# ترجمه پایان نامه

**موضوع پایان نامه : بهبود نسبت فشرده سازی متن برای متن ASCII**

**این ترجمه توسط مریم سادات موردگر دانشجوی کارشناسی ارشد مهندسی نرم افزار کامپیوتر**
**با راهنمایی دکتر سید علی رضوی ابراهیمی در تابستان ۱۴۰۰ به منظور بررسی و مرور پایان نامه**
**بدون هیچ دخل و تصرفی در آن صورت گرفته است.**

UiT The Arctic University of Norway

دانشگاه نروژ (قطب شمال)

دانشکده علم و فناوری

گروه علوم کامپیوتر

# بهبود نسبت فشرده سازی متن برای متنASCII

با استفاده از ترکیبی از کدگذاری دیکشنری، فشرده سازی ASCII و کدگذاری هافمن

**ساندر هالدار-ایورسن**

پایان نامه کارشناسی ارشد INF-3990 در علوم کامپیوتر-نوامبر ۲۰۲۰

# بهبود نسبت فشرده سازی متن برای متنASCII

## با استفاده از ترکیب کدگذاری دیکشنری ، فشرده سازی ASCII و کدگذاری هافمن

ساندر هالدار-ایورسن

**چکیده**

فشرده سازی داده ها زمینه ای است که به طور گسترده مورد تحقیق قرار گرفته است. بیشتر الگوریتم های فشرده سازی که امروزه مورد استفاده قرار می گیرند، از چندین دهه پیش است، مانند *کدگذاری هافمن* و *کدگذاری دیکشنری* که همه منظوره هستند. الگوریتم های فشرده سازی و می تواند در هر چیزی از داده های متنی گرفته تا تصاویر و ویدئو استفاده شود. با این حال، الگوریتم های بسیار کمی برای فشرده سازی انواع خاصی از داده ها، مانند متنASCII وجود دارد که به اتلاف داده باشند. این پروژه در مورد ایجاد یک راه حل فشرده سازی متن با استفاده از ترکیبی از سه مورد است الگوریتم فشرده سازی کدگذاری دیکشنری، الگوریتم فشرده سازی ASCII و الگوریتم فشرده سازی کدینگ هافمن. این راه حل مخصوص فشرده سازی متن ASCII است، اما می تواند در هر شکلی از متن استفاده گردد. الگوریتم ها برای ایجاد یک نمونه اولیه ترکیب می شوند که بر اساس برنامه های فشرده سازی عمومی مورد ارزیابی قرار می گیرد. یک ارزیابی در برابر فشرده سازی ASCII از برنامه Shoco انجام خواهد شد.

نتایج حاصل از ارزیابی نشان می دهد که ترکیب کدگذاری دیکشنری، فشرده سازی ASCII و کدینگ هافمن از نسبت فشرده سازی به دست آمده از برنامه های فشرده سازی عمومی استفاده نمی کند.

**واژگان کلیدی**

فشرده سازی متن، کدگذاری دیکشنری، ASCII، کدینگ هافمن

**فهرست مطالب**

## 1 مقدمه

فشرده سازی داده ها فرایند کاهش تعداد بیت های مورد نیاز برای نمایش رسانه است [۱]. به عبارت دیگر، کاهش حجم داده ها در حفظ تمامیت آن. فشرده سازی داده ها می تواند در هنگام پخش ویدئو از فضای ذخیره سازی ذخیره کرده یا پهنای باند را کاهش دهد [۲]. زیر مجموعه فشرده سازی داده ها فشرده سازی متن است. فشرده سازی متن فشرده سازی فایل های متنی است که فایل هایی هستند که شامل کاراکترها و نمادها می باشد. یک تفاوت مهم بین فشرده سازی متن و داده های باینری، مانند تصاویر و ویدئو، [۱] این است که فشرده سازی متن باید بدون اتلاف باشد. فشرده سازی بدون اتلاف به این معنی است که یک فایل را می تون فشرده کرد بطوری که فایل اصلی بدون از دست دادن اطلاعات بازیابی شود[۱]. الگوریتم های فشرده سازی که قادر به فشرده سازی داده ها نیستند از فشرده سازی اتلافی استفاده می کنند. فشرده سازی از دست رفته بیشتر در داده های باینری استفاده می شود که در آن برخی از داده ها از بین می روند[۳].

### 1.1 پیشینه

فشرده سازی داده ها توسط الگوریتم های فشرده سازی انجام می شود. چندین الگوریتم متداول در برنامه های فشرده سازی استفاده می شود [۴، ۵، ۶]، و بسیاری از آن ها چندین دهه است که وجود دارند. الگوریتم متداولی که هنوزمورد استفاده قرار می گیرد، کدگذاری هافمن [۷] است که در سال ۱۹۵۲ اختراع شد. کدگذاری هافمن نویسه‌های پرکاربردتر را با رشته‌های بیتی کوتاهتری نسبت به آن‌هایی که کاربردشان کمتر است، نشان داده می‌شوند. بنابراین یک متن انگلیسی ممکن است فشرده سازی خوبی به دست آورد، به عنوان مثال، حرف"e" بیشتر ازحرف "z" استفاده می شود [۸]. یکی دیگر از الگوریتم ها، کدگذاری دیکشنری است که تکنیک جایگزینی زیر رشته ها و کلمات با ارجاع به دیکشنری است. دیکشنری و منابع، فایل فشرده را تشکیل می دهند که می توان برای بازیابی فایل اصلی آن ها را از حالت فشرده خارج کرد. LZ77 و LZW از پرکاربردترین الگوریتم های کدگذاری دیکشنری هستند [۴،۵،۶و۹]. هر دو الگوریتم مبتنی بر تکنیک جایگزینی زیر رشته های تکراری با منابع هستند، اما از رویکردهای متفاوتی استفاده می کنند.

الگوریتم DEFLATE ترکیبی از کدگذاری دیکشنری (LZ77) [۱۰] و کدگذاری هافمن [۱۱] است.

DEFLATE در چندین برنامه فشرده سازی مانند Gzip، zip و WinRar استفاده می شود [۴،۵و۶]. با وجود این که در زمینه فشرده سازی داده ها تحقیقات گسترده ای انجام شده است، فناوری های جدید همچنان در حال ظهور هستند [۱۲،۱۳و۱۴]، مانند Shoco [۱۵]، که در سال ۲۰۱۴ منتشر شد[۱۶]. Shoco یک برنامه فشرده سازی متن است که به ویژه بر روی متن ASCII موثر است[۱۵]. ASCII یک کدگذاری برای متن ساده است؛ یعنی متنی که از تنوع زیادی از کارکترها و نمادها استفاده نمی کند. رمزگذاری کاراکتر یک متن به کامپیوتر می گوید چگونه داده ها باید خوانده شوند تا انسان بتواند آن را درک کند[۱۷].

### 1.2 مسئله

انتقال حجم زیادی از داده ها از طریق اینترنت کار زمان بری است. حتی در شرایطی که یک کاربر دارای پهنای باند فوق العاده باشد، باز هم انتقال مگابایت یا گیگابایت داده باز هم چند دقیقه طول می کشد. این مشکل اغلب مربوط به فایل های باینری مانند فیلم ها است، اما ممکن است در مورد داده های مبتنی برمتن نیز صادق باشد. یک مثال می تواند شبیه سازی مخزن گیت باشد، هسته لینوکس مخزن گیت هاب [۱۸] نزدیک به سه گیگابایت حجم دارد [۱۹]. داده های متنی را می توان با استفاده از الگوریتم هایی مانند DEFLATE فشرده کرد، که توسط چندین برنامه فشرده سازی محبوب استفاده می شود [۴،۵ و ۶]. فشرده سازی داده ها حجم فایل را کاهش می دهد، پس انتقال آن از طریق اینترنت سریعتر خواهد بود الگوریتم DEFLATE برای کار بر روی انواع متن ها و داده های باینری طراحی شده است [۱]. با این حال یک الگوریتم فشرده سازی عمومی است که کار با آن می تواند بهینه شود. به عنوان مثال متن ASCII اکثر برنامه های فشرده سازی عمومی هستند و از الگوریتم های بدون اتلاف استفاده می کنند که به فشرده سازی هر نوع داده مانند متن، تصویرو فایل های ویدئویی منجر می شود[۴،۵،۶

و۲۰]. چندین الگوریتم فشرده سازی تلفاتی وجود دارد که به طور خاص برای هر نوع داده استفاده می شوند، اما تعداد ان ها کمتر از الگوریتم های بدون اتلاف است [۲۰]. این امکان وجود دارد که از الگوریتم های بدون اتلاف سفارشی برای فشرده سازی انواع داده های خاص، مانند متن ASCII، نسبت به الگوریتم های فشرده سازی عمومی، بیشتراستفاده شود.

پایان نامه به سوالات تحقیق زیر پاسخ می دهد:

چه ترکیبی از تکنیک ها می تواند نسبت فشرده سازی را بهبود بخشد؟

آیا استفاده از ترکیب کدگذاری دیکشنری و کدگذاری هافمن برای فشرده سازی ASCII، نسبت به فشرده سازی DEFLATE فراتر می رود؟

آیا این ترکیب فشرده سازی متن، فشرده سازی بالاتری نسبت به برنامه های فشرده سازی عمومی برای متن ASCII به دست می آورد؟

آیا این ترکیب فشرده سازی متن، فشرده سازی بالاتری نسبت به برنامه های فشرده سازی که مخصوص متن ASCII هستند، به دست می آورد؟

### 1.3 پیشنهادات و هدف

اکثر برنامه های فشرده سازی از الگوریتم DEFLATE [۵،۴ و ۶] استفاده می کنند، که شامل کدگذاری دیکشنری و کدگذاری هافمن است. در این پروژه هم از این موارد استفاده خواهد شد زیرا موثر بودن این الگوریتم های استاندارد عملا ثبات شده است [۲۱ و ۲۲]. الگوریتم های زیر در این پروژه استفاده می شود:

- کدگذاری دیکشنری؛ به ویژه در متن نوشتاری موثر است، مانند دایرةالمعارف [۲۳].
- فشرده سازی ASCII؛ بر روی متن نوشتاری[۱] به ویژه جایی که از کاراکترهای خاص بیشتر از بقیه استفاده می شود، موثر است [۸].
- کدگذاری هافمن ؛ تکنیکی که به طور گسترده در فشرده سازی داده ها استفاده شده است [۲۴ و ۲۵].

این پروژه با استفاده از این سه روش، راه حلی را ایجاد می کند که برای فشرده سازی متن اسکی مطلوب است و سعی می کند نسبت فشرده سازی بیشتری برای متن اسکی نسبت به یک الگوریتم فشرده سازی عمومی داشته باشد.

### 1.4 انگیزه

راه حل فشرده سازی متن پیشنهادی (که در اینجا به آن اشاره می شود TCSیا "ترکیبی") می تواند برای هرکسی که مقدار زیادی (عمدتا) متن اسکی سر و کار دارد مفید باشد. این راه حل به گونه ای طراحی شده است که روی هر فرم متن کار کند، اما برای متن اسکی سفارشی شده است. این راه حل حجم فایل را کاهش می دهد تا فضای زیادی را اشغال نکند وهمچنین هنگام بارگیری یا انتقال یک فایل فشرده از طریق اینترنت، پهنای باند را کاهش می دهد. کدگذاری دیکشنری [۱۱، ۲۶] و کدگذاری هافمن [۱۱] از قبل به طور گسترده ای در الگوریتم های فشرده سازی استفاده می شود [۴، ۵، ۶] (هر دو در DEFLATE)، اما الگوریتم های فشرده سازی کمی از روش فشرده سازی اسکی استفاده می کنند [۴، ۵، ۶ و۲۷]. بنابراین این ترکیب ممکن است فشرده سازی بهتری برای متن اسکی نسبت به الگوریتم های فشرده سازی عمومی استفاده کند.

