# Computing Assignment 2

**Due Date:** May 6, 2020, 5:00pm

**Note.** Please note that all computing assignments are group assignments. Further note that for the grading we will apply a "10%" rule, i.e. the maximum number of points for this assignment is 165, but 150 will be counted as 100%. Points that exceed 150 will be stored in a separate counter and used later for compensation of lost points in other assignments or (if not used up this way) the final exam.

The task of this computing assignment is to efficiently implement (in **Python**) index structures for relations that are represented by lists of blocks, to implement queries on these relations in an efficient way and to produce graphs as output. For the processing of the queries the two-stack abstract machine (2SAM) developed in Computing Assignment 1 can be used. The relations provide a concrete object store, and the index structures and graph structures for some of the output are to be added.

Other design decisions on algorithms for the realisation of the queries are left open. It is possible to re-use the syntactic conventions from Computing Assignment 1 for the queries that are used as input for the abstract machine.

Points will be given for the correctness of the implementation (75), its efficiency (50), and the documentation of tests, for which relations have to be filled with some data (40). Points will be deducted for incompleteness (parts of the specification have been left out), errors in conception and implementation, and insufficient evidence of correctness.

**total points: 165**

## 1 Relations

**Note.** The description of relations here is re-used from Math 213 – Discrete Mathematics, Lecture 4, slides 103 ff. with some restrictions concerning keys and so-called foreign keys. Consider the following relations:

- Movie of arity 5, where Movie$(m, y, c, t, g)$ is to represent that there is a movie with *title* $m$, produced in *year* $y$ in *country* $c$, which has *run time* $t$, and *genre* $g$. The attributes *title* and *year* together shall form a key, i.e. there cannot be two 5-tuples in Movie that coincide on the first two components.
- Person of arity 4, where Person$(id, f, n, y)$ is to represent that there is a person with *identifier* $id$, *first_name* $f$, *last_name* $n$, and born in the *year* $y$. Here the attribute *identifier* is a key, i.e. there cannot be two 4-tuples in Person that coincide on the first component.
- Award of arity 3, where Award$(n, i, c)$ is to represent that there is an award with *name* $n$ by the *institution* $i$ in *country* $c$. Here the attribute *name* is a key, i.e. there cannot be two triples in Award that coincide on the first component.
- Restriction_Category of arity 2, where Restriction Category$(d, c)$ is to represent that in *country* $c$ there is a restriction with *description* $d$.

- Director of arity 3, where Director$(id, m, y)$ is to represent that a person with *identifier id* is director of a movie with *title m* produced in *year y*. Here, each value appearing as first component in a triple in Director must also appear as first component in a 4-tuple in Person, i.e. every director is a person, and every pair of values $(m, y)$ appearing as second and third component in a triple in Director must also appear as first and second component in a 5-tuple in Movie, i.e. every director is the director of a movie.
- Writer of arity 4, where Writer$(id, m, y, cr)$ is to represent that a person with *identifier id* is script writer of a movie with *title m* produced in *year y*, for which he has *credits cr*. Here, the attributes *identifier*, *title* and *year* together shall form a key, i.e. there cannot be two 4-tuples in Writer that coincide on the first three components.
  Furthermore, each value appearing as first component in a 4-tuple in Writer must also appear as first component in a 4-tuple in Person, i.e. every writer is a person, and every pair of values $(m, y)$ appearing as second and third component in a 4-tuple in Writer must also appear as first and second component in a 5-tuple in Movie, i.e. every writer is a writer of a movie script.
- Crew of arity 4, where Crew$(id, m, y, b)$ is to represent that a person with *identifier id* is a member of the crew of a movie with *title m* produced in *year y*, making the*contribution b*. Here, the attributes *identifier*, *title* and *year* together shall form a key, i.e. there cannot be two 4-tuples in Crew that coincide on the first three components.
  Furthermore, each value appearing as first component in a 4-tuple in Crew must also appear as first component in a 4-tuple in Person, i.e. every crew member is a person, and every pair of values $(m, y)$ appearing as second and third component in a 4-tuple in Crew must also appear as first and second component in a 5-tuple in Movie, i.e. every crew member works for a movie.
- Role of arity 5, where Role$(id, m, y, d, cr)$ is to represent that a person with *identifier id* plays a role with *description d* in a movie with *title m* produced in *year y*, for which he has *credits cr*. Here, the attributes *title*, *year* and *description* together shall form a key, i.e. there cannot be two 5-tuples in Role that coincide on the second, third and fourth components.
  Furthermore, each value appearing as first component in a 5-tuple in Role must also appear as first component in a 4-tuple in Person, i.e. every actor is a person, and every pair of values $(m, y)$ appearing as second and third component in a 5-tuple in Role must also appear as first and second component in a 5-tuple in Movie, i.e. every actor plays in a movie.
- Scene of arity 4, where Scene$(m, y, s, d)$ is to represent that in a movie with *title m* produced in *year y* there is a scene with *number s* and *description d*. Here, the attributes *title*, *year* and *number* together shall form a key, i.e. there cannot be two 4-tuples in Scene that coincide on the first three components.
  Furthermore, each pair of values $(m, y)$ appearing as first and second component in a 4-tuple in Scene must also appear as first and second component in a 5-tuple in Movie, i.e. every scene belongs to a movie.
- Restriction of arity 4, where Restriction$(m, y, d, c)$ states that for a movie with *title m* produced in *year y* the restriction with *description d* in *country c* applies. Here, each pair of values $(m, y)$ appearing as first and second component in a 4-tuple in Restriction must also appear as first and second component in a 5-tuple in Movie, i.e. every restriction belongs to a movie. Also every pair of values $(d, c)$ appearing as third and fourth component in a 4-tuple in Restriction must also appear as pair in Restric-

