I Competição Feminina de Programação da UnB 2022
Editorial

Contents

A Naruto vs Sasuke

Para saber quem ganhou a corrida, precisamos descobrir com quanto tempo Naruto e Sasuke cruzam a linha de chegada. Para isso, simplesmente subtraímos a distância total da corrida pela distância atual de cada um e dividimos pela velocidade, atentando-se pelo fato que isso gera uma operação com números reais.

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

using namespace std;

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

    long long n, q; cin >> n >> q;
    pair<int, int> arr[n];
    long long sum = 0;

    for (int i = 0; i < n; i++) {
        int num; cin >> num;
        arr[i] = {num, -1};
        sum += num;
    }

    int val = 0;
    int lastquery2 = -1;
    for (int i = 0; i < q; i++) {
        int op; cin >> op;
        if (op == 1) {
            int idx, x; cin >> idx >> x;
            if (arr[idx - 1].second < lastquery2) sum += x - val;
            else sum += x - arr[idx - 1].first;
            arr[idx - 1] = {x, i};
        }
        else {
            int x; cin >> x;
            sum = n * x;
            lastquery2 = i;
            val = x;
        }

        cout << sum << '\n';
    }

    return 0;
}
```

B Black Fraude

Essa é uma questão clássica de Programação Dinâmica. Para maximizar a quantidade de produtos que Vlad pode comprar, precisamos ver todas as possibilidades. Existem dois caminhos: ou ele pega uma promoção q_i e então ele não pode pegar a $q_* i + 1$, ou ele ignora a promoção q_i e olha para a próxima q_{i+1}.

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

using namespace std;

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

    long long n, q; cin >> n >> q;
    pair<int, int> arr[n];
    long long sum = 0;

    for (int i = 0; i < n; i++) {
        int num; cin >> num;
        arr[i] = {num, -1};
        sum += num;
    }

    int val = 0;
    int lastquery2 = -1;
    for (int i = 0; i < q; i++) {
        int op; cin >> op;
        if (op == 1) {
            int idx, x; cin >> idx >> x;
            if (arr[idx - 1].second < lastquery2) sum += x - val;
            else sum += x - arr[idx - 1].first;
            arr[idx - 1] = {x, i};
        }
        else {
            int x; cin >> x;
            sum = n * x;
            lastquery2 = i;
            val = x;
        }

        cout << sum << '\n';
    }

    return 0;
}
```

C Coca-Cola

Como a resposta é um número em ponto flutuante, o valor das variáveis é lido como ponto flutuante para facilitar os cálculos. Primeiro, calcula-se a altura B bebida por segundo. Sabe-se que V centímetros cúbicos são bebidos por segundo. Isso equivale à altura bebida (em centímetros) vezes a área do fundo do copo (em centímetros quadrados), ou seja,

$$V = B * A$$

$$B = \frac{V}{A}$$

A área de um círculo é calculada pela fórmula

$$A = \pi * R * R$$

na qual R é o raio do círculo (metade do diâmetro do círculo). Se a altura B bebida por segundo for menor ou igual à altura E que é adicionada ao copo por segundo, é impossível beber tudo. Caso o contrário, o tempo gasto para beber tudo é calculado dividindo a altura H total do copo pela altura total bebida por segundo, que é a altura B bebida por segundo menos a altura E acrescentada ao copo a cada segundo.

$$\frac{H}{(B - E)}$$

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

using namespace std;

#define PI acos(-1)

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

    double d, h, v, e; cin >> d >> h >> v >> e;
    double r = d / 2;

    double b = v / (PI * r * r);

    if (b <= e) cout << "NO\n";
    else {
        cout << "YES\n";
        cout << h / (b - e) << '\n';
    }

