Éléments d'Algorithmique

Séance 1: Introduction

Jérémie Jakubowicz

Un exemple pour commencer

Problème

On se donne deux vecteurs $v = (v_1, \ldots, v_n)$ et $w = (w_1, \ldots, w_n)$. On suppose qu'on peut permuter les coordonnées de ces deux vecteurs à l'envi. Ainsi si σ désigne une permutation de $\{1, \ldots, n\}$, on désigne par v^σ le vecteur $v = (v_{\sigma(1)}, \ldots, v_{\sigma(n)})$.
Objectif : Permuter les coordonnées de v et w de sorte à minimiser le produit scalaire $\langle v^\sigma, w^{\sigma'} \rangle$. Afficher ce produit scalaire minimum.

Exemple

Pour $v = (1, 2)$, $w = (4, 3)$. Le produit scalaire minimum vaut 10.

Format des entrées/sorties

Exemple :

Entrées

2 ← Nombre de cas
3 ← Dimension des deux vecteurs cas 1
1 3 -5 ← Vecteur 1 cas 1
-2 4 1 ← Vecteur 2 cas 1
5 ← Dimension des deux vecteurs cas 2
1 2 3 4 5 ← Vecteur 1 cas 2
1 0 1 0 1 ← Vecteur 2 cas 2

Sorties

Cas 1 : -25
Cas 2 : 6

But du jeu

Ecrire un programme Python qui transforme les entrées en sorties.

entrees.txt

```
2
3
1 3 -5
-2 4 1
5
1 2 3 4 5
1 0 1 0 1
```

sorties.txt

```
Cas 1 : -25
Cas 2 : 6
```

prog.py

```
def solve(f):
    d = int(f.readline())
    v = map(int,f.readline().split())
    w = map(int,f.readline().split())
    v.sort()
    w.sort(reverse=True)
    return sum([x*y for x,y in zip(v,w)])

N = int(sys.stdin.readline())
for i in range(N):
    print "Cas %d : %d" % (i+1, solve(f))
```

```
$ python prog.py < entrees.txt > sorties.txt
```

Règles du jeu

1. Essayer de réfléchir aux problèmes “en direct”.

Règles du jeu

1. Essayer de réfléchir aux problèmes “en direct”. En tous cas, on réservera du temps pour ça en cours.

Règles du jeu

1. Essayer de réfléchir aux problèmes “en direct”. En tous cas, on réservera du temps pour ça en cours.
2. Si possible, assister au cours avec un ordinateur et un interpréteur Python en état de marche.

Différents aspects de l'algorithmique

1. Design
2. Preuve de validité
3. Preuve de terminaison
4. Analyse de complexité

Ici, on s'intéressera surtout au design et à la complexité (c'est-à-dire que dans la majorité des cas, on admettra que nos algorithmes sont corrects et terminent).

Retour à l'exemple introductif

1. **Design**

Algorithm 1 Produit Scalaire Minimum

Input: n, $v = (v_1, \ldots, v_n)$, $w = (w_1, \ldots, w_n)$.

Ouput: $\min_{(\sigma,\sigma') \in \mathfrak{S}_n^2} \sum_{i=1}^{n} v_{\sigma(i)} w_{\sigma'(i)}$

```
1:  mS ← +∞
2:  for σ ∈ 𝔖_n do
3:      S ← Σ_{i=1}^{n} v_{σ(i)} w_i
4:      if S < mS then
5:          mS ← S
6:      else
7:          pass
8:      end if
9:  end for
10: return mS
```

Retour à l'exemple introductif

2. **Validité**

Résultat

$$\min_{(\sigma,\sigma')\in\mathfrak{S}_n^2} \sum_{i=1}^{n} v_{\sigma(i)} w_{\sigma'(i)} = \min_{\sigma\in\mathfrak{S}_n} \sum_{i=1}^{n} v_{\sigma(i)} w_i$$

Démonstration

Par "changement de variable" $j = \sigma'(i)$, on a, pour tout couple σ, σ' :

$$\sum_{i=1}^{n} v_{\sigma(i)} w_{\sigma'(i)} = \sum_{j=1}^{n} v_{\sigma(\sigma'^{-1}(j))} w_j$$

D'autre part, quand le couple (σ, σ') décrit $\mathfrak{S}_n^2$, la composition $\sigma \circ \sigma'^{-1}$ décrit $\mathfrak{S}_n$. □

Retour à l'exemple introductif

3. **Terminaison**

Le fait que l'algorithme précédemment décrit termine provient du fait qu'il n'y a pas de boucle potentiellement infinie.