### 1.5 روش

تحقیقات آماری که با داده های عددی سروکار دارند، کم هستند، در حالی که تحقیقات کیفی، تحقیقاتی و معمولاً غیرآماری است [۲۸]. آزمایشات برای سیستم هنگام مقایسه TCS با داده های عددی استفاده می شود، بنابراین تحقیقات کمی اندازه گیری های بهتری را در مقایسه با تحقیقات کیفی ارائه می دهد [۲۹و ۳۰]. درمقابل آن برای نتیجه گیری از آزمایشات در تحقیقات کیفی به استدلال منطقی نیاز است که این استدلال می تواند استدلال قیاسی یا استدلال استقرایی باشد [۲۹]. استدلال قیاسی تشکیل یک نتیجه گیری بر اساس اظهارات یا حقایق عمومی پذیرفته شده است. استدلال استقرایی از منطق زیر پیروی می کند: اگر $A = B$ و $B = C$ سپس $A = C$ . استدلال استقرایی شامل یک عنصر احتمال است و ممکن است به این نتیجه برسد که A احتمالاً برابر با B است [۳۰].

[۱] " متن نوشتاری" در این پایان نامه به عنوان متنی که به برخی از زبان ها مانند یک زبان گفتاری یا یک زبان برنامه نویسی نوشته شده است تعریف شده است.

در این پروژه تعدادی از تکنیک های فشرده سازی بر روی متون مشابه آزمایش می شوند، و نسبت فشرده سازی برای پیدا کردن تکنیک بیشترین فشرده سازی را به دست می آورد. آزمایشات داده های قطعی در مورد این که کدام راه حل بهترین نسبت فشرده سازی را به دست می آورد، ارائه می دهد و بنابراین نتیجه گیری با استدلال استقرایی یا قیاسی صورت می گیرد.

### 1.6 محدودیت ها

این پروژه به طور خاص برای فشرده سازی متن، و نه داده های باینری، مانند فایل های تصویری یا ویدیویی در نظر گرفته شده است. بنابراین آزمایش های انجام شده در این پروژه تنها روی داده های متنی می باشد. انتظار می رود که از ترکیب برای فشرده سازی مقادیر زیادی از متن استفاده شود، زیرا داده های متنی معمولاً فضای ذخیره سازی زیادی را در مقایسه با داده های ویدئویی یا یک فایل متنی کوچک که کاربرد چندانی ندارد؛ اشغال نمی کند. بنابراین فقط روی فایل های متنی با اندازه قابل توجه آزمایش می شود. اثربخشی راه حل (تأخیر) عاملی در ارزیابی نیست، زیرا ترکیب فقط بر اساس نسبت فشرده سازی آن ارزیابی می شود، نه زمان اجرا. پایتون به عنوان زبان برنامه نویسی برای ماژول فشرده سازی ASCII (ACM) انتخاب شده است تا کد منبع را ساده تر کند. پیاده سازی برنامه در C می توانست راه حل را سریعتر کند [۱۰۴و ۱۰۵]، اما قابلیت فشرده سازی برای ACM اولویت بالاتری نسبت به تأخیر دارد. این پروژه پیاده سازی های مختلفی را برای استفاده در TCS ارزیابی کرد. پیاده سازی ها بر اساس میانگین نتایج حاصل از آزمایشات ارزیابی می شوند. این بدان معناست که موارد استفاده در آزمایش اهمیت برابردارند. این پروژه با هیچ معضل اخلاقی مواجه نمی شود. این پروژه همکاری با یک شرکت نیست، بنابراین هیچ مشکلی با کپی رایت (حق چاپ) آن وجود نخواهد داشت. این پروژه تنها از کد منبع با مجوز باز استفاده می کند. آزمایشات روی فایلهای ویکی پدیا (به بخش ۴،۱،۱ مراجعه کنید) فقط از یک زیرمجموعه از متن برای آزمایش فشرده سازی آنها استفاده می کند، زیرا استفاده از چنین مجموعه داده بزرگی برای آزمایش غیرممکن است. زیر مجموعه به اندازه کافی بزرگ است که نتایج واضحی را ارائه می دهد.

### 1.7 طرح کلی

فصل دوم این پایان نامه توضیح می دهد که فشرده سازی متن چیست و سابقه و کار مرتبط برای سه ماژول TCS را ارائه می دهد. کدگذاری دیکشنری، فشرده سازی اسکی و کدگذاری هافمن. فصل سوم روش های مختلفی که در تحقیقات دانشگاهی برای این پروژه استفاده می شود، و دلیل انتخاب این روش ها را، ارائه می دهد. فصل چهارم طراحی TCS و
الزامات راه حل را ارائه می دهد. فصل پنجم فرایند ارزیابی TCS و ماژول های آن را شرح می دهد و نتایج حاصل از ارزیابی ها را ارائه می دهد. فصل ششم بحث نتایج حاصل از ارزیابی ها را پوشش می دهد و فصل هفتم نتیجه گیری و کار آینده پایان نامه را ارائه می دهد.

## 2 فشرده سازی داده ها

فشرده سازی داده ها فرایند کاهش تعداد بیت های مورد نیاز برای نمایش رسانه است [۱]. این بدان معناست که یک فایل فشرده کوچکتر از فایل اصلی و می تواند به عنوان مثال در مدت زمان کوتاه تری از طریق اینترنت منتقل شود. دو نوع تکنیک فشرده سازی داده ها وجود دارد: اتلافی و بدون اتلاف. فشرده سازی بدون اتلاف به این معنی است که می توان تمام داده های هنگام فشرده سازی را، بازیابی کرد. این بدان معناست که پس از بازیابی فایل فشرده شده با فایل اولیه کاملا یکسان خواهد بود و بدون اتلاف می باشد. از طرف دیگر، فشرده سازی از دست رفته (اتلافی)، برخی از داده ها را در حین فرایند فشرده سازی حذف می کند [۳]. هنگامی که فایل از حالت فشرده خارج می شود، داده های از دست رفته قابل بازیابی نیستند. فشرده سازی اتلافی بیشتر در مورد داده های دودویی استفاده می شود. فشرده سازی تلفاتی برای داده های دودویی [۳] در ترکیب با فشرده سازی بدون اتلاف [۱] به کار می رود. به عنوان مثال فایل های صوتی، تصویری و ویدئویی [۱]. مزیت فشرده سازی تلفاتی این است که معمولا نسبت فشرده سازی بسیار بالاتری نسبت به فشرده سازی بدون تلفات به دست دارد، در حالی که کاربران هنوز می توانند اطلاعات داده های فشرده را درک کنند [۳]. اغلب، داده های از دست رفته در فشرده سازی برای کاربر قابل توجه نیست [۳۱]. کدگذاری هافمن و کدگذاری دیکشنری نمونه هایی از فشرده سازی بدون اتلاف هستند. این الگوریتم ها در بسیاری از برنامه های فشرده سازی مانند Gzip، Zip یا Winrar استفاده می شوند، آنچه برای این الگوریتم ها متداول است این است که آنها بالاترین نسبت فشرده سازی را از فایل هایی که داده ها را تکرار می کنند یا توزیع ناهموار کاراکترها/نمادها را دارند، دریافت می کنند. این بدان معناست که داده های تصادفی یا یک متن "گیج کننده" ممکن است نسبت فشرده سازی کمی داشته باشد. نسبت فشرده سازی اندازه گیری میزان حجم فایل فشرده شده در مقایسه با فایل اصلی است[۳۲]. به عنوان مثال، این الگوریتم ها هنگام فشرده سازی متن نوشته شده به برخی از زبان ها مانند انگلیسی، آلمانی یا زبان برنامه نویسی موثر هستند؛ به این دلیل که برخی از کلمات، عبارات و کاراکترها بیشتراز بقیه استفاده می شوند [۳۲]. این الگوریتم ها فقط در متن استفاده نمی شوند. فرمت PNG از این تکنیک ها برای یافتن مطابقت و الگوهای موجود در تصاویر استفاده می کند [۳۳].

### 2.1کدگذاری دیکشنری

کدگذاری یک تکنیک فشرده سازی است که رشته های تکراری متن را با منابع جایگزین می کند. در حالی که کدگذاری هافمن به کاراکترهای فردی و میزان استفاده آن ها در متن نگاه می کند، کدگذاری دیکشنری از رشته های تکراری استفاده می کند [۲۳] به عنوان مثال، اگر متنی چندین بارحاوی کلمه "فشرده سازی" است، موارد تکراری کلمه را می توان با اشاره به کلمه در دیکشنری جایگزین کرد. هنگامی که متن از حالت فشرده خارج می شود، الگوریتم از مرجع برای جستجوی کلمه استفاده می کند و مرجع را با کلمه اصلی جایگزین می کند. دیکشنری لیستی از رشته های متنی است که اغلب مورد استفاده قرار می گیرد [۳۴]. برخی از رمزگذارهای دیکشنری، مانندLZ77 [۱۰] از یک دیکشنری صریح استفاده نمی کنند، اما در عوض از ارجاعات به انجام شده قبلی درمتن استفاده می کند [۳۵]. رمزگذاری های دیکشنری نیز می توانند در داده های غیر متنی مانند تصاویر استفاده شوند. فرمت تصویر PNG ازکدگذاری دیکشنری LZ77 استفاده می کند [۳۳]. در مورد تصاویر، الگوریتم باید دنباله های پیکسلی را به جای رشته های متنی تکرار کند.

مزیت استفاده از کدگذار دیکشنری این است که می تواند نسبت فشرده سازی بسیار بالایی را برای فایل های خاص به دست آورد. برنامه نویسان دیکشنری به ویژه بر روی داده های تکراری، مانند متن نوشتاری موثر هستند [۲۳]. متن، که به زبان نوشته شده است، معمولا برخی از کلمات یا ترکیب حروف را بیشتر از بقیه تکرار می کند که برنامه نویسان دیکشنری از این مزیت استفاده می کنند. عیب استفاده از برخی کدگذارهای دیکشنری، مانندLZW ، این است که فایل فشرده ممکن است بزرگتر از فایل فشرده نشده باشد [۳۶و۳۷]. این اتفاق در صورتی می افتد که داده هایی که باید فشرده شوند هیچ اطلاعات تکراری نداشته باشند. اگر فشرده سازی یک فایل باعث کاهش حجم آن نشود، پس هدف شکست خورده است.

### 2.1.1 الگوریتم های رایج کدگذاری دیکشنری

این بخش ویژگی ها و تفاوت های اصلی بین الگوریتم های کدگذاری دیکشنری را توصیف می کند. ویژگی های یک الگوریتم می تواند راه حل فنی آن یا چیزی باشد که آن را از بقیه متمایز کند. یک الگوریتم مچنین می تواند به این معنا باشد که کمابیش برای کارهای فشرده سازی مناسب است.

*LZ77*

LZ77 یک کدگذار دیکشنری محبوب است که دربین سایر موارد ،در فرمت تصویر PNG [۳۳] و الگوریتم DEFLATE استفاده می شود [۱۱]. الگوریتم LZ77 هنگام فشرده سازی داده ها از پنجره *لغزان* [۱۰] استفاده می کند (به عنوان مثال یک فایل تصویری یا یک فایل متنی). sliding window یک بافر است که میزان متن (یا داده) را که الگوریتم در هر نقطه تجزیه و تحلیل می کند تا زیر رشته های منطبق را پیدا کند را تعیین می کند. پنجره لغزان شامل یک بافر *پیش بینی* و یک بافر *جستجو است*. بافر پیش بینی، متنی را که هنوز فشرده یا کدگذاری نشده است، تجزیه و تحلیل می کند، به عنوان مثال 20 کاراکتر بعدی، در حالی که بافر جستجو متنی است که اخیراً کدگذاری شده است و معمولاً بسیار بزرگتر از بافر پیش بینی است. اگر یک زیر رشته در بافر جستجو باشد، نیز در بافر پیش رو ظاهر شود، سپس زیررشته در بافر پیش رو با اشاره به بافر قبلی جایگزین می شود [۱۰].

