Manual de Dicas - Maratona de Programação

Prof. Julio Fernando Lieira
Fatec Lins

1. Entendendo as Partes de um Problema

Um problema é composto basicamente de 5 partes: identificação, uma historinha, Entrada, Saída e um exemplo. Considere o exemplo mostrado na Figura 1:

Identificação

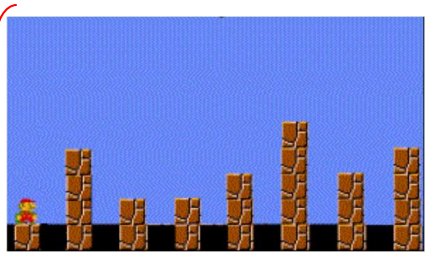
Problema C

Jumping Mario

(Problema 11764 do UVa)

Nome do arquivo fonte: `mario.c`, `mario.java`, `mario.cpp`, *ou* `mario.pas`

historinha



Mário está no final do castelo. Ele agora precisa saltar sobre alguns muros e então entrar na Câmara de Koopa, onde ele deve vencer o monstro e salvar a princesa. Para este problema, nós estamos apenas interessados na parte "saltar sobre alguns muros". Você receberá as alturas de *N* muros que estão posicionados da esquerda para a direita. Mário está posicionado no primeiro muro. Ele precisa pular para os muros seguintes, um após o outro, até atingir o último. Isto é, ele fará um total de (*N*-1) saltos.

Um *salto alto* é aquele em que Mário tem de saltar para um muro mais alto do que aquele em que está posicionado e, analogamente, um *salto baixo* é aquele onde Mário tem de saltar para um muro mais baixo do que aquele onde ele está. Você poderia determinar o número total de *saltos altos* e de *saltos baixos* que Mário precisa fazer?

Entrada

Entrada

A primeira linha da entrada é um inteiro T ($T < 30$) que indica o número de casos de teste. Cada caso inicia com um inteiro N ($0 < N < 50$) que determina a quantidade de muros. A próxima linha dá a altura dos N muros, da esquerda para a direita. Cada altura é um inteiro positivo igual ou inferior a 10.

Saída

Saída

Para cada caso de teste, imprima o número do caso de teste seguido por 2 inteiros, o total de saltos altos e o total de saltos baixos respectivamente. Observe o exemplo para o formato exato da saída.

Exemplo

Exemplo de Entrada	**Saída para o exemplo de entrada**
3 8 1 4 2 2 3 5 3 4 1 9 5 1 2 3 4 5	Case 1: 4 2 Case 2: 0 0 Case 3: 4 0

Figura 1: Exemplo de problema de maratona de programação.

1.1. Identificação do Problema

O problema é identificado por uma letra (**Problema C**), a qual será usada para submeter a solução ao sistema de correção automática. Também recebe um nome (**Jumping Mario**). Depois é indicado o nome do autor ou a fonte de onde foi retirado o problema (***Problema 11764 do UVa***). Por último é indicado qual o nome que deve ser dado ao arquivo do código fonte,

dependendo da linguagem de programação escolhida para a solução. Neste exemplo, as opções são:

- **mario.c** para solução em linguagem C;
- **mario.java** para solução em linguagem Java;
- **mario.cpp** para linguagem C++;
- **mario.pas** para linguagem Pascal.

1.2. Historinha

Em geral o problema é descrito através de uma historinha. É importante ler atentamente para entender o problema. Neste exemplo, note que a historinha define os conceitos de *Salto alto* e *Salto baixo*, os quais são fundamentais para a implementação da solução do problema.

1.3. Entrada

Descreve como deve ser a entrada dos dados, ou seja, o que seu programa receberá de entrada de dados. Aqui temos uma particularidade dos problemas de Maratona de Programação, onde os programas são corrigidos automaticamente por um sistema. Assim, na entrada de dados não temos interação com o usuário, ou seja, não devemos exibir mensagens do tipo "Digite um valor:" ou "Entre com um Nome:". Mais adiante veremos como fazer isso.

Para entender bem a entrada de dados, é preciso acompanhar a descrição das entradas seguindo o exemplo de entrada mostrado no final do enunciado do problema. Faça anotações no próprio exemplo (veja a Figura 2).

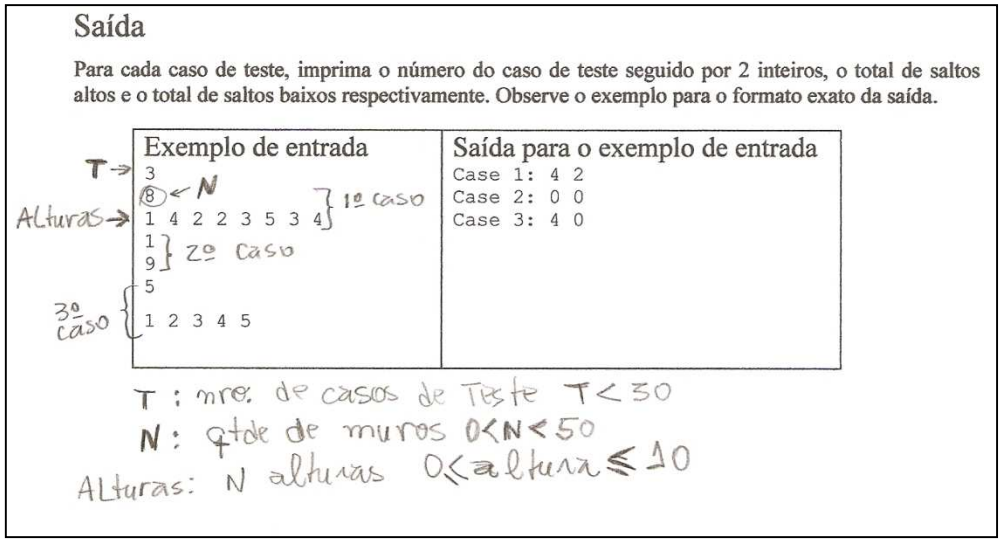


Figura 2: Anotações sobre a entrada de dados do problema Jumping Mario.