    return 0;
}
```

D Desafios

A primeira coisa a notar é que não é possível simplesmente substituir os elementos de acordo com as operações solicitadas, pois no pior caso existe $2 * 10^5$ elementos (maior tamanho de N possível de acordo com o enunciado). Então, se houver $2 * 10^5$ (maior tamanho de Q de acordo com o enunciado) operações do tipo 2 (substituir todos os elementos por um número X, com complexidade O(N) no qual N é o número de elementos), haveria aproximadamente $4 * 10^{10}$ operações, o que demoraria mais de 100 segundos para executar sendo que o limite da questão é de 2 segundos (em um segundo pode-se executar em média 10^8 operações). Dito isso, é necessário armazenar o valor dos elementos sem precisar percorrê-los quando houver uma operação do tipo 2. Assim, é possível usar uma variável *val* para armazenar o valor X quando a operação for do tipo 2. Para cada elemento, armazena-se o valor dele e o índice da última operação do tipo 1 até o momento (mudar o valor de um elemento em uma dada posição) que ocorreu nesse elemento. O índice da última operação do tipo 2 até o momento também é armazenada. Assim, tendo a soma dos elementos iniciais, a cada vez que uma operação do tipo 1 for solicitada, verifica-se se o índice da ultima operação do tipo 2 é maior do que o índice da última operação do tipo 1 realizada sobre aquele elemento. Se for maior, significa que o valor atual daquele elemento é o valor armazenado na variável *val* e a soma é atualizada com a diferença entre X e *val*. Se for menor, a soma é atualizada com a diferença entre X e o valor do próprio elemento. Por fim, o valor atual do elemento e o índice da ultima operação do tipo 1 também são atualizados. Já se a operação for 2, o valor atual da soma será o número de elementos vezes x X e o valor da variável *val* e o índice da ultima operação do tipo 2 também são atualizados.

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

using namespace std;

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

    long long n, q; cin >> n >> q;
    pair<int, int> arr[n];
    long long sum = 0;

    for (int i = 0; i < n; i++) {
        int num; cin >> num;
        arr[i] = {num, -1};
        sum += num;
    }

    int val = 0;
    int lastquery2 = -1;
    for (int i = 0; i < q; i++) {
        int op; cin >> op;
        if (op == 1) {
            int idx, x; cin >> idx >> x;
            if (arr[idx - 1].second < lastquery2) sum += x - val;
            else sum += x - arr[idx - 1].first;
            arr[idx - 1] = {x, i};
        }
        else {
            int x; cin >> x;
            sum = n * x;
            lastquery2 = i;
            val = x;
        }

        cout << sum << '\n';
    }

    return 0;
}
```

E Pirata ou Capitã

Essa questão é dividida em duas partes. A primeira é decifrar a mensagem e a segunda é verificar se a mensagem contém as palavras "pirata" e "capita". Para decifrar a mensagem, é possível usar um dicionário para mapear cada letra do alfabeto cifrado à letra correspondente no alfabeto original e depois traduzir a mensagem cifrada. Para verificar se a mensagem original contém a palavra "pirata" ou "capita", podemos usar uma variável *counter* que armazena a posição da letra que está sendo procurada. Assim, é só percorrer a mensagem e comparar a letra da mensagem com a letra que está sendo procurada. Se forem iguais, a variável *counter* é incrementada para procurar a próxima letra, caso contrário continua-se procurando a mesma letra. Se a variável *counter* chegar ao tamanho da palavra "pirata" (ou "capita"), significa que todas as letras já foram encontradas e, logo, a palavra está contida na mensagem. Caso contrário, a palavra não está contida.

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

using namespace std;

bool containsword(string s, string word) {
    int counter = 0;
    for (auto c : s) {
        if (c == word[counter]) counter++;
        if (counter == word.size()) return true;
    }
    return false;
}

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

    map<char, char> m;
    string s; cin >> s;
    int counter = 0;
    for (char i = 'a'; i <= 'z'; i++) {
        m[s[counter]] = i;
        counter++;
    }

    string a; cin >> a;
    string original = "";
    for (auto c : a) {
        original += m[c];
    }

    if (containsword(original, "pirata") && containsword(original, "capita")) cout << "adulterada" << '\n';
    else if (containsword(original, "pirata")) cout << "falsa" << '\n';
    else if (containsword(original, "capita")) cout << "original" << '\n';
    else cout << "quemestaai?" << '\n';