Retour à l'exemple introductif

4. **Complexité**

La boucle `for` comporte $n!$ itérations. Chaque itération comporte $O(n)$ opérations (n produits, $n-1$ sommes, une comparaison, une affectation). La complexité de l'algorithme est donc en $O(n \cdot n!) = O((n+1)!)$.

n	$n!$
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880

Même avec un supercalculateur, on ne peut pas espérer aller au delà de $n = 20$.

Une alternative

1. **Design**

Algorithm 2 Produit Scalaire Minimum (deuxième version)

Input: n, $v = (v_1, \ldots, v_n)$, $w = (w_1, \ldots, w_n)$.

Ouput: $\min_{(\sigma,\sigma') \in \mathfrak{S}_n^2} \sum_{i=1}^{n} v_{\sigma(i)} w_{\sigma'(i)}$

1: $v^{\text{sorted}} \leftarrow \texttt{sort}(v)$

2: $w^{\text{sorted}} \leftarrow \texttt{sort}(w)$

3: **return** $\sum_{i=1}^{n} v_i^{\text{sorted}} w_{n-i+1}^{\text{sorted}}$

Une alternative

2. **Validité**

Une identité

Si $a \leq b$ et $c \leq d$, alors $ad + bc \leq ac + bd$.

En effet, c'est ce qu'on obtient quand on développe :
$(b - a)(d - c) \geq 0$. □

On peut sans perte de généralité supposer que v est ordonné par ordre croissant : $v_1 \leq \cdots \leq v_n$. Soit, alors $w = (w_1, \ldots, w_n)$ quelconque. On intervertit alors w_1 avec le plus grand élément $\max w_i$ (si c'est w_1) on ne fait rien. On note $w_2 = (w_{(n)}, \ldots)$ le vecteur obtenu. On a, en vertu de l'identité précédente : $\langle v, w_2 \rangle \leq \langle v, w \rangle$. On intervertit alors $w_{2,2}$ avec $w^{(n-1)}$, ce qui diminue encore le produit scalaire, etc. □

Une alternative

3. **Terminaison**

En supposant que `sort` termine, la terminaison est claire.

Une alternative

4. **Complexité**

En admettant que l'algorithme de tri utilisé lors de l'appel à `sort` est de complexité $O(n \log n)$, cette alternative possède une complexité $O(n \log n)$. On peut ainsi résoudre le problème pour des ordres de grandeur $n = 10^6$ sur un ordinateur portable banal.

Une alternative

4. **Complexité**

En admettant que l'algorithme de tri utilisé lors de l'appel à `sort` est de complexité $O(n \log n)$, cette alternative possède une complexité $O(n \log n)$. On peut ainsi résoudre le problème pour des ordres de grandeur $n = 10^6$ sur un ordinateur portable banal.

Morale de cette histoire : Plusieurs algorithmes peuvent résoudre un même problème avec plus ou moins de succès.

Exercice

- Implémenter les deux algorithmes mentionnés en python.
- Comparer en pratique.
- Commenter : est-ce que l'étude de complexité est cohérente avec les résultats observés en pratique ?

Solution

Squelette de code commun :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys

def solve(f):
    pass

def main(f):
    N = int(f.readline())
    for i in range(N):
        print "Cas %d : %d" % (i+1, solve(f))

if __name__=='__main__':
    f = sys.stdin
    main(f)
```

Solution I

Algorithme I :

```
def solve(f):
    import itertools
    d = int(f.readline())
    v = map(int,f.readline().split())
    w = map(int,f.readline().split())
    n = len(v)
    mS = sum([abs(x) for x in v])*sum([abs(x) for x in w])
    for sigma in itertools.permutations(range(n)):
        S = 0
        for i in range(n):
            S += v[sigma[i]] * w[i]
        if S < mS:
            mS = S
    return mS
```

Solution II

