IFT 2015
TP 2
(25 points)
21 février 2014

Les transporteurs calculent le prix pour envoyer un colis en fonction de son poids, mais aussi de son volume. Pour une entreprise qui expédie plusieurs objets à un même destinataire, il est donc important de minimiser le volume de la boîte à utiliser pour envoyer ces objets. Le problème est ici de trouver, pour une liste d'objets donnés, de dimensions différentes, la configuration spatiale qui permette d'utiliser la plus petite boîte, possible, c'est-à-dire de minimiser l'espace perdu dans la boîte. Afin de simplifier un peu le problème, nous allons le réduire à deux dimensions, mais une extrapolation en trois dimensions se fait directement. Pour solutionner ce problème, vous allez implémenter une structure de données que nous appellerons *plan de découpage*.

Un plan de découpage divise un rectangle dont les côtés sont horizontaux et verticaux en utilisant des coupes horizontales et verticales (voir la figure 8.25 et le numéro P-8.67 du livre, aux pages 359-360). Un plan de découpage peut être représenté par un arbre binaire, appelé *arbre de coupe*, dont les noeuds internes représentent les coupes, et les noeuds externes (ou feuilles) représentent les rectangles de base dans lesquels le plan est décomposé par les coupures. Le problème de compactage pour un plan de découpage est défini comme suit. Supposons que chaque rectangle de base d'un plan de découpage est assigné une largeur minimale w et une hauteur minimale h, correspondant aux dimensions de l'objet à y insérer. Le problème de compactage est de trouver la plus petite largeur et la plus petite hauteur possible de chaque rectangle de la du plan de découpage qui est compatible avec les dimensions minimales des rectangles de base. Plus précisément, ce problème nécessite l'affectation de valeurs $\mathsf{h}(p)$ et $\mathsf{w}(p)$ pour chaque position (ou noeud) p de l'arbre de coupe de telle sorte que :

$$\mathsf{w}(p) = \begin{cases} w, & \text{si } p \text{ est une feuille de largeur minimale } w \\ \max(\mathsf{w}(l), \mathsf{w}(r)), & \text{si } p \text{ est un noeud de découpage horizontal avec enfants } l \text{ et } r \\ \mathsf{w}(l) + \mathsf{w}(r), & \text{si } p \text{ est un noeud de découpage vertical avec enfants } l \text{ et } r \end{cases}$$

$$\mathsf{h}(p) = \begin{cases} h, & \text{si } p \text{ est une feuille de hauteur minimale } h \\ \mathsf{h}(l) + \mathsf{h}(r), & \text{si } p \text{ est un noeud de découpage horizontal avec enfants } l \text{ et } r \\ \max(\mathsf{h}(l), \mathsf{h}(r)), & \text{si } p \text{ est un noeud de découpage vertical avec enfants } l \text{ et } r. \end{cases}$$

Vous devez donc créer une structure d'arbre binaire qui vous permette entre autres de créer un plan de découpage, de découper un rectangle horizontalement ou verticalement, d'assigner une largeur et une hauteur minimales à un rectangle de base, de compacter un arbre, de produire une string décrivant l'arbre de coupe (voir plus loin). Vous êtes libres de créer toutes les fonctions ainsi que toutes autres classes dont vous pensez avoir besoin pour résoudre ce problème. Notez que deux noeuds connectés dans l'arbre ne peuvent pas représenter deux découpages successifs dans le même sens, sans quoi le problème s'en trouve complètement différent.

Votre fonction `main()` reçoit en entrée une liste de tupples `("nom_objet",` w, h`)` et doit retourner **une liste contenant au moins un arbre de coupe** (sous forme de string) associé à un plan de découpage qui minimise l'aire de la boîte (il peut y en avoir plusieurs). Ces arbres seront représentés sous forme parenthésée en préfixe. Par exemple, l'arbre de la page 360 serait représenté par `'(-,(|,A,(-,B,(|,C,D))),(|,E,F))'` où $-$ et $|$ représentent respectivement des découpages horizontaux et verticaux, et les lettres les noms des objets.

Votre algorithme doit donc, parmi tous les découpages possibles, c'est-à-dire tous les arbres de coupe possibles, déterminer celui ou ceux qui produisent la plus petite boîte (aire). Notez qu'il existe $n!$ ordres (permutations) possibles des feuilles d'un arbre (incluant les arbres isomorphiques).

De plus, chaque objet (feuille) peut être orienté de 2 façons (rotation de 90 degrés), ce qui peut résulter en une meilleure compaction. Il existe donc 2^n combinaisons possibles de rotations des n objets. Donc, pour chaque topologie d'arbre possible, il existe 2^n arbres différents, selon la rotation des objets. Votre algorithme doit retourner, sous forme d'une liste de strings, tous les arbres qui produisent une boîte dont l'aire est minimale. Chacune de ces string décrivant un arbre doit être sous la forme

`0100:(-,(|,(-,A,B),C),D):12`

où `0100` représente quels objets sont tournés (1 pour oui, 0 pour non) par rapport aux dimensions originales reçues. Dans cet arbre, seul l'objet `B` a été tourné. `(-,(|,(-,A,B),C),D)` représente l'arbre, et `12` est l'aire obtenu par arbre une fois compacté.

À titre d'exemple, si l'on doit placer les objets `("A", 2, 7)` et `("B", 5, 3)`, les arbres possibles (et leurs aires minimales) sont :

```
11:(-,A,B):49
11:(|,A,B):50
11:(-,B,A):49
11:(|,B,A):50
10:(-,A,B):35
10:(|,A,B):36
10:(-,B,A):35
10:(|,B,A):36
01:(-,A,B):36
01:(|,A,B):35
01:(-,B,A):36
01:(|,B,A):35
00:(-,A,B):50
00:(|,A,B):49
00:(-,B,A):50
00:(|,B,A):49,
```

et donc la plus petite aire possible est de 35, obtenue par 4 découpages différents. Votre algorithme devrait donc retourner **au moins un élément de la liste** :

[`"10:(-,A,B):35"`, `"10:(-,B,A):35"`, `"01:(|,A,B):35"`, `"01:(|,B,A):35"`].

Barème de correction (25 points)

Votre structure sera testée avec 5 listes d'objets (5 listes $\times$ 5 points $=$ 25 points), différents de ceux présentement fournis dans le unit test. Pour chacune des 5 listes, voici le barème :

5 points si vous retournez au moins un arbre de coupe minimal et aucun arbre non optimal ;
3 points si vous retournez au moins un arbre de coupe minimal et aucun arbre non optimal, mais que le format de retour n'est pas compatible avec le unit test fourni ;
1 point si vous retournez au moins un arbre de coupe minimal et mais aussi un ou des arbres non optimaux ;
0 point si vous retournez au moins un arbre qui n'est pas minimal, ou si vous ne retournez aucun arbre minimal.

En principe, si votre structure fonctionne correctement, aucune erreur ne devrait être obtenue lors de l'exécution du unit test.

Remise

Remettez dans Studium tous vos fichiers `.py` (exécutables sous Python 3), en vous assurant de bien identifier les coéquipiers au début du fichier `TP2.py`, sous forme d'un dossier compressé.

Bon boulot !