1.4. Saída

Descreve como deve ser a saída de dados do programa. Assim como na entrada, analise o que é descrito aqui observando o exemplo. É importante seguir rigorosamente as recomendações aqui descritas, até mesmo na quantidade de espaços entre os dados mostrados, pois o sistema de correção automática vai realizar uma bateria de testes e comparar a saída de seu programa com a saída correta. Mesmo pequenas diferenças no número de espaços podem invalidar a saída.

2. Implementando a Solução

Após entender o problema, partimos para a estratégia de solução do problema. O algoritmo abaixo mostra uma possível solução.

Leia a quantidade de casos de teste
Para cada caso de teste, faça:
Leia a quantidade de alturas
Leia a primeira altura (onde Mario está inicialmente posicionado)
Leia as próximas alturas e, Para cada altura lida, Compare com a altura anterior,

- se for maior, significa que Mario deve dar um Salto Alto, então incremente o contador de saltos altos;
- Se for menor, significa que Mario deve dar um Salto Baixo, então incremente o contador de saltos baixos;

Após ler todas as alturas do caso de teste atual, imprima os contadores conforme descrito pelo enunciado do problema.

2.1. Codificando

A Figura 3 mostra uma implementação da solução feita em Linguagem C. Digite-a em um editor de textos e salve com o nome **mario.c**, conforme foi indicado na parte de identificação do problema.

```
#include<stdio.h>

int main(){
  int casosTesteT,  //T<30
  qtdeMurosN,  // 0 < N < 50
  alturas,    // 0 <= alturas <= 10
  contSaltosAltos, contSaltosBaixos, //Contadores de saltos
  muroAtual, proxMuro, i,j;

  scanf("%i\n",&casosTesteT);   //Leitura da qtde de casos de teste
  for (i=1;i<=casosTesteT;i++) {
          contSaltosAltos=0;
          contSaltosBaixos=0;
          scanf("%i\n",&qtdeMurosN);  //Leitura qte de muros para este caso de teste
          //Leitura da altura do primeiro muro, onde Mario esta posicionado inicialmente
          scanf("%i ",&muroAtual);
          for (j=1;j<qtdeMurosN;j++) {//Leitura das demais alturas
                  scanf("%i ",&proxMuro);
                  if (proxMuro>muroAtual)
                          contSaltosAltos++;
                  if (proxMuro<muroAtual)
                          contSaltosBaixos++;

                  //Troca valores para a prox. leitura
                  muroAtual = proxMuro;
          }
          //Imprime a saida deste caso de teste
          printf("Case %i: %i %i\n",i,contSaltosAltos,contSaltosBaixos);
  }
  return 0; //Finaliza dizendo que o programa executou sem erros
}//Fim main()
```

Figura 3 - Código em Linguagem C da solução do problema.

2.2. Compilando

Caso esteja usando um ambiente de desenvolvimento integrado (IDE) como o Dev-C++ ou CodeBlocks, basta utilizar o compilador embutido no ambiente. Entretanto, é recomendado utilizar o mesmo compilador do sistema de correção BOCA, o qual executa na plataforma Linux. No Linux, o compilador mais usado é o GCC. Para a plataforma Windows existe uma versão do gcc que é instalada juntamente com o ambiente de desenvolvimento Dev-Cpp ou o CodeBlocks. Assim, mesmo estando na plataforma Windows é possível compilar o programa usando o gcc diretamente. Para isso, verifique na pasta onde foi instalado o Dev-Cpp a existência da parta bin e dentro dela a existência do programa gcc. Caso exista, verifique se o caminho para esta pasta está configurada na variável de ambiente PATH do sistema. Isso é feito acessando: Iniciar>Painel de Controle>Sistema>Configurações Avançadas do Sistema. Na janela que se abre, clique na aba Avançado e depois no botão Variáveis de Ambiente. No quadro Variáveis do Sistema procure pela variável PATH e edite-a. No final acrescente um ponto e vírgula e o caminho completo para a pasta bin do Dev-Cpp de seu sistema (Figura 4).

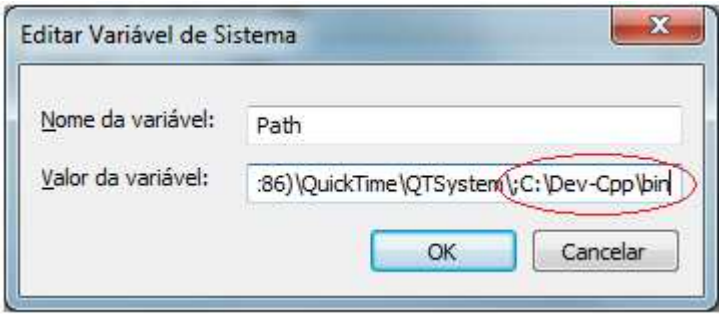


Figura 4 - Configuração da Variável de sistema Path no Windows7.

Para verificar se está funcionando, abra um prompt de comando DOS e digite gcc. A saída deverá ser como na Figura 5.

Figura 5 - Testando a configuração do caminho do gcc no Windows7.

Para compilar o programa **`mario.c`** com o gcc no Windows (vale também para o Linux), digite:

```
gcc mario.c -o mario
```

onde:

`gcc`: é o nome do compilador;
`mario.c`: é o nome do arquivo com o código fonte que será compilado;
`-o mario`: indica o nome que será dado ao arquivo do código executável.

Se não houver erro de compilação, será gerado um arquivo chamado **mario**, o qual é o código executável. Nos ambientes Windows é comum acrescentar a extensão **.exe** aos arquivos executáveis, portanto, mesmo que não tenhamos colocado a extensão, o Windows vai acrescentar, veja na Figura 6.

