statements-manager

リリース *v1.7.14*

tsutaj

2023 年 09 月 11 日

Getting Started

競技プログラミングの問題文作成を支援するツール

第 1 章

ツールの概要

statements-manager (略称: ss-manager) は、Markdown 形式で記述された競技プログラミングの問題文ファイルを、HTML / PDF / Markdown 形式のいずれかに変換して出力します。

競技プログラミングの問題を準備するとき、何らかのヒューマンエラーによって、意図しない状態の問題文やデータセットを作ってしまう危険性があります。例えば以下のようなミスが考えられます。

- 問題文に書いてある制約と、データセットで利用している制約が食い違っており、データセットで制約違反を起こしてしまった
- 想定解法が変化し、あるサンプル入力に対応するサンプル出力を修正する必要があったが、修正前のものをそのまま問題文に書いてしまった
- Google Docs で問題文を管理し、定期的にローカルファイルに転記して問題文を作っていたが、古いバージョンのものを問題文として利用してしまった

このツールでは、このようなミスを防止するための機能が備わっており、コマンドを打つだけで適切に問題文を作ることができます。これについての詳細は **特長** をご覧ください。

サンプルファイルを使って、このツールをすぐに試してみたい方は **クイックスタート** をご覧ください。

1.1 Screencast

問題文中に制約やサンプルの情報が直接的には書かれていないファイルを statements-manager でビルドすると、それらの情報を埋め込んだ HTML ファイルが出力されていることが確認できます。また、制約を表すファイル `constraints.hpp` が自動で生成されていることも確認できます。

1.2 特長

1.2.1 問題の制約・サンプルの一元的な管理を実現

このツールでは、問題それぞれについて `problem.toml` という設定ファイルを作ります。この設定ファイルに、問題制約やサンプルへのパスなどの情報を記載します。

statements-manager では、この設定ファイルをもとに制約ファイル `constraints.hpp` を自動生成します。データセットを作成するスクリプト内でこのファイルを利用することで、 **問題文の制約と同じ制約でデータセットを作ることができます。** また、statements-manager は問題文中に制約やサンプル入出力の情報を自動で埋め込むため、**問題文中に制約やサンプル入出力をハードコーディングする必要がありません。**

実際にどのように設定ファイルを作ればよいかは、以下をご覧ください。

- **本ツールの導入の流れを知りたい方へ**
- *problem.toml* **の書き方**
- *problemset.toml* **の書き方**

1.2.2 Google Docs 上の問題文 / ローカル上の問題文の両方に対応

ローカルに保存されている問題文だけでなく、Google Docs で管理している問題文にも対応しています。この場合ももちろん、先ほどの例のように制約やサンプルを一元的に管理できます。

Google Docs では、テキストに対して修正の提案を出すことができますが、本ツールの実行時には **未解決の提案が存在するかどうかを確認できます。** これは例えば、問題準備の終盤に未解決の提案が存在するかどうか調べるときに役立ちます。

Problem Statement

However, not all instructions are possible to carry out. Now we call an instruction which meets either of following conditions "impossible instruction":

- When Mon harvests~~harvest~~ apples, the amount of apples exceeds the capacity of that box.
- When Mon tries to ship apples, there are not enough apples to ship.

Your task is to detect the instruction which is impossible to carry out.

Input

Input is given from standard input in the following format.

```
$N$
$c_1$ $c_2$ $\cdots$ $c_N$
$Q$
```

