Esiee-Paris - cours d'algorithmique 5I-IN1

~~8 mars 2022~~ 24 mars 2022 - Projet de l'unité
Optimisations locales versus optimisations globales

On demande de comparer les valeurs de stratégies d'optimisation locale, souvent appelées stratégies gloutonnes, avec les valeurs optimales calculées par programmation dynamique.

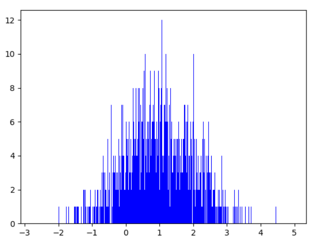
Exemples :

- dans le cas du calcul du chemin du petit robot la stratégie gloutonne consiste, en toute case, à se diriger dans la direction dont le coût en cette case est minimum ;
- dans le cas d'un sac de valeur maximum, la stratégie "gloutonne par valeur" consiste à mettre dans le sac les objets dans l'ordre des valeurs décroissantes, et la stratégie "gloutonne par densité de valeurs" consiste à les mettre dans le sac par densités "valeur/taille" décroissantes ;
- dans le cas de la répartition d'un stock sur un ensemble d'entrepôts, l'affectation gloutonne alloue une unité de stock à l'un des entrepôts pour lesquels l'augmentation de gain consécutif à cette livraison est maximum ;
- ~~de même pour la répartition optimale d'un temps de travail sur un ensemble d'unités ;~~
- dans le cas d'un chemin de somme maximum (`https://projecteuler.net/archives`, problèmes 18 et 67) la stratégie gloutonne consiste à se diriger vers le descendant gauche si sa valeur est supérieure à celle du descendant droit, et réciproquement : voir fichier cheminSommeMax.pdf ci-après.

Pour comparer une stratégie gloutonne et la stratégie optimale, on génère un problème avec des valeurs choisies au hasard et on le résout avec chacune des deux stratégies. Puis on calcule la distance relative entre les deux solutions : (valeur optimale - valeur gloutonne)/(valeur optimale). On répète cette expérience pour un grand nombre de problèmes (on parle de runs), puis on affiche l'histogramme de la distribution des distances relatives, et la moyenne et l'écart-type des distances relatives.

Ci-dessous un exemple de construction et d'affichage d'un histogramme en Python.

```
$ python
>>> import matplotlib.pyplot as plt
>>>
>>> import random
>>> N = 1000
>>>
>>> Deltas = [random.gauss(1,1) for i in range(N)]
>>> num_bins = len(Deltas)//2
>>> h = plt.hist(Deltas, num_bins, facecolor='blue', alph
>>> plt.show()
>>>
>>>
```



Comparer les stratégies gloutonnes et optimales des problèmes des exemples ci-dessus, et sur un ou deux autres problèmes que vous choisirez vous-même. Ce travail est à faire en binôme.

rene.natowicz@esiee.fr 24/03/2022

Chemin de somme maximum.

Il s'agit des problèmes 18 et 67 du site ProjectEuler.com

https://projecteuler.net/archives

Représentation du triangle dans un tableau T[0:n].

Considérons un triangle à m niveaux, numérotés de 0 à m-1. Combien y a-t-il de valeurs
dans ce triangle ?

Au niveau 0 : 1 valeur
au niveau 1 : 2 valeurs
au niveau 2 : 3 valeurs
etc.
au niveau m-1 : m valeurs

Il y a m niveaux et n = (1 + 2 + ... + m) = (m x (m + 1)) / 2 valeurs.

Soit T[0:n] un tableau d'entiers tel que n = (m x (m + 1)) / 2.
Si nous avons une fonction g(i) qui retourne l'indice g_i = g(i) du descendant
gauche de i alors l'indice du descendant droit est d_i = g_i +1.
Connaissant g(i) et d(i) pour tout indice i, nous pouvons regarder le tableau T[0:n]
comme un triangle.

