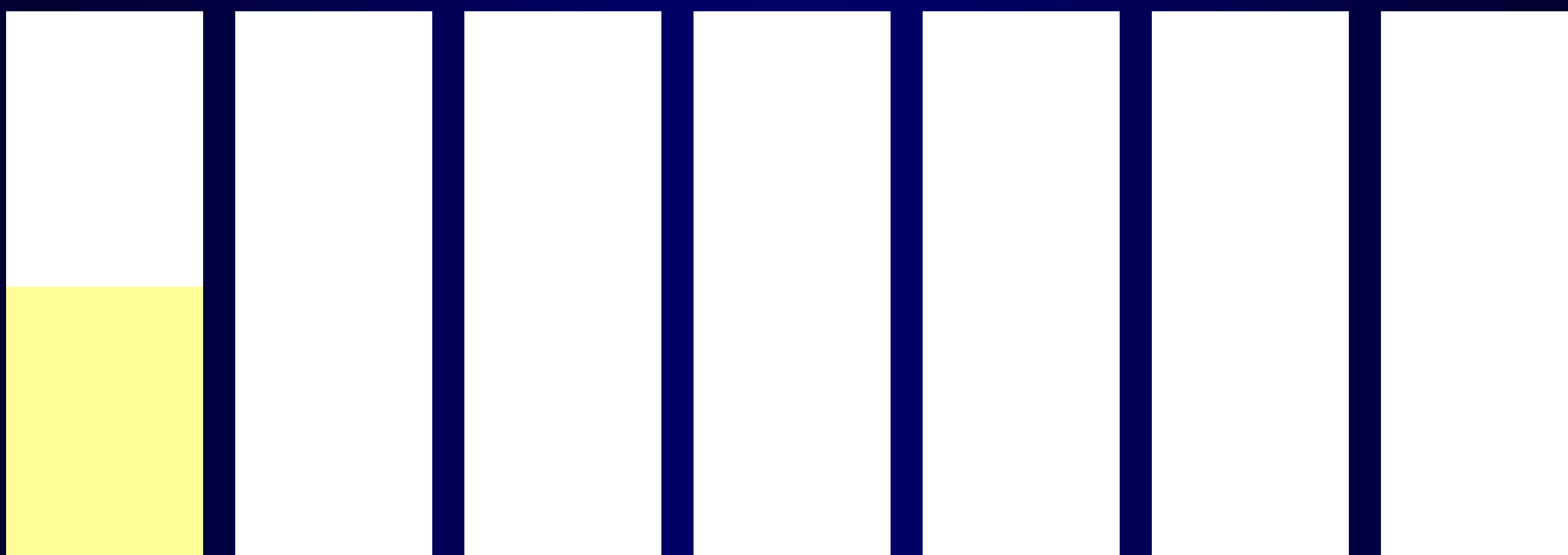
FF (exemple)

B = 420



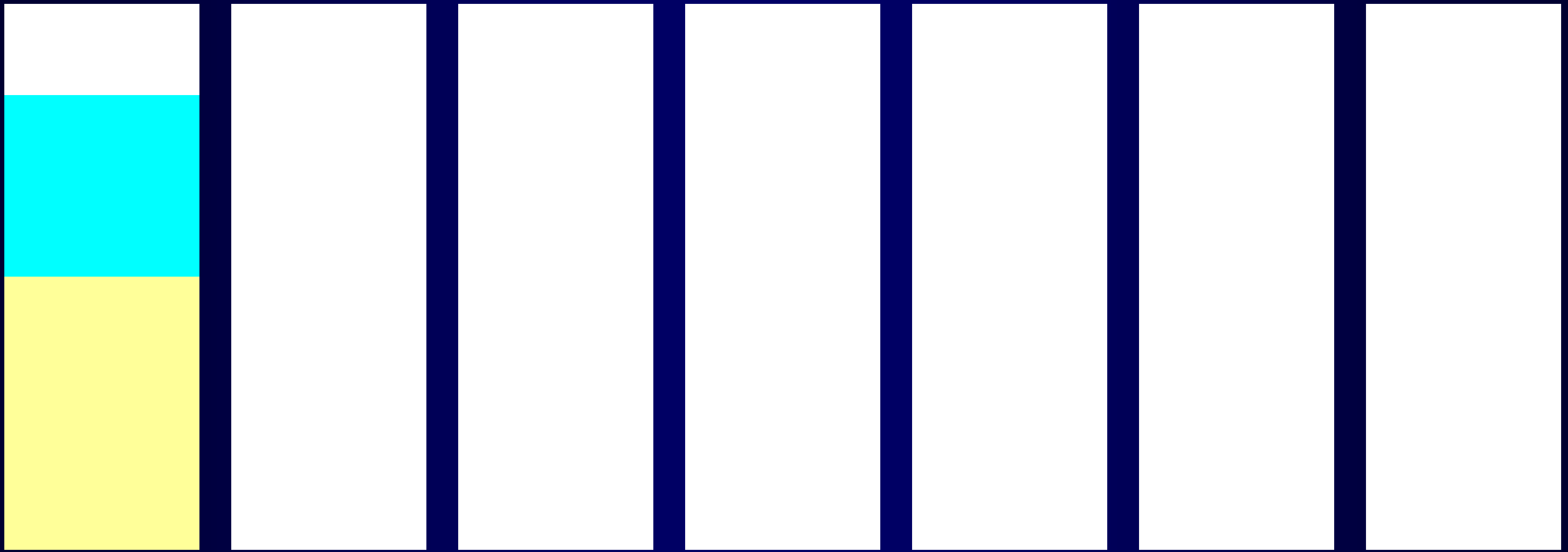
FF (exemple)

B = 420 et 211



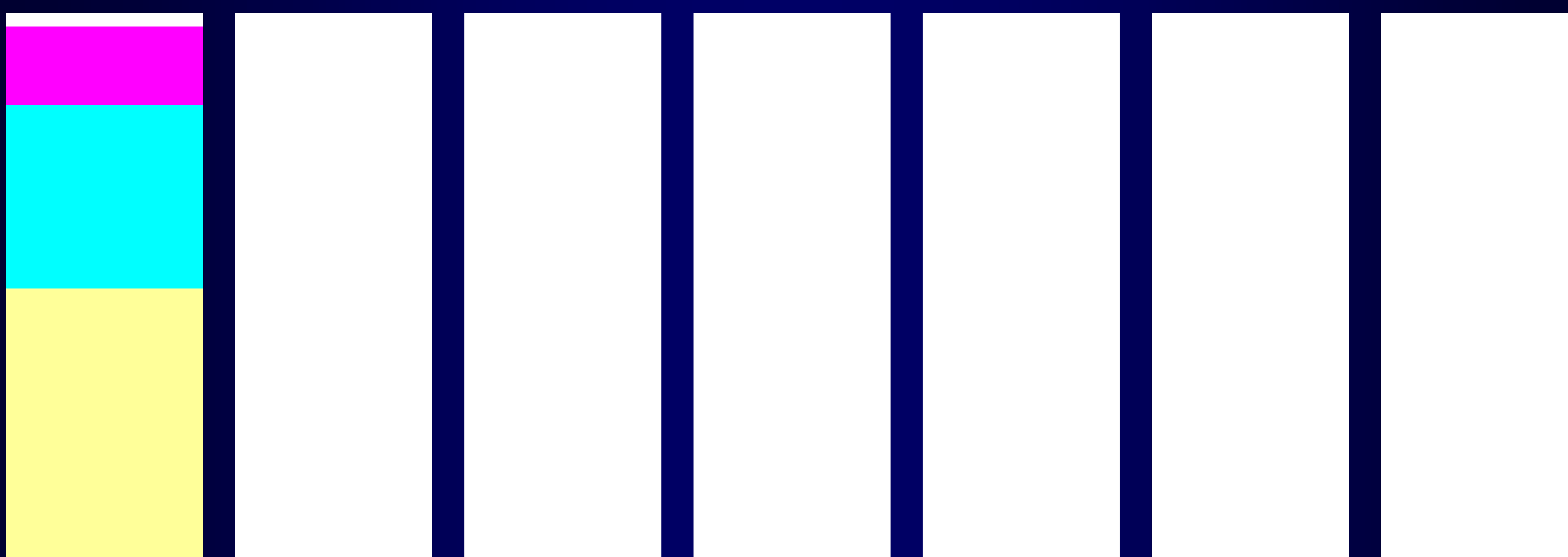
FF (exemple)

B = 420 et 211, 141



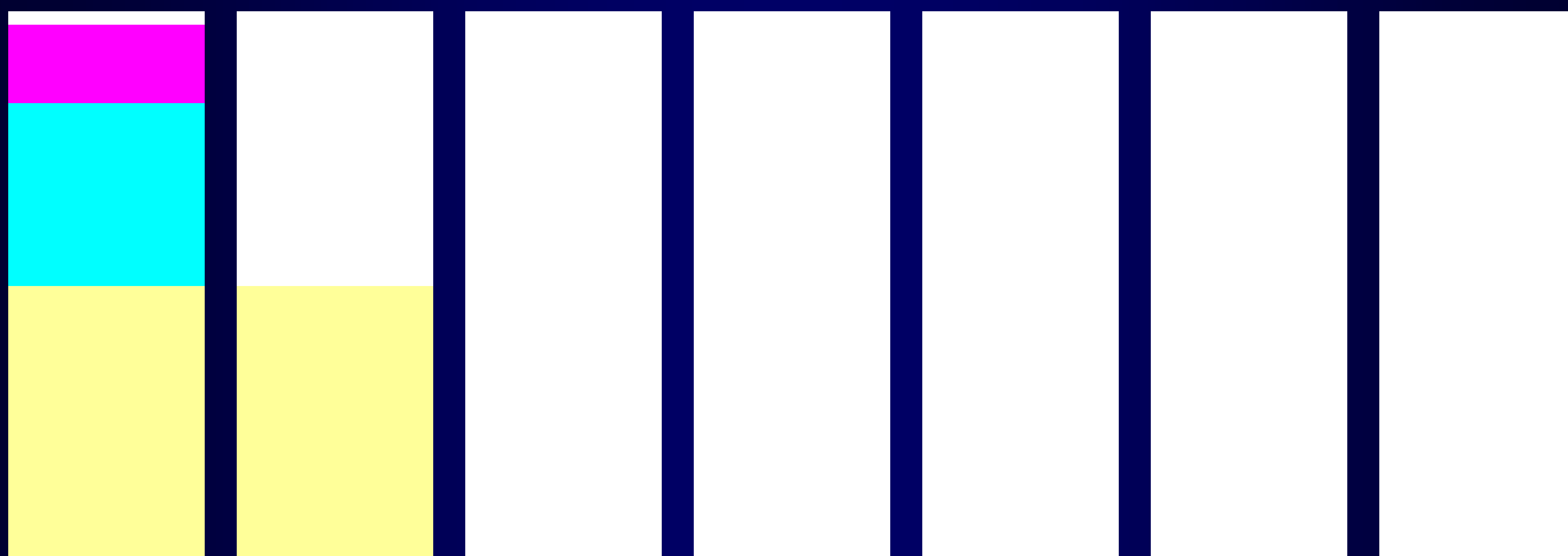
FF (exemple)

B = 420 et 211, 141, 61



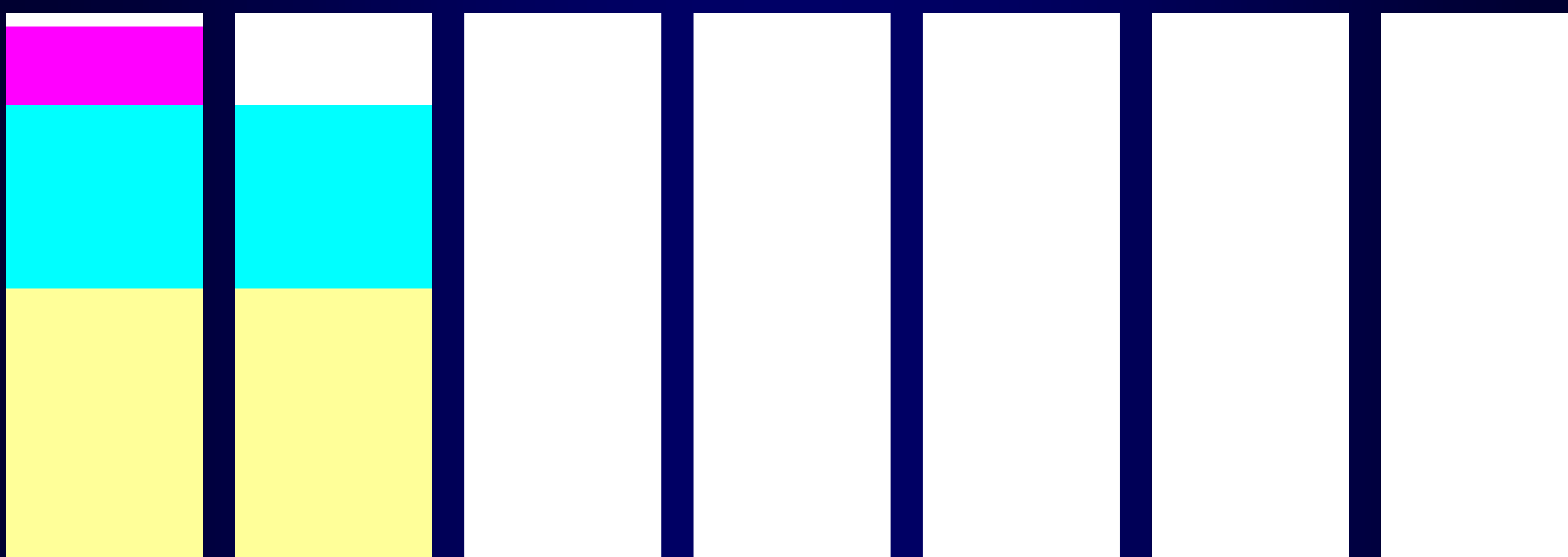
FF (exemple)

B = 420 et 211, 141, 61, 211



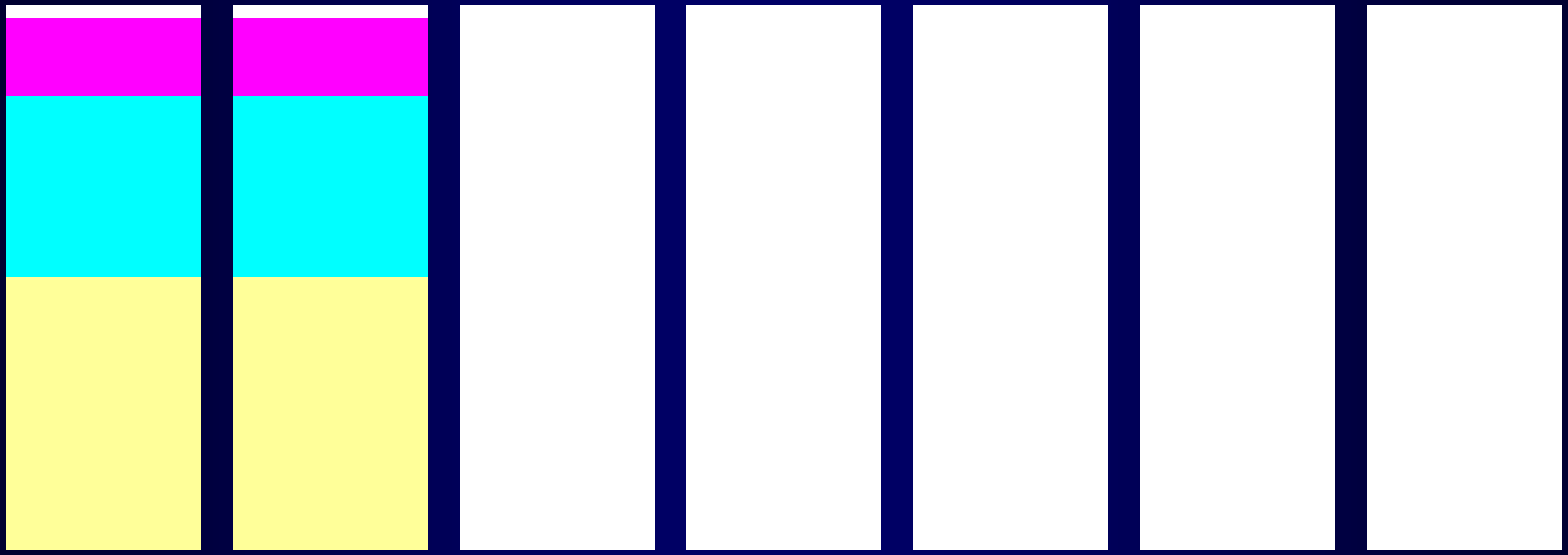
FF (exemple)