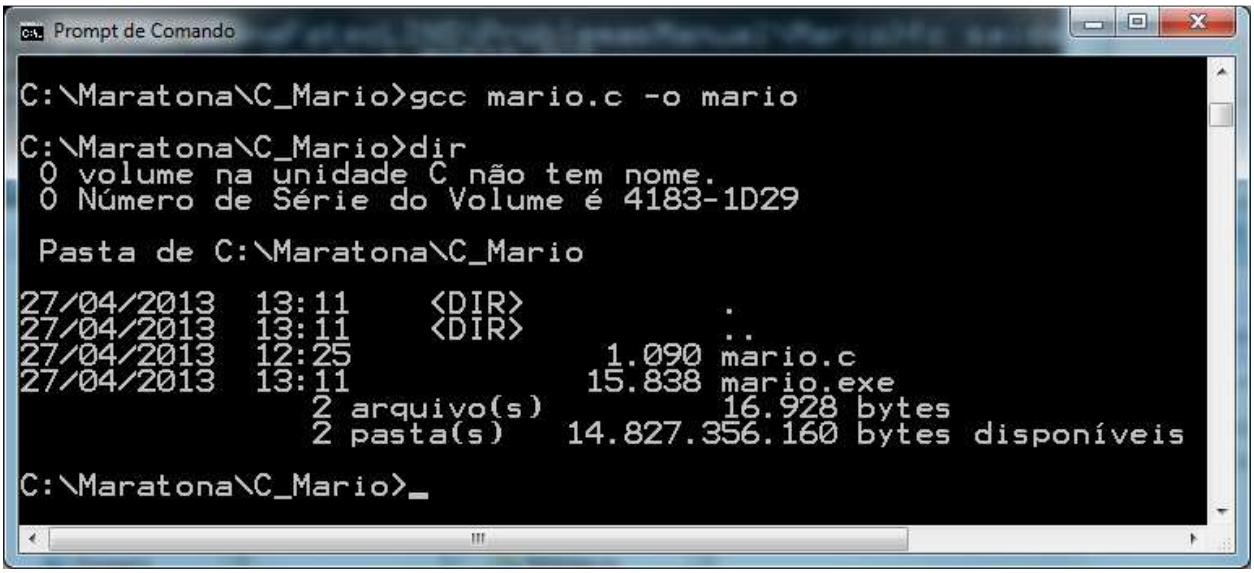


Figura 6 - Compilando o programa mario.c.

2.3. Executando

Para executar o programa na linha de comando, é preciso criar um arquivo texto com as entradas, no formado definido pelo enunciado do problema. Para o exemplo do problema Jumping Mario, digite em um editor de textos (ou copie os dados de entrada do enunciado do problema) e salve com o nome **mario.in** na mesma pasta onde está o executável.

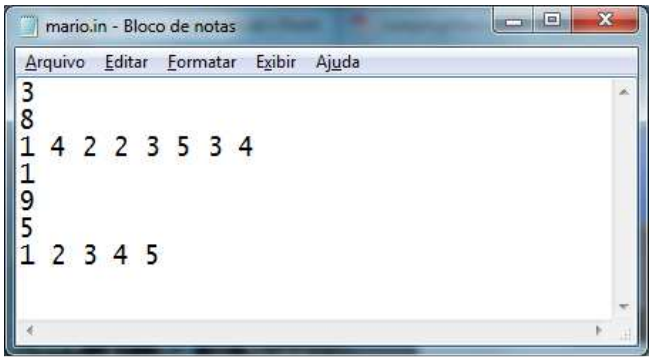


Figura 7 - Arquivo de entrada de dados para teste do problema Jumping Mario.

No prompt de comandos execute o programa da seguinte forma:

```
mario < mario.in
```

onde:

mario: é o nome do executável do programa (não é necessário a extensão .exe);
< : é o redirecionador de entrada;
mario.in: é o nome do arquivo de entrada de dados.

O comando vai executar o programa e fazer toda a entrada de dados do arquivo **`mario.in`**. Veja a saída na Figura 8:

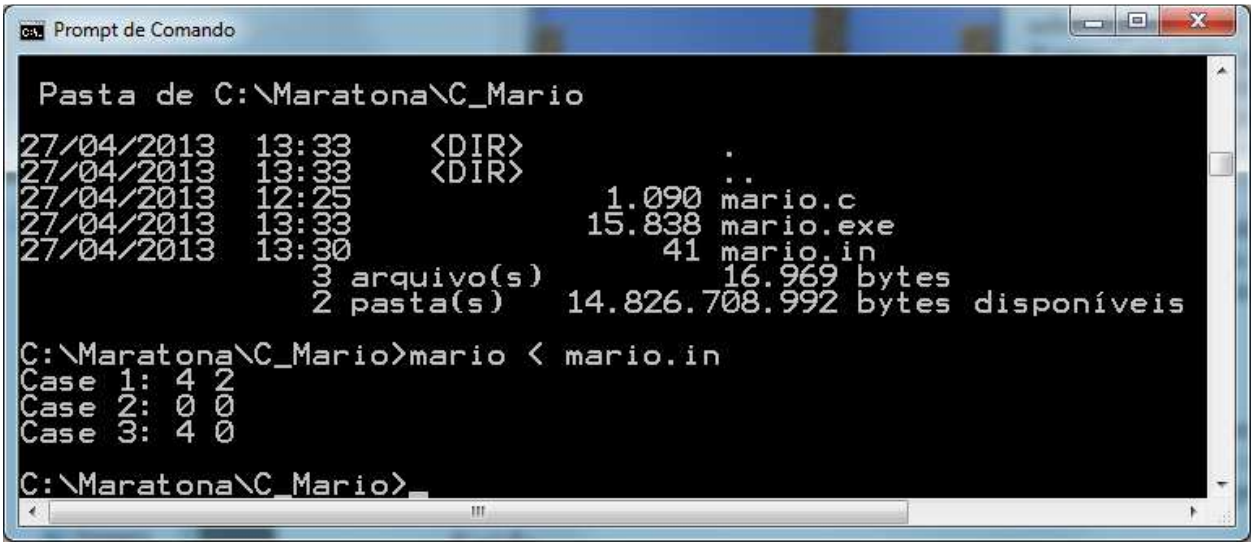


Figura 8 - Executando o programa com redirecionamento da entrada.

Note que aqui utilizamos exatamente os dados de entrada do enunciado do problema, e a saída corresponde à saída esperada do exemplo. Entretanto, é importante acrescentar mais testes, principalmente aqueles que testam os valores extremos do problema.