*LZ78*

در حالی که الگوریتم LZ77 با اشاره به یک نقطه قبلی در متن، زیررشته های مکرر ایجاد می کند، LZ78 از یک دیکشنری صریح استفاده می کند .الگوریتم LZ78 یک دیکشنری از زیررشته ها ایجاد می کند و زیر رشته ها را در یک متن با اشاره به دیکشنری جایگزین می کند [۳۸]. الگوریتم LZ78 تجدید نظر در الگوریتم LZ77 است، و به جز دیکشنری صریح در بقیه موارد مشابه اند [۳۹].

*LZW*

الگوریتم LZW یک نوع LZ78 است و همچنین از یک دیکشنری صریح استفاده می کند[۳۵]. فرض اساسی LZW این است که فرایند کدگذاری را با ۲۵۶ ورودی ها/کاراکتر در دیکشنری شروع می کند[۳۷]. این معمولاً مجموعه کاراکتر اسکی است، که از یک بایت در هر کاراکتر استفاده می کند. این الگوریتم دائما زیررشته هایی را به دیکشنری با مرجع بالاتر از ۲۵۶ اضافه می کند. این این ورودی ها ممکن است در متن تکرار شوند یا اینکه تکرار نشوند. اگر یک زیر رشته تکرار شود، از همان اشاره به زیر رشته استفاده می شود [۳۵].

Re-pair

الگوریتم Re-pair از یک دیکشنری صریح استفاده می کند و شبیه الگوریتم LZ78 است. تفاوت Re-pair با LZ78 این است که-Re pair فقط زیرشته های دو رشته ای را ذخیره می کند [۳۸]. Re-pairهمه جفت های کاراکتر را که بیش از یک بار در یک متن وجود دارد، می یابد و با اشاره به دیکشنری آنها را جایگزین می کند. این امر تا زمانی ادامه می یابد که هیچ جفت کاراکتری که بیش از یک بار در متن ظاهر شود، وجود نداشته باشد [۴۰].

### 2.2 فشرده سازی ASCII

فشرده سازی ASCII در این پایان نامه به عنوان الگوریتم های فشرده سازی تعریف شده است. به ویژه بر روی موثر است. کدگذاری اسکی دارای تغییرات زیادی برای زبانهای مختلف است، اما این پایان نامه بر US-ASCII تمرکز می کند، مگر این که خلاف آن بیان شود. برای ارتباطات اینترنت US-ASCII به کدگذاری اسکی ارجحیت دارد. امروزه US-ASCII با ASCII کاربرد یکسانی دارد [۴۴و۴۵]. رمزگذاری ASCIIیک کدگذاری متنی است که به طور گسترده برای متنی که کاراکترها و نمادهای خیلی متنوع ندارد، استفاده می شود [۴۵و۴۶]. این محدودیت در تعداد

نمادها از نحوه ذخیره متن بر روی کامپیوتر نشأت می گیرد. یک نماد دقیقاً یک بایت (۸ بیت) برای ذخیره سازی استفاده می کند، که در آن ۷ بیت کد نماد و 1 بیت "مهمترین بیت" است. هدف از مهمترین بیت می تواند به عنوان یک بررسی برابری، برای تشخیص خطاها در داده ها استفاده شود [۴۷]. همچنین گروهی از رمزگذاری ها معروف به "اسکی توسعه یافته" از ۸ بیت برای نماد استفاده می کنند و در نتیجه مهمترین بیت را کنار می گذارند. رمزگذاری هایی که از همه ۸ بیت استفاده می کنند، به جای ۱۲۸ نماد استاندارد از ۲۵۶ نماد استفاده می کنند [۴۸].

### Shoco 2.2.1

Shoco یک الگوریتم فشرده سازی متن است که توسط Christian Schramm توسعه یافته است [۱۶]، که به عنوان "کمپرسور سریع برای رشته های کوتاه" توصیف می شود و به ویژه بر روی متن اسکی موثر است. راه حل فشرده سازی شوکو از این واقعیت استفاده می کند که در هر زبان، برخی از کاراکترها بیشتر از بقیه استفاده می شود .همانطور که قبلا
ذکر شد، اولین بیت در یک کاراکتر اسکی اضافی است، مگر اینکه برای اهدافی مانند تشخیص خطا استفاده شود. بنابراین الگوریتم از اولین بیت در یک بایت استفاده می کند تا نشان دهد که آیا بیت های زیر به یک کاراکتر مشترک اشاره می کنند یا خیر. اگر بیت اول روی ۰ تنظیم شود، بدین معنی که ۷ بیت زیر نشان دهنده یک کاراکتر غیر معمول است؛ پس از فشرده شدن کاراکتر اسکی تغییر نخواهد کرد. اگر بیت اول روی ۱ تنظیم شود، بدین معنی که بیت های زیر دو کاراکتر مشترک را نشان می دهند. سپس دو کاراکتر با ۳ یا ۴ بیت نشان داده می شوند. همه با هم از ۸ بیت استفاده می کنند که به عنوان یک بایت ذخیره می شوند.
Shoco همچنین می تواند کاراکترهای غیراسکی را فشرده کند، اما هزینه دارد. "اگر رشته ورودی شما به طور کامل (یا بیشتر) اسکی نباشد، خروجی ممکن است افزایش یابد" [۱۵].
وقتی الگوریتم با کاراکتری مواجه می شود که بیش از ۷ بیت استفاده می کند (مانند یک کاراکتر UTF-8) استفاده می کند، درست قبل از کاراکتر "نشانگر" را وارد می کند. نشانگر یک کاراکتر خاص است که یک بایت را اشغال می کند. هدف آن این است که نشان دهد که کاراکتر بعدی اسکی نیست و بنابراین بیش از ۷ بیت استفاده خواهد کرد. نشانگر همچنین می گوید که کاراکتر بعدی از چند بایت استفاده می کند (به عنوان مثال ۱ یا ۲ بایت) [۱۵].
تکنیکی که شوکو از آن استفاده می کند شمارش فرکانس بیگرام در یک متن است. بیگرامز، در زمینه شوکو، جفت های منحصر به فرد از دو کاراکتر متوالی در یک متن است. یک بیگرام رایج در انگلیسی "qu" است، زیرا بعد از "q" تقریباً همیشه "u" می آید. بعد از اینکه شوکو لیستی از متداول ترین کاراکترهای یک متن تهیه کرد، لیستی دیگر از کاراکترهایی که به احتمال زیاد به دنبال کاراکترهای رایج می آیند، تهیه میکند. اگر شوکو متوجه شد که به عنوان مثال "he" یک بیگرام رایج است، آنگاه می تواند تمام کلمات حاوی این بیگرام، مانند "the"، "she" و "then" فشرده کند [۱۵].
مزیت استفاده از Shoco این است که تا زمانی که ورودی صددرصد اسکی باشد، فایل فشرده هرگز بزرگتر از فیل فشرده نشده نخواهد بود. علاوه بر این، Shoco را می توان به راحتی در ترکیب با سایر الگوریتم های فشرده سازی استفاده کرد،
مانند الگوریتم های کدگذاری دیکشنری، یا کدگذاری هافمن برای دستیابی به فشرده سازی بیشتر. همچنین یک الگوریتم Shoco بسیار سریع است. به گزارش وب سایت [۱۵]، Shoco هنگام فشرده سازی یک فایل ۴,۹ مگابایتی تقریبا ۷ برابر سریعتر از Gzip است. نقطه ضعف Shocoاین است که استفاده از Shoco به تنهایی نسبت فشرده سازی بسیار بدتری از برنامه های فشرده سازی استاندارد دارد. علاوه بر این، کاربر باید بداند که چه نوع متنی را فشرده می کند، اگر
متن شامل مقدار زیادی از کاراکترهای چند بایتی، مانند UTF-8 باشد، ممکن است فایل فشرده بزرگتر از فایل فشرده نشده [۱۵] باشد، که هنگام استفاده از ابزار فشرده سازی بسیار نامطلوب است.

## 2.3 کدگذاری هافمن

برنامه نویسی هافمن به گونه ای طراحی شده است که پرکاربردترین کاراکترها را در یک متن تا حد امکان بیت، نشان دهد. اول، الگوریتم ورودی را با پیمایش متن کاراکتر به کاراکتر، تجزیه و تحلیل می کند. سپس الگوریتم لیستی از هر کاراکتر که در متن استفاده می شود، ایجاد کرده و لیست را بر اساس تکرارها مرتب می کند [۲]. پس از ایجاد لیست، الگوریتم یک درخت دودویی [۴۹] بر اساس جدول می سازد. درخت دوتایی یک ساختار داده است که از "گره" و "شاخه" تشکیل شده است. هر گره می تواند حداکثر دو "گره فرزند" داشته باشد. در شکل ۱، گره ها مستطیل های آبی رنگ هستند. "گره های برگ" آخرین گره های یک شاخه هستند. یک درخت دودویی را به عنوان یک درخت در نظر بگیرید که وارونه است. همه شاخه های یک درخت از یک ریشه شروع می شوند، و برگها در انتهای شاخه ها هستند. هر گره برگ ارزش یک کاراکتر مورد استفاده در متن را دارد [۲].
هنگامی که یک درخت هافمن ساخته می شود، فشرده سازی متن شروع می شود. این الگوریتم درخت را برای هر کاراکتر در یک متن پیمایش می کند. برای رسیدن به یک گره برگ، الگوریتم به سمت گره فرزند چپ یا راست حرکت می کند تا کاراکتر مشخص شده، پیدا می شود. این پیمایش به صورت دوتایی با یک و صفر نشان داده شده است، جایی که ۱ است، به معنای "رفتن به راست" و ۰ به معنای "رفتن به سمت چپ"

است. به منظور فشرده سازی یک متن برای بازیابی داده های اصلی، درخت هافمن نیز باید ذخیره شود.به همین دلیل است که کدگذاری هافمن در متون بزرگتر موثرتر است. به عنوان مثال وقتی یک متن بزرگ وجود دارد که فقط شامل ۲۶ حرف الفبای انگلیسی و چند علامت نقطه گذاری است، درخت صرف نظر از اندازه متن، بیش از ۲۶ گره برگ دارد [۲].

کدگذاری هافمن یک رمزگذار آنتروپی است، به این معنی که داده ها را بر اساس تکرار نماد، فشرده می کند؛ که این برخلاف کدگذاری دیکشنری است که به تکرارهای زیر رشته ها یا توالی نمادها نگاه می کند [۵۰]. برنامه نویسان آنتروپی مانند کدگذاری هافمن هنگامی سودمند است که از داده های خاص بیشتر از بقیه نمادها استفاده شود. کدگذاری هافمن می تواند به آسانی هم در داده متنی و هم در داده های دودویی مورد استفاده قرار گیرد و می تواند در ترکیب با سایر الگوریتم های فشرده سازی مانند کدگذاری دیکشنری استفاده شود [۱۱و۵۱]. کدگذاری هافمن اغلب با کدگذاری حسابی مقایسه می شود، که نوع دیگری از رمزگذار آنتروپی است. نتایج چنین مقایسه هایی نشان می دهد که کدگذاری هافمن سریع تر از کدگذاری حسابی است، اما کدگذاری حسابی به طور کلی نسبت فشرده سازی بهتری دارد [۵۲و۵۳].