B = 420 et 211, 141, 61, 211, 141,



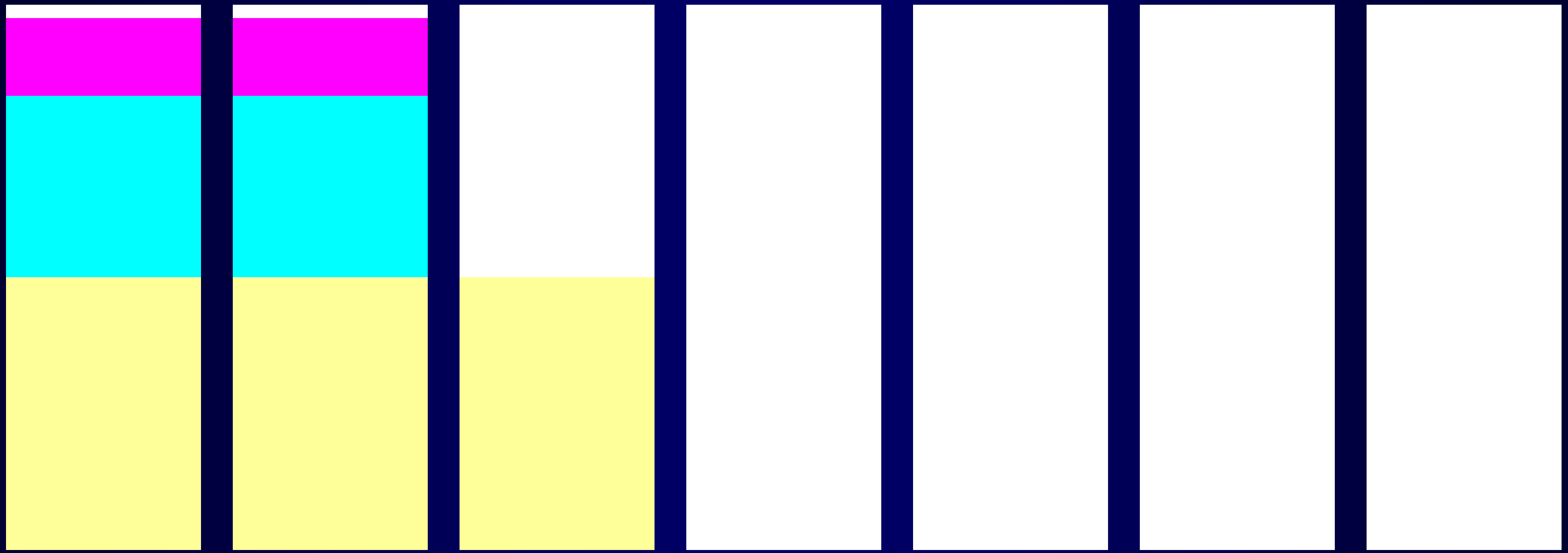
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61



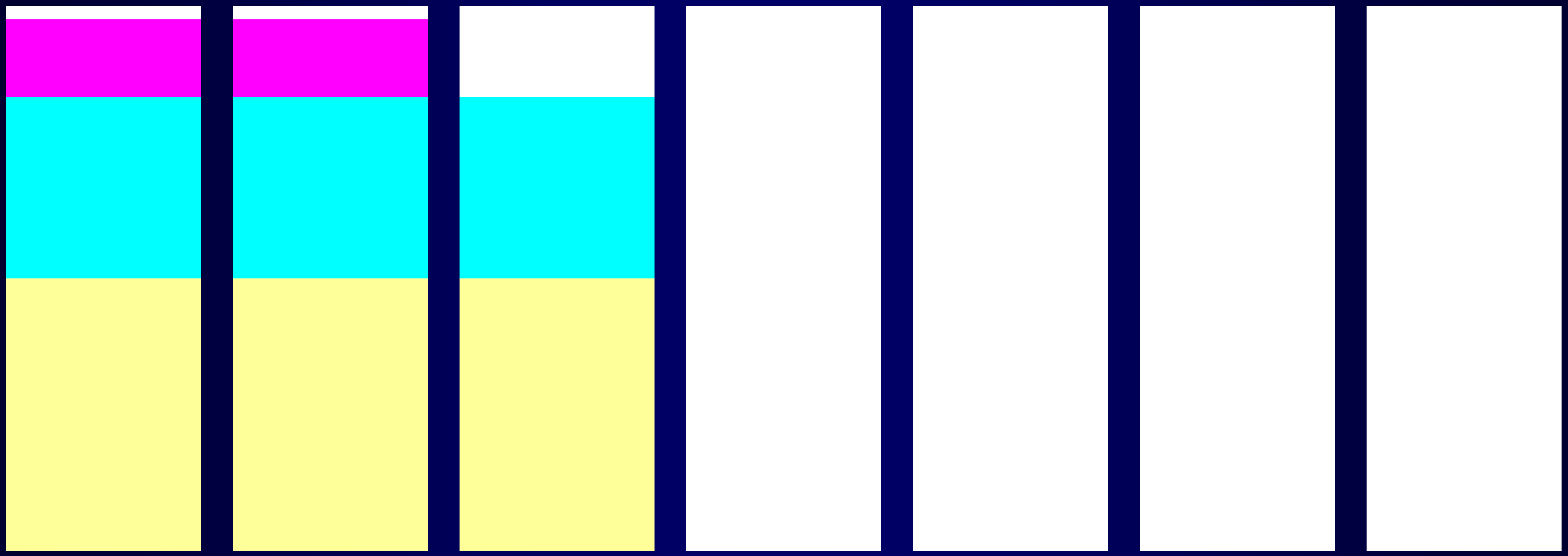
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211



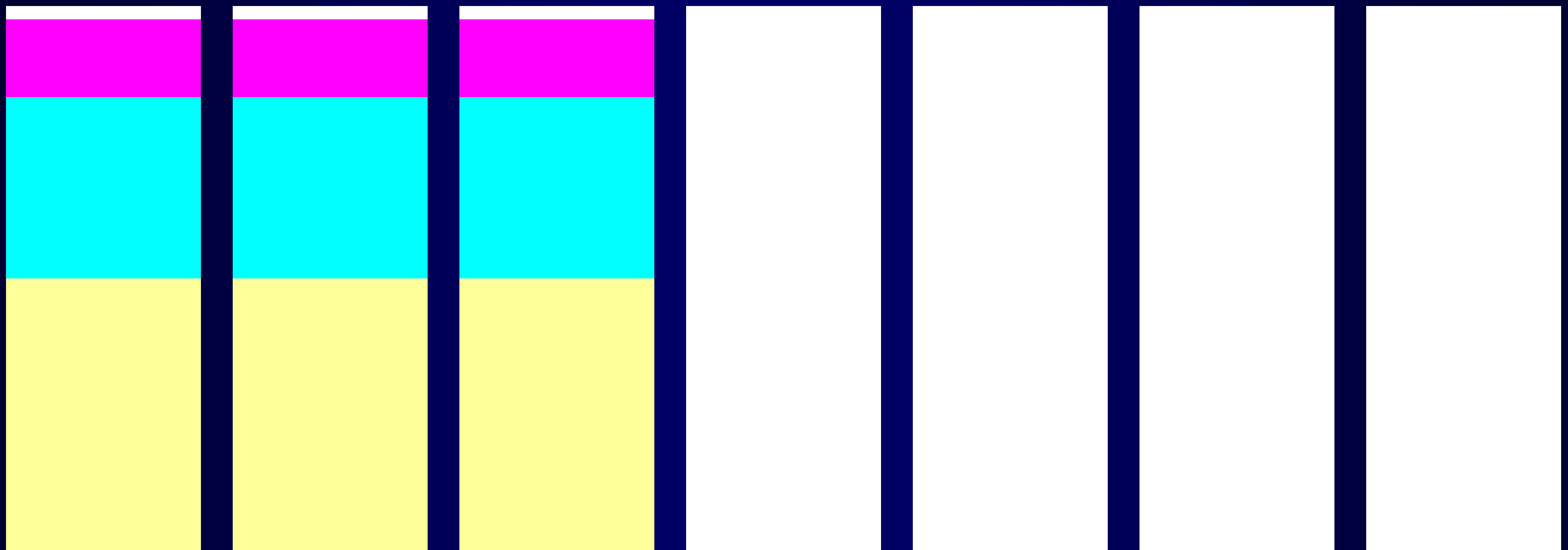
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141



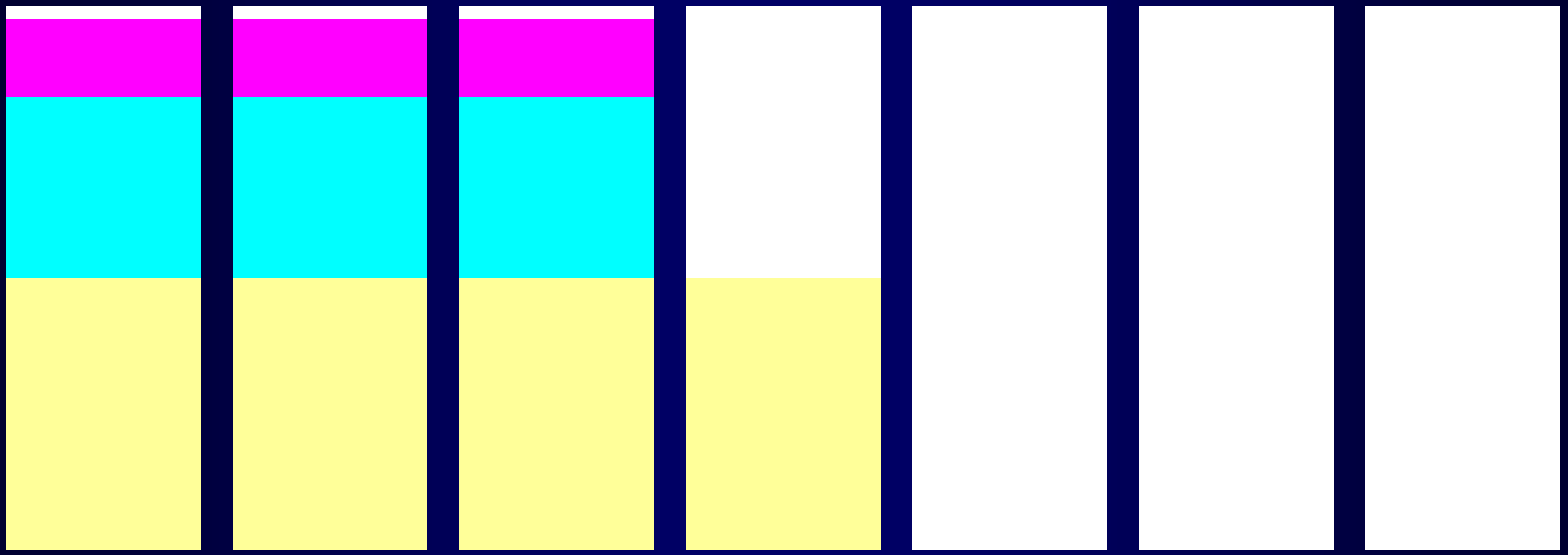
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61



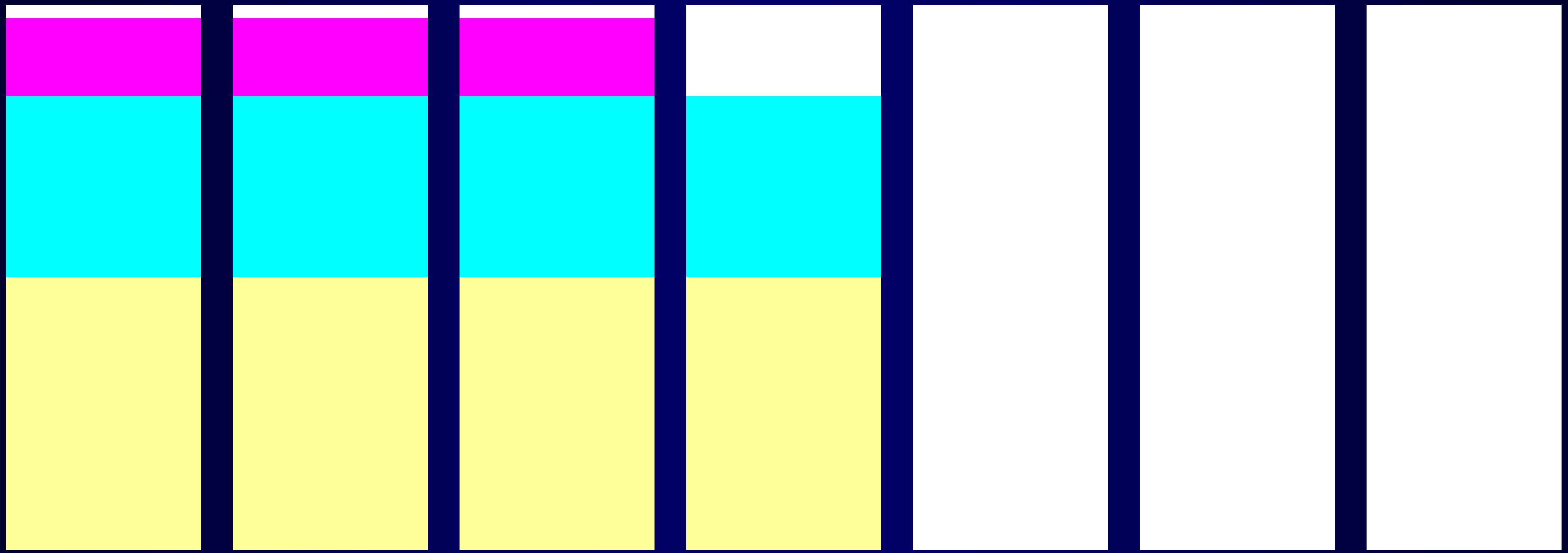
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211



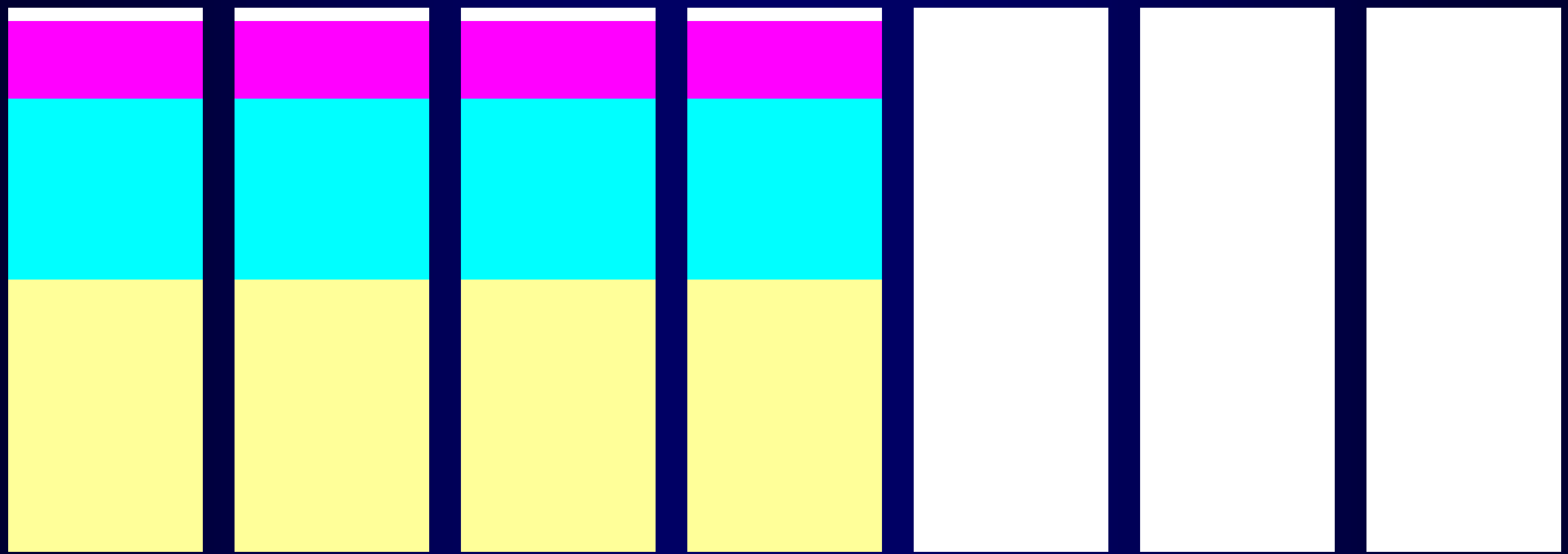
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141