2.4. Verificando a Saída

A saída do programa acima é pequena e fácil de verificar se está correta. Porém, em alguns casos a saída pode ser difícil de ser verificada a olho nu. Nestes casos, podemos utilizar uma ferramenta do sistema: o comando **fc** do Windows e o **diff** do Linux. Para tanto, primeiro é preciso criar um arquivo texto com a saída esperada para a execução do programa para os casos de testes. Assim, do mesmo modo como feito com o arquivo de entrada, digite no editor de texto a saída e grave na mesma pasta do executável do programa com o nome **`mario.out`** (Figura 9).

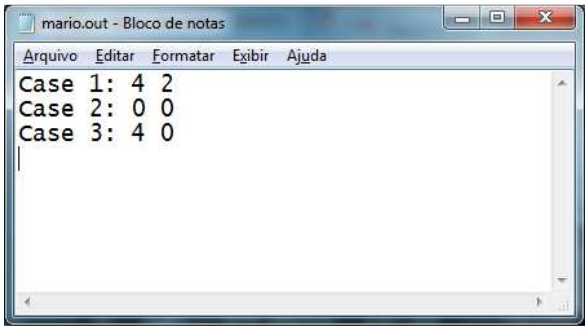


Figura 9 - Arquivo de saída para os testes do problema Jumping Mario.

Execute novamente o programa com o arquivo de entrada e gere um arquivo com a saída, digitando o seguinte comando:

```
mario < entrada > saida
```

Note a presença do redirecionador de saída (**>**), o qual envia a saída da execução do programa para um arquivo chamado **`saida`**. Estando no ambiente Windows, execute o comando **`fc`** para fazer a comparação da saída da execução do programa (arquivo **`saida`**) com a saída esperada (arquivo **`mario.out`**). Como pode ser visto na Figura 10, o comando FC não encontrou diferença entre os arquivos comparados, o que mostra que a resposta gerada pela execução do programa é idêntica à resposta esperada para os testes executados.

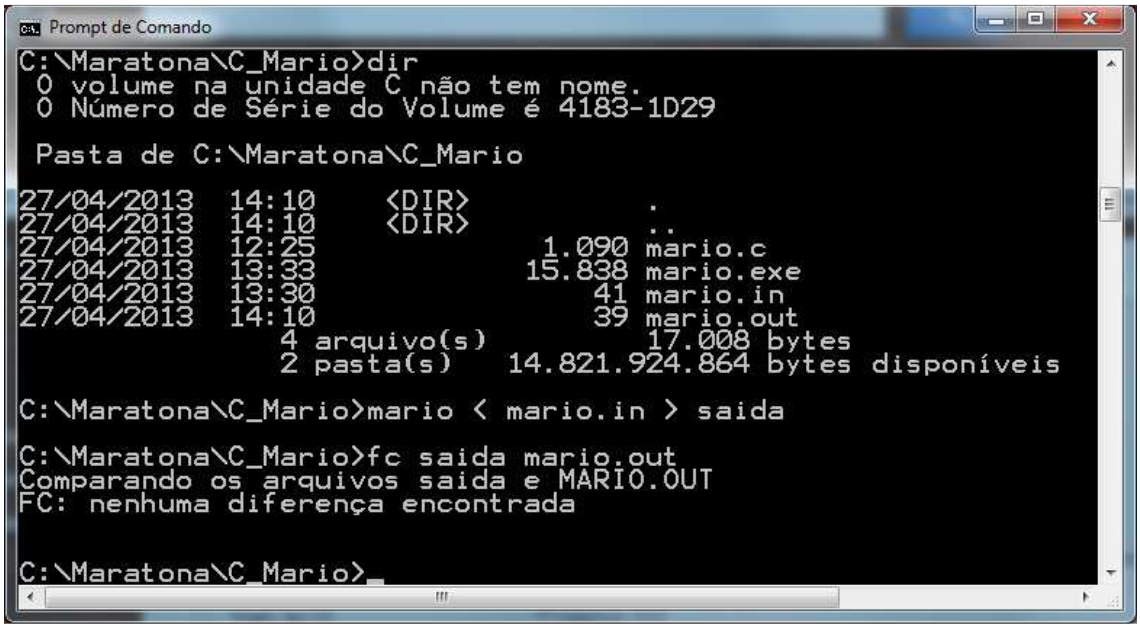


Figura 10 - Comparando a saída do programa com o arquivo de respostas com o comando FC.

3. Sistema de Correção Automática

O sistema de correção automática utilizado para a maioria das competições de Maratona de Programação no Brasil é o BOCA (http://www.bombonera.org). Através dele podemos administrar toda a competição. Cada time recebe um usuário e senha para login no sistema, que permite submeter os programas para correção, além de ter acesso ao placar e a outras funcionalidades, conforme mostrado na Figura 11.

