***RECURSIVITE***

***Exercice 1 : Recherche de chemin - Version récursive***

*Le programme consiste à rechercher dans un réseau routier un chemin entre une ville de départ D et une ville d’arrivée A. A la fin du programme on devra soit afficher qu’il n’existe aucune route entre D et A, et dans le cas contraire on affichera la liste des villes à parcourir entre D et A et le nombre de Km.*

*Exemple : Un réseau est représenté par une matrice carré n\*n sous la forme.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *Réseau :*   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | |  | ***S0*** | ***S1*** | ***S2*** | ***S3*** | ***S4*** | ***S5*** | | ***S0*** | *0* | *7* | *3* | *0* | *0* | *0* | | ***S1*** | *0* | *0* | *0* | *1* | *1* | *0* | | ***S2*** | *4* | *0* | *0* | *1* | *0* | *6* | | ***S3*** | *0* | *2* | *0* | *0* | *4* | *0* | | ***S4*** | *0* | *0* | *0* | *0* | *0* | *1* | | ***S5*** | *0* | *0* | *0* | *2* | *3* | *0* | |

*Réseau[0,1]=7 indique qu’il existe une route entre la ville S0 et la ville S1 qui est de 7km. Par contre Réseau[0,3]=0 correspond au fait qu’il n’existe pas de route entre les villes S0 et S3. Dans un premier temps on cherche un chemin entre deux villes. Pour cela on aura besoin de disposer d’une ville de départ D, d’une ville d’arrivée A, d’une liste contenant les villes parcourues L et d’un nombre de km K.*

* *La première étape consiste à initialiser le réseau, via par exemple la lecture sur un fichier de texte.*
* *La fonction* ***chemin*** *est une fonction qui initialise les différentes variables du problème. Elle demande à l’utilisateur la ville de départ D et la ville d’arrivée A. Elle initialise la liste L avec la ville de départ D, et le nombre de km K=0. La fonction* ***etapeSuivante*** *est la fonction récursive qui reçoit le Réseau, L, K et la ville d’arrivée A.*
* *Au retour d’****etapeSuivante*** *on affiche le résultat si l’on a trouvé un chemin.*
* *La fonction* ***étapeSuivante*** *débute par un test qui regarde si la dernière ville entrée dans L est égale à A. Dans ce cas on a trouvé et l’on doit quitter la fonction. Sinon on doit faire une boucle (V) sur toutes les villes possibles et regarder si la nouvelle ville V est une étape possible. Pour cela la ville ne doit pas être dans L et il doit exister une route entre la dernière ville de L et V. Dans ce cas on peut passer à l’étape suivante. On doit donc ajouter V dans L, ajouter le nombre de Km dans K puis rappeler la fonction* ***etapeSuivante****.*
* *Au retour d’****etapeSuivante*** *si l’on a trouvé un chemin on quitte la procédure sinon on fait un backtracking, qui consiste à retirer V de L et le nombre de Km de K.*

*Dans la fonction* ***étapeSuivante*** *les villes sont testées de la première à la dernière on devrait obtenir L=[S0, S1, S3, S4, S5] et K=13 si la ville de départ est S0 et la ville d’arrivée est S5.*

*Si l’on souhaite maintenant recherche le plus court chemin, il est nécessaire de transformer programme précédant. On doit disposer de deux nouvelles variables Lmin et Kmin qui seront initialisées dans* ***chemin*** *à Lmin=Ø et Kmin=+infini*

* *Dans ce cas nous n’avons plus besoin de la variable trouvée. Seulement au sorti de* ***epateSuivante*** *dans* ***chemin*** *si Lmin et Kmin ont été modifiées il suffit d’afficher le résultat.*
* *La fonction* ***epateSuivante*** *débute toujours par un test qui consiste à savoir si la dernière valeur de L est égale à A. Dans ce cas on modifie Lmin=L et Kmin=K si Kmin est supérieur au courant K, puis on quitte la fonction dans tous les cas.*
* *La suite est identique si ce n’est qu’au retour de l’appel récursif il n’est plus nécessaire de traité le cas où l’on a trouvé une solution. On fait donc le backtraking dans tous les cas.*