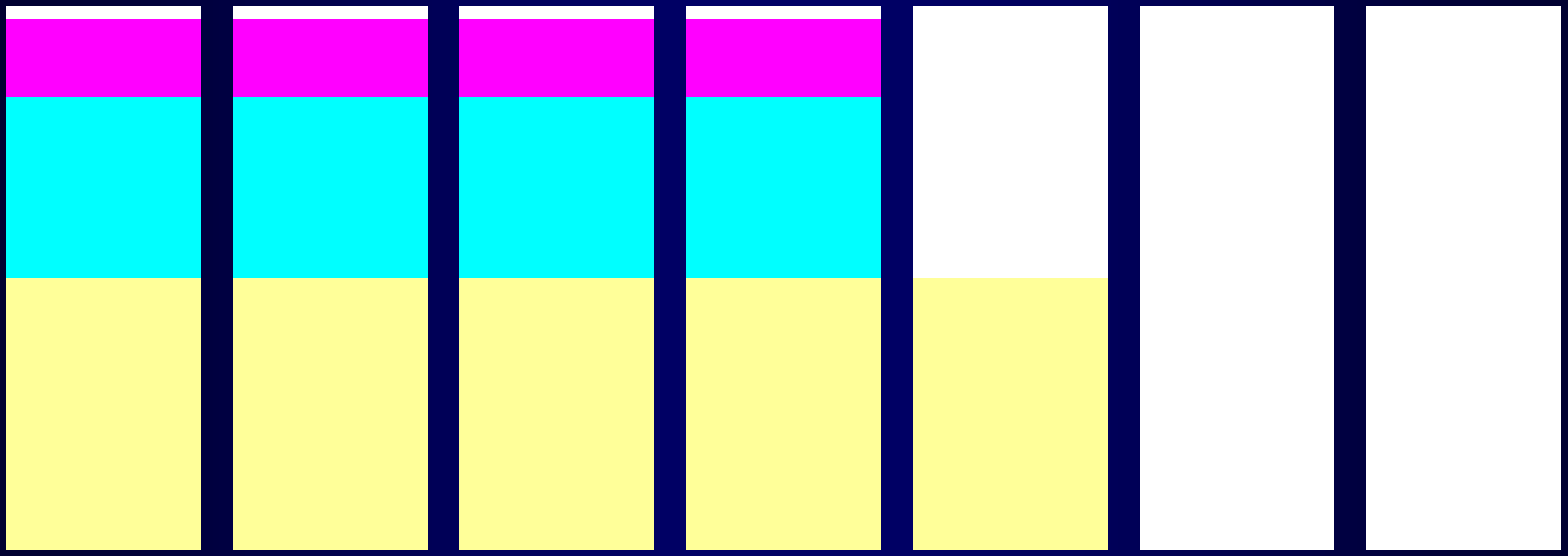
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61



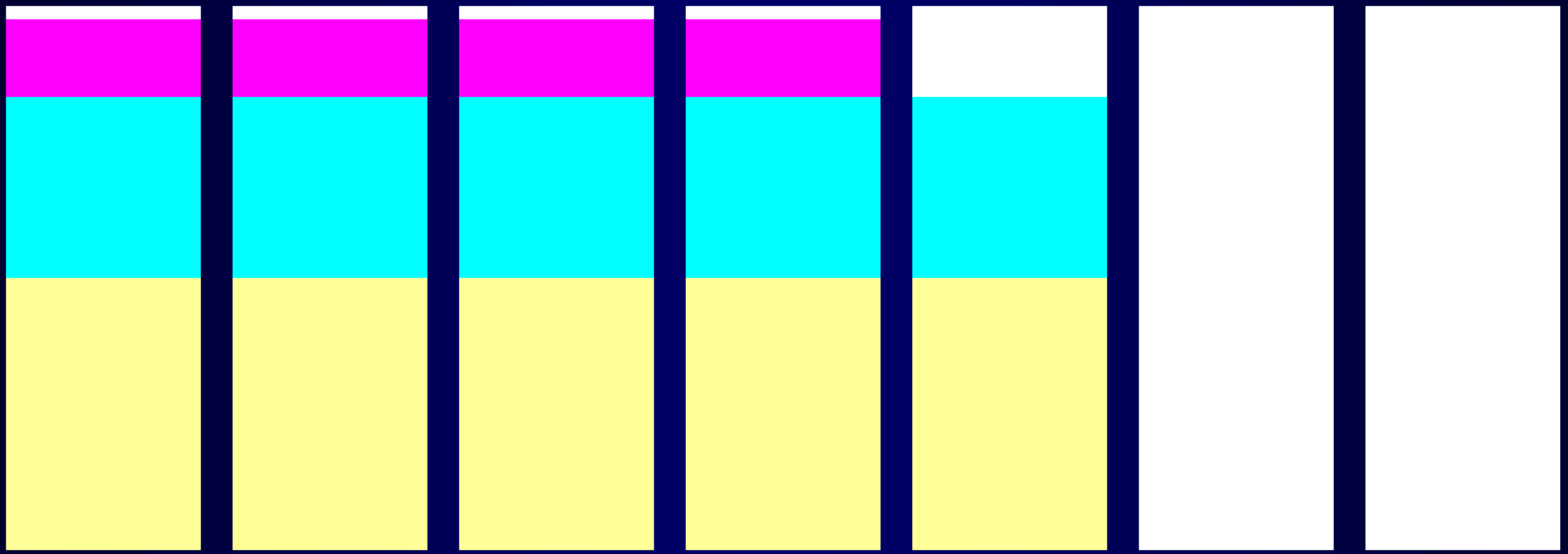
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211



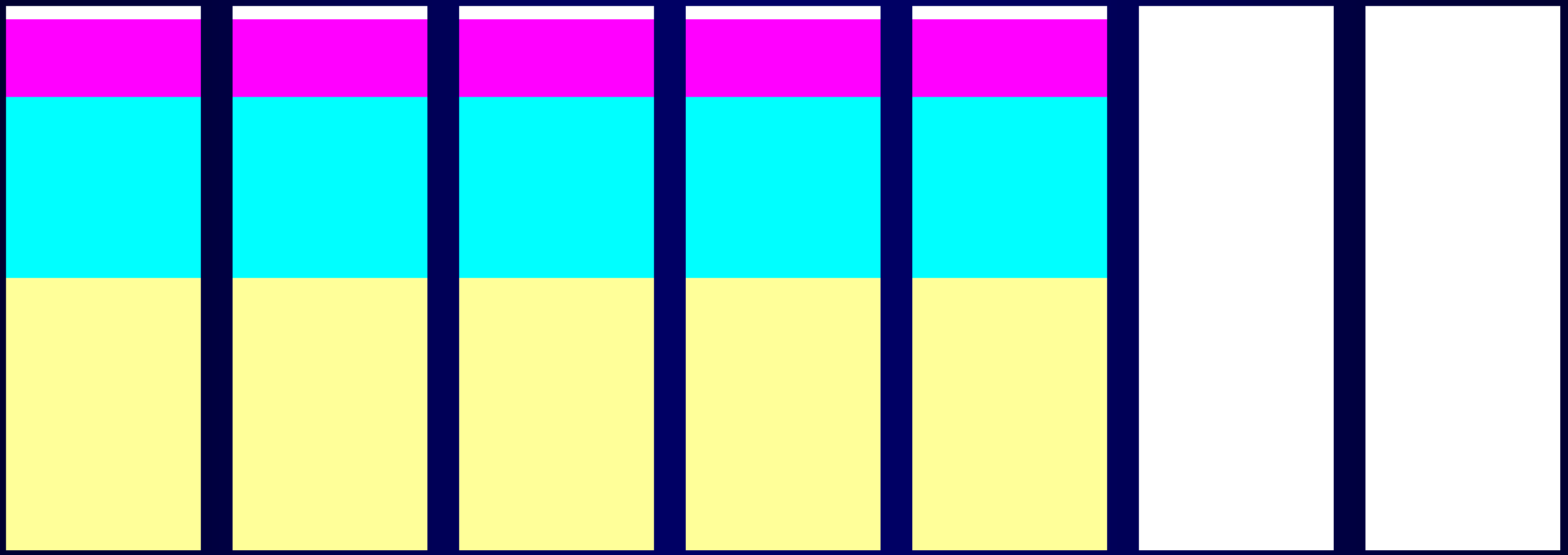
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141



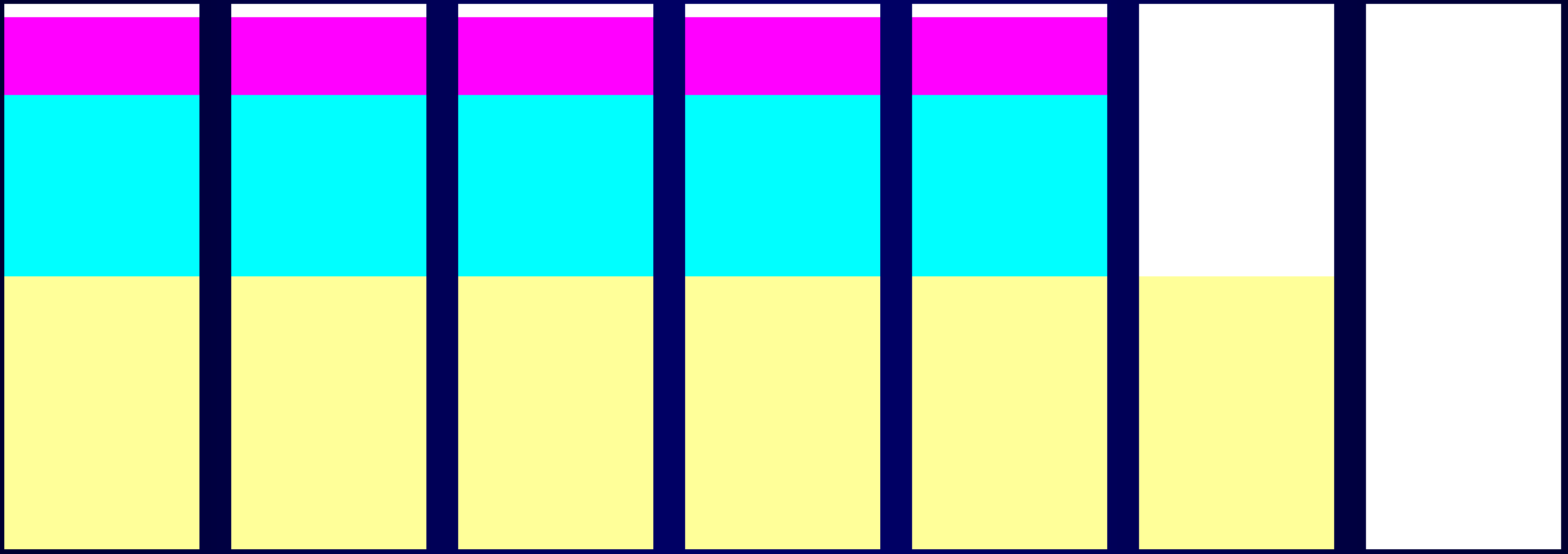
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61



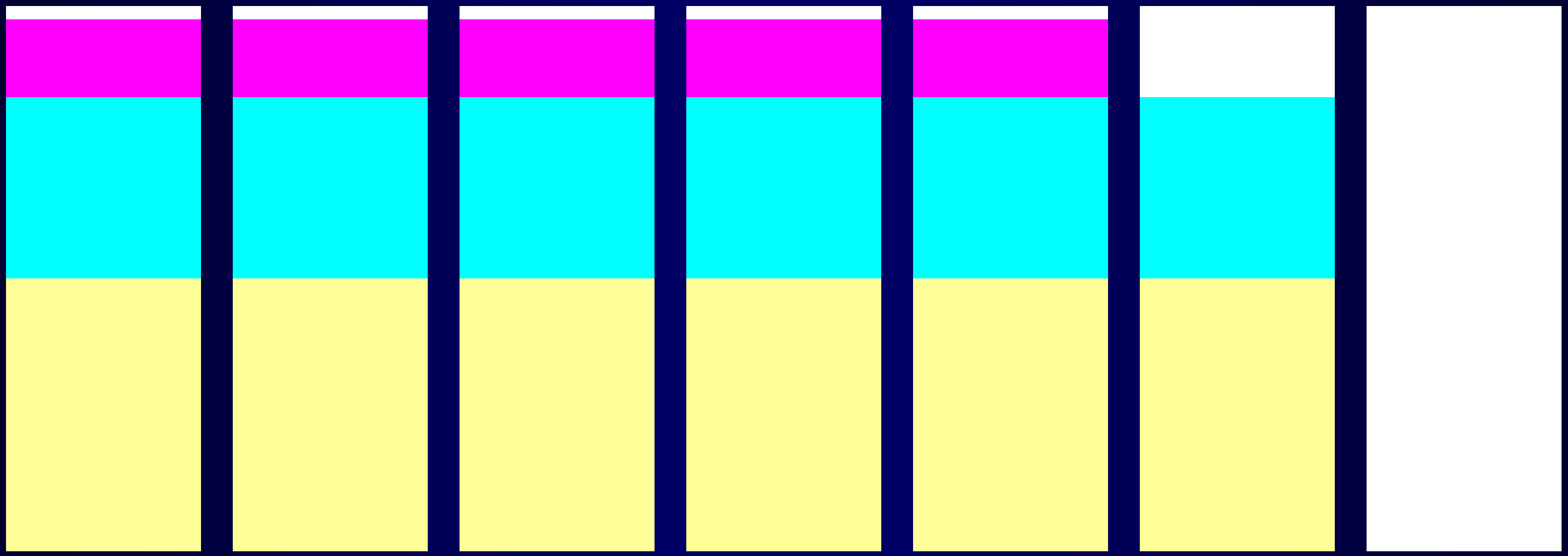
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211



