Soluciones entrenos OIE

Miners

Solución

La idea es utilizar programación dinámica. Sea `M(n, estado1, estado2)` la mayor cantidad de carbón que puede ser producida tras n-1 envíos de comida. "estado1" describe el historial de envíos a la primera mina, "estado2" el de los enviados a la segunda. Sea también value(estado, comida) el valor que se obtiene cuando se envía "comida" a una mina en estado "estado".

La formulación recursiva sería:

```
enviando_a_mina1 = M(n+1, nuevo-estado1-si-se-envia-a-mina1, estado2)
enviando_a_mina2 = M(n+1, estado1, nuevo-estado2-si-se-envia-a-mina2)
valor_obtenido_al_enviar_a_mina1 = value(estado1, envio)
valor_obtenido_al_enviar_a_mina2 = value(estado2, envio)
M(n, estado1, estado2) =
   max( enviando_a_mina1 + valor_obtenido_al_enviar_a_mina1,
        enviando_a_mina2 + valor_obtenido_al_enviar_a_mina2 )
```

El último paso consiste en darnos cuenta de que `M(n+1, ...)` sólo depende del valor de `M(n, ...)`, con lo que sólo es necesario almacenar dos "filas" simultáneamente.

Código

C++

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N = 1e5;

int food[N + 1];
// Dimensiones: n, penultima comida mina 1, ultima comida mina1,
//                 penultima comida mina 2, ultima comida mina2
int dp[2][4][4][4][4];
string str;

int nFood;
```

```cpp
int value(int a, int b, int c) {
  int ans = 0;
  if (a == 1 || b == 1 || c == 1) {
    ++ans;
  }
  if (a == 2 || b == 2 || c == 2) {
    ++ans;
  }
  if (a == 3 || b == 3 || c == 3) {
    ++ans;
  }
  return ans;
}

int main() {
  cin >> nFood;
  cin >> str;
  for (int i = 1; i <= nFood; ++i) {
    char foodchar = str[i - 1];
    // un switch es una alternativa a los bloques 'if' cuando todas las
    ↪  condiciones son sobre el valor de la misma variable
    switch(foodchar) {
      case 'M':
        food[i] = 1;
        break;
      case 'F':
        food[i] = 2;
        break;
      case 'B':
        food[i] = 3;
        break;
    }
  }

  for (int i = nFood + 1; i >= 1; --i) {
    for (int x1 = 0; x1 <= 3; ++x1) {
      for (int x2 = 0; x2 <= 3; ++x2) {
        for (int y1 = 0; y1 <= 3; ++y1) {
          for (int y2 = 0; y2 <= 3; ++y2) {
            int curr = i % 2;
            int next = curr ^ 1;
            if (i == nFood + 1) {
              dp[curr][x1][x2][y1][y2] = 0;
            } else {
              int send_to_mine1 =
                  value(x1, x2, food[i]) + dp[next][food[i]][x1][y1][y2];
              int send_to_mine2 =
                  value(y1, y2, food[i]) + dp[next][x1][x2][food[i]][y1];
              dp[curr][x1][x2][y1][y2] = max(send_to_mine1, send_to_mine2);
            }
          }
        }
      }
    }
  }
  cout << dp[1][0][0][0][0];
  return 0;
}
```