# Exercices

FIBONACCI

La suite de Fibonacci est définie par F0 = 0, F1 = 1 et Fn = Fn-1 + Fn-2  
Fibonacci = { 0, 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , … }

|  |  |
| --- | --- |
| Input | n un entier, 0 ≤ n ≤ 45 |
| Output | Calculer le nième nombre de Fibonacci, sans utiliser la récurrence. |
| Source | Fibonacci.java |

Pour aller plus loin, comparer les temps d’exécution entre votre solution et la solution récurrente.

Chiffre des unités des nombre de Fibonacci

|  |  |
| --- | --- |
| Input | n un entier 0 ≤ n ≤ 107 |
| Output | Calculer le chiffre des unités du nième nombre de Fibonacci.  Attention n peut être très grand, il n’est donc pas possible de calculer le nombre de Fibonacci. |
| Source | FibonacciLastDigit.java |

Exemple :   
F331= 668 996 615 388 005 031 531 000 081 241 745 415 306 766 517 246 774 551 964 595 292 186 469  
Output = 9

PGCD

Le PGCD est le plus grand commun diviseur. Le PGCD de 20 et 15 est 5.

|  |  |
| --- | --- |
| Input | A et b deux entiers 0 ≤ a , b ≤ 108 |
| Output | Calculer PGCD(a,b) en utilisant l’algorithme d’Euclide |
| Source | GCD.java |

Lemme : si a’ est le reste de la division de a par b alors PGCD(a,b)=PGCD(b,a’)

PGCD(1344, 217) = PGCD(217, 42) = PGCD(42, 7) = PGCD(7, 0) = 7  
PGCD(61,13) = 1

PGCD(28851538,1183019) = 17657

PPCM

Le PPCM est le plus petit commun multiple de deux nombres. PPCM(3,5) = 15

|  |  |
| --- | --- |
| Input | A et b deux entiers 0 ≤ a , b ≤ 108 |
| Output | Calculer PPCM(a,b) en utilisant l’algorithme d’Euclide |
| Source | LCM.java |

Tips : utiliser le PGCD.

PPCM(761457,614573) = 467970912861

Car Fueling

Vous partez en voiture pour une destination à une distance de D km.

Votre voiture a une autonomie de X km.

Vous pouvez faire le plein aux kilomètres stop1, stop2, etc.

L’objectif est de faire le voyage avec un minimum d’arrêts pour faire le plein.

|  |  |
| --- | --- |
| Input | L’entier dist est le nombre total de km, 0 ≤ dist ≤ 107  L’entier tank est l’autonomie en km, 0 ≤ tank ≤ 103  Le tableau d’entier stop[] contient les kilométrages des arrêts possibles |
| Output | Calculer le nombre minimum d’arrêt  Si le parcours est impossible, retourner -1 |
| Source | CarFueling.java |

|  |  |
| --- | --- |
| Input | dist = 950 , tank = 400 , stop = [200,375,550,750] |
| Output | 2 |

Intervalles

On vous donne un ensemble de n segments : { [ a1 , a1 ] , [ a2 , a2 ] , … , [ an , an ] }, trouver le nombre minimum de point tel que chaque segment contienne au moins 1 point.

Par exemple avec { [ 0 , 3] , [ 2 , 5 ] [ 6 , 7] } deux points suffisent : { 3 et 7 }

|  |  |
| --- | --- |
| Input | Un tableau d’objet Segment[] , un segment est composé de deux entiers, start et end. |
| Output | Liste des points – La solution est unique si pour chaque point choisi, on prend la plus grande valeur possible. |
| Source | CoveringSegments.java |

Sous sommes

Le but est d’écrire un entier n sous la forme d’une somme d’entier, telle que chacun de ces entiers soient différents et telle que cette somme soit la plus longue possible.

Soit N un entier, trouver a1, a2, …, ak tels que :

N = a1 + a2 + … + ak, ai ≠ aj pour tout 0 ≤ (i,j) ≤ k , et k est maximum

|  |  |
| --- | --- |
| Input | Une entier n |
| Output | La liste [a1, a2, …, ak] |
| Source | DifferentSummands.java |

Binary Search

Trouver un entier k dans une liste croissante d’entiers, tous différent.

Soit E = [a1 < a2 < …< an], l’entier k est-il un élément de E ?

|  |  |
| --- | --- |
| Input | Une liste d’entier [a1 , …, an] avec a1 < a2 < …< an  Un entier k |
| Output | L’indice de k dans la liste, -1 si non trouvé |
| Source | BinarySearch.java |

QuickSort

Implémenter l’algorithme QuickSort.

|  |  |
| --- | --- |
| Input | Une liste d’entier L = [a1 , …, an] |
| Output | La liste d’entier L triée par ordre croissant |
| Source | Sorting.java |

Money Change

L’objectif est de rendre la monnaie avec un minimum de pièces.

Quelle est le nombre minimum de pièces à rendre sur une somme S avec un ensemble donné de pièces ? Par exemple, COINS = {1 , 3 , 4 } , Sum = 6 , solution = 2 { 3 , 3 }

|  |  |
| --- | --- |
| Input | Une liste d’entier coins = [a1 , …, an]  Une somme à rendre S |
| Output | Le nombre minimum de pièce |
| Source | ChangeDP.java |

Knapsack

Un sac à dos a une capacité maximum de W kg.  
Une liste d’objets avec des valeurs et des poids différents est à disposition.  
Quels sont les objets à mettre dans le sac à dos pour maximiser la valeur totale emportée ?

