Social Math: A collaborative, visual host for mathematical proof trees

Jianchi Chen
Dept. of Electronical Engineering
jchen2@caltech.edu

Ying-Yu Ho
Dept. of Physics, Mathematics, and Astronomy
yingyu@caltech.edu

Tim Holland
Dept. of Computing and Mathematical Sciences
th0114nd@gmail.com

Kexin Rong
Dept. of Computing and Mathematical Sciences
krong@caltech.edu

Abstract

Mathematical proofs and theorems, and the inter-dependencies among them (edges) have the structure of a directed acyclic graph (DAG). However, in most educational processes or online mathematical libraries, theorems and proofs are typically presented in a linear fashion, which fails to connotate the theorem's inter-relationship with others, or how it is positioned in the mathematical world. Therefore, we propose a web service to better present mathematical proof trees as graphs, and allows users from all over the world to contribute to the graph, and use it for their own interest. Supposedly, the visualization should help users both navigate through the existing mathematical library and also easily contribute new proofs to expand the reach of the proof trees. Specification of proofs could either be very formal, in which case a program could verify them, or more relaxed and reviewed by other users in a collaborative effort to find mathematical truth. People can also benefit by using the visualization tool we provided to construct graphs for other purposes that visualization of logical inter-relationship can help with.

Keywords

proof trees, visualization, social network, education, graph

1. INTRODUCTION

The course of mathematical study can be frustrating. After proving one lemma after another, students can easily get caught in the details and forget about the big picture. In another case, students can know 10 different theorems without understanding how they are related to each other. This is due to the lack of a more intuitive ways to visualize these concepts and their relationships. If we model mathematical theorems and proofs as nodes, and their dependencies as directed edges, a mathematical library is indeed a set of "proof trees" with the structure of a DAG. Acyclic is guaranteed by the fallacy of circular reasoning.

We therefore created a service and web based front end that allows the entry, modification, inspection, and visualization of theorems, definitions, proofs and the dependencies between them. Dependencies include the relationship between a theorem and its proof, between a proof and its lemma, or a theorem/definition and its special case/generalization.

We have two possible choices to proceed on how to keep the DAG logically sound. The first is more of an advanced technical problem: automatic proof-logic verification, requiring highly precise statements of additions to the tree to be parsed, so that the logical statements can then be verified (or rejected) by machine. Alternatively, with a more social emphasis proofs could be stated in natural mathematical language. Under this scheme, the correctness would have to be verified by users, through a combination of moderation, reputation, and democracy.

We hope that this is a service with fruitful pedagogical applications. For example, in a class taught via the Moore Method, the service could be used as a study tool: A set of isolated definitions and theorems are given at the start of the term. Students' progress can be easily assessed from the completeness of the graph visualization.

Alternatively, the service can be incorporated as an useful feature for other web services. The idea of a mathematical proof tree can be generalize to a knowledge graph in any field. For sites like Wikipedia, having a graphic interface navigation to provide a map of topics to learn and modify can greatly enhance user experiences.

The same idea can apply to other fields of study as well.

2. RELATED WORK

TODO: will talk about MOOC

3. TIMELINE

3.1 Initial Timeline

Below is the initial project timeline we proposed at the project proposal presentation. The plan was to build a working prototype before midterm, and to add as many features as time allows afterwards.

- Week 2 - 5: Basics

  - Client-server API
  - Database API
  - Implement views and templates for browsing, editing, and adding theorems
  - Front end visualization

- Week 6: Milestone:
  Should have a simple working prototype, which allows adding, deleting and viewing nodes from the proof DAG
- Week 7 - 9: Useful advanced features
  - Search by keywords
  - User profile
  - Private graph
  - Auto-keyword-extraction
  - Suggested theorems

3.2 Actual Timeline

Below is a summary of our actual weekly progress based on the blogs. The front end progress was highly non-linear because we either underestimated the amount of work or overestimated the ability to stick to schedule of the person in charge. Thus most features were first implemented on the server-side, and then enhanced with Ajax. Nevertheless, we were able to implement the three most useful advanced features: keyword search, user profile and private graph. A detail description of the end product is given in the next section.