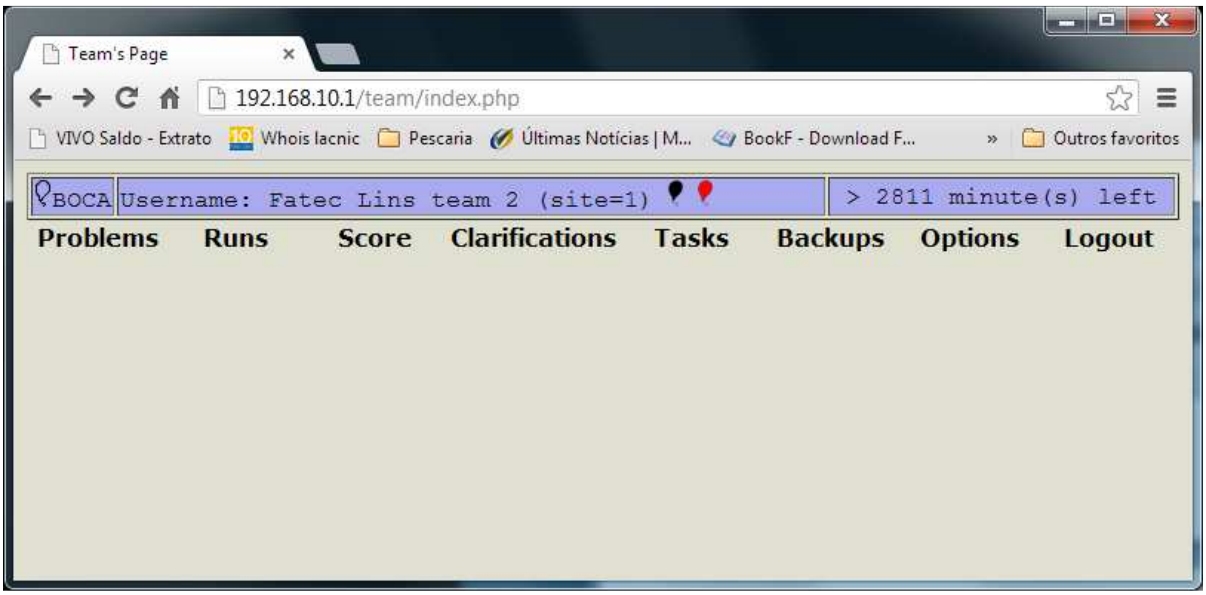


Figura 11 - Interface principal do BOCA.

Na interface principal estão disponíveis as seguintes opções:

- **Problems**: mostra a descrição dos problemas e as cores dos balões (Figura 12);

Figura 12 - Opção **Problems**.

- **Runs**: mostra todas as submissões feitas pelo time e os resultados (Figura 13);

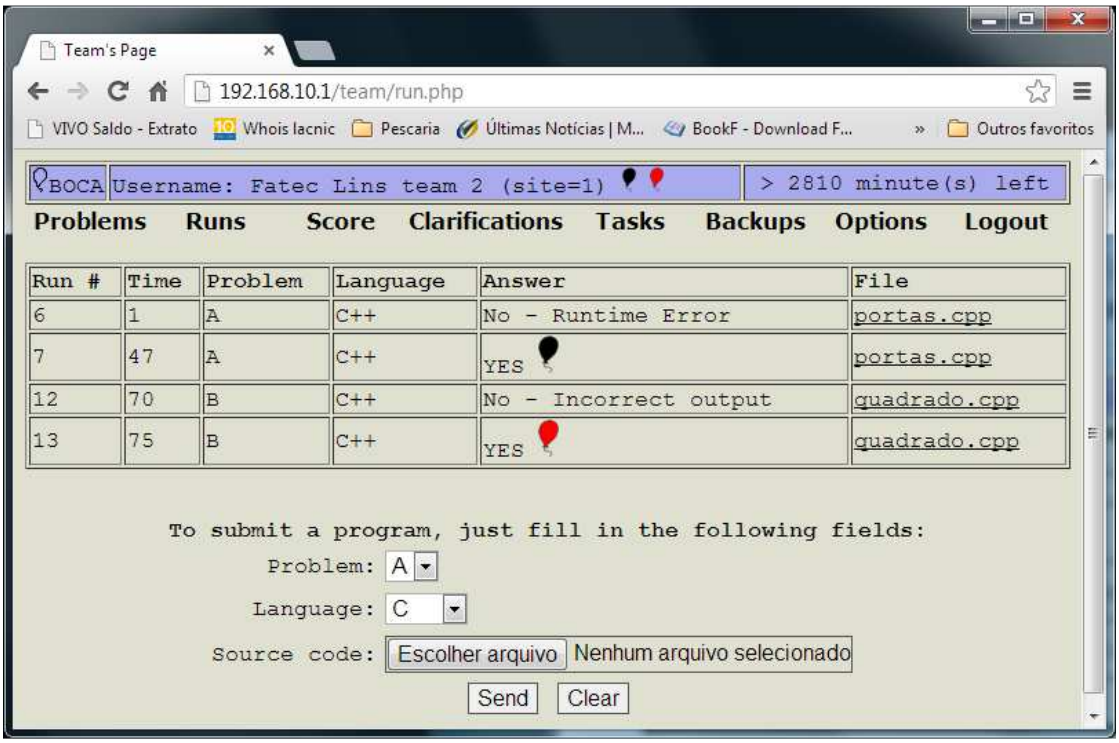


Figura 13 - Opção **Runs**.

- **Score**: mostra o placar da competição (Figura 14);
- **Clarifications**: interface para fazer perguntas aos juízes (Figura 15);
- **Tasks**: permite enviar um arquivo para ser impresso ou solicitar a presença de alguém do staff (Figura 16);
- **Backups**: permite fazer backup de seus programas no servidor (Figura 17);
- **Options**: aqui é possível trocar a senha de acesso ao sistema (Figura 18).

Figura 14 - Opção **Score**.

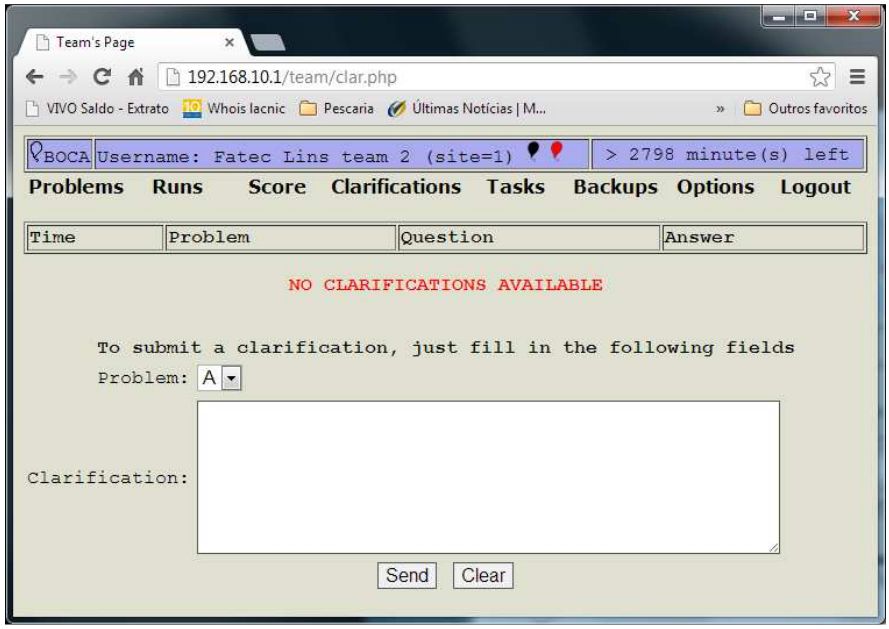


Figura 15 - Opção **Clarifications**.