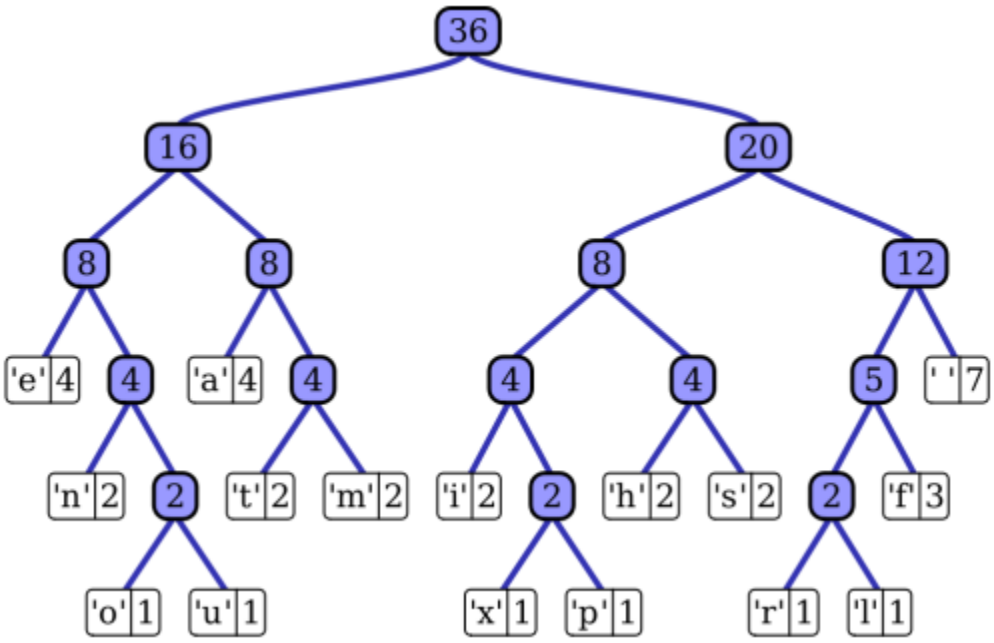
### 2.3.1 فرایند فشرده سازی

این بخش فرایند فشرده سازی برای کدگذاری هافمن را توضیح می دهد. یک مثال ورودی برای الگوریتم می تواند "this is an example of a huffman tree " جدول ۱ فراوانی از هر کاراکتر در جمله مثال است.

| کاراکتر | فراوانی |
|---|---|
| Space | 7 |
| a | 4 |
| e | 4 |
| f | 3 |
| h | 2 |
| i | 2 |
| m | 2 |
| n | 2 |
| s | 2 |
| t | 2 |
| l | 1 |
| o | 1 |
| p | 1 |
| r | 1 |
| u | 1 |
| x | 1 |

*جدول ۱. مثال فراوانی کاراکتر*

از این جدول می بینیم که کاراکتر space بیشترین فراوانی را دارد، پس از آن "a" و"e" با چهار مورد. شکل ۱ نشان می دهد که چگونه جمله مثال شبیه درخت دودویی هافمن است.




*شکل ۱. درخت دودویی هافمن. منبع [۵۴]*

گره های این شکل همه دارای اعدادی هستند که در گره های فرزند نشان می دهد چند باراز هر کاراکتر استفاده شده است. این عدد هنگام ساختن درختان هافمن مهم است زیرا کاراکترهایی که بیشتر تکرار می شوند باید تا حد ممکن بالای درخت باشند (نزدیک به گره ریشه). جدول ۱ نشان می دهد که فروانی (تکرار) کاراکتر "a" و "e" بیشترین هستند و بنابراین به گره ریشه نزدیک هستند. این سه کاراکتر را می توان پس از سه عبور از گره ریشه به دست آورد.

جمله مثال "this is an example of a huffman tree"، که با کدگذاری اسکی ذخیره می شود، از ۳۶ بایت ذخیره سازی استفاده می کند. این اندازه فایل قبل از فشرده سازی است، جایی که هر کاراکتر از یک بایت (یا ۸ بیت) ذخیره سازی استفاده می کند. هنگامی که فشرده سازی شروع می شود، با اولین کاراکتر "t" شروع می شود، برای رسیدن به گره برگ که مقدار "t" را از گره ریشه نگه می دارد، الگوریتم ابتدا به گره فرزند چپ، سپس دو بار به گره فرزند راست و در نهایت به چپ برود. بنابراین نمایش "t" فشرده دودویی (باینری) "0110" خواهد بود، که فقط ۴ بیت است. الگوریتم این کار را برای هر کاراکتر در متن انجام می دهد و پس از اتمام، با یک جریان دودویی ۱۳۵ بیت به پایان می رسد. این جریان فقط از ۱۷ بایت حافظه در کامپیوتر استفاده می کند که کمتر از نیمی از متن فشرده نشده است. البته، درخت هافمن نیز باید برای فشرده سازی متن در آن ذخیره شود.

### 2.4 کارهای مرتبط

این بخش مقاله ها و راه حل های مربوط به فشرده سازی متن یا به طور خاص، فشرده سازی اسکی، را مورد بحث قرار می دهد، زیرا راه حل ارائه شده در این پایان نامه یک فشرده ساز متن است که برای فشرده سازی متن اسکی سفارشی شده است.

#### 2.4.1 تقویت فشرده سازی متن با رمزگذاری آماری مبتنی بر کلمه

این مقاله از نسخه ۲۰۱۲ مجله کامپیوتر[۵۵] است که توسط آنتونیو فاریا، گونزالو ناوارو و خوزه آر پاراما [۱۳] نوشته شده است. این مقاله پیشرفت احتمالی در برنامه های فشرده سازی را ارائه می دهد. هدف از راه حل پیشنهادی این است که نسبت فشرده سازی متن را افزایش داده و زمان فشرده سازی را برای فشرده سازهای متن کاهش دهد. این راه حل از تکنیک های فشرده سازی مبتنی بر کلمه و بایت جهت پیش پردازش برنامه های فشرده سازی عمومی استفاده می کند [۱۳].

فشرده سازی مبتنی بر کلمه نوعی کدگذاری دیکشنری است که در آن الگوریتم کل کلمات را جستجو می کند [۵۶]. اکثر برنامه نویسان دیکشنری، مانند الگوریتم های Lempel-Ziv، زیر رشته های مکرر را جستجو می کنند [۱۰] [۳۸]. زیر رشته ها می توانند هر طولی داشته باشند و می توانند بخشی از یک کلمه باشند. فشرده سازی مبتنی بر کلمه فقط کلمات کامل را به دیکشنری خود اضافه می کند. این مقاله بیان می کند که استفاده از فشرده سازی مبتنی بر کلمه، بر خلاف کدگذارهای استاندارد دیکشنری، زمان فشرده سازی و فشرده سازی بهتری را ارائه می دهد. علاوه بر این، امکان جستجوی کلمات و عبارات روی فایل فشرده بدون نیاز به فشرده سازی آن وجود دارد [۱۳]. فشرده سازی بایت گرا دومین پیش پردازشی است که راه حل پیشنهادی از آن استفاده می کند. فشرده سازی بایت گرا تکنیکی است که هنگام نوشتن کدهای فشرده سازی فقط از بایت های کامل استفاده می کند و نه از بیت ها [۵۷]. برعکس، کدگذاری استاندارد هافمن یک تکنیک فشرده سازی است که از توالی بیت ها به عنوان کد برای حرکت در درخت هافمن استفاده می کند [۲]. بنابراین کدگذاری هافمن باید به عنوان دنباله ای از بیت ها خوانده شود تا بایت. که می تواند باعث افزایش تاخیر شود، زیرا داده ها به صورت بایت در کامپیوتر ذخیره و خوانده می شوند. راه حل پیشنهادی از یک الگوریتم بایت گرا به نام Tagged Huffman استفاده می کند [۵۸]. برچسب گذاری هافمن (Tagged Huffman) از تکنیک فشرده سازی مشابه کدگذاری استاندارد هافمن استفاده می کند، اما هر کد برای جستجوی درخت هافمن از تعداد مشخصی بایت استفاده می کند. بر اساس این مقاله، استفاده از Tagged Huffman به جای کدگذاری استاندارد هافمن، تاخیر را کاهش می دهد، اما نسبت فشرده سازی را در حدود ۰/۶ کاهش می دهد، که آنها به عنوان یک معامله پذیرفتند [۱۳].

نتایج آزمایشات نشان می دهد که یک برنامه فشرده سازی در فشرده سازی یک فایل از پیش پردازش شده، ۵ برابر سریعتر از فایل اصلی است. با این حال، زمان فشرده سازی پیش پردازنده را شامل نمی شود. مقایسه ها نشان می دهد استفاده از پیش پردازنده قبل از یک فشرده ساز عمومی (از جمله Gzip, Bzip2, 7-Zip, Re-pair)، فایل های فشرده نهایی بین ۵,۵ تا ۱۰ درصد کوچکتر از فایل فشرده شده با فشرده ساز عمومی بوده اند [ ۱۳].

*مقایسه با TCS*

فاریا (Fariña) و همکاران: راه حل (FS) و TCS هر دو فشرده ساز متنی هستند که نسبت فشرده سازی بالاتری را برای متن های نوشته شده به برخی از زبان ها بدست می آورد. روش فشرده سازی مبتنی بر کلمه به فشرده سازی FS می انجامد زیرا در زبان ها بیشتر از کلمات خاص استفاده می شود تا کلمات دیگر. همچنین TCS از ویژگی های زبان ها برای افزایش نسبت فشرده سازی استفاده می کند؛ ماژول فشرده سازی اسکی (ASCII compression module (ACM)) از یک مدل فشرده سازی (توضیح داده شده در بخش 4.4.1) بر اساس فراوانی حروف در زبان انگلیسی استفاده می کند، و ماژول کدگذاری دیکشنری زمانی فشرده سازی می کند که یک متن دارای ترکیبی از کاراکترها باشد. هدف FS افزایش نسبت فشرده سازی و کاهش تأخیر در برنامه های فشرده سازی است. هر دو روش مبتنی بر کلمه و بایت جهت کاهش زمان فشرده سازی است [۱۳]. با این حال، TCS فقط نسبت فشرده سازی را در اولویت قرار می دهد، نه سرعت را.

### 2.4.2 فشرده سازی آنلاین فایل های اسکی

این مقاله از کنفرانس بین المللی ۲۰۰۴ در زمینه فناوری اطلاعات: کدگذاری و محاسبه [۵۹] است که توسط جان ایستل، پاملا مندلباوم و اما رجنتوا [۶۰] نوشته شده است. در واقع راه حل پیشنهادی این است که در یک متن انگلیسی کدهای کوتاه تری به بیگرام های مکرر بدهید و در نتیجه به فشرده سازی برسید. این مقاله ادعا می کند که فشرده سازی باید رخ دهد زیرا ترکیبات خاصی از حروف بیشتر به زبان انگلیسی هستند نسبت به سایر زبان ها. این راه حل فقط بر روی متون دارای کد اسکی کار می کند و نه هیچ کدگذاری دیگری [۶۰]. این الگوریتم با استفاده از ۲۸ دیکشنری استاتیک کار می کند. دیکشنری های استاتیک به این معنی است که با تجزیه و تحلیل یک متن خاص، برای یک مورد استفاده، ایجاد نشده اند، بلکه مجموعه ای از واژه نامه های عمومی هستند که باید فشرده شوند – مخصوصاً برای متون انگلیسی. یکی از دیکشنری ها برای اعداد، دیگری برای علائم نگارشی و کاراکترهای خاص و ۲۶ دیکشنری باقی مانده برای هر حرف الفبا استفاده می شود. اگر متنی که فشرده می شود حاوی حرف "a" باشد، الگوریتم، دیکشنری آن حرف خاص را جستجو می کند و پیدا می کند که حرف بعدی در چه شاخصی قرار دارد. به عنوان مثال، دیکشنری حرف "a" حرف "t" را به عنوان اولین فهرست دارد، زیرا "t" حرفی است که بیشتر از به دنبال "a" می آید. اگر متن حاوی حرف "t" بود با شماره ۱ جایگزین می شد [۶۰].

برای هر کلمه جدید در یک متن، الگوریتم با جستجوی جایی که اولین حرف یا نماد در "دیکشنری پیش فرض" قرار دارد شروع می شود. دیکشنری پیش فرض بعد از علائم نگارشی یا کاراکترهای خاص استفاده می شود. اگر کلمه فشرده شود "hat" باشد، "h" در دیکشنری پیش فرض فهرست ۱۳ خواهد بود، "a" در دیکشنری حرف "h" فهرست ۲ خواهد بود و حرف "t" در دیکشنری "a" فهرست ۱ خواهد بود. سپس اعداد ۱۳، ۲ و ۱ به دنباله های بیتی ترجمه می شوند [۶۰]. دنباله های بیتی کوتاهتر برای هر کاراکتر کمتر از یک بایت استفاده می کنند، در مقابل کاراکترهای فشرده نشده اسکی که از بایت برای هر کاراکتر استفاده می کنند. جایگزینی کاراکترهای اسکی با دنباله های بیتی باعث فشرده سازی می شود.

