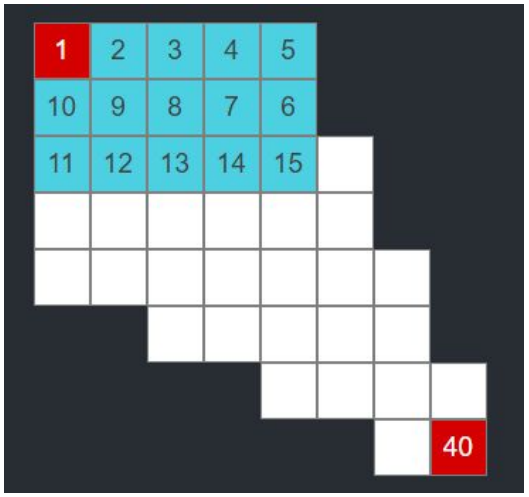
Tugas 2 Pemrograman Logika:
Hidato Solver

http://www.hidato.herokuapp.com

Kelompok 8
Albertus Angga Raharja (1606918401)
Usama (1606862766)
Wira Abdillah Siregar (1506690630)

1. Deskripsi Permasalahan: Hidato

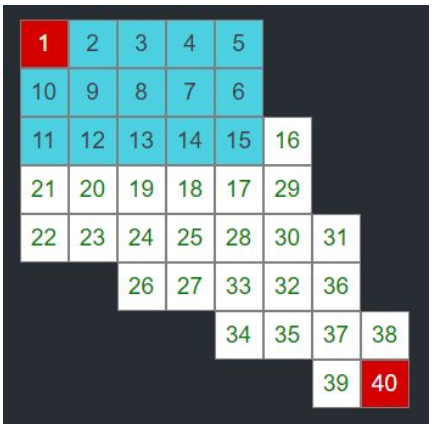


Gambar 1.1: Hidato *Puzzle* yang belum terselesaikan

Hidato yang juga dikenal sebagai Hidoku adalah sebuah *puzzle* yang dimulai dengan petak (*grid*) yang sudah terisi sebagian dengan bilangan. Umumnya *puzzle* ini berbentuk persegi seperti Sudoku dengan ukuran petak 7 × 7 atau 9 × 9. Namun, Hidato juga dapat berbentuk segi enam atau segi lainnya yang dapat membentuk *tessellation.*

Tujuan dari *puzzle* ini adalah mengisi seluruh petak dengan bilangan yang terurut dan terhubung secara horizontal, vertikal, maupun diagonal. Di setiap Hidato *puzzle,* bilangan terkecil dan bilangan terbesar selalu diberikan dalam petak. Hidato *puzzle* juga mengandung bilangan-bilangan lain yang berguna untuk membantu mengarahkan pemain untuk menyelesaikan atau memecahkan *puzzle.* Selain itu, bilangan-bilangan yang terdapat dalam petak juga berguna untuk memastikan bahwa *puzzle* Hidato yang diberikan hanya memiliki tepat satu solusi.

Pada tugas kali ini, kami membuat program *Prolog* untuk menyelesaikan *puzzle* Hidato dengan cara mendapatkan **input berupa petak seperti pada Gambar 1.1** dan mengembalikan **output berupa *puzzle* Hidato yang terselesaikan** seperti pada Gambar 1.2 di bawah.

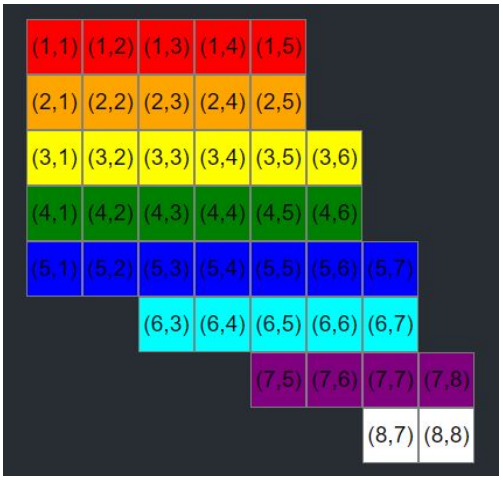


Gambar 1.2: Hidato *Puzzle* yang sudah terselesaikan

2. Metode Penyelesaian Hidato

Pada bagian ini akan dibahas metode penyelesaian Hidato dari sisi pemodelan masalah, algoritma, dan *interface*.

