DOCUMENTATION

dnb

Par
Grégory SCHAAF

SOMMAIRE

Variables

Définition des variables utilisées

nCar: Nombre de caractères existants (type entier)

l: Ensemble des éléments ordonnés générés (type liste)

long: Longueur de la liste des éléments ordonnés (type entier)

m: Modèle (type liste)

num: Numéro d'ordre d'un résultat (type entier)

Une liste, une ligne ou une colonne commence par la position 0.

1

Suite

Introduction

La classe Suite permet de générer la suite logique d'un ensemble ordonné de valeurs contenues dans ***l.***

Structure

Suite.<algorithme>(***l***, ***nCar***)

1 résultat de type liste

Si ***l*** n'a pas encore été générée il est possible d'indiquer une liste remplie avec la valeur ***None*** lors de l'éxécution de la fonction.

1.1

Suite.arrangement

Fonctionnement

l est parcourue en partant du dernier élément jusqu'à trouver un élément qui n'est pas égal à la dernière valeur possible (***nCar*** - 1), l'augmente de 1, réinitialise à 0 les éléments parcourus et s'arrête.

Démonstration

Suite.arrangement({0, 0, 10, 1}, 11)

- l_3 est < à ***nCar*** – 1 : l_3 += 1

Résultat : {0, 0, 10, 2}

Suite.arrangement({0, 0, 10, 10}, 11)

- l_3 == ***nCar*** – 1 : l_3 = 0
- l_2 == ***nCar*** – 1 : l_2 = 0
- l_1 < ***nCar*** – 1 : l_1 += 1

Résultat : {0, 1, 0, 0}

Suite.arrangement({None, None, None, None}, 11)

Résultat : {0, 0, 0, 0}

nCar peut être inférieur, égal ou supérieur à ***long***

1.2

Suite.permutation

Fonctionnement

l est parcourue en partant du dernier élément pour vérifier si le caractère actuel peut être modifié par un caractère qui le succède. Si c'est le cas il augmente le caractère suivant de 1 et remplace dans l'ordre croissant chaque caractère parcouru par le plus petit n'étant pas encore inclu.

Démonstration

Suite.permutation({0, 3, 1, 2}, 4)

- l_3 : en augmentant de 1, la valeur 3 est déjà présente dans l'une des positions précédentes de ***l***, analyse de la position suivante.
- l_2 : en augmentant de 1, la valeur 2 n'est pas présente dans l'une des positions précédentes de ***l***.
- **Résultat** : {0, 3, 2, 1}

Suite.permutation({1, 2, 3, 0}, 5)

- l_3 : les valeurs 1, 2 et 3 sont déjà présentes dans les positions précédentes sauf le caractère 4.
- **Résultat** : {1, 2, 3, 4}

Suite.permutation({None, None, None, None}, 5)

- **Résultat** : {0, 1, 2, 3}

nCar doit être égal ou supérieur à ***long***.

1.3

Suite.Combinaison

Fonctionnement

- Si l'élément de la dernière position de ***l*** n'est pas égale à la dernière valeur possible (***nCar*** - 1) alors il est remplacé par le caractère logique suivant.
- Sinon, la liste est parcourue en commençant par la fin jusqu'à atteindre la position contenant une valeur obligatoirement inférieure, laquelle est alors augmentée de 1. Les positions déjà parcourues voient leur valeur mise à jour pour correspondre à cette nouvelle valeur augmentée.

Démonstration

Suite.Combinaison({0, 0, 8, 10}, 11)

- l_3 = 10, analyse de la position suivante
- l_2 < 10 : l_2 += 1, toutes les positions parcourues = l_2
- **Résultat** : {0, 0, 9, 9}

Suite.Combinaison({1, 1, 1, 1}, 11)

- l_3 < 10 : l_3 += 1
- **Résultat** : {1, 1, 1, 2}

Suite.Combinaison({None, None, None, None}, 11)

- **Résultat** : {0, 0, 0, 0}

nCar peut être inférieur, égal ou supérieur à ***long***.