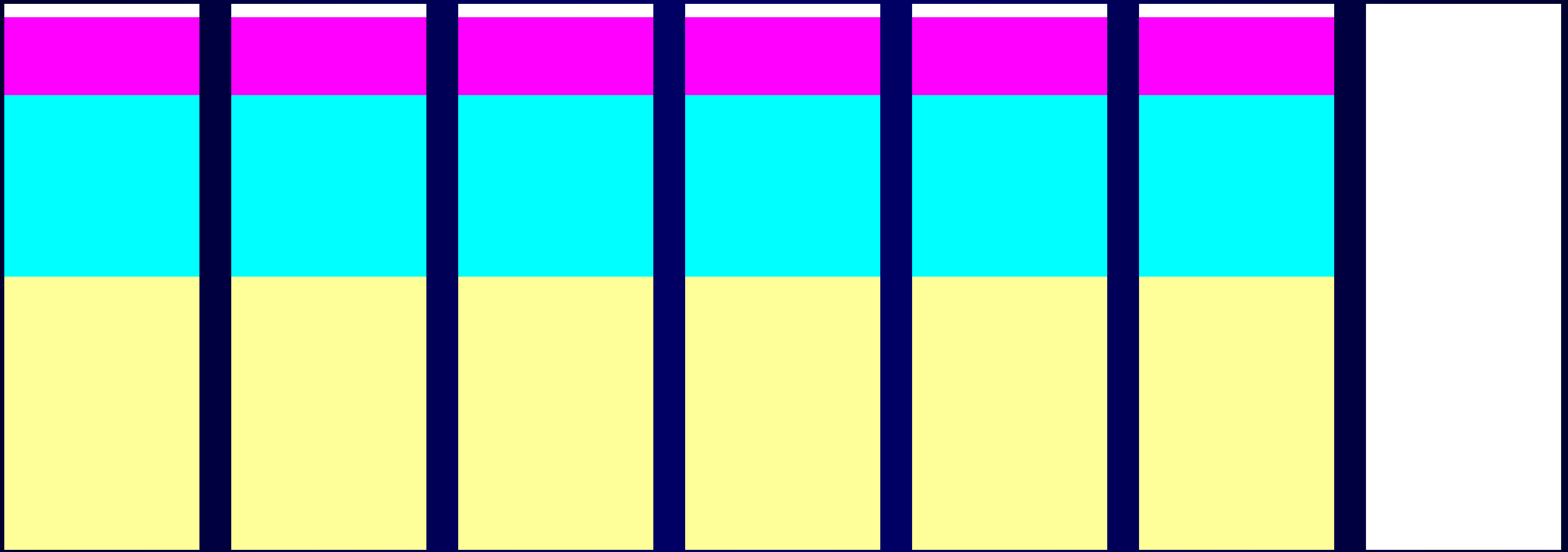
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141



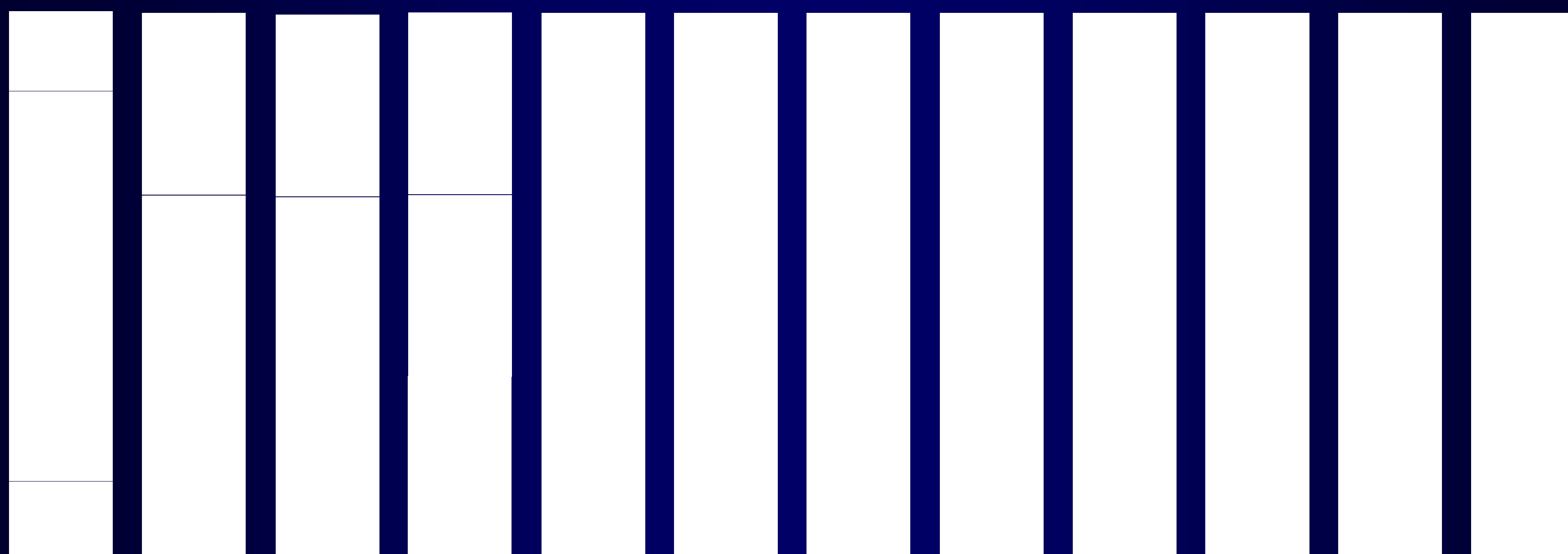
FF (exemple)

B = 420 et 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61, 211, 141, 61.



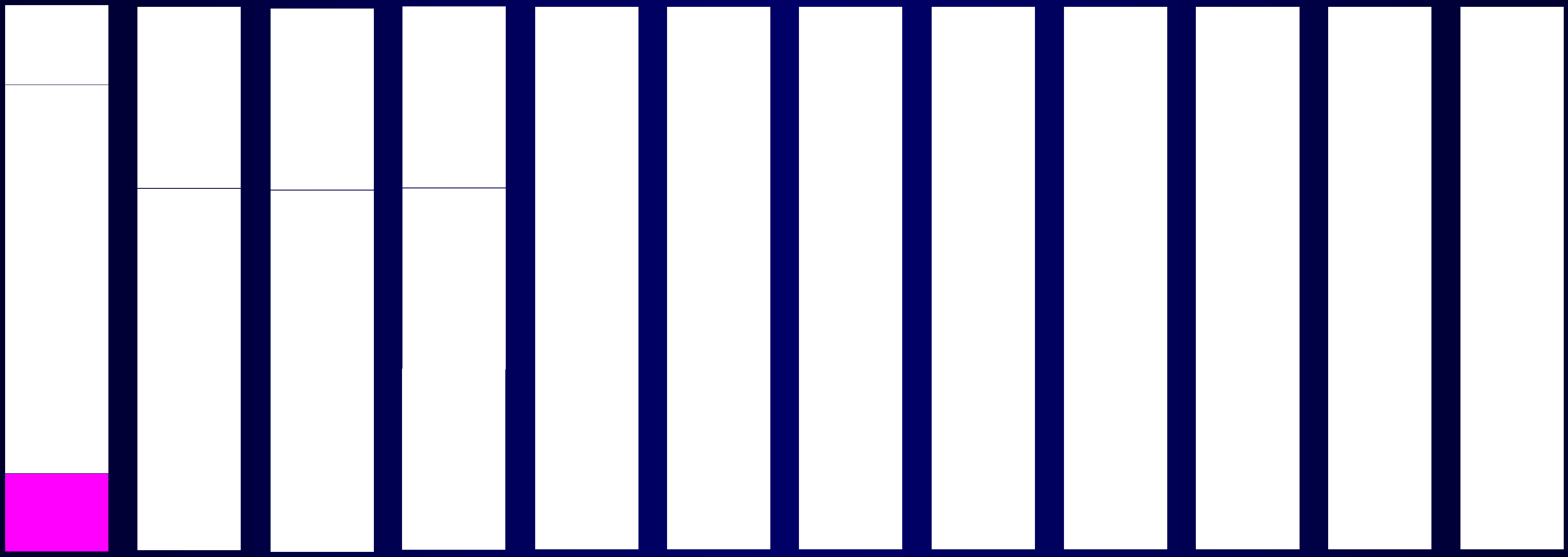
FF (exemple 2)

B = 420 et



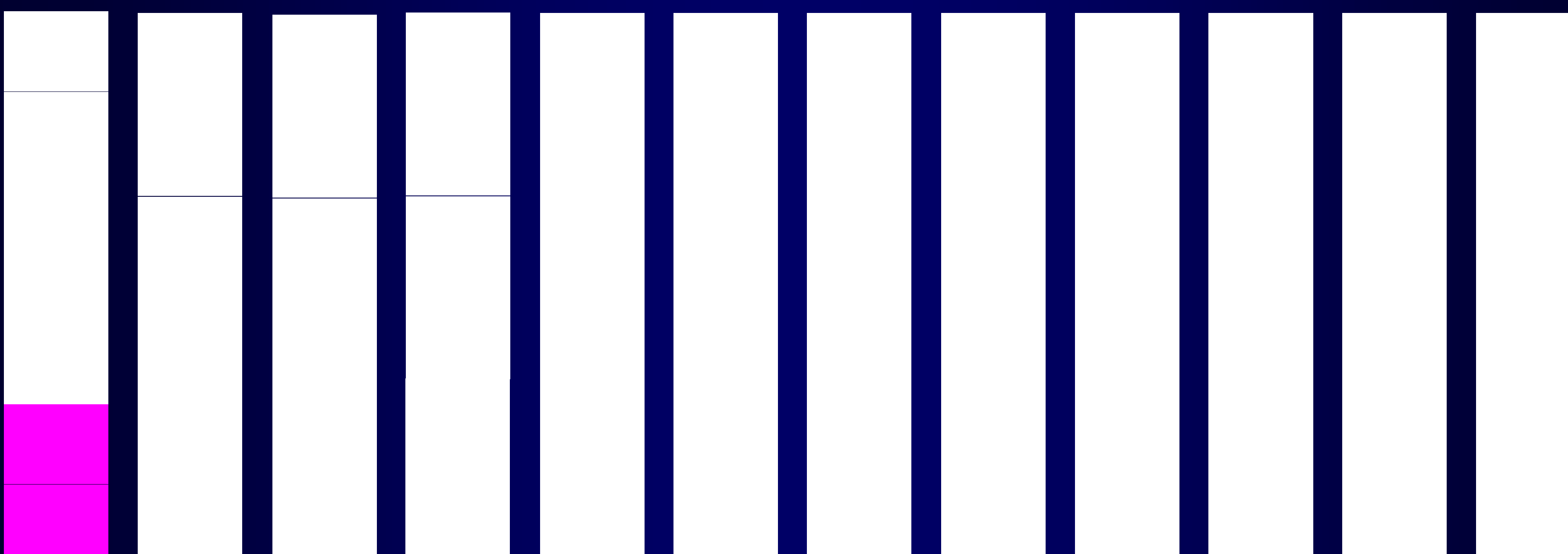
FF (exemple 2)

B = 420 et 61



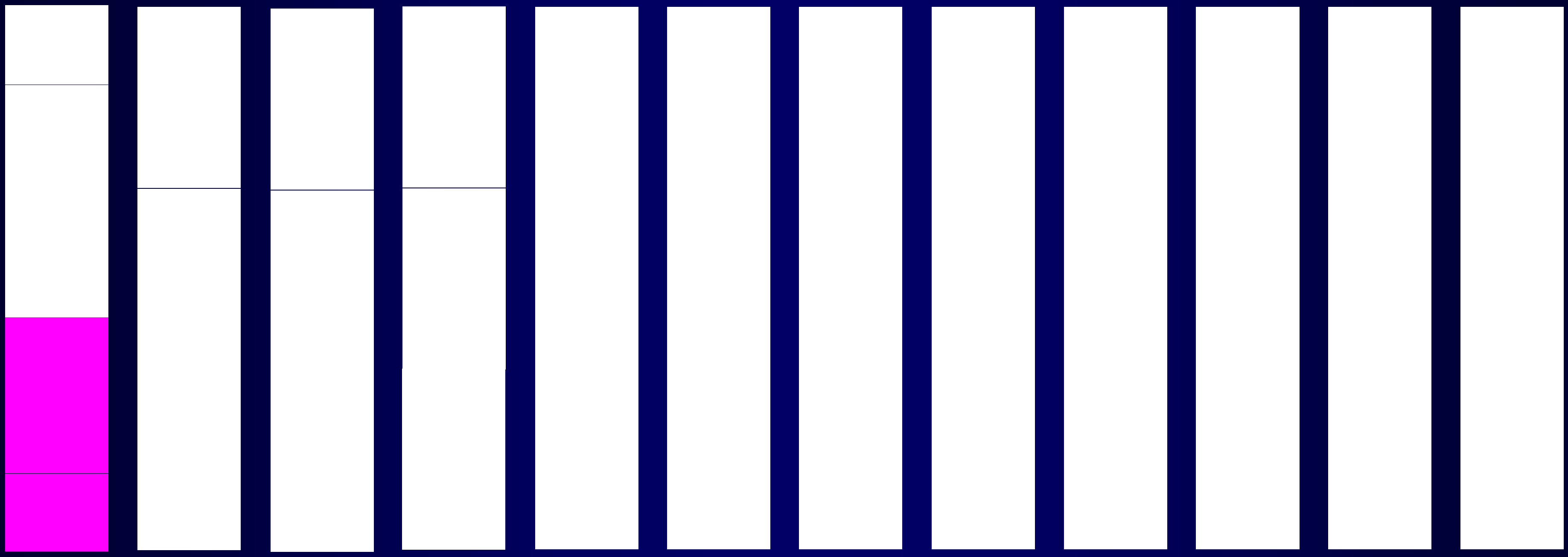
FF (exemple 2)

B = 420 et 61, 61



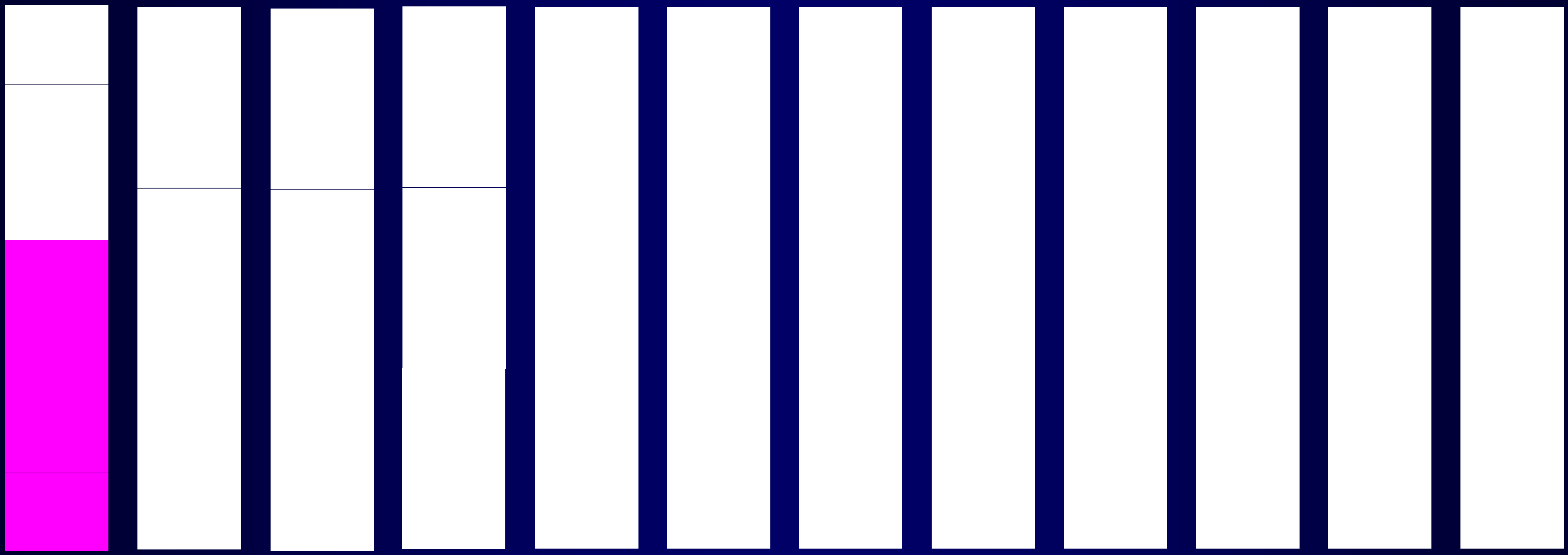
FF (exemple 2)