    return 0;
}
```

F Lebre e Tartaruga

A solução dessa questão é percorrer a matriz usando busca em profundidade (DFS) ou busca em largura (BFS). Essa solução usa a busca em profundidade. A lebre e a tartaruga podem ocupar a mesma casa, então é possível usar uma busca em profundidade para cada uma delas sem se preocupar com a posição da outra. Há um único caminho para cada uma delas sair do labirinto sem passar por uma casa que elas já visitaram, assim esse caminho será o tempo mínimo possível. Nessa solução, as casas na borda do labirinto são preenchidas por um caractere que não existe no enunciado (o caractere "-" nesse caso), assim ao chegar a uma casa com esse caractere, sabe-se que a tartaruga (ou lebre) chegou à borda do labirinto. Também é possível verificar isso olhando a posição da casa atual. Como elas não podem passar por uma casa já visitada, uma matriz *visited* é usada para saber se aquela casa já foi visitada. Como nem a lebre nem a tartaruga podem ocupar uma casa com uma parede "#", essas casas já são marcadas como visitadas. Assim a função da *dfs* é chamada partindo da posição inicial da tartaruga e o tempo percorrido é mantido na variável *total_time*, que começa com 0. A cada chamada da função, a casa atual é marcada como visitada. Se a casa atual for uma casa da borda do labirinto, *total_time* é a resposta e a função retorna. Se não, a tartaruga (ou lebre) vai para uma casa vizinha (pra cima, pra baixo, pra direita ou pra esquerda) se ela ainda não tiver sido visitada, cada uma sendo uma chamada de função, e a variável *total_time* é incrementada de acordo com a casa que será visitada. Se a tartaruga nunca chegar à uma casa da borda, a resposta é -1. A solução para a lebre é análoga à solução para a tartaruga.

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

using namespace std;
using ll = long long;

const int MAX = 1e3+7;

int n, m;
char grid[MAX][MAX];
bool visited_hare[MAX][MAX], visited_turtle[MAX][MAX];
int time_turtle, time_hare = 0;

int calculate_time_turtle(int i, int j, int total_time) {
    if (grid[i][j] == 'F') return total_time;
    if (grid[i][j] == 'P') return total_time + 2;

    return total_time + 1;
}

int calculate_time_hare(int i, int j, int total_time) {
    if (grid[i][j] == 'C') return total_time;
    if (grid[i][j] == 'A') return total_time + 2;

    return total_time + 1;
}




void dfs_turtle(int i, int j, int total_time) {
    visited_turtle[i][j] = true;
```

```
32
33      if (grid[i][j] == '-') {
34          time_turtle = total_time;
35          return;
36      }
37
38      vector<pair<int, int>> pos = {
39          {i + 1, j},
40          {i - 1, j},
41          {i, j + 1},
42          {i, j - 1},
43      };
44
45      for (auto [it, jt] : pos) {
46          if (!visited_turtle[it][jt]) dfs_turtle(it, jt,
                calculate_time_turtle(it, jt, total_time));
47      }
48  }
49
50  void dfs_hare(int i, int j, int total_time) {
51      visited_hare[i][j] = true;
52
53      if (grid[i][j] == '-') {
54          time_hare = total_time;
55          return;
56      }
57
58      vector<pair<int, int>> pos = {
59          {i + 1, j},
60          {i - 1, j},
61          {i, j + 1},
62          {i, j - 1},
63      };
64
65      for (auto [it, jt] : pos) {
66          if (!visited_hare[it][jt]) dfs_hare(it, jt, calculate_time_hare(it,
                jt, total_time));
67      }
68  }
69
70  int main() {
71      ios::sync_with_stdio(false);
72      cin.tie(NULL);
73
74      cin >> n >> m;
75      pair<int, int> turtle, hare;
76
77      for (int i = 0; i <= n + 1; i++) {
78          for (int j = 0; j <= m + 1; j++) {
79              grid[i][j] = '-';
80          }
81      }
82
83
84
85
86
87      for (int i = 1; i <= n; i++) {
88          for (int j = 1; j <= m; j++) {
```

```
            cin >> grid[i][j];
            if (grid[i][j] == 'T') turtle = {i, j};
            if (grid[i][j] == 'L') hare = {i, j};
            if (grid[i][j] == '#') {
                visited_turtle[i][j] = true;
                visited_hare[i][j] = true;
            }
        }
    }

    dfs_turtle(turtle.first, turtle.second, 0);
    if (time_turtle != 0) cout << time_turtle << ' ';
    else cout << "-1 ";

    dfs_hare(hare.first, hare.second, 0);
    if (time_hare != 0) cout << time_hare << '\n';
    else cout << "-1\n";

    return 0;
}
```