1.4

Suite.combinaison

Fonctionnement

- Si la valeur de la dernière position de ***l*** n'est pas égale à la dernière valeur possible (***nCar*** - 1) alors elle est remplacée par la valeur logique suivante.
- Sinon la liste est parcourue en commencant par la fin jusqu'à la position contenant une valeur différente d'au moins de 2 (donc obligatoirement inférieure) qui est augmenté de 1 et les positions parcourues voient leur valeur augmenter de 1 par rapport au nombre de la position précédente.

Démonstration

Suite.combinaison({0, 7, 9, 10}, 11)

- l_3 == ***nCar*** - 1: analyse de la position suivante
- l_3 - l_2 < 2 : analyse de la position suivante
- l_2 - l_1 = 2 : l_2 += 1, l_2 = l_1 + 1 et l_3 = l_2 + 1.
- **Résultat** : {0, 8, 9, 10}

Suite.combinaison({1, 2, 8, 9}, 11)

- l_3 < ***nCar*** - 1 : l_3 += 1
- **Résultat** : {1, 2, 8, 10}

Suite.combinaison({None, None, None, None}, 11)

- **Résultat** : {0, 1, 2, 3}

nCar doit être supérieur à ***long***.

2

Modeliser

Introduction

La classe Modeliser permet de générer un modèle qui devra être utilisé pour convertir un numéro d'ordre en son ensemble de valeurs équivalente et inversement.

Structure

Modeliser.arrangement(***nCar***, ***long***)

1 résultat de type liste

Modeliser.permutation(***nCar***, ***long***)

1 résultat de type liste

Modeliser.Combinaison(***nCar***, ***long***)

1 résultat de type liste

Modeliser.combinaison(***nCar***, ***long***)

1 résultat de type liste contenant 2 éléments distincts de type liste

2.1

Modeliser.arrangement

Fonctionnement

- Un tableau de 1 ligne et ***long*** colonnes est considéré.
- La dernière colonne de la ligne contient toujours 1, peu importe ***long*** ou ***nCar*** et chaque colonne à partir de l'avant-dernière colonne répète ceci: La colonne actuelle doit contenir la valeur de la colonne précédente x ***nCar***.

Démonstration

Modeliser.arangement(6, 4)

- Un tableau de 1 ligne et ***long*** colonnes est créé.

- $V_3 = 1$
- $V_2 : V_3$ x ***nCar*** = 6
- $V_1 : V_2$ x ***nCar*** = 36
- $V_0 : V_1$ x ***nCar*** = 216

Résultat:

$V_{0,0}$	$V_{0,1}$	$V_{0,2}$	$V_{0,3}$

216	36	6	1

2.2

Modeliser.permutation

Fonctionnement

- Un tableau de 2 lignes et ***long*** colonnes est créé.
- A partir de la première position, la première ligne est complétée en commencant par la valeur ***nCar*** qui est déduite de 1 à chaque colonne, ce qui résulte d'une ligne commençant par ***nCar***, puis ***nCar*** - 1, ***nCar*** - 2, etc…
- La deuxième ligne est complétée en partant de la dernière colonne. La dernière colonne contiendra toujours 1 et les colonnes suivantes contiennent la valeur de la colonne précédente de la première ligne x la valeur de la colonne précédente de la deuxième ligne.

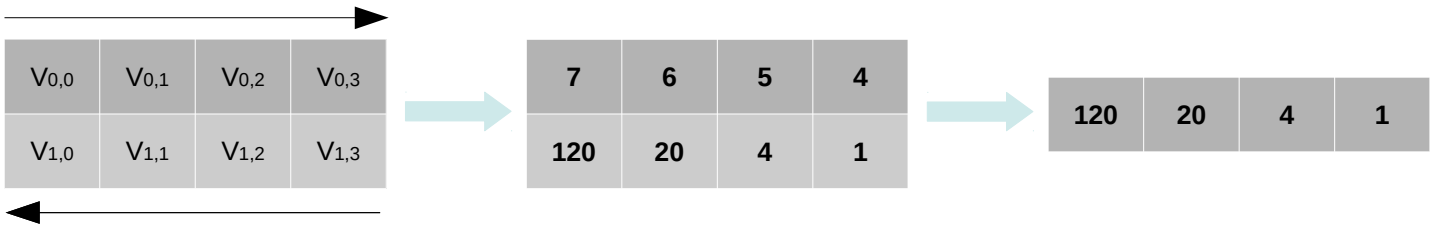
Démonstration

Modeliser.permutation(7, 4)

- Un tableau de 2 lignes et ***long*** colonne est créé.

