Problemas para Competições de Programação

Formatação de Problemas na Plataforma Polygon - Codeforces

Prof. Bruno Ribas - UnB/FGA
Prof. Daniel Saad - IFB/Taguatinga
Prof. Edson Alves - UnB/FGA
Prof. John Lennon - UnB/FGA

2022

Sumário

Introdução

Apresentação

- Daniel Saad Nogueira Nunes - IFB.
- Atua nos cursos superiores em Computação: BCC e LC.
- Doutor em Informática (UnB).
- Atua desde 2014, junto aos colegas do grupo Maratonas DF, na preparação de eventos de programação.

Polygon

- O Polygon é um sistema de preparação de problemas que tem suporte nativo para plataforma Codeforces
- Tanto o Polygon quanto o Codeforces foram desenvolvidos pelo russo Mike Mirzayanov
- O Polygon automatiza uma série de tarefas e faz uma gama de verificações para certificar que um problema está bem formatado
- Ele permite a realização de *contests* pela Internet através de grupos da plataforma `Codeforces`
- O sistema fornece um ambiente *web* para todas as etapas da preparação de um problema
- Por ser *web*, não é necessário instalar nenhum ambiente de desenvolvimento, de modo que tudo pode ser feito diretamente nesta interface
- Entretanto, o recomendável é fazer parte do processo *offline*, e só depois subir os arquivos gerados

Polygon: testlib

- A `testlib` é um biblioteca de apoio para a elaboração de problemas
- Ela é implementada como um único arquivo *header* do C++
- A preparação de problemas é facilitada quando a Testlib é empregada
- A `testlib` provê uma interface simples para geração e validação de testes, bem como para a elaboração de corretores customizados
- Ela também provê implementações multiplataforma de funções de geração de números aleatórios
- Seu uso não é obrigatório, mas sua ausência pode causar incompatibilidade ou mal funcionamento de alguns dos recursos da plataforma Polygon

Formatação de Problemas no Polygon

Principais funcionalidades do Polygon

Após a criação de um novo problema, as principais funcionalidades disponíveis no Polygon são:

General Info: informações gerais do problema

Statement: contextualização do problema

Files: códigos-fontes de correção, validação e geração de testes e arquivos auxiliares da formatação do texto do problema

Checker: seleção do corretor do problema e testes do corretor escolhido

Validator: seleção do validador e testes do validador escolhido

Tests: geração dos testes e seleção dos testes que aparecerão como exemplos no problema

Stresses: especificação de casos extremos de testes sobre as soluções propostas

Principais funcionalidades do Polygon

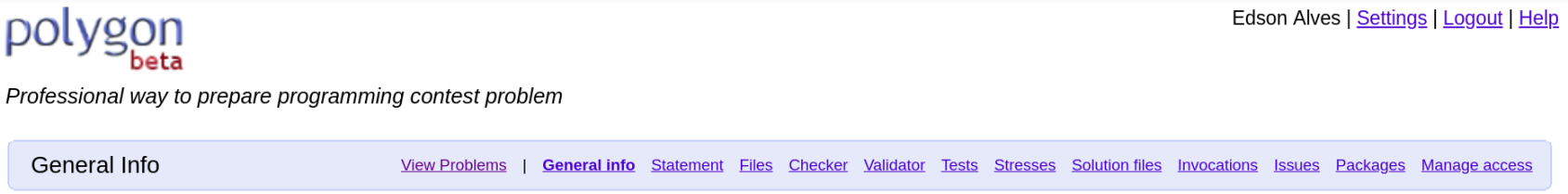
Solution files: códigos-fontes de soluções do problema

Invocations: invocação da validação de testes, seguida da execução das soluções e correção a partir dos arquivos de saída gerados

Issues: forma de registrar pendências ou melhorias que se apliquem ao problema

Packages: permite geração de pacotes do problema após a sua formatação

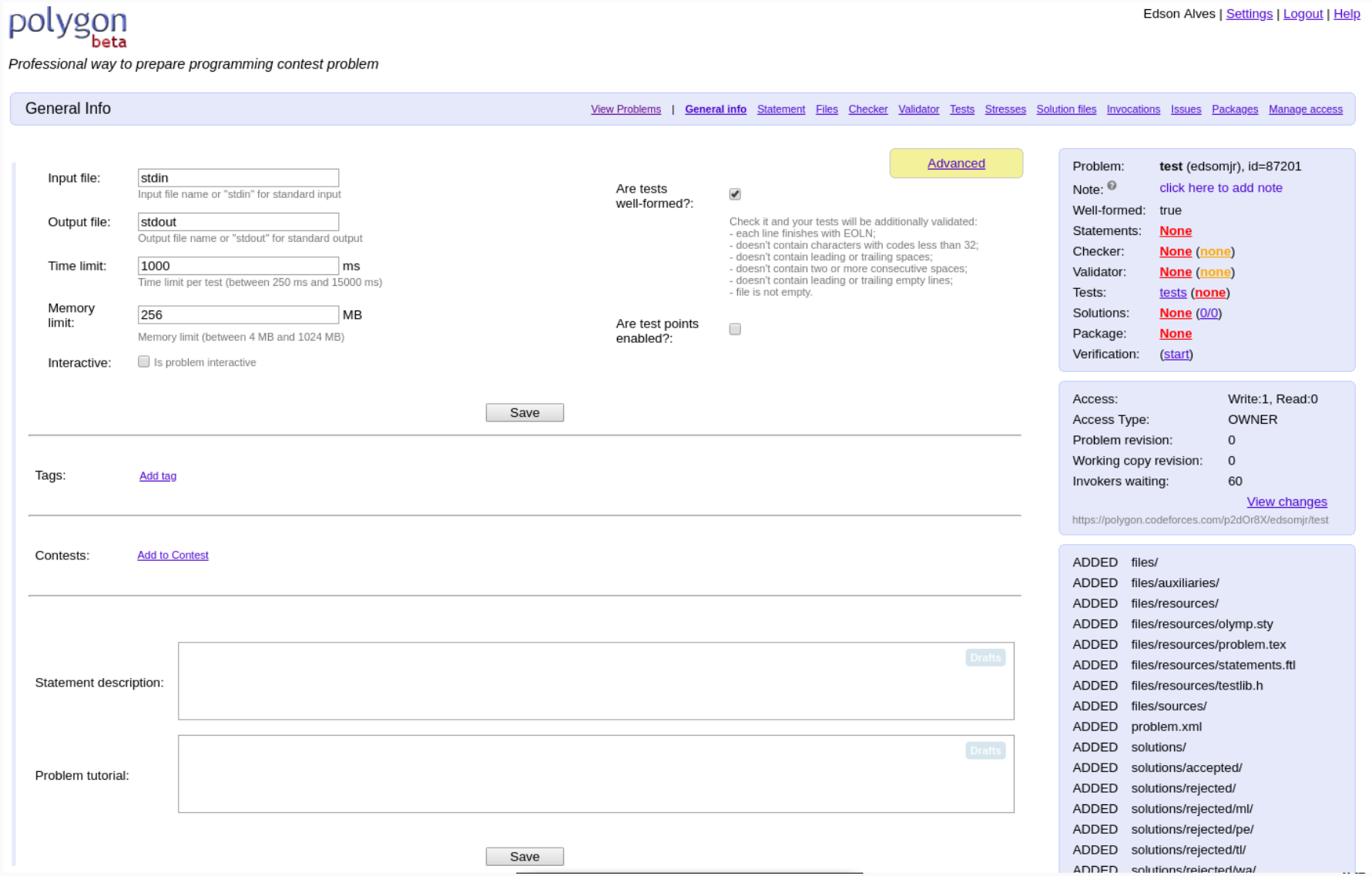
Manage access: gerencia usuários responsáveis pela leitura/escrita de um problema



