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
Aryam Almutairi	**444203968**	photo Class LinkedList Class BST Class main Class Project Report
Leen binmueqal	**444200885**	photoManager Class InvIndexPhotoManager Class main Class Project Report
Reema Alraqibah	**444201069**	Album Class invAlbum Class main Class Project Report

1. Introduction

This project is a photo management application that allows users to organize, add, and delete photos by associating them with tags. Users can create albums based on specific tag conditions, such as "animal AND grass". In the second phase, we optimized photo retrieval using an inverted index implemented using a Binary Search Tree (BST). This report covers the specifications, design, implementation, and performance analysis of the system.

2. Specification

Public class Photo
- Photo(String path, LinkedList<String> tags): constructor
- String getPath(): returns the photo file path
- LinkedList<String> getTags(): returns the photo tags

public class PhotoManager
- PhotoManager(): constructor
- LinkedList<Photo> getPhotos(): returns all photos
- void addPhoto(Photo p): adds a photo
- void deletePhoto(String path): deletes a photo by path

public class Album
- Album(String name, String condition, PhotoManager manager): constructor
- String getName(): returns album name
- String getCondition(): returns album condition
- PhotoManager getManager(): returns the manager
- LinkedList<Photo> getPhotos(): returns matching photos
- int getNbComps(): returns the number of comparisons

public class InvIndexPhotoManager
- InvIndexPhotoManager(): constructor
- void addPhoto(Photo p): adds a photo and updates inverted index
- void deletePhoto(String path): deletes a photo and updates index
- BST<LinkedList<Photo>> getPhotos(): returns the inverted index

3. Design

The project is divided into modular classes, each with specific responsibilities:
- Photo: Stores photo path and associated tags.
- LinkedList<T>: A custom generic linked list used in all classes.
- PhotoManager: Manages the collection of photos.
- Album: Filters photos based on a tag condition using logical AND.
- InvIndexPhotoManager: Uses a BST to implement an inverted index that maps tags to corresponding

photos.

When a photo is added, it is stored along with its tags. Albums dynamically filter photos by checking if all tags in the condition are present in a photo.
The inverted index improves search by associating each tag with a list of matching photos using a BST, ensuring efficient retrieval.

4. Implementation

The critical part of the implementation lies in filtering photos based on album conditions and optimizing retrieval with a BST.
In Album.getPhotos(), the condition is split by 'AND', and for each photo, we check if all tags exist. This requires tag comparison.
In InvIndexPhotoManager, a BST is used to store tags as keys and LinkedLists of Photos as values. When adding or deleting photos, the BST is updated accordingly.
This ensures that lookups for tags occur in O(log n) time on average, significantly improving performance compared to linear scanning.

Example snippet for photo filtering in Album:

```
for each photo:
   for each tag in condition:
      if tag not in photo.tags → skip
      else → continue checking
```

5. Performance analysis

Before using the inverted index:
- Album.getPhotos() iterates over all photos (n total).
- For each photo, it checks whether all condition tags (k tags) exist in the photo's tags.
- This involves comparing with each tag in the photo (average t tags).
- Therefore, the time complexity is O(n * k * t), where:
  N = number of photos
  K = number of tags in the condition
  T = average number of tags per photo

After using the inverted index:
- For each tag in the condition (k tags), we perform a lookup in a BST.
- The BST stores unique tags (t unique tags), so each lookup is O(log t).
- After fetching the photo lists for each tag, we perform intersection (AND condition).
- If m is the total number of photos involved in the intersection, then:
- Total complexity is O(k * log t + m), where:
  t = number of unique tags
  m = total size of all photo lists for the tags in the condition

This shows that the inverted index improves retrieval by avoiding full scans over all photos and instead directly accessing relevant photos per tag.

Tables of Time Complexities:

Before using the inverted index:

Class photoManager:

Method	Big-O	Description
`getPhotos()`	O(1)	Returns the list of all photos.
`addPhoto(Photo p)`	O(1)	Add a photo to the list.
`deletePhoto(String path)`	O(N)	Deletes a photo by scanning the list.

Class Album:

Method	Big-O	Description
`getPhotos()`	O(N*K*T)	Scans all photos and checks tags.
`getNbComps()`	O(N*K*T)	Counts tag comparisons.

N: Number of photos, **K**: Number of tags in the condition, **T**: average number of tags per photo

After using the inverted index:

Class InvIndexPhotoManager:

Method	Big-O	Description
`getPhotos()`	O(1)	Returns the BST-based inverted index.
`addPhoto(Photo p)`	O(n log n)	Updates BST for each tag in the photo.
`deletePhoto(String path)`	O(n log n)	Removes photo from BST entries.

Class InvAlbum:

Method	Big-O	Description
`getPhotos()`	O(K(log T + m))	Uses BST to find photos by tags.
`getNbComps()`	O(K*log T)	Counts BST searches for tags.

K = number of tags in the condition, T = number of unique tags,

m = total size of all photo lists for the tags in the condition

6. Conclusion

In this project, we implemented a complete photo management system using custom data structures. We provided functionalities for photo organization, album creation based on tag conditions, and optimized tag-based search using a BST-based inverted index. The system avoids using any built-in Java

collections, ensuring full control and understanding of data structure behavior.