پویا امید 14014421005

فشرده سازی

بخش‌های مختلف کد

1. **کلاس:Node**

   این کلاس هر گره (Node) در درخت هافمن را نمایش می‌دهد.

   هر گره شامل یک کاراکتر، تعداد تکرار آن (فرکانس) و اشاره‌گرهایی به گره‌های چپ و راست است.

   گره‌ها بر اساس تعداد تکرار کاراکترها مرتب می‌شوند.

2. **کلاس:HuffmanCoding**

   کلاس اصلی که شامل تمام متدهای لازم برای فشرده‌سازی و از حالت فشرده خارج کردن فایل‌ها است.

   در متدmain ، برنامه از کاربر می‌خواهد که یکی از دو عملیات فشرده‌سازی یا از حالت فشرده خارج کردن را انتخاب کند و سپس فایل‌های مورد نظر را وارد کند.

فشرده‌سازی

1. **خواندن فایل** :محتوای فایل را به صورت یک رشته می‌خواند.
2. **ایجاد جدول فرکانس** :تعداد تکرار هر کاراکتر در متن را محاسبه می‌کند.
3. **ساخت درخت هافمن** :از یک صف اولویت‌دار (PriorityQueue) برای ساخت درخت بر اساس فرکانس کاراکترها استفاده می‌کند.
4. **تولید کدهای هافمن** :با پیمایش درخت هافمن، کدهای باینری مربوط به هر کاراکتر را ایجاد می‌کند.
5. **رمزگذاری متن** :هر کاراکتر در متن را با کد هافمن متناظر جایگزین می‌کند.
6. **نوشتن فایل فشرده‌شده** :متن رمزگذاری شده را در یک فایل جدید با پسوند `huffman`.ذخیره می‌کند.
7. **ذخیره‌سازی کدهای هافمن** :کدهای کاراکترها را در یک فایل جداگانه ذخیره می‌کند.

از حالت فشرده خارج کردن

1. **خواندن فایل فشرده‌شده** :محتوای فایل رمزگذاری شده را می‌خواند.
2. **خواندن کدهای هافمن** :کدهای کاراکترها را از فایل مربوطه می‌خواند.
3. **بازسازی درخت هافمن** :با استفاده از کدهای ذخیره شده، درخت هافمن را بازسازی می‌کند.
4. **رمزگشایی متن** :رشته باینری را به متن اصلی تبدیل می‌کند.
5. **نوشتن فایل از حالت فشرده خارج شده** :متن اصلی را در یک فایل جدید با پسوند `decoded`.ذخیره می‌کند.

توضیحات هر متد

1. **readFile**:محتوای فایل ورودی را می‌خواند و به صورت یک رشته برمی‌گرداند.
2. **writeFile**:محتوای رشته‌ای را در یک فایل ذخیره می‌کند.
3. **writeHuffmanCodes**:کدهای هافمن را در یک فایل ذخیره می‌کند.

4. **readHuffmanCodes**: کدهای هافمن را از یک فایل می‌خواند و به صورت یک Map برمی‌گرداند.
5. **createFrequencyTable**: جدول فرکانس کاراکترها را از متن ورودی ایجاد می‌کند.
6. **buildHuffmanTree**: درخت هافمن را با استفاده از جدول فرکانس کاراکترها می‌سازد.
7. **createHuffmanCodes**: کدهای هافمن را با پیمایش درخت هافمن ایجاد می‌کند.
8. **huffmanEncoding**: متن اصلی را با استفاده از کدهای هافمن رمزگذاری می‌کند.
9. **huffmanDecoding**: متن رمزگذاری شده را با استفاده از درخت هافمن به متن اصلی تبدیل می‌کند.
10. **buildHuffmanTreeFromCodes**: درخت هافمن را از روی کدهای ذخیره شده می‌سازد.

متدهای اصلی

1. **readFile**

هدف :خواندن محتوای یک فایل و بازگرداندن آن به صورت یک رشته.

الگوریتم:

1. یک `StringBuilder` برای ذخیره محتوای فایل ایجاد کنید.
2. یک `BufferedReader` برای خواندن فایل ایجاد کنید.
3. هر خط از فایل را بخوانید و به `StringBuilder` اضافه کنید.
4. در پایان، محتوای `StringBuilder` را به صورت رشته برگردانید.

2. **writeFile**

هدف :نوشتن یک رشته در یک فایل.

الگوریتم:

1. یک `FileWriter` برای نوشتن در فایل ایجاد کنید.
2. محتوای رشته ورودی را در فایل بنویسید

3. **writeHuffmanCodes**:

هدف :ذخیره کدهای هافمن در یک فایل.

الگوریتم:

1. یک `FileWriter` برای نوشتن در فایل ایجاد کنید.
2. برای هر ورودی در `Map` کدهای هافمن، کلید و مقدار را در فایل بنویسید.

4. **readHuffmanCodes**:

هدف :خواندن کدهای هافمن از یک فایل و ذخیره آن‌ها در یک `Map`.

الگوریتم:

1. یک `BufferedReader` برای خواندن فایل ایجاد کنید.
2. هر خط از فایل را بخوانید و آن را به دو قسمت (کاراکتر و کد) تقسیم کنید.

3. کاراکتر و کد را در `Map` ذخیره کنید.

5. **createFrequencyTable**:

هدف :ایجاد جدول فرکانس کاراکترها از یک متن.

الگوریتم:

1. یک `Map` برای ذخیره فرکانس کاراکترها ایجاد کنید.
2. برای هر کاراکتر در متن، فرکانس آن را در `Map` به روز کنید.

6. **buildHuffmanTree**:

هدف :ساخت درخت هافمن با استفاده از جدول فرکانس کاراکترها.

