SQL Automarker

Introduction

- Goals of Project
    - What we want to achieve
- Existing Automarker
- The Team

Project Goals

- “Deployment” of existing automarker
  - Building of application layer around automarker
  - Single responsibility principle
  - Avoid complete rewrites

- “End-to-end” automarking system
  - From downloading submissions, to marking, to delivering grades

- Modularity
  - “Plug & Play” with various platforms
  - Easy for other universities to adopt

The Existing Automarker

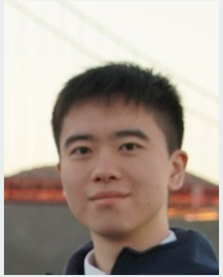
- Support varieties of marking styles and database systems
- Provide feedback to students based on the text distance per question
- Not enough documentation
- Not generalized for all purpose use
- Lack of useful feedback for the instructor
- Lack of supporting multi-solution for the same question

The Team

Shahmeer Shahid

Engineering Team

Architecture, Development, Leadership



Sandy Wang

Research / Engineering Team

ERD Automarker, Word Meaning Comparison, SQL Automarker (SQAM)

Erik Holmes

Engineering Team

SQL Automarker (SQAM)

Alberto Gateno

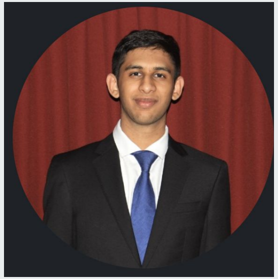
Research Team

ERD Marker, Quantitative Graph Comparison

Jarrod Servilla

Engineering Team

Admin Site, Admin API



Vaishvik Maisuria

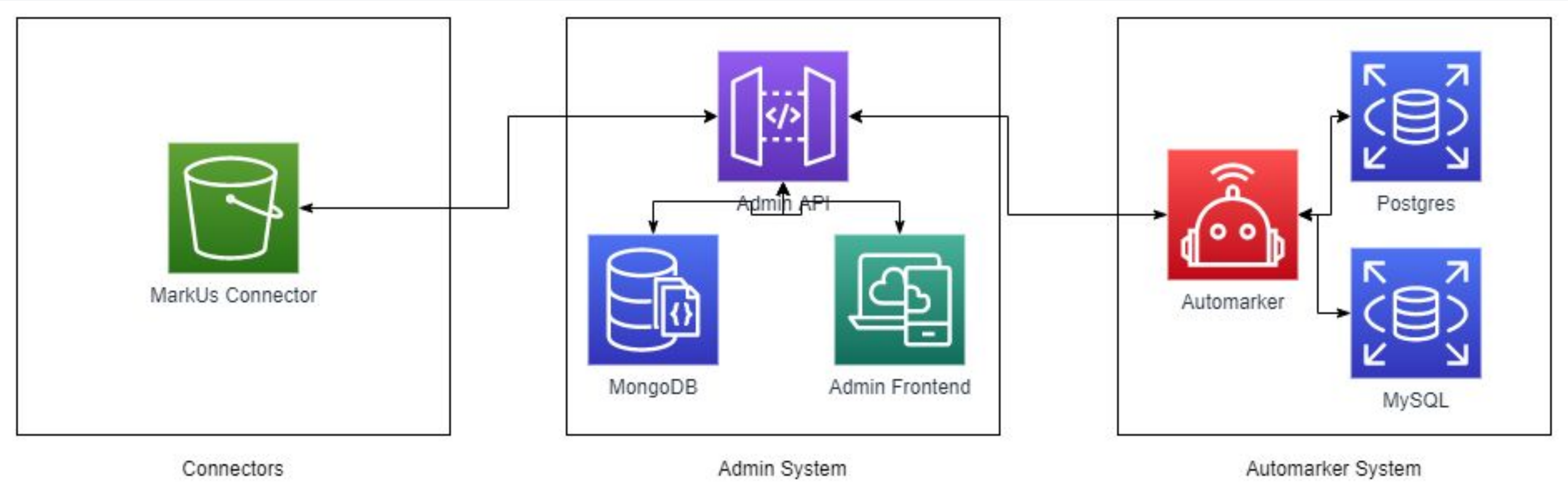
Engineering & Research Team

Admin Site, Admin API, ERD Marker

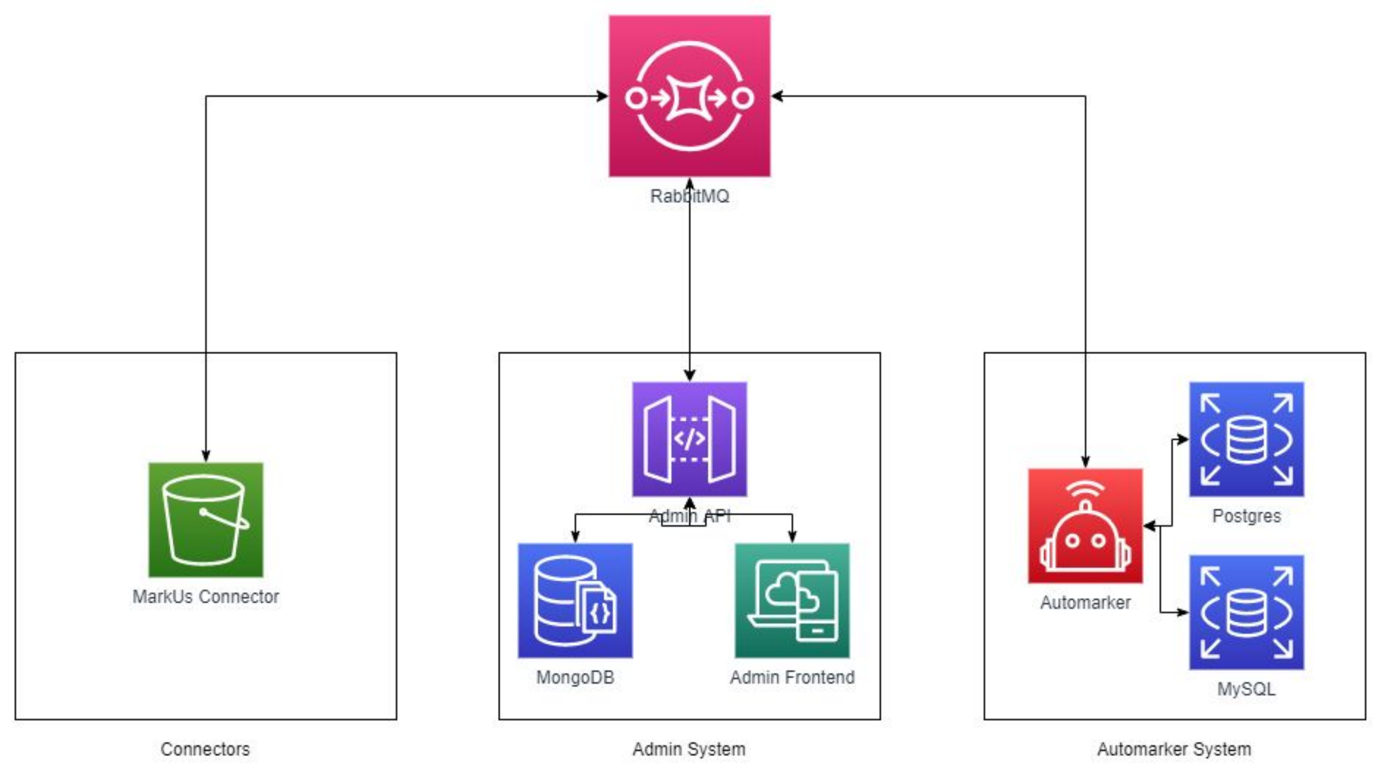
SQAM Overview

- Architectural Overview
  - Architecture
  - Microservices
  - Docker
  - RabbitMQ
- Automarker
  - Design
    - Tasks
    - Queriers
    - Graders
  - Additional Improvements
- Admin API
  - Overview
  - Technologies
- Admin Frontend
  - React
  - Views
- Connectors
  - Standard

Architectural Overview - Initial Design



Architectural Overview - Improved Design



Architectural Overview

- Microservices vs Monolith
- Containerization
- Event driven architecture

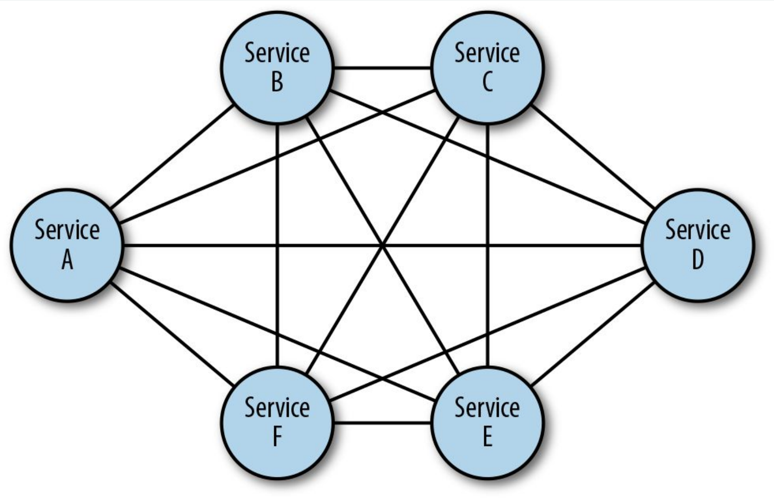
Architectural Overview - Microservices

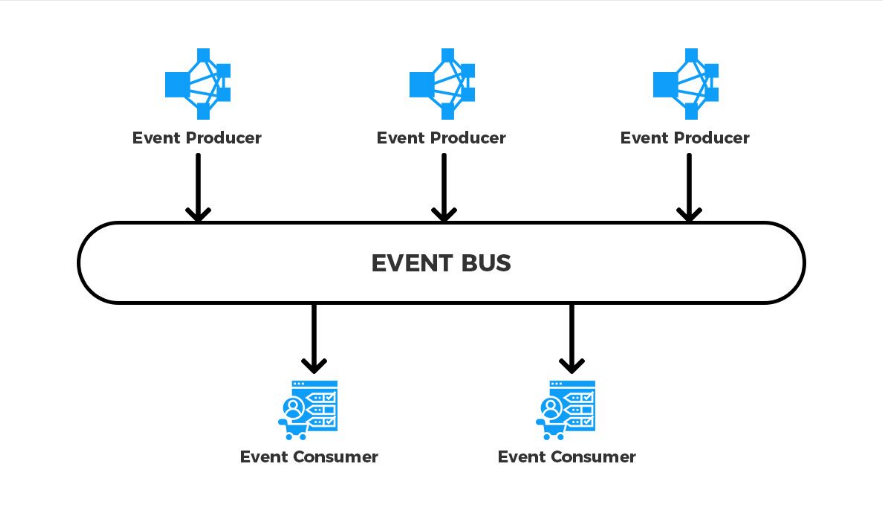
- Avoid large restructuring of codebase
  - Python for automarker, Node for other services

- Conway's law
  - "Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure."
  - No cross contamination in codebase

- Enabled by tools like Docker

Architectural Overview - Event Driven Architecture

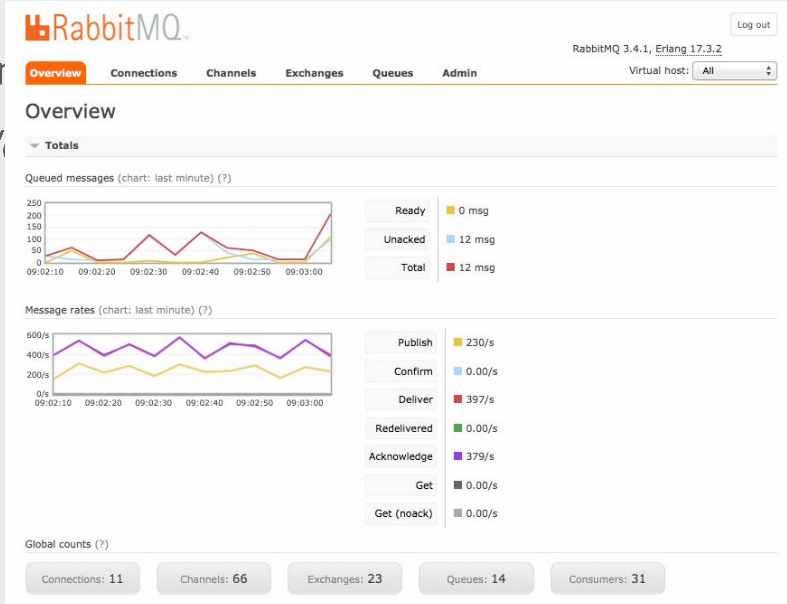




Architectural Overview - Event Driven Architecture

RabbitMQ

- Good library support across lar
- Admin interface for messages/
- Commonly used in industry



RabbitMQ 3.4.1, Erlang 17.3.2

Overview

Queued messages (chart: last minute) (?)

Ready	🟨 0 msg
Unacked	🟦 12 msg
Total	🟥 12 msg

Message rates (chart: last minute) (?)

Publish	🟨 230/s
Confirm	🟦 0.00/s
Deliver	🟥 397/s
Redelivered	🟩 0.00/s
Acknowledge	🟪 379/s
Get	■ 0.00/s
Get (noack)	■ 0.00/s

Global counts (?)

