TP4 : Programmation dynamique

Le but de ce TP est d'implanter des algorithmes de type « programmation dynamique » vus en cours. On fournit une trame de code pour les exercices dans les fichiers `TP4Exo1ChoixCours.py` *et* `TP4Exo2DistEd.py`.

Exercice 1. *Choix de cours valués*

Le but de cette partie est d'implémenter l'algorithme de choix de cours valués vu en cours, celui-ci étant basé sur de la programmation dynamique. Le fichier *TP4Exo1ChoixCours.py* contient une trame de programme.
Un cours est encodée par un tableau à trois entrée : sa date de début, sa date de fin, et sa valeur. Un ensemble de cours est un tableau de cours. Votre programme fonctionnera sur un exemple fixé, qui est affiché ci-dessou, ou un ensemble de cours généré de manière aléatoire.

1. Compléter la fonction `CoursAuHasard` pour tirer n cours au hasard. Les débuts de cours auront lieu entre le moment 1 et 80, leur durée sera choisie entre 1 et 20 unités de temps et leur valeur entre 1 et 10.
2. Compléter le code de la fonction `TriBullesCours` pour trier les cours par date de fin (c'est-à-dire par valeur `Cours[i][1]`) croissante. On utilisera un tri à bulles pour cela (que l'on pourra optimiser plus tard...)
   Implémenter la fonction `CalculPred` qui a pour but de remplir le tableau `Pred` afin de vérifier la condition suivante. Pour chaque cours `i`, le cours `Pred[i]` est le cours de fin la plus tardive possible, compatible avec le cours `i` et se terminant strictement avant `Cours[i][0]`. Si un tel cours n'existe pas, on gardera `Pred[i]=-1`.

   ▷ Noter la façon utilisée d'intialiser le tableau `Pred` avec la valeur -1.
3. Implémenter la fonction `ChoixMaxProgD` qui calcule la valeur d'une solution optimale suivant l'algorithme vu en cours. Pour rappel, la valeur `ValMax[i]` contient la valeur d'un choix de cours deux-à-deux compatibles de valeur totale maximale et finissant au pire à la date `Cours[i][1]`.
4. Compléter la fonction `ChoixMaxRec` qui renvoie la valeur d'un choix de cours optimal mais en ne faisant seulement que des appels récursifs (stratégie 'top down', sans utiliser la variable `ValMax` pour stocker les valeurs intermédiaires calculées). Pour simplifier l'écriture, on fera retourner 0 par la fonction lorsque le paramètre k vaut -1.
   Sur des exemples aléatoires, en faisant varier le nombre d'activités en entrée, comparer les temps d'exécution des fonctions `choixMaxProgD` et `choixMaxRec`. On notera que très rapidement l'approche 'programmation dynamique' est bien plus efficace que le 'calcul récursif' sur ce problème.

```
-------------------------------------------------------------------------------------------
Taper 1 pour un test sur l'exemple donné, 2 pour un ensemble de cours aléatoire: 1
Cours non triés:  [[76, 78, 10], [12, 17, 2], [13, 15, 1], [19, 28, 8], [12, 20, 7],
[44, 45, 9], [43, 45, 5], [1, 8, 3]]

Cours tries par dates de fin croissantes:  [[1, 8, 3], [13, 15, 1], [12, 17, 2],
[12, 20, 7], [19, 28, 8], [44, 45, 9], [43, 45, 5], [76, 78, 10]]

Calcul des prédécesseurs:
Pred du cours 0 : -1  / Pred du cours 1 : 0  / Pred du cours 2 : 0  / Pred du cours 3 : 0  /
Pred du cours 4 : 2  / Pred du cours 5 : 4  / Pred du cours 6 : 4  / Pred du cours 7 : 6  /

Valeur maximale d'un choix de cours en prog dyn:  32

Valeur maximale d'un choix de cours en récursif:  32

Calcul d'une solution de valeur maximale d'un choix de cours en prog dyn:
Valeur maximale:  32
Solution correspondante : [[1, 8, 3], [12, 17, 2], [19, 28, 8], [44, 45, 9], [76, 78, 10]]
-------------------------------------------------------------------------------------------
```

FIGURE 1 – Résultats à obtenir sur l'exemple fixé.