2. 1 *Modelling* Permasalahan Hidato



Gambar 2.1: Penomoran Petak pada *Puzzle* Hidato

Bentuk *puzzle* Hidato yang merupakan permasalahan pada tugas kali ini adalah *puzzle standard* dengan bentuk persegi (*standard-sized square-shaped* Hidato *puzzle*) seperti gambar 2.1 di atas. Kami memodelkan permasalahan ini dengan cara melakukan penomoran pada masing-masing petak seperti pada gambar 2.1 di atas terlebih dahulu. Penomoran dilakukan dengan cara membayangkan *puzzle* tersebut sebagai petak dengan bentuk persegi dengan ukuran 8 × 8 yang tidak penuh.

Setelah melakukan penomoran pada masing-masing petak, permasalahan menyelesaikan *puzzle* Hidato ini dapat dimodelkan sebagai ***Constraint Satisfaction Problem*** dengan *constraint* berikut:

Didefinisikan sebuah himpunan *tuple Grid* yang berisi *tuple* yang menunjukkan nomor (*baris, kolom*) yang ada pada bentuk *standard-sized square-shaped* Hidato *puzzle* sebagai berikut*:*

$$Grid = \{(1,1), (1,2), (1,3), (1, 4), (1,5), (2,1), (2,2), (2,3), (2,4), (2,5), (3,1), (3,2), (3,3), (3,4), (3,5), (3,6), (4,1), (4,2), (4,3), (4,4), (4,5), (4,6), (5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (5,7), (6,3), (6,4), (6,5), (6,6), (6,7), (7,5), (7,6), (7,7), (7,8), (8,7), (8,8)\}$$

Input dari permasalahan ini adalah himpunan *tuple Input* yang memenuhi syarat sebagai berikut:

$$Input = \{(i, j, val) \mid (i, j) \in Grid,\ val \in Z,\ val \geq 1,\ val \leq 40\}$$
$$\exists i_1 \exists j_1 (i_1, j_1) \in Grid \wedge (i_1, j_1, 1) \in Input$$
$$\exists i_2 \exists j_2 (i_2, j_2) \in Grid \wedge (i_2, j_2, 40) \in Input$$

Permasalahan penyelesaian *puzzle* Hidato pada tugas ini kami modelkan sebagai permasalahan *Constraint Satisfaction Problem* untuk mencari himpunan *Output* yang memenuhi syarat sebagai berikut:

$$\begin{aligned} Output = \{ (i,\ j,\ val_{i,j}) \mid (i, j) \in Grid, val_{i,j} \in Z,\ val_{i,j} \geq 1,\ val_{i,j} \leq 40,\ val_{i,j} \neq 40 \rightarrow (\\ ((i, j-1) \in Grid \wedge val_{i,j} - val_{i,j-1} = 1) \vee \\ ((i, j+1) \in Grid \wedge val_{i,j} - val_{i,j+1} = 1) \vee \\ ((i-1, j) \in Grid \wedge val_{i,j} - val_{i-1,j} = 1) \vee \\ ((i+1, j) \in Grid \wedge val_{i,j} - val_{i+1,j} = 1) \vee \\ ((i-1, j-1) \in Grid \wedge val_{i,j} - val_{i-1,j-1} = 1) \vee \\ ((i-1, j+1) \in Grid \wedge val_{i,j} - val_{i-1,j+1} = 1) \vee \\ ((i+1, j-1) \in Grid \wedge val_{i,j} - val_{i+1,j-1} = 1) \vee \\ ((i+1, j+1) \in Grid \wedge val_{i,j} - val_{i+1,j+1} = 1) \\ )\} \end{aligned}$$

Untuk menghindari pencarian solusi yang terlampau lama, kelompok kami menambahkan syarat tambahan untuk *Input* yaitu minimal terdapat 12 bilangan yang sudah terisi pada petak ($|Input| \geq 12$).