*La variable trouvée n’est plus nécessaire, car il faut ici chercher toutes les possibilités et donc laisser le programme regarder toutes les villes étapes possibles avant de terminer. Et la solution sera alors Lmin=[S0, S2, S3, S1, S4, S5] et Kmin=8 si la ville de départ est S0 et la ville d’arrivée est S5.*

***Partie 2 - non récursive****. Le plus court chemin dans un graphe possède une propriété de sous-structure optimale. En effet, le plus court chemin entre deux villes est forcément composé que de plus courts chemins. Exemple : le plus court chemin entre S0 et S5 est donné par [S0, S2, S3, S1, S4, S5]=8km, et si l'on choisit deux sommets quelconque de cette liste (S2 et S4) alors [S2, S3, S1, S4]=4km est le plus court chemin entre ces deux sommets.*

*Ainsi le plus court chemin du sommet D au sommet S est donc un chemin qui va passer par un sommet prédécesseur de S.*



.

*L'algorithme de Dijsktra permet d'exploiter cette propriété afin de ne pas avoir à utiliser la récursivité pour chercher les plus courts chemins. Le principe consiste à parcourir tous les sommets accessibles depuis le sommet de départ (D) et à chaque étape on traite tous les successeurs pour mettre à jour le plus court chemin.*

*Pour cela chaque sommet pourra être dans un des trois états non-traité (blanc), a traiter (gris) ou traité (noir). A chaque sommet est associé un poids qui correspond à la distance minimum permettant d'atteindre le sommet depuis le sommet de départ D, et un prédécesseur qui référencie le sommet à emprunter.*

* *Tous les sommets sont initialisés à blanc, les poids à MaxInt et les prédécesseurs à null, excepté le sommet de départ qui est initialisé à gris et le poids à 0.*
* *L'algorithme tourne tant que la liste des sommets à traiter n'est pas nulle. A chaque étape on retire de cette liste le sommet qui a le poids minimum, et l'on évalue le plus court chemin pour y arriver en fonction des sommets déjà traités.*
* *Le principe consiste à mettre à jour les différents successeurs du sommet que l'on traite S en regardant si le chemin passant par S n'est pas meilleur que celui que l'on connait pour ce successeur.*
* *Si le successeur est noir il ne sera pas traité, un sommet blanc passe à gris. Le sommet S traité passe ensuite de gris à noir.*

*Exemple : Plus court chemin depuis D=S0.*

*Au départ Blanc=[S1, S2, S3, S4, S5] ; Gris=[S0] ; Noir=[]*

*Etape 1 : On retire S0 qui a un poids de 0. Les deux successeurs S2 et S3 passent à gris, et leurs paramètres poids et prédécesseurs passent pour S1 à (7/S0) et pour S2 à (3/S0). S0 pas lui à noir. Si un chemin nous ramène à ce sommet il n'est plus nécessaire de le traiter car il contient le plus court chemin de S0 à S0.*

*Etape 2 : On doit retirer S2 qui est le plus petit chemin, les deux successeurs S3 et S6 passent à gris, et leurs paramètres poids et prédécesseurs passent pour S3 à (4/S2) et pour S6 à (9/S2). S2 passe lui à noir et n'aura plus besoin d'être traité car il contient le plus court chemin depuis S0.*

*Etape 3 : On doit retirer S3 qui est le plus petit chemin, son successeur S1 reste à gris et S4 passe à gris, et de S4 sont (8/S3) et pour S1 ses paramètres passent (6/S3) qui est meilleur que le (7/S1). S3 qui contient le plus court chemin depuis S0 peut passer à noir.*

*A l'étape 4 c'est le sommet S1 qui sortira, puis le sommet S4 et enfin le S5.*

* *Implémentez l'algorithme de Dijsktra.*
* *L'algorithme commence par la lecture du réseau routier, puis on doit initialiser toutes les variables liées aux sommets (poids, prédécesseur, état).*
* *On doit ensuite boucler tant qu'il y a un sommet à traiter. On retirer le sommet avec le poids minimum, et on traite tous ses successeurs.*

***Exercice 2 :*** *Recherche de chemin pour infiltration d’eau.*

*L’infiltration peut être définie comme l’écoulement d’un fluide dans un sol poreux. Un terrain n’est pas étanche s’il existe un chemin permettant au fluide de le traverser. Dans cet exercice, on souhaite proposer un modèle simplifié, qui test l’étanchéité d’un terrain. Un terrain sera représenté par une structure de type tableau à deux dimensions dont les casses (ou zones) peuvent être considérées comme non poreuse (représenté par exemple par le caractère ‘#’) ou poreuse (caractère ‘ ‘).*