proposed replacement

tsutaj
1:05 今日
置換: 「harvest」 を 「harvests」 に

proposed addition

tsutaj
1:06 今日
追加: 「from standard input」

Console Output

```
[01:12:16 INFO] rendering [problem id: H]
[01:12:16 INFO] trying to get docs file using token (/home/tsutaj/.ss-manager/token.pickle)
[01:12:16 INFO] file_cache is only supported with oauth2client<4.0.0
[01:12:17 WARNING] proposed element for addition (ignored in rendering): harvests
[01:12:17 WARNING] proposed element for deletion: harvest
[01:12:17 WARNING] proposed element for addition (ignored in rendering):  from standard input
[01:12:17 INFO] create params file
[01:12:17 WARNING] skip dumping constraints file: same result as before
[01:12:17 WARNING] output directory '/home/tsutaj/statements/repo/tsutaj-repo/statements-manager/sample/H/ss-out' already exists.
[01:12:17 INFO] saving replaced html
[01:12:17 INFO] constraints: MIN_N => 1
[01:12:17 INFO] constraints: MAX_N => 1000
[01:12:17 INFO] constraints: MIN_C => 1
[01:12:17 INFO] constraints: MAX_C => 100000
[01:12:17 INFO] constraints: MIN_Q => 1
[01:12:17 INFO] constraints: MAX_Q => 100000
[01:12:17 INFO] constraints: MIN_D => 1
[01:12:17 INFO] constraints: MAX_D => 100000
[01:12:17 INFO] replace sample 1 (00_sample_00)
[01:12:17 INFO] replace sample 2 (00_sample_01)
[01:12:17 INFO] replace sample 3 (00_sample_02)
[01:12:17 WARNING] 00_sample_02: There is no explanation.
[01:12:17 INFO] replace sample 4 (00_sample_03)
[01:12:17 WARNING] 00_sample_03: There is no explanation.
[01:12:17 WARNING] skip dumping html: same result as before
```

showing unresolved proposal
(they are ignored in rendering)

第 2 章

クイックスタート

2.1 リポジトリにある例を試してみたい方へ

まずは動作させてみたいという方は、以下のようにコマンドを実行してください。

```
$ pip install statements-manager
$ git clone https://github.com/tsutaj/statements-manager.git
$ cd statements-manager

$ ss-manager run ./sample
```

`ss-manager run ./sample` を実行した後は、H 問題以外の各問題ディレクトリについて、`ss-out` ディレクトリ内に HTML が生成されており、`tests` ディレクトリ内に制約ファイルが生成されているはずです。

H 問題の問題文は Google Docs にあるため、この手順だけでは HTML 生成ができません。*Google Docs API* **を使用可能にする** で述べられている設定を行うと、H 問題に関しても HTML 生成が可能です。

デフォルトでは HTML が生成されますが、Markdown や PDF の出力にも対応しています。出力形式を指定するには `-o`, `--output` オプションを使います。

```
# generate Markdown files
ss-manager run ./sample -o md

# generate PDF files
ss-manager run ./sample -o pdf
```

また、`-p`, `--make-problemset` オプションによって、問題セット全体をまとめたファイルの生成も可能です。

```
# generate problemset PDF
ss-manager run ./sample -o pdf -p
```

コマンドの詳しい使い方については **コマンド** をご覧ください。

2.2 本ツールの導入の流れを知りたい方へ

問題文と設定ファイルを実際に用意して動作させることで、本ツールの導入の流れを説明します。

ここでは、「整数 A と B が与えられるので、$A + B$ の計算結果を出力してください」という問題を作りたいときに、どのように問題文を準備すればよいか説明します。ファイル構成などの詳細を確認したいときは リポジトリ内のサンプル を参照してください。

2.2.1 問題文を用意する

問題文を普通に書くと、以下のような Markdown ファイル `statement.md` を作ることになるでしょう。(数式は MathJax を想定した記法になっています)

````
### Problem Statement

You are given two integers $A$ and $B$. Print the calculation result of $A + B$.

### Input

Input is given from standard input in the following format:

```
$A$ $B$
```

### Output

You have to output the calculation result of $A + B$ to the standard output.

### Constraints

- $1 \leq A, B \leq 10^9$

### Sample Input 1