Polygon: General Info

- Possibilita a edição das informações gerais dos problemas
- As informações gerais consistem de:
  - Tempo limite
  - Memória máxima
  - Arquivos de entrada e saída (stdin e stdout por padrão)
- Além disso, é possível classificar os problemas de acordo com as *tags* do Codeforces, adicionar o problema em um *contest* específico e colocar informações breves acerca do tutorial do problema
- Atenção: este *contest* não está ligado diretamente a um evento do Codeforces: é apenas uma organização lógica dos problemas

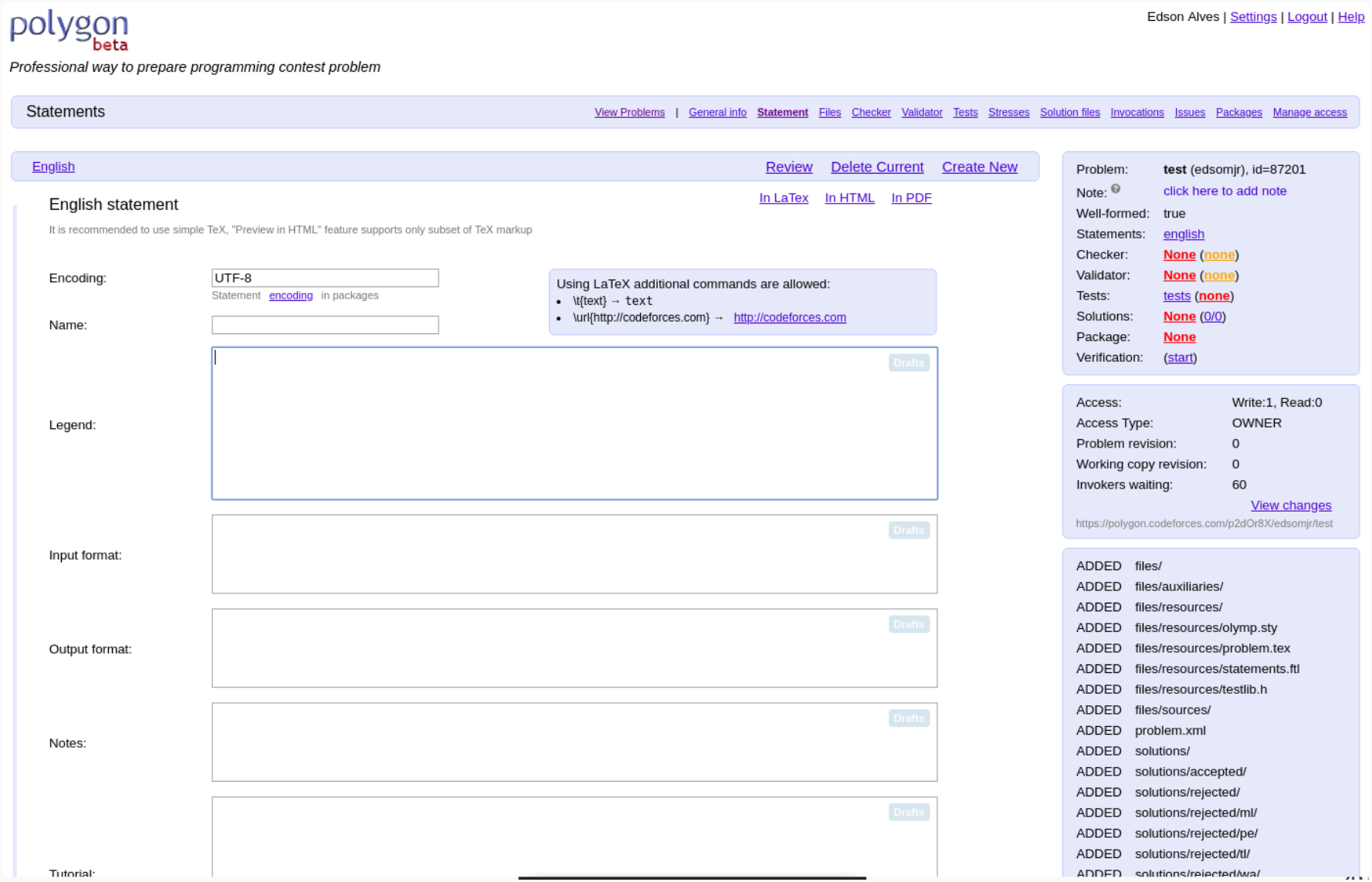
Visão do General Info após a criação de um novo problema