- $V_{1,3} = 1$
- $V_{1,2} : V_{1,3} \times V_{0,3} = 4$
- $V_{1,1} : V_{1,2} \times V_{0,2} = 20$
- $V_{1,0} : V_{1,1} \times V_{0,1} = 120$

Résultat:



2.3

Modeliser.Combinaison

Fonctionnement

- Un tableau de ***long*** lignes et ***nCar*** colonnes est considéré.
- Toutes les colonnes de la première ligne contiendront toujours 1, peut importe les paramètres.
- Chaque colonne des lignes suivantes contiennent la somme des valeurs des colonnes de la ligne précédente en partant de la colonne actuelle.

Démonstration

Modeliser.Combinaison(5, 4)

- De $V_{0,0}$ à $V_{0,4}$: 1
- $V_{1,0}$: somme des valeurs de $V_{0,0}$ à $V_{0,4}$
- $V_{1,1}$: somme des valeurs de $V_{0,1}$ à $V_{0,4}$
- etc...

Résultat:

$V_{0,0}$	$V_{0,1}$	$V_{0,2}$	$V_{0,3}$	$V_{0,4}$
$V_{1,0}$	$V_{1,1}$	$V_{1,2}$	$V_{1,3}$	$V_{1,4}$
$V_{2,0}$	$V_{2,1}$	$V_{2,2}$	$V_{2,3}$	$V_{2,4}$
$V_{3,0}$	$V_{3,1}$	$V_{3,2}$	$V_{3,3}$	$V_{3,4}$

1	**1**	**1**	**1**	**1**
5	**4**	**3**	**2**	**1**
15	**10**	**6**	**3**	**1**
35	**20**	**10**	**4**	**1**

2.4

Modeliser.combinaison

Fonctionnement

- Un tableau de ((***nCar*** – ***long***) + 1) lignes et ***long*** colonnes est considéré.
  - La première colonne de chaque ligne contient un bloc contenant la position de la ligne – 1.
  - La colonne suivante contient un seul bloc contenant ((***nCar*** – ***long***) + 1) éléments commencant par l'élément du bloc de la colonne précédente + 1 et terminant par la suite croissante.
  - Le nombre de blocs de la colonne suivante est égal au nombre total d'éléments de tous les blocs de la colonne actuel.
  - Le nombre d'éléments dans un bloc est déterminé par le nombre d'éléments dans le bloc parent - la position de l'élément parent dans son bloc. Le premier élément d'un bloc est égal à l'élément parent du bloc + 1.

- Un deuxième tableau du même nombre de lignes, de colonnes, de blocs et d'éléments est considéré.
  - Celui-ci contient le numéro d'ordre de chaque résultat possible.
  - La dernière colonne de chaque ligne est une suite croissante allant de 0 à ***long***.
  - Pour les colonnes suivantes chaque élément contient un intervalle commencant par le plus petit élément et finissant par le plus grand élément du bloc enfant.