Connections: 11 Channels: 66 Exchanges: 23 Queues: 14 Consumers: 31

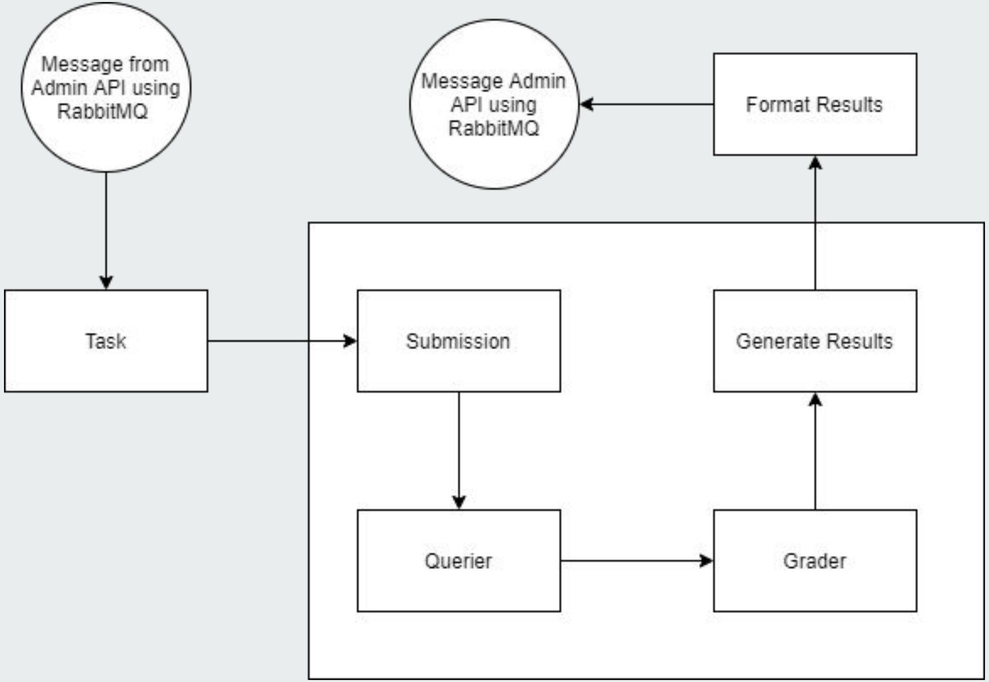
Overview of Work Done on the Automarker

- Dockerize into Flask Application
- Remove hardcoded configuration settings
- Remove coupling with Markus submission format
- Remove coupling with UAM
- Refactor assignments into more abstract Tasks

- Refactor Queriers
- Improve Logging
- Improve Error Handling
- Implement RabbitMQ
- Began refactoring Graders

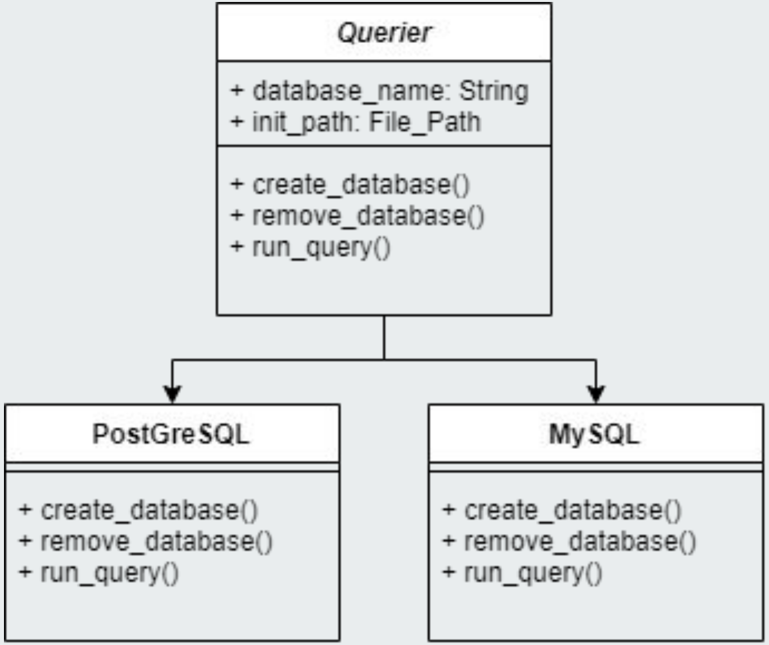
Automarker Structure

- Queriers
- Graders
- Tasks


Message from Admin API using RabbitMQ
Task
Submission
Querier
Grader
Generate Results
Format Results
Message Admin API using RabbitMQ

Queriers

- Connection to Databases
- Used to execute Queries
- Currently Supported:
    - PostGreSQL
    - MYSQL



Querier

+ database_name: String
+ init_path: File_Path

+ create_database()
+ remove_database()
+ run_query()

PostGreSQL

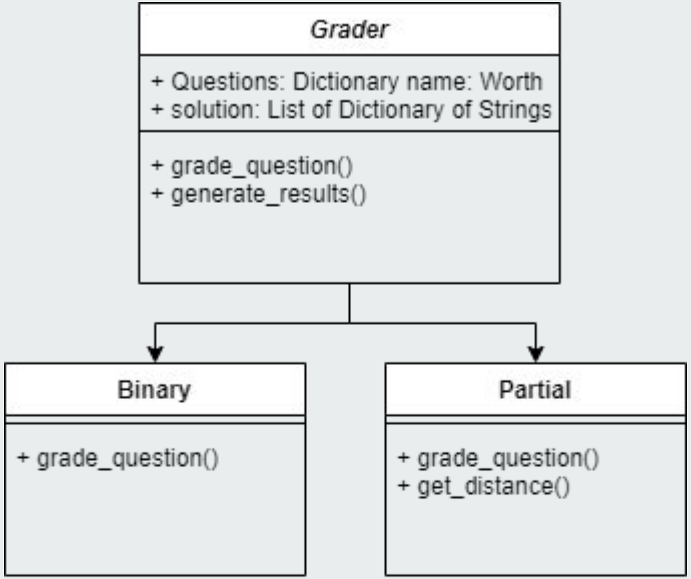
+ create_database()
+ remove_database()
+ run_query()

MySQL

+ create_database()
+ remove_database()
+ run_query()

Graders

- Provides Grade and Feedback
- Currently Supported:
  - Binary Grader
  - Partial Mark Grader
    - Levenshtein Distance
    - Jaro-Winkler Distance
    - Ratcliff Obershelp Pattern Recognition



Grader

+ Questions: Dictionary name: Worth
+ solution: List of Dictionary of Strings

+ grade_question()
+ generate_results()

Binary

+ grade_question()

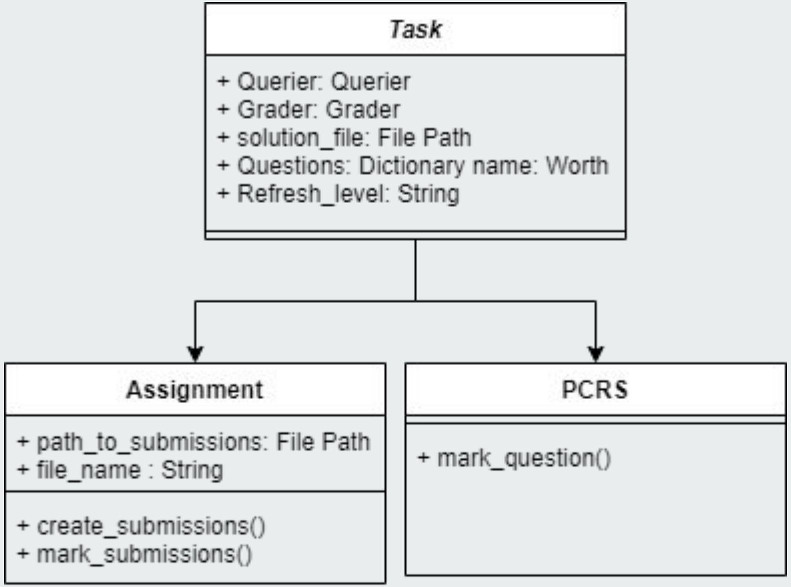
Partial

+ grade_question()
+ get_distance()

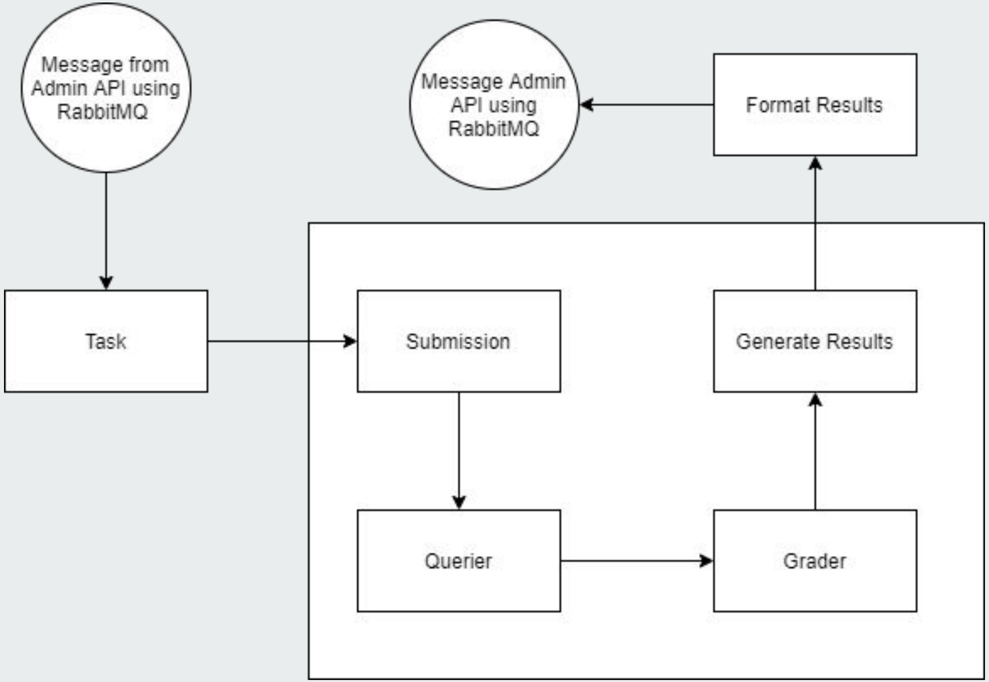
Tasks

- Collects Submissions then uses Querier and Grader to provide feedback

- Currently Supported:
  - Assignments
  - Future work - PCRS



Automarker Structure

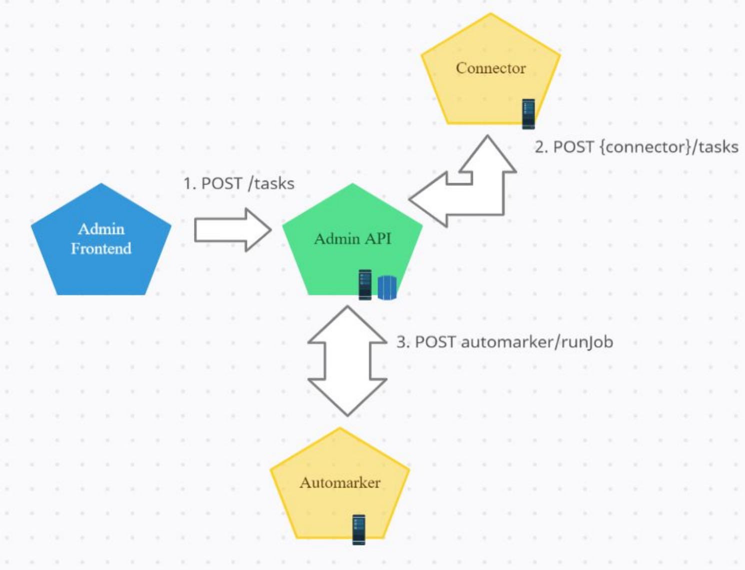

Message from Admin API using RabbitMQ
Message Admin API using RabbitMQ
Format Results
Task
Submission
Generate Results
Querier
Grader

Additional Improvements

- Support multiple query answers - E.g Use of Temporary Tables
- Support multiple correct solutions
    - Including the ability to allow any solution to be correct
- Database Refresh Level
- Configuration option to mark using column names

Admin API

- Serves as an API gateway to handle communication between all microservices
- Manages tasks, submissions, and logs



Admin API

- Tasks required a flexible schema due to the additional information required for each connector ex. MarkUs requires the markus URL, assignment_id, and api_key
- This additional information is stored in tasks as the “extra_fields” property
- MongoDB was employed to allow for this flexibility

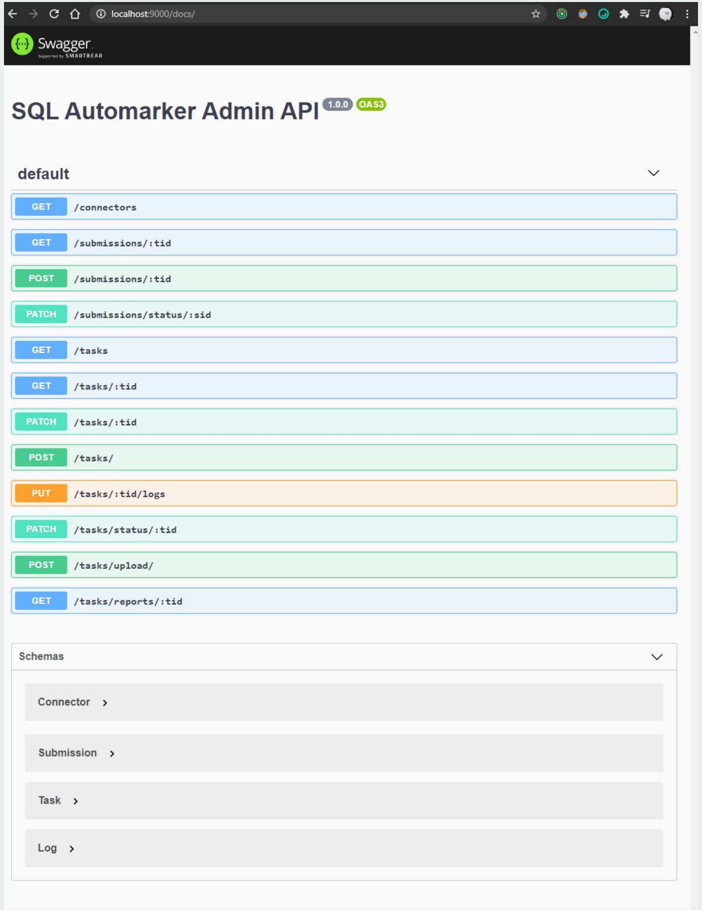
Admin API

- The admin API was built with Typescript, Node.js, Express, MongoDB, and Mongoose.
- Typescript was chosen for it’s strongly-typed nature, allowing for stronger debugging and bug prevention.
- Node/Express was selected primarily for its compatibility with NoSQL databases and to allow for fast, asynchronous operations
- Mongoose is an ODM library allowing for elegant object modelling

Admin API

- Documentation interface accessible at http://localhost:9000/docs/

- Provides an interactive, detailed overview of the endpoints and schemas created using Swagger UI.



localhost:9000/docs/

Swagger

SQL Automarker Admin API 1.0.0 OAS3

default

GET	/connectors
GET	/submissions/:tid
POST	/submissions/:tid
PATCH	/submissions/status/:sid
GET	/tasks
GET	/tasks/:tid
PATCH	/tasks/:tid
POST	/tasks/
PUT	/tasks/:tid/logs
PATCH	/tasks/status/:tid
POST	/tasks/upload/
GET	/tasks/reports/:tid

Schemas

- Connector
- Submission
- Task
- Log

Frontend

An admin site was developed for the purpose of configuring/enqueueing SQL marking jobs, as well as viewing information about the job's status.