```
3 5
````

(次のページに続く)
````

(前のページからの続き)

```
```

Sample Output 1

```
8
```

Sample Input 2

```
10000 100
```

Sample Output 2

```
10100
```
```

この Markdown にはいくつか問題点があります。

- 入力 $A, B$ に対する制約がファイルに直接書かれています。仮に $A, B$ の上限をともに $10^9$ から $10^{18}$ に変更しなければならないとき、問題文の制約とデータセット生成器で使う制約を両方変更しなければなりませんが、これを別々に変更するとミスが起きやすいです。
- サンプル入出力も直接書かれているため、上で述べたことと同様の問題が起きやすいです。また、サンプルのナンバリングもファイルに直接書かれているため、番号が重複したり抜け落ちたりする可能性があります。

これらの問題を解消するため、statements-manager を導入して問題文を書き直していきます。

### 2.2.2 問題制約とサンプル入出力を別ファイルに移す

サンプル入出力を別ファイルに移します。入力例 1 に対応するファイルは `00_sample_00.in` に、出力例 1 に対応するファイルは `00_sample_00.out` に用意します。同様に入出力例 2 に対応するファイル `00_sample_01.in`, `00_sample_01.out` も用意します。これらのファイルは全て `tests` というディレクトリ内に格納しておきます。

また、問題制約を問題文ファイルから分離するため、以下のような設定ファイル `problem.toml` を用意します。
```

```
# problem id (it is used for the name of output html file)
id = "A"

# output params
params_path = "./tests/constraints.hpp"

# path to statements
[[statements]]
path = "./statement.md"
lang = "en"

# write constraints
[constraints]
    MIN_AB = 1
    MAX_AB = 1_000_000_000
```

これで問題制約とサンプル入出力を問題文から分離できました！最後に、問題文のファイル statement.md を次のように書き換えます。

```
### Problem Statement

You are given two integers $A$ and $B$. Print the calculation result of $A + B$.

### Input

Input is given from standard input in the following format:

```
$A$ $B$
```

Output

You have to output the calculation result of $A + B$ to the standard output.

Constraints

- ${@constraints.MIN_AB} \leq A, B \leq {@constraints.MAX_AB}$

{@samples.all}
```

重要なのは、 **問題制約とサンプル入出力が問題文ファイルに直接書かれていない** ことです。

ここまで作ったファイルの階層を整理してみましょう。以下と同じであれば OK です。

```
problem.toml
statement.md
tests/
├── 00_sample_00.in
├── 00_sample_00.out
├── 00_sample_01.in
└── 00_sample_01.out
```

2.2.3 ツールを実行して出力結果を得る

ここまで用意できたら、statements-manager を実行していきます。まだインストールしていない方は、次のコマンドでインストールしてください。

```
$ pip install statements-manager
```

インストール後、 `problem.toml` と同じ階層で以下のコマンドを実行します。

```
$ ss-manager run
```

実行が終わると `ss-out/A.html` というファイルと、 `tests/constraints.hpp` というファイルが出来ているはずです。

`ss-out/A.html` ファイルは、問題制約やサンプルが埋め込まれた後の HTML 形式の問題文です。

You are given two integers A and B. Print the calculation result of $A + B$.

Input

Input is given from standard input in the following format:

```
A B
```

- $1 \leq A, B \leq 10^9$

Output

You have to output the calculation result of $A + B$ to the standard output.

Sample Input 1

```
3 5
```

Sample Output 1

```
8
```

Sample Input 2

```
10000 100
```

Sample Output 2

```
10100
```

`tests/constraints.hpp` は、問題制約が書かれた C++ 形式のファイルであり、内容は以下のとおりです。 **このファイルを使ってデータセットを作ることで、問題文と同じ制約でデータセットを作ることができます。**

第 3 章

コマンド

statements-manager の実行は次のように行います。

```
$ ss-manager COMMAND [OPTIONS] [ARGS]
```

使用できるコマンド COMMAND は以下のとおりです。

3.1 run

```
usage: ss-manager run [-h] [-o {html, md, pdf}] [-p] [-f] [-c] [working_dir]
```

用意した Markdown ファイルを読み込み、指定された形式の出力ファイルを作成します。また、制約ファイルを出力する設定になっているときは、制約ファイルも出力します。

3.1.1 オプション

working_dir

問題文の生成対象となるディレクトリを指定します。何も指定しない場合、コマンドが実行された階層を指定したとみなします。

working_dir 以下を再帰的に探索し、見つかった problem.toml それぞれについて問題文の生成が行われます。

-o, --output

出力ファイルの形式を指定します。オプジョンに続けて html, md, pdf のいずれかを指定します。

このオプションが指定されていない場合、 html が指定されているとみなして実行されます。

-p, --make-problemset

問題セットのファイルも出力します。出力形式は -o, --output オプションで指定されたものに従います。

-f, --force-dump

キャッシュファイルの情報を無視し、常に出力ファイルを更新します。

このオプションが付いておらず、既に存在する出力ファイルから内容が変化していない場合は、出力ファイルは更新されません。

Tip: statements-manager を実行すると、出力ファイルのバージョンを管理するためのファイル cache.json も出力されます。通常、このファイルに書かれているハッシュ値と一致するときはファイルの更新を行いません。