Polygon: Statement

- Esta tela diz respeito ao enunciado do problema, composto pela
  - contextualização do problema
  - descrição da entrada
  - descrição da saída
  - observações: normalmente utilizadas para elucidar o problema a partir dos casos de testes dos exemplos
  - tutorial
- Existe um suporte básico ao LaTeX(Mathjax), porém ambientes e comandos mais complexos não são suportados
- Boa prática: sempre escreva as observações para elucidar os exemplos e o tutorial, para que todo o material esteja disponível logo após o término do *contest*
- Importante: mesmo que o problema seja escrito em português, para que o mesmo possa ser visualizado no Codeforces é preciso utilizar a opção English

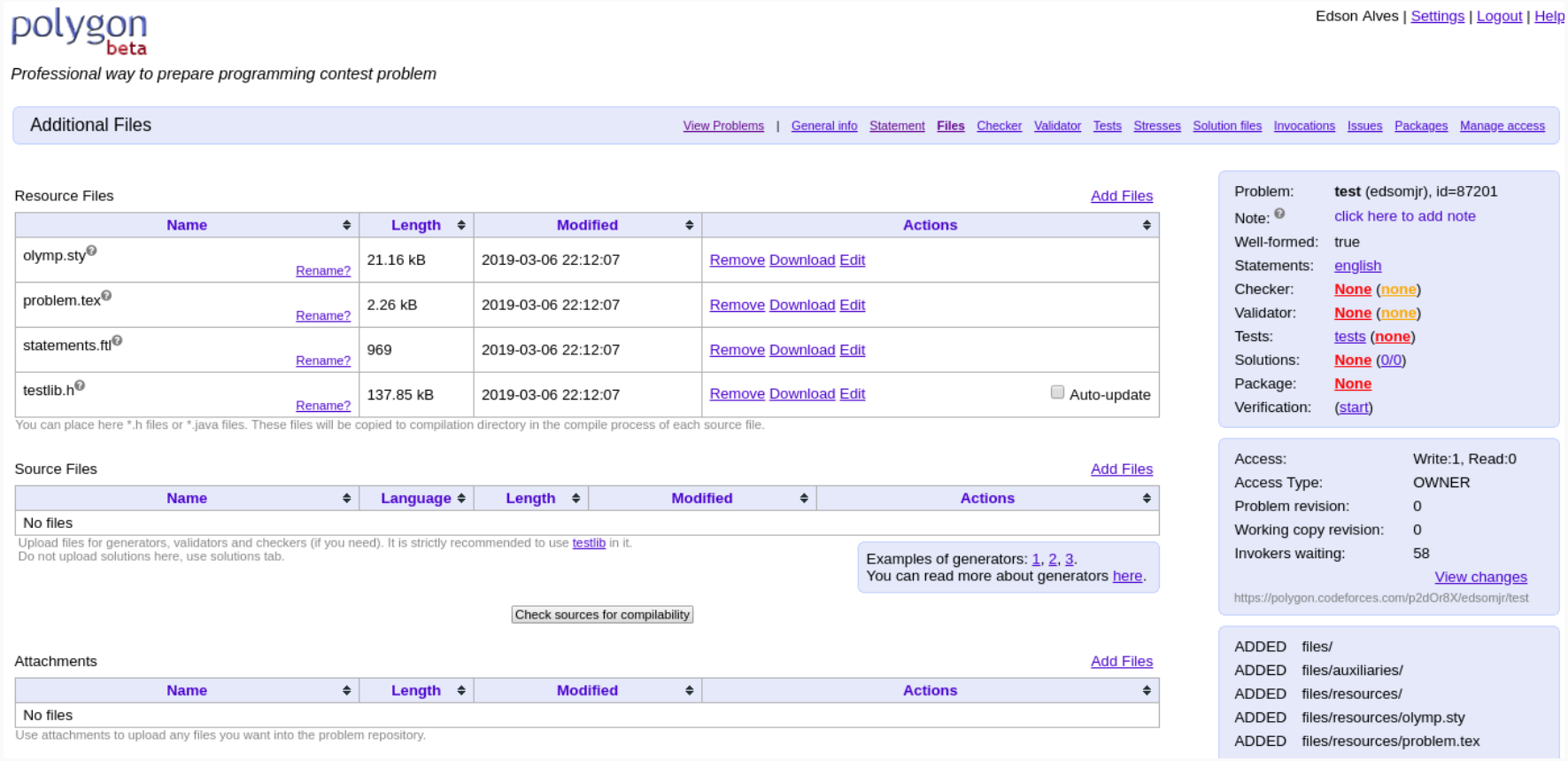
Visão do `Statement` após a criação de um novo problema



Polygon: Files

- Aqui deverão ser inseridos os códigos-fontes referentes à geração e validação de testes e à correção de submissões
- Outros arquivos de apoio ao problema também devem ser inseridos aqui
- Importante: os arquivos relacionados ao **Statement**, como figuras, devem ser inserido na própria aba **Statement**, caso contrário não serão localizados
- Não é necessário inserir arquivos de estilo para a formatação do problema, uma vez que estes são adicionados automaticamente por padrão
- A interface oferece a possibilidade de testar se os arquivos são compilados corretamente ou não no ambiente de testes do Polygon
- É importante checar esta compatibilidade para que os demais recursos de validação fiquem disponíveis

Visão da aba `Files` após a criação de um novo problema



Polygon: Checker

- O corretor é responsável por avaliar os arquivos de saída produzidos pela submissão e produzir um veredito
- O *problem setter* pode selecionar um corretor padronizado previamente disponível na plataforma `Polygon`
- Caso nenhuma das opções disponíveis sejam o suficiente para o problema, um corretor customizado pode ser selecionado nesta tela, desde que o mesmo seja incluído previamente em **Files**
- Além de selecionar o corretor a ser utilizado, é possível criar testes unitários para verificar o comportamento do corretor
- Boa prática: escreva os testes unitários para o corretor a fim de que erros sejam detectados o quanto antes
- Boa prática: teste aqui os *corner cases* do problema

Exemplo de corretor customizado: Dominó Solitário

- No problema Dominó Solitário (II Maratona de Programação do IFB), dado um conjunto de dominós, deve ser impressa a maior sequência de dominós que uma pessoa conseguiria formar utilizando as regras do jogo
- A solução não é única, logo o corretor deve verificar se
  1. a solução apresentada é compatível com as regras do jogo
  2. a solução utiliza apenas peças que estão presentes na entrada
  3. o número de peças apresentado na solução é igual ao do gabarito
- Importante: se as duas primeiras condições forem verificadas, mas o número de peças utilizadas pelo competidor é maior do que o gabarito, a solução do autor está errada!