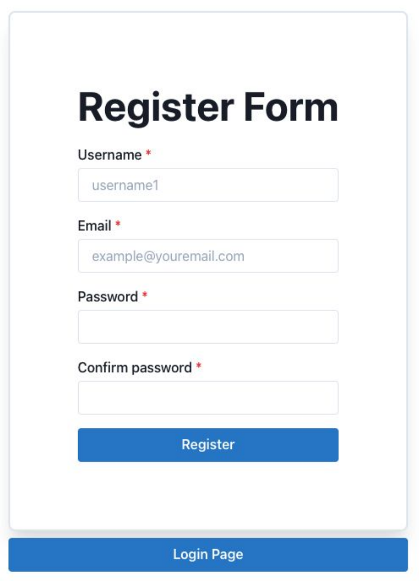
It enables users to:

1. Select a file repository to pull from (Markus, Github, ..)
2. Fill in the required information for that assessment (No. Questions, Marks, …)
3. Run the automarker with that job with a push of a button
4. View the results of marking jobs

Frontend

1. The admin site was developed with React, Chakra-UI, Formik, and Yup. For testing, we employed React-Testing-Library, known for its philosophy of testing user behaviour.
2. Includes functionality to smoothen the process of uploading and configuring the marking process.
3. All components in the frontend have test files verifying its functionality.

Frontend - User Authentication



Register Form

Username *

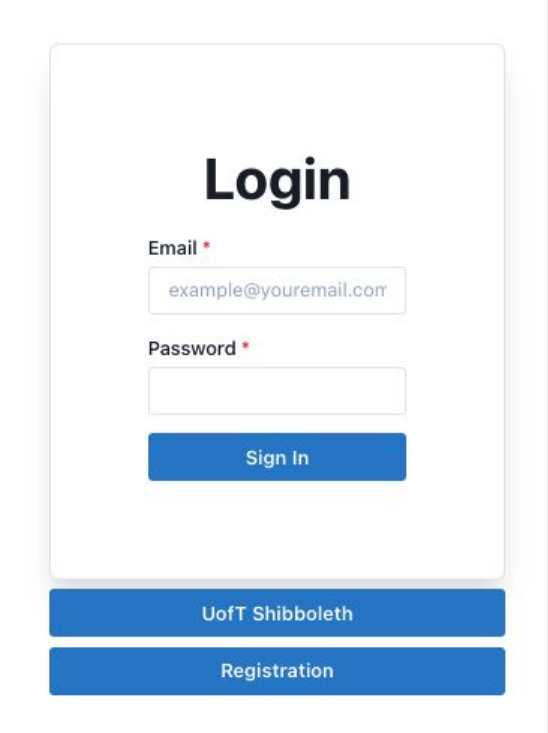
username1

Email *

example@youremail.com

Password *

Confirm password *

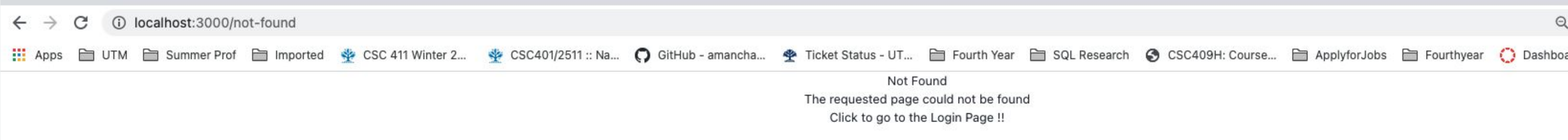


Login

Email *

example@youremail.com

Password *



Not Found

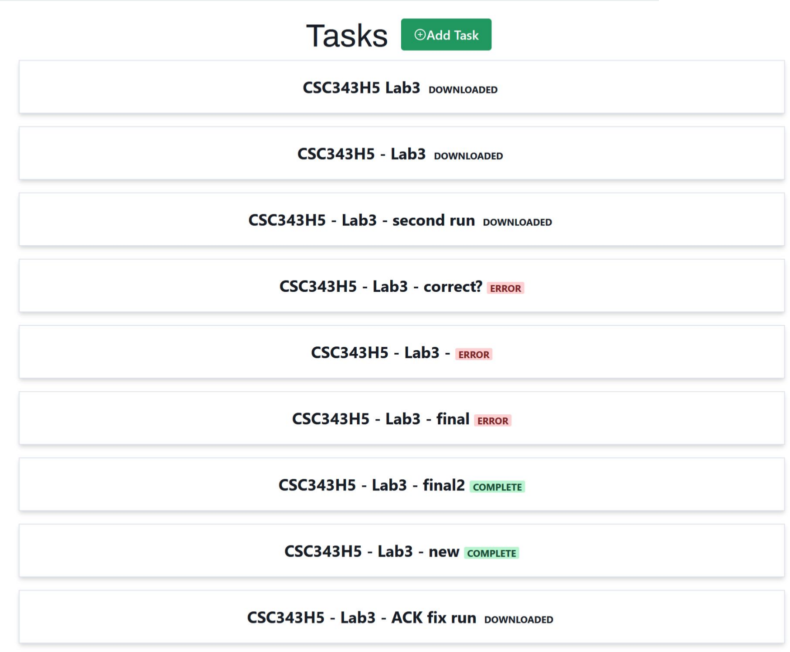
The requested page could not be found

Click to go to the Login Page !!

User Authentication - Technologies

- Work resides in branch **77-admin-fix-login-ui-and-create-shibboleth-button**
- Authentication via Shibboleth done using Passport.js and JWT
- When deployed, it will be accessible at http://csc398dev.utm.utoronto.ca/sqam/homepage

Frontend - Home Page



Tasks ⊕Add Task

CSC343H5 Lab3 DOWNLOADED

CSC343H5 - Lab3 DOWNLOADED

CSC343H5 - Lab3 - second run DOWNLOADED

CSC343H5 - Lab3 - correct? ERROR

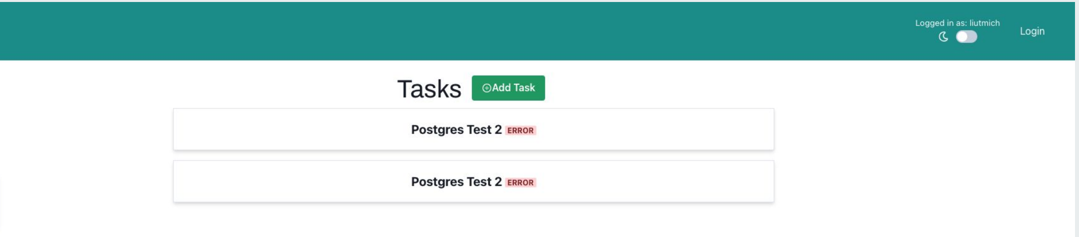
CSC343H5 - Lab3 - ERROR

CSC343H5 - Lab3 - final ERROR

CSC343H5 - Lab3 - final2 COMPLETE

CSC343H5 - Lab3 - new COMPLETE

CSC343H5 - Lab3 - ACK fix run DOWNLOADED



SQAM Tasks

Logged in as: liutmich

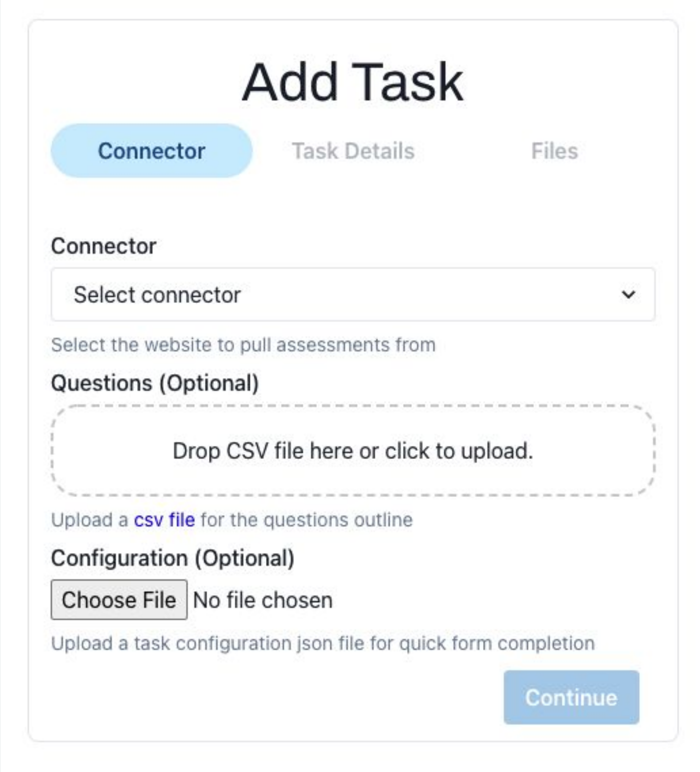
Login

Tasks ⊕Add Task

Postgres Test 2 ERROR

Postgres Test 2 ERROR

Frontend - Task Page



Add Task

Connector

Select connector

Select the website to pull assessments from

Questions (Optional)

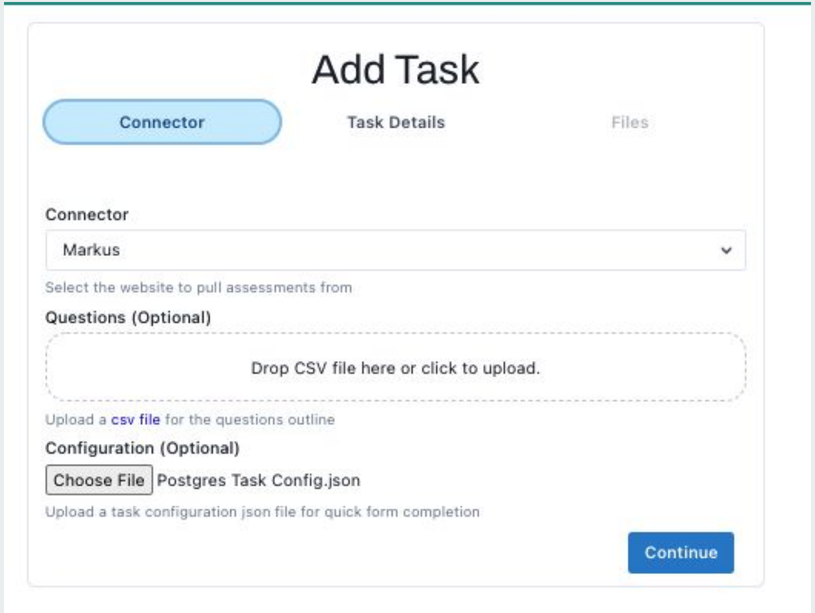
Drop CSV file here or click to upload.

Upload a csv file for the questions outline

Configuration (Optional)

Choose File No file chosen

Upload a task configuration json file for quick form completion



Add Task

Connector

Markus

Select the website to pull assessments from

Questions (Optional)

Drop CSV file here or click to upload.

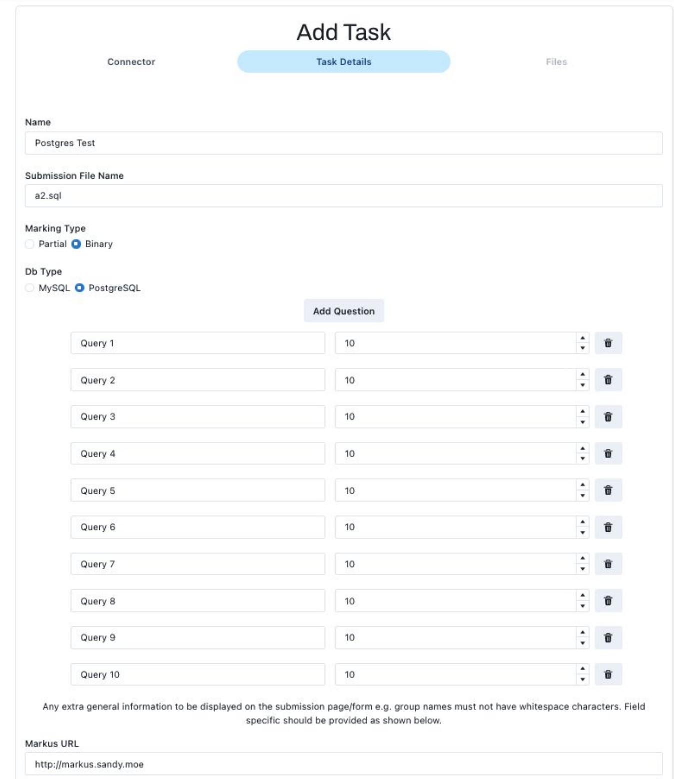
Upload a csv file for the questions outline

Configuration (Optional)

Choose File Postgres Task Config.json

Upload a task configuration json file for quick form completion

Frontend - Task Page



Add Task

Name

Postgres Test

Submission File Name

a2.sql

Marking Type

○ Partial ● Binary

Db Type

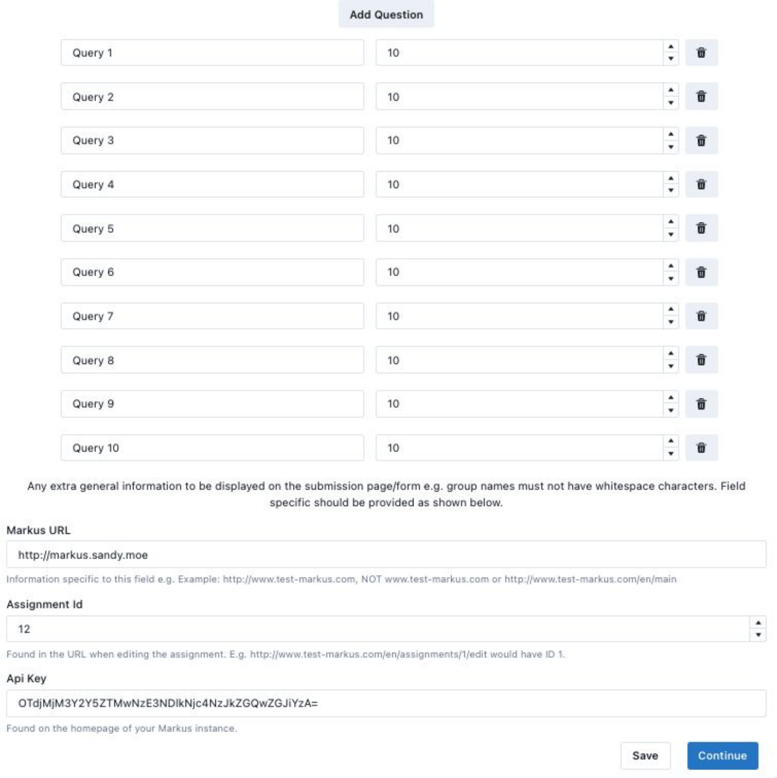
○ MySQL ● PostgreSQL

Query 1	10
Query 2	10
Query 3	10
Query 4	10
Query 5	10
Query 6	10
Query 7	10
Query 8	10
Query 9	10
Query 10	10

Any extra general information to be displayed on the submission page/form e.g. group names must not have whitespace characters. Field specific should be provided as shown below.