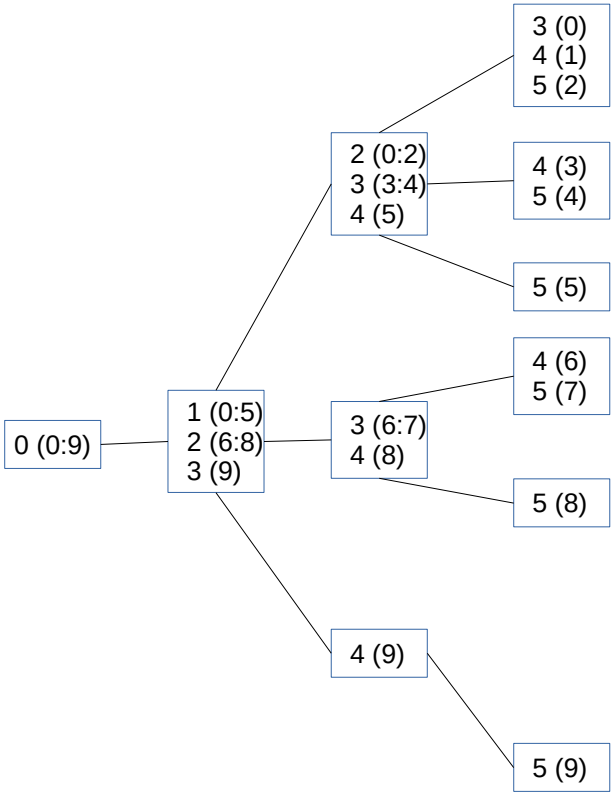
Démonstration

Modeliser.combinaison(6, 4)

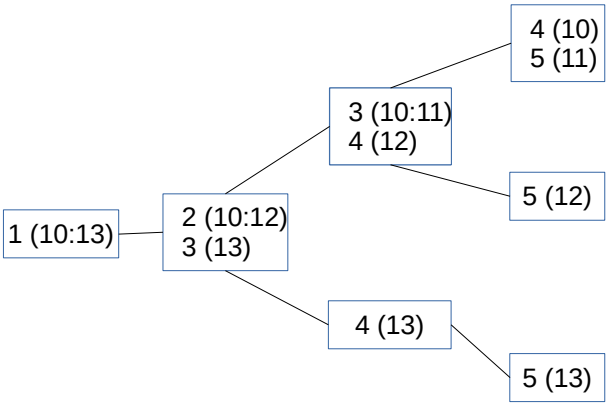
- Deux tableaux indépendants sont créés, chacun ayant ((***nCar*** - ***long***) + 1) lignes et ***long*** colonnes. Le premier tableau contient les valeurs, le deuxième tableau contient les numéros ordres de chaque résultat possible.

2.4

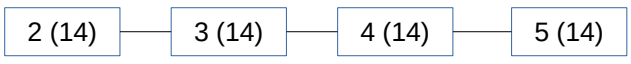
Modeliser.combinaison



0 : 0123
1 : 0124
2 : 0125
3 : 0134
4 : 0135
5 : 0145
6 : 0234
7 : 0235
8 : 0245
9 : 0345



10: 1234
11: 1235
12: 1245
13: 1345



14: 2345

Résultat:

Tableau des valeurs: [[[[0]], [[1, 2, 3]], [[2, 3, 4], [3, 4], [4]], [[3, 4, 5], [4, 5], [5], [4, 5], [5], [5]]], [[[1]], [[2, 3]], [[3, 4], [4]], [[4, 5], [5], [5]]], [[[2]], [[3]], [[4]], [[5]]]]

Tableau des numéros d'ordre: [[[[[0, 9]]], [[[0, 5], [6, 8], [9, 9]]], [[[0, 2], [3, 4], [5, 5]], [[6, 7], [8, 8]], [[9, 9]]], [[[0], [1], [2]], [[3], [4]], [[5]], [[6], [7]], [[8]], [[9]]]], [[[[10, 13]]], [[[10, 12], [13, 13]]], [[[10, 11], [12, 12]], [[13, 13]]], [[[10], [11]], [[12]], [[13]]]], [[[[14, 14]]], [[[14, 14]]], [[[14, 14]]], [[[14]]]]]

3

Convertir.Liste

Introduction

La classe Liste de Convertir permet de générer ***l*** à partir de ***num***. Chaque fonction requiert son modèle correspondant ainsi que ***num***.

