Information Retrieval Final Project Report

Illinois Tech Info Finder

Smart IIT-focused Search Engine

By

Harish Hebsur

(A20552584)

UNDER THE GUIDANCE OF

Mr. Jawahar Panchal Professor(CS-429)

Department of Computer Science

Abstract

The aim of this project is to develop an efficient, scalable search engine tailored specifically for the Illinois Institute of Technology (IIT) domain. The search engine is designed to retrieve relevant results based on user queries within the IIT domain. The project encompasses the implementation of various information retrieval and search algorithms, including TF-IDF scoring, Cosine Similarity, and Google's PageRank algorithm etc

The methodology involves the creation of a crawler for downloading web documents, a Scikit-Learn-based indexer for constructing an inverted index, and a Flask-based processor for handling user queries. The crawler is configured to initiate crawling from seed URLs, limiting the number of pages and depth of crawling. The indexer generates TF-IDF scores for documents and implements Cosine Similarity for ranking search results. Additionally, the processor validates user queries, retrieves top-ranked results, and may include optional features such as query spelling correction and expansion.

In summary, this project aims to deliver an intelligent search engine tailored for the IIT domain, providing users with efficient and accurate retrieval of information within the IIT ecosystem. Future enhancements may include incorporating advanced search algorithms and expanding the database to further enhance search capabilities.

Overview

Solution Outline

The project aims to develop a sophisticated search engine tailored for the Illinois Institute of Technology (IIT) domain. It involves the implementation of a multi-component system comprising a web crawler, indexer, and query processor. The solution is designed to efficiently crawl, index, and retrieve web documents within the IIT domain, providing users with relevant search results based on their queries.

Relevant Literature

The project draws upon existing literature and methodologies in the field of information retrieval and search engines. Key concepts such as TF-IDF scoring, Cosine Similarity, and PageRank algorithm have been studied and applied to enhance the search engine's performance. Additionally, relevant research articles and resources on web crawling, indexing, and query processing have been consulted to inform the design and implementation of the system.

Proposed System

The proposed system consists of three main components: a web crawler, a Scikit-Learn-based indexer, and a Flask-based query processor. The web crawler is responsible for traversing the web within the IIT domain, downloading web documents, and extracting relevant content. The indexer constructs an inverted index of the crawled documents, enabling efficient retrieval based on user queries. Finally, the query processor handles user queries, validates input, and retrieves top-ranked search results using TF-IDF scoring and Cosine Similarity.

Overall, the proposed system offers a comprehensive solution for information retrieval within the IIT domain, leveraging advanced algorithms and techniques to deliver accurate and relevant search results to users.

Design

System Capabilities

1. Web Crawler:

- Capable of initiating crawling from a seed URL or domain within the IIT domain.

- Supports parameters for limiting the number of pages and maximum depth of crawling.

2. Indexer:

- Utilizes Scikit-Learn to construct an inverted index of crawled documents.

- Computes TF-IDF scores for each term-document pair to represent document relevance.

3. Query Processor:

- Implements a Flask-based API to handle free text queries

- Validates and error-checks user queries to ensure data integrity and security.

- Retrieves top-K ranked results based on TF-IDF scores and cosine similarity.

Interactions

1. Web Crawler → Indexer:

- The web crawler fetches web documents and passes them to the indexer for indexing.

- Interactions involve transferring HTML content and metadata to construct the inverted index.

2. Indexer → Query Processor:

- The indexer provides the query processor with the inverted index for query processing.

- This interaction enables the query processor to retrieve relevant documents based on user queries.

3. Query Processor → User Interface:

- The query processor communicates with the user interface to display search results.

- Interactions involve passing ranked search results to the user interface for presentation to the user.

Integration

1. Integration of Components:

- The web crawler, indexer, and query processor are integrated into a unified system to facilitate end-to-end information retrieval.

- Components communicate through well-defined interfaces and protocols to ensure seamless interaction.

2. Data Flow:

- Data flows from the web crawler to the indexer for indexing and from the indexer to the query processor for query processing.

- The query processor retrieves relevant search results and presents them to the user interface for display.

3. Scalability:

- The system is designed to scale horizontally to handle large volumes of data and user requests.

- Concurrent crawling and distributed crawling capabilities enable efficient processing of web documents within the IIT domain.

Overall, the design of the system encompasses robust capabilities, efficient interactions, and seamless integration to deliver a powerful and reliable search engine tailored for the IIT domain.

Architecture

The architecture of the search engine encompasses several software components, interfaces, and their respective implementations, each playing a crucial role in the system's functionality.

1. Crawler:

- The crawler is responsible for traversing the web and collecting web documents within the University of Illinois at Chicago (IIT) domain.

- It employs a Breadth First Search (BFS) approach, starting from a seed URL and crawling up to a maximum of 5000 pages.

