**Project Report: B+ Tree Photo Storage System**

**Introduction:**

The B+ Tree Photo Storage System is a C++ project that implements a B+ tree data structure to efficiently store and manage a large collection of JPG photo files. The system provides functionality to insert photos into the tree, search for photos by their unique hash, remove photos from the tree, and display the stored photo information.

**Features Implemented:**

1. Photo Storage: The system allows storing information about JPG photo files, including the file name, unique SHA-512 hash, and the date the photo was added to the tree.

2. File Compression: Before inserting a photo into the B+ tree, the system compresses the photo file using the zlib library to reduce storage space.

3. File Decompression: When retrieving a photo from the B+ tree, the system decompresses the photo file to its original format.

4. Insertion: The system provides an option to insert a photo into the B+ tree based on its unique hash value.

5. Searching: The system allows searching for a photo in the B+ tree using its hash value.

6. Deletion: The system supports removing a photo from the B+ tree based on its hash value.

7. Display: The system can display the keys (hashes) of the photos stored in the leaf nodes of the B+ tree.

8. Directory Processing: The system can process photos stored in a directory on the computer, adding them to the B+ tree.

9. Multiple File Formats: The system can process photos stored both as jpgs or pngs.

**Code Functionality:**

This section explains how different parts of the application work together to achieve this functionality.

1. **Photo Struct:** The Photo struct represents a photo object and stores essential information about each photo. It includes the file name, a unique hash value generated from the photo's content, the date when the photo was added to the system, and the compressed photo data.
2. **BPlusTree Class:** The BPlusTree class is the core component of the application, implementing the B+ tree data structure. It has inner classes for nodes and methods for insertion, deletion, searching, and displaying the tree. The B+ tree is designed to efficiently store and retrieve photo objects based on their unique hash values.
3. **Hashing Photos:** Before inserting a photo into the B+ tree, the application generates a unique hash value for each photo using the SHA-512 hashing algorithm. The generateHash function takes the path of the photo file as input, reads the file content, and calculates the SHA-512 hash value. This hash value serves as a unique identifier for each photo and is used as the key in the B+ tree.
4. **Compressing Photos:** To optimize storage space, the application compresses photo files before storing them in the B+ tree. The compressPhoto function uses the zlib library to compress the photo data. It reads the photo file, compresses the data using zlib's deflate algorithm, and returns the compressed data as a vector of unsigned characters.
5. **Storing Photos in the B+ Tree:** The insert function of the BPlusTree class is responsible for inserting photo objects into the B+ tree. It takes the photo's hash value as the key and the Photo object as the value. The function traverses the tree to find the appropriate leaf node to insert the photo. If the leaf node has space, the photo is inserted directly. If the leaf node is full, the node is split, and the photo is inserted into the newly created leaf node. The splitting process may propagate up the tree to maintain the balance and structure of the B+ tree.
6. **Retrieving Photos from the B+ Tree:** The find function of the BPlusTree class allows searching for a photo in the B+ tree using its hash value. It traverses the tree to locate the leaf node containing the photo with the matching hash value. If found, it returns a pointer to the Photo object, which contains the compressed photo data and metadata. The decompressPhoto function is then used to decompress the photo data, restoring the original photo file.
7. **Deleting Photos from the B+ Tree:** The remove function of the BPlusTree class enables the deletion of a photo from the B+ tree based on its hash value. It searches for the photo in the tree and removes it from the corresponding leaf node. If the leaf node becomes underutilized after the deletion, the node may be merged with its sibling or redistributed to maintain the balance of the tree.
8. **Displaying the B+ Tree:** The display function of the BPlusTree class allows visualizing the structure of the B+ tree. It traverses the leaf nodes of the tree and prints the keys (hash values) stored in each leaf node, providing an overview of the stored photos.
9. **Directory Processing and Photo Loading:** The application can process photos stored in a directory on the computer. It uses the <filesystem> library to iterate over the files in the specified directory. It filters the files based on their file extensions (JPG or PNG) and adds them to the B+ tree without the need to generate file names dynamically.
10. **Progress Bar:** To provide visual feedback during the photo loading process, the application includes a ProgressBar class. The progress bar displays the percentage of photos processed and loaded into the B+ tree. It is initialized with the total number of photo files to be processed and is updated after each photo is added to the tree. The progress bar uses characters to represent the progress visually, filling up as more photos are processed.
11. **User Interaction:** The main function of the application provides a user interface for interacting with the photo storage system. It offers options to display the B+ tree, find a photo by its hash value, remove a photo by its hash value, and terminate the program. When a photo is found or removed, the application displays the photo's metadata, including the file name, hash value, and date added. It also decompresses the photo data and saves it as a new file for viewing or further processing.

**Benefits of compressing photos:**

1. Reduced storage space: By compressing the photos before storing them in the B+ tree, you can save storage space. Compressed photos will occupy less memory compared to their uncompressed counterparts, allowing users to store more photos within the same space constraints.
2. Faster disk I/O: When reading or writing compressed photos from/to disk, the amount of data transferred is reduced due to compression. This can lead to faster disk I/O operations, especially when dealing with a large number of photos.

**Potential drawbacks:**

1. Compression overhead: Compressing and decompressing photos introduce additional computational overhead. The compression process takes time and consumes CPU resources, which can impact the overall performance of the program, particularly when adding a large number of photos.
2. Decompression overhead: Whenever a photo needs to be accessed or displayed, it needs to be decompressed first. The decompression process also requires time and CPU resources, which can affect the responsiveness of the program when retrieving and viewing photos.

**Required Packages:**

To compile and run the code, the following packages need to be installed:

1. C++ Compiler: A C++ compiler that supports C++17 or later, such as GCC or Clang.

2. OpenSSL Library: The OpenSSL library is required for the SHA-512 hash calculation. It should be installed and properly linked to the project.

3. Zlib Library: The zlib library is required for photo file compression and decompression. It should be installed and properly linked to the project.

4. Filesystem Library: The `<filesystem>` library is used for directory processing. It is part of the C++17 standard library, so no additional installation is required if using a compatible C++ compiler.

**LIMITATIONS**

* Fixed Compression Algorithm:
* The system uses the zlib library for photo file compression and decompression which is an efficient compression library but it may not always provide the best compression ratio for all types of photo files.
* Lack of Metadata Management:
* The system currently stores only basic information about each photo, such as the file name, hash, and date added. Additional metadata associated with the photos, such as camera settings, geolocation data, or user-defined tags is not implemented.
* Single-Machine Deployment:
* The current implementation of the system is designed to run on a single machine, with the photos stored in a local directory.
* Lack of User Interface:
* The system is implemented as a command-line application and does not provide a graphical user interface (GUI) for interacting with the photo collection.
* Limited Search Capabilities:
* The system currently supports searching for photos based on their exact hash values. More efficient search methods can be implemented such as meta-data based search or content-based Image Retrieval using feature extraction and indexing based on image characteristics like color, texture, or shape.

**Conclusion:**

The B+ Tree Photo Storage System provides an efficient and optimized solution for storing and managing a large collection of JPG photo files. By utilizing a B+ tree data structure, the system achieves fast insertion, searching, and deletion operations while minimizing memory usage through file compression. With the necessary packages installed, the system can be compiled and run to store, retrieve, and manage photo files efficiently.

**BIBLIOGRAPHY**

[How to Thoroughly Install ZLIB on Windows 10 and 11 for CUDA and cuDNN – TheLinuxCode](https://thelinuxcode.com/install-zlib-windows-10-11/)

<https://www.kaggle.com/datasets/scolianni/mnistasjpg?resource=download>