-c, --constraints-only

制約ファイルのみを更新します。このオプションを付けた場合、問題文の出力ファイルは更新されません。

-h, --help

ヘルプメッセージを出力します。

3.1.2 実行例

次のコマンドを考えます。

```
$ ss-manager run ./problems -o pdf -p
```

このコマンドは次のように実行されます。

- ./problems 以下にある問題文を対象として出力ファイルを作成する
- PDF 形式で出力する
- 問題セットも出力する
- 出力ファイルの内容に変化がなければファイルを更新しない

3.2 reg-creds

```
usage: ss-manager reg-creds [-h] creds_path
```

Google Docs の API credentials を登録します。詳しい登録方法は *Google Docs API* **を使用可能にする** をご覧ください。

警告: Google Docs にある問題文を扱いたい場合は、このコマンドによる API credential の登録が必須となります。 問題文がすべてローカル環境に存在する場合はこの操作は不要です。

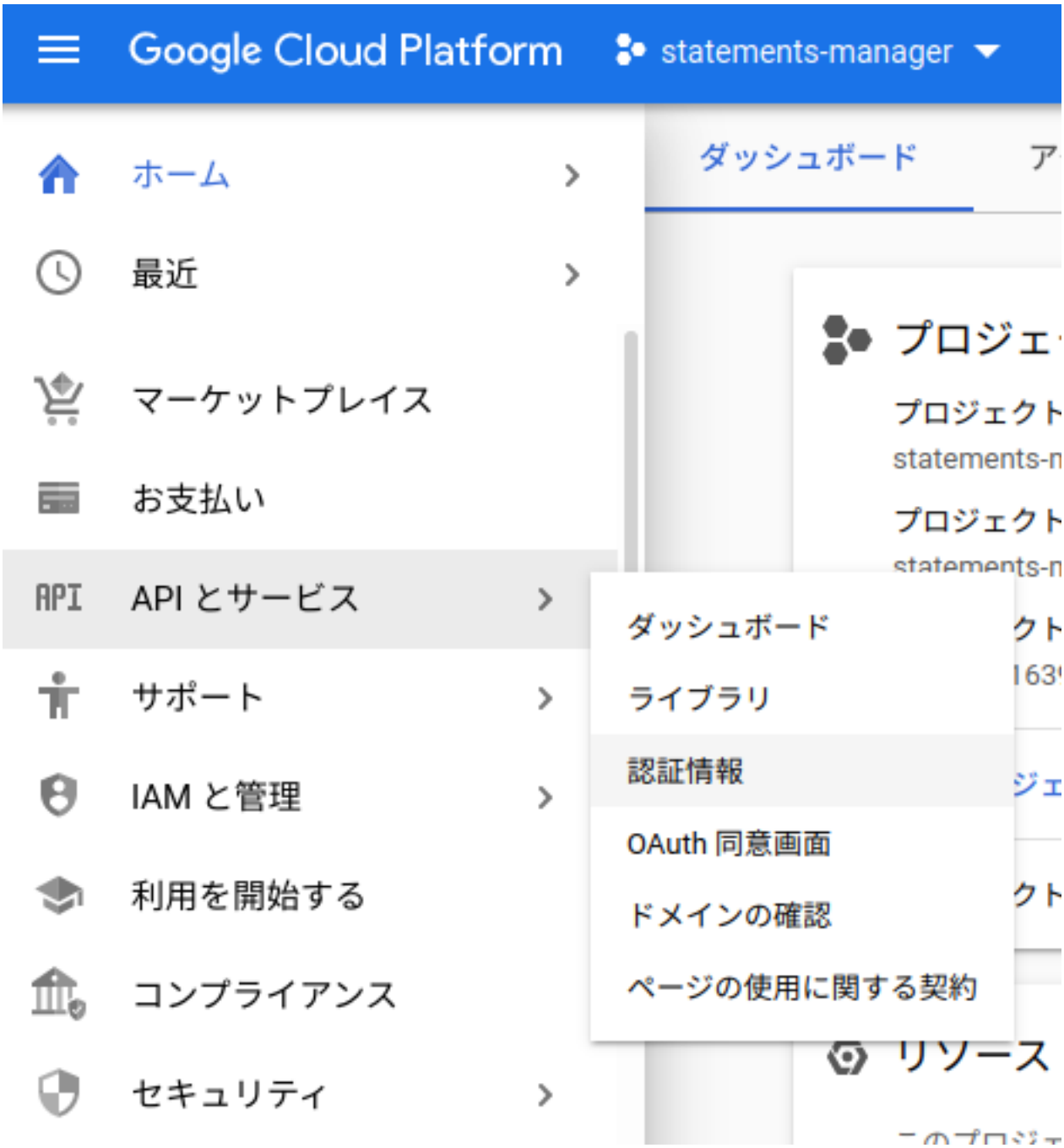
第 4 章

Google Docs API を使用可能にする

> **警告: Google Docs にある問題文を扱いたい場合は、この操作が必須となります。** 問題文がすべてローカル環境に存在する場合はこの操作は不要です。

作業ディレクトリ `WORKING_DIR` に対して、以下で説明する credentials というものを登録します。

- Google Docs - Quickstart **の手順通りに進め、API を使える状態にします。リンク先のサンプルを実行できるかどうかで動作確認が可能です。**
  - 扱いたい Docs ファイルが閲覧できる権限を持っているアカウントで作成しなければならないはずですので、アカウントの選択に注意してください
- Google Cloud Platform にアクセスし、「API とサービス」→「認証情報」に進みます



以下の画面で OAuth **クライアントをダウンロードします。**JSON **ファイルを任意の場所にダウンロードしてください**

– 以降の説明では、ダウンロードした場所が `CREDS_PATH` であるとします

以下のコマンドを打って、JSON ファイルを登録します