* *Proposez un algorithme naïf permettant de remplir le tableau de façon aléatoire. La fonction aléatoire utilisera un paramètre correspondant au pourcentage des zones poreuses par rapport au nombre total de zones.*

*On considère que l’eau ne peut s’écouler entre deux zones que si celles-ci sont voisines horizontalement ou verticalement. Il n’y aura donc pas d’écoulement en oblique. On suppose que de l’eau peut arriver depuis le haut du terrain et l’apport est infini, c’est à dire que si un interstice se remplit d’eau (caractère ‘o’), il ne vide pas son voisin qui était déjà rempli d’eau. Les limites du terrain ne sont pas poreuses.*

* *Écrire une fonction récursive basée sur la recherche de chemin qui permet de savoir si le terrain est poreux ou pas.*
* *Proposez maintenant un algorithme de recherche dynamique et non récursif qui permet d’obtenir le même résultat.*
* *Évaluer les complexités des deux algorithmes proposés.*

***Exercice 3 :*** *Fibonacci.*

*La suite de Fibonacci est une suite permettant de calculer une série de nombre de manière récursive.*

*F(0) = F(1) = 1*

*F(n) = F(n-1) + F(n-2) si n>2*

* *Ecrire une fonction récursive permettant de calculer la valeur de F(n).*
* *Proposez une fonction dynamique et non récursive qui permet de calcul F(n). Pour cela on remarquera que pour calculer F(n-1) on a besoin de F(n-2) et de F(n-3), et il en est de même pour F(n-2) et F(n-3). Aussi après avoir calculé F(n-2), si l'on a besoin de calculer F(n-1) il n'est peut-être pas nécessaire de recalculer F(n-2). Proposez une version qui soit conserve toutes les valeurs intermédiaires F(0) ... F(n-1) dans un tableau, soit une version qui ne conserve que les deux dernières valeurs de F.*
* *Evaluer les complexités des deux algorithmes proposés.*

***Exercice 4 :*** *Le compte est bon.*

*Le jeu consiste à trouver un nombre N en additionnant les nombres présents dans une liste L. Les nombres de L ne pouvant être utilisés qu'une seule fois.*

* *Proposez une version récursive qui affiche le premier résultat qui sera trouvé.*
* *Modifiez cette version pour afficher toutes les solutions possibles.*

*Si maintenant l'on peut effectuer également des soustractions entre les éléments de la liste, comment pourrait-être modifié le programme. Proposez une version très simple.*

***Exercice 5 :*** *Notation polonaise*

*La notation polonaise, est une notation mathématique d'expressions arithmétiques où les opérandes (nombres) sont écrits après les opérateurs (+,-,\*,/). Cette notation est adaptée à un usage technique en informatique/électronique, et a pour caractéristique d'éviter l'utilisation de parenthèses.*

*Exemples : (1) 2\*3 + 4 s'écrit + \* 2 3 4 ; (2) 2 \* ( 3 + 4) s'écrit \* 2 + 3 4*

*Aussi pour évaluer un opérateur, il faut trouver deux opérandes après. Ainsi dans l'exemple 1 l'addition ne peut pas s'effectuer directement elle attend deux opérandes. Donc on évalue \* avec 2 et 3 qui retournera un premier opérande et à ce moment + pourra s'effectuer avec le retour de \* 2 3 et la variable 4. De même pour le second exemple la multiplication ne peut récupérer qu'un opérande et doit attendre le résultat de + pour réaliser son travail.*

* *A partir d'un chaine de caractère qui contient une expression en notation polonaise, créer la liste (Expression) des opérandes et des opérateurs grâce à la fonction split().*
* *Écrire une fonction qui teste si un élément de la liste est un opérateur ou pas.*
* *Proposez maintenant une fonction récursive evalExpression qui évalue une expression en notation polonaise. La fonction recevra la liste Expression, elle retournera la valeur se trouvant en début de liste si celle-ci n'est pas un opérateur. Dans le cas contraire, elle devra récupérer l'instruction du début de la liste, puis évaluer les deux opérandes utiles à son évaluation.*
* *Proposez une évaluation de pour des expressions logiques, opérateurs (and, or et not).*