Structure

Convertir.Liste.<algorithme>(***m***, ***num***)

1 résultat de type liste

3.1

Convertir.Liste.arrangement

Fonctionnement

Le modèle est parcouru en partant de la première colonne et divise le reste de ***num*** par la valeur de la position actuelle. Chaque résultat est enregistré dans une liste finale

Démonstration

Convertir.Liste.arrangement([216, 36, 6, 1], 7)

- $v_{1,0}$: ***num*** / $v_{0,0}$ (7 / 216) = 0
- $v_{1,1}$: ***num*** / $v_{0,1}$ (7 / 36) = 0
- $v_{1,2}$: ***num*** / $v_{0,2}$ (7 / 6) = 1, **num** = 7 - 6
- $v_{1,3}$: ***num*** / $v_{0,3}$ (1 / 1) = 1, **num** = 1 - 1

Résultat :

0	**0**	**1**	**1**

3.2

Convertir.Liste.permutation

Fonctionnement

- Le premier modèle (augmenté de 1 ligne) est parcouru en partant de la première colonne et divise le reste de ***num*** par l'élément de la position actuelle de la première ligne pour inscrire le résultat dans la deuxième ligne de la même colonne.

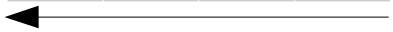
- Les éléments sont considérés comme des emplacements d'une liste imaginaire commencant par 0 et finissant par **nCar**. La valeur actuelle est augmentée de 1 à chaque fois qu'une valeur précédement enregistrée lui est inférieure ou égale.

Démonstration

Convertir.Liste.permutation([120, 20, 4, 1], 26)

- $v_{1,0}$: ***num*** / $v_{0,0}$ (26 / 120) = 0
- $v_{1,1}$: ***num*** / $v_{0,1}$ (26 / 20) = 1, ***num*** = 26 - 20
- $v_{1,2}$: ***num*** / $v_{0,2}$ (6 / 4) = 1, ***num*** = 6 - 4
- $v_{1,3}$: ***num*** / $v_{0,3}$ (2 / 1) = 2, ***num*** = 2 - 2

120	20	4	1
0	1	1	2



- 0 : [**0**, 1, 2, 3, 4, 5, 6], aucune valeur n'a été précédement retirée, 0
- 1 : [1, **2**, 3, 4, 5, 6] 1 élément (0) a été précédement retiré, 1 + 1 = 2
- 1 : [1, **3**, 4, 5, 6] 1 élément (0) a été précédement retiré, 1 + 1 = 2. 2 a précédement été retiré, 2 + 1 = 3
- 2 : [1, 4, **5**, 6] 2 éléments (0, 2) ont été précédement retiré, 2 + 2 = 4. 1 élément (3) a été précédement retiré, 4 + 1 = 5

Résultat :

0	2	3	5

3.3

Convertir.Liste.Combinaison

Fonctionnement

- Le modèle est parcouru en partant de la première colonne de la dernière pour comparer la valeur de la colonne actuelle avec ***num***.
- Si ***num*** est plus grand ou égal à la valeur de la position actuelle alors cette valeur est déduite de ***num*** et les valeurs des colonnes suivantes de la même ligne sont également déduites de ***num*** tant que ***num*** reste plus grand ou égal à la valeur de la colonne parcourue.
- Sinon la valeur de la colonne actuelle de la ligne suivante est comparée avec ***num*** et passe à la ligne suivante de la même colonne si ***num*** n'est pas plus grand que la valeur de la colonne actuelle.