B = 420 et 61, 61, 61



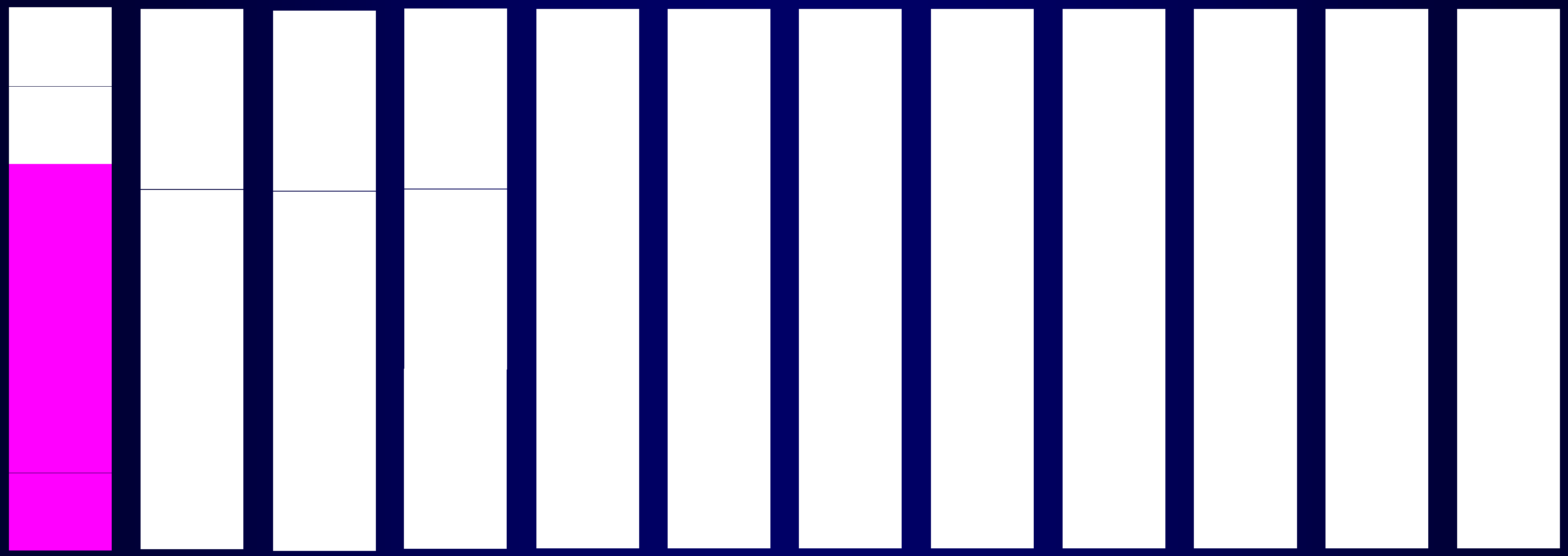
FF (exemple 2)

B = 420 et 61, 61, 61, 61



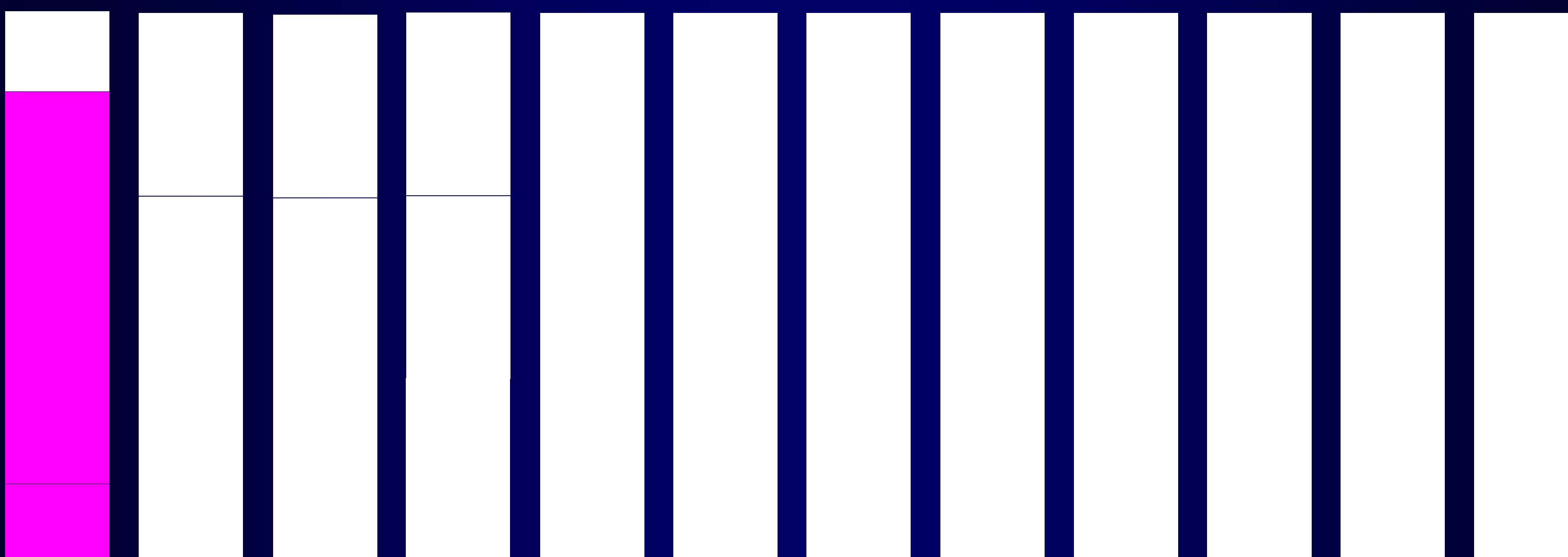
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61



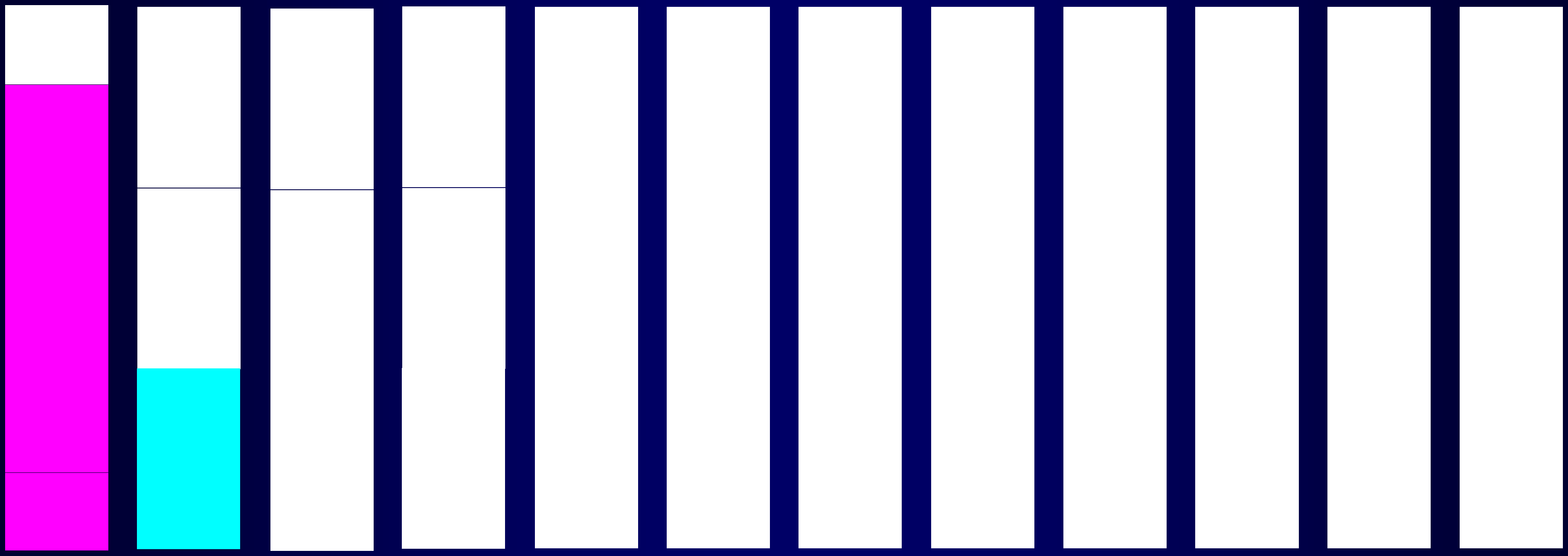
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61



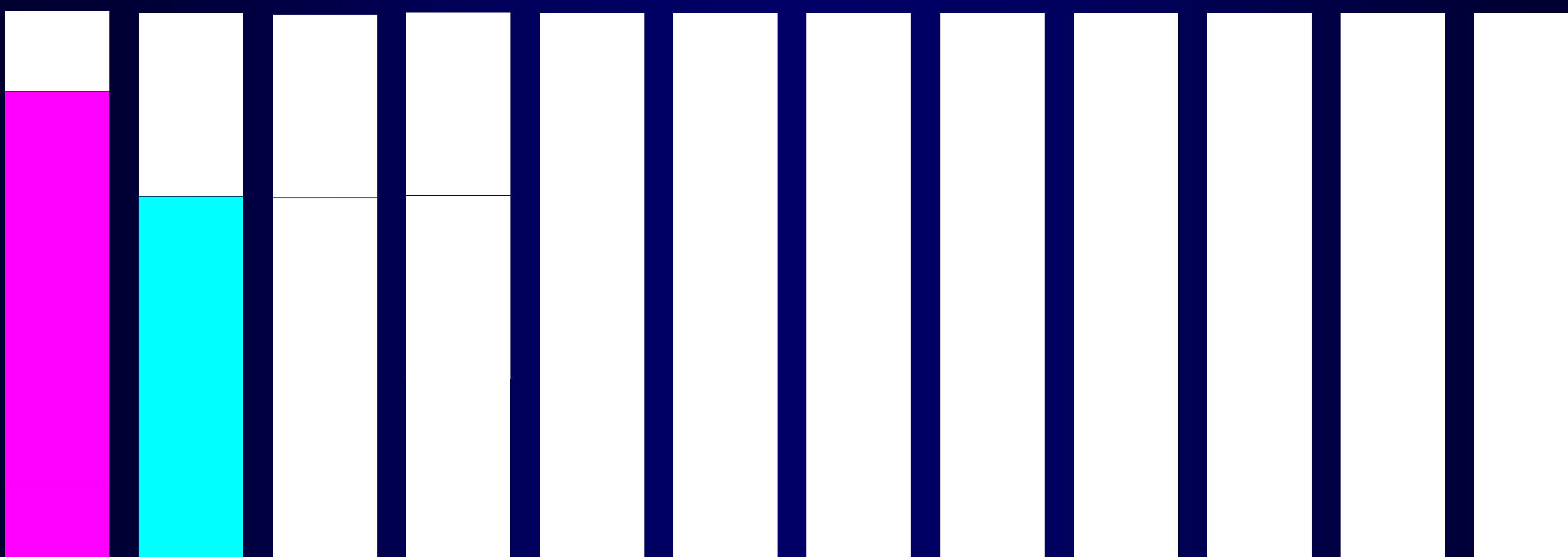
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141



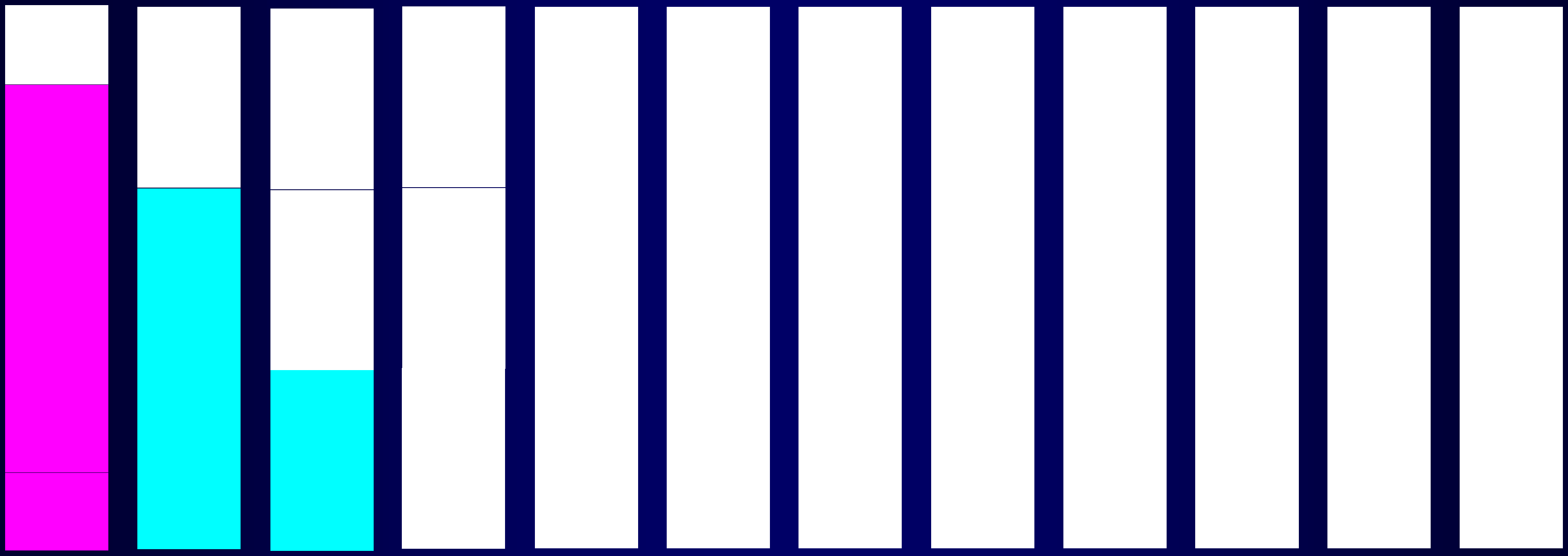
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141



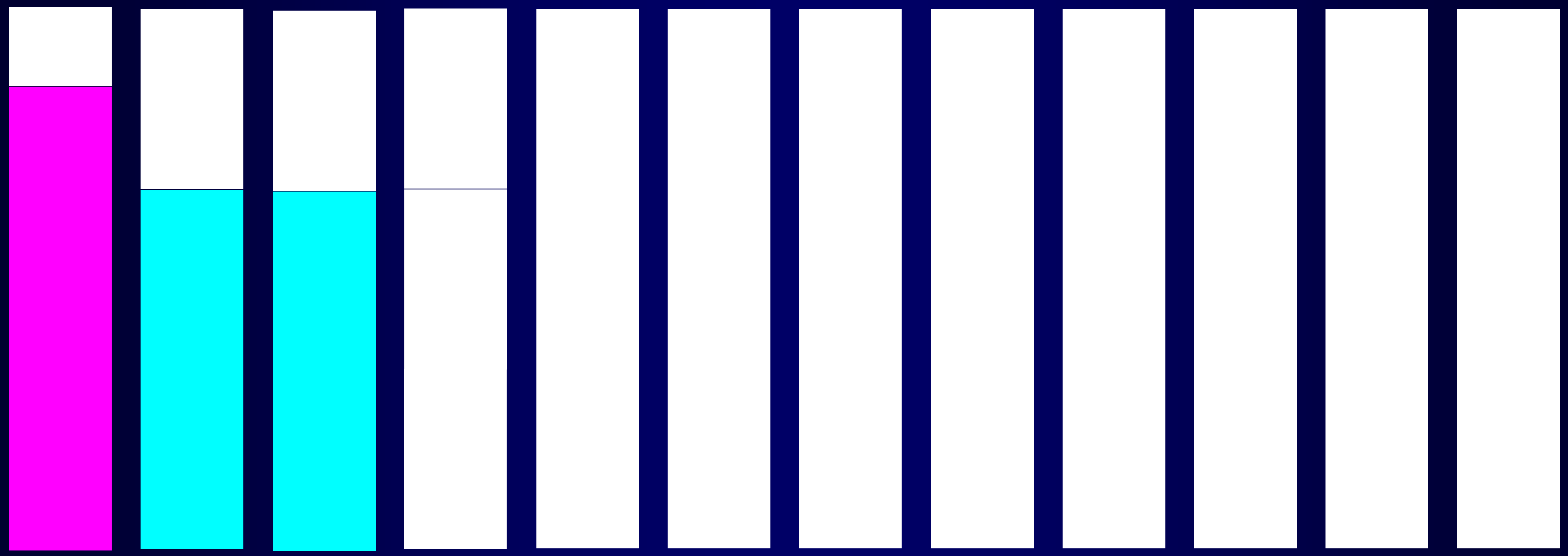
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141