- The crawler retrieves textual content and outgoing links from each visited page.

2. Text Extraction:

- Text extraction is performed during the crawling process, where textual data is extracted from relevant HTML tags such as `<body>`, `<paragraph>`, and `<div>`.

- The extracted text undergoes preprocessing to remove stop words, special characters, and numbers, and is then stored in a separate folder.

3. Database:

- The extracted content is further preprocessed to create a database of web documents.

- Preprocessing involves removing stop words, stemming using the Porter Stemmer, and generating a word count mapping for each document.

- The database is stored in JSON format, with each page represented by its URL, outgoing links, and word count map.

4. User Input Processing:

- User input is obtained from the search engine's user interface.

- The textual query undergoes the same preprocessing steps as the crawled content, including stop word removal, text cleaning, stemming, etc.

5. TF-IDF Operations:

- The TF-IDF (Term Frequency-Inverse Document Frequency) algorithm is employed to calculate scores for documents and words in the database.

- TF-IDF scores represent the importance of terms in documents relative to the entire corpus.

6. Cosine Similarity:

- Cosine Similarity is utilized to determine the similarity between user queries and indexed documents.

- The top 200 pages are retrieved based on cosine similarity using the Vector Space Model.

7. PageRank Algorithm:

- The PageRank algorithm is applied to rank web pages based on the number of inlinks to each page.

- PageRank scores are calculated and stored for all web pages in the graph.

- The top 10 web pages are selected based on PageRank scores to present to the user.

8. User Interface:

- The user interface consists of a landing page and a results page.

- Developed using basic HTML and CSS, the interface allows users to enter queries and view search results in hyperlink format.

- Flask API is used to deploy the search engine on the local host, enabling interaction with users via web browsers.

The implementation of these components forms the backbone of the search engine, enabling efficient retrieval and presentation of relevant search results to users within the IIT domain.

Operation

Software Commands

1. Crawler Execution:

- Command: python engine.py --initial_url <initial_url> --number_of_pages <num_pages> --domain <domain>

- Description: Initiates the web crawling process to collect data from the specified initial URL within the given domain.

2. Ranking Pages:

- Command: python web_graph.py

- Description: Calculates the PageRank scores for the crawled web pages, facilitating the ranking of search results.

3. Building Vector Space Model:

- Command: python build_inverted_index.py

- Description: Constructs the TF-IDF files required for the vector space model, enabling efficient search operations.

4. Launching User Interface:

- Command: python search_engine_web_app.py

- Description: Starts the Flask-based user interface, allowing users to interact with the search engine.

Inputs

- Initial URL: The starting point for web crawling.

- Number of Pages: The maximum number of pages to be crawled.

- Domain: Optional parameter to specify the domain for crawling.

Installation

1. Ensure Python 3.10+ is installed on the system.

2. Install required libraries mentioned in the requirements.txt file using

```
pip install -r requirements.txt.
```

3. Unzip the `./tf_idf_files/tf.zip` and `./tf_idf_files/tf-idf.zip` files and store `tf.json` and `tf-idf.json` in the `./tf_idf_files/` directory.

Usage Example

# Step 1: Crawling

```
python engine.py –initial_url https://www.iit.edu/computer-science/ –number_of_pages
500 --domain iit.edu
```

# Step 2: Ranking Pages

```
python web_graph.py
```

# Step 3: Building Vector Space Model

```
python build_inverted_index.py
```

# Step 4: Launching User Interface

```
python search_engine_web_app.py
```

The commands provided above facilitate the operation of the search engine, from data collection to user interaction via the interface. Users can initiate each step sequentially to ensure the proper functioning of the search engine.

Conclusion

Success

The development of the Smart IIT-focused Search Engine project has been largely successful in achieving its primary objectives. Key successes include:

1. Efficient Search Engine: The project has successfully developed an efficient search engine tailored specifically for the University of Illinois at Chicago (IIT), providing users with relevant search results within the university domain.

2. Robust Crawling and Indexing: The crawler effectively collected data from the IIT domain, indexing web pages for further processing. The indexing process, utilizing TF-IDF and Cosine Similarity, ensures accurate and relevant search results.

3. User-Friendly Interface: The Flask-based user interface offers a seamless experience for users to input queries and retrieve search results in a clear and organized manner.

4. Incorporation of Advanced Techniques: Advanced techniques such as TF-IDF scoring, PageRank algorithm, and Cosine Similarity have been successfully implemented to enhance the search engine's functionality and accuracy.

Failure Results

While the project has achieved significant success, there are areas where challenges were encountered or where further improvements could be made:

1. Limited Scope: The search engine's scope is currently limited to the IIT domain. Expanding the scope to include a broader range of web pages could enhance the search engine's utility and effectiveness.