- 登録が終われば、 CREDS_PATH にある json ファイルは削除しても構いません
- JSON ファイルは、ホームディレクトリに生成される隠しフォルダ .ss-manager の中に格納されます

```
$ ss-manager reg-creds CREDS_PATH
```

第 5 章

問題文の書き方

ローカル・Google Docs のいずれにおいても、問題文は Markdown 形式で記述してください。

入力形式を表す箇所はバッククオート 3 つで囲みます。以下がその例です。

```
```
N M
u_1 v_1
u_2 v_2
$\vdots$
u_M v_M
```
```

問題文中では以下の記法が使用できます。いずれの記法に関しても、出力ファイル上では何らかのパラメータ・ファイルに置換されます。使用例は リポジトリ内のサンプル をご覧ください。

{@constraints.<CONSTRAINT_NAME>}

問題制約のパラメータに置換されます。パラメータ名 <CONSTRAINT_NAME> は problem.toml の [constraints] で記述されている定数名である必要があります。

{@samples.s<NUMBER>}

サンプルに関連するファイル群のうち、<NUMBER> 番目 (leading-zero は許容しない) のものに置換されます。

サンプルの名前は拡張子を無視した状態で集合として管理されており、辞書順で小さいものから 1, 2, 3, ... と番号付けられています。例えばサンプルに関連するファイルが 00_sample_00.in, 00_sample_00.out, 00_sample_00.md, 00_sample_01.in, 00_sample_01.out の 5 つであった場合、 00_sample_00 が 1 番目・ 00_sample_01 が 2 番目となります。

{@samples.all}

problem.toml の sample_path で指定されたディレクトリ以下にある、サンプルに関連するすべてのファイル群に置換されます。

サンプルの挿入順番は、上述したサンプルの順序の通りに行われます。

第 6 章

problem.toml の書き方

ツールを使うために、問題ごとに設定ファイル problem.toml を作成します。

Tip: Rime を使用したことがある方向け: このファイルは Rime で言うところの PROBLEM ファイルに似た位置づけです。 PROBLEM と同じ階層に保存することを推奨します。