1. *Week 1: Changing Gears*

   Due to the discovery of a fairly comprehensive open source traffic simulator project, we decided to abandon the original 144 project proposal and to work, instead, on a web application that accommodates the visualization of mathematical proof tree structures. We discussed the basic application architecture, and decided to use the typical LAMP stack with a REST API and CRUD functionality. In our case the P stands for Python with Django REST framework, the M stands for SQLite, and the A could either be Apache or Amazon Web Service.

   The first steps we took were similar to any team making a CRUD app: setting up the database and a method of communication between the client and server. Specifically, we determined a schema for the database and fleshed out data structures for the DAG. We also started to familiarized ourselves with the frameworks that we will be working with for the rest of the term.

   On the front end side, several JavaScript frameworks quickly came to our mind: AngularJS[**?**] and/or jQuery for DOM manipulation, Bootstrap[**?**] for style and UI components, and D3.js [**?**] for data visualization. These frameworks have some overlapping functions so it was important to use them in a way that avoided conflicts and, preferably, reflected separation of concerns. Since we had very little experience in web development, it took a significant amount of time to study JavaScript and its frameworks before we could make design decisions.

2. *Week 2: Getting Started*

   This week, the first draft of the API was written. It's a fairly standard one that supports the collection of all entries in a paginated manner, getting a single entry for a close up view, and the CRUD operations.

   On the server side, we've decided to go for AWS instead of Apache. As a result, we switched to using virtualenv, which gave us consistent versioning across our local machines, as well as allowing us to use versions required by AWS without having conflicts of the development computers. We also migrated from SQLite to MySQL, since AWS had the best support for the latter. Although setting up MySQL was slightly frustrating.

   On the backend side, we implemented serializers to help form JSON responses, and view methods for creating and reading theorems/proofs.

   On the front end side, we started by finding data visualization examples to understand the power of available tools. One of our particular interest was d3 process map [**?**], which demonstrated a hierarchical layout of graph that we could potentially mimic. Meanwhile, we realized that AngularJS, jQuery, and D3.js all provided DOM manipulation methods. To avoid potential conflicts among the three frameworks, we decided to structure front end applications on MVC (model-viewer-controller) framework of AngularJS, avoid jQuery as much as possible, and use D3.js mainly as a graph layout engine. No JavaScript codes were written yet, but we were able to write some Django view templates to provide text-based interaction with DAG.

3. *Week 3: Backend Milestone*

   After adding templates and view methods for editing and deleting, we had a fully working set of CRUD functionality implemented this week. We also did substantial unit tests on database models and api calls to make sure that our backend infrastructure had been robust so far.

   At this point, the backend side has already reached the original midterm milestone. On the front end side, we succeeded in driving AngularJS templates with D3 graph layout engine. However, the hierarchical force layout algorithm in d3 process map wasn't as useful as we had expected. While D3.js provided nice force layout algorithm, it required users to manually specify the hierarchy relationships in the dataset. Therefore, we started to design our own algorithm. Another obstacle was that we formatted graphs as adjacency lists while D3 Force Layout used nodes and edges as native objects. Therefore, proper conversion was needed in order to make use of the D3 framework.

4. *Week 4: Frontend Milestone*

   After reaching the milestone, the backend side has moved on to advanced features, starting with the keyword related ones. We first implemented functionalities for keyword creation, edit and search . Users can now tag theorems and proofs with appropriate keywords. A search bar was added on the index page to allow search by keywords.

   As we moved one person from the backend to the front end, the front end side was able to make substantial progress this week. We wrote a stylesheet for the website to make it more user friendly. We also set up MathJax [**?**], a free online LaTeX compiling tool, to support adding and displaying LaTeX code on the website. Finally, we built a client-side knowledge graph editor that made it possible to implement and test a working graph layout algorithm. This graph editor also served as an exercise on Bootstrap UI components and a reference of our final user interface. The next step was to turn the graph editor into an Ajax application that was able to communicate with the server.

