Soluciones OIFem II Nivel 2

Concurso de práctica I

Every change

Teoría

- Backtracking

Solución

Haremos un backtracking en el que mantendremos el número actual para el que tenemos que dar cambio, el índice de la moneda que nos toca usar y el vector de las monedas usadas hasta este momento. En cada paso podemos volver a usar la misma moneda o cambiar a la siguiente.

Código

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

using vi = vector<int>;

const vi coins = {50, 20, 10, 5, 2, 1};
vi change;

void solve(int n, int ci) {
    if (n == 0) {
        cout << change[0];
        for (int i = 1; i < change.size(); ++i) {
            cout << ' ' << change[i];
        }
        cout << endl;
    }
    else {
        for (int i = ci; i < coins.size(); ++i) {
            int c = coins[i];
            if (c <= n) {
                change.push_back(c);
                solve(n-c, i);
                change.pop_back();
            }
        }
    }
}
```

```
28
29  int main() {
30      ios::sync_with_stdio(0);
31      cin.tie(0);
32
33      int n;
34      while (cin >> n) {
35          solve(n, 0);
36          cout << endl;
37      }
38  }
```

Broken palindromes

Teoría

- Backtracking

Solución

Hacemos un backtracking manteniendo el string actual y un vector de booleanos que indiquen si hemos usado ya el trozo j-esimo o no. En cada paso miramos todos los trozos y probamos todos los no usados hasta el momento. Cuando hemos usado n trozos habremos colocado ya todos, por lo que verificamos si el string formado es un palíndromo.

Código

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

using vs = vector<string>;
using vi = vector<int>;
using vb = vector<bool>;

int n;
vs v;
vb used;
string s;
bool solved;

void bt(int i) {
    if (not solved) {
        if (i == n) {
            int m = s.size();
            bool pal = true;
            for (int j = 0; pal and j <= m/2; ++j) {
                pal &= s[j] == s[m-1-j];
            }
            if (pal) {
                solved = true;
                cout << s << endl;
            }
        }
        else {
            for (int j = 0; j < n; ++j) {
                if (not used[j]) {
                    used[j] = true;
                    s += v[j]; // ponemos el trozo j-esimo
                    bt(i+1);
                    for (int k = 0; k < v[j].size(); ++k) {
                        s.pop_back(); // quitamos cada letra del trozo j-esimo
                    }
                    used[j] = false;
                }
            }
        }
    }
}
```

```
42
43  int main() {
44      while (cin >> n) {
45          v = vs(n);
46          for (int i = 0; i < n; ++i) {
47              cin >> v[i];
48          }
49          solved = false;
50          used = vb(n, false);
51          s.clear();
52          bt(0);
53      }
54  }
```

Interest rates

Teoría

- Búsqueda binaria

Solución

Por abreviar, llamemos a los profesores A y B. A le presta a B m euros. B tendría que pagar f euros diarios a A, mientras que si B elige la opción del banco, le cobran un interés del $r\%$ diario. Es decir, si pasan d días, el banco le cobra $(1 + \frac{r}{100})^d \times m$, mientras que A le habría cobrado $m + d \times f$.
El problema nos pide encontrar la mínima d entera tal que $m + df < (1 + \frac{r}{100})^d m$. La expresión de la derecha es exponencial y podríamos tener números demasiado grandes incluso usando long double, por lo que podemos aplicar logaritmos a ambos lados para tener los números en una mejor escala (como los logaritmos son una función estrictamente creciente, preservan las desigualdades), hacemos búsqueda binaria sobre d tal que: $\log(m + df) < d \log(r) + m$

Código

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

using ld = long double;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    ld m, f, r;
    while (cin >> m >> f >> r) {
        r = 1 + r/100;
        int lo = 1, hi = 10000000, ans = 10000000;
        while (lo <= hi) {
            int d = (lo + hi)/2;
            ld a = log(m + f*d);
            ld b = 1.0*d*log(r) + log(m);
            if (a < b) {
                ans = d;
                hi = d-1;
            }
            else {
                lo = d+1;
            }
        }
        cout << ans << endl;
    }
}
```

Graffiti covering

Teoría

- Búsqueda binaria

Solución

Vemos que el problema cumple la propiedas necesaria para aplicar búsqueda binaria: Si una longitud l funciona, entonces $l+1$ seguro que funciona, por lo que trataremos de buscar la mínima l probando de forma eficiente, con búsqueda binaria. Es fácil verificar de forma greedy si una l concreta funciona.

Código

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

using vi = vector<int>;

int g, p;
vi v;

bool ok(int l) { // verifica si con longitud l es posible
    int k = p;
    int curr = -2e9;
    for (int i = 0; i < g; ++i) {
        if (v[i] >= curr + l) {
            if (k == 0) return false;
            --k;
            curr = v[i];
        }
    }
    return true;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    while (cin >> g) {
        v = vi(g);
        for (int i = 0; i < g; ++i) cin >> v[i];
        sort(v.begin(), v.end());
        cin >> p;
        int l = 1, r = v.back(), ans = -1;
        while (l <= r) {
            int m = (l+r)/2;
            if (ok(m)) {
                ans = m;
                r = m-1;
            }
            else {
                l = m+1;
            }
        }
        cout << (long long)p*ans << endl;
```

```
43          }
44      }
```