***Exercice 6 :*** *Les interrupteurs.*

*Un interrupteur est ouvert si certaines conditions sont remplies. Ces conditions portent sur des variables de type booléen, qui représentent les positions droite ou gauche. Des règles composées des opérateurs logiques (and, or et not) appliquées sur les variables indiquent si l'interrupteur est ouvert ou fermé.*

*Le principe de l'exercice consiste à savoir si pour un ensemble d'interrupteurs N liées à M variables, il existe une suite de position pour ces variables qui ouvrent tous les interrupteurs.*

*Exemple : N=5 (I1 à I5) et M=5 (V1 à V5)*

*I1 : V1 and not V5 ou and V1 not V5*

*I2 : V2 or (V3 and V4) ou or V2 and V3 and V4*

*I3 : not V1 or not V2 or or not V1 not V2*

*I4 : (V2 and V4) or (V3 and not V5) or or and V2 V4 and V3 not V5*

*I5 : V3 or (V2 and V5) or or V3 and V2 or V5*

*Dans ces circonstances (V1,V2,V3,V4,V5)=(1,0,1,1,0) est une solution pour ouvrir les 5 interrupteurs.*

* *Proposez des structures permettant d'enregistrer les valeurs des variables, ainsi que les expressions pour les différents interrupteurs.*
* *Créer une fonction qui teste si une configuration donnée pour les M variables autorisent l'ouverture de tous les interrupteurs.*
* *Implémentez un algorithme récursif qui va tester toutes les possibilités. La résolution est basée sur un parcours d'arbre binaire en profondeur, comme dans le cas de l'exercice 4. Au début de l'algorithme on doit vérifier si l'on a donné une valeur à toutes les variables, et dans ce cas on pourra tester si tous les interrupteurs sont ouverts. Sinon on doit donner une valeur à la variable i d’abord False et ensuite True, puis de passer à une autre variable.*

***Exercice 7 :*** *Problème du sac à dos.*

*Étant donné une série d'objets possédant chacun un poids et une valeur, et un sac à dos de capacité limité. La question est de savoir quels sont les objets que l'on peut mettre dans le sac à dos tout en maximisant la valeur transportée.*

*Exemple : Sac à dos (taille = 15kg)*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Objets* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* |
| *Prix* | *4* | *4* | *3* | *3* | *5* | *5* | *3* | *3* | *5* | *6* | *5* | *5* |
| *Poids kg* | *5* | *7* | *2* | *2* | *6* | *8* | *3* | *4* | *9* | *11* | *8* | *7* |

* *Proposez une structure de données permettant d'enregistrer les objets, le prix et les valeurs de chacun d'eux.*
* *En fonction de la taille du Sac à dos proposez un algorithme récursif "prendreObjet" qui test les différentes configurations acceptables, et retourne celle qui permet d'obtenir le meilleur prix.*
* *Votre fonction devra tout d'abord tester si l'on doit poursuivre ou arrêter la récursivité. Pour cela si l'on a rempli totalement le sac ou si l'on est sur le dernier objets de la liste on sauvegarde la solution si elle est meilleur que celle que l'on a déjà trouvé, et l'on quitte le programme.*
* *Si le programme peut continuer on teste si l'élément sur lequel on travaille peut-être mis, c'est à dire que son poids peut encore être transporté, on le place dans le sac et on passe à la récursivité suivante. Au retour on devra effectuer un backtracking avant de passer à l'appel récursif suivant sans prendre l'élément courant.*

***Partie 2 : Non récursive****. Ce problème possède une propriété de sous-structure optimale, c'est à dire que la solution optimale au niveau n contient une solution optimale à un niveau inférieur. Ainsi la solution optimale au niveau n peut être construite avec la solution optimale au niveau inférieur.*

*La solution optimale du problème de pour n élément avec un sac de taille w dépendant des solutions optimales d'un problème de sac à dos avec n-1 éléments. Si l'on souhaite calculer C(n,w) la solution optimale d'un sac à dos de contenance w pour n objets, il nous faut connaitre toutes les solutions optimales C(n-1,0) ... C(n-1,w).*