2.2 Algoritma Penyelesaian Hidato

Permasalahan *Hidato* dapat diselesaikan menggunakan algoritma *Depth-first search* (DFS) dengan mencari seluruh kemungkinan solusi dan melakukan *pruning* dan *backtracking* saat kemungkinan yang saat ini sedang diperhitungkan menjadi tidak valid.

2.3 Program Code Hidato Solver

```
% hidato.pl
:- use_module(movement, [move/2]).
:- use_module(validator, [valid/2, grid/2]).
:- use_module(storage, [memo/3, ans/3, reader/0]).

:- dynamic solved/0.

iterate_ans(41).

iterate_ans(Now) :-
   ans(_, _, Now), !,
   Next is Now + 1,
   iterate_ans(Next).

iterate_ans(Now) :-
   memo(R, C, Now),
   assert(ans(R, C, Now)),
   Next is Now + 1,
   iterate_ans(Next).

is_valid(R, C) :-
   valid(R, C),
   grid(R, C).

assert(R, C, Val) :-
   \+ memo(R, C, _),
   assert(memo(R, C, Val)), !.

assert(R, C, Val) :-
   memo(R, C, X),
   Val =< X,
   retract(memo(R, C, X)),
   assert(memo(R, C, Val)).

iterate(_, _, 41) :-
   iterate_ans(1), !.

iterate(R_now, C_now, Val_now) :-
   retractall(memo(_, _, Val_now)),
```

```
   ans(_, _, Val_now), !,
   ans(R_now, C_now, Val_now),
   is_valid(R_now, C_now),
   assert(R_now, C_now, Val_now),
   move(R, C),
   R_next is R_now + R,
   C_next is C_now + C,
   Val_next is Val_now + 1,
   iterate(R_next, C_next, Val_next), !.

iterate(R_now, C_now, Val_now) :-
   retractall(memo(_, _, Val_now)),
   is_valid(R_now, C_now),
   \+ ans(R_now, C_now, _),
   assert(R_now, C_now, Val_now),
   move(R, C),
   R_next is R_now + R,
   C_next is C_now + C,
   Val_next is Val_now + 1,
   iterate(R_next, C_next, Val_next), !.

find_ans :-
   retractall(solved),
   ans(R, C, 1),
   iterate(R, C, 1), !,
   assert(solved).

clear :-
   retractall(memo(_, _, _)),
   retractall(ans(_, _, _)).
```

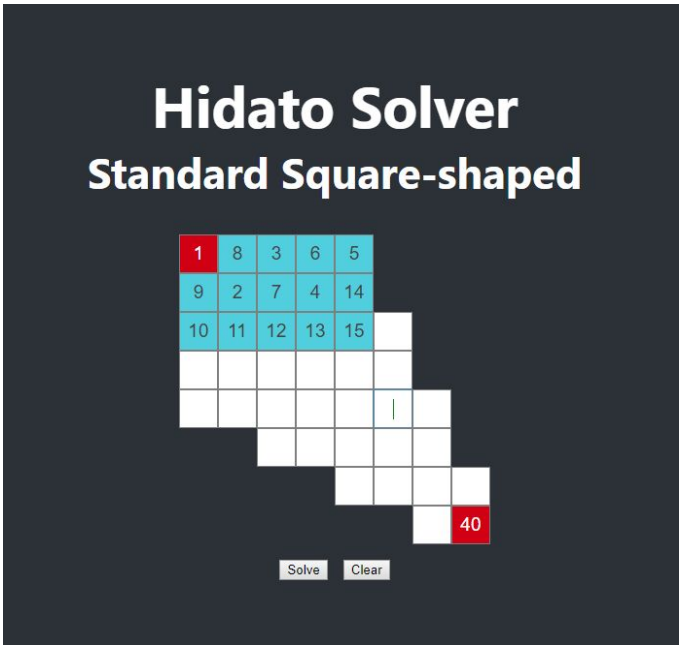
```
% storage.pl
:- module(storage,
   [memo/3, ans/3, reader/0]).

:- dynamic memo/3.
:- dynamic ans/3.
```

```
% validator.pl
```

```
:- module(validator,
   [valid/2, grid/2]).

valid(R, C) :-
   1 =< R, R =< 8,
   1 =< C, C =< 8.

grid(R, C) :-
   C =< 2, !, R =< 5.
grid(R, C) :-
   C =< 4, !, R =< 6.
grid(R, 5) :-
   !, R =< 7.
grid(R, 6) :-
   !, 3 =< R, R =< 7.
grid(R, 7) :-
   !, 5 =< R.
grid(R, 8) :-
   !, 7 =< R.
```

```
% movement.pl
:- module(movement, [move/2]).

move(-1, -1).
move(-1, 0).
move(-1, 1).
move(0, -1).
move(0, 1).
move(1, -1).
move(1, 0).
move(1, 1).
```