Démonstration

Convertir.Liste.Combinaison([[1, 1, 1, 1, 1], [5, 4, 3, 2, 1], [15, 10, 6, 3, 1], [35, 20, 10, 4, 1]], 7)

- ***num*** < $v_{3,0}$ (7 < 35) : résultat = {0, x, x, x}, ligne suivante
- ***num*** < $v_{2,0}$ (7 < 15) : résultat = {0, 0, x, x}, ligne suivante
- ***num*** > $v_{1,0}$ (7 > 5) : 7 - 5 = 2, colonne suivante :
  num < $v_{1,1}$ (2 < 4) : résultat = {0, 0, 1, x}, ligne suivante
- ***num*** > $v_{0,1}$ (2 > 1) : 2 - 1 = 1, colonne suivante :
  num == $v_{0,2}$ (1 == 1) : 1 - 1 = 0, colonne suivante :
  num < $v_{0,2}$ (0 < 1) : résultat = {0, 0, 1, 3}, fin

Résultat :

1	1	1	1	1
5	4	3	2	1
15	10	6	3	1
35	20	10	4	1

num = 0
num = 2
num = 7
num = 7

0	0	1	3

3.4

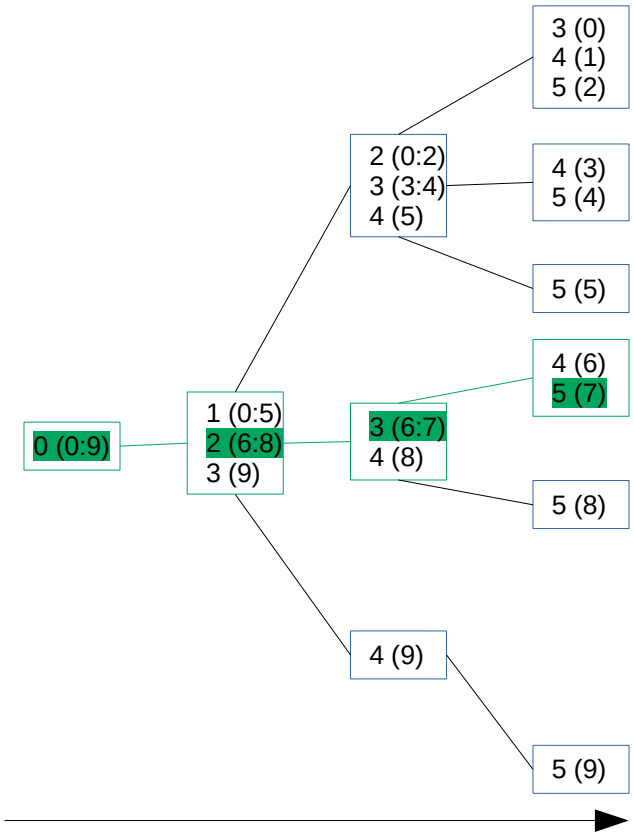
Convertir.Liste.combinaison

Fonctionnement

- Une boucle parcours d'abord chaque ligne du modèle des intervalles pour vérifier si ***num*** est inclu dans l'intervalle du bloc de la première profondeur.
- Si c'est le cas alors la ligne concernée est parcourue en profondeur pour vérifier si un des blocs enfant du bloc actuel comprend un intervalle incluant ***num*** pour poursuivre la vérification des blocs enfants du bloc vérifié jusqu'à arriver à un simple numéro au lieu d'un intervalle. La position de l'invervalle incluant ***num*** est utilisée sur le modèles des éléments pour enregistrer l'élément de la même position.

Démonstration

Convertir.Liste.combinaison_r([[[[0]], [[1, 2, 3]], [[2, 3, 4], [3, 4], [4]], [[3, 4, 5], [4, 5], [5], [4, 5], [5], [5]]], [[[1]], [[2, 3]], [[3, 4], [4]], [[4, 5], [5], [5]]], [[[2]], [[3]], [[4]], [[5]]], [[[[0, 9]]], [[[0, 5], [6, 8], [9, 9]]], [[[0, 2], [3, 4], [5, 5]], [[6, 7], [8, 8]], [[9, 9]]], [[[0], [1], [2]], [[3], [4]], [[5]], [[6], [7]], [[8]], [[9]]]], [[[[10, 13]]], [[[10, 12], [13, 13]]], [[[10, 11], [12, 12]], [[13, 13]]], [[[10], [11]], [[12]], [[13]]]], [[[[14, 14]]], [[[14, 14]]], [[[14, 14]]], [[[14]]]]], 7)



