Program sprawdzający czy grupa jest podgrupą drugiej grupy oraz czy są izomorficzne

Projekt zaliczeniowy na przedmiot „Obliczeniowa Teoria Grup” w semestrze letnim 2023r. na Uniwersytecie Gdańskim.

Autorzy: Marcin Michnik, Dawid Leman, Łukasz Gładyś

Opis programu

Program został napisany w języku Python, wykorzystując komendy sprawdzające napisane w języku GAP. Pozwala on na:

- sprawdzenie czy dana grupa jest podgrupą drugiej grupy,
- sprawdzenie czy dana grupa jest izomorficzna.

Kod źródłowy generuje skrypt w języku GAP na podstawie zdefiniowanych instrukcji, a następnie zapisuje go do pliku „simulation.g". Kolejnym krokiem jest utworzenie skryptu powłoki, który uruchamia skrypt GAP oraz zapisuje wynik jego działania do pliku „gap_output.txt".

Opis klas

Klasa GapFunction

Celem klasy GapFunction jest reprezentowanie funkcji w języku GAP oraz serializacja do typu str przy użyciu metody __str__(). Reprezentacja tekstowa funkcji może być zapisana do pliku oraz wykonana przy użyciu powłoki Shell. Klasa GapFunction posiada następujące pola:

- Pole name (typ str) – nazwa funkcji gap
- Pole statements (typ list[str]) - lista instrukcji, które zostaną wykonane wewnątrz funkcji gap.

```
class GapFunction():
    def __init__(self, name, statements):
        self.name = name
        self.statements = statements

    def __str__(self):
        statement_literals = "\n    ".join(self.statements)
        return f"""{self.name} := function()
    {statement_literals}
end;"""
```

Rys. 1.1 Kod klasy GapFunction()

Klasa ShellScript

Celem klasy ShellScript jest reprezentowanie skryptu powłoki Shell. Przechowuje ona jedynie:

- Pole statements (typ list[str]) - lista instrukcji, które zostaną wykonane w powłoce Shell.

```
class ShellScript():
    def __init__(self, statements):
        self.statements = statements

    def __str__(self):
        statement_literals = "\n".join(self.statements)
        return f"""#!/bin/sh
{statement_literals}"""
```

Rys. 1.2 Kod klasy ShellScript()

Zarówno jak klasa GapFunction, ShellScript może być serializowany do reprezentacji tekstowej przy użyciu metody __str__().

Działanie programu

Początkowo losowane są dwie liczby, które zostaną użyte w dalszej części programu do określenia wielkości dwóch zdefiniowanych grup.

```
rand1 = random.randrange(2, 14, 2)
rand2 = random.randrange(2, 14, 2)
```

Rys. 1.3 Losowanie liczb parzystych od 2 do 14.

Następnie przedstawiona jest lista instrukcji *statements*, która wykonuje kolejne instrukcje języka GAP. Po zadeklarowaniu zmiennych, funkcja sprawdza czy pierwsza grupa wygenerowana na podstawie losowo wybranych liczb jest podgrupą drugiej grupy.

```
    f"G := SymmetricGroup({rand1});",
    f"H := DihedralGroup(IsPermGroup, {rand2});",
     """Print("G: ", G, "\\n");""",
     """Print("H: ", H, "\\n");""",
     """if IsSubgroup(G, H) then
     Print("H is a subgroup of G.\\n");
else
    Print("H is not a subgroup of G.\\n");
fi;""",
```

Rys. 1.4 Kod sprawdzający, czy grupa H jest podgrupą grupy G.

Wynikiem działania powyższego fragmentu kodu gap jest następujący tekst:

```
G: SymmetricGroup( [ 1 .. 2 ] )
H: Group( [ (1,2,3), (2,3) ] )
H is not a subgroup of G.
```

Kolejnym krokiem jest zdefiniowanie generatorów (wyłącznie jako informacja dla użytkownika) oraz sprawdzenie, czy grupy G i H są względem siebie izomorficzne.

```
    # Print the generators for groups
    """G_generators := GeneratorsOfGroup(G);""",
    """H_generators := GeneratorsOfGroup(H);""",
    """Print("G_generators: ", G_generators, "\\n");""",
    """Print("H_generators: ", H_generators, "\\n");""",
    # Check group Isomorphism
    """isomorphism := IsomorphismGroups(G, H);;""",
    """if isomorphism <> fail then
    Print("G and H are isomorphic.\\n");
    Print("Isomorphism: ", isomorphism, "\\n");
else
    Print("G and H are not isomorphic.\\n");
fi;"""
```

Rys. 1.5 Utworzenie generatorów oraz sprawdzenie czy grupy są izomorficzne.

Wyniki działania wyżej zamieszczonego kodu gdy grupy nie są izomorficzne:

```
G: SymmetricGroup( [ 1 .. 4 ] )
H: Group( [ (1,2,3,4), (2,4) ] )
H is a subgroup of G.
G_generators: [ (1,2,3,4), (1,2) ]
H_generators: [ (1,2,3,4), (2,4) ]
G and H are not isomorphic.
```

I gdy są izomorficzne:

```
G: SymmetricGroup( [ 1 .. 2 ] )
H: Group( [ (1,2) ] )
H is a subgroup of G.
G_generators: [ (1,2) ]
H_generators: [ (1,2) ]
G and H are isomorphic.
Isomorphism: GroupHomomorphismByImages( SymmetricGroup( [ 1 .. 2 ] ), Group(
[ (1,2) ] ), [ (1,2) ], [ (1,2) ] )
```

Po utworzeniu listy instrukcji *statements,* tworzony jest nowy obiekt *simulation* klasy *GapFunction*, zawierający nazwę funkcji oraz listę instrukcji do wykonania w języku GAP.

```
simulation = GapFunction("Simulate", statements)
```

Rys. 1.6 Obiekt simulation

Następnie tworzony jest plik *simulation.g*, do którego zapisywana jest reprezentacja tekstowa obiektu *simulation*. Jest to skrypt, który uruchamia zadeklarowane instrukcje w języku GAP.

```
gap_file_name = "simulation.g"
with open(gap_file_name, "w") as file:
    file.write(str(simulation))
```

Rys. 1.7 Tworzenie pliku *simulation.g*

Na podstawie określonych instrukcji powłoki, tworzony jest obiekt *sh* klasy *ShellScript*, który jest odpowiedzialny za uruchomienie skryptu napisanego w języku GAP.

```
sh_statements = [
    "gap -r -b -q << EOI",
 f"""Read("{gap_file_name}");""",
    "Simulate();",
    "EOI"
]
sh = ShellScript(sh_statements)
sh_file_name = "gap_executor.sh"
with open(sh_file_name, "w") as file:
    file.write(str(sh))
script_path = f"./{sh_file_name}"
```

Rys. 1.8 Tworzenie obiektu *sh*

Ostatnim krokiem jest wywołanie polecenia *subprocess.run()*, które uruchamia skrypt powłoki *gap_executor.sh*. Wynik działania skryptu jest przekierowywany do pliku "gap_output.txt".

```
with open(result_file_name, "w") as file:
    subprocess.run(["sh", script_path], stdout=file)
```

Rys. 1.9 Uruchomienie programu

Podsumowanie

Zaprezentowany kod tworzy skrypt w języku GAP, który jest uruchamiany za pomocą skryptu powłoki. Umożliwia to wykorzystanie języka GAP i zapisanie wyników do pliku tekstowego, korzystając z języka Python. Do najważniejszych operacji przeprowadzone w języku GAP należy sprawdzenie czy dana grupa jest podgrupą drugiej, a także czy podane grupy są izomorficzne.