آزمایش های انجام شده در مقاله نتایج حاصل از راه حل پیشنهادی را با برنامه فشرده سازی Winzip مبتنی بر DEFLATE مقایسه کرده است [۶۱]. یک فایل "کمتر از ۱۰۰ کیلوبایت " و " تا ۱ مگابایت " برای مقایسه استفاده شد [۶۰]. راه حل پیشنهادی نسبت فشرده سازی ۱,۶۷ را برای هر دو فایل بدست آورد، در حالی که WinZip نسبت فایل بزرگتر را ۲,۵ و نسبت فایل کوچکتر را کمتر از ۱ بدست آورد. از این نتایج، آنها به این نتیجه رسیدند که "تکنیک دیکشنری چندگانه در فشرده سازی فایل ها در هر اندازه ای سازگار است" [۶۰].

*مقایسه با ماژول فشرده سازی ASCII*

شباهت های زیادی بین روش دیکشنری چندگانه و ACM از TCS وجود دارد. هر دو فقط می تواند فایل های متنی را فشرده کنند، هر دو برای متن اسکی در نظر گرفته شده اند، و هر دو از دیکشنری های استاتیک که برای متون انگلیسی سفارشی شده اند استفاده می کنند. راه حل دیکشنری چندگانه از تکنیکی همتراز با کدگذاری هافمن استفاده می کند. کدگذاری هافمن کدهای کوتاه تری به کاراکترهای مکرر می دهد. راه حل دیکشنری چندگانه، کدهای کوتاه تری را به بیگرام های تکراری می دهد، ACM کاراکترها را به کدها تبدیل نمی کند، بلکه از تکنیکی به نام ادغام استفاده می کند که در بخش 4.4.1 توضیح داده می شود. تفاوت دیگر این است که ACM می تواند متن کدگذاری های مختلف را فشرده کند، در حالی که راه حل دیکشنری چندگانه نمی تواند. در مقاله آنها این را به عنوان یک کار آینده بیان کردند [۶۰].

## 3 روش ها و متدولوژی ها

این فصل روش ها و متدولوژی های مختلف را تشریح و مقایسه می کند و توضیح می دهد که کدام روش ها برای این پروژه انتخاب شده اند و چرا. این فصل روش های مختلف دانشگاهی را که برای پروژه های تحقیقاتی استفاده می شود و همچنین روش ها و مدلهایی را که بطور خاص برای توسعه نرم افزار استفاده می شود، ارائه می دهد.

### 3.1 روش های تحقیق

چندین روش تحقیق وجود دارد که می تواند برای پروژه های تحقیقاتی مورد استفاده قرار گیرد. روش های تحقیق روش هایی برای انجام وظایف تحقیق هستند و نحوه انجام تحقیق را توضیح می دهند. روش های رایج تحقیق شامل تجربی، توصیفی و بنیادی است. رویکرد تجربی علل، آثار و متغیرها را بررسی می کند. آزمایشات انجام شده و نتایج تجزیه و تحلیل می شوند. روش تحقیق توصیفی ویژگی های یک موقعیت را توصیف می کند، اما علل آن را توضیح نمی دهد. روش توصیفی اغلب از نظرسنجی یا مطالعات موردی استفاده می کند. روش تحقیق بنیادی ابتکاری است و ایده ها، اصول و نظریه های جدیدی را ایجاد می کند. تحقیقات بنیادی بر اساس کنجکاوی انجام می شود و در مورد مشاهده یک پدیده است [۶۲]. این پروژه از روش تحقیق تجربی استفاده می کند، زیرا آزمایش بخش مهمی از پروژه است. آزمایش روابط بین متغیرها و نحوه تغییر متغیرها بر نتایج را بررسی می کند.

### 3.2 استراتژی تحقیق

استراتژی تحقیق، رویکرد مشخص تری برای نحوه انجام تحقیق است، در حالی که روش تحقیق چارچوبی برای تحقیق است. یک استراتژی تحقیق یک راهنما یا یک روش است. یک استراتژی تحقیق می تواند تجربی باشد، درست مانند روش تحقیق تجربی. رویکرد تجربی با هدف کنترل همه عواملی که ممکن است بر نتیجه یک آزمایش تأثیر بگذارد، کنترل می شود. این استراتژی بر اساس نتایج یک آزمایش، یک فرضیه را تأیید یا جعل می کند. یکی دیگر از استراتژی های رایج تحقیق استفاده از نظرسنجی یا پرسشنامه است. این استراتژی شامل جمع آوری اطلاعات از افراد است. این نظرسنجی ها می تواند عمیق باشد و ممکن است فقط شامل چند نفر باشد که روش کیفی است یا می توان نظرسنجی ها را برای تجزیه و تحلیل داده های تعداد بیشتری از افراد طراحی کرد که روش کمی است [۶۲]. این پروژه به همان دلیلی که از روش تحقیق تجربی استفاده می کند، از استراتژی تحقیق تجربی استفاده می کند.

### 3.3 جمع آوری داده ها

از روش جمع آوری داده ها برای جمع آوری داده ها برای تحقیق استفاده می شود. این پایان نامه از تحقیقات کمی استفاده می کند و بنابراین، مناسب ترین روش های جمع آوری داده ها برای پروژه های تحقیقاتی کمی، آزمایش، پرسشنامه، مطالعات موردی و مشاهدات است. آزمایش ها مجموعه داده بزرگی را که برای متغیرها استفاده می شود جمع آوری می کند. پرسشنامه ها داده ها را از طریق سوالات کمی یا کیفی جمع آوری می کند، مطالعات موردی تجزیه و تحلیل عمیق یک یا چند شرکت کننده است. مشاهدات رفتار را با تمرکز بر موقعیت ها و فرهنگ بررسی می کند [۶۲]. این پروژه از آزمایشات برای جمع آوری داده ها استفاده می کند، همانطور که از استراتژی تحقیق تجربی استفاده می کند.

### 3.4 تجزیه و تحلیل داده ها

برای تحقیقات کمی، متداول ترین روش های تجزیه و تحلیل داده ها آمار و ریاضیات محاسباتی است. آمار استنباطی است و شامل نتایج تجزیه و تحلیل برای نمونه های آماری و همچنین ارزیابی اهمیت نتایج است. ریاضیات محاسباتی شامل محاسبه روش های عددی، مدل سازی و استفاده از الگوریتم ها است [۶۲]. ریاضیات محاسباتی از کد کامپیوتری برای تجزیه و تحلیل داده ها استفاده می کند، بر خلاف آمار، که توسط افراد قابل تجزیه و تحلیل است. نتایج حاصل از آزمایشات این پایان نامه داده های عددی را ارائه می دهد. تجزیه و تحلیل این داده ها شامل مقایسه داده ها است. ریاضیات محاسباتی برای تجزیه و تحلیل داده ها مورد نیاز نیست. بنابراین، آمار روش تجزیه و تحلیل داده ها برای این پروژه خواهد بود.

### 3.5 تضمین کیفیت

تضمین کیفیت اعتبار و تأیید تحقیق است. داده ها باید قابل اعتماد، معتبر، قابل تکرار و اخلاقی باشند تا در تحقیق مورد استفاده قرار گیرند. پایایی به ثبات و ثبات اندازه گیری ها اشاره دارد. این بدان معناست که هنگام در نظر گرفتن همه متغیرهای آزمایش، اندازه گیری های مختلف باید نتایج قابل قبولی را ارائه دهند. قابلیت اطمینان در هر آزمایشی که در این پروژه انجام می شود ارزیابی می شود.

تصدیق (روایی) اطمینان می دهد که ابزارهای یک آزمایش آنگونه که انتظار می رود اندازه گیری شود، انجام می دهند. اعتبار این پروژه با ارزیابی مراحل و ابزارهای مورد نیاز برای انجام آزمایشات حفظ می شود.

تکرارپذیری اطمینان حاصل می کند که یک آزمایش با تکرار دقیق متغیرها نتایج یکسانی را به همراه خواهد داشت. در این پروژه، همه آزمایش ها تکرار می شوند تا بررسی شود آیا نتایج یکسان هستند یا خیر.

تحقیق همچنین باید اخلاقی باشد. اخلاق اصول اخلاقی برنامه ریزی، انجام و گزارش نتایج حاصل از تحقیقات و همچنین حفظ حریم خصوصی و برخورد محرمانه با مطالب را پوشش می دهد [۶۲]. این جنبه های اخلاقی در تحقیق این پروژه ارزیابی می شود.

### 3.6 روش های توسعه نرم افزار

یک روش توسعه نرم افزار یک برنامه (یا مجموعه ای از دستورالعمل ها) برای توسعه یک سیستم از ابتدا تا پیاده سازی است. مزایای استفاده از چنین روشی عبارتند از: این به درک چرخه حیات فرایند کمک می کند، یک رویکرد ساختاریافته برای توسعه را اجرا می کند و برنامه ریزی منابع را که در پروژه مورد استفاده قرار می گیرد، امکان پذیر می سازد [۶۳].

مدل آبشار از فازهای متوالی و خطی استفاده می کند، که در آن هر مرحله ادامه فاز قبلی است. مراحل در مدل آبشار عبارتند از: امکان سنجی (درک مشکل)، الزامات و مشخصات، طراحی راه حل، کدگذاری و تست ماژول، آزمایش سیستم، تحویل و نگهداری [۶۳].

روش نمونه اولیه، نسخه های ناقص از یک محصول را که نمونه اولیه نامیده می شود، در حین توسعه ایجاد می کند. مزیت استفاده از نمونه های اولیه این است که توسعه دهندگان می توانند بازخورد کاربران را در حین توسعه محصول دریافت کنند و تغییرات احتمالی که باید انجام شود به راحتی انجام می شود. نمونه های اولیه همچنین می توانند توسعه دهندگان را درمورد قطعات ناتمام قسمت بهتری به توسعه دهندگان ارائه دهند، که می تواند برآورد بهتری در مورد زمان به پایان رساندن محصول ارائه دهد [۶۴].

مدل چابک (سریع) یک رویکرد سازگار و انعطاف پذیر است که بر تحویل زودهنگام و مداوم نرم افزار به مشتری تمرکز دارد. در حالی که مدل آبشار خطی و سازگار است، مدل چابک انعطاف پذیر و قابل تغییر است. مدل چابک ۱۲ اصل را تعریف می کند که "بیانیه" توسعه چابک را تشکیل می دهند [۶۵].

همه این روش ها معمولاً برای تیم توسعه دهندگان سازماندهی می شود که در آن محصول برای مشتری اجرا می شود. این پایان نامه یک پروژه انفرادی است و محصولی را برای مشتری ایجاد نمی کند، بنابراین پیروی از تمام اصول مرتبط با این روش های توسعه غیر ضروری و یا غیرممکن است. در عوض، ایده کلی پشت روش ها مورد توجه قرار می گیرد. روش نمونه اولیه نمونه هایی را ایجاد می کند که برای آزمایش در حین توسعه محصول در نظر گرفته شده است. این پروژه شامل سه ماژول است که به قدری ساده اند که ایجاد نمونه های اولیه ضروری نخواهد بود. در عوض نرم افزار در حین توسعه، آزمایش می شود. بنابراین این پروژه از روش نمونه اولیه استفاده نمی کند. انعطاف پذیری مدل چابک برای این پروژه مناسب تر از مدل آبشار است زیرا این پروژه فازهایی ندارد که به مراحل قبلی وابسته باشد تا تکمیل شود. بنابراین این پروژه از ایده کلی پشتیبان مدل چابک استفاده خواهد کرد.

## 4 طراحی و نیازمندی ها

این فصل الزامات کلی سیستم و نیز الزامات مجموعه داده و پیاده سازی های مورد استفاده در TCS را شرح می دهد. این فصل همچنین طراحی کلی TCS و راه حل فنی ماژول فشرده سازی اسکی (ACM) را توضیح می دهد.