Markus URL

http://markus.sandy.moe



Query 1	10
Query 2	10
Query 3	10
Query 4	10
Query 5	10
Query 6	10
Query 7	10
Query 8	10
Query 9	10
Query 10	10

Any extra general information to be displayed on the submission page/form e.g. group names must not have whitespace characters. Field specific should be provided as shown below.

Markus URL

http://markus.sandy.moe

Information specific to this field e.g. Example: http://www.test-markus.com, NOT www.test-markus.com or http://www.test-markus.com/en/main

Assignment Id

12

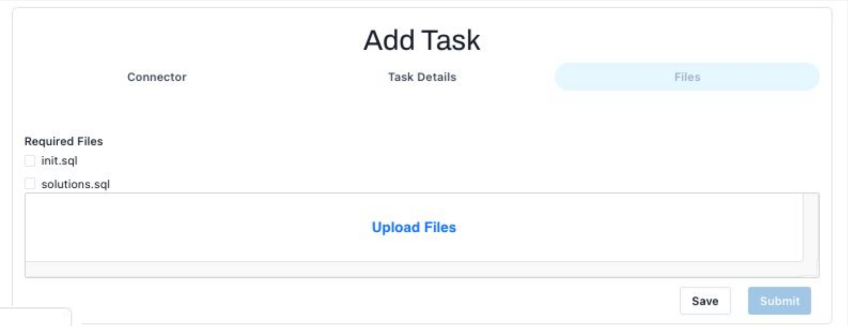
Found in the URL when editing the assignment. E.g. http://www.test-markus.com/en/assignments/1/edit would have ID 1.

Api Key

OTdjMjM3Y2Y5ZTMwNzE3NDlkNjc4NzJkZGQwZGJiYzA=

Found on the homepage of your Markus instance.

Frontend - Task Page

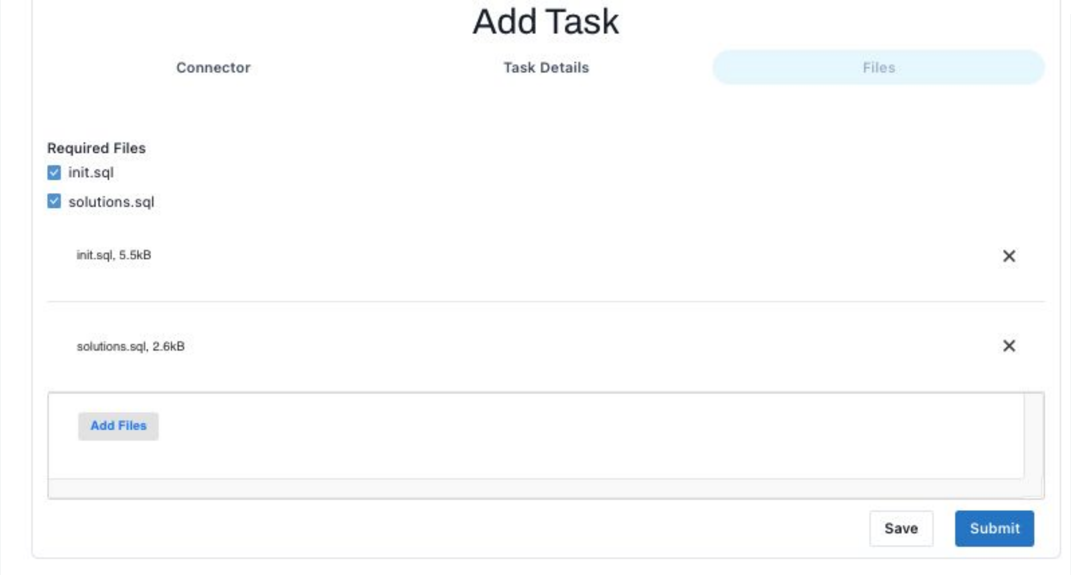


Add Task

Required Files

□ init.sql

□ solutions.sql



Add Task

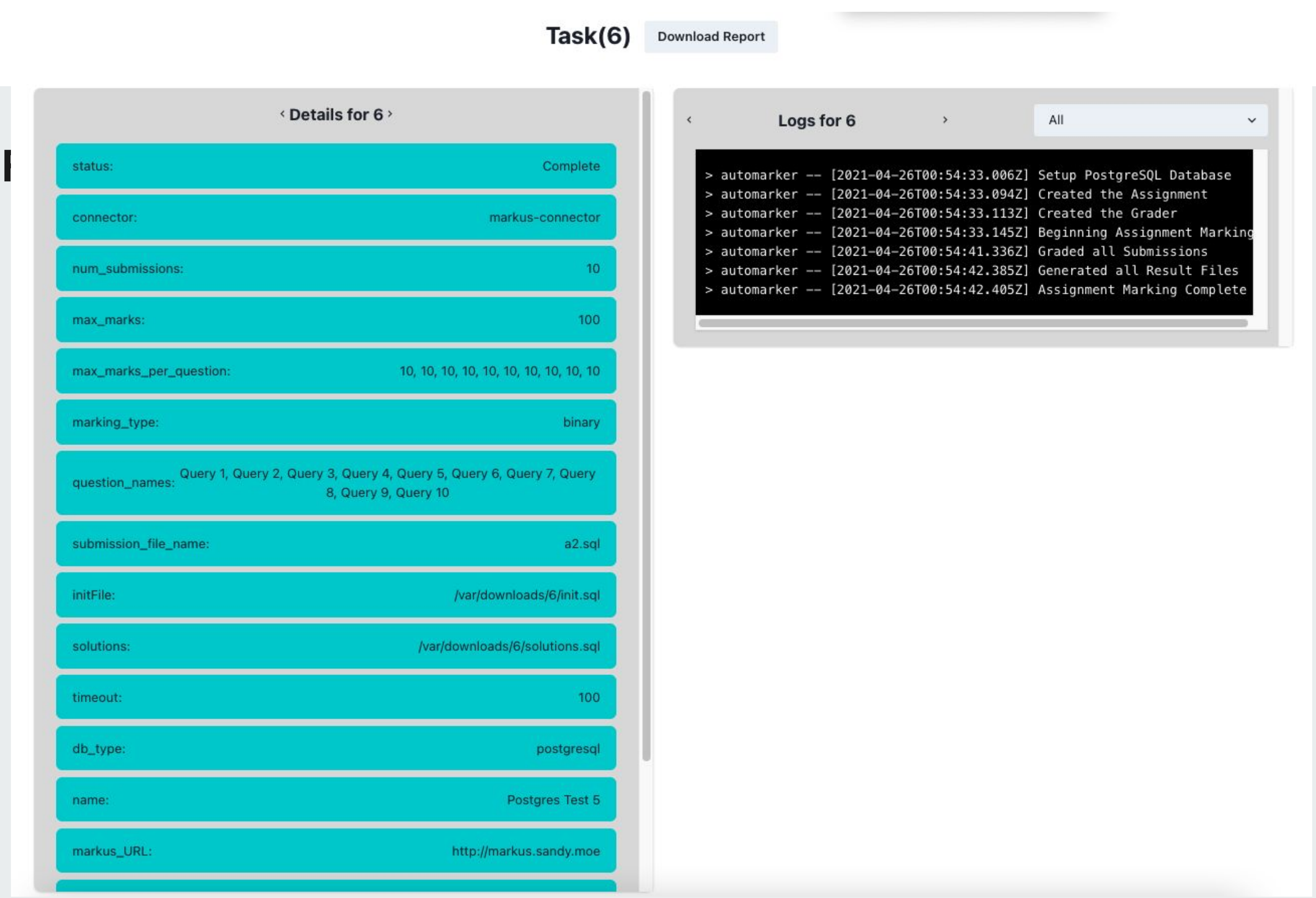
Required Files

☑ init.sql

☑ solutions.sql

init.sql, 5.5kB ✕

solutions.sql, 2.6kB ✕



status: Complete

connector: markus-connector

num_submissions: 10

max_marks: 100

max_marks_per_question: 10, 10, 10, 10, 10, 10, 10, 10, 10, 10

marking_type: binary

question_names: Query 1, Query 2, Query 3, Query 4, Query 5, Query 6, Query 7, Query 8, Query 9, Query 10

submission_file_name: a2.sql

initFile: /var/downloads/6/init.sql

solutions: /var/downloads/6/solutions.sql

timeout: 100

db_type: postgresql

name: Postgres Test 5

markus_URL: http://markus.sandy.moe

```
> automarker -- [2021-04-26T00:54:33.006Z] Setup PostgreSQL Database
> automarker -- [2021-04-26T00:54:33.094Z] Created the Assignment
> automarker -- [2021-04-26T00:54:33.113Z] Created the Grader
> automarker -- [2021-04-26T00:54:33.145Z] Beginning Assignment Marking
> automarker -- [2021-04-26T00:54:41.336Z] Graded all Submissions
> automarker -- [2021-04-26T00:54:42.385Z] Generated all Result Files
> automarker -- [2021-04-26T00:54:42.405Z] Assignment Marking Complete
```

Connectors

- Connecting a Learning Management System (LMS) to the SQAM's admin API and automarker.
- Submission is accessible to the automarker by sharing the volume
- Be able to commit the feedback to the LMS/Repository (with limited support LMS)
- Modularized
- Current developed connector:
  - MarkUs

Live Demo

Results

- Which goals we achieved
- Future Work

Reflecting on Goals

- Deployment of automarker
- End to end system
- Modularity

Future Work

- PCRS integration
- Integrate user authentication work
- Further research into string similarity marking
- Refactor grading to use less memory
- Save grading results in a database
- CI/CD
- Provide more configuration for each question

Research

- Graph Comparison
    - Overview
    - Tree Sorting/Labelling
    - Loop Unwinding
    - Sample Outputs
- Computer Vision
- Word recognition & abbreviation
- Demo

Algorithm Overview

- ***How much of the intended database does the student's answer model?***
  - For graphs: How much is one graph able to be embedded in the other?
  - Heuristic: it's simpler to study tree embedding.

- ***How can we award partial marks for partially-correct work?***
  - For graphs: How do we quantitatively assess the degree to which one graph is isomorphic to another?
  - Heuristic: tree traversals can give us standard ways to treat graphs as sequential data, for which we have good quantitative similarity metrics.

- ***How can we account for graphical variations in correct answers?***
  - For graphs: How do we make sure our comparison is invariant under graph relabelling?
  - Heuristic: Similarity should approach 100% as the graphs become closer to being isomorphic, which is by its nature independent of labelling.

