Éléments d'Algorithmique

Séance 2: Algorithmes Gloutons, Programmation Dynamique

Jérémie Jakubowicz

Récapitulatif de la précédente séance

La dernière fois, nous avions traité le problème :

$$\min_{\sigma,\tau} \sum_i v_{\sigma(i)} w_{\tau(i)}$$

Sur ce problème on avait proposé deux algorithmes pour lesquels nous avions montré :

- Qu'ils terminaient
- Qu'ils donnaient la bonne réponse
- Qu'ils n'avaient pas la même *complexité*
  - Le premier avait une complexité $O(n \cdot n!)$, où n est le nombre de composantes de v et w
  - Le second avait une complexité $O(n \log n)$

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 ?

L'approche gloutonne

1. On commence à essayer de couvrir 0 avec un intervalle $[a, b]$ tel que b soit le plus grand possible.
2. Puis parmi les intervalles $[c, d]$ qui couvrent b, on en choisit un tel que d soit maximal
3. *etc.*

L'approche gloutonne

1. On commence à essayer de couvrir 0 avec un intervalle $[a, b]$ tel que b soit le plus grand possible.
2. Puis parmi les intervalles $[c, d]$ qui couvrent b, on en choisit un tel que d soit maximal
3. *etc.*

Si à l'issue de cette procédure on a couvert M, alors on affiche les intervalles qu'on a utilisés et on termine par "0 0" suivi d'une ligne vide. Sinon on affiche 0 suivi d'une ligne vide.

L'algorithme

Algorithm 1 Couverture Minimale par Intervalles

Input: M, $[a_i, b_i]$, $1 \leq i \leq n$

Ouput: `seglist`

1: `sort`$([a_i, b_i], 1 \leq i \leq n)$ en classant suivant les a_i croissants
2: `next_l`$\leftarrow 0$, `next_r`$\leftarrow -1$, `pending_r`$\leftarrow 0$, `seglist`$\leftarrow []$
3: **for** $i \leftarrow 1 : n$ **do**
4: **if** $a_i >$`pending_r` **then**
5: `seglist.append((next_l,next_r))`
6: `pending_r`$\leftarrow$`next_r` , `next_l`$\leftarrow a_i$, `next_r`$\leftarrow b_i$
7: **else**
8: **if** $b_i >$`next_r` **then**
9: `next_l`$\leftarrow a_i$, `next_r`$\leftarrow b_i$
10: **if** `pending_r`$\geq M$ `or` `next_l`$>$`pending_r` **then**
11: `break`
12: **if** pending_r$< M$ **then**
13: `seglist`$\leftarrow []$

Illustration

Preuve

Soit la fonction $f : \mathbb{N} \to \mathbb{N}$ définie par :

$$f(x) = \max\{y \in \mathbb{N} : \exists 1 \leq i \leq n, [x, y] \subset [a_i, b_i]\}$$

Et plus généralement, soient les fonctions :

$$f_r(x) = \max\left\{y \in \mathbb{N} : \exists 1 \leq i_1, \ldots, i_r \leq n, [x, y] \subset \cup_{i \in \{i_1, \ldots, i_r\}} [a_i, b_i]\right\}$$

On vérifie facilement que $f_1 = f$ et que le problème posé correspond à :

$$\min\{r : f_r(0) \geq M\}$$

La preuve repose alors sur la propriété (exercice !) :

$$f_r(0) = f(f_{r-1}(0))$$

Preuve

Soit la fonction $f : \mathbb{N} \to \mathbb{N}$ définie par :

$$f(x) = \max\{y \in \mathbb{N} : \exists 1 \leq i \leq n, [x, y] \subset [a_i, b_i]\}$$

Et plus généralement, soient les fonctions :

$$f_r(x) = \max\left\{y \in \mathbb{N} : \exists 1 \leq i_1, \ldots, i_r \leq n, [x, y] \subset \cup_{i \in \{i_1, \ldots, i_r\}} [a_i, b_i]\right\}$$

On vérifie facilement que $f_1 = f$ et que le problème posé correspond à :

$$\min\{r : f_r(0) \geq M\}$$

La preuve repose alors sur la propriété (exercice !) :

$$f_r(0) = f(f_{r-1}(0))$$

[indice : on pourra raisonner par double inégalité.]

Problème 3 : Codeforces 483A - Contrexemple

Une de vos amies a récemment appris la notion de nombres premiers entre eux. Deux nombres sont dit premiers entre eux, s'ils n'ont pas d'autres diviseurs commun que 1. Depuis elle formule souvent différents énoncés. Récemment, elle a supposé que si la paire (a, b) était formée de nombres premiers entre eux, ainsi que la paire (b, c), alors la paire (a, c) était également formée de nombres premiers entre eux.
Vous voulez trouver un contrexemple à l'énoncé de votre amie. Ainsi, votre tâche consiste à trouver trois nombres distincts (a, b, c), pour lesquels l'énoncé est faux. Ces trois nombres doivent être choisis dans l'intervalle $[l, r]$, fourni dans le problème. Plus précisément, on doit avoir $l \leq a < b < c \leq r$, avec $1 \leq l \leq r \leq 10^{18}$ et $r - l \leq 50$.

Entrées/Sorties

Entrées

La première ligne contient le nombre de cas à traiter, puis les autres lignes contiennent les cas et sont formatées de la manière suivante. Deux entiers positifs l et r, sur la même ligne, séparés par un espace. On sait que $l \leq r \leq 10^{18}$ et $r - l \leq 50$.

Exemples

2 4
10 11
90000000000000009 90000000000000029

Sorties

Trois entiers distincts $a < b < c$ qui forment le contrexemple, s'il existe ; ou -1 s'il n'y a pas de contrexemple. S'il y a plusieurs contrexemples, on peut en choisir un quelconque.