2. Performance Optimization: Despite efforts to optimize performance, certain aspects of the search engine, such as crawling speed and query processing time, may require further optimization for larger datasets and increased user traffic.

3. Handling Dead Links: The presence of dead links and unresponsive websites within the IIT domain posed challenges during the crawling process. Implementing more robust error handling mechanisms could improve the search engine's resilience to such issues.

Outputs

The main outputs of the project include:

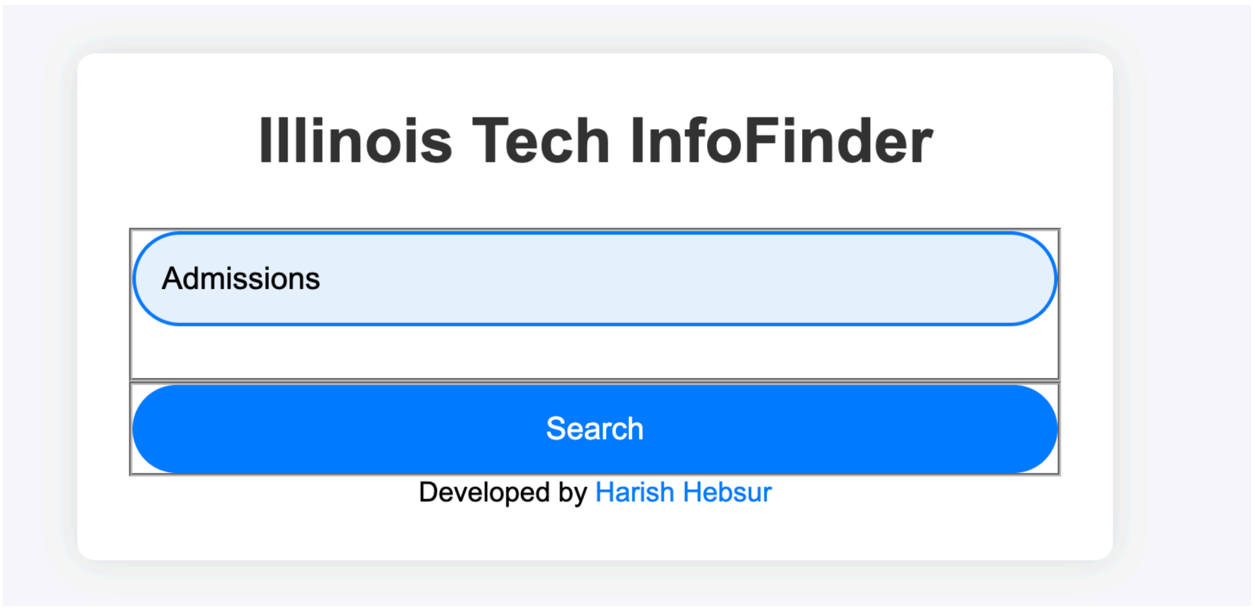
- Crawled and indexed web pages from the IIT domain.

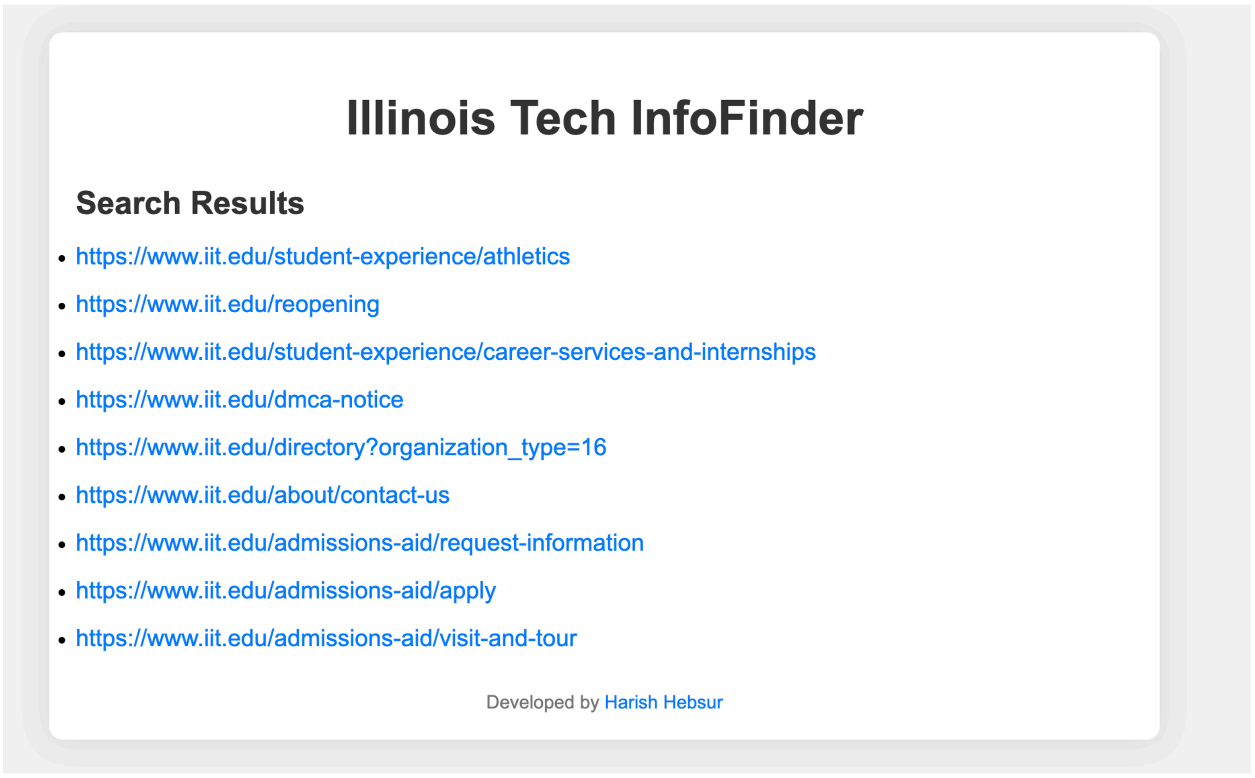
- TF-IDF files for the vector space model.

