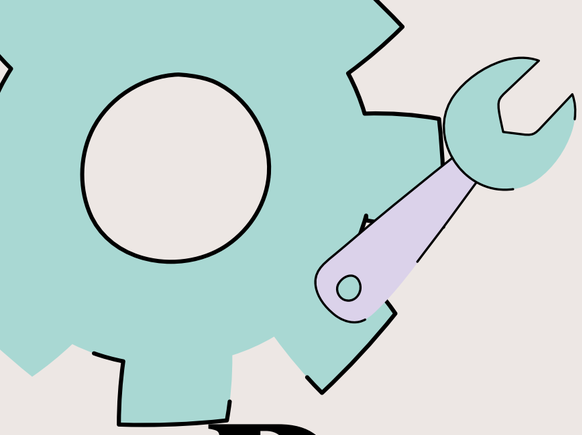
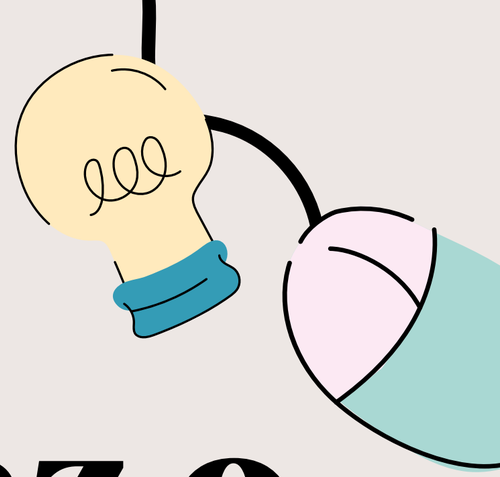
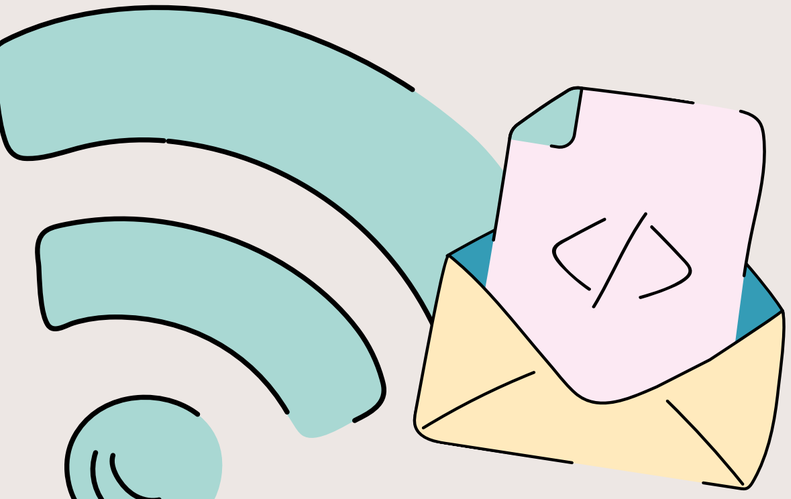