* *Si le dernier élément n a les caractéristiques (cn, wn) pour le prix et le poids la solution C(n,w) est soit la solution C(n-1,w) c'est à dire la même que pour n-1 objets à laquelle on ne prend pas l'objet n, soit la solution C(n-1,w-wn)+cn c'est à dire la solution optimale avec une place dans le sac (w-wn) suffisante pour y mettre l'objet n.*
* *Implémentez une solution dynamique. A chaque étape i on doit calculer toutes les solutions optimales pour C(i,0) à C(0,W) ou W est la contenance du sac. Pour i=0 toutes les valeurs sont nulles. Pour i on a C(i,w) = Max( C(i-1,w) ; C(i-1, w-wn)+cn).*
* *A chaque calcul étape on devra sauvegarder la liste des objets mis dans le sac. Si C(n,w) = C(n-1,w) la liste de objets dans le sac pour C(n,w) est la même que celle avec le dernier élément en moins, sinon la liste de objets dans le sac est celle avec n-1 objets et un sac de contenance w-wn plus l'élément n.*

***Exercice 8 :*** *Plus grand carré blanc.*

*Le but est de trouver le plus grand carré blanc présent dans une image monochrome n×n. Pour chaque pixel (x,y) on regarde quel est la taille d'un carré pour lequel (x,y) est un point situé en bas à droite de ce carré. Si le pixel x,y n'est pas blanc sa taille est à 0. Si le pixel (x,y) est le coin en bas à droite d'un carré de taille n alors les carrés de de taille n-1 des pixels (x, y-1) (x-1, y) et (x-1, y-1) sont aussi blancs.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  |  |  |  | |
| *PGCL(x,y)* | *PGCL(x-1,y-1)* | *PGCL(x-1,y)* | *PGCL(x,y-1)* |

* *La relation de récurrence permettant de calculer pour un pixel le carré blanc se trouvant au-dessus et à sa droite.*
* *Si (x,y) est un pixel noir PGCL(x,y) = 0.*
* *Si (x,y) est un pixel sur le bord de gauche et sur le bord du haut PGCL(x,y)=1.*
* *Pour tous les autres pixels on a PGCL = 1 + min(PGCL(x-1,y-1), nu*
* *Implémentez un algorithme permettant de calculer le plus grand carré blanc.*

***Exercice 9 :*** *Ordonnancement d'intervalles.*

*Un intervalle i est défini par une date de début di, une date de fin fi, et une valeur vi. Deux intervalles sont compatibles s'ils n'ont pas de date d'intersection. Le but est de trouver dans une série d'intervalles un sous-ensemble d'intervalles mutuellement compatibles et de valeur maximum.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *num* | *d* | *f* | *der* | *num* | *d* | *f* | *der* |
| *1* | *2* | *3* | *0* | *4* | *2* | *8* | *0* |
| *2* | *1* | *4* | *0* | *5* | *7* | *9* | *3* |
| *3* | *4* | *7* | *2* | *6* | *6* | *12* | *2* |

*Afin de pouvoir traiter ce problème on classe les intervalles par date de fin croissantes, et pour chacun des intervalles on identifie parmi tous les intervalles j tels que fj < di l'intervalle qui a le f maximum (der(i)).*

*La solution optimale d'une série de N intervalles peut se définir à partir de la solution optimale d'une série d'un nombre d'intervalles moins important. On peut identifier deux cas :*

* *Si N est dans la solution optimale, alors seuls les intervalles compris entre 1 et der(N) peuvent être pris pour calculer la solution optimale.*
* *Si N n'est pas dans la solution optimale alors tous les intervalles compris entre 0 et N-1 peuvent être choisi.*

*Afin de pouvoir traiter ce problème on classe les intervalles par date de fin croissantes, et pour chacun des intervalles on identifie parmi tous les intervalles j tels que fj < di l'intervalle qui a le f maximum (der(i)).*

* *Implémentez une version récursive de l'ordonnancement des intervalles.*
* *On souhaite maintenant implémenter une version dynamique, en choisissant de mémoriser toutes les Optimums entre 0 et N-1 avant de calculer l'optimum en N.*

***Exercice 10 :*** *Distance d'édition.*

*La distance d’édition permet de comparer deux séquences de symboles entre elles ou deux mots afin de trouver le nombre minimum de changements, d'insertions de suppressions ou de remplacements pour passer d'un mot à l'autre. Il existe plusieurs méthodes pour parvenir à ce résultat, celle de Levenshtein (Damerau Levenshtein Matching - DLM) est très utilisée. Les traitements de texte l’utilisent pour trouver pour un mot mal orthographié, quel est le mot le plus proche dans le dictionnaire.*