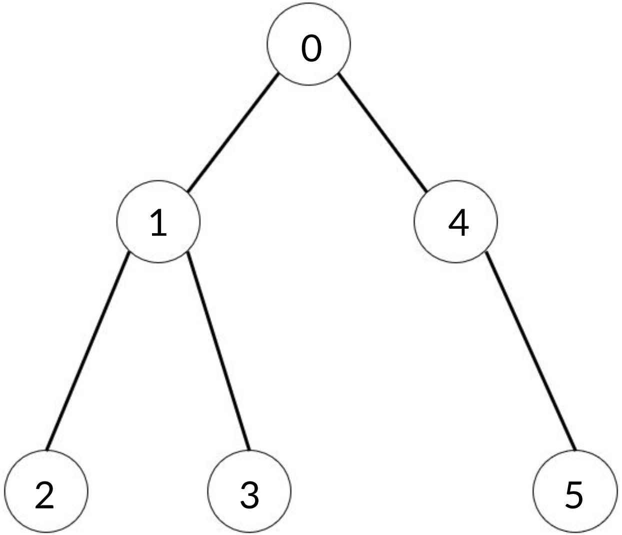
Algorithm Overview

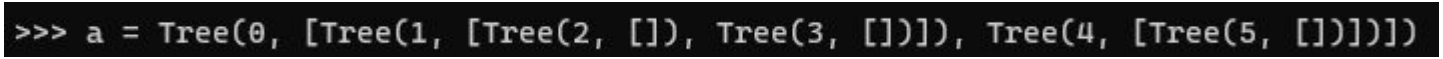
- Separate connected components

- For each connected component:
  - Perform loop unwinding to convert to an acyclic graph
  - Convert the acyclic graph to a tree
  - Perform tree sorting and obtain a list representation
  - Obtain a comparison score using the longest common subsequence
  - Obtain a penalty from the comparison score and apply it based on the graph edit distance between the two components

- Perform bipartite optimization to maximize average comparison score between pairs

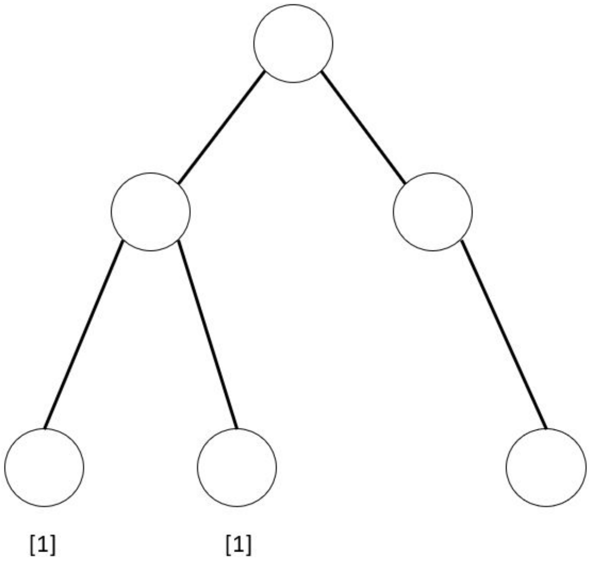
Tree Sorting and Labelling



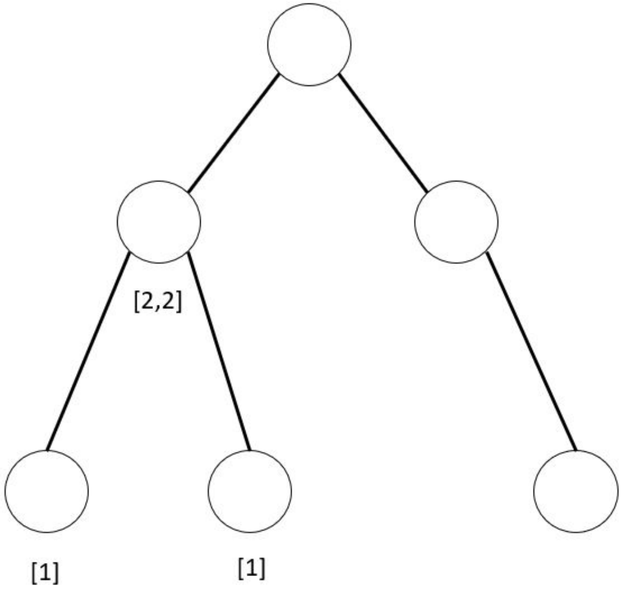


```
>>> a = Tree(0, [Tree(1, [Tree(2, []), Tree(3, [])]), Tree(4, [Tree(5, [])])])
```