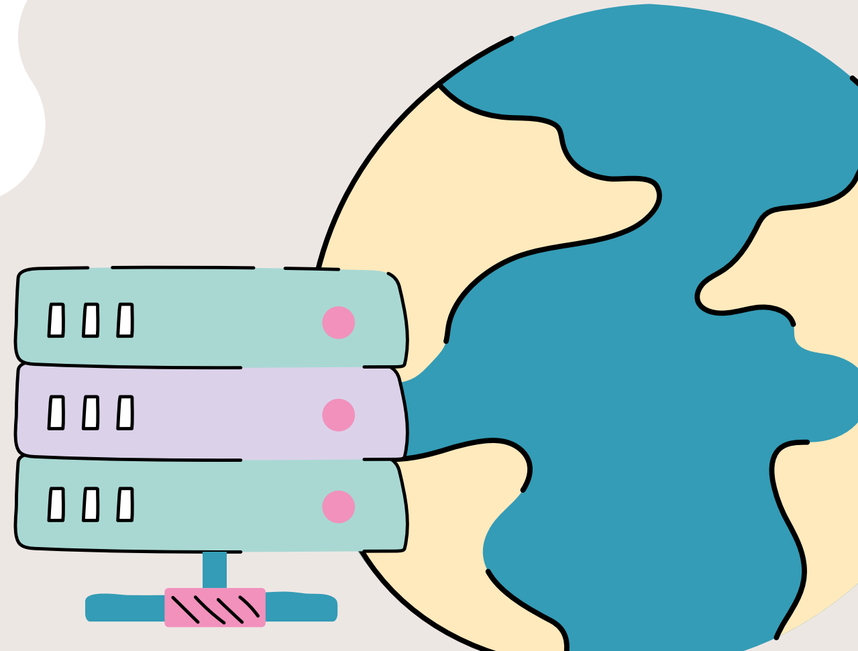


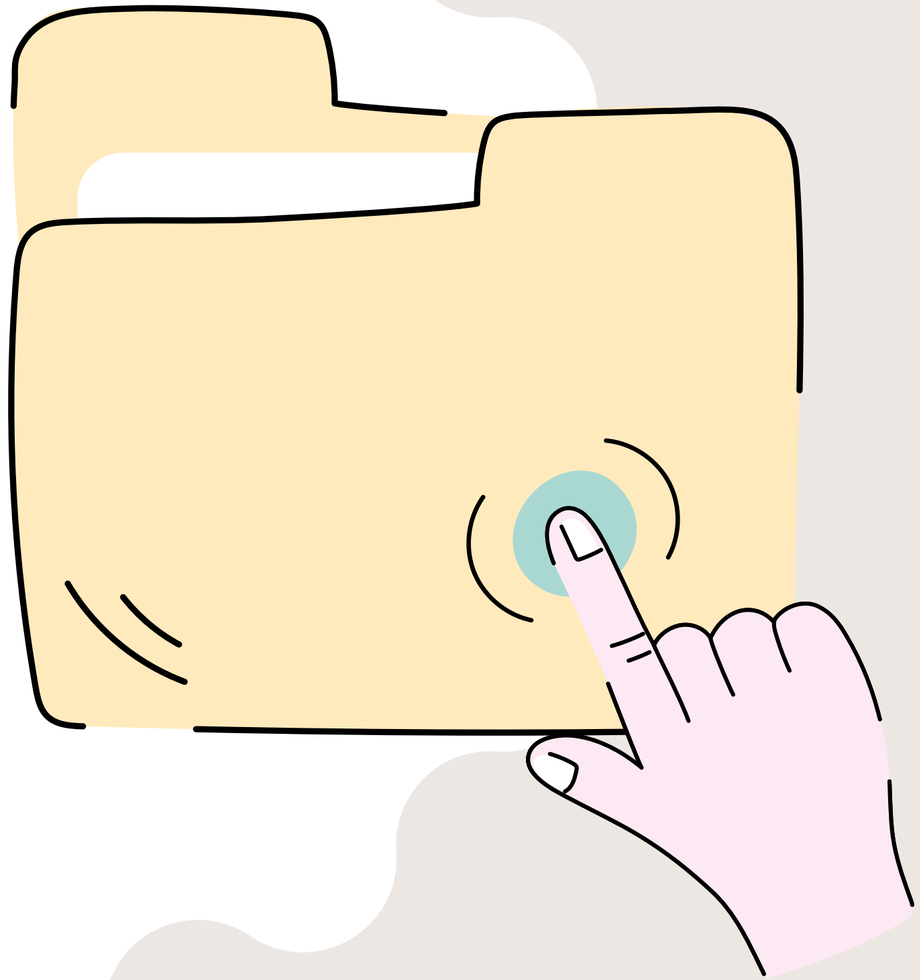
Programación voraz o greedy

CPC
UNMSM

2024







Contenidos

- Técnicas avanzadas de diseño y análisis
- ¿Qué es la programación voraz o Greedy?
- Algoritmos Greedy
  - Elementos de la estrategia Greedy
  - Códigos Huffman
- Ejercicios a revisar

Técnicas avanzadas de diseño y análisis

Dentro de las técnicas avanzadas para el diseño y análisis de algoritmos eficientes se destaca la programación dinámica, los algoritmos voraces y el análisis de amortización.

En esta ocasión, nos centraremos en la segunda técnica (Greedy), que normalmente se aplica a problemas de optimización en los que se toman una serie de decisiones para llegar a una solución óptima. La idea de estos algoritmos es tomar cada elección óptima sin tener en cuenta los datos futuros.

¿Qué es la programación voraz o Greedy?