Corretor customizado: Dominó Solitário

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

using namespace std;
const int max_number_of_pieces = 8; //maximum number of dominoe pieces
/*** This function receives stream as an argument, reads an answer
 * from it, checks its correctness with _wa outcome if stream = ouf
 * (contestant) or with  _fail outcome if stream = ans (jury).
 ***/
int check_ans(InStream& stream,set<pair<int,int>>& s) {
    // reading participant answer
    // Impossible to have dominoe chain > 8
    int n = stream.readInt(0,max_number_of_pieces);
    stream.readEoln();
    if(n > s.size()){
        stream.quitf(_wa,"WA: wrong answer, more pieces than"
                         "optimal solution");
    }
    pair<int,int> last_pair;
    for(int i=0;i<n;i++){
```

Corretor customizado: Dominó Solitário

```
21          int a,b;
22          string str = stream.readToken();
23
24          if(!isdigit(str[0]) || !isdigit(str[2]) || str[1]!='|'){
25              stream.quitf(_wa,"WA: not a valid domino piece");
26          }
27          // Read piece from output
28          a = str[0]-'0';
29          b = str[2]-'0';
30
31          pair<int,int> p,pswp;
32          p = make_pair(a,b);
33          pswp = make_pair(b,a);
34          if(s.find(p)==s.end() && s.find(pswp)==s.end()){
35              stream.quitf(_wa,"WA: invalid solution, piece "
36              "not available");
37          }
38          else{
39              // Remove piece
40              s.erase(p);
```

Corretor customizado: Dominó Solitário

```
41              s.erase(pswp);
42          }
43          if(i>0){
44              if(p.first != last_pair.second){
45                  stream.quitf(_wa,"WA: invalid solution, not a "
46                                          "domino chain");
47              }
48          }
49          last_pair = p;
50      }
51      stream.readEoln();
52      stream.readEof();
53      cout << "returning " << n << endl;
54      return n;
55  }
```

Corretor customizado: Dominó Solitário

```
56
57 int main(int argc, char* argv[]) {
58     registerTestlibCmd(argc, argv);
59     int max_n = inf.readInt(0,max_number_of_pieces);
60     inf.readEoln();
61     set<pair<int,int>> s;
62     for(int i=0;i<max_n;i++){
63         // Read input to obtain the pieces which are available
64         string str = inf.readToken();
65         ensuref(isdigit(str[0]) && isdigit(str[2]), "must be numbers");
66         int a,b;
67         a = str[0]-'0';
68         b = str[2]-'0';
69
70         // Store the piece into a set
71         pair<int,int> p,pswp;
72         p = make_pair(a,b);
73         s.insert(p);
74     }
75     // Check for jury solution
76     int jans = check_ans(ans,s);
```

Corretor customizado: Dominó Solitário

```
77      // Check for the participant solution
78      int pans = check_ans(ouf,s);
79      if (jans > pans){
80          quitf(_wa, "WA: jury has the better answer: "
81                          "jans = %d,pans = %d\n", jans, pans);
82      }
83      else if (jans == pans){
84          quitf(_ok, "AC: answer = %d\n", pans);
85      }
86      // This should not happen and can indicate a problem
87      // of the jury solution.
88      else{ // (jans < pans)
89          quitf(_fail, "FAIL: :( participant has the better answer"
90                          ": jans = %d, pans = %d\n", jans, pans);
91      }
92      return 0;
93  }
```

Polygon: Validator

- Os validadores são responsáveis por validar as entradas produzidas pelos geradores
- O Polygon sinalizará se um teste não for aprovado pelo validador
- Eles verificam se a formatação está correta (espaços, caracteres de fim de linha) e se a entrada obedece os limites e particularidades do problema
- O validador inserido na Seção **Files** pode ser selecionado neste passo
- Assim como nos corretores, é possível produzir testes unitário para verificar se os validadores estão corretos.
- As boas práticas são as mesmas citadas para os geradores: escrever testes unitários e contemplar os *corner cases*
- Dica: escreva testes com todos os possíveis erros na formatação da entrada: espaços, quebras de linha, extrapolar os limites, etc

Exemplo de validador: Rodovias

- No problema Rodovias (VI Maratona UnB de Programação), a entrada consiste em uma lista de K vértices distintos e um grafo simples não-direcionado, com peso nas arestas
- Abaixo é dado um exemplo de entrada: os comentários devem ser ignorados

```
// Exemplo de Entrada
3 3 2       // 3 vértices, 3 arestas e  lista de 2 vértices
1 3         // A lista de 2 vértices consiste dos elementos {1,3}
1 2 3       // Aresta (1,2) com custo 3
2 3 4       // aresta (2,3) com custo 4
1 3 5       // Aresta (1,3) com custo 5
```

Exemplo de validador: Rodovias

- O validador deve checar se
  1. os espaços e quebras de linha encontram-se nos lugares e nas quantidades corretas
  2. os dados lidos correspondem a um grafo simples não-direcionado (não contém *loops* e nem arestas múltiplas)
  3. os valores estão de acordo com as restrições de tamanho do problema
- Repare que, na implementação do validador, os símbolos de espaço e fim de linha devem ser lidos na quantidade e ordem que foram especificados no problema
- Este tipo de leitura e verificação é facilitado com o uso da `testlib.h`, que traz várias funções que atendem os objetivos da validação

Validator: Rodovias

```cpp
#include "testlib.h"
#include <set>

using namespace std;
using ii = pair<int, int>;

int main(int argc, char* argv[])
{
    registerValidation(argc, argv);

    auto N = inf.readInt(2, 20, "N");
    inf.readSpace();

    auto M = inf.readInt(0, N*(N - 1)/2, "M");
    inf.readSpace();

    auto K = inf.readInt(2, N, "K");
    inf.readEoln();
```

Validator: Rodovias

```
19
20      set<int> ts;
21
22      for (int i = 0; i < K; ++i)
23      {
24          auto t = inf.readInt(1, N, "t");
25          ts.insert(t);
26
27          if (i + 1 != K)
28              inf.readSpace();
29      }
30      inf.readEoln();
31
32      ensuref(K == (int) ts.size(), "All t values must be different");
33
34      set<ii> es;
35
```

Validator: Rodovias