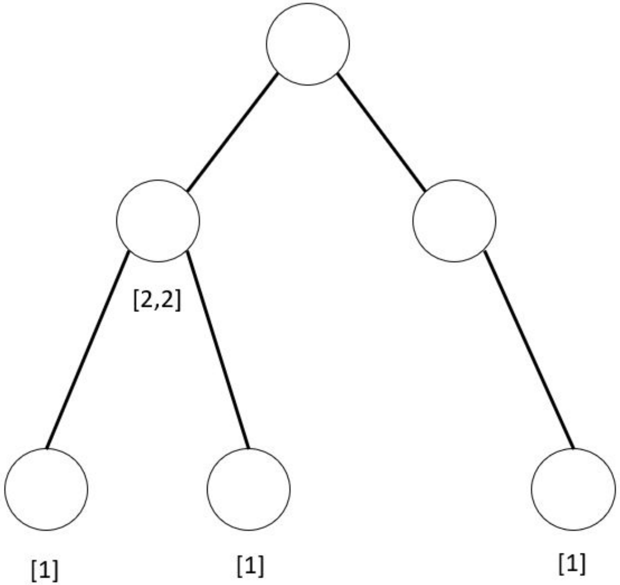
Tree Sorting and Labelling



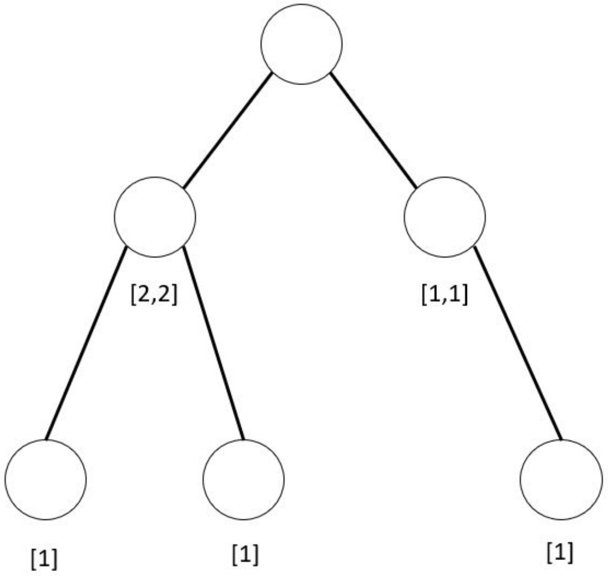
Tree Sorting and Labelling



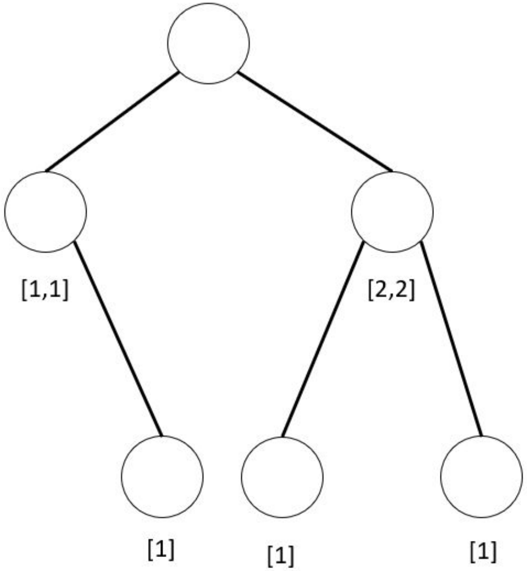
Tree Sorting and Labelling



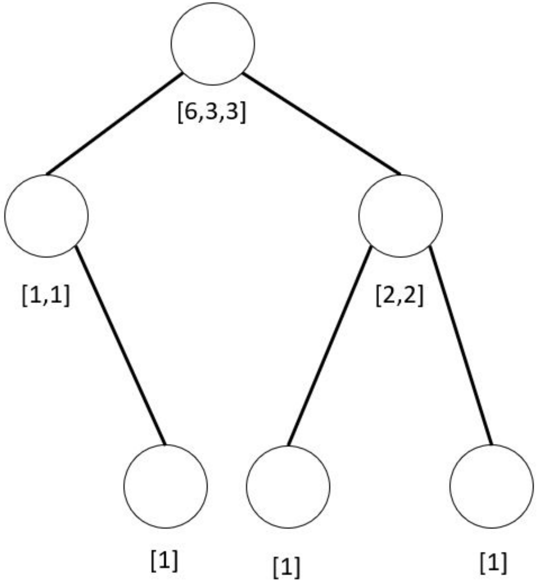
Tree Sorting and Labelling



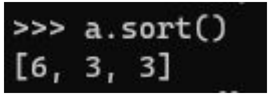
Tree Sorting and Labelling



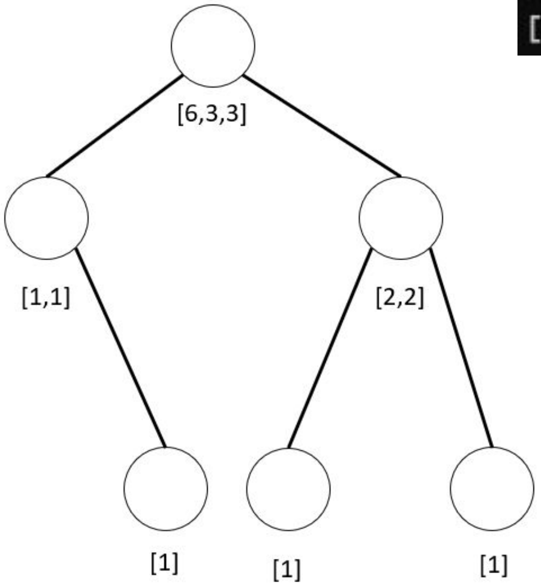
Tree Sorting and Labelling



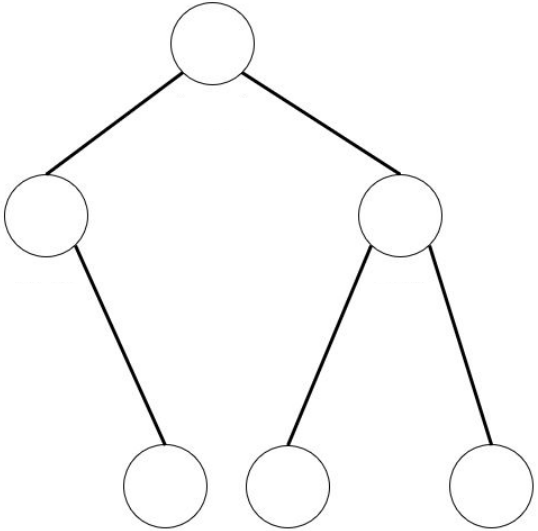
Tree Sorting and Labelling



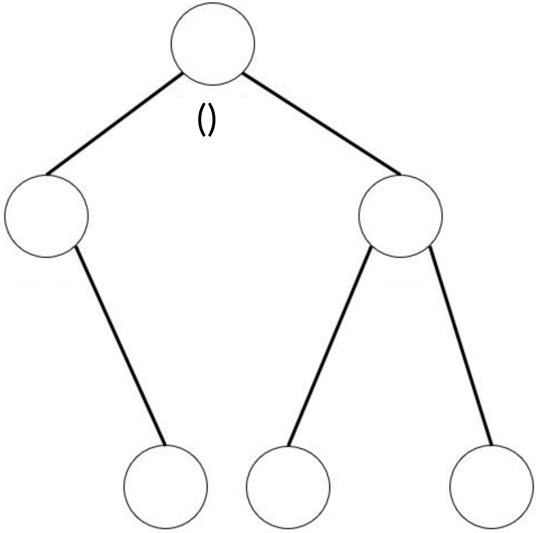
```
>>> a.sort()
[6, 3, 3]
```



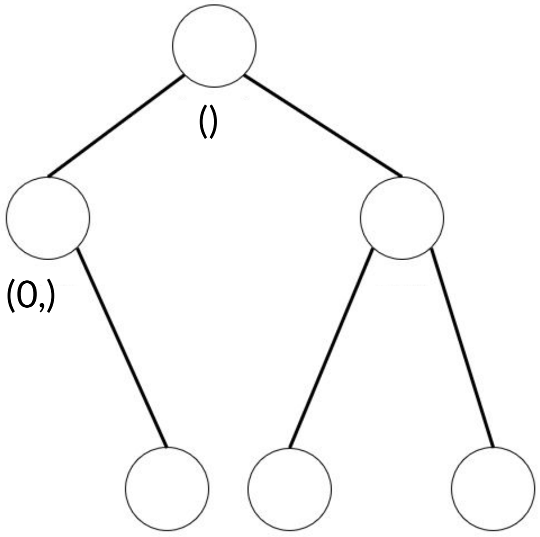
Tree Sorting and Labelling



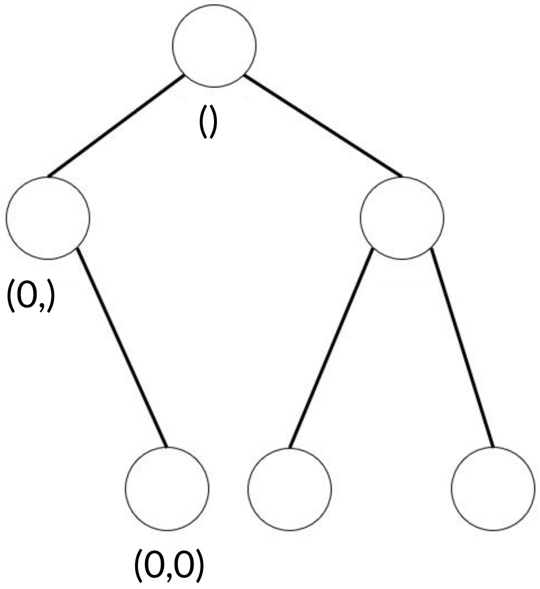
Tree Sorting and Labelling



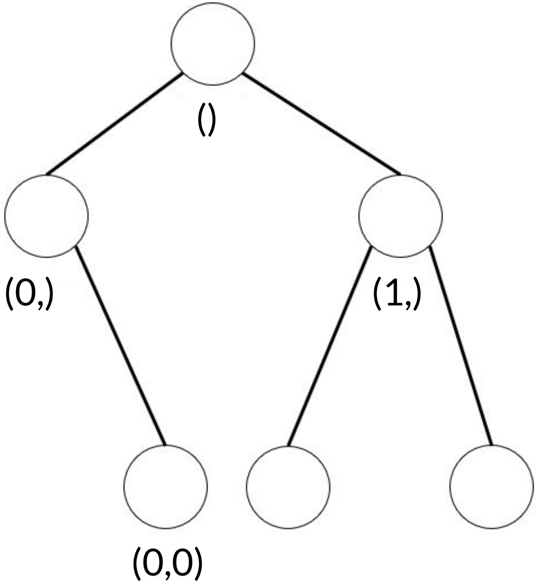
Tree Sorting and Labelling



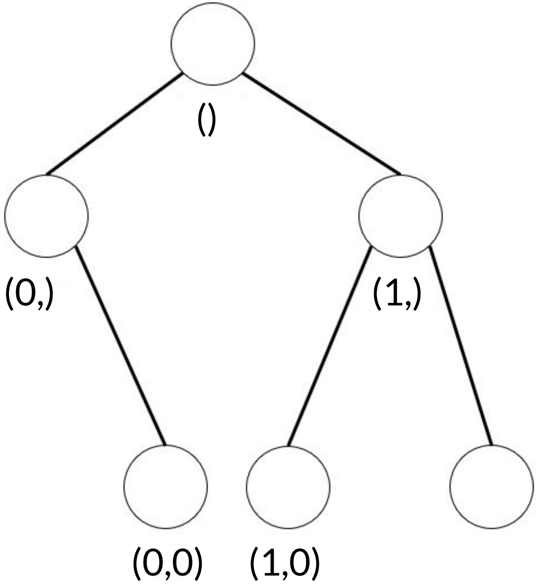
Tree Sorting and Labelling



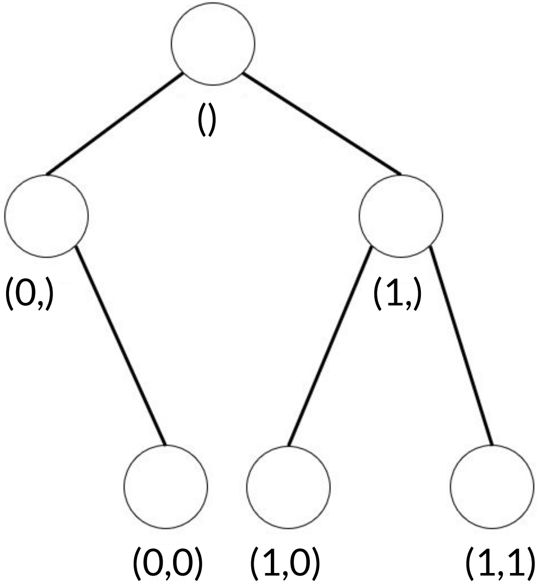
Tree Sorting and Labelling



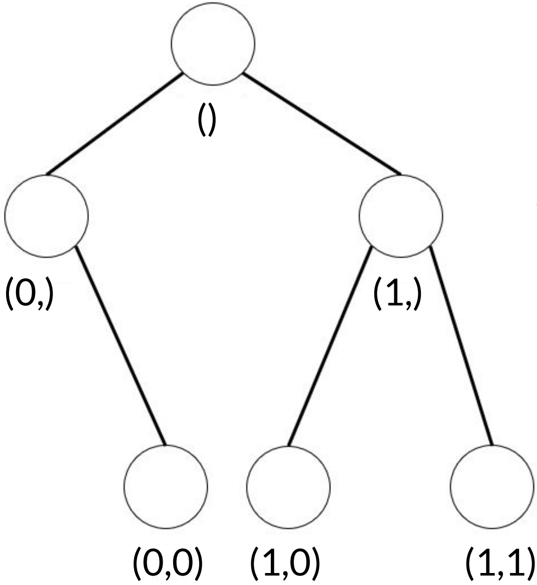
Tree Sorting and Labelling



Tree Sorting and Labelling

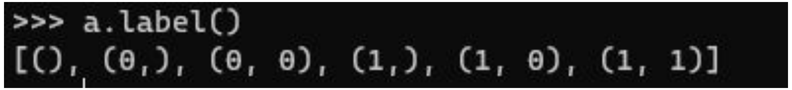


Tree Sorting and Labelling

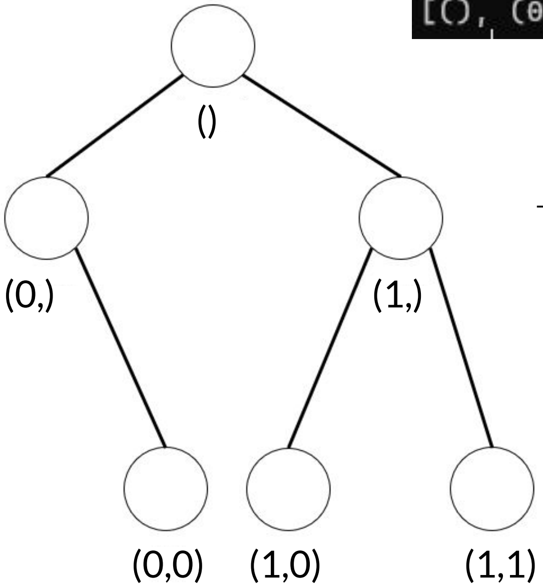


→ [(), (0,), (0,0), (1,), (1,0), (1,1)]

Tree Sorting and Labelling

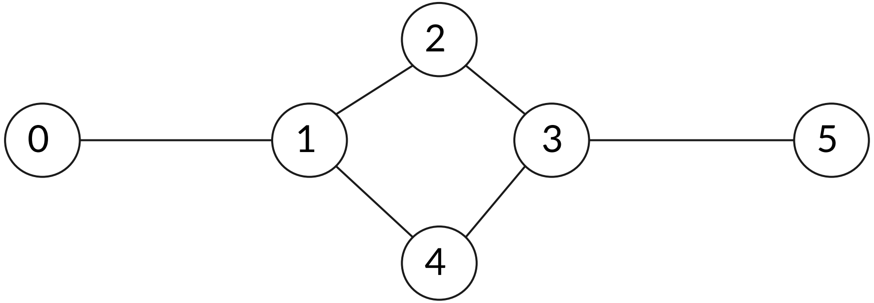


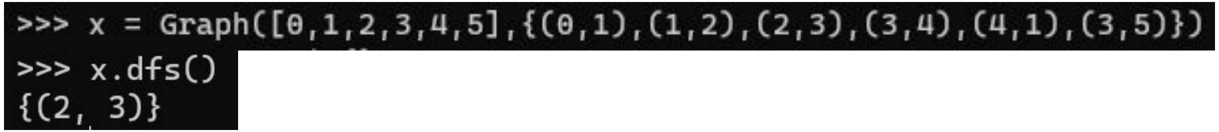
```
>>> a.label()
[(), (0,), (0, 0), (1,), (1, 0), (1, 1)]
```



→ [(), (0,), (0,0), (1,), (1,0), (1,1)]

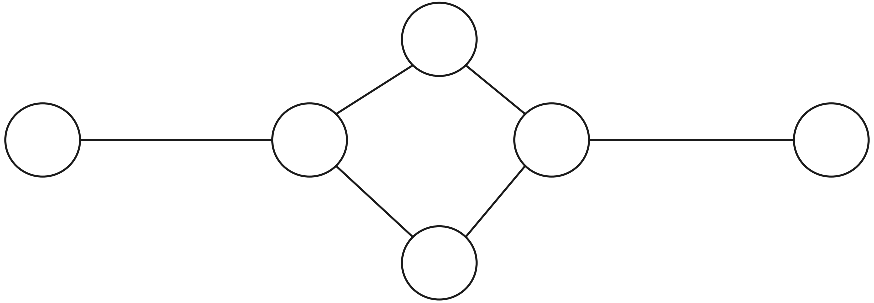
Loop Unwinding



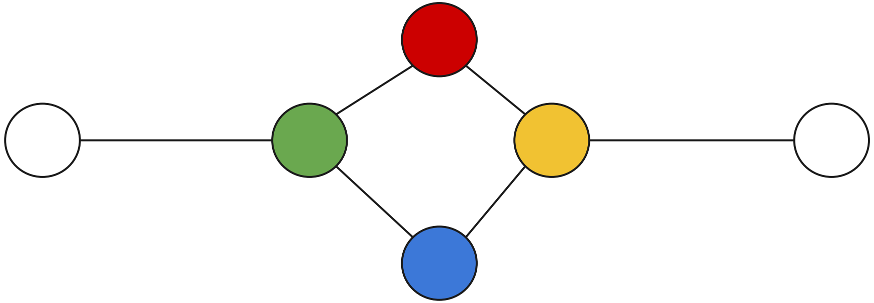


```
>>> x = Graph([0,1,2,3,4,5],{(0,1),(1,2),(2,3),(3,4),(4,1),(3,5)})
>>> x.dfs()
{(2, 3)}
```