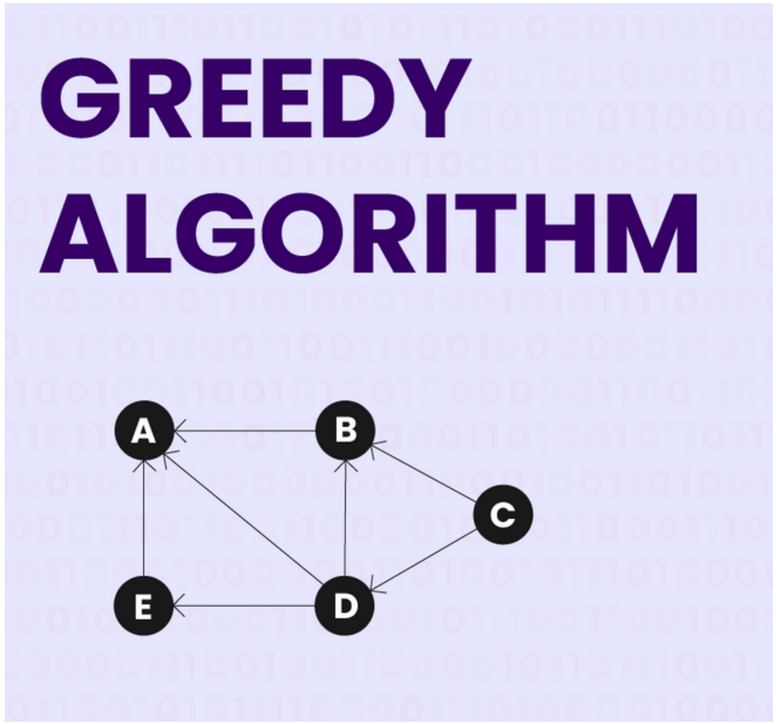
La programación voraz es una estrategia de búsqueda o enfoque utilizado para procesar datos y tomar decisiones óptimas.

Escoge la mejor opción disponible en el momento

No observan opciones generales

Usualmente son muy eficientes

Procesan un dato a la vez



Elementos de la estrategia Greedy

1. Presentar el problema de optimización como uno en el que toma una decisión y se queda con un subproblema para resolver.

2. Demostrar que siempre hay una solución óptima para el problema original que hace la elección codiciosa, para que la elección codiciosa sea siempre segura.

3. La subestructura óptima se demuestra al evidenciar que, tras la elección codiciosa, se obtiene un subproblema.

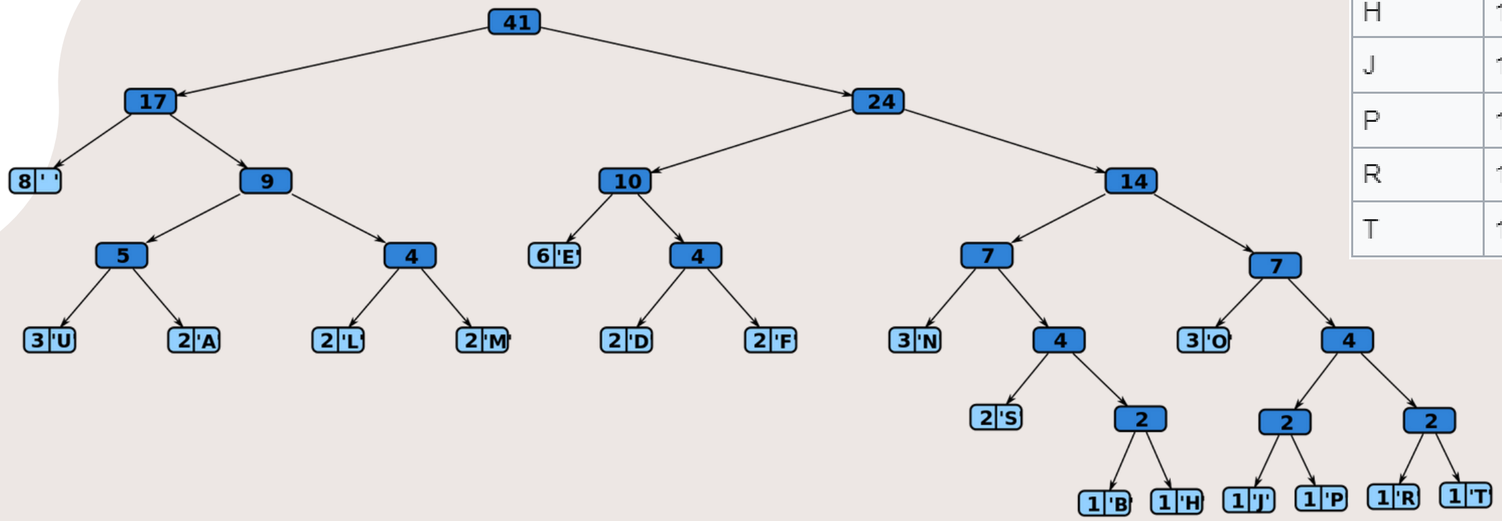
Código Huffman

Encargado de comprimir datos, ofreciendo ahorros significativos (20% al 90%). Se basa en una tabla de frecuencias de caracteres para asignar códigos binarios óptimos.