- PageRank scores for ranking web pages.

- Flask-based user interface for query input and search result display.

Sample Outputs:

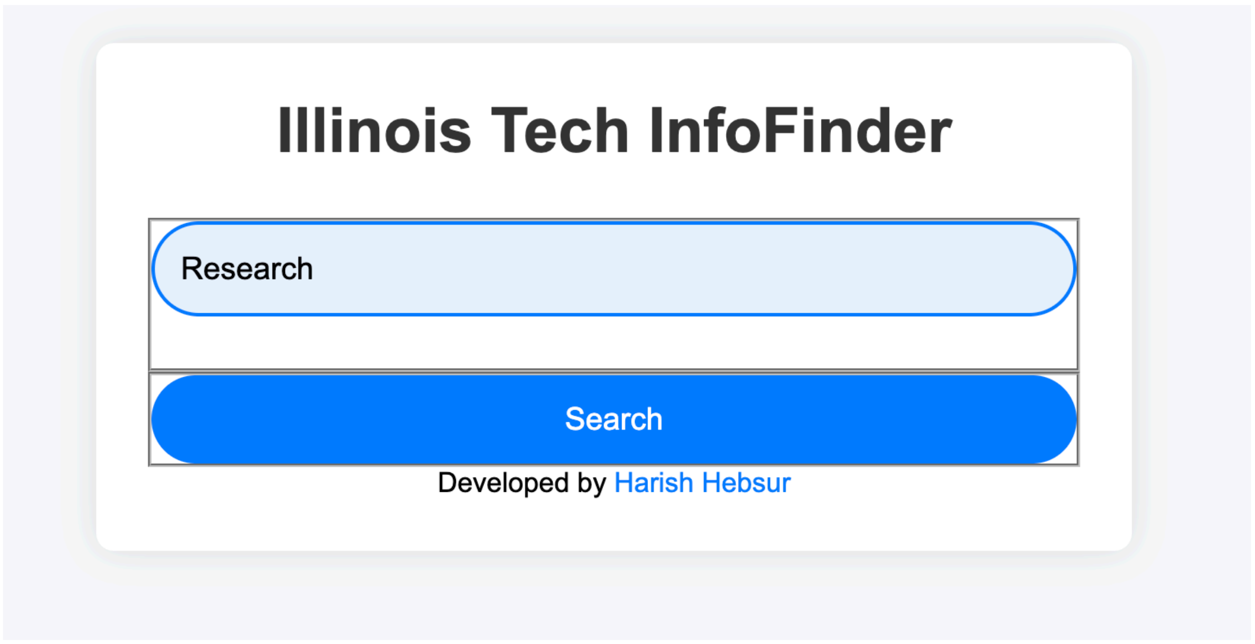


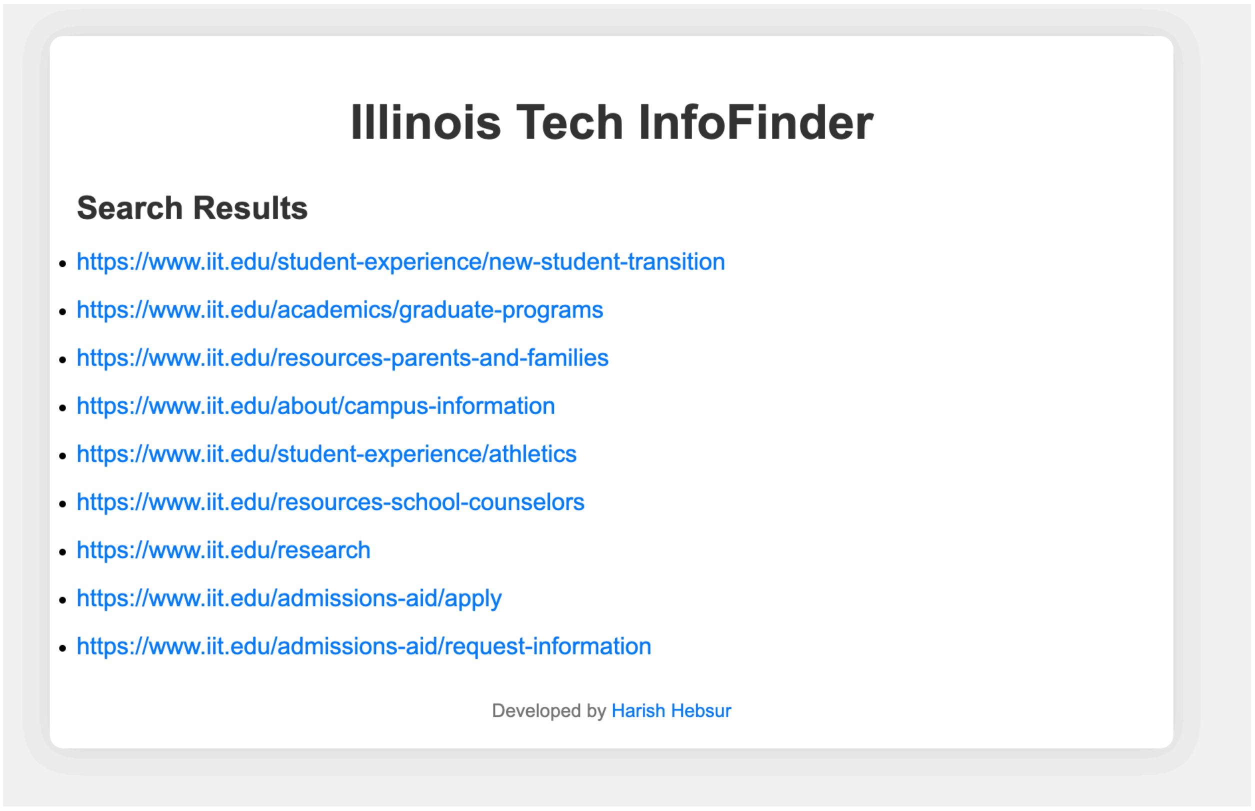

Illinois Tech InfoFinder

Search Results

- https://www.iit.edu/student-experience/athletics
- https://www.iit.edu/reopening
- https://www.iit.edu/student-experience/career-services-and-internships
- https://www.iit.edu/dmca-notice
- https://www.iit.edu/directory?organization_type=16
- https://www.iit.edu/about/contact-us
- https://www.iit.edu/admissions-aid/request-information
- https://www.iit.edu/admissions-aid/apply
- https://www.iit.edu/admissions-aid/visit-and-tour