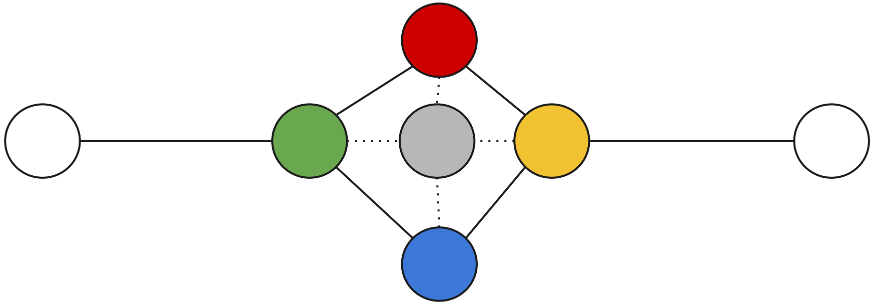
Loop Unwinding



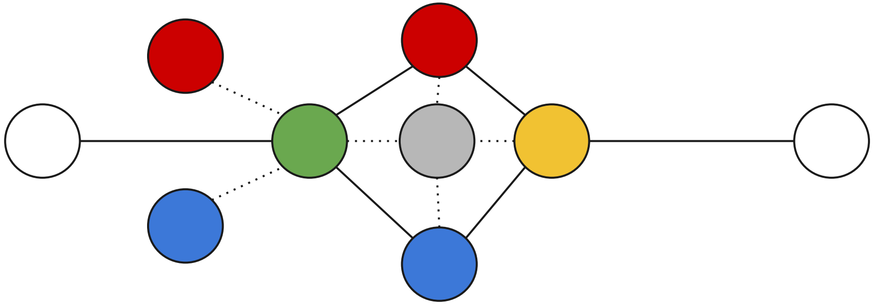
Loop Unwinding



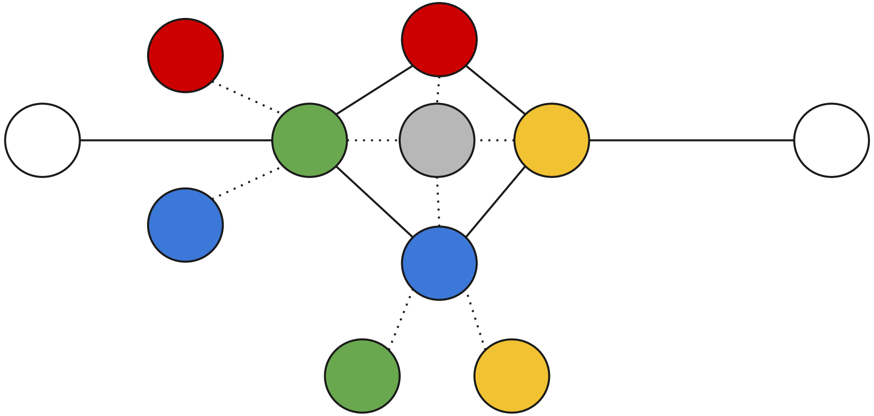
Loop Unwinding



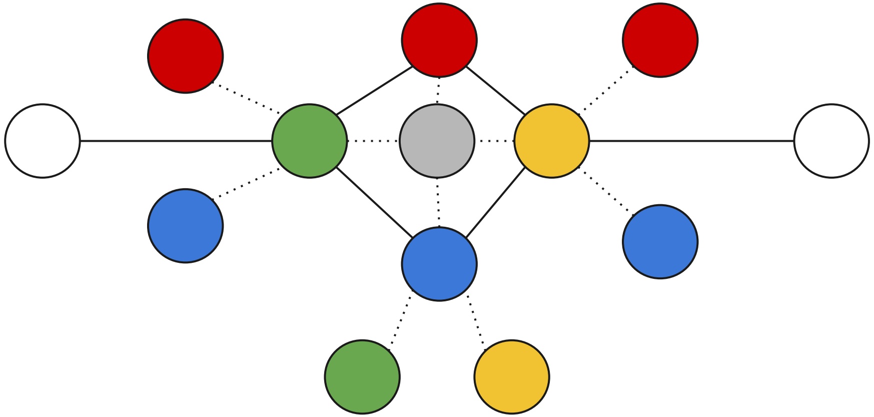
Loop Unwinding



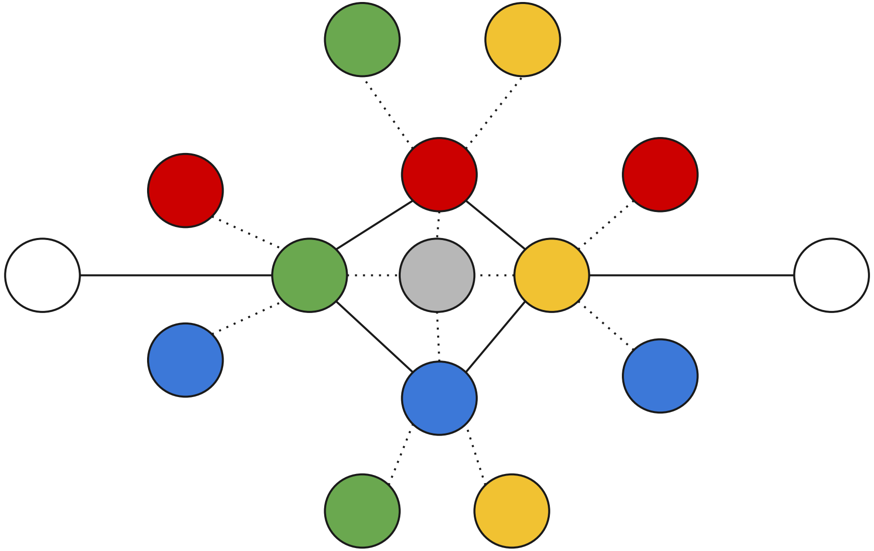
Loop Unwinding



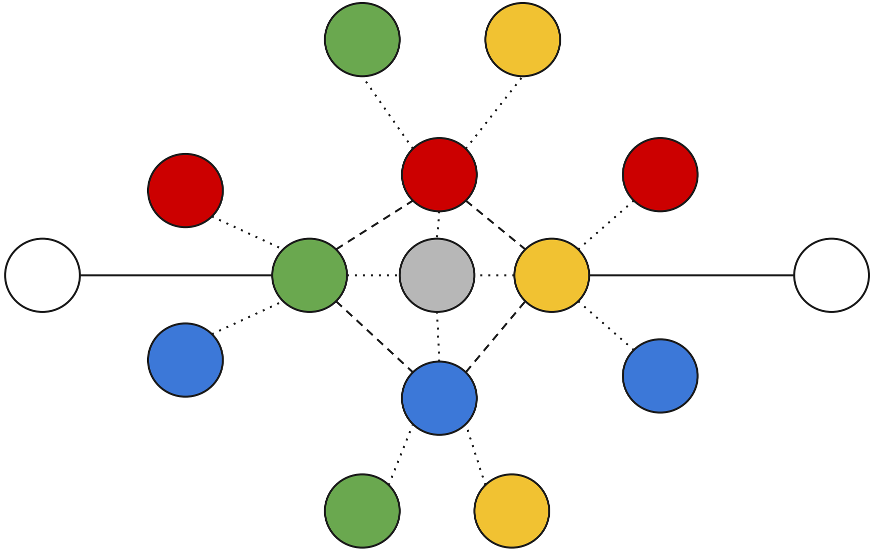
Loop Unwinding



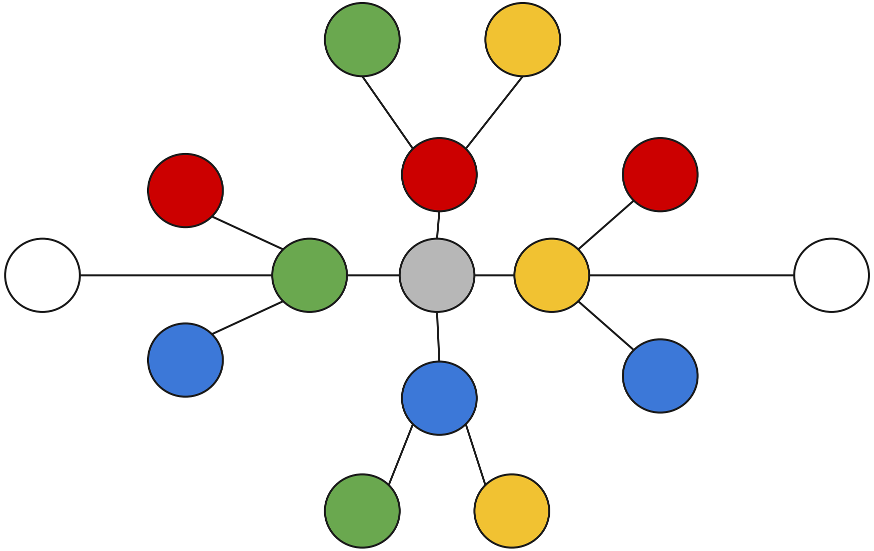
Loop Unwinding



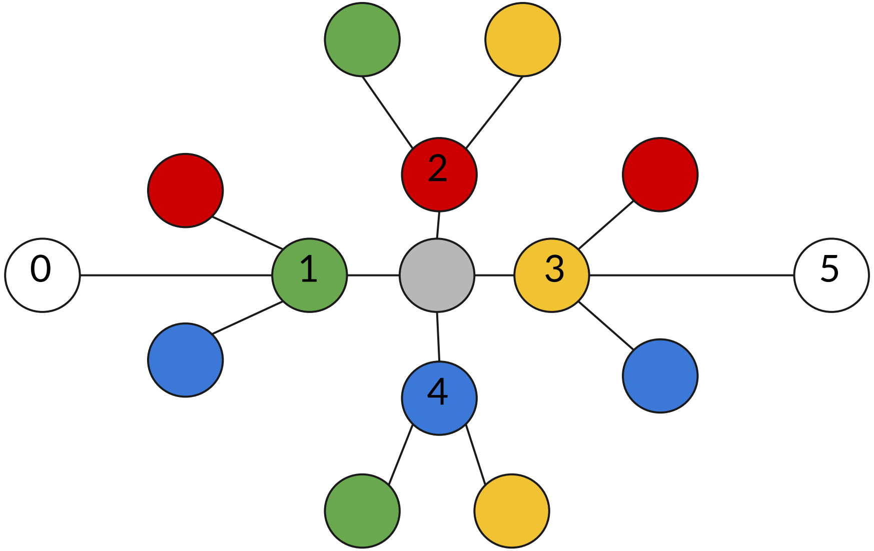
Loop Unwinding



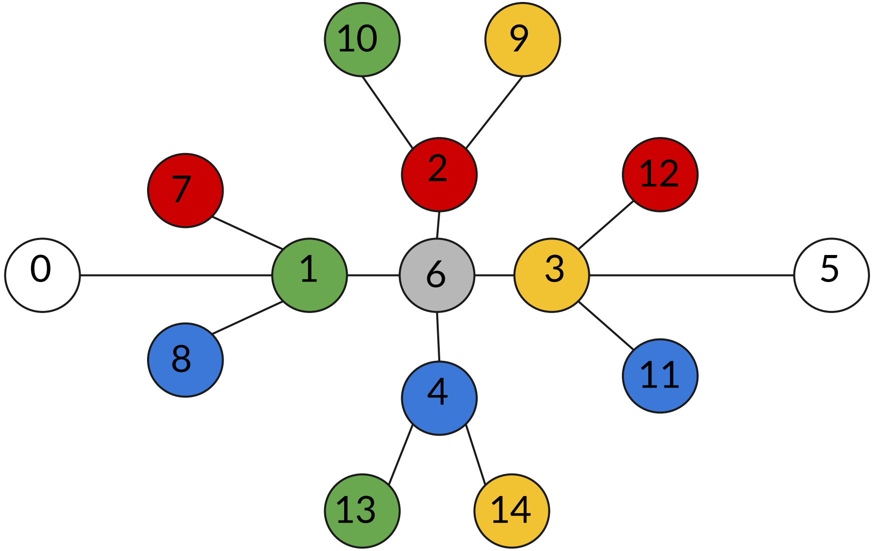
Loop Unwinding



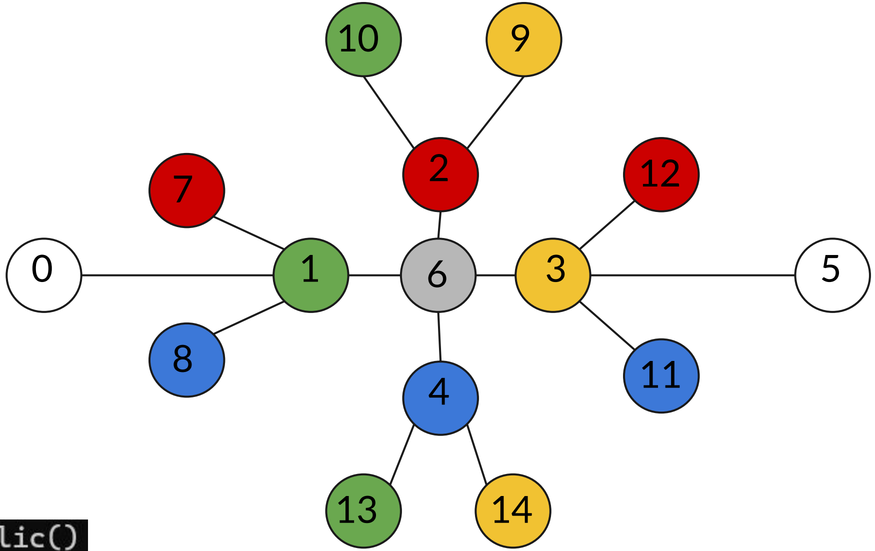
Loop Unwinding

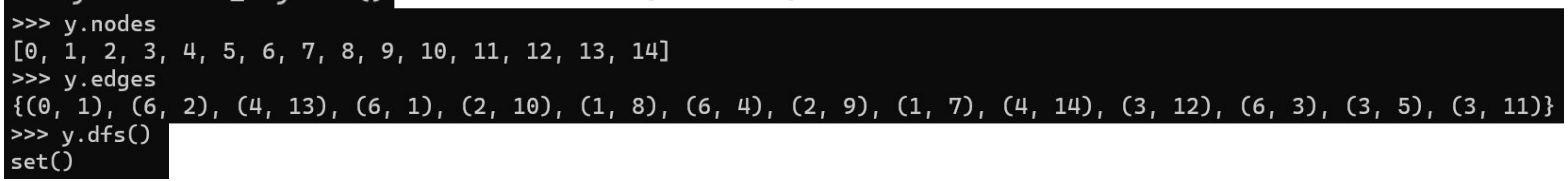


Loop Unwinding



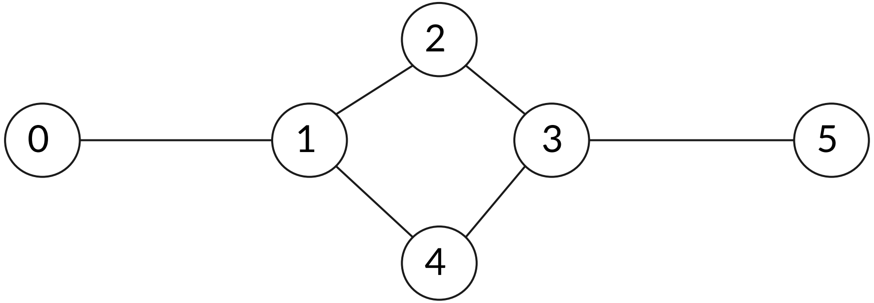
Loop Unwinding

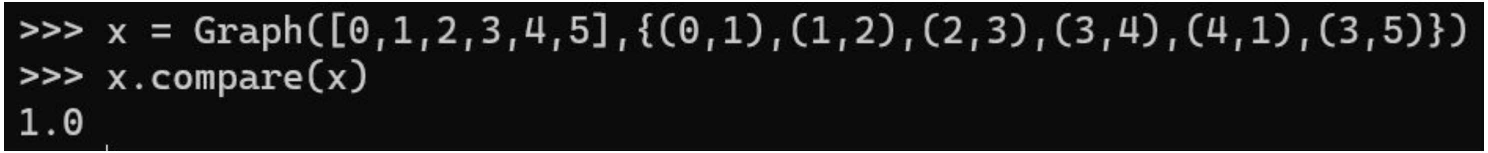




```
>>> y = x.make_acyclic()
>>> y.nodes
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>> y.edges
{(0, 1), (6, 2), (4, 13), (6, 1), (2, 10), (1, 8), (6, 4), (2, 9), (1, 7), (4, 14), (3, 12), (6, 3), (3, 5), (3, 11)}
>>> y.dfs()
set()
```

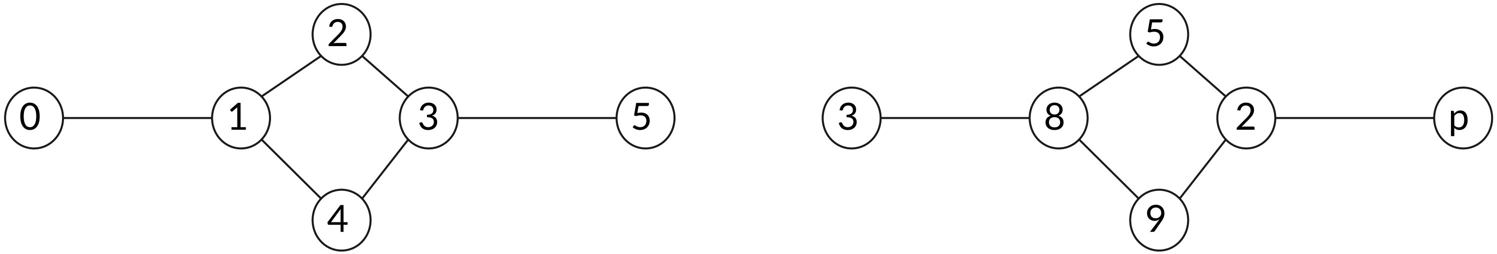
Sample Outputs (Graph vs Itself)

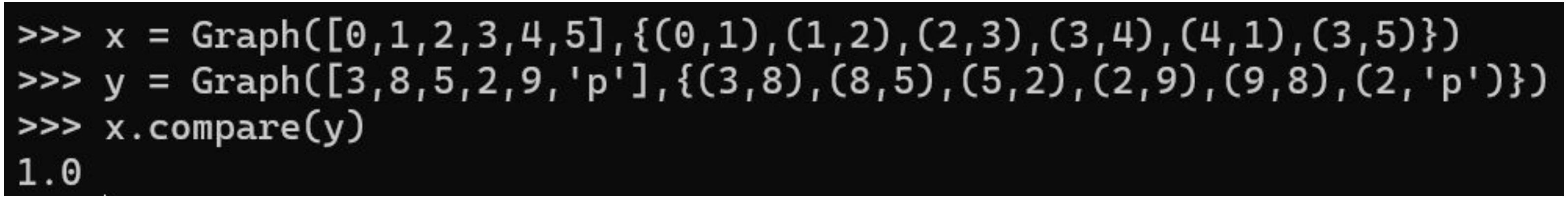




```
>>> x = Graph([0,1,2,3,4,5],{(0,1),(1,2),(2,3),(3,4),(4,1),(3,5)})
>>> x.compare(x)
1.0
```

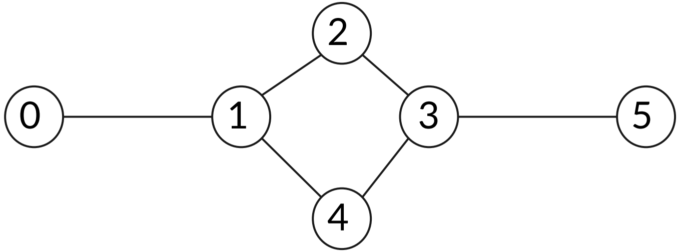
Sample Outputs (Graph vs Rebabelling)