ファイルの一例は次のとおりです。より具体的な例は リポジトリ内のサンプル をご覧ください。

```toml
id = "A"

params_path = "./tests/constraints.hpp"
assets_path = "./statement/assets/"
sample_path = "./tests/"
ignored_samples = []

[[statements]]
path = "./statement/statement_ja.md"
lang = "ja"
markdown_extensions = ["md_in_html", "tables", "fenced_code"]

[[statements]]
path = "./statement/statement_en.md"
lang = "en"
markdown_extensions = ["md_in_html", "tables", "fenced_code"]

[constraints]
MIN_N = 1
MAX_N = 100_000
```

(次のページに続く)

(前のページからの続き)

```
MIN_M = 1
MAX_M = 100_000
MIN_D = 0
MAX_D = 2_000_000_000
```

設定項目それぞれについて説明します。

id

これは必須項目です

問題 ID を指定します。この ID はツール実行中の問題判別や、出力ファイルの名前に使用されます

ID は、実行時に操作対象となる設定ファイルそれぞれで **一意でなければなりません**。例えば、 `id = "A"` となる設定ファイルが複数存在してはいけません。

assets_path

問題文に添付する画像などが含まれているディレクトリへのパスを指定します。問題文に図が必要な場合などにご利用ください。

`assets_path` 以下に存在する全てのファイル・ディレクトリが `ss-out` ディレクトリ中の `assets` ディレクトリにコピーされます。画像などのリンクを張る際は、この仕様を念頭に置いて指定してください。

Tip: パスの記述は絶対パスでも良いですし、 `problem.toml` からの相対パスでも構いません。

---

sample_path

サンプルケースが含まれているディレクトリへのパスを指定します。何も指定しなかった場合は、`problem.toml` が存在する階層下の `tests` ディレクトリが設定されます。

指定されたディレクトリ内のファイルであって、以下に全て当てはまるものはサンプルケース関連のファイルとみなし、問題文に記載されます。

- 拡張子が `.in` / `.out` / `.diff` / `.md` のいずれかである
  - `.in` ファイル: 入力例を表すファイル
  - `.out` / `.diff` ファイル: 出力例を表すファイル
  - `.md` ファイル: インタラクティブの入出力例を表すファイル (`sample` ディレクトリの I 問題参照)
  - `[言語名]/*.md` ファイル: 入出力例に関する説明 (`sample` ディレクトリの A 問題を参照)
    - 例: 日本語で `00_sample_00` に関する説明をしたいならば、 `[sample_path]/ja/00_sample_00.md` というファイルを用意します

- ファイル名に sample が部分文字列として含まれる

ignore_samples

sample_path で指定されたディレクトリにある、サンプルケースとして認識されるファイル名のうち、問題文に反映してほしくないものをリスト形式で指定します。拡張子は含めてはなりません。何も指定されなかった場合、見つかった全てのサンプルケースが問題文に反映されます。

ファイル名の指定には Unix のシェル形式のワイルドカード も利用できます。

例えば 00_sample_00 および 00_sample_hoge を問題文に含めてほしくない場合、 ignore_samples = ["00_sample_00", "00_sample_hoge"] のように設定します。

params_path

問題制約となるパラメータの値を、generator や validator で利用できるようにファイルに出力したいときに、パラメータを記載したファイルの出力パスを指定します。何も指定しなかった場合は、ファイルが出力されません。

既存のファイルと全く同じ出力になる場合、出力をスキップします。

> **警告:** 現状は C++ 形式の出力のみ (.cpp, .cc, .h, .hpp) 対応しています。今後対応言語は増やす予定です

[[statements]]

この設定は必須です

用意する問題文ファイルそれぞれについて設定します。設定方法の例は sample ディレクトリにある A 問題・C 問題などを参照してください。

- サンプルの A 問題 では、英語・日本語の両方で問題文を作成する例を示しています
- サンプルの C 問題 では、英語・日本語の両方で問題文を作成することに加えて、制約のみが異なる問題を作成する例も示しています

各問題文ファイルについて以下を設定します。

path

この設定は必須です

- ローカルに問題文が存在する場合: 問題文が記載されているファイル名を指定します
- Google Docs に問題文が存在する場合: Google Docs の ID か、もしくは Google Docs のファイルの URL を指定します。設定方法の例は サンプルの H 問題 を参照してください。