Exemple :   
La capacité maximum est de 10 kg.  
4 objets sont à disposition :

* Objet 1 : poids = 6 Kg, valeur = 30 EUR
* Objet 2 : poids = 3 Kg, valeur = 14 EUR
* Objet 3 : poids = 4 Kg, valeur = 16 EUR
* Objet 4 : poids = 2 Kg, valeur = 9 EUR

La solution est 1 et 2 pour 6 Kg + 4 Kg et 46 EUR

|  |  |
| --- | --- |
| Input | Un entier W : capacité maximale  Deux listes pour les n objets :  w[] la liste des poids (liste d’entier)  v[] la liste des valeurs (liste d’entier) |
| Output | Valeur optimale |
| Source | Knapsack.java |

Check Brackets

Le but de cette exercice est de vérifier la syntaxe d’une expression en vérifiant que toutes les parenthèses ouvrantes de types  [, (, { sont bien fermées par une parenthèse correspondante ],)ou}.

Exemples :   
[] est ok   
jk()[] est ok  
jk{)} est nok  
(4545(4545(4545(54)))) est ok etc.

|  |  |
| --- | --- |
| Input | String text |
| Output | void |
| Exception | ParseException |
| Source | CheckBrackets.java |

Tips : utiliser une pile pour empiler les parenthèses ouvrantes.

Heap Binary Tree Implementation

Le but est d’implémenter une structure de type Max Heap Binary Tree afin de stocker une Priority Queue.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Le but est d’implémenter la structure à l’aide d’une liste.   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | 42 | 29 | 18 | 14 | 7 | 18 | 12 | 11 | … | … | |

Les fonctions à implémenter sont :

|  |  |
| --- | --- |
| GetParentIdx(int i) : int | Indice du parent de i  Ex : pour i = 3 ou 4, la fonction retourne 1 |
| GetParent(int i) : Integer | Valeur du parent de i |
| GetLeftChildIdx(int i) : int | Indice du « left child » de i  Ex : pour i = 2, la fonction retourne 5 |
| GetLeftChild (int i) : Integer | Valeur du « left child » |
| GetRightChildIdx(int i) : int | Indice du « right child » de i  Ex : pour i = 2, la fonction retourne 6 |
| GetRightChild (int i) : Integer | Valeur du « right child » |
| Swap(int i , int j):void | Swap la valeur de i et de j, cette fonction est utilisée dans les fonctions suivantes. |
| SiftUp(int i) : void | Tant que la valeur du parent de i est inférieure à la valeur de i   * On échange i et son parent |
| SiftDown(int i) :void | Si l’un des enfants de i a une valeur supérieure à i, on échange l’enfant qui possède la plus grande valeur avec i.  Puis on réapplique ce test sur l’enfant concerné. (Récursivité) |
| Insert(Integer p) :void | Ajouter l’élément P à la liste (à la fin), puis appliquer la fonction siftUp à cet élément. |
| ExtractMax() :Integer | Cette fonction retourne le premier élément de la liste. Cet élément doit être supprimé de la liste, pour cela :  Swap(0,dernier)  Delete(dernier)  siftDown(0) |
| Remove(int i) :void | Affecter à la valeur de i, +∞  SiftUp(i)  ExtractMax() |
| changePriority(int i, Integer p) :void | Affecter à i sa nouvelle valeur p  Si sa nouvelle valeur > ancienne valeur   * siftUp(i)   sinon   * siftDown(i) |

Source : MaxHeap.java

Plus court chemin

Best special index sum

Soit E = [a1, a2, …, aN], un ensemble d’entier (positifs ou négatifs)

On définit pour chaque indice i, la somme suivante :  
Sum(i) = E[i]   
 + E[i+1] + E[i+2]  
 + E[i+3] + E[i+4] + E[i+5]  
 + E[i+6] + E[i+7] + E[i+8] + E[i+9]  
 + …

Dis autrement,   
Sum(i) = somme de somme des éléments de E d’indice i à K,   
avec K = 1+ 2 +3 + ...+ n  
et n est choisi tel que K soit maximum et i+K ne dépasse pas N

Trouver Sum(i) maximum

Exemple avec E = [-3,2,3,-4,3,1] :  
Sum(0) = -3 + 2+3 + (-4)+3+1 = 2  
Sum(1) = 2 + 3+(-4) = 1  
Sum(2) = 3 + (-4)+3 = 2  
Sum(3) = (-4) + 3+1 = 0  
Sum(4) = 3 🡪 La solution  
Sum(5) = 1

|  |  |
| --- | --- |
| Input | Une liste d’entier  Ex : 2 1 3 9 2 4 -10 -9 1 3 |
| Output | Valeur optimale Résultat : 9 |
| Source | BestSpecialIndexSum.java |
| Performance | Pour N = 100000 , temps d’exécution < 5s |

import math as math

a = list(map(int, input().split()))

a=range(1,1000000)

def findn(N):

result = -1;

for i in range(int(math.sqrt(2\*N)),200000):

if (i\*(i-1)>2\*N):

break

result=i

return int(result\*(result-1)/2)

def findMax(a):

end = findn(len(a))

s=maxi=sum(a[0:end])

for i in range(1,len(a)):

end\_1=end

s\_1 = s

end=findn(len(a)-i)+i

s=s\_1-a[i-1]

if (end>end\_1):

s=s+sum(a[end\_1:end])

elif (end<end\_1):

s=s-sum(a[end:end\_1])

if (s>maxi):

maxi = s

return maxi

start\_time = time.time()

print(findMax(a))

print("--- %s seconds ---" % (time.time() - start\_time))