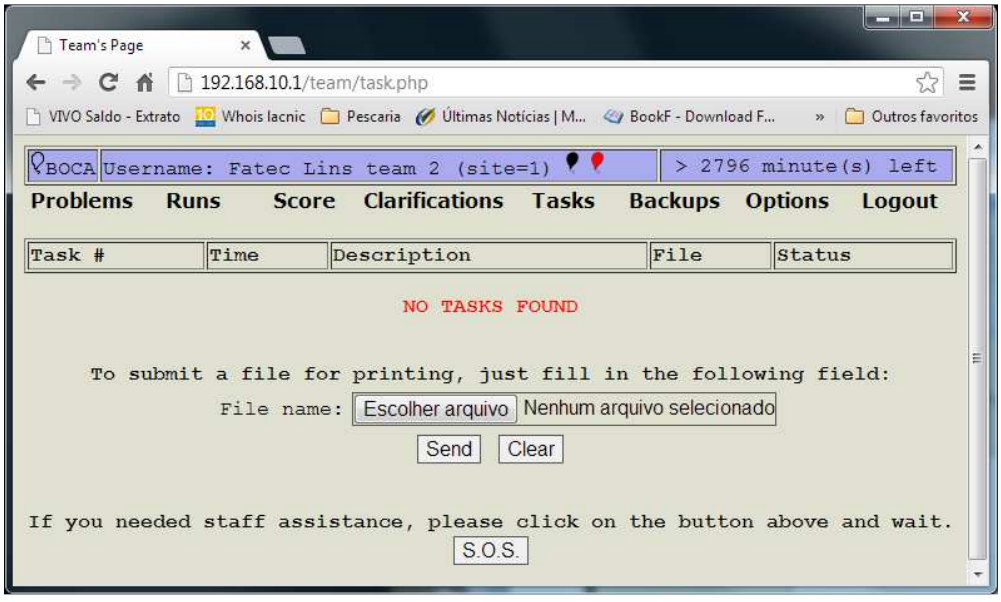


Figura 16 - Opção **Tasks**.

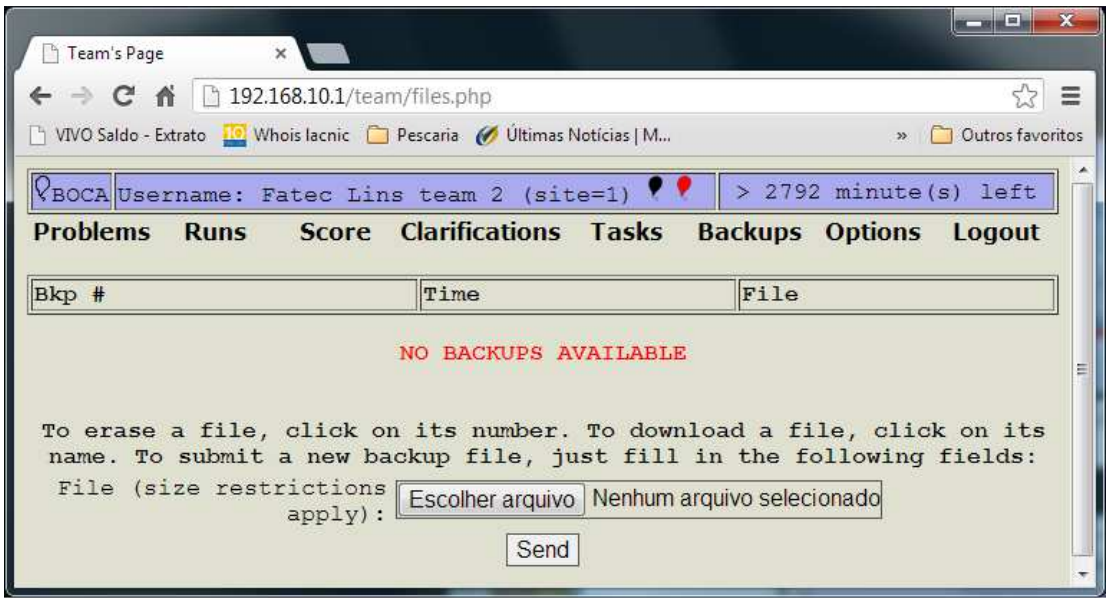


Figura 17 - Opção **Backups**.

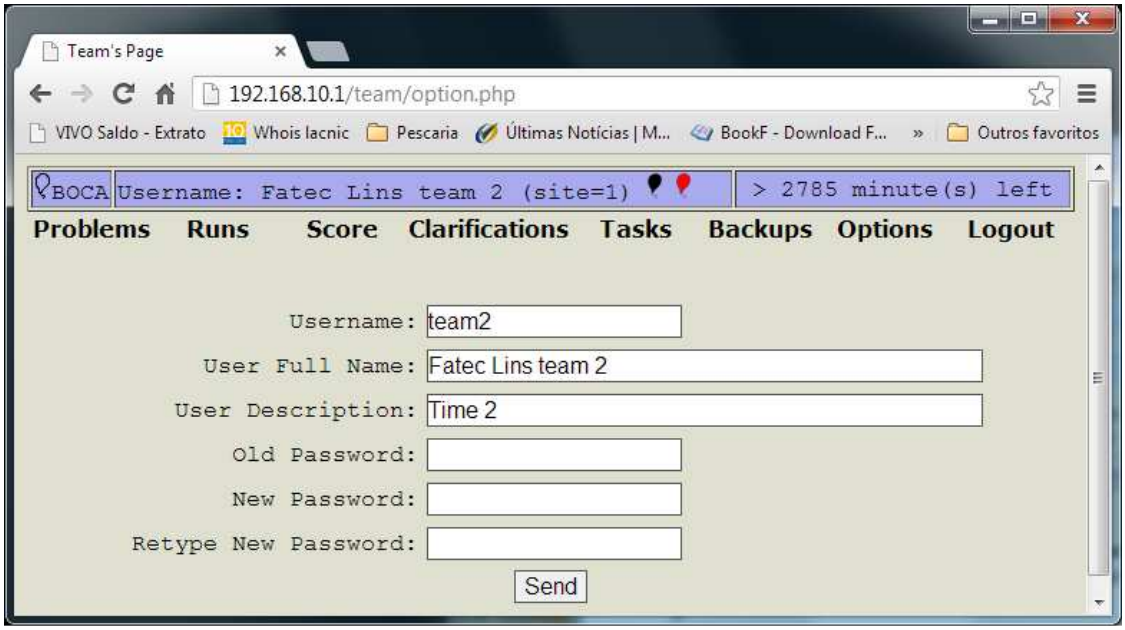


Figura 18 - Opção **Options**.