2.4 Interface Hidato *Solver*



Gambar 3.1: *Interface Hidato Solver* yang dapat diakses pada https://hidato.herokuapp.com

Interface pada program Hidato *Solver* kami menggunakan tampilan *user interface* berbasis web dengan bantuan teknologi HTML, CSS, JavaScript (React.js). Kode dari *user interface* ini dapat dilihat pada folder `front-end`.

Kami juga membuat program Prolog sebagai *back end web server* agar *user interface* dapat berkomunikasi secara langsung dengan program Prolog yang kami buat pada bagian 2.4. Kami memanfaatkan protokol **HTTP dengan REST API (JSON)** agar program *user interface* dapat berkomunikasi dengan program Prolog yang kami buat.

Hal ini dimungkinkan karena sudah terdapat implementasi library **http** pada SWI Prolog. Berikut ini potongan kode server Prolog yang digunakan untuk serve *static file* (HTML, CSS, JavaScript) untuk *program interface* kami dan satu *end point* (URL) untuk terhubung dengan program Hidato solver.

`server.pl`

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/http_server_files)).
:- use_module(library(http/http_cors)).
:- use_module(library(http/http_json)).
:- (multifile user:file_search_path/2).
:- consult(hidato).

user:file_search_path(root_dir, 'build').
```

```
:- set_setting(http:cors, [*]).
:- http_handler(root(solve), solve, []).
:- http_handler('/', http_reply_file('build/index.html', []), []).
:- http_handler(root(.), serve_files_in_directory(root_dir),
[prefix]).

server(Port) :-
   http_server(http_dispatch, [port(Port)]).

fill_constraint([]).
fill_constraint([H|T]):-
   H=json([row=R, col=C, val=V]),
   assert(ans(R, C, V)),
   fill_constraint(T).

solve(Request) :-
     option(method(options), Request), !,
     cors_enable(Request,
                 [ methods([get,post,delete])
                 ]),
     format('~n').

solve(Request):-
   cors_enable,
   http_read_json(Request, Data),
   fill_constraint(Data),
   find_ans,
   findall(json([row=R, col=C, val=V]), ans(R, C, V), L),
   reply_json(L),
   clear.

:- initialization(run, main).

run :- server(8000), thread_get_message(stop).
```

Program *Hidato Solver* ini dapat kita jalankan dengan menjalankan perintah:

```
swipl server.pl
```

Setelah menjalankan perintah tersebut, *user interface* dari program kami akan dapat diakses di `localhost:8000`.

API Documentation

Berikut ini API Documentation yang digunakan oleh *user interface* untuk menyelesaikan *puzzle Hidato*.

`localhost:8000/solve`

Request Headers:

```
"content-type": "application/json"
```

Request Body:

```
[
   {
      "row":1,
      "col":1,
      "val":1
   },
   {
      "row":1,
      "col":2,
      "val":2
   },
   {
      "row":1,
      "col":3,
      "val":3
   },
   {
      "row":1,
      "col":4,
      "val":4
   },
   {
      "row":1,
      "col":5,
      "val":5
   },
   {
      "row":2,
      "col":5,
      "val":6
   },
   {
      "row":2,
```

```
            "col":4,
            "val":7
        },
        {
            "row":2,
            "col":3,
            "val":8
        },
        {
            "row":2,
            "col":2,
            "val":9
        },
        {
            "row":2,
            "col":1,
            "val":10
        },
        {
            "row":3,
            "col":1,
            "val":11
        },
        {
            "row":3,
            "col":2,
            "val":12
        },
        {
            "row":3,
            "col":3,
            "val":13
        },
        {
            "row":3,
            "col":4,
            "val":14
        },
        {
            "row":3,
            "col":5,
```