```
def solve(f):
    d = int(f.readline())
    v = map(int,f.readline().split())
    w = map(int,f.readline().split())
    v.sort()
    w.sort(reverse=True)
    return sum([x*y for x,y in zip(v,w)])
```

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".
2. Il y a une raison à ça : un mathématicien / statisticien est plus utile dans la plupart des industries s'il sait (bien) programmer.

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".
2. Il y a une raison à ça : un mathématicien / statisticien est plus utile dans la plupart des industries s'il sait (bien) programmer.
3. Bien programmer implique différents savoirs et qualités. Par exemple, ça peut passer par :

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".
2. Il y a une raison à ça : un mathématicien / statisticien est plus utile dans la plupart des industries s'il sait (bien) programmer.
3. Bien programmer implique différents savoirs et qualités. Par exemple, ça peut passer par :
   - 3.1 Connaître les méthodes agiles / scrum, etc. pour la gestion de projet (informatique)

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".
2. Il y a une raison à ça : un mathématicien / statisticien est plus utile dans la plupart des industries s'il sait (bien) programmer.
3. Bien programmer implique différents savoirs et qualités. Par exemple, ça peut passer par :
   - 3.1 Connaître les méthodes agiles / scrum, etc. pour la gestion de projet (informatique)
   - 3.2 Maîtriser les design patterns, UML, les éléments d'architecture logicielle

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".
2. Il y a une raison à ça : un mathématicien / statisticien est plus utile dans la plupart des industries s'il sait (bien) programmer.
3. Bien programmer implique différents savoirs et qualités. Par exemple, ça peut passer par :
   - 3.1 Connaître les méthodes agiles / scrum, etc. pour la gestion de projet (informatique)
   - 3.2 Maîtriser les design patterns, UML, les éléments d'architecture logicielle
   - 3.3 Parler couramment un langage informatique : Python / Java / Javascript / C++ / etc.

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

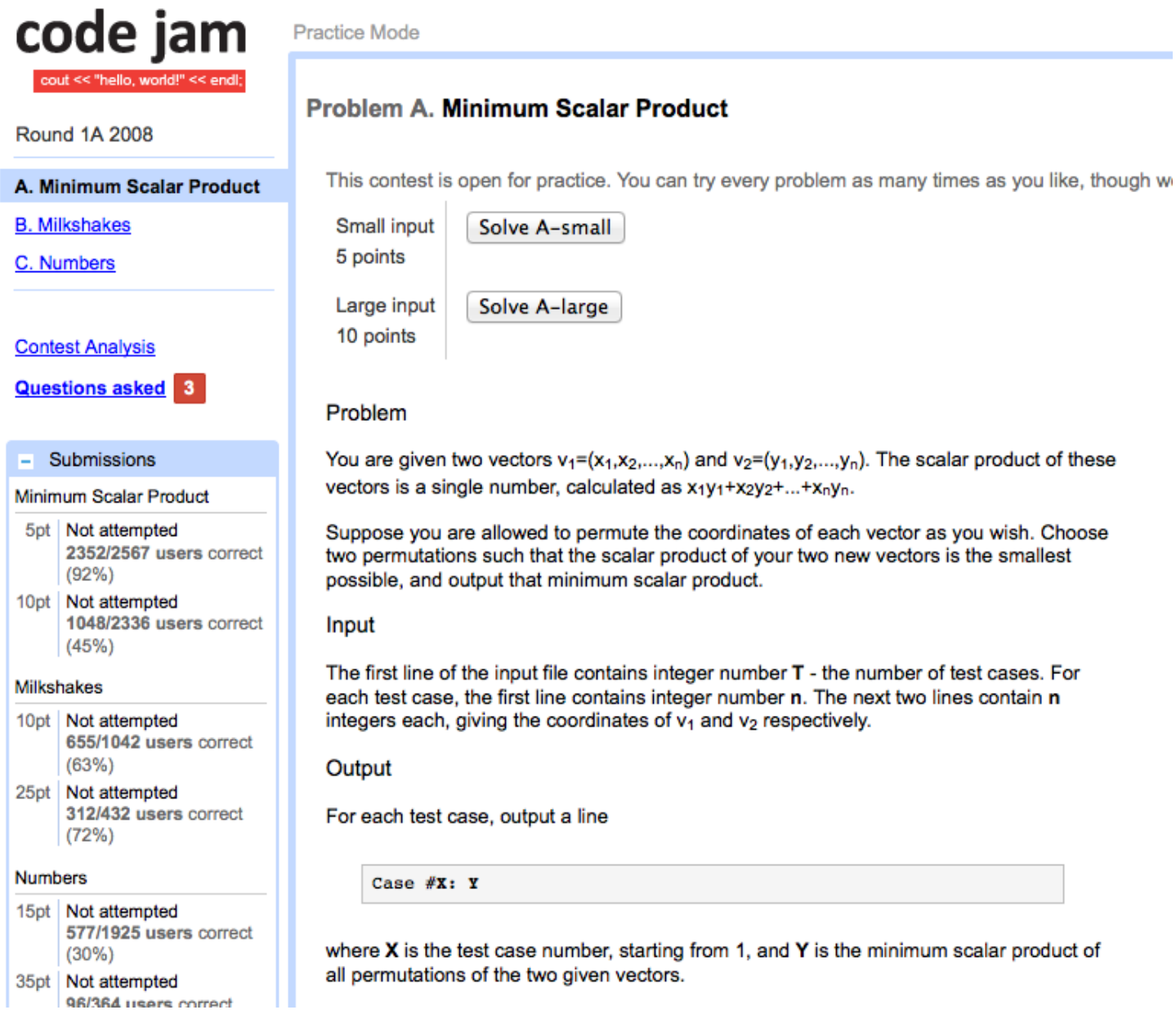
Motivations pour continuer à suivre le cours

La question qui vous préoccupe (peut-être) : Quel intérêt pour un élève de l'ENSAE ?

1. Un certain nombre d'entreprises font passer des tests d'algorithmique avant l'embauche sur un poste de "Datascientist / Software Engineer".
2. Il y a une raison à ça : un mathématicien / statisticien est plus utile dans la plupart des industries s'il sait (bien) programmer.
3. Bien programmer implique différents savoirs et qualités. Par exemple, ça peut passer par :
   - 3.1 Connaître les méthodes agiles / scrum, etc. pour la gestion de projet (informatique)
   - 3.2 Maîtriser les design patterns, UML, les éléments d'architecture logicielle
   - 3.3 Parler couramment un langage informatique : Python / Java / Javascript / C++ / etc.
   - 3.4 Comprendre l'algorithmique et les structures de données

Le dernier point peut-être "rentable" pour un élève à l'ENSAE.

Ca vous dit quelque chose ?



code jam

cout << "hello, world!" << endl;

Practice Mode

Round 1A 2008

A. Minimum Scalar Product

B. Milkshakes

C. Numbers

Contest Analysis

Questions asked 3

Submissions

Minimum Scalar Product

5pt | Not attempted | 2352/2567 users correct (92%)

10pt | Not attempted | 1048/2336 users correct (45%)

Milkshakes

10pt | Not attempted | 655/1042 users correct (63%)

25pt | Not attempted | 312/432 users correct (72%)

Numbers

15pt | Not attempted | 577/1925 users correct (30%)

35pt | Not attempted | 96/364 users correct

Problem A. Minimum Scalar Product

This contest is open for practice. You can try every problem as many times as you like, though w

Small input
5 points
Solve A-small

Large input
10 points
Solve A-large

Problem

You are given two vectors $v_1=(x_1,x_2,...,x_n)$ and $v_2=(y_1,y_2,...,y_n)$. The scalar product of these vectors is a single number, calculated as $x_1y_1+x_2y_2+...+x_ny_n$.

Suppose you are allowed to permute the coordinates of each vector as you wish. Choose two permutations such that the scalar product of your two new vectors is the smallest possible, and output that minimum scalar product.

Input

The first line of the input file contains integer number **T** - the number of test cases. For each test case, the first line contains integer number **n**. The next two lines contain **n** integers each, giving the coordinates of v_1 and v_2 respectively.

Output

For each test case, output a line