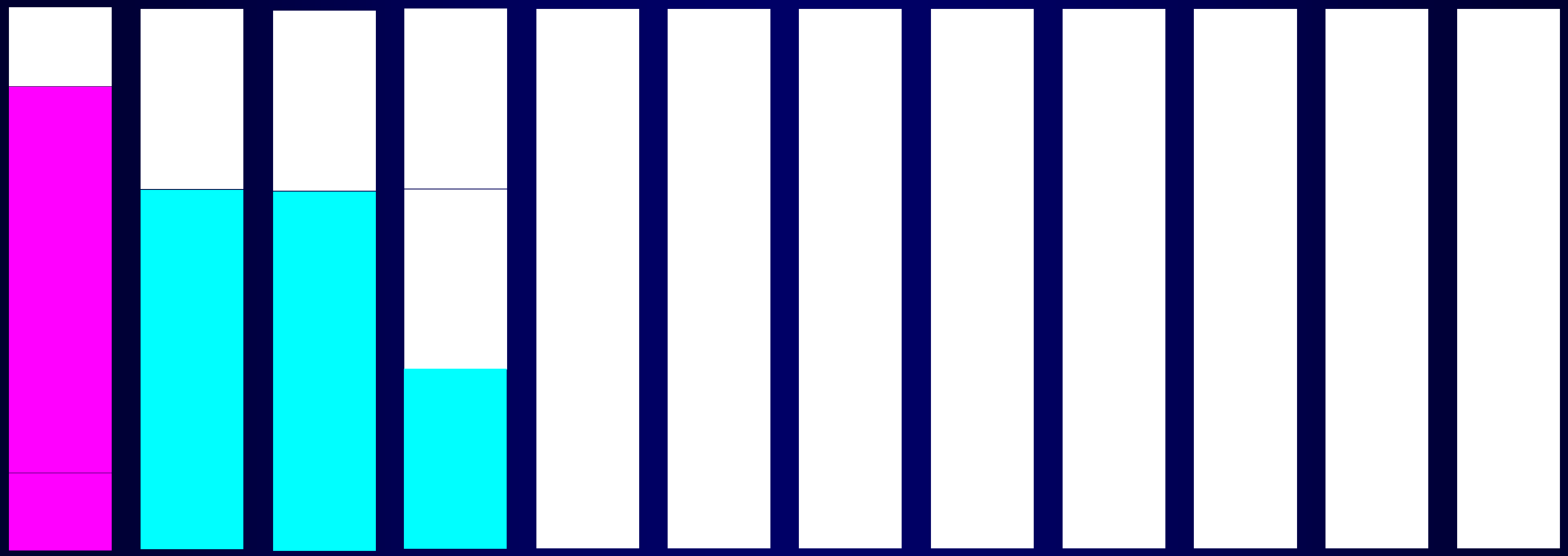
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141



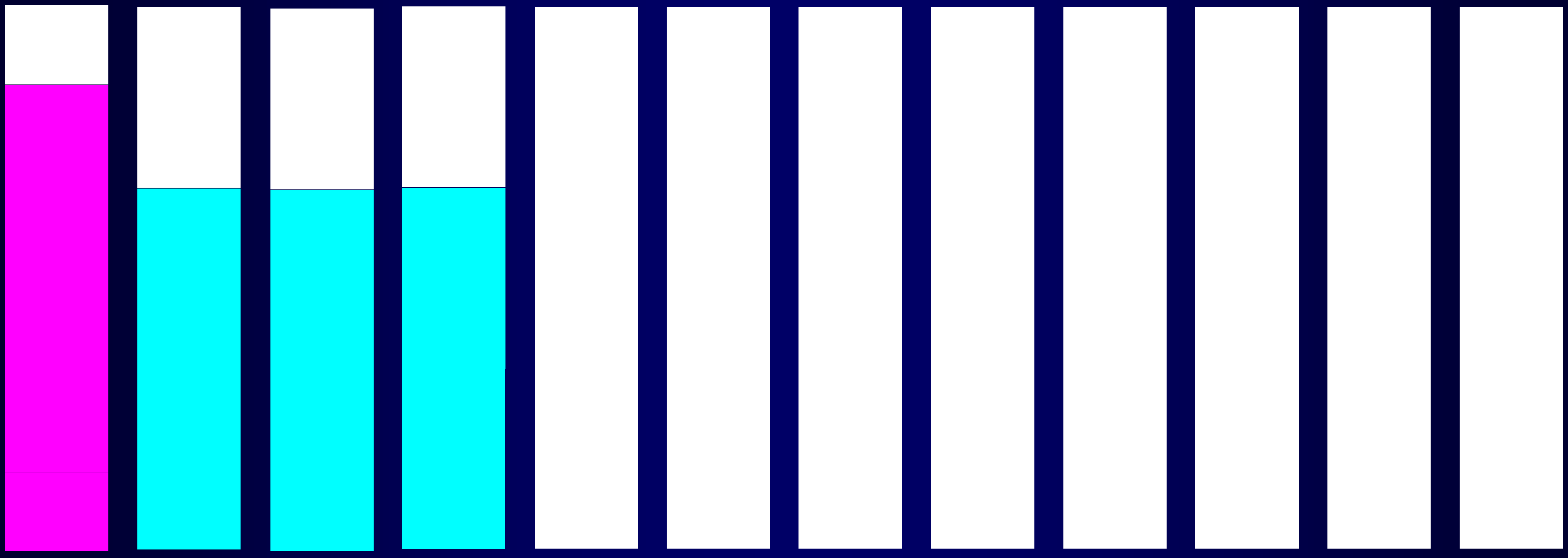
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141



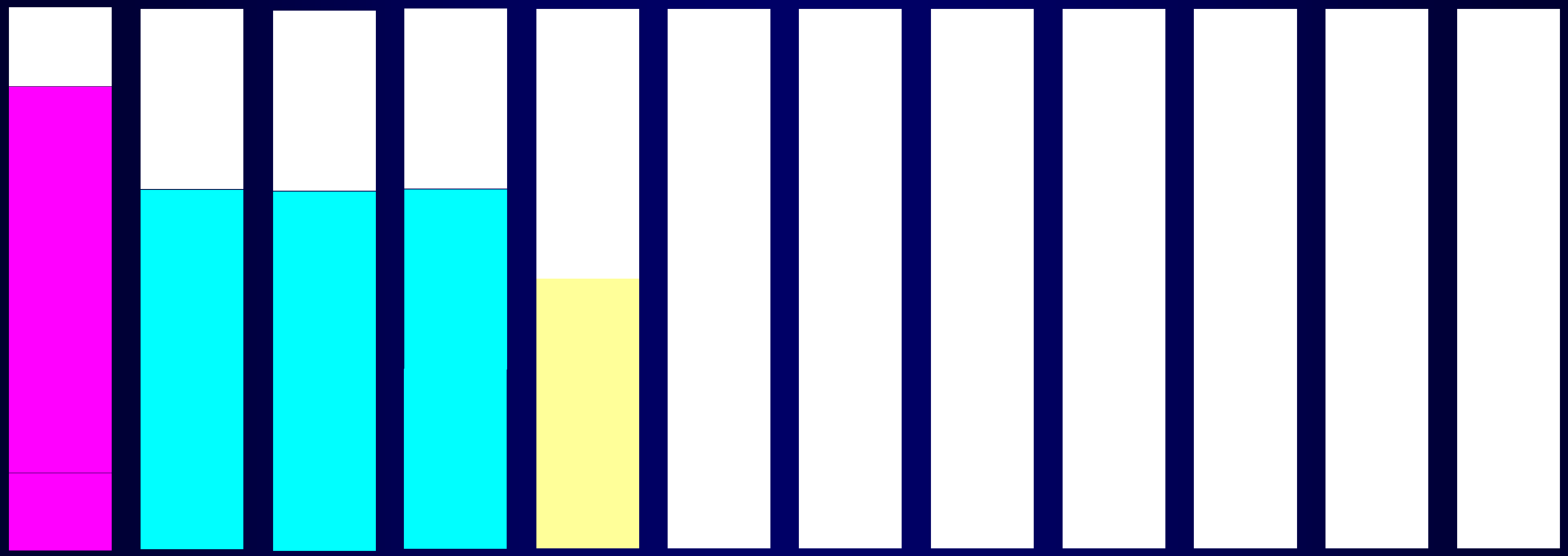
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141



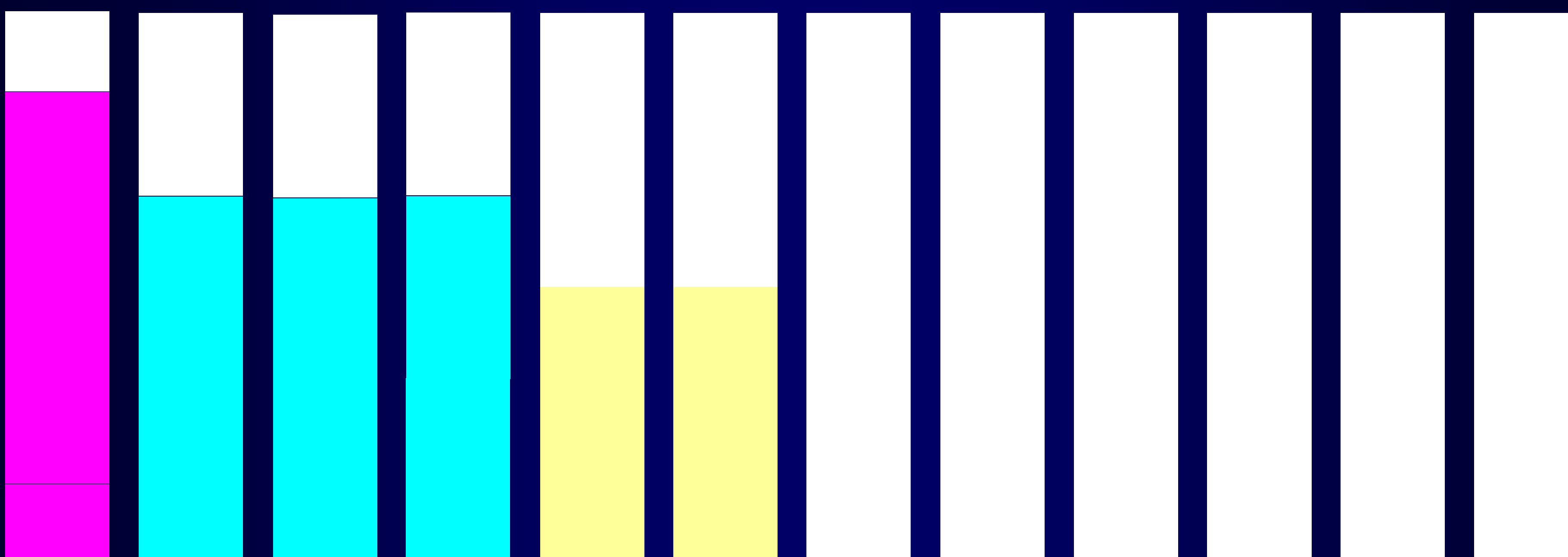
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141, 211



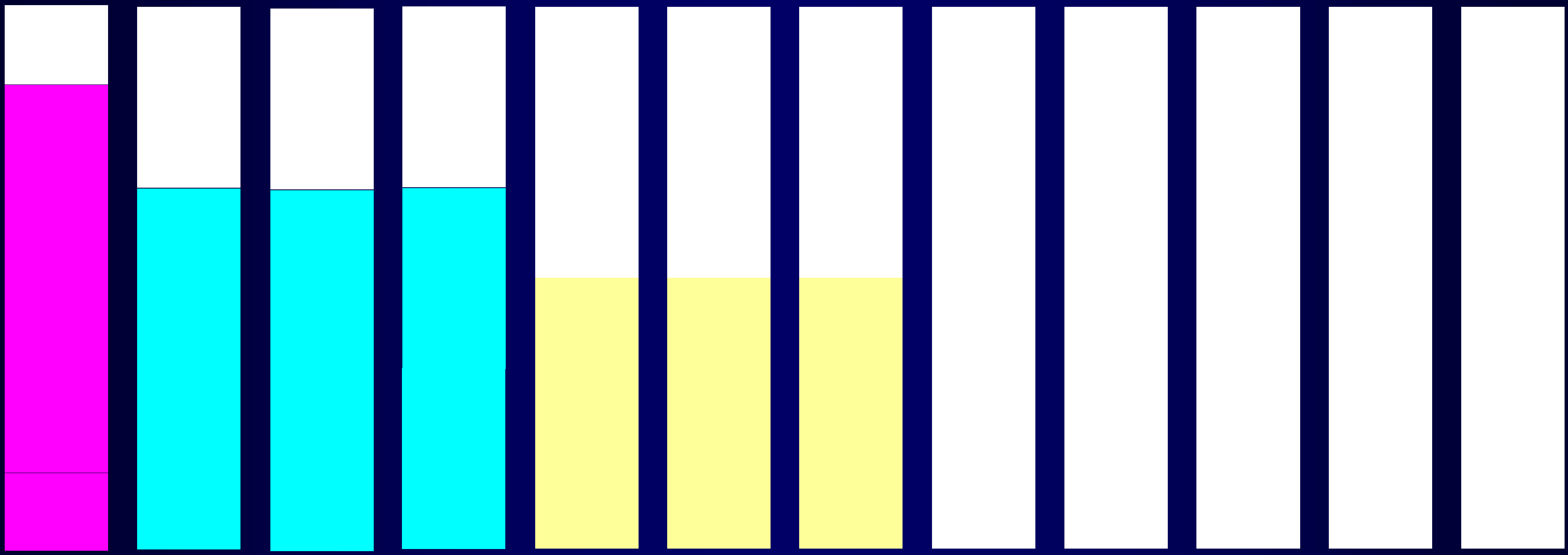
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141, 211, 211,



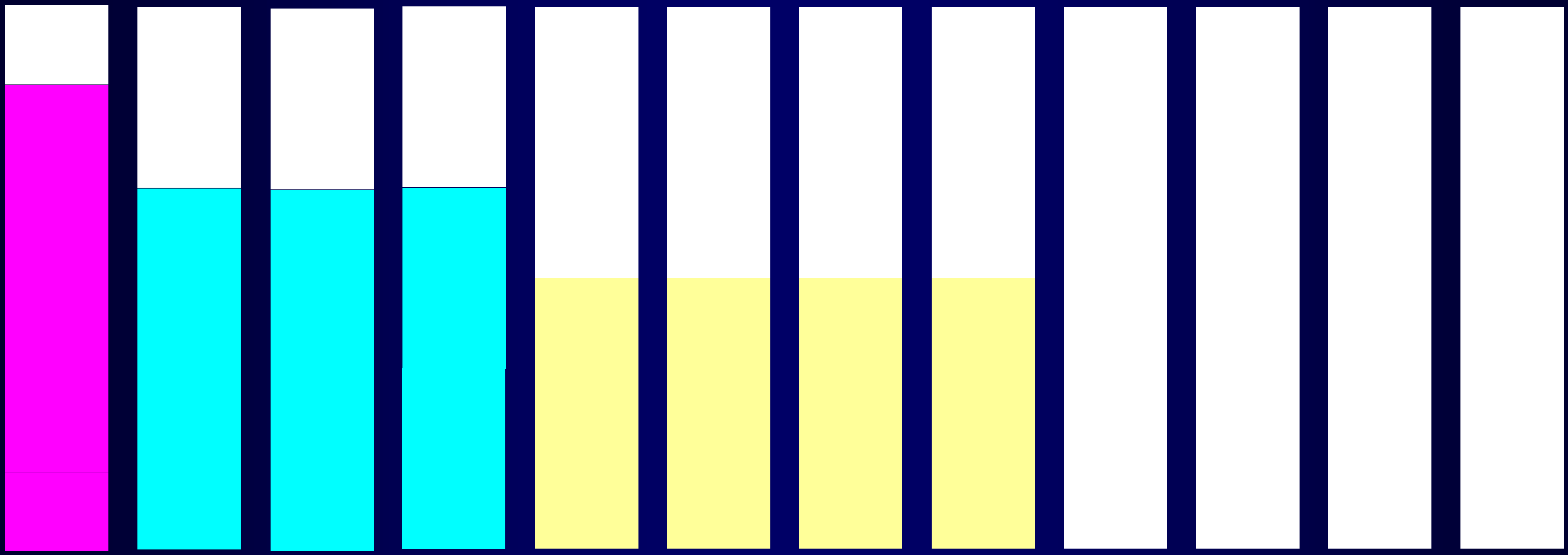
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141, 211, 211, 211