### 4.1 الزامات سیستم

الزامات سیستم الزامات عملکردی و غیر عملکردی هستند که باید مورد استفاده قرار گیرند تا TCS قابل استفاده و قوی باشد. الزامات بر اساس موارد مورد استفاده برای TCS تعریف می شود. الزامات عملکردی خدماتی را که راه حل باید ارائه دهد و نحوه عملکرد راه حل در شرایط خاص را توصیف می کند. الزامات غیر کاربردی محدودیت هایی در خدمات ارائه شده توسط یک راه حل است. الزامات غیر کاربردی ممکن است بر عملکرد، قابلیت اطمینان و قابلیت استفاده متمرکز شوند [۶۶].

الزامات عملکردی TCS عبارتند از:

- **در طول فشرده سازی هیچ داده ای از بین نمی رود.**

TCS از فشرده سازی بدون اتلاف استفاده می کند، بدین معنی که داده های اصلی باید پس از فشرده سازی قابل بازیابی باشد. اگر داده ها در طول فشرده سازی از بین بروند، راه حل فایده ای ندارد.

- **TCS می تواند متن را با رمزگذاری های معمول فشرده کند.**

رمزگذاری های رایج به کدگذاری UTF-8 و ASCII اشاره می کند. TCS باید بتواند رایج ترین رمزگذاری های متنی را مدیریت کند تا یک راه حل مفید باشد. چهار فایل با رمزگذاری UTF-8 و یک فایل با رمزگذاری ASCII برای آزمایش این نیاز استفاده می شود.

- **TCS می تواند متن را با رمزگذاری های غیر معمول فشرده کند.**

در صورتی که TCS قادر به فشرده سازی متن با رمزگذاری هایی باشد که اغلب استفاده نمی شوند، علاوه بر رمزگذاری های رایج، این راه حل محکم است و می تواند انواع بیشتری از موارد استفاده را اداره کند. برای آزمایش این مورد از فایلی با کدگذاری cpo37 استفاده می شود.

- **ماژول های TCS به راحتی متصل می شوند.**

سه واحد ماژول TCS برنامه های مستقل هستند که می توانند در ترکیب با یکدیگر استفاده شوند. ماژول ها باید به طور جداگانه به هم متصل شوند تا به صورت جداگانه ارزیابی شوند.

الزامات غیر کاربردی برای TCS عبارتند از:

- **TCS می تواند فایل های بالای ۲۰ مگابایت را فشرده کند.**

کاهش حجم فایل بیشتر از فایل های کوچکتر روی فایلهای بزرگتر تأثیر می گذارد. برای ارزیابی این نیاز، یک فایل XML با حجم ۲۱,۶ مگابایت استفاده می شود.

- **TCS می تواند فایلها را حداقل به نصف اندازه اصلی خود فشرده کند.**

چندین تست روی هر یک از متون انجام می شود و اگر حداقل یکی از تستها دارای ضریب فشرده سازی حداقل ۲ باشد، شرایط مورد نیاز برآورده می شود.

- **تمام کدهای شخص ثالثی که در TCS استفاده می شود باید منبع باز باشد.**

کد شخص ثالثی که در TCS استفاده می شود باید دارای مجوز باز باشد که اجازه اصلاحات را می دهد. به منظور ادغام ماژول ها در یک راه حل، ممکن است تغییرات در کد لازم باشد.

### 4.2 ویژگی مجموعه داده ها

ویژگی مجموعه داده ها در این پایان نامه به عنوان مجموعه ای از متون مختلف که TCS هنگام ارزیابی قابلیت فشرده سازی از آنها استفاده می کند، تعریف شده است. این متون از رمزگذاری های مختلف استفاده می کنند، به عنوان مثال UTF-8 و ASCII، کدگذاری های معمولی هستند که ممکن است کاربر از آنها استفاده کند. UTF-8 یک رمزگذاری کاراکتر بسیار رایج است که شامل انواع زیادی از نمادها است [۶۷]. ASCII برای متنی استفاده می شود که از تنوع زیادی از کاراکترها و نمادها استفاده نمی کند و معمولاً به زبان انگلیسی نوشته می شود [۶۸]. از کدگذاری های مختلف برای ارزیابی متفاوت در فشرده سازی استفاده می شود و اینکه چه نوع کدگذاری برای راه حل بهینه است. متون از کدگذاری ASCII، cpo37 و UTF-8 استفاده خواهند کرد.

مجموعه داده همچنین باید نمایانگر نوع متنی باشد که راه حل می تواند برای آن استفاده شود. یک مورد کاربرد فشرده سازی معمولی می تواند یک کتاب یا یک دانشنامه باشد. این نوع متن معمولاً با UTF-8 یا کدگذاری دیگری که شامل انواع نمادها است کدگذاری می شود زیرا ASCII (یا سایر رمزگذاری های کاراکتر ۸/۷ بیتی) بسیار محدود است. مورد استفاده دیگر می تواند کد برنامه نویسی یا داده های متنی مانند XML [۶۹] یا JSON [۷۰] باشد. این نوع متن را می توان در ASCII کدگذاری کرد [۷۱].

#### 4.2.1 تعریف مجموعه داده ها

مجموعه داده های مورد استفاده در ارزیابی نمونه هایی از فایل های متنی بزرگتر هستند که کاربر ممکن است TCS را روی آنها اعمال کند. مجموعه داده ها شامل ۶ متن با ویژگی های مختلف است؛

- یک فایل XML ویکی پدیا. رمزگذاری UTF-8 ؛ ۲۱,۶ مگابایت [۷۲]
- یک فایل XML ویکی پدیا. کدگذاری cpo37 ؛ ۱۸,۱ مگابایت [۷۲]
- یک فایل کد نوشته شده با C. کدگذاری ASCII؛ ۶۴ کیلوبایت [۷۳]
- کتابی که به زبان انگلیسی نوشته شده است. رمزگذاری UTF-8 ؛ ۶۸۰ کیلوبایت [۷۴]
- کتابی که به زبان ایتالیایی نوشته شده است. رمزگذاری UTF-8 ؛ ۶۲۶ کیلوبایت [۷۵]
- کتابی که به زبان چینی نوشته شده است. رمزگذاری UTF-8 ؛ ۲۸۵ کیلوبایت [۷۵]

فایل XML ویکی پدیا یک نسخه محلی و فقط متنی از مجموعه ای از مقالات (صفحات جداگانه) از ویکی پدیا است. فایل XML بیشتر شامل کاراکترهای ASCII است، بدین معنا که اکثر کاراکترهای متن را می توان در اسکی کدگذاری کرد مجموعه داده ها نیز شامل همان فایل XML

است که با cpo37 رمزگذاری شده است. Cpo37 رمزگذاری یک فایل پس از فشرده سازی آن با ماژول فشرده سازی ASCII است. این متن برای تست وجود دو ماژول دیگر راه حل (کدگذاری هافمن و کدگذاری دیکشنری) با ACM سازگار است. فایل کد C از مخزن سیستم عامل لینوکس GitHub گرفته شده است و فقط شامل کاراکترهای ASCII است. این سه کتاب برای آزمایش وجود تفاوت بین زبان کد ساختار یافته و زبان طبیعی گنجانده شده است. کتاب انگلیسی بیشتر شامل کاراکترهای ASCII است، کتاب ایتالیایی نیز شامل بیشتر کاراکترهای ASCII است، اما از چندین کاراکتر استفاده می کند که در طرح کدگذاری ASCII گنجانده نشده است، و کتاب چینی شامل هیچ کاراکتر اسکی نمی شود. متون مختلف دارای اندازه های بسیار متنوعی هستند که برخی از آنها چند کیلوبایت بزرگ و برخی دیگر چند مگابایت هستند. صرف نظر از این، متن ها باید اندازه کافی برای ارزیابی صحیح TCS داشته باشند.

### 4.3 الزامات پیاده سازی

TCS شامل سه ماژول است. ACM، کدگذاری دیکشنری و کدگذاری هافمن. ACM به طور خاص برای این پروژه ایجاد شده است، اما کدگذاری دیکشنری و کدگذاری هافمن پیاده سازی هایی است که توسط افراد دیگر انجام شده و در راه حل یکپارچه شده است. پیاده سازی برنامه هایی است که دیگران برای پیاده سازی الگوریتم ها ساخته اند، پیاده سازی ها باید مجموعه ای از الزامات را برآورده کنند، اما همچنین باید بتوانند با سایر ماژول های TCS ادغام شوند تا راه حل به طور کلی بتواند الزامات سیستم را برآورده کند. چندین پیاده سازی برای الگوریتم های کدگذاری دیکشنری و کدگذاری هافمن یافت می شود. پیاده سازی ها باید شرایط زیر را داشته باشند:

- پیاده سازی ها باید دارای مجوز باز باشند [۷۷].
- پیاده سازی ها باید درست عمل کنند، حتی زمانی که داده های ورودی می توانند چند مگابایت باشند.
- پیاده سازی ها باید با متن کدگذاری های مختلف سازگار باشند.

در صورت اعمال تغییرات در کد، باید دارای مجوز باز باشند. به طور خاص، پیاده سازی ها نیاز به مجوز دارند که اجازه اصلاحات را می دهد. ممکن است اصلاحات در پیاده سازی برای ادغام ماژول ها در یک راه حل کاربردی ضروری باشد. پیاده سازی ها باید بتوانند فایل های با اندازه قابل توجه را فشرده کنند. بزرگترین فایلی که فشرده می شود ۲۱,۶ مگابایت است. انجام این کار برای TCS برای برآوردن نیاز سیستم ضروری است. " TCSمی تواند فایلهای بیش از ۲۰ مگابایت را فشرده کند".

پیاده سازی ها همچنین باید بتوانند با متن کدگذاری های مختلف کار کنند. مجموعه داده های مورد استفاده برای ارزیابی از ۷ بیتی ASCII، ۸ بیتی cpo37 و طول متغیر UTF-8 استفاده می کند. این مورد نیاز است تا راه حل بتواند با انواع متن کار کند. پیاده سازی ها ترجیحاً به زبان برنامه نویسی پایتون نوشته می شوند، همانطور که ACM در پایتون نوشته شده است. با این حال، این یک الزام نیست. هیچ الزامی برای پیاده سازی نسخه های پایتون وجود ندارد. پیاده سازی هایی با نسخه متفاوت از ACM می توانند تبدیل شوند تا ماژول ها بهتر ادغام شوند. پیاده سازی هایی برای ارزیابی یافت می شود و سپس با C پیاده سازی می شود، زیرا C به راحتی می تواند با برنامه های پایتون ادغام شود.

پیاده سازی هایی که الزامات را برآورده می کنند در برابر یکدیگر آزمایش می شوند و پیاده سازی هایی که بهترین نتیجه را می گیرند (بالاترین نسبت فشرده سازی) به عنوان کدگذاری دیکشنری و ماژول کدگذاری هافمن برای TCS انتخاب می شوند.

### 4.4 طراحیTCS

TCS یک راه حل فشرده سازی است که از سه ماژول تشکیل شده است: ماژول فشرده سازی (ASII (ACM، ماژول کدگذاری دیکشنری و ماژول کدگذاری هافمن. این ماژول ها به هم پیوسته هستند، به این معنی که آنها برنامه های مستقل هستند که می توانند در یک راه حل ادغام شوند،

یا می توانند به عنوان برنامه های مستقل استفاده شوند [۷۸]. داشتن ماژول های به هم پیوسته، چابکی را در جستجوی بالاترین نسبت فشرده سازی ایجاد می کند و برای ارزیابی ماژول ها به صورت جداگانه ضروری است. ماژول ها در C یا Python نوشته شده اند و می توانند به صورت متوالی با استفاده از یک برنامه پایتون یا یک اسکریپت پوسته یونیکس اجرا شوند.

