Palíndromo ABC

Dados tres enteros *a b c*, escribid el palíndromo lexicográficamente más pequeño con *a* 'A', *b* 'B' y *c* 'C'. En caso que no exista ningún palíndromo con ese número de letras escribid 'ROP'.

Input Format

La entrada empieza con un entero *t* que indica el número de casos. Cada caso contiene una línea con tres enteros *a*, *b* y *c*.

Constraints

0 ≤ a, b, c ≤ 1000

t ≤ 1000

Al menos uno de los enteros es diferente de 0.

14 puntos: Dos de los enteros son 0.

27 puntos: Todos los enteros son pares.

59 puntos: Sin condiciones extras.

Output Format

Para cada caso una línea con la respuesta.

Solución en Python

```
ncases = int(input())

for case in range(ncases):
    line = input()
    a, b, c = int(line[0]), int(line[1]), int(line[2])

    if (a%2 + b%2 + c%2 > 1): print("ROP")
    else :
        print((a//2)*'A' + (b//2)*'B' + (c//2)*'C', end='')
        if (a%2 == 0): print('A', end='')
        elif (b%2 == 0): print('B', end='')
        elif (c%2 == 0): print('C', end='')
        print((c//2)*'C' + (b//2)*'B' + (a//2)*'A')
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>

using namespace std;

int main() {
    int t;
    cin >> t;
    while (t--) {
        int a, b, c;
        cin >> a >> b >> c;
        if ((a%2) + (b%2) + (c%2) > 1) {
               cout << "ROP" << endl;
               continue;
        }
        cout << string(a/2,'A');
        cout << string(b/2,'B');
        cout << string(c/2,'C');
        if (a%2) cout << 'A';
        else if (b%2) cout << 'B';
        else if (c%2) cout << 'C';
        cout << string(c/2,'C');
        cout << string(b/2,'B');
        cout << string(a/2,'A');
        cout << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Borrando letras del SMS

Pedro quiere enviar un SMS a sus padres, como no le queda mucho saldo ha decidido que va a prescindir de los espacios. Haciendo eso le ha quedado un mensaje de *n* letras, pero solo tiene saldo para pagar *k* letras, con $k \leq n$.

Pedro quiere borrar letras, sin alterar el orden de las demás, de manera que el mensaje final sea el lexicográficamente menor posible con *k* letras.

Dados *n*, *k*, y el mensaje ayuda a Pedro a reducir el mensaje.

Input Format
La entrada empieza con un entero t, el número de casos. Cada caso consiste en una línea con los enteros n y k, seguida de otra línea con el mensaje original de n letras, todas ellas minúsculas.

Constraints

$1 \leq k \leq n \leq 10000$

$t \leq 100$

1 punto: $k = n$

3 puntos: $k = 1$

5 puntos: $k = 2$

10 puntos: $k \leq 10$

3 puntos: $k = n-1$

5 puntos: $k = n-2$

10 puntos: $k \geq n-10$

30 puntos: $n \leq 1000$

33 puntos: $n \leq 10000$

Output Format
Para cada caso escribid una línea con el mensaje final.

Solución en Python

```
from collections import deque

ncases = int(input())

for case in range(ncases):
    n, k = input().split(' ')
    n, k = int(n), int(k)

    a = input()

    if (n == k):
        print(a)
        continue
    elif (k >= n-10):
        s = n - k
        sol = a
        for i in range(s):
                sol_temp = ''
                b = 0
                for i in range(len(sol)-1):
                        if(ord(sol[i]) > ord(sol[i+1]) and b == 0): b = 1
                        else: sol_temp += sol[i]
                if(b == 1): sol_temp += sol[-1]
                sol = sol_temp

        print(sol)
        continue

    A = [deque([])  for i in range(26)]

    for i in range(n):
        A[ ord(a[i]) - ord('a') ].append(i)


    it = 0
    sol = ''
    for i in range(k):
        for j in range(26):

                while (len(A[j]) != 0 and A[j][0] < it):
                        A[j].popleft()

                if len(A[j]) == 0:
                        continue

                d = A[j][0]

                if (n - d >= k - i):
                        it = d
                        A[j].popleft()
                        sol += chr(ord('a') + j)
```

```
                        break

    print(sol)
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```
#include <iostream>
#include <queue>

using namespace std;

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, k;
        string a;
        cin >> n >> k >> a;
        queue<int> A[26];
        for (int i = 0; i < n; ++i)
                A[a[i]-'a'].push(i);
        int it = 0;
        for (int i = 0; i < k; ++i) {
                for (int j = 0; j < 26; ++j) {
                        while (!A[j].empty() and A[j].front() < it) A[j].pop();
                        if (A[j].empty()) continue;
                        int d = A[j].front();
                        if (n - d >= k - i) {
                                it = d;
                                A[j].pop();
                                cout << char('a' + j);
                                break;
                        }
                }
        }
        cout << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Las galletas de Jan

Todos sabemos que a Jan le encantan las galletas. En este preciso momento, Jan tiene ante sus ojos montones de galletas de muchos tipos distintos: de chocolate, de limón, de coco, pretzels, crackers, biscotti, macarons, lebkuchen, stroopwafels... Como todo amante de las galletas, quiere comer el máximo número de tipos de galletas distintos, pero tiene un problema, y es que hay otras personas que también quieren comérselas. Para cada galleta, Jan sabe de qué tipo es y el instante en el que alguien se la comerá si él no lo hace antes. Las buenas noticias son que Jan puede comer un número ilimitado de galletas (incluso puede tomar varias del mismo tipo) y que las come de forma instantánea. La mala es que entre dos galletas consecutivas debe descansar un tiempo proporcional al número de galletas que se ha comido hasta ese momento. Si Jan se come la primera galleta en el instante *t*, deberá esperar al instante *t+1* para comer la segunda. Si se come la segunda en el instante *t*, deberá esperar al instante *t+2* para comer la tercera. Y en general, si se come la *k*-ésima en el instante *t*, deberá esperar al instante *t+k* para comer la siguiente.