TION_CATEGORY, i.e. every restriction belongs to a category.

- APPEARANCE of arity 4, where APPEARANCE$(m, y, d, s)$ is to represent that a role with *description* $d$ appears in a scene with *number* $s$ in a movie with *title* $m$ produced in *year* $y$. Here, each triple $(m, y, d)$ appearing as first, second and third components in a 4-tuple in APPEARANCE must also appear as the second, third and fourth component of a 5-tuple in ROLE, and each triple $(m, y, s)$ appearing as first, second and fourth components in a 4-tuple in APPEARANCE must also appear as the the first three components of a 4-tuple in SCENE.
- MOVIE_AWARD of arity 6, where MOVIE_AWARD$(m, y, a, z, cg, r)$ is to represent that a movie with *title* $m$ produced in *year* $y$ received an award with *name* $a$ in the *year* $z$ in the *category* $cg$ with a *result* $r$. Here, the attributes *title*, *year* and *name* together shall form a key, i.e. there cannot be two 6-tuples in MOVIE_AWARD that coincide on the first three components.

  Furthermore, each pair of values $(m, y)$ appearing as first and second component in a 6-tuple in MOVIE_AWARD must also appear as first and second components in a 5-tuple in MOVIE, i.e. every movie award belongs to a movie, and each value $a$ appearing as third component in a 6-tuple in MOVIE_AWARD must also appear as first component in a triple in AWARD.
- CREW_AWARD of arity 7, where CREW_AWARD$(id, m, y, a, z, cg, r)$ is to represent that a crew member with *identity* $id$ of a movie with *title* $m$ produced in *year* $y$ received an award with *name* $a$ in the *year* $z$ in the *category* $cg$ with a *result* $r$. Here, the attributes *identifier*, *title*, *year* and *name* together shall form a key, i.e. there cannot be two 7-tuples in CREW_AWARD that coincide on the first four components.

  Furthermore, each triple $(id, m, y)$ appearing as the first three components in a 7-tuple in CREW_AWARD must also appear as the first three components in a tuple in CREW, and each value $a$ appearing as fourth component in a 7-tuple in CREW_AWARD must also appear as first component in a triple in AWARD.
- DIRECTOR_AWARD of arity 6, where DIRECTOR_AWARD$(m, y, a, z, cg, r)$ is to represent that the director of a movie with *title* $m$ produced in *year* $y$ received an award with *name* $a$ in the *year* $z$ in the *category* $cg$ with a *result* $r$. Here, the attributes *title*, *year* and *name* together shall form a key, i.e. there cannot be two 6-tuples in DIRECTOR_AWARD that coincide on the first three components.

