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
Mariah alnfisah	444200965	
Ieen Aldbays		
Jood Mutarrid		

1. Introduction

This project aims to implement a simple photo management application using core data structures such as linked lists and binary search trees (BST). The system allows users to organize and retrieve photos based on associated tags, and supports building albums based on tag-based conditions. The report presents the implementation details, design choices, performance evaluation, and key findings of the application.

2. Specification

ADT: Photo

Elements: path: String, tags: LinkedList<String>
Structure: A photo has a file path and a list of tags
Domain: Unique path, tags must not be null
Operations:

- Photo(path, tags): constructor
- getPath(): returns photo path
- getTags(): returns tag list
- toString(): returns path as string

ADT: LinkedList<T>

Elements: T
Structure: Singly linked nodes, each has data and next
Domain: Can be empty, uses current for navigation
Operations:

insert(val): insert after current

remove(): delete current node

retrieve(): get current element

findFirst(), findNext(): navigation

last(), empty(), full(), find(val)

ADT: BST<T>

Elements: key: String, data: T, left/right nodes
Structure: Binary Search Tree ordered by key
Domain: Unique keys only
Operations:

insert(key, val): add key-value

findKey(key), retrieve()

removeKey(key), getNumComp()

inOrder(): in-order traversal

ADT: PhotoManager

Elements: photosList: LinkedList<Photo>
Structure: Stores all photos in linked list
Domain: Photos must be unique by path
Operations:

addPhoto(p): add if not exists

deletePhoto(path): remove by path

getPhotos(): return all photos

ADT: InvIndexPhotoManager

Elements: index: BST<LinkedList<Photo>>, photosList
Structure: BST maps tags → list of photos
Domain: Unique tags in BST
Operations:

addPhoto(p), deletePhoto(path)

getPhotos(), containsPhoto(list, p)

ADT: Album

Elements: name, condition, manager, nbcomp
Structure: Uses AND logic to filter photos by tags
Domain: Valid condition string; manager must not be null
Operations:

getPhotos(): photos matching condition

getName(), getCondition(), getNbComps()

3. Design

The system is designed to organize and search photos using tags. Each photo has a name and a list of descriptive tags. To support different search speeds and efficiencies, we implemented two types of managers.

In the basic design (**`PhotoManager`**), all photos are stored in a simple linked list, and search is done by scanning every photo. In the optimized version (**`InvIndexPhotoManager`**), a Binary Search Tree (BST) is used to index tags. This allows faster retrieval by directly accessing the photos related to each tag.

When a user creates an album with a condition like `"animal AND grass"`, the system splits the condition into tags, searches for each tag separately, and finds the common photos between them.

Adding and deleting photos updates both the main list and the tag index (if using the BST version). This layered design separates data storage from search logic and makes the system scalable and easy to understand

Component	Description
`Photo`	Stores the photo's file name and a list of tags.
`LinkedList<T>`	Used to store tags and photo lists.
`PhotoManager`	Manages photos using only a linked list.
`InvIndexPhotoManager`	Manages photos and creates a tag index using BST.
`BST<T>`	Each node holds a tag and a list of photos with that tag.
`Album`	Handles album creation based on conditions (e.g., `"animal AND grass"`).
`Test`	Adds test photos, creates albums, and prints results.

4. Implementation

```
Photo p = new Photo("fox.jpg", toTagsLinkedList("animal, fox, forest"));

photoManager.addPhoto(p);
```

In **`PhotoManager`**, it's simply added to a LinkedList.
In **`InvIndexPhotoManager`**, it's also added to a BST where each tag points to a list of photos.

Searching with Conditions:

In the `Album` class:

1. The condition (e.g., `"animal AND grass"`) is split into tags.
2. The manager checks for photos that match **all** these tags.

- In the basic version, all photos are scanned and matched manually.
- In the optimized version, each tag is searched in the BST, and the resulting photo lists are intersected.

```
photoManager.deletePhoto("fox.jpg");
```

- In **`InvIndexPhotoManager`**: removed from the photo list and also from each tag list in the BST.

BST Structure:

Each BST node looks like this:

```
class BSTNode<T> {
   String keyStr;
   T data;        ]}
```

And insertion uses standard BST logic:

```
if (keyStr.compareTo(current.keyStr) < 0)
   current.left = newNode;
else
   current.right = newNode;
```

5. Performance analysis

In our project, the performance of `Album.getPhotos()` depends on the type of manager used.

Using `PhotoManager`:

- Time Complexity: O(n × k × m)
- n: number of photos
- k: number of tags in the condition
- m: number of tags per photo
- Explanation: The system checks every photo and compares all its tags with each tag in the condition. This makes it slow when the number of photos increases.

Using `InvIndexPhotoManager` (with BST):

- Time Complexity: O(k × log t + k × s)
- t: number of tags stored in the BST
- s: number of photos under each tag
- Explanation: The system searches for each tag directly in the BST (log t), then retrieves the list of photos and intersects them. This is much faster than checking all photos.

The optimized version is faster and better for large datasets because it avoids scanning the entire list.

6. Conclusion

We developed a photo management system that supports adding, searching, and deleting photos by tags. Two managers were implemented: one using a linked list and another using a BST for faster search.

The `Album` class allows tag-based filtering like `"animal AND grass"`. Testing showed that the optimized version improves performance.

Limitations: Only AND conditions, no GUI, tags are case-sensitive.
Future Work: Add OR/NOT support, GUI, and file saving/loading.