MDS
Winter Term 2024/25
miniHive, Milestone 2

The goal of the coding project is to build a mini-version of Apache Hive, called *miniHive*. The second milestone requires you to perform logical optimization on relational algebra queries.

Selection Pushing in Relational Algebra

In relational algebra, the following equivalencies apply (among others):

$$\sigma_{p_1 \wedge p_2 \wedge \cdots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots)) \tag{1}$$
$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R)) \tag{2}$$
$$\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2 \tag{3}$$
$$\sigma_{R_1.A_1=R_2.A_2}(R_1 \times R_2) = R_1 \bowtie_{R_1.A_1=R_2.A_1} R_2 \tag{4}$$

Remarks:

- Rule (1) states that a conjunction in a selection predicate may be broken into several nested selections. At the same time, nested selections may be merged into a single selection with a conjunctive predicate.
- Rule (2) states that nested selections may swap places.
- Rule (3) states that a selection can be pushed down over a cross product, if it only requires the attributes of one of the operands. In the rule as stated above, we assume that predicate p only requires attributes from R_1. (We need to consult the data dictionary recording the name and attributes of each relation).
- Rule (4) describes how a selection and a cross product may be merged into a theta join, provided that the selection predicate is a join condition. This is the case if it compares attributes of R_1 and R_2.

Implement rule-based selection pushing on relational algebra queries. Proceed in these phases:

1. Complex selection predicates are broken up, according to rule (1).
2. All selections are pushed down as far as possible, according to rules (2) and (3).
3. Nested selections are merged again, according to rule (1).
4. Joins are introduced, where possible, according to (4).

Note that there are more rules for the logical optimization of relational algebra, such as projection pushing. For the second *miniHive* milestone, we will make do with this small set of optimization rules.

Write a Python module `raopt` that takes a relational algebra query as input, and that pushes selections. You may assume that the query is the result of the canonical translation of SQL into relational algebra, so it uses only the operators σ, π, ρ and $\times$.

Below is how it should work in the Python console. The data dictionary `dd` contains the relational schema and can be consulted during selection pushing.

```
>>> import radb.parse
>>> import raopt
>>>
>>> # The data dictionary describes the relational schema.
>>> dd = {}
>>> dd["Person"] = {"name": "string", "age": "integer", "gender": "string"}
>>> dd["Eats"] = {"name": "string", "pizza": "string"}
>>>
>>> stmt = """"\project_{Person.name, Eats.pizza}
...             \select_{Person.name = Eats.name}(Person \cross Eats);"""
>>> ra = radb.parse.one_statement_from_string(stmt)
>>>
>>> ra1 = raopt.rule_break_up_selections(ra)
>>> ra2 = raopt.rule_push_down_selections(ra1, dd)
>>> ra3 = raopt.rule_merge_selections(ra2)
>>> ra4 = raopt.rule_introduce_joins(ra3)
>>>
>>> print(ra4)
\project_{Person.name, Eats.pizza} (Person \join_{Person.name = Eats.name} Eats)
```

Test Suite. We provide a suite of unit tests at `https://github.com/miniHive/assignment/blob/master/milestone2/test_raopt.py`. Your solution must pass all tests.

Remarks: For Milestone 2, you are not required to make your implementation particularly efficient; instead, focus on correctness.

Do not implement a hard-coded solution, i.e., a solution that only works for the test cases provided. We will check all solutions for plagiarism using an automated tool.

Praktomat will run the suite of unit tests of `test_raopt.py` when you upload your solution. Upload `raopt.py` as a *single* file in Praktomat, in time for the deadline.

The deadline for submitting Milestone 2 is November 11, 2024, 12 noon. A successful submission that passes all public tests and passes the plagiarism check earns 5 points.