```
36      for (int i = 0; i < M; ++i)
37      {
38          int u = inf.readInt(1, N, "u");
39          inf.readSpace();
40
41          int v = inf.readInt(1, N, "v");
42          inf.readSpace();
43
44          ensuref(u != v, "Autoloops are not allowed");
45
46          es.insert(ii(u, v));
47          es.insert(ii(v, u));
48
49          inf.readInt(1, 100000, "w");
50          inf.readEoln();
51      }
52
53      ensuref(2*M == (int) es.size(), "Multiedges are not allowed");
54
55      inf.readEof();
56  }
```

Polygon: Tests

- A aba **Tests** é responsável por gerar as entradas do problema (casos de teste) e selecionar aquelas que deverão ser públicas e, portanto, estar presentes no enunciado do problema
- Para gerar os casos de teste é necessário especificar a linha de comando a ser executada
- O executável responsável pela geração é criado a partir do código-fonte do gerador previamente incluído em **Files**
- Além dos testes gerados pelo executável, é possível inserir testes manualmente
- Após a geração dos casos de teste é possível, através de um *checkbox*, selecionar aqueles que devem constar no enunciado do problema
- Boa prática: inserir testes manuais não triviais para detectar problemas na solução do autor

Exemplo de gerador de casos de teste: Rodovias

- Considere o problema Rodovias citado anteriormente
- O gerador deve gerar os grafos de acordo com a restrição do problema
- São criadas três categorias de teste:
  1. `samples`: testes públicos
  2. `manual`: testes criados manualmente
  3. `random`: testes aleatórios
- Esta categorização é só uma forma de organização, e não é obrigatória
- Importante: a `testlib` garante que os mesmos números aleatórios serão gerados a cada execução
- Assim, se a entrada não variar os testes gerados serão sempre os mesmos

Gerador de casos de testes: Rodovias

```cpp
#include <iostream>
#include <sstream>
#include <vector>

#include "testlib.h"

using namespace std;
using ll = long long;
using ii = pair<int, int>;

vector<string> sample_tests {
    // Exemplo simples, a resposta é a aresta que une os dois vértices
    "3 3 2\n1 3\n1 2 3\n2 3 4\n1 3 5\n",
    // Exemplo simples, a resposta é o caminho mais curto
    "4 3 2\n1 4\n1 2 1\n2 3 2\n3 4 3\n",
    // Exemplo com 3 terminais, a resposta é o caminho mais curto
    "6 5 3\n2 4 5\n1 2 3\n2 3 7\n3 4 2\n4 5 11\n3 6 5\n",
};
```

Gerador de casos de testes: Rodovias

```
19
20 vector<string> manual_tests {
21     // Exemplo sem solução
22     "3 1 2\n1 3\n1 2 3\n",
23     // Exemplo simples, a resposta é o caminho mais curto
24     "4 6 2\n1 2\n1 2 4\n1 3 5\n1 4 1\n2 3 1\n2 4 5\n3 4 1\n",
25     "4 5 2\n1 2\n1 2 10\n1 3 1\n1 4 4\n2 4 6\n3 4 2\n",
26     "3 3 3\n1 3 2\n1 2 3\n2 3 4\n1 3 5\n",
27     "4 6 4\n1 2 3 4\n1 2 4\n1 3 5\n1 4 1\n2 3 1\n2 4 5\n3 4 1\n",
28     "4 5 4\n1 2 4 3\n1 2 10\n1 3 1\n1 4 4\n2 4 6\n3 4 2\n",
29     // Grafo completo
30     "7 21 7\n1 2 3 4 5 6 7\n1 3 3\n1 2 2\n1 4 8\n1 5 3\n1 6 9\n1 7 1\n2 3 6\n2 4 4
31     // Exemplo com um nó de Steiner
32     "5 7 3\n1 2 3\n1 2 10\n1 4 11\n1 5 15\n2 3 10\n2 4 5\n3 4 3\n3 5 15\n",
33     // Valores mínimos
34     "2 1 2\n1 2\n1 2 100000\n",
35 };
36
```

Gerador de casos de testes: Rodovias

```
37 vector<string> generate_test(int qtd, int N)
38 {
39     vector<string> tests;
40     vector<ii> edges;
41     vector<int> ks;
42 
43     for (int u = 1; u <= N; ++u)
44         for (int v = u + 1; v <= N; ++v)
45             edges.push_back(ii(u, v));
46 
47     for (int u = 1; u <= N; ++u)
48         ks.push_back(u);
49 
50     while (qtd--)
51     {
52         auto M = rnd.next(0, N*(N - 1)/2);
53         shuffle(edges.begin(), edges.end());
54 
```

Gerador de casos de testes: Rodovias

```
55          auto K = rnd.next(2, N);
56          shuffle(ks.begin(), ks.end());
57
58          ostringstream os;
59          os << N << " " << M << " " << K << endl;
60
61          for (int i = 0; i < K; ++i)
62              os << ks[i] << (i + 1 == K ? "\n" : " ");
63
64          for (int i = 0; i < M; ++i)
65          {
66              auto w = rnd.next(1, 100000);samp
67
68              os << edges[i].first << " " << edges[i].second << " " << w << endl;
69          }
70
71          tests.push_back(os.str());
72      }
```

Gerador de casos de testes: Rodovias

```
73
74      return tests;
75  }
76
77  vector<string> extra_test(int N)
78  {
79      vector<string> tests;
80      vector<ii> edges;
81      vector<int> ks;
82
83      for (int u = 1; u <= N; ++u)
84          for (int v = u + 1; v <= N; ++v)
85              edges.push_back(ii(u, v));
86
87      for (int u = 1; u <= N; u += 2)
88          ks.push_back(u);
89
90      int M = edges.size();
```

Gerador de casos de testes: Rodovias