3.1. Submetendo a Solução

Após login no sistema, o time pode enviar uma solução para correção clicando na opção **Runs**. Na interface selecione a letra do problema, a linguagem de programação utilizada e escolha o arquivo fonte (neste exemplo, mario.c) e clique no botão Send (Figura 19). Após submeter a solução, a interface mostrará o status (Answer) como Not answered yet até que um juiz analise a correção e emita uma resposta (Figura 20). Na Figura 21 é mostrado o novo status após a correção. Neste caso a solução recebeu um YES, significando que a solução está correta e foi aceita (mais um balão para a equipe).

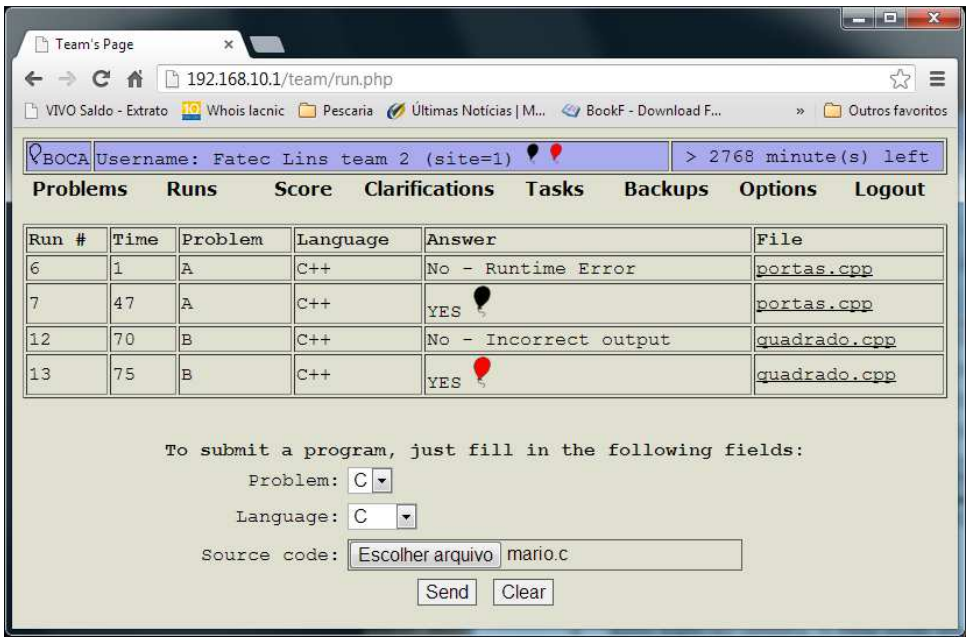


Figura 19 - Submetendo uma solução.

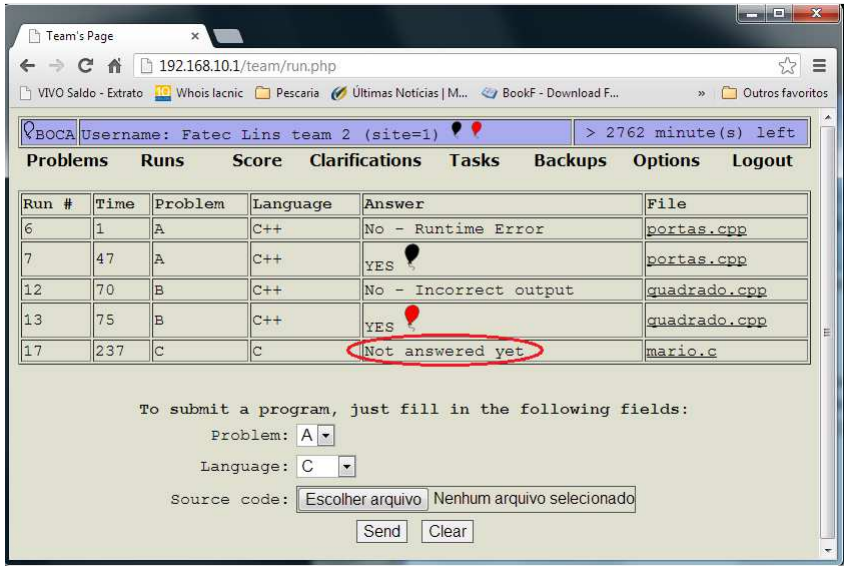


Figura 20 - Status após submissão da solução.

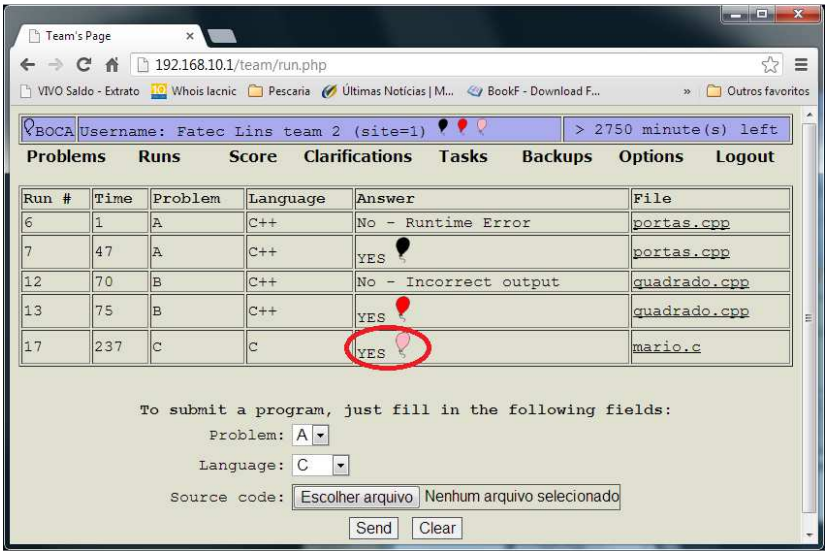


Figura 21 - Status após correção pelo juiz.

