A Fantástica Máquina de Tubos e Bolinhas do Enigmático Senhor Sibério

Este projeto visa responder a seguinte pergunta:

> Quando uma bolinha é jogada em cada um dos canos, qual o número de bolinhas que sairão em cada um dos canos? Qual o cano por onde saem mais bolinhas?

Rodando o programa

Este projeto contém um arquivo *Makefile*, facilitando a compilação.

Para compilar, basta digitar **`make`** em um simulador de terminal. Para rodar o programa, basta digitar **`./TubesAndBalls`** (em Windows, **`.\TubesAndBalls.exe`**).

Um pouco sobre o programa

Este programa é munido de uma básica interface de interação com o usuário, dada por meio de uma estrutura chamada **`Log`**. Essa estrutura contém métodos para imprimir menus na tela e serve como auxiliar para impressão de mensagens mais verbosas para que o usuário não tenha que recompilar o código para que sejam exibidas mensagens mais verbosas. A **`struct Log`** possui uma variável global chamada de **`Screen`** e um membro booleano que representa sua verbosidade. Sempre que esse membro é verdadeiro, todas as chamadas do método **`Info()`** imprimirão suas mensagens na tela. Para mudar o nível de verbosidade do programa é preciso entrar em Opções e dar *Enable* ou *Disable* na verbosidade dependendo de sua vontade.

Modelando o problema

O problema dos tubos foi modelado de maneira relativamente simples. Cada tubo pertence à classe **`Tube`** e contém um dicionário (**`std::unordered_map`**) de posições para um par (**`std::pair`**) de **`Tube*`** e **`int`**, onde o primeiro membro do par é o tubo destinatário e o segundo é a posição no tubo destinatário onde a bolinha redirecionada. Além disso, cada **`Tube`** possui um identificador único e um tamanho ou comprimento.

Resolvendo o problema

Ao rodar o programa, uma mensagem de seleção de arquivo é exibida pedindo para que o usuário selecione um arquivo de teste. Esses arquivos estão na pasta **`./data/`** e são simples arquivos texto com as configurações da máquina, como especificada no enunciado do problema. Caso o usuário entre com um arquivo que não existe, o programa exibe uma mensagem de erro e termina sua execução. Do contrário, o

programa carrega as configurações utilizando um **tokenizer**, que é um método que recebe uma linha e um delimitador, **string** e **char** respectivamente, e retorna um **std::vector** que representa a **string** dividida em cada ocorrência do delimitador. Os tubos são configurados na mesma rotina de leitura do arquivo de configuração (**ReadConfigFile**).

O método **FindMostCommonEnding** calcula a saída mais comum do sistema (o tubo onde saem mais bolinhas).

```
void FindMostCommonEnding(unordered_map<int, Tube*> SiberiusMachine)
{
    // First: Tube ID
    // Second: Number of times a ball ended in this tube
    unordered_map<int, int> endings;
    for (auto&& IDToTube : SiberiusMachine)
    {
        ++endings[RollBall(IDToTube.second)];
    }

    auto min = *min_element(endings.begin(), endings.end(),
        [](pair<int, int> a, pair<int, int> b) {
            return a.second > b.second;
        }
    );

    Screen.Show("The tube in which most balls ended is tube " +
to_string(min.first) + ", with a total of " + to_string(min.second) + "
balls.\n");
}
```

Nesse método temos um dicionário de finais do tipo **<int, int>**, onde o primeiro valor representa o ID de um tubo e o segundo valor representa a quantidade de bolinhas que saíram por aquele tubo. O primeiro **for** itera pela máquina do Sr. Sibério (um mapa com todos os tubos e seus IDs) e calcula a saída de cada tubo utilizando o método **RollBall**, cuja saída (o ID do tubo resultante) é a chave do dicionário **endings**, cujo valor é pré-incrementado. Após esse laço, a variável **min** recebe o mínimo elemento do mapa **endings**, dado por uma função lambda - que compara o número de vezes que uma bolinha terminou em um tubo - descrita na própria chamada da função **min_element**. **min** é do tipo **std::pair<int, int>**, onde o primeiro valor é o ID de um tubo e o segundo valor é a quantidade de vezes que uma bolinha saiu por aquele tubo.

O método **RollBall** recebe um tubo e calcula onde que uma bolinha sairá quando inserida nesse tubo. O valor de retorno é o ID do tubo resultante.

```
int RollBall(Tube* tube)
{
    int TubeSize = tube->TubeSize(), originalID = tube->ID;
    for (int i = 0; i <= TubeSize; ++i)
    {
        Screen.Info("Tube [" + to_string(tube->ID) + "]: " + to_string(i));
        if (tube->PosToTube.find(i) != tube->PosToTube.end())
        {
            int oldID = tube->ID;
            auto newTubePair = tube->PosToTube[i];
```

```
            tube = newTubePair.first;
            i = --newTubePair.second;
            Screen.Info("Tube " + to_string(oldID) + " on position " +
to_string(i) + ": " + "Redirecting to tube " + to_string(tube->ID));
        }
    }

    Screen.Info("Tube " + to_string(originalID) + ": Ball ended on tube " +
to_string(tube->ID) + ".");
    return tube->ID;
}
```

Resultados dos casos de teste

Caso de Teste	Número de Tubos	Tubo Mais Comum	Número de Bolinhas
1	20	7	5
2	50	9	13
3	100	86	36
4	200	149	86
5	500	211	192
6	1000	130	365
7	2000	811	710
8	5000	4754	917

Em cada caso de teste, o programa executou e retornou o tubo que saíam mais bolinhas, com o número de bolinhas que saíram naquele tubo na coluna ao lado na tabela.

No caso de teste 8, com 5000 tubos, o tubo mais comum é o tubo 4754, com um total de 917 bolinhas saindo por esse tubo.