TCS ابتدا برای ACM در نظر گرفته شده است، سپس رمزگذار دیکشنری و در نهایت ماژول کدگذاری هافمن استفاده کند. ACM ابتدا استفاده می شود زیرا فقط قادر به فشرده سازی کاراکترهای اسکی است. خروجی رمزگذار دیکشنری یا ماژول های کدگذاری هافمن، داده های دودویی هستند که شامل کاراکترهای اسکی نیستند، بنابراین این ماژول ها نباید قبل از ACM استفاده شوند. خروجی ACM یک فایل متنی در کدگذاری cpo37 است. این خروجی به کدگذار دیکشنری ارسال می شود که زیر رشته های تکراری را در متن جستجو می کند. سپس خروجی دودویی از رمزگذار دیکشنری به ماژول کدگذاری هافمن ارسال می شود که مقادیر بایت را که بیشتر از بقیه استفاده می شود، جستجو می کند. فایل خروجی از ماژول کدگذاری هافمن آخرین فایل فشرده شده از TCS است.

### 4.4.1 طراحی ACM

ACM برنامه ای است که در Python 3.6.9 نوشته شده است [۷۹] و شامل دو فایل اسکریپت است: ascii-compression.py و charprocessing.py. ascii-compression.py مسئول خواندن و نوشتن فایل هاست، در حالی که charprocessing.py فشرده سازی متن واقعی را کنترل می کند.

Ascii-compression.py الگوریتم فشرده سازی است که ACM از آن استفاده می کند مشابه الگوریتم شوکو است (بخش 2.2.1 را ببینید) و با ادغام کاراکترها با یکدیگر کار می کند. ادغام کاراکتر به این معنی است که دو کاراکتر از یک فایل متنی، به عنوان مثال "a" و "b"، می توانند با هم در فضای یک کاراکتر واحد ذخیره شوند. این الگوریتم تنها قادر به ادغام کاراکترهای اسکی است. همانطور که در بخش 2.2 ذکر شد، کاراکترهای اسکی فقط از ۷ بیت آخر برای کد نماد استفاده می کنند، در حالی که از اولین بیت می توان برای تشخیص خطاها در داده ها استفاده کرد. خروجی ACM (فایل فشرده) یک فایل متنی با کدگذاری cpo37 است. بر خلاف ASCII، کدگذاری cpo37 از همه ۸ بیت برای کد نماد استفاده می کند. در مورد ACM، اولین بیت از یک کاراکتر cpo37 برای نشان دادن ادغام یا نبودن کاراکتر استفاده می شود و ۷ بیت آخر کد نماد یک یا دو کاراکتر است. رمزگذاری cpo37 به عنوان فرمت خروجی انتخاب می شود زیرا از همه ۸ بیت به جای ۷ بیت ASCII استفاده می کند. cpo37 همچنین یک کدگذاری استاندارد در پایتون است و درست مانند US-ASCII برای زبان انگلیسی در نظر گرفته شده است [۸۰].

Asci-compression.py

اسکریپت ascii-compression.py یک فایل متنی با کدگذاری ASCII، UTF-8 یا cpo37 به عنوان ورودی می گیرد. فایل های باینری یا دیگر کدگذاری های متنی را نمی پذیرد. سپس هر خط از فایل متنی را به اسکریپت charprocessing.py ارسال می کند که فشرده سازی را کنترل می کند. پس از نوشتن فایل خروجی، یک آزمایش ساده را انجام می دهد تا بررسی کند که آیا فایل خروجی بزرگتر از فایل اصلی است یا خیر. این می تواند در صورتی اتفاق بیفتد که فایل ورودی دارای تعداد زیادی کاراکتر باشد که نمی توان آنها را در اسکی کدگذاری کرد، اگر فایل خروجی بزرگتر از فایل اصلی باشد، داده های فشرده شده حذف می شوند، زیرا هدف برنامه فشرده سازی ایجاد فایل های کوچکتر است. در عوض، فایل خروجی با داده های اصلی بازنویسی می شود و فایل فشرده شده را به اندازه فایل اصلی می سازد. در نهایت، فایل فشرده یک خط جدید اضافه می شود که شامل o یا ۱ است. این عدد نشان می دهد که آیا فایل فشرده تغییر کرده است (o) یا کپی از فایل اصلی است (۱). این اطلاعات برای فشرده ساز مهم است، که الگوریتم فشرده سازی فایل را کاهش می دهد یا همانطور که هست کپی می کند. هنگامی که یک فایل فشرده از حالت فشرده خارج می شود، عدد حذف شده و فایل متنی به حالت اصلی خود بازگردانده می شود.

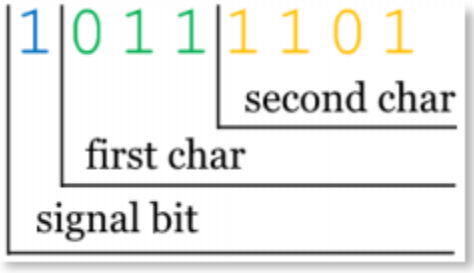
اسکریپت charprocessing.py وظیفه فشرده سازی و رفع فشار را بر عهده دارد. الگوریتم فشرده سازی هر کاراکتر یک متن را تکرار کرده و آنها را در یک فایل فشرده با کدگذاری cpo37 ذخیره می کند. نحوه ذخیره هر کاراکتر به دو عامل بستگی دارد. آیا می توان کاراکتر را در ASCII رمزگذاری کرد و آیا bigram زیر فقط شامل کاراکترهای معمولی است. اگر کاراکتر در برنامه کدگذاری ASCII گنجانده نشده باشد، به این معنی است که کاراکتر ممکن است بیش از یک بایت ذخیره سازی را اشغال کند. مثال UTF-8 کاراکتر "e" است که از دو بایت استفاده می کند. وقتی این کاراکتر فشرده می شود، الگوریتم باید علامت دهد که از بیش از یک بایت استفاده می کند. در غیر این صورت، هنگام خروج از فشرده سازی الگوریتم تصور می کند که دو کاراکتر وجود دارد که هر کدام از یک بایت استفاده کرده اند. قبل از نوشتن کاراکتر، دو علامت بک اسلش "\\" به کاراکتر اضافه می شود، که نشان می دهد بایت بعدی کاراکتر اسکی نیست. سپس کاراکتر ویژه به عددی که نشان دهنده کاراکتر است تبدیل می شود. این امر ضروری است زیرا اگر کاراکتر خاص بیشتر از یک بایت باشد، نمی تواند در cpo37 کدگذاری شود. سرانجام، یک بک اسلش "\" پس از کاراکتر اضافه می شود، به این معنی که کاراکنر ویژه به پایان رسیده است. داشتن علامت قبل و بعد از کاراکتر ضروری است زیرا طول کاراکتر می تواند بین ۲ تا ۴ بایت باشد. نسخه فشرده شده کاراکتر "e" به صورت \332\\ نشان داده می شود.

اگر فایل اصلی حاوی کاراکترهای بک اسلش "\" است، باید آنها را علامت دهید تا هنگام خروج از فشرده سازی الگوریتم برای یک کاراکتر غیر ASCII اشتباه نگیرد. به عنوان مثال اگر دو بک اسلش "\\" در یک سطر توسط کمپرسور به عنوان علامت نباشند، دیکمپرسور انتظار دارد کاراکتر بعدی یک کاراکتر ویژه باشد. روشی که سیگنال کمپرسور به بک اسلش نشان می دهد این است که قبل از آن دو بک اسلش دیگر اضافه می کند. این بدان معناست که هر بک اسلش در فایل اصلی با سه بک لش جایگزین می شود. هنگامی که دستگاه فشرده ساز در یک فایل فشرده یا بک اسلش مواجه می شود، می داند که کاراکنر بعدی نیز یک بک اسلش خواهد بود، اما یکی پس از آن می تواند آغاز یک کاراکتر ویژه یا سومین بک اسلش باشد. اگر کاراکتر خاصی باشد، به درستی از حالت فشرده خارج می شود، و اگر سومین بک اسلش باشد، دو بک اسلش برداشته می شود. اگر یک کاراکتر ASCII و کاراکتر بعدی آن هر دو در لیست کاراکترهای مشترک گنجانده شوند، آنها با هم ادغام می شوند. ادغام تنها در صورتی کار می کند که دو کاراکتر متوالی مشترک باشند. اگر هیچ یک، یا فقط یک مورد، از کاراکترهای متوالی در لیست کاراکترهای معمولی نباشند، ادغام نمی شوند. در عوض، آنها همانند فایل اصلی در فایل فشرده ذخیره می شوند، به عنوان مثال، کاراکتر "a" دارای یک بیت ارزش است ۰۱۱۰۰۰۰۱ . این نشان دهنده باینری عدد ۹۷ است و به این صورت است که ACM کاراکترها را می خواند. اولین بیت 0 است که به الگوریتم می گوید که ۷ بیت بعدی نشان دهنده یک کاراکتر ادغام نشده است.

```
2
3  firstCommonChars = ["e","t","a","o","n","i","h"," "]
4  secondCommonChars = ["e","t","a","o","n","i","h"," ","s","r","l","d","u","c","m","w"]
5
```

شکل ۲. فهرست کاراکترهای متداول

اگر یک کاراکتر در لیست firstCommonChars گنجانده شده باشد، همانطور که در شکل ۲ نشان داده شده است، و کاراکتر بعدی در لیست secondCommonChars گنجانده شده باشد، با هم ادغام می شوند. این لیستی از رایج ترین حروف (به علاوه نماد فاصله) در زبان انگلیسی است [۸] و مدل فشرده سازی ACM است. یک مدل فشرده سازی می تواند لیستی از تکرار کاراکترها، نمادها یا بیگرام ها باشد. ACM برای همه انواع متون از یک مدل استفاده می کند، به این معنی که متون انگلیسی باید نسبت فشرده سازی بهتری نسبت به متون سایر زبان ها داشته باشند.




شکل ۳. نمایش باینری از شخصیت ادغام شده

یک کاراکتر ادغام شده از ۳ قسمت تشکیل شده است: بیت سیگنال، کاراکتر اول و کاراکتر دوم. برای یک کاراکتر ادغام شده، اولین بیت روی ۱ تنظیم می شود. سه بیت بعدی نشان دهنده موقعیت اولین کاراکتر در لیست firstCommonChars است. در مثال شکل ۳، اولین بیت های کاراکتر ۰۱۱ هستند، که مربوط به عددی است که اولین کاراکتر دارای کاراکتر با شاخص ۳ در firstCommonchars است که "o" است. کاراکتر دوم از ۴ بیت استفاده می کند، به این معنی که می تواند دو برابر کاراکتر اول را نشان دهد. در مثال، بیت های کاراکتر دوم ۱۱۰۱ هستند که با عدد ۱۳ مطابقت دارد. کاراکتر با شاخص ۱۳ در secondCommonChars کاراکتر "c" است. نتایج حاصل از فشرده سازی نشان می دهد که کاراکتر ادغام شده در شکل ۳ نشان دهنده بیگرام "oc" است.

# 5 ارزیابی و نتایج

این فصل روند ارزیابی واحدهای مختلف واحد TCS و ارزیابی سیستم را به طور کلی شرح می دهد. ارزیابی ماژول های کدگذاری دیکشنری و کدگذاری هافمن شامل انتخاب پیاده سازی هایی با بیشترین نسبت فشرده سازی برای استفاده در TCS است. ارزیابی ACM شامل اندازه گیری نسبت فشرده سازی آن و اندازه گیری نسبت فشرده سازی Shoco و مقایسه نتایج است. ارزیابی کل سیستم (TCS) شامل اندازه گیری نسبت فشرده سازی آن و مقایسه عملکرد آن با سایر برنامه های فشرده سازی عمومی است.