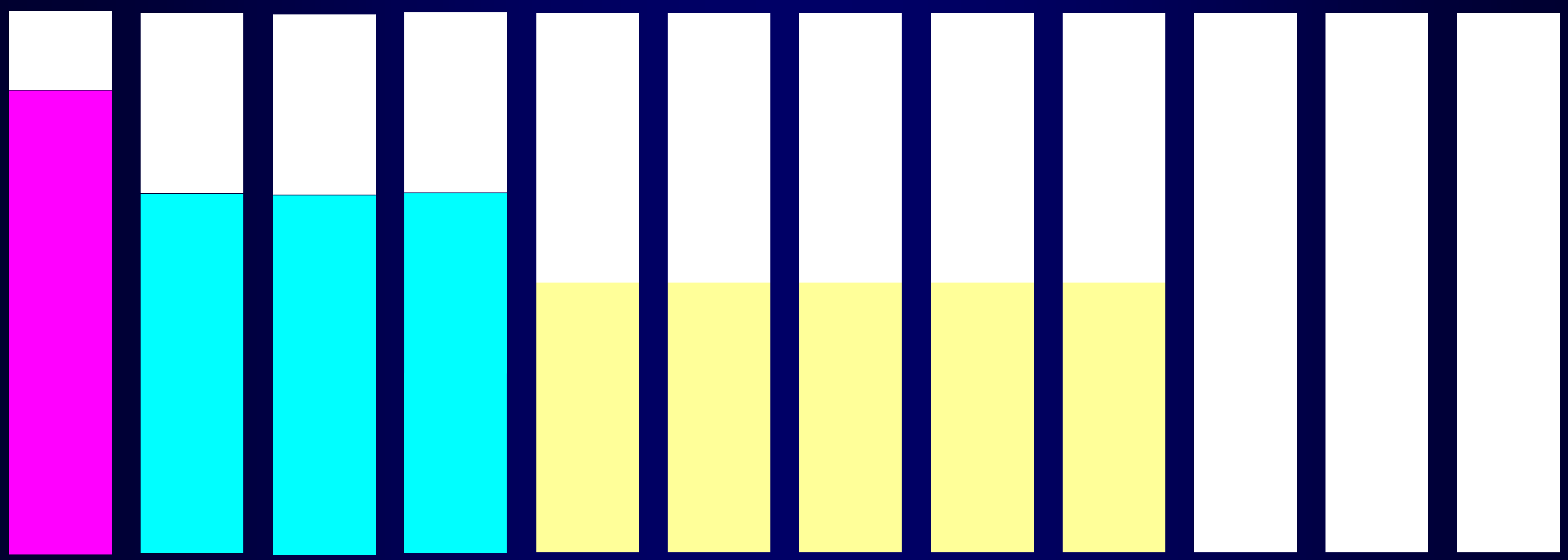
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141, 211, 211, 211, 211,



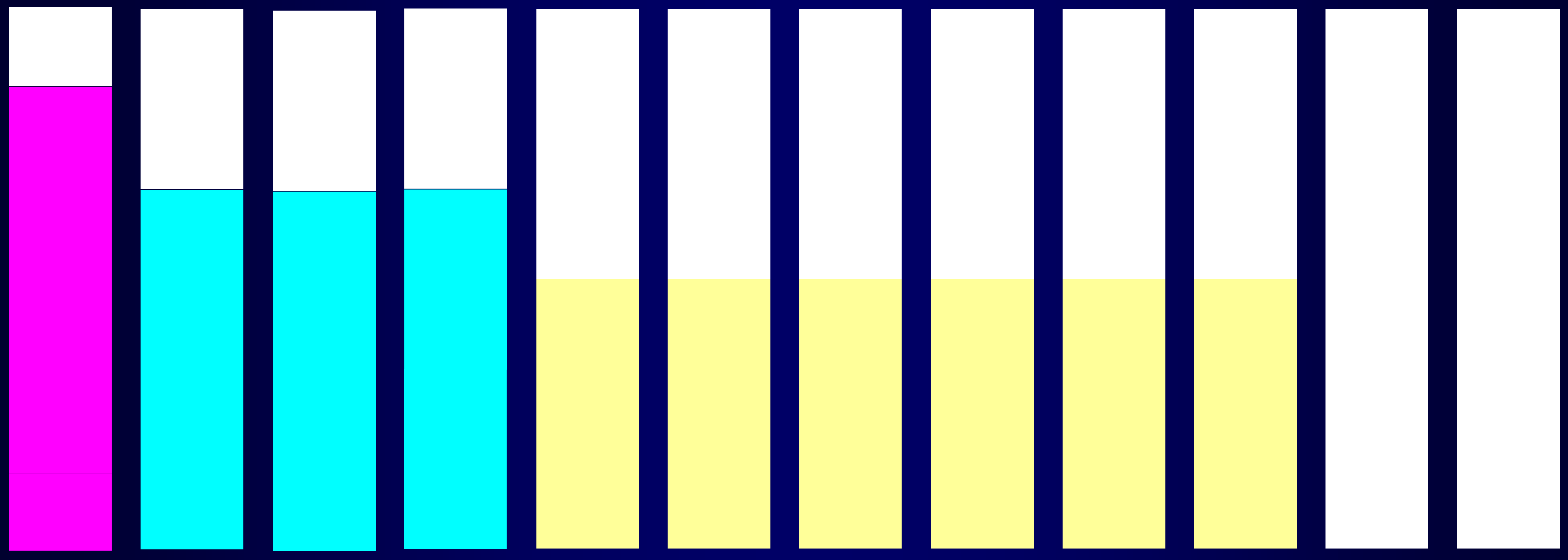
FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141, 211, 211, 211, 211, 211



FF (exemple 2)

B = 420 et 61, 61, 61, 61, 61, 61, 141, 141, 141, 141, 141, 141, 211, 211, 211, 211, 211, 211.



FF (suite)

On peut remarquer que pour les mêmes données, on a utilisé dans le premier exemple 6 bins et 10 dans le deuxième.

Le problème Bin Packing (2)

NOM : BIN PACKING

DONNEES : ensemble fini d'éléments U (taille $\in(0,1]$).

QUESTION : Quel est le plus petit k tel qu'on peut partitionner U en U_1, U_2, …, U_k sans que la somme des tailles des éléments des U_i dépasse 1 ?

Généralisation des exemples

$n = 18m$ et les valeurs sont :

$$u_i = \begin{cases} 1/7 + \varepsilon & 1 \leq i \leq 6m \\ 1/3 + \varepsilon & 6m < i \leq 12m \\ 1/2 + \varepsilon & 12m < i \leq 18m \end{cases}$$

Solution obtenue par FF :

- m bins contenant 6 objets de taille $1/7 + \varepsilon$
- $3m$ bins contenant 2 objets de taille $1/3 + \varepsilon$
- $6m$ bins contenant 1 objet de taille $1/2 + \varepsilon$

Solution obtenue par FF :

- *m* bins contenant 6 objets de taille $1/7 + \varepsilon$
- $3m$ bins contenant 2 objets de taille $1/3 + \varepsilon$
- $6m$ bins contenant 1 objet de taille $1/2 + \varepsilon$

Donc FF(D) = $10m$

Solution optimale :

- $6m$ bins contenant 3 objets (un de taille $1/7 + \varepsilon$, un de taille $1/3 + \varepsilon$ et un de taille $1/2 + \varepsilon$)

Donc OPT(D) = $6m$

FF (suite)

Propriété : Pour tout D

$$FF(D) < 2OPT(D)$$

Preuve :

On peut remarquer que la somme du contenu du bin 1 et celui du bin 2 est > 1

Et cela est vrai pour bin *i* et bin *i*+1 !

suite

Ainsi nous avons

$b_1 + b_2 > 1$

$b_2 + b_3 > 1$

…

$b_{k\text{-}1} + b_k > 1$

$b_1 + b_k > 1$

donc

$\mathrm{FF(D)} = \mathrm{k} < 2\Sigma_{i=1..n}\, u_i$

Par ailleurs,

$\Sigma_{i=1..n}\, u_i \leq \mathrm{OPT(D)}$

d'où

$\mathrm{FF(D)} < 2\mathrm{OPT(D)}$

CQFD

meilleures bornes ?

Théorème :

Pour toute donnée D

$$FF(D) < 17OPT(D)/10$$

Il existe des données arbitrairement grandes D, telles que

$$FF(D) > 17\big(OPT(D)-1\big)/10$$

Peut-on faire beaucoup mieux ?

NON

Un résultat négatif

Théorème :

Pour tout ε>0, il n'existe pas d'approximation de ratio 3/2-ε pour le problème de BIN PACKING, sauf si P=NP.

On prouve :

Pour tout ε>0, le problème de l'approximation de ratio 3/2-ε pour le problème de BIN PACKING, est NP-difficile.

la preuve

Preuve :

On réduit PARTITION

Pour ce faire on prend comme taille des bin, 1/2S

Ainsi OPT(D) sera 2, et tout approximation de ratio 3/2-ε comprendra <u>moins</u> de 3 bins (donc 2) ce qui revient à 2 bins.

CQFD

L'heuristique FFD

FFD = First-Fit-Decreasing

- **on trie les objets en ordre décroissant**
- **on applique FF**

Complexité : $O(n \log n + nk)$

Bornes pour FFD

Théorème :

Pour toute donnée D

$$FFD(D) < 11OPT(D)/9 + 4$$

Il existe des données arbitrairement grandes D, telles que

$$FF(D) > 11OPT(D)/9$$

Schéma d'approximation

Pour toute valeur $\varepsilon > 0$ elle propose une approximation avec erreur rélative ε.

PTAS (Polynomial time approximation scheme)

Schéma polynomial d'approximation : pour tout $\varepsilon > 0$ le schéma est polynomiale en n (taille du problème).

Schéma d'approximation

FPTAS

(Fully polynomial time approximation scheme)

Schéma polynomial d'approximation : si le schéma est polynomiale en n (taille du problème) et 1/ε.

APTAS

Asymptotic polynomial-time approximation scheme

- ***k* donné. Pour tout $\varepsilon > 0$ il existe un algorithme A_ε tel que**

  $$A_\varepsilon(D) \leq (1+\varepsilon)\ OPT(D) + k.$$

  L'algorithme A_ε est polynomial en $|D|$.

  pour le cas d'un pb. de maximisation :

  $$A_\varepsilon(D) \geq (1-\varepsilon)\ OPT(D) - k$$

Un résultat pour le BIN PACKING

Théorème :

Pour tout ε, $0 \leq \varepsilon \leq ½$ il existe un algorithme polynomiale A_ε qui trouve une solution utilisant au plus $(1+\varepsilon)$ OPT(D) + 1 bins.

Classes de complexité (1)

APX

Approximables

Problèmes qui admettent algorithmes d'approximation en temps polynomial avec un ratio d'approximation borné par un constant.

Autrement dit, les problèmes de cette classe admettent des algorithmes efficaces qui trouve une solution optimale à un constant multiplicative près.

Classes de complexité (2)

EPTAS

Efficient polynomial-time approximation scheme

Problèmes tels que si on a une donnée de taille n, on peut trouver une solution bornée par 1+ε fois l'optimum en temps f(ε)p(n), où p est un polynôme et f une fonction quelconque.

Classes de complexité (3)

PTAS

Polynomial-time approximation scheme

Problèmes admettant un schema d'approximation t.q. pour tout $\varepsilon>0$, il existe un algorithme polynomial donnant une solution d'au plus $1+\varepsilon$ fois l'optimale.

Cependant l'exposant du polynôme peut fortement dépendre de ε.

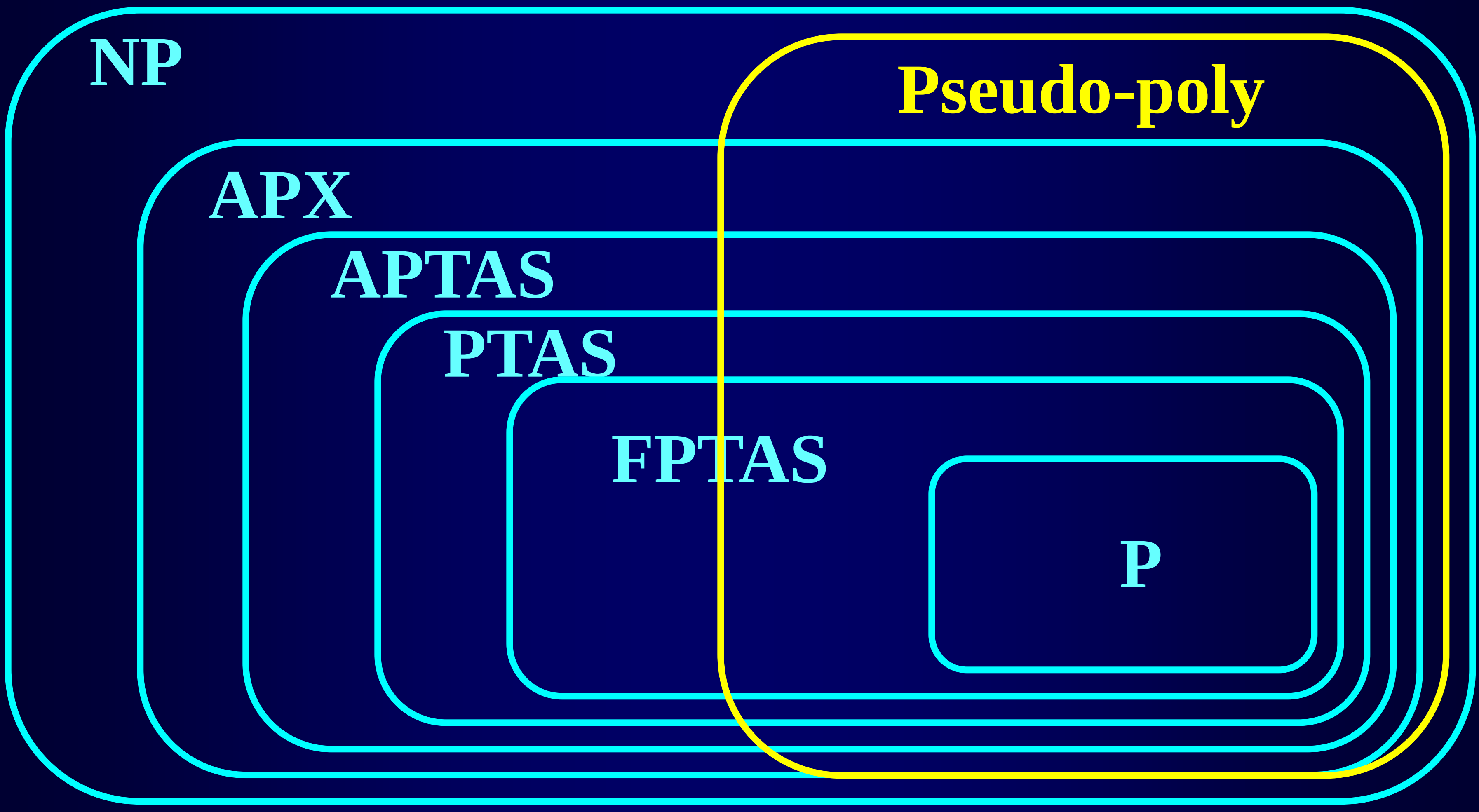
Classes de complexité (4)

FPTAS

Fully polynomial-time approximation scheme

Problèmes admettant un schéma d'approximation t.q. pour tout $\varepsilon>0$, il existe un algorithme don't le résultat est au plus $1+\varepsilon$ fois l'optimale et qui de plus ont une complexité polynomial en n et $1/\varepsilon$.