```
91      int K = ks.size();
92
93      ostringstream os;
94      os << N << " " << M << " " << K << endl;
95
96      for (int i = 0; i < K; ++i)
97          os << ks[i] << (i + 1 == K ? "\n" : " ");
98
99      for (int i = 0; i < M; ++i)
00      {
01          auto w = 100000;
02
03          if (i % 2)
04              os << edges[i].first << " " << edges[i].second
05                                  << " " << w << endl;
06          else
07              os << edges[i].second << " " << edges[i].first
08                                  << " " << w << endl;
```

Gerador de casos de testes: Rodovias

```
09      }
10
11      tests.push_back(os.str());
12
13      return tests;
14  }
15
16
17  void append(vector<string>& dest, const vector<string>& orig)
18  {
19      dest.insert(dest.end(), orig.begin(), orig.end());
20  }
21
22  int main(int argc, char* argv[])
23  {
24      registerGen(argc, argv, 1);
25
26      string cmd = argc > 1 ? argv[1] : "random";
```

Gerador de casos de testes: Rodovias

```
27      vector<string> tests;
28      size_t test = 0;
29
30      if (cmd == "samples")
31          tests.insert(tests.end(), sample_tests.begin(), sample_tests.end());
32      else if (cmd == "manual")
33      {
34          tests.insert(tests.end(), manual_tests.begin(), manual_tests.end());
35          test += sample_tests.size();
36      } else
37      {
38          int N = argc > 2 ? stoi(argv[2]) : 100;
39          test += sample_tests.size();
40          test += manual_tests.size();
41
42          int qtd = N / 10;
43
44          for (int k = 20; k >= 4; k -= 2)
```

Gerador de casos de testes: Rodovias

```
44          for (int k = 20; k >= 4; k -= 2)
45          {
46              auto more = generate_test(qtd, k);
47              append(tests, more);
48              N -= qtd;
49          }
50
51          auto more = generate_test(N, 8);
52          append(tests, more);
53
54          more = extra_test(20);
55          append(tests, more);
56      }
57
58      for (const auto& t : tests)
59      {
60          startTest(++test);
61          cout << t;
62      }
63  }
```

Script de geração e seleção dos testes públicos

- Para gerar os arquivos de entrada, basta instruir a interface *web* a rodar o arquivo executável com os seguintes parâmetros:

Script:

```
generator samples > {1-3}
generator manual > {4-12}
generator random > {13-113}
```

Drafts

- Serão gerados 113 casos de teste, sendo 3 da categoria `samples`, 9 da categoria `manual` e 101 da categoria `random`
- Os testes categorizados como `samples` devem ser marcados como público manualmente
- Para cada um destes testes, clique em **Edit** e no *checkbox* **Use in statements**

Polygon: visualização da aba Tests

Tests (113) Delete Current Create Testset

Testset: tests

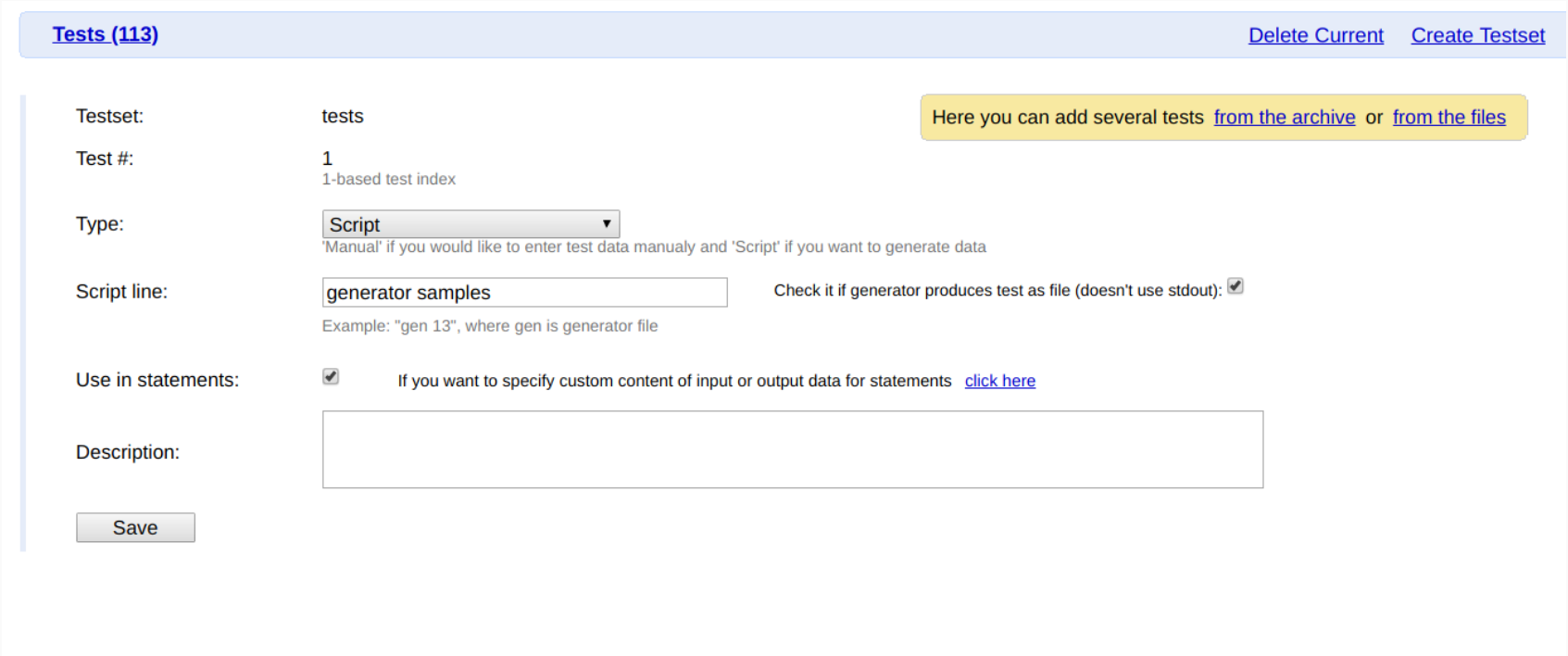
Test count: 113

Enable groups:

Tests Preview Tests Add Test

#	Content	Size	Description	In statements	Actions Delete Copy Move	
1	generator samples			Y	Delete Edit	
2	generator samples			Y	Delete Edit	
3	generator samples			Y	Delete Edit	
4	generator manual				Delete Edit	
5	generator manual				Delete Edit	
6	generator manual				Delete Edit	
7	generator manual				Delete Edit	
8	generator manual				Delete Edit	
9	generator manual				Delete Edit	
10	generator manual				Delete Edit	
11	generator manual				Delete Edit	
12	generator manual				Delete Edit	
13	generator random				Delete Edit	

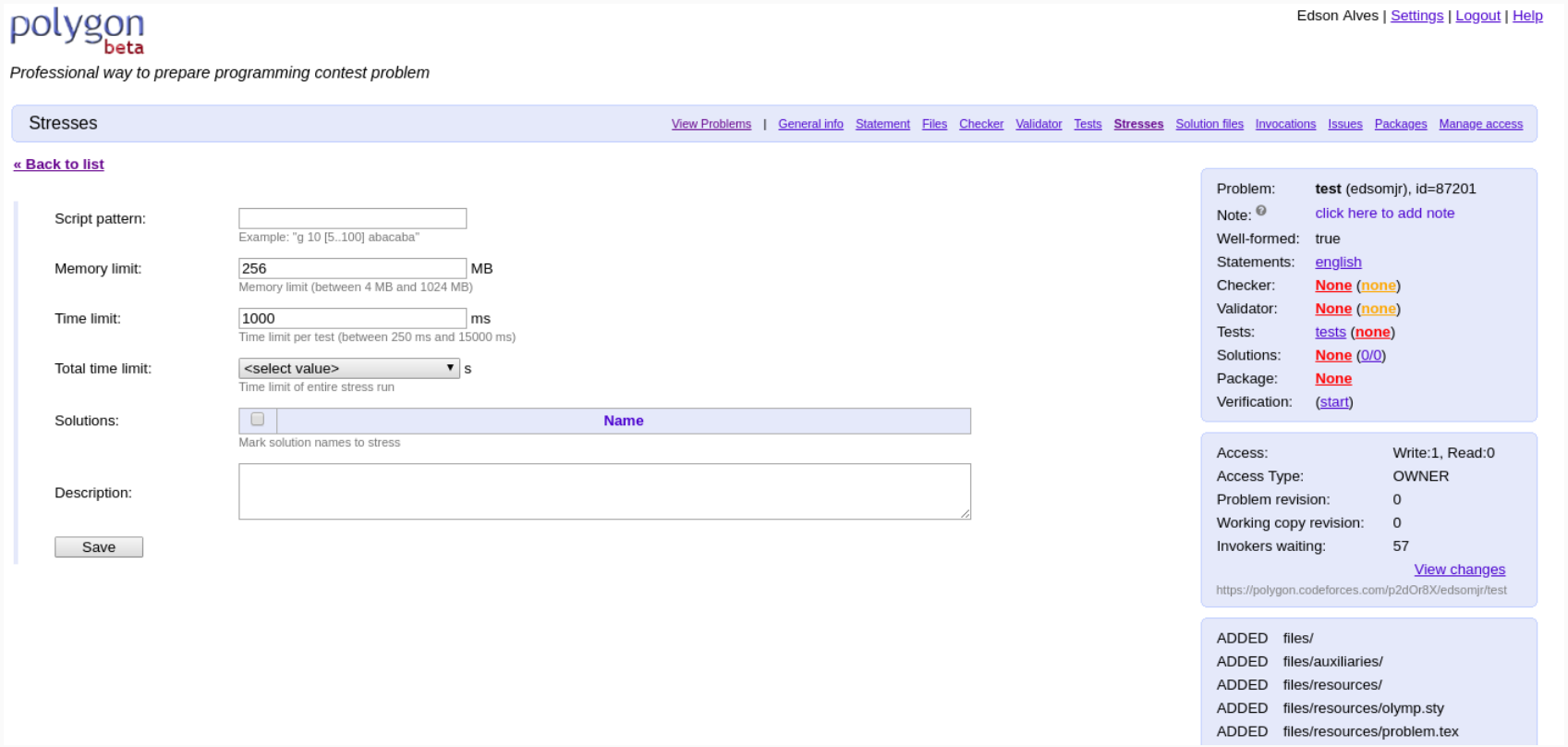
Polygon: visualização da edição de um caso de teste



Polygon: Stresses

- Em **Stresses** é possível configurar um script ara gerar casos de teste extremos para os problemas
- Devem ser especificados os parâmetros deste script e quais as soluções que devem executar os testes extremos gerados
- Outros parâmetros que podem ser configurados são o limite de memória e o tempo de execução máximo para os casos extremos
- Útil para observar o comportamento das soluções em casos extremos

Polygon: visualização da edição de teste de stress



Polygon: Solution Files

- Nesta aba é possível incluir os códigos-fontes das soluções e classificá-las de acordo com o veredito esperado
- É obrigatório classificar uma das soluções como *main correct solution*, pois a partir dela são gerados os arquivos de saídas a partir dos casos de teste
- Soluções corretas alternativas ou soluções TLE e WA também podem ser inseridas
- Boa prática: fornecer diferentes soluções corretas, TLE e WA é extremamente vantajoso, pois elas podem ajudar a detectar problemas na solução principal, fraqueza nos testes ou fraqueza nas definições de recursos do problema (tempo, memória, etc)

Exemplo de Solution Files: Presente de Natal Palindrômico

- O exemplo a seguir foi tirado do problema Presente de Natal Palindrômico (III Maratona de Programação do IFB)
- Foram inseridas uma série de soluções propostas por diversos usuários e que cada solução dessas tem uma classificação baseada no veredito esperado
- A probabilidade da solução principal estar correta aumenta significativamente se ela for compatível com as outras soluções classificadas como corretas
- Resultado: *contest* tranquilo, juízes confiantes nas *clarifications*

Exemplo de Solution Files: Presente de Natal Palindrômico

Solution files

Add Solutions

Author	Name	Language	Length	Modified	Type	Actions Delete Download Edit	
daniel.saad.nunes	ac.cpp Rename?	cpp.g++17	2.83 kB	2018-11-21 23:14:59	Correct Change?	Delete Download Edit	
j3r3mias	j3r3mias-ac.cpp Rename?	cpp.g++17	1.05 kB	2018-11-22 00:29:07	Correct Change?	Delete Download Edit	
arthurkomatsu	mattioli.cpp Rename?	cpp.g++17	1.31 kB	2018-11-22 00:45:43	Correct Change?	Delete Download Edit	
arthurkomatsu	solution.cpp Rename?	cpp.g++17	1.16 kB	2018-11-22 16:40:38	Main correct solution Change?	Delete Download Edit	
daniel.saad.nunes	tle.cpp Rename?	cpp.g++17	2.06 kB	2018-11-21 19:29:17	Time limit exceeded Change?	Delete Download Edit	
daniel.saad.nunes	wa.cpp Rename?	cpp.g++17	2.68 kB	2018-11-21 22:32:56	Wrong answer Change?	Delete Download Edit	

Upload solution files here.
There should be exactly one "Main correct solution" (also known as "model solution"). It will be used to generate jury answers.

Check solutions for compilability

Polygon: Invocations

- As invocações compilam os códigos-fontes, validam as entradas selecionadas, executam as soluções indicadas em cada uma das entradas e aplicam o corretor nas saídas produzidas
- Um erro é emitido se o resultado da execução for diferente do especificado para a solução
- As restrições de tempo e memória também são considerados durante uma invocação
- As invocações são essenciais para atestar o veredito das soluções incluídas, a formatação correta das entradas e o comportamento do corretor
- Boa prática: sempre execute as invocações de modo a atestar a formatação do problema como um todo