الگوریتم:

1. یک صف اولویت‌دار (`PriorityQueue`) از گره‌ها ایجاد کنید.
2. برای هر کاراکتر در جدول فرکانس، یک گره جدید ایجاد کرده و آن را به صف اضافه کنید.
3. تا زمانی که بیشتر از یک گره در صف وجود دارد:
   - دو گره با کمترین فرکانس را از صف بیرون بیاورید.
   - یک گره جدید که فرکانس آن مجموع فرکانس دو گره قبلی است، ایجاد کنید.
   - گره جدید را به صف اضافه کنید.
4. گره باقیمانده در صف، ریشه درخت هافمن است.

7. **createHuffmanCodes**:

هدف :تولید کدهای هافمن برای هر کاراکتر با پیمایش درخت هافمن.

الگوریتم:

1. اگر گره فعلی تهی نیست:
   - اگر گره یک برگ است (یعنی هیچ فرزندی ندارد)، کد جاری را به عنوان کد هافمن کاراکتر ذخیره کنید.
   - اگر گره برگ نیست:
     - به سمت چپ گره بروید و کد "0" را به کد جاری اضافه کنید.
     - به سمت راست گره بروید و کد "1" را به کد جاری اضافه کنید.

8. **huffmanEncoding**:

هدف :رمزگذاری متن اصلی با استفاده از کدهای هافمن.

الگوریتم:

1. یک `StringBuilder` برای ذخیره متن رمزگذاری شده ایجاد کنید.
2. برای هر کاراکتر در متن، کد هافمن آن را از `Map` بگیرید و به `StringBuilder` اضافه کنید.
3. متن رمزگذاری شده را به صورت رشته برگردانید.

9. **huffmanDecoding**:

هدف :رمزگشایی متن رمزگذاری شده با استفاده از درخت هافمن.

الگوریتم:

1. یک `StringBuilder` برای ذخیره متن رمزگشایی شده ایجاد کنید.
2. از ریشه درخت هافمن شروع کنید و برای هر بیت در متن رمزگذاری شده:
   - اگر بیت "0" است، به سمت چپ درخت بروید.
   - اگر بیت "1" است، به سمت راست درخت بروید.
   - اگر به یک گره برگ رسیدید، کاراکتر آن گره را به `StringBuilder` اضافه کنید و به ریشه درخت برگردید.
3. متن رمزگشایی شده را به صورت رشته برگردانید.

10. **buildHuffmanTreeFromCodes**:

هدف :ساخت درخت هافمن با استفاده از کدهای ذخیره شده.

الگوریتم:

1. یک گره ریشه جدید ایجاد کنید.
2. برای هر ورودی در `Map` کدهای هافمن:
   - از ریشه شروع کنید و برای هر بیت در کد:
     - اگر بیت "0" است و گره چپ تهی است، یک گره جدید در سمت چپ ایجاد کنید و به سمت چپ بروید.
     - اگر بیت "1" است و گره راست تهی است، یک گره جدید در سمت راست ایجاد کنید و به سمت راست بروید.
   - وقتی به انتهای کد رسیدید، کاراکتر را در گره جاری ذخیره کنید.

متد `main` در برنامه نقش مرکزی را در اجرای فشرده‌سازی و از حالت فشرده خارج کردن فایل‌ها با استفاده از الگوریتم هافمن ایفا می‌کند.

خواندن ورودی کاربر

برنامه از کاربر می‌خواهد که عملیات فشرده‌سازی یا از حالت فشرده خارج کردن را انتخاب کند و فایل‌های مورد نظر را وارد کند

پردازش فایل‌ها بر اساس انتخاب کاربر

بر اساس ورودی کاربر(choice) ، برنامه یکی از دو عملیات فشرده‌سازی یا از حالت فشرده خارج کردن را انجام می‌دهد.

فشرده‌سازی(choice == 1)

اگر کاربر عدد 1 را انتخاب کند، برنامه وارد بلوک فشرده‌سازی می‌شود

مراحل فشرده‌سازی:

1. خواندن محتوای فایل با استفاده از .readFile
2. ایجاد جدول فرکانس کاراکترها با استفاده از .createFrequencyTable
3. ساخت درخت هافمن با استفاده از .buildHuffmanTree
4. ایجاد کدهای هافمن با استفاده از .createHuffmanCodes
5. رمزگذاری متن با استفاده از .huffmanEncoding
6. ذخیره متن رمزگذاری شده در یک فایل با استفاده از .writeFile
7. نمایش پیام موفقیت آمیز فشرده‌سازی فایل.
8. ذخیره کدهای هافمن در یک فایل جداگانه با استفاده از .writeHuffmanCodes

از حالت فشرده خارج کردن(choice == 2)

اگر کاربر عدد 2 را انتخاب کند، برنامه وارد بلوک از حالت فشرده خارج کردن می‌شود

مراحل از حالت فشرده خارج کردن:

1. خواندن محتوای فایل رمزگذاری شده با استفاده از .readFile
2. خواندن کدهای هافمن از فایل مربوطه با استفاده از .readHuffmanCodes
3. ساخت درخت هافمن با استفاده از .buildHuffmanTreeFromCodes
4. رمزگشایی متن با استفاده از .huffmanDecoding
5. ذخیره متن رمزگشایی شده در یک فایل با استفاده از .writeFile
6. نمایش پیام موفقیت آمیز از حالت فشرده خارج کردن فایل.

برخورد با ورودی نامعتبر

اگر کاربر ورودی نامعتبری (غیر از 1 یا 2) وارد کند، برنامه پیام خطا را نمایش می‌دهد