5. Pour calculer explicitement une solution de valeur optimale, on utilise un tableau `Sol` rempli par l'appel de la fonction `int ChoixMaxProgDSol`. La variable `Sol[i]` contient 1 si, et seulement, si le cours `i` est le dernier cours d'une sous-solution optimale au problème restreint aux cours se terminant au pire à date `Cours[i][1]`. Il faut reprendre le code de la fonction `ChoixMaxProgDSol` et la modifier afin de remplir le tableau `Sol`.
   Enfin, compléter la fonction `CalculSolProgDyn` qui remplit le tableau `CoursChoisis` avec un ensemble de cours formant une solution de valeur maximale. On aura besoin pour cela des tableaux pré-calculés `Pred` et `Sol`. On pourra trouver le cours `i` avec plus grande date de fin et vérifiant `Sol[i]=1`, puis le cours précédent dans une solution optimale et ainsi de suite.

   ▷ Il sera pratique de remplir le tableau `CoursChoisis` par le début (et non par la fin, comme le fait `.append()`). Pour cela, on peut utiliser `CoursChoisis.insert(0,Cours[i])`.

Exercice 2. *Distance d'édition*

Le but de cet exercice est d'implanter le calcul de la distance d'édition vu en cours, ainsi que l'alignement de deux mots. La trame de code correspondant à l'exercice est `TP4Exo2DistEd.py`. Un exemple d'exécution du programme complété est donné ci-dessous.

```
-----------------------------------------------------------------------------------------------
Entrer S1 la première chaine de caractères: algorithme
Entrer S2 la seconde chaine de caractères: agorrytne

Les deux chaines S1 et S2 sont à distance:  5

Tableau des distances partielles:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 1, 1, 1, 2, 3, 4, 5, 6, 7, 8]
[3, 2, 2, 2, 1, 2, 3, 4, 5, 6, 7]
[4, 3, 3, 3, 2, 1, 2, 3, 4, 5, 6]
[5, 4, 4, 4, 3, 2, 2, 3, 4, 5, 6]
[6, 5, 5, 5, 4, 3, 3, 3, 4, 5, 6]
[7, 6, 6, 6, 5, 4, 4, 3, 4, 5, 6]
[8, 7, 7, 7, 6, 5, 5, 4, 4, 5, 6]
[9, 8, 8, 8, 7, 6, 6, 5, 5, 5, 5]

Alignement de S1 et S2:
algorithme
a-gorrytne
-----------------------------------------------------------------------------------------------
```

1. Compléter la fonction `Init(E,n1,n2)` qui permet d'initialiser le tableau E de telle sorte que $E[0][j] = j$ pour $j = 0,\dots,n_1$, $E[i][0] = i$ pour $i = 0,\dots,n_2$ et $E[i][j] = 0$ sinon. **Attention !** Par rapport au cours, les indices du tableau E sont décalés : pas d'indice -1, c'est plus simple à écrire dans le code, mais il faut gérer ce décalage...

   ▷ Noter comment le tableau E est initialisé à 0 et affiché dans le programme principal.

2. Compléter la fonction `DistanceEdition(S1,S2,E)` qui calcule la distance entre les chaines de caractères $S1$ et $S2$ en remplissant le tableau E selon l'algorithme donné en cours.
3. Compléter la fonction `Alignement(S1,S2,E)` qui calcule à partir du tableau E un alignement des deux chaînes `s1` et `s2` : la fonction doit modifier `s1` et `s2` en insérant le caractère - (*underscore*) dans `s2` pour marquer une suppression, ou dans `s1` pour marquer une insertion. Décommenter les `while` dans le code pour effectuer votre traitement.

   ▷ Pour insérer le caractère avant la position `i1` dans la chaîne `S1` on pourra utiliser l'instruction `S1=S1[:i1]+"-"+S1[i1:]`, ou plus simplement si $i_1 = 0$ l'instruction `S1="-"+S1`.

Exercice 3. **Exercice bonus :** *Voyageur de commerce*

Implémenter l'algorithme vu en cours pour résoudre le problème du voyageur de commerce en temps exponentiel (simple). Aucune trame de code n'est fourni pour cet exercice.

Implémenter l'algorithme consistant à générer toutes les permutations de l'ensemble de villes et à choisir la tournée correspondante de longueur minimale. Comparer le temps de calcul de vos deux programmes en faisant varier le nombre de villes considérées.

▷ Pour générer tous les sous-ensembles de taille k d'un ensemble S, on pourra faire appel à la fonction `combinations(S,k)` du paquet `intertools`.

▷ Pour générer toutes les permutations de l'ensemble $\{0,\dots,n-1\}$, on pourra faire appel à la fonction `permutations(range(n))` du même paquet `intertools`.