Exemple : n = (3 x 4) / 2 = 12/2 = 6, T[0:6] = [0,10,20,30,40,50].

```
       00         niveau 0, indice  0, intervalle d'indices [0:1]
     10  20       niveau 1, indices 1, 2, intervalle d'indices [1:3]
   30  40  50     niveau 2, indices 3, 4, 5, intervalle d'indices [3:6]
   ----------
 60  70  80   90
```

g(0) = 1, d(0) = 2
g(1) = 3, d(1) = 4
g(2) = 4, d(2) = 5
g(3) = 6, d(3) = 7 indices hors de [0:n], donc 3 est une feuille
g(4) = 7, d(4) = 8 indices hors de [0:n], donc 4 est une feuille
g(5) = 8, d(5) = 9 indices hors de [0:n], donc 5 est une feuille

Comment calculer l'indice g(i) du descendant situé à gauche de l'indice i ?
En remarquant que la position de i dans son niveau est la position de g(i) dans son niveau
Exemples :
-- i = 0 position p = 0 niveau l = 0, g(0) = 1 position p = 0 niveau l+1 = 1,
-- i = 1 position p = 0 niveau l = 1, g(1) = 3 position p = 0 niveau l+1 = 2,
-- i = 2 position p = 1 niveau l = 1, g(2) = 4 position p = 1 niveau l+1 = 2,
-- i = 5 position p = 2 niveau l = 2, g(5) = 8 position p = 2 niveau l+1 = 3.

D'où le calcul de g(i).
1) trouver son niveau dans l'arbre.
Plaçons nous au niveau l.
a) Il y a 1+2+...+l-1 = (l-1)*l/2 valeurs au dessus de ce niveau.
b) les l+1 valeurs du niveau l sont dans l'intervalle [(l-1)*l/2 : (l-1)*l/2 + l = l * (l+1)/2]
La fonction suivante retourne le niveau dans lequel est situé l'indice i :

```
def niveau(i):
    l = 0;
    while i < (l-1)*l/2 :
        l = l+1
    return l
```

2) Ayant le niveau l dans lequel se trouve l'indice i, nous connaissons sa position p dans
ce niveau : p = i - (l-1)*l/2
3) Ayant le niveau l et la position p de l'indice i dans son niveau l, nous avons l'indice
de son descendant gauche : c'est l'indice situé à la position p du niveau l+1.
C'est donc l'indice l * (l+1)/2 + p.
La fonction g(i) en découle :

```
def g(i) :
    l = niveau(i)
    p = i - (l-1)*l/2
    return l * (l+1)/2 + p
```

Ayant la fonction la valeur g_i = g(i) nous connaissons l'indice du descendant droit de
l'indice i : d_i = g_i + 1. Nous pouvons donc regarder le tableau T[0:n] comme un triangle.

Programmation dynamique pour le calcul de la valeur du chemin de somme maximum.

Ecrire une fonction calculerM(int[] T, int i) qui prend en entrée le triangle T[0:n]
et retourne un tableau M[0:n] de terme général M[i] = m(i) = la valeur d'un chemin de
somme maximum qui commence à l'indice i. La valeur M[0] est la valeur d'un chemin de
somme maximum du triangle T.
Pour cette fonction calculerM(int[] T, int i) il vous faut l'équation de récurrence
des valeurs m(i).
A) "Supposons le problème résolu" : donner l'expression de m(0).
B) Généraliser cette expression afin d'obtenir l'équation de récurrence des valeurs m(i).
1) Base : si i est une feuille, m(i) = ...
2) Cas général : i n'est pas un feuille, m(i) = ...

La fonction ci-dessous retourne la valeur de vérité de la proposition "l'indice i est une feuille du triangle de taille n" :

```
def feuille(i,n) :
    return g(i) >= n
```

Calcul d'un chemin glouton.

Un chemin glouton est une maximisation locale : en chaque point du chemin on se dirige à gauche si la valeur située à gauche est plus grande que celle située à droite, et réciproquement.

Validation statistique.

-- soit Mmax = le nombre de niveaux maximum. Par exemple : Mmax = 1000
-- soit Nruns le nombre de "runs" de la validation statistique. Par exemple : Nruns = 5000
-- soit Vmax la valeur maximum d'un triangle. Par exemple : Vmax = 100

```
1) Définir le tableau D[0:Nruns] qui contiendra pour chaque run, la distance
relative entre la valeur du chemin de somme maximum et la valeur du chemin glouton.
2) Pour chaque r dans [0:Nruns] :
    choisir m au hasard dans l'intervalle [1:Mmax+1]
    soit n = m * (n+1)/2
    soit T[0:n] un tableau d'entiers dont les valeurs sont au hasard dans [0:Vmax+1]
    soit vmax = m(0) la valeur d'un chemin de somme max. et soit g celle du chemin glouton.
    La distance relative est D[r] = (vmax - g)/vmax
retouner D
3) afficher l'histogramme de D, la moyenne, la médiane et l'écart-type des valeurs de D.
```