Tabla de frecuencias

Carácter	Frecuencia	Código
Espacio	8	00
E	6	100
N	3	1100
O	3	1110
U	3	0100
A	2	0101
D	2	1010
F	2	1011
L	2	0110
M	2	0111
S	2	11010
B	1	110110
H	1	110111
J	1	111100
P	1	111101
R	1	111110
T	1	111111

Árbol binario



Ejercicios a revisar

A. Polycarp and Coins

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarp must pay **exactly** n burles at the checkout. He has coins of two nominal values: 1 burle and 2 burles. Polycarp likes both kinds of coins equally. So he doesn't want to pay with more coins of one type than with the other.

Thus, Polycarp wants to minimize the difference between the count of coins of 1 burle and 2 burles being used. Help him by determining two non-negative integer values c_1 and c_2 which are the number of coins of 1 burle and 2 burles, respectively, so that the total value of that number of coins is **exactly** n (i. e. $c_1 + 2 \cdot c_2 = n$), and the absolute value of the difference between c_1 and c_2 is as little as possible (i. e. you must minimize $|c_1 - c_2|$).

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases. Then t test cases follow.

Each test case consists of one line. This line contains one integer n ($1 \le n \le 10^9$) — the number of burles to be paid by Polycarp.

Output

For each test case, output a separate line containing two integers c_1 and c_2 ($c_1, c_2 \ge 0$) separated by a space where c_1 is the number of coins of 1 burle and c_2 is the number of coins of 2 burles. If there are multiple optimal solutions, print any one.

Example

input Copy

```
6
1000
30
1
32
1000000000
5
```

output Copy

```
334 333
10 10
1 0
10 11
333333334 333333333
1 2
```

Note

The answer for the first test case is "`334 333`". The sum of the nominal values of all coins is $334 \cdot 1 + 333 \cdot 2 = 1000$, whereas $|334 - 333| = 1$. One can't get the better value because if $|c_1 - c_2| = 0$, then $c_1 = c_2$ and $c_1 \cdot 1 + c_1 \cdot 2 = 1000$, but then the value of c_1 isn't an integer.

The answer for the second test case is "`10 10`". The sum of the nominal values is $10 \cdot 1 + 10 \cdot 2 = 30$ and $|10 - 10| = 0$, whereas there's no number having an absolute value less than 0.

1551 - A. Polycarp and Coins
https://codeforces.com/problemset/problem/1551/A

B. Plus-Minus Split

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a string s of length n consisting of characters "+" and "−". s represents an array a of length n defined by $a_i = 1$ if $s_i =$ "+" and $a_i = -1$ if $s_i =$ "−".

You will do the following process to calculate your penalty:

1. Split a into non-empty arrays $b_1, b_2, \ldots, b_k$ such that $b_1 + b_2 + \ldots + b_k = a^\dagger$, where $+$ denotes array concatenation.
2. The *penalty* of a single array is the absolute value of its sum multiplied by its length. In other words, for some array c of length m, its penalty is calculated as $p(c) = |c_1 + c_2 + \ldots + c_m| \cdot m$.
3. The total penalty that you will receive is $p(b_1) + p(b_2) + \ldots + p(b_k)$.

If you perform the above process optimally, find the minimum possible penalty you will receive.

† Some valid ways to split $a = [3, 1, 4, 1, 5]$ into $(b_1, b_2, \ldots, b_k)$ are $([3], [1], [4], [1], [5])$, $([3, 1], [4, 1, 5])$ and $([3, 1, 4, 1, 5])$ while some invalid ways to split a are $([3, 1], [1, 5])$, $([3], [\,], [1, 4], [1, 5])$ and $([3, 4], [5, 1, 1])$.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 5000$) — the length of string s.

The second line of each test case contains string s ($s_i \in \{+, -\}$, $|s| = n$).

Note that there are **no** constraints on the sum of n over all test cases.

Output

For each test case, output a single integer representing the minimum possible penalty you will receive.

Example

input Copy

```
5
1
+
5
-----
6
+-+-+-
10
--+++++++-
20
+---++++-+++++---++-
```

output Copy

```
1
5
0
4
4
```

1919 - B. Plus-Minus Split
https://codeforces.com/problemset/problem/1919/B

1551 - A. Polycarp and Coins

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t, n, a, b;
    cin>>t;

    while(t--){
        cin>>n;
        a=n/3;
        b=a;
        if(n%3==1) a++;
        else if(n%3==2) b++;
        cout<<a<<" "<<b<<"\n";
    }

    return 0;
}

```

1919 - B. Plus-Minus Split

```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t, n;
    string s;
    cin>>t;

    while(t--){
        int cont=0;
        cin>>n>>s;
        for(int i=0; i<n; i++){
            if(s[i] == '+') cont+=1;
            else cont+=-1;
        }
        cout<<abs(cont)<<"\n";
    }

    return 0;
}
```

Gracias