5. *Week 5: Prototype online*

   This week, we have been focusing on putting a simple working version of our product online. On one hand, we tried to connect the knowledge graph visualization with backend database. On the other hand, we worked on setting up Amazon EC2 (Amazon Elastic Compute Cloud) to host the website. We were able to seed the database with number theory theorems and deploy the latest backend code. Knowledge graph visualization was not fully integrated yet, due to bugs in API for Ajax calls.

   Meanwhile, we have also been extending the website's functionalities. We started to implement user authentication system, in the hope of enabling users to set up private knowledge graphs for classroom settings in the future.

6. *Week 6: Makeover*

   We did a makeover to the website by refactoring the HTML files using UI Bootstrap. We also redesigned the index page so that it could nicely combine the graph visualization with the search functionality. In addition, we finished implementing user authentication systems and user profile. This was our first step of incorporating the "social" component in the website.

7. *Week 7: Merging*

   This week we successfully merged the backend Django framework with the front end graph editor for knowledge graph visualization. We made a single-page application that can load latest submissions and perform searches via Ajax calls. We were also able to display all nodes and their dependencies in database with the graph layout algorithm. Future work was needed to make the visualization more interactive.

   We extended user profiles to include recent activities, so users can keep track of their latest contributions. We started to keep track of users who published and last modified each theorem/proof. We also added a "following" feature for theorems to make it easier for users to keep up with their interests. We moved the website to a new domain: socmat.com/prooftree.

8. *Week 8: MVP*

   This week we further polished the product. For the front end, we merged and deployed user profile related features. We were able to add a few interactive features to the knowledge graph:

   (a) Zoom and Panning: Since it is hard to fit the entire knowledge graph on the index page, we decided, instead, to giver users full control of which part of the graph they would like to see.

   (b) Recenter: In addition to the global knowledge graph, we also allow users to view a local graph centered around any chosen node. This feature comes in handy when a user wants to explore a specific theorem/proof. By default, only the chosen node and its immediate neighbors are displayed in the local graph. However, users can adjust the number of higher-order neighbors shown from the interface.

   (c) Add child: This feature allows one to add a new theorem/proof directly from the knowledge graph. This serves as a more intuitive way for users to make contributions.

   (d) Preview: This feature allows users to view the content of a node without redirecting to the detail page.

   Finally, we finished implementing private graphs, which was mainly designed for a classroom setting use case. Users can choose to create their own knowledge graphs, which will not be displayed on the public global graph. CRUD functionality and keyword search were also implemented in the private graph.

4. END PRODUCT

The end product would hopefully be a useful web application that would allow publishing of proofs in a graphical format subject to immediate review of other users, and a repository for a large collection of proofs to be seen in a graphical format upon request.

4.1 Graph Visualization

4.1.1 Layout

We build the graph layout engine based on D3 Force Layout, which models nodes as repulsing charged particles and edges as elastic cables holding nodes together. The layout algorithm then runs a physics simulation to determine the position of each node. For each node, we draw a text box

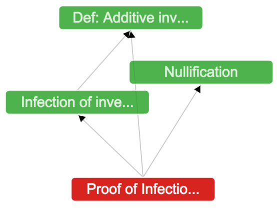


Figure 1: Parent nodes positioned above child nodes (Green: theorem. Red: Proof.)

with its title inside, and an arrow from Node c to Node p to indicate that c depends on p. Different types of dependencies will be discussed in the next subsection.

So far the layout engine only takes care of undirected graph structure. Ideally, for two nodes c, p with c depending on p, we want Node p to stay a little above Node c, so that the root of a tree can always be pushed to the top. In order to do so, we invented our own algorithm. If our graph is a tree, we can unambiguously compute the height of each node and assign a y-coordinate accordingly. To deal with cycles, we can minimize the potential

$$U(\mathbf{y}) = \sum_{<c,p>} (y_p - y_c - H)^2 , \qquad (1)$$

where the constant H is vertical spacing per level. Note that $U(\mathbf{y}) \geq 0$ and that $U(\mathbf{y}) = 0$ has solutions if the graph is a tree. The solution is not unique because $U(\mathbf{y})$ is invariant under translation of all nodes by the same displacement, but this issue turns out to be unimportant.