3.2. Entendendo o Score

Além da classificação geral das equipes na competição, o Score também mostra detalhes do desempenho de cada equipe, e outras informações importantes. No canto superior direito é mostrado o tempo restante da competição em minutos (> 2713 minute(s) left). Nas colunas que mostram os problemas resolvidos por cada equipe, também podemos ver a quantidade de tentativas feitas e o tempo da tentativa cuja solução foi considerada correta. Por exemplo, na Figura 22 podemos ver que a equipe team2 obteve sucesso na solução do problema B na segunda tentativa feita após decorridos 75 minutos de competição (2/75).

Lembrando que a cada submissão incorreta a equipe é punida com 20 minutos, o tempo total mostrado na última coluna é calculado somando-se o tempo de cada submissão mais 20 minutos por tentativa incorreta. Por exemplo, a equipe team2 resolveu 3 problemas num tempo total de 399 minutos (3 (399)). Neste caso, o cálculo do tempo (399 minutos) foi feito da seguinte forma:

- Problema A (2/47): 47 minutos + 20 minutos de punição por uma tentativa incorreta;
- Problema B (2/75): 75 minutos + 20 minutos de punição por uma tentativa incorreta;
- Problema C (1/237): 237 minutos;

TOTAL: 359 minutos + 40 minutos de punição = 399 minutos

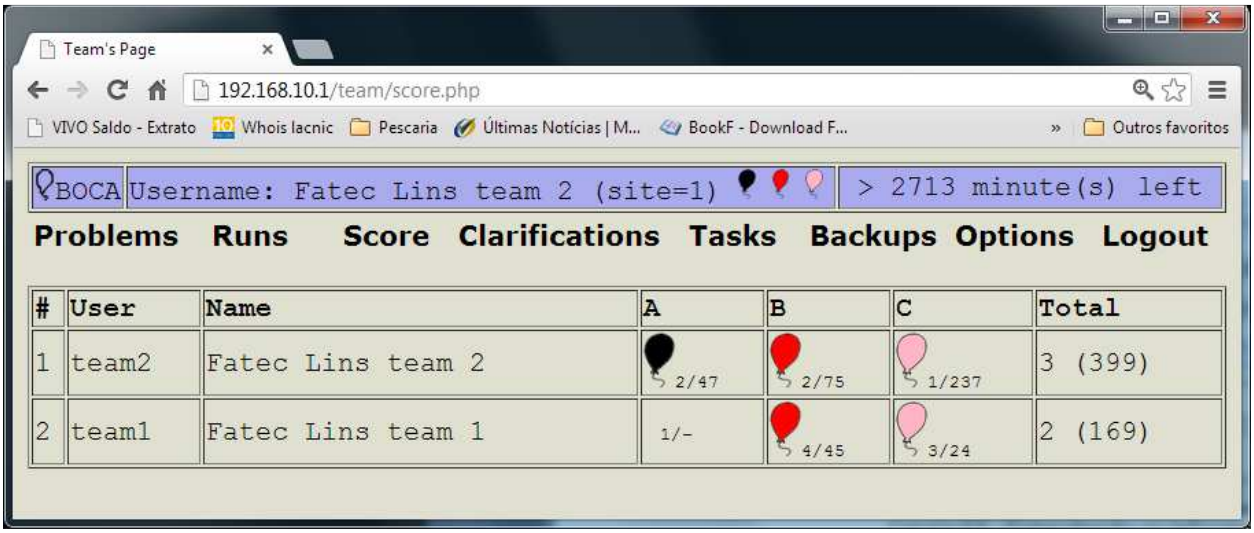


Figura 22 - Entendendo as informações do Score.

3.3. Códigos de Erros

Os possíveis códigos de erro retornados após uma submissão são:

- `Not answered yet`: aguardando a análise do juiz;
- `No - Compilation Error`: o programa não compilou. Isso se deve muitas vezes ao fato de se utilizar bibliotecas não padrões da Linguagem. Por exemplo, o competidor cria e testa o programa no ambiente Windows usando o Dev-Cpp e faz uso da biblioteca conio.h, a qual não é padrão da Linguagem C e, como o sistema de correção automática executa em ambiente Unix, o programa não compila;

- No - Runtime Error: o programa compilou, mas houve erro em tempo de execução. Embora o programa tenha executado corretamente para os casos de teste do exemplo, os juízes utilizam uma bateria de testes maior, com casos que exploram os limites extremos dos valores das variáveis. Erros típicos são: acesso a elementos inexistentes de um vetor (o vetor possui 1000 elementos e tentamos acessar o elemento 1001); erro de divisão por zero; estouro de memória, estouro de pilha do programa com chamadas recursivas excessivas, etc.;

- No - Time limit exceeded: quando o programa não é aceito por ultrapassar o limite de tempo estipulado para execução dos casos de teste. Isso acontece, por exemplo, quando algum caso de teste aplicado pelo sistema faz com que o programa entre em loop infinito;

- No - Incorrect output (ou Wrong Answer): o programa executou, mas a saída não está correta. Ou seja, para algum caso de teste na bateria de testes aplicada pelo juiz o programa está apresentando uma saída incorreta. Refaça os testes, utilizando casos de testes que explorem os valores extremos suportados pelas variáveis;

- No - Output format error (ou Presentation Error): o programa executou corretamente para a bateria de testes do juiz, porém a saída apresentada pelo programa difere do gabarito do juiz apenas no número de espaços em branco ou quebras de linhas;

- YES (ou ACCPTED): a solução foi aceita (subiu mais um balão para a equipe).