`lang`

問題文が書かれている言語を設定します。`ja` (日本語) もしくは `en` (英語) のいずれか一方を指定します。

何も指定しなかった場合は `en` が設定されているとみなして実行します。

`markdown_extensions`

Python-Markdown が公式でサポートしている拡張機能 のうち、使用したいものの Entry Point をリスト形式で指定します。

`mode`

`docs` または `local` のどちらかを指定します。問題文ファイルが存在する場所に応じて設定ください。

何も設定しなかった場合はモードが自動で認識されますので、通常は `mode` を設定する必要はありません。

`[constraints]`

問題制約を記述します。[**定数名**] = [**定数**] のように記載します。

第 7 章

problemset.toml の書き方

必要であれば、HTML・PDF に適用されるテンプレートを指定するためのファイル `problemset.toml` を作成します。このファイルが無い場合は、デフォルトのテンプレートが使用されます。

`problemset.toml` は、 `ss-manager run` を実行するときの `WORKING_DIR` の階層と一致しているときにのみ参照されます。

Tip: Rime を使用したことがある方向け: このファイルは Rime で言うところの `PROJECT` ファイルに似た位置づけです。 `PROJECT` と同じ階層に保存することを推奨します。

書き方の一例は次のとおりです。より具体的な例は リポジトリ内のサンプル をご覧ください。

```
[template]
    template_path = "./templates/default.html"
    sample_template_path = "./templates/sample_default.html"
    preprocess_path = "./templates/preprocess.py"
    postprocess_path = "./templates/postprocess.py"

[pdf]
    [pdf.common]
        encoding = "UTF-8"
        page-size = "A4"
        margin-top = 24
        margin-right = 16
        margin-bottom = 16
        margin-left = 16
        enable-local-file-access = ""
        disable-smart-shrinking = ""
```

(次のページに続く)

(前のページからの続き)

```
    debug-javascript = ""
[pdf.problem]
    javascript-delay = 1000
[pdf.problemset]
    javascript-delay = 10000
    header-center = "Practice Contest, YYYY-MM-DD"
    header-font-name = "Times New Roman"
    header-font-size = 12
    header-spacing = 12
    footer-center = "[page] / [toPage]"
    footer-font-name = "Times New Roman"
    footer-font-size = 10
    footer-spacing = 8
```

設定項目それぞれについて説明します。

[template]

template_path

HTML および PDF 出力で使用されるテンプレート HTML へのパスを指定します。指定されていない場合、デフォルトのテンプレートが適用されます。

テンプレートでは、問題文本文に相当する部分に `{@problem.statement}` 文を記述する必要があります。詳細は sample/templates/default.html などをご覧ください。

sample_template_path

入出力例の部分に使われるテンプレート HTML へのパスを指定します。指定されていない場合、デフォルトのテンプレートが適用されます。

テンプレートの書き方は sample/templates/sample_default.html などをご覧ください。

preprocess_path

Markdown ファイルに関して前処理を行う **Python スクリプト** へのパスを指定します。Markdown が HTML 形式にレンダリングされる前に適用したい処理を記述してください。指定されていない場合、前処理は行われません。

Markdown ファイルの中身は標準入力で与えられ、前処理の結果は標準出力で返す必要があります。終了コードが 0 以外であった場合は異常終了とみなし、エラーになります。詳細は sample/templates/icpc_domestic/preprocess.py をご覧ください。

postprocess_path

HTML ファイルに関して後処理を行う **Python スクリプト** へのパスを指定します。HTML 形式にレン

ダリングされた後に適用したい処理を記述してください。指定されていない場合、後処理は行われません。

HTML ファイルの中身は標準入力で与えられ、後処理の結果は標準出力で返す必要があります。終了コードが 0 以外であった場合は異常終了とみなし、エラーになります。詳細は sample/templates/icpc_domestic/postprocess.py をご覧ください。

[pdf]

PDF 出力時の wkhtmltopdf (PDF にレンダリングする際に使用されるサードパーティライブラリ) の設定を書きます。設定項目の詳細については wkhtmltopdf のリファレンス をご覧ください。

[pdf.common]

各問題のファイルにも、問題セットのファイルにも適用されてほしい設定をここに記載します。

[pdf.problem]