Exemples

2 3 4
-1
90000000000000009 90000000000000010 90000000000000021

Des idées ?

L'approche par "force brute"

Les ordres de grandeur pour l et r sont impressionnants. C'est fait pour. Mais en fait, on va le voir, le facteur limitant est leur écart, qui, lui, est garanti inférieur ou égal à 50. On peut donc attaquer le problème de façon exhaustive en faisant attention aux types d'entiers utilisés. En python, on est tranquille de ce point de vue, grâce au basculement automatique vers les entiers longs "L".
On va donc énumérer tous les triplets de nombres entiers distincts (a, b, c) vérifiant $l \leq a < b < c \leq r$.

L'algorithme

Algorithm 2 Contrexemple

Input: $[l, r]$
Ouput: a, b, c or -1

```
1: for a ← l : r do
2:     for b ← a + 1 : r do
3:         for c ← b + 1 : r do
4:             if coprime(a,b) and coprime(b,c) then
5:                 if not coprime(a,c) then
6:                     return (a,b,c)
7: return -1
```

Il reste encore à écrire la fonction **coprime**.

coprime

Algorithm 3 coprime

Input: (a, b)

Ouput: **False**/**True**

```
1: c ← min(a, b), d ← max(a, b)
2: if a == 0 then
3:     if b == 1 then
4:         return True
5:     else
6:         return False
7: else
8:     return coprime (d-c,c%(d-c))
```

Preuve

La seule chose à vérifier est la correction de la routine **coprime** qui est une déclinaison de l'algorithme d'Euclide. L'identité fondamentale étant :

$$\mathbf{pgcd}(a, b) = \mathbf{pgcd}(a, b - a)$$

Dans notre cas on sait que $b - a$ est majoré par 50. Si on note r le reste de la division Euclidienne de a par $b - a$ on a donc :

$$\mathbf{pgcd}(a, b) = \mathbf{pgcd}(r, b - a)$$

Question subsidiaire

Quelle serait la complexité de la méthode “force brute” pour le problème de la couverture minimale ? Pour quel ordre de grandeur de n serait-il possible de traiter le problème ainsi ?
($n = 10, 10^2, 10^3, 10^6$?)

Problème 4 : GCJ 2009 1C - Soudoyer les prisonniers

Dans un royaume il y a des cellules de prison numérotées de 1 à P construites de telle sorte à former un segment de ligne droite. Les cellules i et $i+1$ sont adjacentes et leur prisonniers sont appelés "voisins". Un mur muni d'une fenêtre les sépare et ils peuvent communiquer via cette fenêtre.

Tous les prisonniers vivent en paix jusqu'à ce qu'un prisonnier soit relâché. Quand cela se produit, le prisonnier libéré fait part de la nouvelle à ses voisins, qui en parlent à leurs voisins, etc., jusqu'à atteindre la cellule 1, P ou une cellule dont la cellule voisine est vide. Quand un prisonnier découvre qu'un autre prisonnier a été libéré, de colère, il casse tout dans sa cellule, sauf si il a été préalablement soudoyé par une pièce d'or. Il faut donc veiller à soudoyer tous les prisonniers susceptibles de tout casser dans leur cellule avant de libérer un prisonnier.

En supposant que toutes les cellules sont initialement occupées par un unique prisonnier et qu'un prisonnier par jour au plus puisse être relaché, et en connaissant la liste des Q prisonniers à relâcher, il faut trouver l'ordre de libération des prisonniers de cette liste qui soit le moins coûteux en pièce d'or. Ordres de grandeur : $1 \leq P \leq 10^4$, $1 \leq Q \leq 10^2$.

A noter que le soudoiement n'est actif qu'un seul jour.

Entrées/Sorties

Entrées

La première ligne donne le nombre de cas à traiter N. Chaque cas consiste alors en deux lignes. La première ligne présente les deux entiers P et Q séparés par un espace. Enfin la deuxième ligne contient les indices des Q cellules à libérer, triés par ordre croissant et separés par un espace.

Exemple

2
8 1
3
20 3
3 6 14

Sorties

Chaque cas doit correspond à une ligne formatée de la manière suivante :
Cas #X : G
où G est le nombre de pièces d'or à dépenser au minimum pour ce cas.

Exemple

Cas #1 : 7
Cas #2 : 35

Des idées ?

L'approche par programmation dynamique

Une observation fondamentale à faire pour ce problème est la suivante. Imaginons qu'un oracle nous donne le nombre de pièces d'or minimum à dépenser pour toutes les prisons contenant strictement moins de P cellules et strictement moins de Q prisonniers. On peut alors résoudre notre problème par :

$$min_{q \in Q} G(1, q-1) + G(q+1, P) + P - 1$$

On va donc utiliser cette observation pour calculer la quantité cherchée récursivement. D'autre part, afin d'éviter de refaire les calculs plusieurs fois, on va *mémoriser* les réponses.

L'algorithme

Algorithm 4 Soudoyer les prisonniers

Input: a, b, $Q = [q_1, \ldots, q_Q]$

Ouput: G

1: **if card**$([q_1, \ldots, q_Q]) == 1$ **then**
2: **return** $b - a$
3: $r \leftarrow \infty$
4: **for** q **in** $[q_1, \ldots q_Q]$ **do**
5: `rtemp`$\leftarrow G(a, q-1, Q) + G(q+1, b, Q) + b - a$
6: **if** `rtemp`$< r$ **then**
7: $r \leftarrow$`rtemp`
8: **memoreturn**$(a, b) = r$

Questions subsidiaires

- Pour quels ordres de grandeur, serait-il possible d'implémenter une approche force brute ?
- Une approche gloutonne serait-elle envisageable ?