For example, if we have nodes $1, 2, 3, 4$ with edges $(2 \to 1), (3 \to 1), (3 \to 2), (3 \to 4)$, the solution (subject to translation) to equation (1) is

$$y_1 = \frac{4}{3}, \ y_2 = \frac{2}{3}, \ y_3 = 0, \ y_4 = 1.$$

We want to use this information to adjust equilibrium positions of nodes in D3 Force Layout, so we promote minimization of $U(\mathbf{y})$ to a dynamical process, and a natural choice is gradient descent. Thus we differentiate $U(\mathbf{y})$ with respect to y_i (and multiply by a constant $-\eta$) to obtain equations of motion

$$\frac{\mathrm{d}y_i}{\mathrm{d}t} = \eta \left[\sum_{<i,p>} (y_p - y_i - H) - \sum_{<c,i>} (y_i - y_c - H) \right], \qquad (2)$$

which pulls nodes away from their D3 equilibrium positions and minimize the sum of D3 potential and $U(\mathbf{y})$.

4.1.2 *Manipulation*

- Scroll horizontally / vertically
- Zoom in / zoom out / reset
- Recenter: change the center node around which the graph is explored

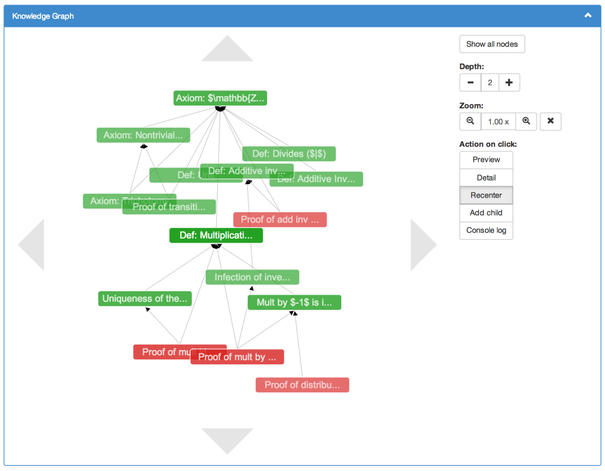


Figure 2: Knowledge graph UI Overview

- Depth: the maximal distance from the center node to display. For example, 0 only displays the center node, 1 also displays its parents and children, and 2 also displays its grandparents and grandchildren.
- Show all nodes: display the whole graph or only nodes around a center node.
- Preview: open a small window to show the node's content.

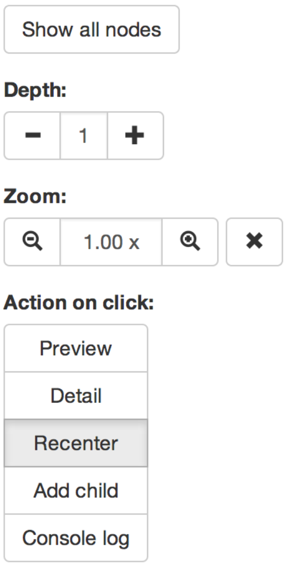


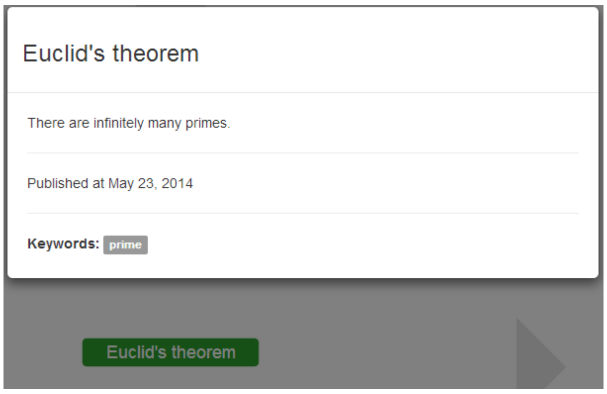
Figure 3: Knowledge graph UI Toolbar

4.2 Web Framework

The server middleware is based on the Django web-framework. It provides a full set of functionalities for socialmath users to conveniently contribute to the open mathematics library and to build their own knowledge graphs:

- *Nodes and Dependencies*

The fundamental functionality in the socialmath web framework is the node-dependency system. In the framework, nodes contain information about its theme, content, type, publication and modification time. We store node-node relationship as well as node-to-other-object relationships.