Developed by Harish Hebsur

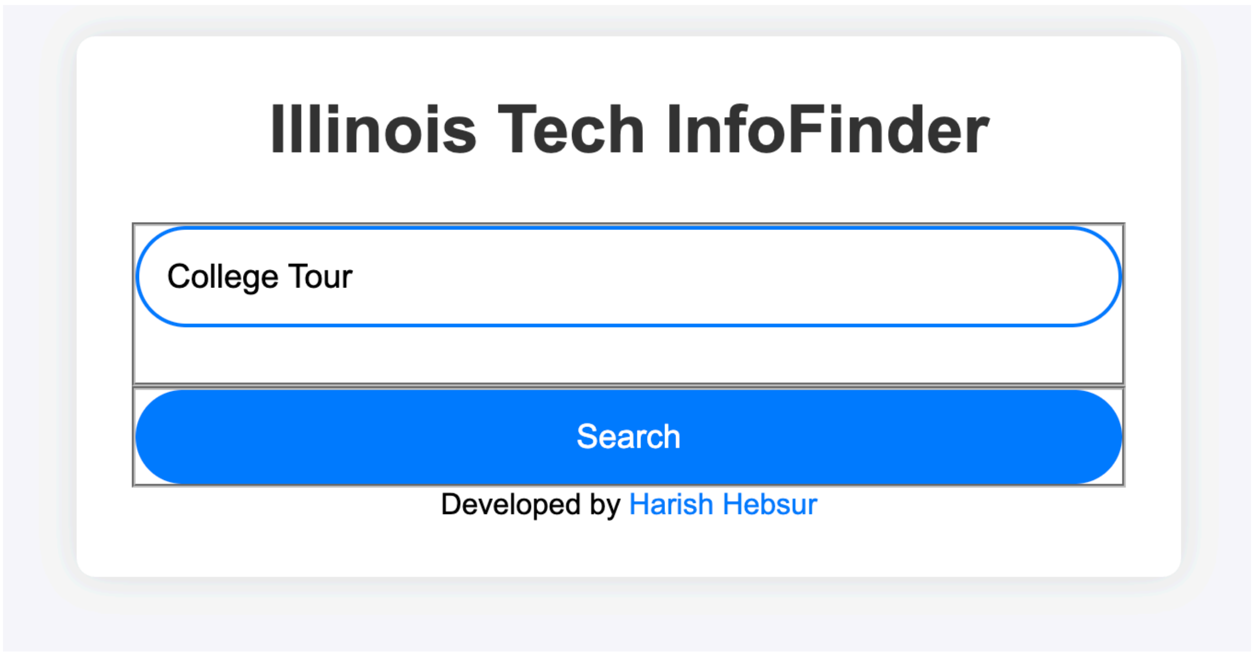



Illinois Tech InfoFinder

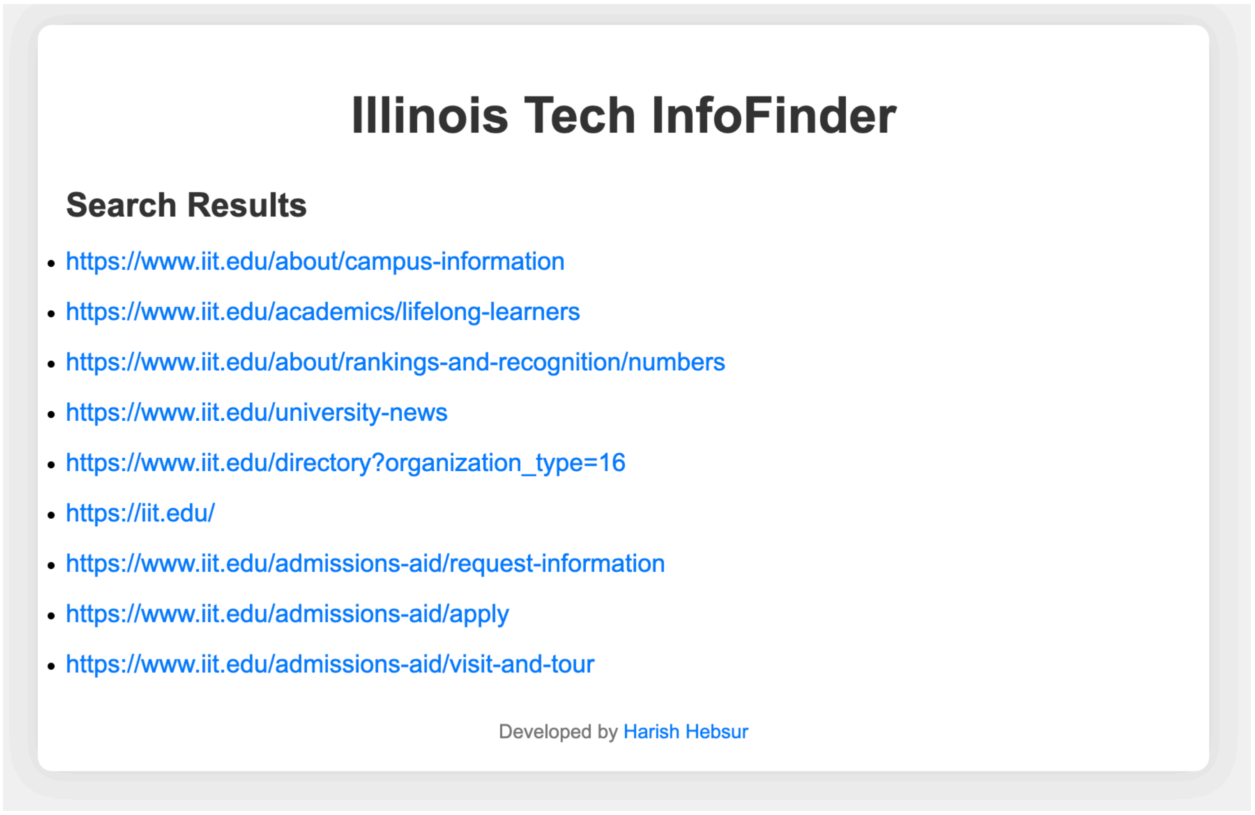
Search Results

- https://www.iit.edu/student-experience/new-student-transition
- https://www.iit.edu/academics/graduate-programs
- https://www.iit.edu/resources-parents-and-families
- https://www.iit.edu/about/campus-information
- https://www.iit.edu/student-experience/athletics
- https://www.iit.edu/resources-school-counselors
- https://www.iit.edu/research
- https://www.iit.edu/admissions-aid/apply
- https://www.iit.edu/admissions-aid/request-information

