
Decodificar Base64 (2)**T77863_ca**

(Aquest problema fa servir la funció `char_to_base64` del problema "Decodificar Base64 (1)".)

Es tracta de fer un programa que, donada una seqüència de caràcters a l'entrada (un dels 64 que representen els dígit de base 64), la **decodifiqui en els seus bytes**. La decodificació funciona de la següent manera:

1. Primer, per a cada grup de 4 caràcters (o quartet) de l'entrada c_1, c_2, c_3 i c_4 , els transformem en els seus dígit corresponents. d_1, d_2, d_3, d_4 , usant la funció `char_to_base64`.
2. Després, amb els 4 dígit d_i reconstruim el natural x aplicant la fórmula:

$$x = ((d_1 \cdot 64 + d_2) \cdot 64 + d_3) \cdot 64 + d_4$$

3. Tot seguit, reinterpretem x en base 256, i n'extraïem les xifres, que ara són 3: B_1, B_2 , i B_3 . El procés és totalment anàleg a extreure les xifres d'un nombre en base 10, però en base 256. El que haurem fet és calcular els dígit B_i de la fórmula següent:

$$x = (B_1 \cdot 256 + B_2) \cdot 256 + B_3$$

(Cal recordar que el procés d'extracció de les xifres treballat a PRO1 produeix les xifres al revés, començant per B_3)

4. Finalment, mostrem B_1, B_2 i B_3 com a nombres naturals per pantalla, en aquest ordre.

La codificació en base 64 sempre té un nombre de caràcters múltiple de 4, però just al final de la seqüència pot haver-hi '=' o '==', que ens diu que el número de bytes de l'últim quartet són 2 o 1, i no 3:

Si l'últim quartet té algun '=' al final:

- Si té '==': el número de bytes serà 1; assignem $d_3 = 0$ i $d_4 = 0$, decodifiquem segons els passos anteriors, però només mostrem B_1 per pantalla.
- Si té '=': el número de bytes és 2; assignem $d_4 = 0$, decodifiquem segons els passos anteriors, però només mostrem B_1 i B_2 per pantalla.

Entrada

L'entrada consisteix en diversos casos, a on cada cas és una seqüència de caràcters acabada en punt en una línia.

Sortida

La sortida ha de ser una línia per a cada cas amb els bytes com a nombres naturals. Abans de cada byte, inclòs el primer, hi ha d'haver un espai.

Observació

- Aquest problema té com a centres d'interès la **correctesa** i la **llegibilitat**. En particular, es valorarà que el programa utilitzi funcions per evitar repetició i separar les diverses tasques.
- Si teniu la funció `char_to_base64` i el Jutge us l'ha acceptat, useu-la directament. Si no, copieu la següent definició, (i afegiu `#include <algorithm>`):

```
int char_to_base64(char c) {  
    static char _syms[65] =  
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
        "abcdefghijklmnopqrstuvwxyz"  
        "0123456789+/";  
    return std::find(_syms, _syms + 64, c) - _syms;  
}
```

Exemple d'entrada 1

```
AAAA.  
AQEB.  
AgIC.  
////.  
AA==.  
AAA=.  
AQ==.  
AQE=.  
ZGRkZA==.  
CgAUAB4=.  
////AAAA.
```

Exemple de sortida 1

```
0 0 0  
1 1 1  
2 2 2  
255 255 255  
0  
0 0  
1  
1 1  
100 100 100 100  
10 0 20 0 30  
255 255 255 0 0 0
```

Exemple d'entrada 2

```
LA==.  
Jg==.  
lg==.  
Sg==.  
3Q==.  
Mg==.  
Rw==.  
Ng==.  
BQ==.  
TA==.  
og==.
```

Exemple de sortida 2

```
44  
38  
150  
74  
221  
50  
71  
54  
5  
76  
162
```

Exemple d'entrada 3

```
9qE=.  
xpM=.  
b9M=.  
DUk=.  
CH0=.  
rPA=.  
kSk=.  
wV8=.  
bQg=.  
ZvY=.
```

Exemple de sortida 3

```
246 161  
198 147  
111 211  
13 73  
8 125  
172 240  
145 41  
193 95  
109 8  
102 246
```

Exemple d'entrada 4

u1Zi.
q6nx.
NJFe.
tjy1.
1Bq2.
/eRh.
q0A/.
2K1L.
ANVh.
eg+n.

Exemple de sortida 4

187 86 98
171 169 241
52 145 94
182 60 181
212 26 182
253 228 97
171 64 63
216 173 75
0 213 97
122 15 167

Informació del problema

Autor : Pau Fernández

Generació : 2025-10-30 16:19:43

© *Jutge.org*, 2006–2025.

<https://jutge.org>