```
            "val":15
        },
        {
            "row":8,
            "col":8,
            "val":40
        }
    ]
```

Response:

```
    [
      {"row":1, "col":1, "val":1},
      {"row":1, "col":2, "val":2},
      {"row":1, "col":3, "val":3},
      {"row":1, "col":4, "val":4},
      {"row":1, "col":5, "val":5},
      {"row":2, "col":5, "val":6},
      {"row":2, "col":4, "val":7},
      {"row":2, "col":3, "val":8},
      {"row":2, "col":2, "val":9},
      {"row":2, "col":1, "val":10},
      {"row":3, "col":1, "val":11},
      {"row":3, "col":2, "val":12},
      {"row":3, "col":3, "val":13},
      {"row":3, "col":4, "val":14},
      {"row":3, "col":5, "val":15},
      {"row":8, "col":8, "val":40},
      {"row":3, "col":6, "val":16},
      {"row":4, "col":5, "val":17},
      {"row":4, "col":4, "val":18},
      {"row":4, "col":3, "val":19},
      {"row":4, "col":2, "val":20},
      {"row":4, "col":1, "val":21},
      {"row":5, "col":1, "val":22},
      {"row":5, "col":2, "val":23},
      {"row":5, "col":3, "val":24},
      {"row":5, "col":4, "val":25},
      {"row":6, "col":3, "val":26},
      {"row":6, "col":4, "val":27},
      {"row":5, "col":5, "val":28},
      {"row":4, "col":6, "val":29},
```

```
    {"row":5, "col":6, "val":30},
    {"row":5, "col":7, "val":31},
    {"row":6, "col":6, "val":32},
    {"row":6, "col":5, "val":33},
    {"row":7, "col":5, "val":34},
    {"row":7, "col":6, "val":35},
    {"row":6, "col":7, "val":36},
    {"row":7, "col":7, "val":37},
    {"row":7, "col":8, "val":38},
    {"row":8, "col":7, "val":39}
  ]
```

Selain dapat dijalankan di local, seluruh program kami dapat diakses di URL: http://www.hidato.herokuapp.com

3. Diskusi dan Analisis Algoritma Hidato Solver

Kompleksitas Waktu :

Jika jumlah grid tidak terbatas maka waktunya sekarang menjadi $8 \times 7^{38 - |input|}$. Anggap nilai yang diinput sebanyak 15, waktu kompleksitas waktu yang digunakan adalah 8×7^{38}. Namun karena *grid*-nya terbatas dan dilakukan *prunning* untuk nilai yang sudah sudah diinput maka kompleksitas . Maka waktunya adalah 8×7^{23}, hal ini terjadi jika grid-nya tak terbatas. Pada Hidoku dengan board cascade mengurangi waktu yang signifikan. Jika nilai 1 berada di tengah *board* maka maksimal menghasilkan 7^4. Kompleksitas menjadi $8 \times 7^4 \times L$. Nilai L maksimal yang mungkin adalah 6^{19}, seperti halnya kasus di atas, karena *grid*-nya berbentuk *cascade* dan *prunning* untuk nilai yang sudah menjadi input maka nilai L akan sangat jauh lebih kecil dari 6^{19}. Sehingga pada praktiknya kompleksitas waktu yang dihasilkan masih dapat ditoleransi.

Kompleksitas Memori :

Menyimpan jawaban sebanyak 40 nilai. $O(40) \approx O(1)$

4. Kesimpulan

Pada laporan ini kami menjelaskan bagaimana cara memodelkan Hidato sebagai ***Constraint Satisfaction Problem*** lalu menentukan *constraintnya*. Kami juga menganalisis *run time* dan *space complexity* dari program yang kami buat untuk menyelesaikan permasalahan Hidato. Setelah melakukan pemodelan masalah Hidato ini, kami berhasil membuat program yang dapat menyelesaikan *puzzle* Hidato (Hidato *solver*) berbasis *web* dengan menggunakan HTML, CSS, JavaScript sebagai *user interface* dan Prolog sebagai *back end server* yang dapat saling berkomunikasi satu sama lain melalui protokol HTTP REST.