```
Case #X: Y
```

where **X** is the test case number, starting from 1, and **Y** is the minimum scalar product of all permutations of the two given vectors.

Plan pour la suite

1. Je vous soumets un problème
2. Vous y réfléchissez directement
3. Vous essayez d'implémenter une solution sur votre ordinateur portable / tablette / téléphone
4. On échange
5. Je vous propose une solution
6. On échange en essayant de comprendre ce qui peut être utile pour résoudre d'autres problèmes.

Problème 2 : UVa 10020 - Couverture Minimale

Le problème

Couvrir le segment $[0, M]$ avec le nombre minimum d'intervalles de la forme $[a_i, b_i]$.

Les entrées

La première ligne correspond au nombre de cas T. Chaque cas commence par un entier $0 \leq M \leq 5\,000$. Ensuite chaque ligne est composée de deux entiers a_i b_i et la dernière ligne du cas est marquée par $0, 0$; suivie d'une ligne vide.

Les sorties

Si $[0, M]$ n'est pas couvrable, la sortie doit être "0" ? Sinon la sortie commence par une ligne mentionnant le nombre minimal d'intervalles dont on a besoin pour couvrir $[0, M]$, suivi de lignes avec les intervalles en question sous la forme a_i b_i et finit sur une ligne vide.

Exemple d'entrée-sortie

Exemple d'entrée

```
2

1
-1 0
-5 -3
2 5
0 0

1
-1 0
0 1
0 0
```

Exemple de sortie

```
0

1
0 1
```

Des idées ?