  Furthermore, each pair of values $(m, y)$ appearing as first and second component in a 6-tuple in DIRECTOR_AWARD must also appear as second and third components in a triple in DIRECTOR, i.e. every director award belongs to a director, and each value $a$ appearing as third component in a 6-tuple in DIRECTOR_AWARD must also appear as first component in a triple in AWARD.
- WRITER_AWARD of arity 7, where WRITER_AWARD$(id, m, y, a, z, cg, r)$ is to represent that a script writer with *identity* $id$ of a movie with *title* $m$ produced in *year* $y$ received an award with *name* $a$ in the *year* $z$ in the *category* $cg$ with a *result* $r$. Here, the attributes *identifier*, *title*, *year* and *name* together shall form a key, i.e. there cannot be two 7-tuples in WRITER_AWARD that coincide on the first four components.

  Furthermore, each triple $(id, m, y)$ appearing as the first three components in a 7-tuple in WRITER_AWARD must also appear as the first three components in a tuple in WRITER, and each value $a$ appearing as fourth component in a 7-tuple in WRITER_AWARD must also appear as first component in a triple in AWARD.
- ACTOR_AWARD of arity 7, where ACTOR_AWARD$(id, m, y, a, z, cg, r)$ is to represent that

an actor with *identity id* of a movie with *title m* produced in *year y* received an award with *name a* in the *year z* in the *category cg* with a *result r*. Here, the attributes *identifier*, *title*, *year* and *name* together shall form a key, i.e. there cannot be two 7-tuples in ACTOR_AWARD that coincide on the first four components.
Furthermore, each triple $(id, m, y)$ appearing as the first three components in a 7-tuple in ACTOR_AWARD must also appear as the first three components in a tuple in ROLE, and each value $a$ appearing as fourth component in a 7-tuple in ACTOR_AWARD must also appear as first component in a triple in AWARD.

EXERCISE 1. Define and implement the basic data structure for the storage of relations:

**(i)** Represent each of these relations in a data structure that uses an ordered sequence of blocks, where each block is realised by an array.

**(ii)** Permit an additional overflow block for each block in the sequence.

**(iii)** Implement operations

- for the insertion and deletion of records;
- for the reorganisation of blocks by merging, sorting and splitting;
- for the retrieval of a record from a relation.

## 2 Queries

Next exploit the 2SAM implemented in the Computing Assignment 1 to realise some queries on an object store that is defined by the relations above.

All queries are taken from Lecture 4 of Math 213 – Discrete Mathematics (slides 109f.) or the associated Discussion 4.

EXERCISE 2. Implement the following queries on the relations above:

**(i)** List all pairs of actors who have played together in an awarded movie, but never appeared together in a scene.

**(ii)** List all 'comedy' movies that have been suggested for a US award in 2000 for one of their actors provided this actor has never played in an 'action' movie nor written or directed an Italian movie.

**(iii)** List all actors who never played in a movie suitable for small children.

**(iv)** Give a list of all movie pairs of the same genre produced in the same country and year with different directors and no common actor.

**Hint.** In the logical formalisation of these queries you can eliminate universal quantifiers and implications, so that the resulting query shows close similarity to operations using projection, selection and equi-join together with the algebraic operators EXIST and IN.

## 3 Index Structures

In a third step you are to realise index structure that will support the fast access to the relations to facilitate the processing of the queries. For this you have to identify primary and secondary keys and implement B+-trees, B-trees or hash tables.

EXERCISE 3. Implement index structures on the relations above and integrate them into the query processing:

- **(i)** Identify for each of the relations above a primary key and organise the data structure in Exercise 1 accordingly.
- **(ii)** Implement index structures for your primary keys using B+-trees or hash tables. Use at least once a B+-tree.
- **(iii)** Realise your operations from Exercise 1 using your index structure.
- **(iv)** Identify useful secondary keys and implement at least two index structures for them, using at least once a B-tree and at least once a hash table.
- **(v)** Integrate your index structures into the processing of the queries in Exercise 2.

## 4 Graph Queries

Finally, consider two queries that require an iteration until a fixed-point is reached. Define two different actors to be *associates* if they appeared together in the same scene of a movie. Further define two actors to be also associates, if one is an associate of an associate of the other.

Furthermore, define an actor $a$ to be an *admirer* of an actor $b$ iff $a$ and $b$ worked together in a movie and $b$ received an actor award (not necessarily for the joint movie). Further define an actor $a$ to be also an admirer of an actor $c$, if $a$ is an admirer of $b$ who is an admirer of $c$.

EXERCISE 4. Implement the following fixed-point queries and represent the results by undirected or directed graphs containing keys of records in one of the above relations:

- **(i)** Determine all pairs of actors that are associates.
- **(ii)** Determine all pairs $(a, b)$ of persons, where $a$ is an admirer of $b$.