La hiérarchie des classes de complexité



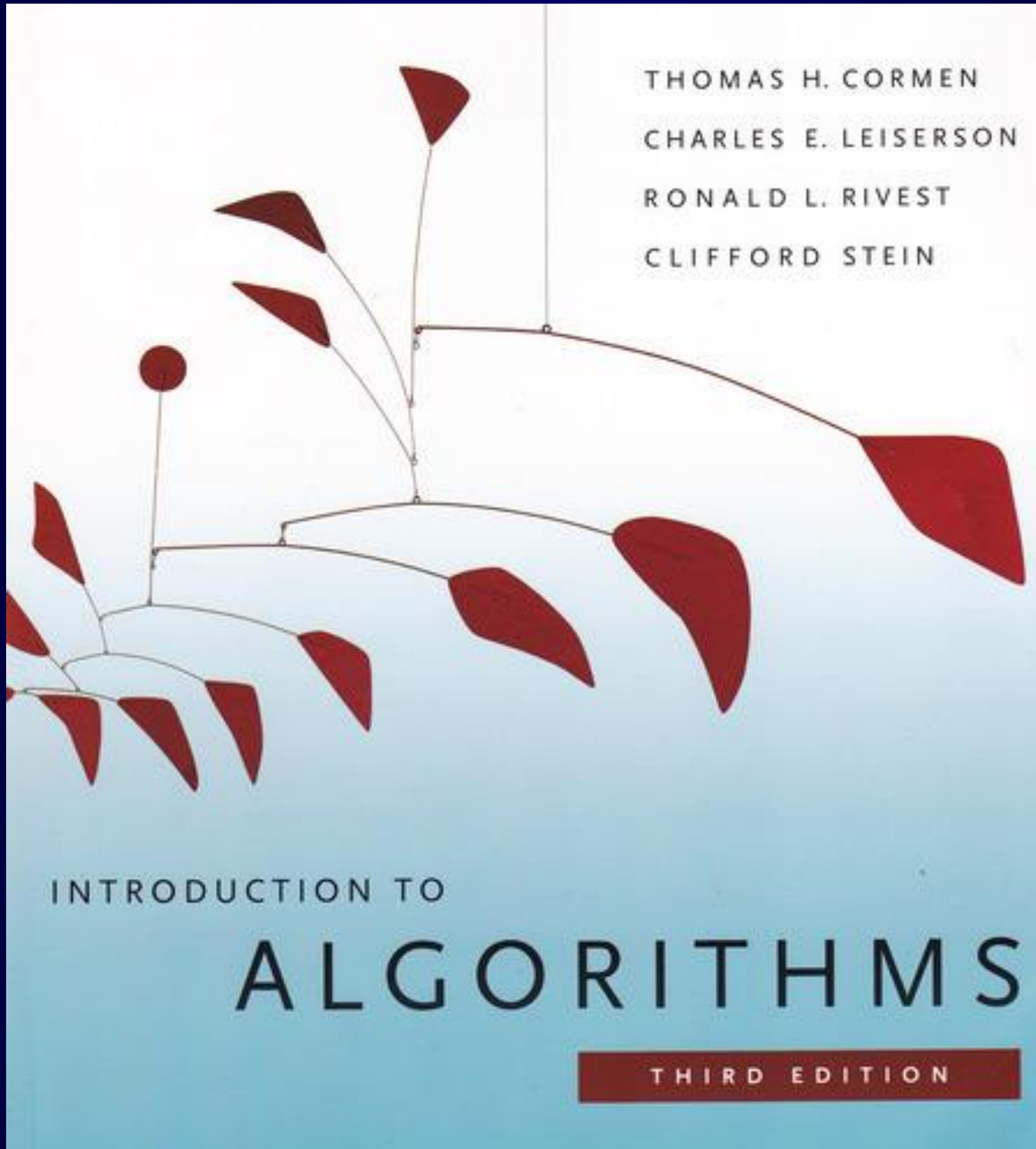
Le LIVRE

Pour toute la partie Algorithmique :

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest : Introduction to algorithms, MIT Press, 1990.

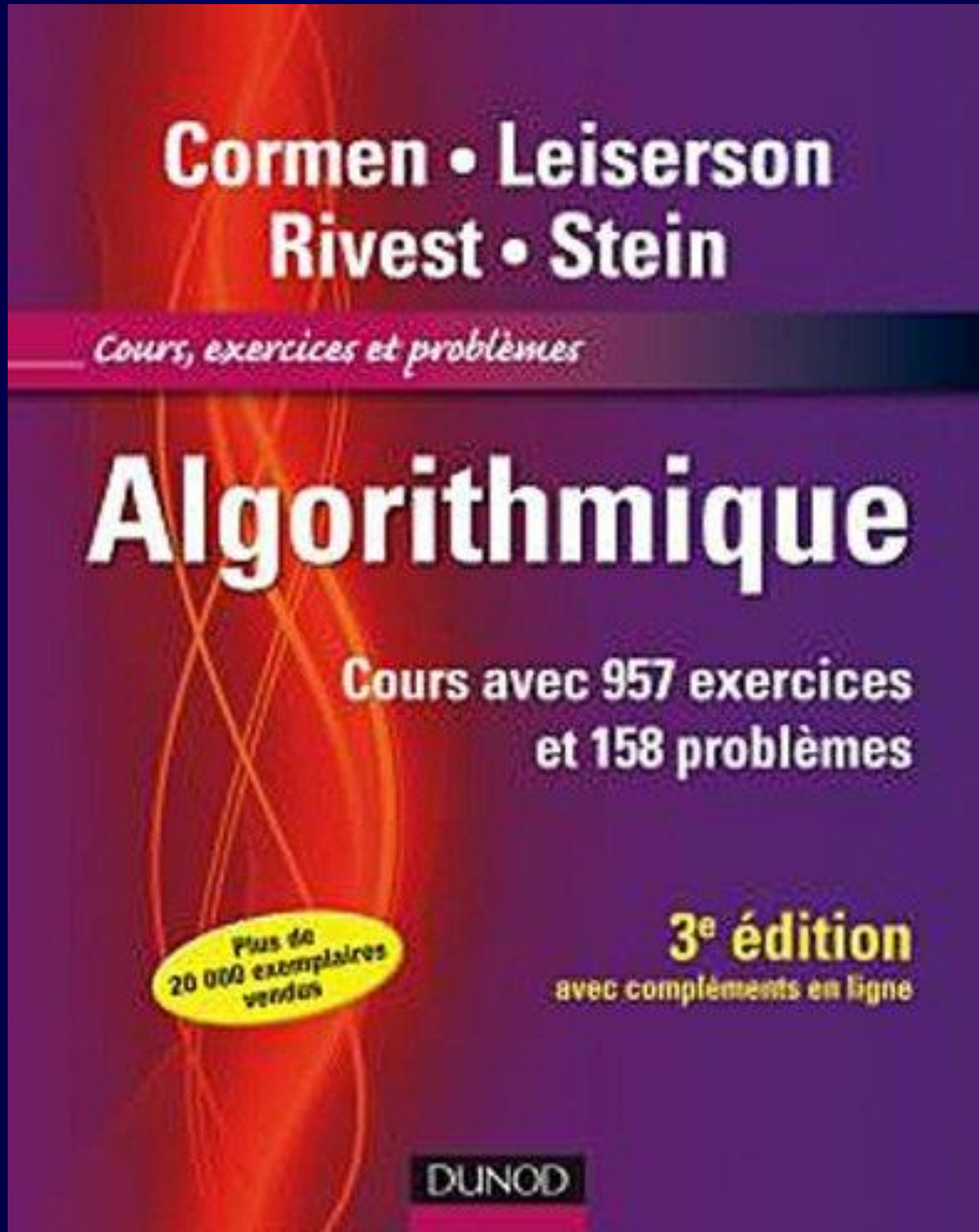
La dernière edition :

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein : Introduction to algorithms, MIT Press, 2009.


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION

VF

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein : Introduction à l'algorithmique, *troisième édition*, Dunod, juin 2010.


Cormen • Leiserson
Rivest • Stein
Cours, exercices et problèmes
Algorithmique
Cours avec 957 exercices
et 158 problèmes
Plus de
20 000 exemplaires
vendus
3e édition
avec compléments en ligne
DUNOD

Approximation de SSP

Le problème de décision : étant donné l'ensemble de nombres naturels $S=\{x_1, x_2, \ldots, x_n\}$ et un nombre naturel t, on pose la question s'il existe un sous-ensemble de S, dont la sommme des éléments soit t.

Le problème d'optimisation : étant donné l'ensemble de nombres naturels $S=\{x_1, x_2, \ldots, x_n\}$ et un nombre naturel t, on cherche le plus grand $t' \leq t$, tel qu'il existe un sous-ensemble de S, dont la sommme des éléments soit t'.

Notation

- on notera "L + *x*", la liste triée obtenue par addition de *x* à chaque élément de la liste triée L

- on notera "merge(L,L')" la liste triée et sans doublons obtenue à partir des listes triées L et L'

- on notera "supr(L,*k*)" la liste triée obtenue à partir de la liste triée L par suppression des éléments supérieurs à *k*

Algo exponentiel pour SSP

$n \leftarrow |S|$

$L_0 \leftarrow$ <0>

pour $i = 1$ **à** n **faire**

  $L_i \leftarrow \text{merge}(L_{i\text{-}1}, L_{i\text{-}1} + x_i)$

  $L_i \leftarrow \text{supr}(L_i, t)$

return(maximum de L_n)

Exemple

$S=\{1, 4, 7, 10\}$

$t=20$

$L_0=\{0\}$

$L_1=\{0, 1\}$

$L_2=\{0, 1, 4, 5\}$

$L_3=\{0, 1, 4, 5, 7, 8, 11, 12\}$

$L_4=\{0, 1, 4, 5, 7, 8, 10, 11, 12, 14, 15, 17, 18\}$

Résultat : 18

Le même exemple encore

S={10, 7, 4, 1}

t=20

L_0={0}

L_1={0, 10}

L_2={0, 7, 10, 17}

L_3={0, 4, 7, 10, 11, 14, 17}

L_4={0, 1, 4, 5, 7, 8, 10, 11, 12, 14, 15, 17, 18}

Résultat : 18

Exemple de l'exponentialité

$S=\{1, 2, 4, 8, 16, \ldots\}=\{2^{i-1} \mid i=1, \ldots, n\}$

t assez grand $\Theta(2^n)$

$L_1=\{0, 1\}$

$L_2=\{0, 1, 2, 3\}$

…

$L_i = \{0, 1, \ldots, 2^i-1\}$

et comme la taille des L_i est exponentielle, l'algorithme l'est.

Un FPTAS pour SSP

L'idée est d'utiliser un procédée d'*élagage*.

Soit $0 < \delta < 1$. Un *élagage* de la liste L par δ, consiste en la suppression de maximum d'éléments de L, de manière à obtenir une liste L', qui pour tout élément supprimé y contient un élément proche, c.a.d. $z \leq y$, tel que $(y-z)/y \leq \delta$

c.a.d. $(1-\delta)y \leq z \leq y$

On peut considérer que z représente y ; ainsi chaque valeur est soit présente soit représentée par une valeur assez proche.

Exemple d'élagage

$L = \{10, 11, 12, 15, 20, 21, 22, 23, 24, 29\}$

$\delta = 1/10$

$L = \{10, 11, 12, 15, 20, 21, 22, 23, 24, 29\}$

11 représenté par 10

21 et 22 representés par 20

24 representé par 23

$L' = \{10, 12, 15, 20, 23, 29\}$

Algorithme d'élagage en O(m)

ELAGAGE(L,δ)

$m \leftarrow |L|$

L' $\leftarrow \{ y_1 \}$

last $\leftarrow y_1$

pour $i = 2$ à m faire

si last $< (1-\delta)\, y_i$

alors

L' $\leftarrow$ L' $\leftarrow \{ y_i \}$

last $\leftarrow y_i$

reurn(L')

Le schéma d'approximation

Approx SSP(S, t, ε)

$n \leftarrow |S|$

$L_0 \leftarrow \{ 0 \}$

pour $i = 1$ à n faire

$L_i \leftarrow$ merge(L_{i-1}, $L_{i-1} + x_i$)

$L_i \leftarrow$ elagage(L_i, ε/n)

$L_i \leftarrow$ supr(L_i, t)

return(maximum de L_n)

Exemple

$S = \{24, 28, 36, 32\}$

$t = 66$

$\varepsilon = 0{,}3 \Rightarrow$

$\delta = \varepsilon/4 = 0{,}075$

$L_0 = \{ 0 \}$

M $L_1 = \{ 0, 24 \}$

E $L_1 = \{ 0, 24 \}$

S $L_1 = \{ 0, 24 \}$

M $L_2 = \{ 0, 24, 28, 52 \}$

E $L_2 = \{ 0, 24, 28, 52 \}$

S $L_2 = \{ 0, 24, 28, 52 \}$

M $L_3 = \{ 0, 24, 28, 36, 52, 60, 64, 88 \}$

E $L_3 = \{ 0, 24, 28, 36, 52, 60, 88 \}$

S $L_3 = \{ 0, 24, 28, 36, 52, 60 \}$

M $L_4 = \{ 0, 24, 28, 32, 36, 52, 56, 60, 68, 84, 88, 92 \}$

E $L_4 = \{ 0, 24, 28, 32, 36, 52, 60, 68, 84, 88, 92 \}$

S $L_4 = \{ 0, 24, 28, 32, 36, 52, 60 \}$

Résultat : 60

(Le résultat optimal est 64)

Preuve de l'algorithme

- Dans toutes les opérations on manipule des sommes d'éléments (et éventuellement on en supprime) donc le résultat est la somme de certains éléments de S.

- Reste à montrer que

  $$\text{Résultat} \geq (1 - \varepsilon)\ \text{Optimum}$$

- Il faut montrer que la complexité de l'algorithme est polynomiale.

L'approximation

Un élément élagué, y est représenté par un z tel que :

$$(1 - \varepsilon/n)\, y \leq z \leq y$$

Lors d'une phase ultérieure, on obtient

$$(1 - \varepsilon/n)^i\, y \leq z \leq y$$

Ainsi, pour chaque élément dans L_i, y est soit présent soit représenté par un z tel que

$$(1 - \varepsilon/n)^i\, y \leq z \leq y$$

Donc le résultat vérifie

$$(1 - \varepsilon/n)^n\, Opt \leq Rés \leq Opt$$

Approximation (suite)

Mais comme la fonction

$$f(n) = (1 - \varepsilon/n)^n$$

est croissante en *n* (dérivé positive !), on a

$$(1 - \varepsilon) < (1 - \varepsilon/n)^n$$

Donc

$$(1 - \varepsilon)\, Opt \leq Rés \leq Opt$$

La complexité

Après l'élagage, si z et z' sont des éléments consécutifs dans L_i, alors on ne peut pas avoir

$$z' (1 - \varepsilon/n) < z < z'$$

donc on a

$$z' (1 - \varepsilon/n) > z$$

c.a.d.

$$z'/z > 1/(1 - \varepsilon/n)$$

Ainsi le nombre d'éléments de L_i est au plus

$$\log_{1/(1 - \varepsilon/n)} t = \ln t / (-\ln(1 - \varepsilon/n)) \le n \ln t / \varepsilon$$

Complexité (suite)

Le nombre d'éléments de Li est donc polynomial en n, $\ln t$ et $1/\varepsilon$.

La première valeur est bornée par la taille des données, la seconde est la taille de la représentation de t et la troisième ($1/\varepsilon$) est celle demandée par un FPTAS.

CQFD

Une dernière remarque concernant la NP-complétude

Problèmes de reconnaissance vs optimisation

Si on sait reconnaître en temps polynomial, alors en un nombre O(log *Rés*) de reconnaissances on peut trouver l'optimum.

Si on sait trouver l'optimum alors on sait répondre au problème de reconnaissance.

Remarque : **pour les problèmes d'optimisation on parle de NP-difficulté seulement.**