- Além disso, procure incluir nas invocações soluções com diferentes naturezas (AC, TLE, WA) para verificar a correção da solução principal, a robustez dos casos de teste e os parâmetros de tempo e espaço do problema

Polygon: Issues

- As *issues* fornecem um sistema de *tickets* para indicar um potencial problema ou melhoria do problema
- Os usuários com acesso ao problema podem abrir estes *tickets* e responsabilizar outros usuários para tratar as pendências
- *Tickets* podem ser classificados como: *discussion* (discussão), *enhancement proposition* (proposição de melhorias) e bug (problemas)
- Fazem parte do suporte elementar de controle de versão que o Polygon oferece
- Outras características de controle de versão integradas são os *commits* e o controle de revisões
- Boa prática: marcar os *tickets* como resolvidos sempre que os problemas forem sanados

Exemplo de Issues

Want to add new issue?

ENHANCEMENT, status CLOSED , assigned to Arthur Luis Komatsu Aroeira Change issue status?

Text: Remover testes repetidos.

By Daniel Saad Nogueira Nunes, 2018-11-21 23:19:02

Comments: Add comment or change assign?

By Arthur Luis Komatsu Aroeira, 2018-11-22 18:09:54, and changed status to CLOSED
Tests fixed

ENHANCEMENT, status CLOSED , assigned to Arthur Luis Komatsu Aroeira Change issue status?

Text: Adicionar tutorial.

By Daniel Saad Nogueira Nunes, 2018-11-22 04:17:28

Comments: Add comment or change assign?

By Arthur Luis Komatsu Aroeira, 2018-11-22 18:09:42, and changed status to CLOSED
Tutorial added

Polygon: Packages

- Esta aba é responsável por gerar os pacotes, unidades que encapsulam um problema
- Na geração dos pacotes, o problema é testado e *warnings* ou *errors* são emitidos caso o problema não esteja bem formatado
- Os pacotes permitem que problemas sejam adicionados a *contests* do Codeforces através da URL que identifica o problema
- É possível gerar os pacotes desde que os problemas estejam bem formatados
- Cada pacote está associado a uma revisão do problema, e o Codeforces usará a versão mais recente
- Caso seja feita alguma modificação no problema, uma nova revisão é criada e um novo pacote deve ser gerado
- Boa prática: não modificar os pacotes durante um *contest*, a menos que seja estritamente necessário (há um *delay* entre a modificação do pacote e a visualização da mudança por parte dos competidores)

Exemplo de pacotes para duas revisões do problema

Package problem

View Problems | General info Statement Files Checker Validator Tests Stresse

Packages

Verify: Create package: Standard, Full

#	Problem revision	Creation time	State	Comment	Download	Package size
180439	2	2018-09-23 16:30:06	READY	Package created in 71987 ms with verification	Standard	4.20 MB
179583	1	2018-09-20 01:22:02	READY	Package created in 73204 ms with verification	Standard	4.16 MB

Polygon: Manage Access

- O Polygon fornece uma forma de adicionar outros usuários no problema, possibilitando sua leitura ou modificação do mesmo
- Isso agiliza o processo de formatação e revisão dos problemas, uma vez que estes outros usuários podem propor soluções alternativas, novos testes ou modificações na contextualização do problema
- Caso um problema esteja sendo considerado para um *contest* no Codeforces, é importante adicionar o usuário `codeforces` com permissão de leitura, caso contrário, não será possível incluir o problema
- As permissões são configuradas por problema, e podem variar entre problemas distintos de um mesmo autor
- Boa prática: adicionar os outros *problem setters* no problema agiliza o processo de formatação e revisão do mesmo

Exemplo de vários usuários com permissão a um mesmo problema

Users

Add Users

Login	Name	Access Type	Reviewer	Supervisor	Translator	Add Time	Actions
codeforces	Codeforces Judge System	READ	☐	☐	☐	2018-09-20 01:21:40	Remove
daniel.saad.nunes	Daniel Saad Nogueira Nunes	WRITE	☑	☐	☐	2018-09-20 01:21:55	Remove
duerno	Felipe Duerno	WRITE	☑	☐	☐	2018-09-20 01:21:48	Remove
edsomjr	Edson Alves	OWNER	☑	☐	☐	2018-09-19 22:33:16	
gnramos	Guilherme Ramos	WRITE	☑	☐	☐	2018-09-20 01:21:55	Remove
matheusfaria	Matheus Faria	WRITE	☑	☐	☐	2018-09-20 01:21:48	Remove
viniciusrpb	Vinicius Borges	WRITE	☑	☐	☐	2018-09-20 01:21:55	Remove

Users with Reviewer access can add issue.
Users with Supervisor access will always receive email notifications about committed changes, even if "Don't send email notification" is on.
Users with READ access can start edit sessions, but can't commit changes.
Type comma-separated list of user logins in "Add User" form if you want to process more than one user.

Formatando um Problema