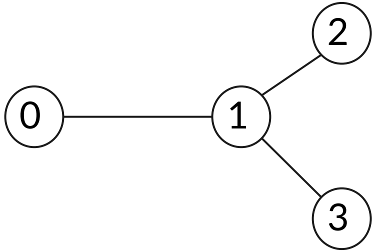


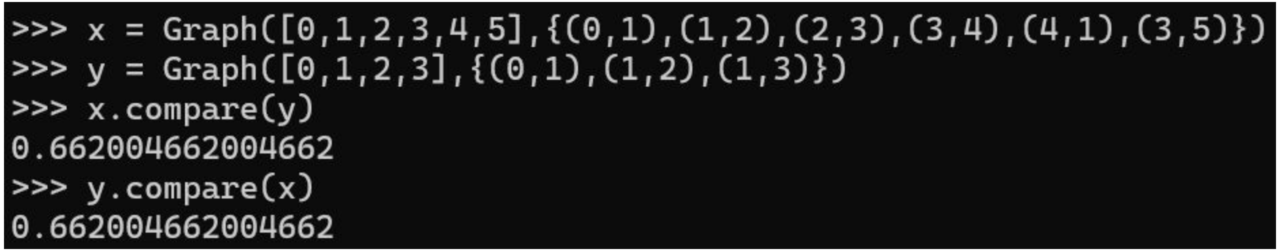


```
>>> x = Graph([0,1,2,3,4,5],{(0,1),(1,2),(2,3),(3,4),(4,1),(3,5)})
>>> y = Graph([3,8,5,2,9,'p'],{(3,8),(8,5),(5,2),(2,9),(9,8),(2,'p')})
>>> x.compare(y)
1.0
```

Sample Outputs (Graph vs Subgraph)

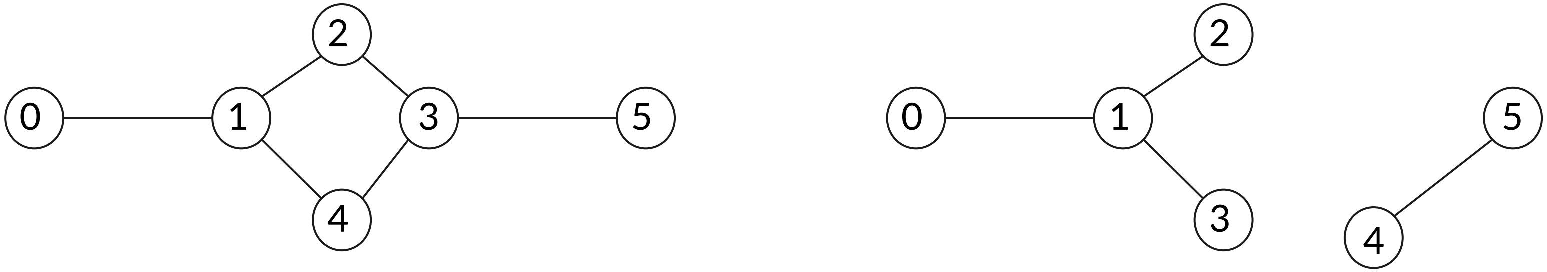


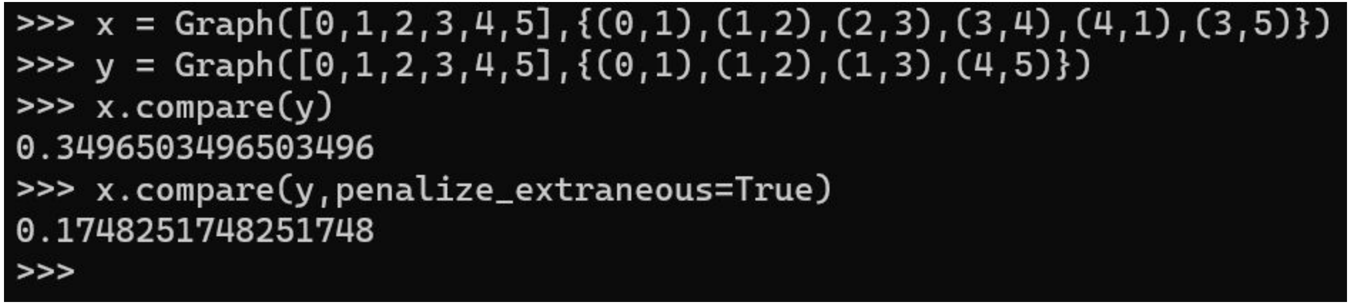




```
>>> x = Graph([0,1,2,3,4,5],{(0,1),(1,2),(2,3),(3,4),(4,1),(3,5)})
>>> y = Graph([0,1,2,3],{(0,1),(1,2),(1,3)})
>>> x.compare(y)
0.662004662004662
>>> y.compare(x)
0.662004662004662
```

Sample Outputs (Connected vs Disconnected)





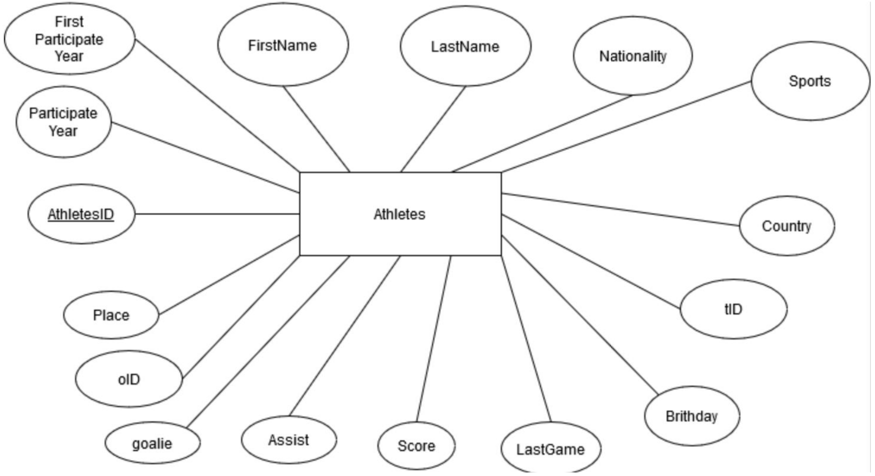
```
>>> x = Graph([0,1,2,3,4,5],{(0,1),(1,2),(2,3),(3,4),(4,1),(3,5)})
>>> y = Graph([0,1,2,3,4,5],{(0,1),(1,2),(1,3),(4,5)})
>>> x.compare(y)
0.3496503496503496
>>> x.compare(y,penalize_extraneous=True)
0.1748251748251748
>>>
```

Computer Vision

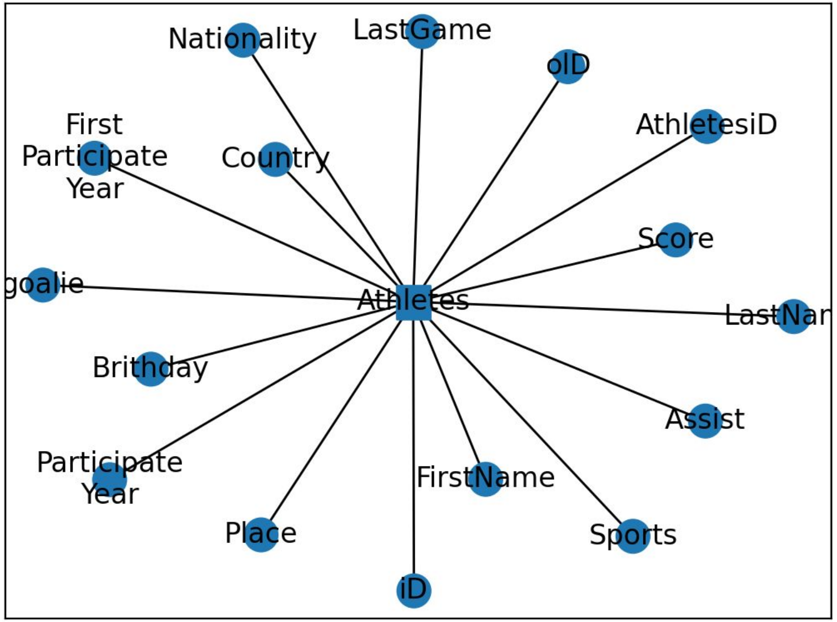
- Based on OpenCV
  - Recognize shapes
  - Convert the Entity Relation Diagram (ERD) to a NetworkX graph with properties (e.g. shape, connected to, connect type, etc.)
  - Support computer-draw ERD with Chen’s notation

Computer Vision

Sample ERD

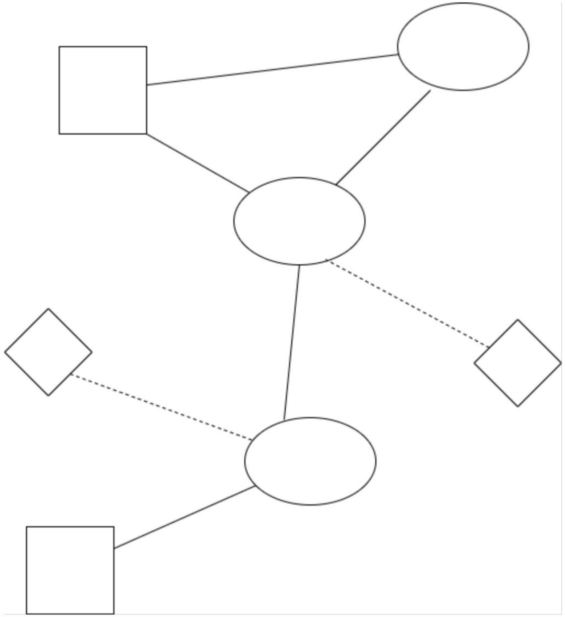


NetworkX Graph

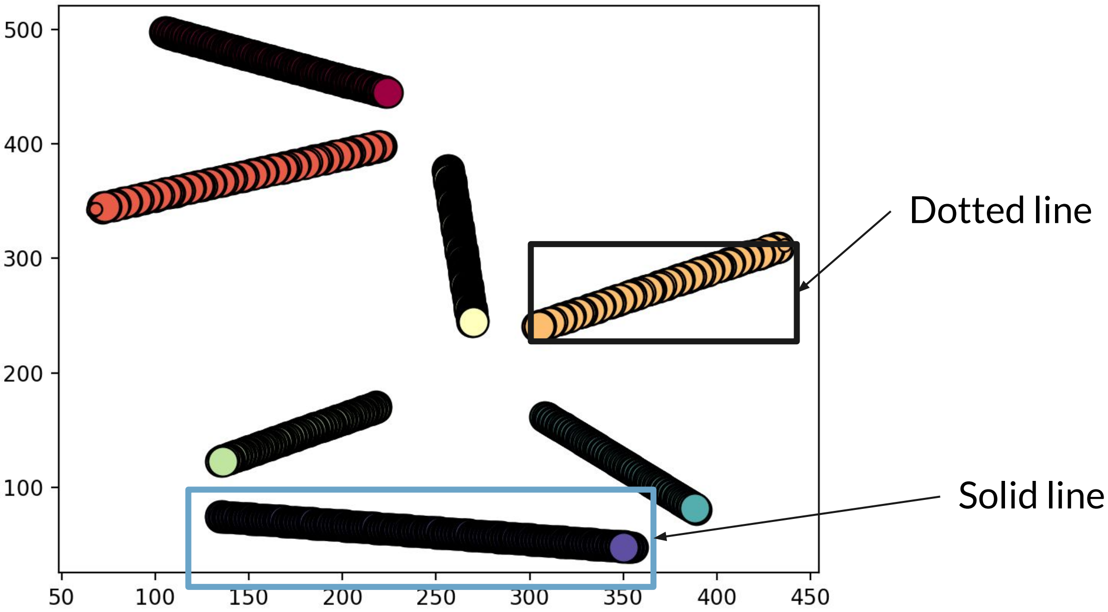


Computer Vision

- Line density check



Computer Vision



Dotted line

Solid line

Computer Vision

- Optical Character Recognition (OCR)

Word Meaning Comparison

- Spacy & levenshtein text distance
  - Get the max of between spacy similarity mark and the levenshtein's text distance

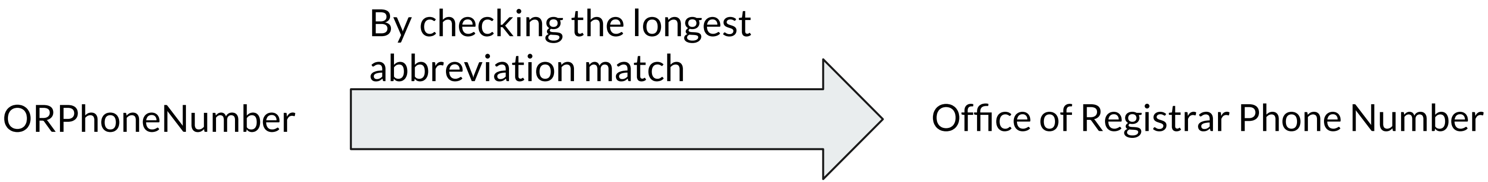
Word Meaning Comparison

- Pyenchant
  - Checking the validity of the words that recognized in OCR
  - Use for the case of sticked words check (e.g. PhoneNum VS. Phone Num)

Word Meaning Comparison

- Abbreviation Check
  - ○ Using the dictionary match and pyenchant

{“OR”: “Office of Registrar”} in the customized dictionary + pyenchant



ORPhoneNumber

By checking the longest abbreviation match

Office of Registrar Phone Number

Research Demo

Goals we Achieved: Research

- Devised and implemented a metric for quantitative graph comparison

Future Work: Research

- Exact (as opposed to approximate) bipartite matching for connected components

- Train model to make ERD marker adhere as much as possible to TA marking

- Use noisy channel for the abbreviation checking & ML on the user’s customized dictionary

Conclusion

Key Learnings

- Engineering architecture
- Agile development
- Interteam and Stakeholder communication