Euclid's theorem

There are infinitely many primes.

Published at May 23, 2014

Keywords: prime

Figure 4: A node in socialmath's knowledge graph

Dependencies are classified into three types:

1. *Proof-to-Theorem*: Lemma dependency. This relationship exists when a proof of one theorem cites another theorem in the process.
2. *Theorem-to-Proof*: Prove dependency. This relationship exists when a proof proves a theorem.

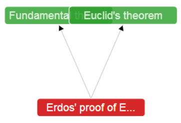


Figure 5: Erdõs' proof of Euclid's theorem exhibits prove relationship to Euclid's theorem, and lemma relationship to Fundamental Theorem of Arithmetic

3. *Theorem-to-Theorem*: High-Relevancy dependency. This relationship exists when a theorem is the special case or a generalization of another, or they have strong logical relationship with each other. This also includes the relationship between theorems and definitions. We also model definitions as theorems in our system, except that definitions can't have proofs.

- *Keywords and Search*

  Keywords are words that imply a node's content and branch of study. They can be added when adding the node, or when editing an existing node. Node-keyword relationships are stored in a relationship set called "Tags". The search functionality is supported by keyword mapping. Search can be used in two ways: you can either enter text in the search bar displayed at the index page, or click on the link of an existing

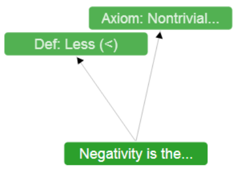


Figure 6: Theorem-theorem and theorem-definition relationship

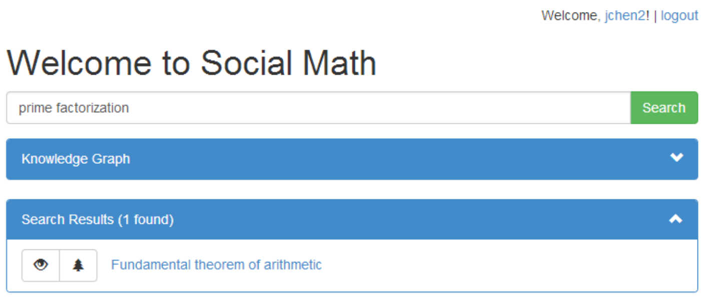


Figure 7: Search results for the keyword "prime factorization"

  keyword.

- *Submission and Editing*

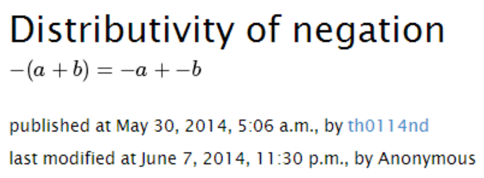
  Adding a new theorem or proof can be done in two ways: Click on the button at the bottom of the index page, or use the "add child" tool on the graph.

  Users can edit an existing theorem/proof by clicking the "edit" button on the node's detail page. The button links to an interface similar to the add-node page, except that the existing information will be pre populated.

- *Users and Profile*

  Socialmath doesn't require a user to login to view its content, but we provide additional functionalities for logged in users. Once a user is logged in, his recent activities (adding/editing nodes) will be recorded as events in the database. The events will show up both on the detail page of the modified node, and on the user's profile page.

  Follow is another feature provided for registered users to help keep track of topics they are interested in. The user may click on the "follow" button of any given node, and the server will capture it as a following event. Clicking on a username redirects to that user's profile page. Since there is no search-for-user functionality yet, the only entrance for viewing other users' profiles is through the node's change history.