- La première ligne est séléctionnée pour vérifier le bloc de la première profondeur : l'intervalle du bloc inclu ***num***, cette ligne sera parcourue, résultat = {0, x, x, x}
- 1 bloc enfant est présent, le deuxième élément inclu ***num***, son bloc enfant sera analysé, résultat = {0, 2, x, x}
- Le premier intervalle inclu ***num***, son bloc enfant sera analysé, résultat = {0, 2, 3, x}
- Le deuxième élément correspond à ***num***, résultat = {0, 2, 3, 5}

Résultat :

0	2	3	5

4

Convertir.Num

Introduction

La classe Num de Convertir permet de générer ***num*** à partir de ***l***. Chaque fonction requiert son modèle correspondant ainsi que ***l***.

Structure

Convertir.Num.<algorithme>(***m***, ***l***)

1 résultat de type entier

4.1

Convertir.Num.arrangement

Fonctionnement

l est parcouru à partir de la première colonne pour additionner le résultat de l'élément de la colonne actuelle de ***m*** multiplié par l'élément de ***l*** de la même position à ***num*** initialement à 0.

Démonstration

Convertir.Num.arrangement([216, 36, 6, 1], {0, 0, 1, 1})

- ***num*** += m_0 x l_0 (216 x 0) : 0, ligne suivante
- ***num*** += m_1 x l_1 (36 x 0) : 0, ligne suivante
- ***num*** += m_2 x l_2 (6 x 1) : 6, ligne suivante
- ***num*** += m_3 x l_3 (1 x 1) : 1, fin

216	36	6	1
0	0	1	1
+0	+0	+6	+1

Résultat : **7**

4.2

Convertir.Num.permutation

Fonctionnement

Une liste imaginaire commencant par 0 et finissant par ***nCar*** est considérée. La valeur actuelle est soustraite de 1 à chaque fois qu'une valeur précédement enregistrée lui est inférieure.

Cette nouvelle liste est ensuite parcourue pour multiplier chacun de ses éléments par la valeur de la même position dans le modèle. Une fois terminé, l'ensemble des résultats des multiplications sont additionnés.

Démonstration

Convertir.Num.permutation([120, 20, 4, 1], {0, 2, 3, 5})

- 0 : [**0**, 1, 2, 3, 4, 5, 6] Aucune valeur n'a été précédement retirée, 0
- 2 : [1, **2**, 3, 4, 5, 6] 1 élément (0) a été précédement retiré, 2 – 1 = 1
- 3 : [1, **3**, 4, 5, 6] 2 éléments (0, 2) ont été précédement retirés, 3 – 2 = 1
- 5 : [1, 4, **5**, 6] 3 éléments (0, 2, 3) ont été précédement retirés, 5 – 3 = 2

120	20	4	1
0	1	1	2
+0	+20	+4	+2

- ***num*** += $v_{1,0}$ x $v_{0,0}$ (0 x 120) : 0
- ***num*** += $v_{1,1}$ x $v_{0,1}$ (1 x 20) : 0 + 20
- ***num*** += $v_{1,2}$ x $v_{0,2}$ (1 x 4) : 20 + 4
- ***num*** += $v_{1,3}$ x $v_{0,3}$ (2 x 1) : 24 + 2

Résultat : **26**

4.3

Convertir.Num.Combinaison

Fonctionnement

Le modèle est parcouru à partir de la première colonne de la dernière ligne pour additionner tous les éléments valides entre eux.

