CSC 212 Project Report

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC 212 Data Structures Project Report – 2nd Semester 2024-2025

Developing a Photo Management Application

Authors

Name	ID	List of all methods implemented by each student
Rawan Alshammari	**444200348**	Photo: constructor, getPath, getTags Test: main, createTags, showAlbum. PhotoManager: addPhoto, deletePhoto.
Layan AlMurayshid	**444201057**	Album: constructor, getName, getCondition, getManager, getPhotos, getNbComps PhotoManager: addPhoto.
Razan Almutairi	**444200495**	PhotoManager: constructor, getPhotos, InvIndexPhotoManager: constructor, addPhoto, deletePhoto, getPhotos

Introduction

In this project, we developed a Photo Management Application that allows users to store images and organize them using descriptive tags. The focus of the system is to group photos based on tag-based conditions, making it easier to search and retrieve relevant images. For example, users can create albums that display only the photos matching certain tags, like “creature AND jungle.” The project was implemented entirely using custom-built data structures, including linked lists and binary search trees, without relying on any built-in Java libraries. In this report, we describe the different components of the system, starting with the specification, which explains the methods and classes that were implemented. We then cover the design, where we discuss how the system was structured and how the different parts interact. This is followed by the implementation section, which goes into detail about how key features were developed. Finally, we analyze the system’s performance and conclude with a summary of the work and suggestions for future improvements.

Specification

Photo

- Photo(String path, LinkedList<String> tags): Constructor
- String getPath(): Returns photo path
- LinkedList<String> getTags(): Returns associated tags

PhotoManager

- PhotoManager(): Constructor
- void addPhoto(Photo p): Adds photo if not already stored
- void deletePhoto(String path): Deletes photo by path
- LinkedList<Photo> getPhotos(): Returns all stored photos

Album

- Album(String name, String condition, PhotoManager manager): Constructor
- String getName(): Returns album name
- String getCondition(): Returns search condition
- PhotoManager getManager(): Returns manager
- LinkedList<Photo> getPhotos(): Returns photos matching condition
- int getNbComps(): Returns number of tag comparisons made

InvIndexPhotoManager

- InvIndexPhotoManager(): Constructor
- void addPhoto(Photo p): Adds photo to index
- void deletePhoto(String path): Removes photo from all tags
- BST<LinkedList<Photo>> getPhotos(): Returns inverted index tree

Design

The application relies on two main data structures: singly linked list and binary search tree. Photos are stored and accessed via the PhotoManager class. Albums can be created using logical AND tag conditions, evaluated dynamically using the tags stored in each Photo instance.

To improve search speed, the InvIndexPhotoManager builds an inverted index using a BST, where each node holds a tag as key and a list of photo references as value. This allows fast retrieval and scalable performance.

A helper class Test contains the main() method and demonstrates functionality by creating test photos and albums.

Implementation

- The Photo class encapsulates a photo's file path and associated tags.
- The PhotoManager maintains a linked list of all photos.
- The Album filters photos based on tag conditions using two approaches:
  1. Basic iteration through all photos.
  2. Optimized filtering through BST-based inverted index.
- The InvIndexPhotoManager class maintains the index and supports updates.
- All tag lookups and photo comparisons are done case-insensitively.

Sample code snippet for tag parsing:

```
private static LinkedList<String> createTags(String tagLine) {
   LinkedList<String> tagList = new LinkedList<>();
   String[] parts = tagLine.split(",");
   for (String tag : parts) {
      tagList.insert(tag.trim());
   }
   return tagList;
}
```

Performance analysis

- Without inverted index:
- Album.getPhotos() loops through all photos (n) and tags (m)
- Time Complexity: O(n * m)

- With inverted index (BST):
- Lookup each tag in BST: O(log t) where t is number of tags
- Intersect photo lists per tag: depends on tag frequency
- Time Complexity: O(m * log t + k), where k is the total number of photos in intersection

In conclusion, using the inverted index reduces time significantly, especially for large datasets.

Conclusion

This project successfully implemented a photo management system using custom data structures (linked list and BST). It supports album creation based on tag filtering and optimizes photo retrieval through an inverted index. Limitations include lack of OR/NOT query support and UI. Future improvements can include GUI support and more advanced query operators for richer search capabilities.

GitHub

Click here to view our project via GitHub