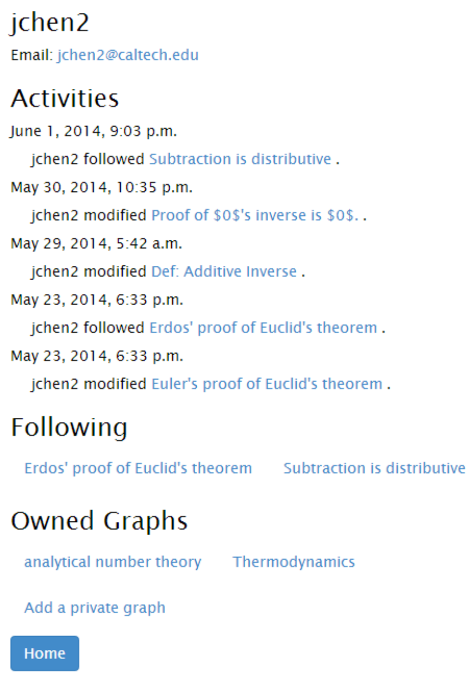
Distributivity of negation

$-(a+b) = -a + -b$

published at May 30, 2014, 5:06 a.m., by th0114nd

last modified at June 7, 2014, 11:30 p.m., by Anonymous

Figure 8: Change history of a node shows the initial author and the last modifier.



jchen2

Email: jchen2@caltech.edu

Activities

June 1, 2014, 9:03 p.m.

jchen2 followed Subtraction is distributive .

May 30, 2014, 10:35 p.m.

jchen2 modified Proof of \$0\$'s inverse is \$0\$. .

May 29, 2014, 5:42 a.m.

jchen2 modified Def: Additive Inverse .

May 23, 2014, 6:33 p.m.

jchen2 followed Erdos' proof of Euclid's theorem .

May 23, 2014, 6:33 p.m.

jchen2 modified Euler's proof of Euclid's theorem .

Following

Erdos' proof of Euclid's theorem   Subtraction is distributive

Owned Graphs

analytical number theory   Thermodynamics

Add a private graph

Figure 9: The profile page of Jianchi Chen, consist of basic info, activities, following-nodes, and private graphs he owns

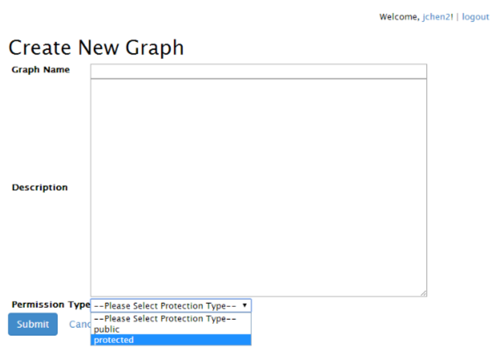
- *Private Graphs*

  In addition to one public knowledge graph visible to everyone, socialmath also provides *private graph* feature so that each registered user can create and edit his own graph. Each private graph has an individual index page. Permission relationship-set makes sure that no nodes in the private graph can be accessed from the public one.
  Upon creation, the user may choose to make the graph public or only visible to authorized users. An attempt to view private graph contents when you're not invited will result in a 401 unauthorized error.

4.3 Database

4.3.1 Entity-set Schema

- Node(<u>node_id</u>, kind, title, statement, pub_time, last_modified)
- Keyword(<u>kw_id</u>, word)
- PrivateGraph(<u>pgraph_id</u>, name, description, pub_time, perm_type)



Welcome, jchen2! | logout

Create New Graph

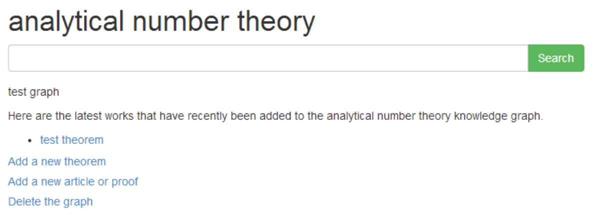
Graph Name

Description

Permission Type --Please Select Protection Type--

--Please Select Protection Type--
public
protected

Figure 10: Creating a private graph



analytical number theory

test graph

Here are the latest works that have recently been added to the analytical number theory knowledge graph.