Developed by Harish Hebsur


Illinois Tech InfoFinder

Developed by Harish Hebsur


Illinois Tech InfoFinder

Search Results

- https://www.iit.edu/about/campus-information
- https://www.iit.edu/academics/lifelong-learners
- https://www.iit.edu/about/rankings-and-recognition/numbers
- https://www.iit.edu/university-news
- https://www.iit.edu/directory?organization_type=16
- https://iit.edu/
- https://www.iit.edu/admissions-aid/request-information
- https://www.iit.edu/admissions-aid/apply
- https://www.iit.edu/admissions-aid/visit-and-tour

Developed by Harish Hebsur

Cautions

1. Data Privacy: Care should be taken to ensure that user data and search queries are handled securely and in compliance with data privacy regulations.

2. Ethical Considerations: The search engine should prioritize delivering accurate and unbiased search results, adhering to ethical standards and principles of fairness and transparency.

3. Maintenance and Updates: Regular maintenance and updates are essential to address any issues, improve performance, and incorporate new features or enhancements.

In conclusion, while the Smart Search project has achieved significant success in developing an intelligent search engine for the IIT domain, ongoing efforts are needed to address challenges, optimize performance, and ensure the continued effectiveness and reliability of the search engine.

Data Sources

The Smart Search - Intelligent Search Engine project primarily relies on web data from the Illinois Institute Of Technology (IIT) domain for indexing and search operations. The data collection process involves crawling and indexing web pages within the IIT domain to build a comprehensive database for search queries.

Links and Access Information

-Seed URL/Domain:

The initial URL for crawling and indexing is provided as input to the crawler. For this project, the seed URL is set to `https://www.iit.edu`, representing the homepage of IIT.

-Access Information:

The crawling process is initiated programmatically using the provided seed URL and additional parameters such as the maximum number of pages to crawl and the maximum depth for traversal. These parameters are configurable based on the project requirements.

Downloads

- Web Data:

The web data collected during the crawling process is stored locally in the `./documents` directory. Each crawled web page is saved in HTML format along with relevant metadata.

- TF-IDF Files:

After preprocessing and indexing, the TF-IDF files representing the vector space model are generated and stored in the `./tf_idf_files` directory. These files contain term frequencies, inverse document frequencies, and other relevant information necessary for search operations.

Access Information

- Flask UI:

The user interface for the search engine is accessible locally through a Flask-based web application. Users can interact with the search engine by entering queries and retrieving search results through this interface.

Test Cases

Testing is an essential aspect of ensuring the reliability and effectiveness of the Search Engine. Several test cases are devised to evaluate different facets of the search engine, covering crawling, indexing, query processing, and user interface functionality. Here are the specifics of the test cases along with considerations for the testing framework, harness, and coverage.

Testing Framework:

- Unit Testing: Python's built-in `unittest` framework is utilized for crafting and executing unit tests to validate individual components and functions of the search engine codebase.

- Integration Testing: Integration tests are conducted to confirm the interactions and compatibility among various modules and subsystems of the search engine.

- UI Testing: The user interface undergoes testing to ensure its usability and functionality, encompassing input validation, query processing, and result presentation.

Test Harness:

- Automated Testing: Python scripts and testing frameworks are employed for automating test cases, streamlining the testing process, and facilitating continuous integration and deployment (CI/CD) pipelines.

- Test Suites: Test suites are organized based on different categories such as crawling, indexing, query processing, and UI testing. Each test suite comprises multiple test cases targeting specific functionalities.

Coverage:

- Code Coverage: Code coverage metrics are utilized to evaluate the percentage of code lines, branches, and functions covered by the executed test cases. Tools like `coverage.py` aid in measuring code coverage and identifying areas necessitating additional testing.

- Functional Coverage: Functional coverage criteria are defined to ensure comprehensive testing of all functional requirements specified in the project documentation. Test cases encompass various scenarios and edge cases to achieve comprehensive functional coverage.

Sample Test Cases:

1. Crawling Test Case:

- Objective: Confirm that the crawler retrieves web pages within the specified domain.

- Steps:

- Provide a seed URL within the IIT domain.
- Set the maximum number of pages to crawl.
- Verify that the crawler collects HTML content from the specified pages.
- Check for any errors or exceptions during the crawling process.

2. Indexing Test Case:

- Objective: Validate that the indexing process generates accurate TF-IDF representations for the crawled web pages.

- Steps:

- Preprocess the text extracted from crawled pages.

- Calculate TF-IDF scores for each term in the corpus.