Jan quiere comer el máximo número de tipos distintos de galletas y vuestro objetivo es decirle cual es ese máximo. Aun así, notad que Jan puede comer tantas galletas como quiera del mismo tipo. Inicialmente Jan tiene el estómago vacío y puede comer la primera galleta sin tener que esperar. Además, si una galleta va a desaparecer en el instante *t*, Jan puede comérsela hasta el mismo instante *t* (si es necesario se la quitará de las manos al que se la fuera a comer).

Input Format

La entrada empieza con un entero t, el número de casos. Le siguen dos enteros *m, n*, la cantidad de tipos de galletas y el número de galletas, respectivamente. A continuación, se describen las *n* galletas. Cada una viene descrita por dos enteros, a_i, t_i. El primero indica el tipo de galleta y el segundo el instante en que desaparecerá la galleta si Jan no se la come antes.

Constraints

Para todos los casos $t \leq 40$, $m \leq n$ y $1 \leq a_i \leq m$. Además para cualquier número entre *1* y *m*, habrá, como mínimo, una galleta de ese tipo.

Caso 1 (15 puntos): $m \leq 10$, $n \leq 100$, $ti \leq 1000$

Caso 2 (15 puntos): $m \leq 10$, $n <= 1000$, $ti \leq 1000000$

Caso 3 (20 puntos): $m \leq 100$, $n = m$, $ti \leq 10000$

Caso 4 (25 puntos): $m \leq 10000$, $n = m$, $ti \leq 10^9$

Caso 5 (25 puntos): $m \leq 50000$, $n \leq 200000$, $ti \leq 10^9$

Output Format

Imprimid *t* líneas, cada una con un único entero: el número máximo de tipos distintos de galletas que puede comer Jan.

Solución en Python

```python
ncases = int(input())

for case in range(ncases):
    m, n = input().split(' ')
    m, n = int(m), int(n)

    V = [-1 for i in range(m)]

    for i in range(n):
        a, t = input().split(' ')
        a, t = int(a), int(t)

        V[a-1] = max(V[a-1], t)

    V = sorted(V)

    cont, t = 0, 0
    for i in range(m):
        if (V[i] >= t):
                cont += 1
                t += cont
    print(cont)
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int t;
    cin >> t;
    while (t--) {
        int m, n;
        cin >> m >> n;
        vector<int> V(m,-1);
        while (n--) {
                int a,t;
                cin >> a >> t;
                V[a-1] = max(V[a-1],t);
        }
        sort(V.begin(),V.end());
```

```
        int cont = 0;
        int t = 0;
        for (int i = 0; i < m; ++i) {
                if (V[i] >= t) {
                        ++cont;
                        t += cont;
                }
        }
        cout << cont << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Excursiones multidimensionales

Considérese la cuadrícula $\mathbb{Z}^k$, es decir, el conjunto de puntos de la forma $(a_1, ..., a_k)$ con a_i entero. Por ejemplo, para $k=2$, esta es la cuadrícula habitual que uno encontraría en una hoja cuadriculada extendida hasta el infinito. Dado un n, se deben calcular el número de caminos que parten del origen y vuelven a este en $2n$ pasos, donde cada movimiento consiste en sumar o restar uno a cualquiera de las coordenadas. El resultado se debe dar módulo *998244353*.

Input Format

La entrada consiste en distintos casos. La primera línea contendrá un natural *t*, el número de casos.
Cada caso consistirá en una línea con los enteros *n* y *k* separados por un espacio.

Constraints

1 <= t <= 200000

21 puntos: *1 <= n <= 10^5, k = 1*

12 puntos: *1 <= n <= 30, k <= 5*

43 puntos: *1 <= n <= 100, k <= 100*

24 puntos: *1 <= n <= 10^5, k <= 2*

Output Format

La salida debe ser un natural para cada caso, el número de caminos distintos que vuelven al origen módulo *998244353.*

Solución en C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

typedef long long ll;
typedef vector<ll> vi;
typedef vector<vi> vvi;

const ll mod = 998244353;

ll pot(ll x, ll e) {
    if (e == 0) return 1;
    ll k = pot(x*x%mod, e/2);
    if (e%2) return k*x%mod;
```

```
    return k;
}

const int N = 100000;
const int M = 100;

int main() {
    vi fact(2*N+1,1);
    vi invf(2*N+1,1);
    vi conv(N+1,1);
    for (ll i = 1; i <= 2*N; ++i) {
        fact[i] = i*fact[i-1]%mod;
        invf[i] = pot(fact[i],mod-2);
    }
    for (int i = 1; i <= N; ++i) {
        conv[i] = fact[2*i]*invf[i]%mod*invf[i]%mod;
    }
    vvi D(M+1, vi(M+1,0));
    D[0][0] = 1;
    for (int j = 1; j <= M; ++j) {
        for (int i = 0; i <= M; ++i) {
                for (int w = 0; w <= i; ++w) {
                        D[i][j] = (D[i][j] + D[w][j-1]*conv[i-w]%mod*invf[2*(i-w)])%mod;
                }
        }
    }

    int t;
    cin >> t;
    while (t--) {
        int n, k;
        cin >> n >> k;
        if (k == 1) {
                cout << conv[n] << endl;
        } else if (k == 2) {
                cout << conv[n]*conv[n]%mod << endl;
        } else {
                cout << D[n][k]*fact[2*n]%mod << endl;
        }
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)