- test theorem

Add a new theorem

Add a new article or proof

Delete the graph

Figure 11: The index page of a private graph, with a node called "test theorem" added.

- We used the default Django model for user authentication: User(<u>user_id</u>, username, email, password, ...)

4.3.2 Relationship-set Schema

- Depends(<u>node_id</u>, <u>node_id</u>, dep_type)
- Tags(<u>node_id</u>, <u>kw_id</u>)
- Modifies(<u>node_id</u>, <u>user_id</u>, pub_time, event_type)
- Owns(<u>user_id</u>, <u>pgraph_id</u>)
- Includes(<u>pgraph_id</u>, <u>node_id</u>)

5. FUTURE WORK

Here are a few additional Although the current socialmath is ready for public usage, there are several additional goals that we plan to achieve with socialmath in the future in order to make it a better product:

1. *Protection against abuses*
   Right now, socialmath has no protection against malicious activities such as deleting everything from the knowledge graph, creating dummy nodes, or adding unrelated contents. We can model our protection mechanism after Wikipedia, and build a more comprehensive change history, which allows the system to revert malicious changes and penalize misbehaved users.

2. *Correctness system*
   Another concern is that the website currently has no way of assessing the correctness of a proof or a theorem. We can either implement an user upvote system similar to Stack Overflow's, or a user "questioning" system where users can request verification on statements.

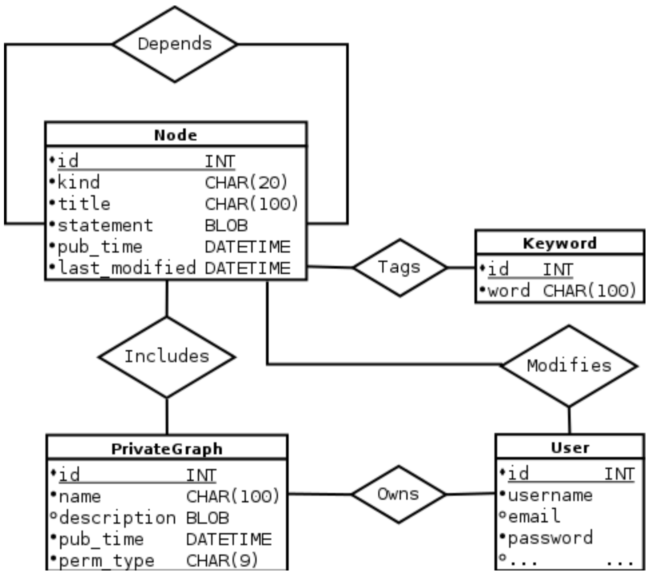


Figure 12: ER Diagram

3. *A more collaborative private graph system*
   An ideal model for a private graph is the github repository. Users are able to collaborate on a graph, fork someone else's work, assign and discuss issues, and use version control. We would also like to include visualization in the private graph.

4. *File support*
   In addition to in-box writing, we should support pdf upload and display, so that users can also make contributions by finding existing write ups.

5. *Lemma and keyword detection*
   A more advanced version of the socialmath node submission system includes lemma and keyword auto-detection. A simple algorithm is to collect statistics on the text's N-grams (sequences of n consecutive words). Sequences with a significant number of occurrences are then candidates for the keyword/lemma [**?**].

6. *Better user interaction*
   There's a lot of room for enhancing the "social" side of the current product. The first step is to implement handy features such as user searching, following, and in-site messaging. We can implement a level-based experience system to reward active and reliable users. We should also reward users who manage to invite new users.

6. ACKNOWLEDGMENTS

We would like to thank our faculty advisor, Professor Steven Low, and our teaching assistant, Qiuyu Peng, for their feedback and guidance.We would also like to thank Professor Adam Wierman for co-hosting the class.

7. REFERENCES