*Le nombre d'opération à réaliser est donné par une relation de récurrence dépendant de la distance optimale d(m,n) entre les m premiers caractères de la première chaîne et les n caractères de la seconde. Si une chaine est vide le nombre d'opération est égal au nombre de caractère de la seconde chaine soit :*

*Et la fonction de récurrence :*

* *Créez une fonction distanceEdition qui à partir de deux mots calcul le nombre d'opération nécessaire pour changer le premier mot vers le second.*
* *Afin d'éviter d'utiliser une fonction récursive, vous utiliserez un tableau distance dans lequel distance[m,n] contiendra le nombre d'opération nécessaire pour passer d'un mot de taille m vers un mot de taille n.*
* *L'initialisation de distance consiste à mettre les valeurs dans toutes les cases qui ont une coordonnée à 0.*
* *Le programme est composé de deux boucles, l'une sur les caractères du premier mot et l'autre sur les caractères du second. Ces boucles consiste à mettre à jour la variable de distance au regard de la formule de récurrence.*
* *L'algorithme est ici un algorithmique montant. On part des valeurs basses pour arriver en haut du tableau des distances.*

*Pour obtenir une distance d'édition minimale on peut également chercher la plus longue sous-suite commune (PLSS) entre deux mots afin de trouver les caractères qui pourraient être conservés entre les mots Si entre deux caractères de la PLSS il y a N caractères dans les deux mots on aura N remplacements à effectuer, s'il y a M caractères en plus dans le premier mot on devra faire M suppression en plus des comparaisons et dans le cas inverse il y aura des suppressions.*

*Exemple : Distance d'édition entre DETENTES => DISTANCE*

*PLSS = DTNE. Ces quatre caractères doivent être alignés pour un coût de 0.*

*Entre D et T - 1 lettre dans mot1 - 2 dans mot2 => 1 remplacement + 1 insertion*

*Entre T et N - 1 lettre dans mot1 et mot2 => 1 remplacement*

*Entre N et E - 1 lettre dans mot1 et mot2 => 1 remplacement*

*Après E - 1 lettre dans mot1 - 0 dans mot2 => 1 suppression*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Mot* | *Mot* | *Opération* | *Coût* | *Mot* | *Mot* | *Opération* | *Coût* |
| *D* | *D* | *Comparaison* | *0* | *N* | *N* | *Comparaison* | *0* |
| *E* | *I* | *Remplacement* | *1* | *T* | *C* | *Remplacement* | *1* |
| *-* | *S* | *Insertion* | *1* | *E* | *E* | *Comparaison* | *0* |
| *T* | *T* | *Comparaison* | *0* | *S* | *-* | *Suppression* | *1* |
| *E* | *A* | *Remplacement* | *1* |  |  | *TOTAL* | *5* |

*La fonction de récurrence PLSS(m,n) pour un mot1 de taille m et un mot2 de taille n va dépendre du fait que le deux derniers caractères des mots 1 et 2 sont égaux ou pas. Si les deux caractères sont égaux alors PLSS(m,n) est égal à PLSS(m-1,n-1)+1, sinon PLSS(m,n) est le max de la PLSS avec un caractère de moins dans le premier mot ou un caractère de moins dans le second mot.*

*A l'initialisation PLSS(0,j)=PLSS(i,0)=PLSS(0,0)=0.*

* *Implémentez l'algorithme PLSS, en utilisant une approche récursive. Par contre les résultats intermédiaires PLSS(m,n) seront enregistrés (tableau tab) afin d'éviter d'effectuer l'appel récursif si la valeur a déjà été calculée.*
* *Pour cela on initialisera toutes les valeurs du tableau à -1 par exemple. Ainsi lors de l'appel de PLSS si la case (m,n) est à un il faut quitter la fonction.*
* *L'algorithme s'arrête également si l'une des variables m ou n est à 0. On enregistre la valeur avant de quitter la fonction.*
* *Le calcul d'un élément PLSS(m,n) correspond à trois appels de PLSS à un niveau inférieur. L'algorithme est ici descendant, depuis les derniers caractères des deux mots jusqu'au calcul du PLSS entre des chaines vides.*