Démonstration

Convertir.Num.Combinaison([[1, 1, 1, 1, 1], [5, 4, 3, 2, 1], [15, 10, 6, 3, 1], [35, 20, 10, 4, 1]], {0, 0, 1, 3})

- 0 : ***num*** = 0
- 0 : ***num*** += 0
- 1 : ***num*** += 5
- 3 : ***num*** += 1 + 1

1	1	1	1	1
5	4	3	2	1
15	10	6	3	1
35	20	10	4	1

num = 5 + 1 + 1

Num = 0 + 5

num = 0

num = 0

Résultat : **7**

4.4

Convertir.Num.combinaison

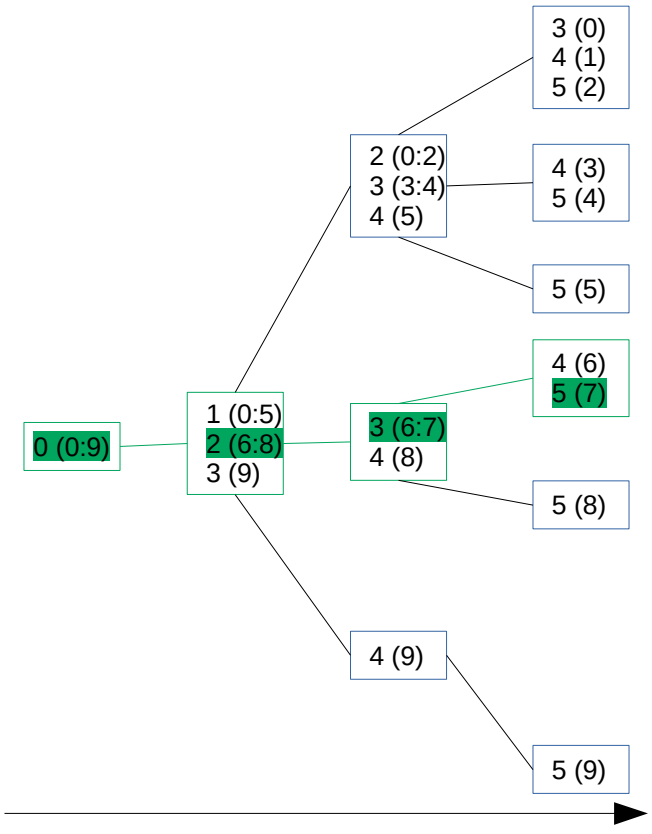
Fonctionnement

- Chaque profondeur du modèle d'éléments est parcourue en suivant le chemin imposé par ***l***. Le dernier élément est associé au résultat final.

Démonstration

Convertir.Num.combinaison([[[[0]], [[1, 2, 3]], [[2, 3, 4], [3, 4], [4]], [[3, 4, 5], [4, 5], [5], [4, 5], [5], [5]]], [[[1]], [[2, 3]], [[3, 4], [4]], [[4, 5], [5], [5]]], [[[2]], [[3]], [[4]], [[5]]], [[[[0, 9]]], [[[0, 5], [6, 8], [9, 9]]], [[[0, 2], [3, 4], [5, 5]], [[6, 7], [8, 8]], [[9, 9]]], [[[0], [1], [2]], [[3], [4]], [[5]], [[6], [7]], [[8]], [[9]]]], [[[[10, 13]]], [[[10, 12], [13, 13]]], [[[10, 11], [12, 12]], [[13, 13]]], [[[10], [11]], [[12]], [[13]]]], [[[[14, 14]]], [[[14, 14]]], [[[14, 14]]], [[[14]]]]], {0, 2, 3, 5})

- l_0 correspond au seul bloc, l'éléments est en 1ère place : ***num*** est entre 0 et 9,
- l_1 correspond donc au premier bloc, l'élément est en 2ème place : ***num*** est entre 6 et 8,
- l_2 correspond donc au 2ème bloc, l'élément est en 4ème place : ***num*** est entre 6 et 7,
- l_3 correspond donc au 4ème bloc, ***num*** est 7.



Résultat : **7**

5

Estimer

Introduction

La classe Estimer permet d'obtenir l'estimation maximum du nombre de possibilités via ***nCar*** et ***long***.

Structure

Estimer.<algorithme>(***nCar***, ***long***)

1 résultat de type entier

Estimer.arrangement(*10*, *4*)

Résultat : 10000

Estimer.permutation(*10*, *4*)

Résultat : 5040

Estimer.Combinaison(*10*, *4*)

Résultat : 715

Estimer.combinaison(*10*, *4*)

Résultat : 210