- Verify the correctness of TF-IDF calculations by comparing with manually calculated scores for a subset of pages.

3. Query Processing Test Case:

- Objective: Ensure the search engine's capability to process user queries and retrieve relevant search results.

- Steps:

- Enter sample queries representing different information needs (e.g., course search, faculty search).

- Submit queries through the user interface.

- Verify that the returned search results are relevant and ranked appropriately based on relevance scores.

Coverage Analysis:

- Code Coverage Analysis: Employ `coverage.py` to assess the percentage of code coverage achieved by the unit tests and integration tests.

- Functional Coverage Analysis: Review test results and ascertain whether all functional requirements specified in the project documentation are adequately addressed by the test cases.

- Regression Testing: Perform regression testing to ensure that new features or changes do not introduce regressions or unintended side effects in existing functionalities.

Through a robust testing strategy encompassing unit tests, integration tests, and UI tests, the Smart Search - Intelligent Search Engine endeavors to provide a dependable and high-quality search experience for users within the University of Illinois at Chicago community.

Source Code

The source code for the Smart Search - Intelligent Search Engine project is available on GitHub. Below are the key components of the source code repository along with relevant documentation and dependencies.

Repository Structure:

1. Crawler Module: Contains scripts responsible for crawling web pages within the IIT domain and extracting textual content for indexing.

- crawl_all_sites.py: Main script for initiating web crawling process.

2. Indexer Module: Includes scripts for preprocessing crawled data, calculating TF-IDF scores, and building the inverted index.

- build_inverted_index.py: Script for constructing the inverted index.

3. Ranking Module: Implements the PageRank algorithm for ranking web pages based on their importance within the web graph.

- page_rank.py: Script for calculating PageRank scores for web pages.

4. Query Processor Module: Facilitates processing user queries and retrieving relevant search results.

- search.py: Script for processing user queries and fetching search results.

5. User Interface Module: Contains files related to the Flask-based user interface for interacting with the search engine.

- search_engine_web_app.py: Main script for running the Flask application.

- templates/: Directory containing HTML templates for rendering the user interface.

- static/: Directory for storing static files such as CSS stylesheets and JavaScript scripts.

6. Data Files: Stores data generated during the crawling, indexing, and ranking processes.

- tf_idf_files/: Directory storing TF-IDF representations of web pages.

Documentation:

- README.md: Provides an overview of the project, setup instructions, and usage guidelines.

- requirements.txt: Lists all Python dependencies required to run the project, including Scrapy, Flask, and NLTK.

Dependencies:

The project relies on several open-source libraries and frameworks, including:

- Scrapy (version 2.11+): For web crawling and extraction of textual content.

- Flask (version 2.2+): For building the user interface and handling HTTP requests.

- NLTK (Natural Language Toolkit): For text preprocessing tasks such as stop word removal and stemming.

- Scikit-Learn (version 1.2+): For calculating TF-IDF scores and cosine similarity.

- Beautiful Soup: For parsing HTML and XML documents.

- Other dependencies listed in the `requirements.txt` file.

The source code repository is well-documented with comments and explanatory README files to aid users and developers in understanding and contributing to the project. Developers are encouraged to fork the repository, suggest improvements, and contribute to the enhancement of the Smart Search - Intelligent Search Engine.

Bibliography

1. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.
2. Scrapy developers. (n.d.). Scrapy documentation. Retrieved from https://docs.scrapy.org/en/latest/
3. Flask Pallets Team. (n.d.). Flask documentation. Retrieved from https://flask.palletsprojects.com/en/2.0.x/
4. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media.
5. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
6. Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab.
7. Richardson, M., & Domingos, P. (2006). The intelligent surfer: Probabilistic combination of link and content information in PageRank. Advances in Neural Information Processing Systems, 18.
8. Floridi, L. (2011). The Philosophy of Information. Oxford University Press.
9. Mines, Human-Oriented. (2010). Similarity. Retrieved from http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/bhoenes/similarity.html
10. Lifewire. (n.d.). The 6 Best Search Engines for Academic Research. Retrieved from https://www.lowcountrygradcenter.org/the-6-best-search-engines-for-academic-research/
11. Freecodecamp. (n.d.). How to Process Textual Data Using TF-IDF in Python. Retrieved from https://medium.freecodecamp.org/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3
12. GeeksforGeeks. (n.d.). Page Rank Algorithm.

13. Wikipedia contributors. (2024). Web crawler. In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Web_crawler
14. ResearchGate. (n.d.). The Evaluation of WWW Search Engines. Retrieved from https://www.researchgate.net/publication/235286070_The_evaluation_of_WWW_search_engines
15. LowcountryGradCenter. (n.d.). How to Search Specific Domain in Google. Retrieved from https://www.lowcountrygradcenter.org/how-to-search-specific-domain-in-google-3481807