*L'algorithme peut être amélioré. En effet, la valeur de (m,n) ne peut augmenter que de 1. Aussi si les caractères en position m et n sont égaux il n'est pas nécessaire de prendre en compte PLSS(m-1,n) et PLSS(m,n-1), et dans le cas contraire il n'est pas nécessaire de calculer PLSS(m,n)*

* *Modifié l'algorithme pour tenir compte de cette propriété.*
* *En affichant le tableau que constate-t-on.*

*On souhaite maintenant retrouver à quelles positions se trouvent les caractères communs dans les chaines.*

* *Implémentez une fonction qui identifie les positions de ces caractères. Pour cela on part du dernier élément du tableau jusqu'au point (0,0).*
* *A chaque couple de (m,n) si les caractères mot1[m] est égale à mot2[n] on peut enregistrer les postions dans les listes respectives et décrémenter les deux valeurs du couple de 1.*
* *Si les caractères ne sont pas égaux on diminue de 1 m si tab[m-1,n] est supérieur à tab[m, n-1] et n dans le cas contraire.*

***Exercice 11 :*** *Découpe d'une barre de métal.*

*Le but est de chercher comment découper une barre de métal afin de maximiser les profits, sachant que chaque longueur a un prix. La valeur maximale peut être exprimée par une formule de récurrence, qui dépend de deux cas. Le premier consiste à conserver la barre entière, le second consiste à couper la barre en deux partie la première est une découpe entière de longueur i et l'autre de longueur (n-i) que l'on peut découper à nouveau.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Longueur* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
| *Prix* | *0* | *2* | *5* | *7* | *9* | *10* | *12* | *14* | *15* |

*La fonction de récurrence.*

* *Implémentez la fonction récurrence maxPrix(n) permettant de calculer le prix maximal que l'on peut obtenir à partir de la barre.*
* *Afin d'éviter une implémentation par un algorithme récursif il est nécessaire d'enregistrer les valeurs successives de MaxPrix.*
* *L'algorithme est une fonction croissante qui calcul toutes les valeurs de MaxPrix(i) pour i compris entre 1 et n. A chaque fois une boucle évalue la meilleure valeur parmi toutes les possibilités, c'est à dire une découpe en j < i et le meilleur des choix sur une barre de taille (i-j).*
* *Si l'on enregistre la meilleur découpe j pour un niveau i il est possible de retrouver la solution un fois l'algorithme terminé.*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Longueur* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
| *Prix* | *0* | *2* | *5* | *6* | *11* | *13* | *15* | *17* | *21* |
| *Max* | *0* | *2* | *5* | *7* | *11* | *13* | *16* | *18* | *22* |
| *Coupe* | *0* | *1* | *2* | *1* | *4* | *1* | *2* | *1* | *4* |

*A partir d'une Coupe(n) il est alors possible de retrouver la solution via en remontant les différentes découpes par l'algorithme.*

*while n > 0 : print(coupe[n]) ; n=n-coupe(n)*

*Pour n=7, la découpe est composée d'une découpe en 1, 2 et 4.*

***Exercice 12 :*** *Labyrinthe.*

*Le problème est représenté sous la forme d’une matrice N\*M (de char ‘ ’) dans laquelle les murs sont représentés par une valeur particulière (exemple le caractère #). On souhaite partir de l'entrée (qui se trouve dans la case 0) trouver un chemin qui nous mène vers la sortie. Quatre directions sont autorisées : haut, bas, droite et gauche, l’entrée est toujours en haut à gauche, pour une sortie en bas à droite.*



* *Ecrire en pseudo code l’algorithme de recherche d’un chemin.*
* *Proposez une fonction permettant d’initialiser Le labyrinthe à partir d’un fichier de texte.*
* *Implémentez les algorithmes des fonctions* ***Existe****(****int*** *val),* ***Ajouter****(****int*** *val) et* ***Supprimer****() qui teste si une valeur* ***val*** *se trouve dans le tableau****Chemin****,**ajouter la valeur en tant que dernier élément dans le tableau et supprimer le dernier élément du tableau ;*
* *A partir de ces fonctions implémentez l’algorithme de recherche d’un chemin vers la sortie, et affichez la solution.*