各問題のファイルにのみ適用されてほしい設定をここに記載します。

[pdf.problemset]

問題セットのファイルにのみ適用されてほしい設定をここに記載します。

第 8 章

推奨するファイル構成

以下のようなディレクトリ構成を推奨しています。作問支援ツールである Rime を使用するときのディレクトリ構成と似ています。

```
--- WORKING_DIR/
    |
    |- A/                 (directory of Problem A)
    |  |- tests/          (generator / validator / sample cases)
    |  |- statement/      (problem statement, assets such as images)
    |  |- problem.toml    (problem config file for ss-manager)
    |  |- solution_1/     (solution)
    |  |- solution_2/     (solution)
    |  ...
    |
    |- B/                 (directory of Problem B)
    |  |- (same as above)
    |  ...
    |
    ...
```

第 9 章

Rime と組み合わせて使う

statements-manager は、問題作成支援ツール Rime と組み合わせて使うことを想定した作りになっています。statements-manager は問題文の準備作業を補助し、Rime は問題文作成を除くすべての工程の準備作業を補助するツールです。

このため、想定しているディレクトリ構成も Rime と似ています。設定ファイル類は以下のように配置することを推奨しています。

- 問題セットに関する設定を行うファイル `problemset.toml` は、Rime のプロジェクト設定ファイル `PROJECT` と同じ階層に置く
- 各問題に関する設定を行うファイル `problem.toml` は、Rime の問題設定ファイル `PROBLEM` と同じ階層に置く
- 制約ファイルの出力先 `params_path` は、Rime で入力生成器・入力検証器を配置するディレクトリ `tests` と同じ階層のパスにする

また、組み合わせて使う際に推奨している実行順は以下のとおりです。

```
# create constraints files
$ ss-manager run -c WORKING_DIR

# create correct sample outputs
$ rime clean WORKING_DIR
$ rime test WORKING_DIR

# create problem statements
$ ss-manager run WORKING_DIR
```

statements-manager によって、問題文の出力ファイルと制約ファイルを更新します。その後、その制約ファイルを使って Rime でデータセットを生成し、用意された解法が正しく動作するかどうかをチェックします。この順番で操作することで、問題文とデータセットの制約のズレを減らすことができます。

第 10 章

リポジトリにある問題文を半自動で更新する

GitHub Actions などの CI サービスと併用することで、リポジトリに変更が加えられたときに問題文に関する成果物の差分を push し、常にリポジトリ内の問題文を最新の状態に保つことが可能です。

設定の一例を以下に示します。これは `master` ブランチに push された際に `ss-manager run` を実行し、差分を自動で push するものです。以下の実装を、リポジトリに `.github/workflows/statements-manager.yml` として保存すると動作するはずです。

```yaml
# run statements-manager and commit/push diffs
name: update-statements

on:
push:
    branches: [master]

jobs:
build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3
        uses: actions/setup-python@v2
        with:
        python-version: 3.8
    - name: Install dependencies
        run: |
        python -m pip install --upgrade pip
        pip install statements-manager
    - name: Run statements-manager
```

(次のページに続く)

(前のページからの続き)

```
      run: |
      ss-manager run ./
  - name: Commit files
      run: |
      git add --all
      git config --local user.email "github-actions[bot]@users.noreply.github.com"
      git config --local user.name "github-actions[bot]"
      git commit -m "[ci skip] [bot] Updating to ${{ github.sha }}."
  - name: Push changes
      uses: ad-m/github-push-action@master
      with:
      branch: ${{ github.ref }}
```

第 11 章

Links

- library-checker-problems
  - 本ツールの機能は、これの影響を強く受けています。library-cheker-problems の作問機能で出来ることを網羅しつつ Rime と親和性が良い設計にし、さらに作問時に便利な Google Docs とも連携させたいというモチベーションがあり、このツールが作られました。
- Rime
  - 競技プログラミングの問題作成において、問題文以外の全ての工程の補助を行うツールです。本ツールは Rime と組み合わせて使うことが想定されています。

第 12 章

For Contributors

Issue / PR など contribute してくださる方を募集しています。詳細は CONTRIBUTING.md をご覧ください。

索引