## 5.1 ارزیابی ماژول های کدگذاری دیکشنری و کدگذاری هافمن

مجموعه ای از پیاده سازی های احتمالی برای کدگذاری دیکشنری و کدگذاری هافمن با هم مقایسه می شوند. مجموعه داده ها (بخش 4.2.1 را ببینید)، که شامل ۶ متن با ویژگی های مختلف است، روی هر یک از پیاده سازی ها آزمایش می شود. مجموعه داده ها به صورت محلی در کامپیوتر ذخیره می شود. نسبت فشرده سازی برای هر یک از این متون و برای هر یک از پیاده سازی ها اندازه گیری می شود. نسبت فشرده سازی اندازه گیری میزان کوچکتر بودن فایل فشرده در مقایسه با فایل فشرده نشده است و با تقسیم اندازه فشرده نشده بر اندازه فشرده شده یک فایل محاسبه می شود. به عنوان مثال، اگر یک فایل فشرده نشده ۱۰ مگابایت و فایل فشرده آن ۸ مگابایت است، نسبت پیاده سازی برای آن فایل خاص ۱,۲۵ است. اگر نسبت فشرده سازی ۱ باشد، فایل فشرده شده دارای اندازه یکسانی با فایل فشرده نشده است، بدین معنی که الگوریتم قادر به فشرده سازی فایل نیست. پیاده سازی هایی که بالاترین نسبت فشرده سازی را به طور متوسط برای هر متن در مجموعه داده دارند، به عنوان ماژول های کدگذاری دیکشنری و کدگذاری هافمن برای TCS انتخاب می شوند. میانگین نسبت فشرده سازی به عنوان معیار انتخاب شد زیرا انتظار می رود که راه حل برای انواع متن استفاده شود و پیاده سازی هایی که بیشترین نسبت فشرده سازی را برای هر متن دارند، مناسب ترین پیاده سازی ها هستند.

## 5.2 انتخاب پیاده سازی کدگذاری دیکشنری

چندین کدگذاری دیکشنری برای TCS در نظر گرفته شده است. این پیاده سازی ها برخی از الگوریتم های کدگذاری دیکشنری را نشان می دهند. الگوریتم دیکشنری LZW [۸۱]، LZ77 [۱۰]، LZ78 [۳۸] وRe-Pair [۴۰] است. این الگوریتم ها در نظر گرفته می شوند زیرا توسط برنامه های فشرده سازی محبوب استفاده می شوند [۴و۵و۶] و بنابراین موثربودن آنها ثابت شده است. الگوریتم های کدگذاری دیکشنری بیشتری وجود دارد، اما توسط برنامه های فشرده سازی محبوب، استفاده نمی شود.

بسیاری از پیاده سازی های موجود در وب سایت کنترل نسخه GitHub مشخص نمی کند که راه حل آنها دارای چه مجوزی است و بنابراین نمی توان آنها را در نظر گرفت. به نظر می رسد که مجوز باز الگوریتم LZ78 با نام Mezip [82] با متن ASCII کار می کند، اما با کدگذاری cpo37 کار نمی کند. پیاده سازی الگوریتم LZW برای کدگذاری UTF-8 کار نمی کند و نمی تواند مورد توجه قرار گیرد. پیاده سازی دیگری از الگوریتم LZW به نظر می رسد برای همه رمزگذاری ها کار می کند، اما فشرده سازی فایل فشرده باعث ایجاد خطا می شود. وب سایت غیر

انتفاعی Rosettacode [۸۵] پیاده سازی هایی برای الگوریتم LZW در چندین زبان برنامه نویسی، از جمله پایتون [۸۶] دارد، اما این راه حل برای کدگذاری همه کاراکترها کار نمی کند.

ماژول های استانداردی در پایتون برای برنامه های فشرده سازی محبوب مانند Gzip [۸۷]، Bz2 [۸۸] یا Zipfile [۸۹] وجود دارد، اما این ماژول ها همه از چند الگوریتم در ترکیب با یکدیگر استفاده می کنند. به عنوان مثال، همه این ماژول ها از کدگذاری هافمن استفاده می کنند و برخی از کدگذاری دیکشنری استفاده می کنند، اما نمی توان فقط از الگوریتم کدگذاری دیکشنری و کدگذاری هافمن استفاده کرد. پیاده سازی هایی که مورد استفاده قرار می گیرند نمی توانند بسته های چند الگوریتم باشند.

### 5.2.1 مقایسه پیاده سازی ها

یک پیاده سازی پایتون و دو پیاده سازی C شرایط مورد نیاز را برآورده می کند. آنها دارای مجوز باز هستند، حتی برای فایل های بزرگ نیز به درستی کار می کنند و با متن کدگذاری های مختلف سازگار هستند. سپس پیاده سازی ها با یکدیگر مقایسه شده و نسبت فشرده سازی برای هر متن در مجموعه داده ها اندازه گیری می شود. پیاده سازی پایتون با پایتون نسخه 2.7.17 اجرا می شود و پیاده سازی C با GNU Compiler Collection نسخه 7.5.0 کامپایل می شود.

یک پیاده سازی الگوریتم LZ77 است و در پایتون نوشته شده است [۹۰]. پیاده سازی ۱۵ ژوئن ۲۰۲۰ بارگیری شد. این پیاده سازی به کاربر این امکان را می دهد که اندازه پنجره الگوریتم (پنجره لغزان) [۱۰] را تغییر دهد. اندازه پنجره تعیین می کند که چقدر الگوریتم متن در هر نقطه برای پیدا کردن زیر رشته های منطبق تجزیه و تحلیل می کند. اندازه پنجره بزرگتر نسبت فشرده سازی بیشتری را ارائه می دهد، اما الگوریتم را کندتر می کند. اندازه پنجره ۴۰۰ کاراکتر برای همه آزمایش ها استفاده شد، زیرا این حداکثر اندازه ممکن برای پنجره بود.

پیاده سازی دیگر، پیاده سازی C الگوریتم LZW است [۹۱]. پیاده سازی در ۹ ژوئن ۲۰۲۰ بارگیری شد. الگوریتم LZW مانند LZ77 پنجره های لغزان استفاده نمی کند، اما در عوض زیر رشته ها را به یک دیکشنری اضافه می کند که ممکن است در متن تکرار شود یا نشود. اگر یک زیر رشته تکرار شود، آن را با اشاره به ورودی در دیکشنری جایگزین می کند [۳۵].

آخرین پیاده سازی پیاده سازی C الگوریتم LZ77 [۹۲] است. پیاده سازی ۱۵ ژوئن ۲۰۲۰ بارگیری شد. درست مانند اجرای پایتون، امکان تغییر اندازه پنجره نیز وجود دارد. حداکثر اندازه پنجره ممکن برای این پیاده سازی ۱,۰۴۸,۵۷۶ کاراکتر است و بنابراین از این برای همه آزمایش ها استفاده می شود.

| Data set | مجموعه داده ها | LZ77 (Python) | LZW (C) | Lz77 (C) |
|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 1.47 | 1.92 | 1.85 |
| cpo37 encoded XML file (18.1 MB) | فایل XML کدگذاری cpo37 | 1.45 | 1.61 | 1.75 |
| C code file (64 KB) | فایل کد C | 1.75 | 2.13 | 2.5 |
| English book (680 KB) | کتاب انگلیسی | 1.3 | 1.96 | 1.61 |
| Italian book (626 KB) | کتاب ایتالیایی | 1.33 | 2.13 | 1.56 |
| Chinese book (285 KB) | کتاب چینی | 1.16 | 1.59 | 1.34 |
| Average | میانگین | 1.41 | 1.89 | 1.77 |

جدول ۲. نتایج حاصل از پیاده سازی کدگذاری دیکشنری

جدول ۲ نشان دهنده نسبت فشرده سازی سه کدگذاری دیکشنری برای هر متن در مجموعه داده است. همانطور که در بخش 5.1 ذکر شد، نسبت فشرده سازی اندازه گیری میزان کوچکتر بودن فایل فشرده در مقایسه با فایل فشرده نشده است. ما می بینیم که اجرای پایتون از الگوریتم LZ77 بدترین نسبت فشرده سازی را دارد. این به احتمال زیاد به دلیل محدودیت اندازه پنجره پیاده سازی است. پیاده سازی C برای الگوریتم LZ77 دارای نسبت فشرده سازی بسیار بهتری است، حتی اگر از الگوریتم LZ77 نیز استفاده شود. این به این دلیل است که اندازه پنجره بسیار بزرگتری داشت. مشاهده می کنیم که پیاده سازی LZW نسبت به پیاده سازی C در LZ77 برای برخی از متون دارای نسبت فشرده سازی بالاتری است، اما برای متون دیگر نسبت کمتری دارد. همانطور که در بخش 5.1 ذکر شد، پیاده سازی که بهترین نتایج متوسط را از آزمایش داشته باشد، به عنوان ماژول کدگذاری دیکشنری برای TCS انتخاب می شود. بنابراین پیاده سازی C الگوریتم LZW به عنوان ماژول کدگذاری دیکشنری استفاده می شود.

### 5.3 انتخاب پیاده سازی کدگذاری هافمن

تعدادی پیاده سازی منبع باز از الگوریتم کدگذاری هافمن پیدا شده و مورد ارزیابی قرار می گیرد. اکثر پیاده سازی هایی که پیدا می شوند مشخص نمی کنند که راه حل آنها دارای چه مجوزی است و بنابراین نمی توان آنها را در نظر گرفت، درست مانند زمانی که پیاده سازی رمزگذاری دیکشنری پیدا شد. همانطور که در بخش 5.2 ذکر شد، ماژول های استاندارد پایتون برای برنامه های فشرده سازی محبوب که از الگوریتم هافمن استفاده می کنند وجود دارد، اما با الگوریتم های دیگر ترکیب شده است. هیچ ماژول استاندارد پایتون فقط برای الگوریتم هافمن وجود ندارد. دو پیاده سازی پایتون یافت می شود که قادر به فشرده سازی متن ASCII و UTF-8 هستند، اما هیچ یک از آنها قادر به فشرده سازی متن کد شده cpo37 نیستند [۹۳] [۹۴]. پیاده سازی در C یافت شد، اما قادر به ساخت کد منبع نیست، بنابراین نمی توان آن را ارزیابی کرد [۹۵].

#### 5.3.1 مقایسه پیاده سازی ها

سه پیاده سازی الزامات را برآورده می کنند و با یکدیگر مقایسه می شوند. یک پیاده سازی پایتون و دو پیاده سازی C. پیاده سازی پایتون با پایتون نسخه 3.6.9 اجرا می شود. و پیاده سازی C با GNU Compiler Collection نسخه 7.5.0 کمپایل می شود.

پیاده سازی پایتون Dahuffman نام دارد و توسط کاربر GitHub، Stefaan Lippens ساخته شده است [۹۶]. پیاده سازی ۲۶ ژوئیه ۲۰۲۰ بارگیری شد. پیاده سازی این قابلیت را دارد که بر اساس فهرستی از تکرارهای نماد، یک جدول کد (درخت هافمن) بسازد. فراوانی های نماد تعداد دفعاتی است که از کاراکترهای جداگانه در یک متن استفاده می شود. فراوانی نماد می تواند توسط کاربر ارائه شود یا پیاده سازی می تواند آن را از یک متن ورودی محاسبه کند. اگر فراوانی نماد از یک متن ورودی محاسبه شود، پیاده سازی ابتدا یک جدول کد ایجاد می کند، و سپس متن را با استفاده از جدول کد ذکر شده فشرده می کند.

دو پیاده سازی C، توسط کاربران GitHub، Gagarine Yaikhom و Doug Richardson [۹۷و۹۸]، جداول کد را به طور خودکار در طول فشرده سازی ایجاد می کنند. پیاده سازی Yaikhom در ۲۶ ژوئیه ۲۰۲۰، و اجرای Richardson در ۲۸ ژوئیه ۲۰۲۰ بارگیری شد.

| Data set | مجموعه داده ها | Dahuffman | Yaikhom C impl. | Richardson C impl. |
|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 1.51 | 1.5 | 1.5 |
| cpo37 encoded XML file (18.1 MB) | فایل XML کدگذاری cpo37 | 1.2 | 1.2 | 1.2 |
| C code file (64 KB) | فایل کد C | 1.48 | 1.47 | 1.47 |
| English book (680 KB) | کتاب انگلیسی | 1.8 | 1.7 | 1.7 |
| Italian book (626 KB) | کتاب ایتالیایی | 1.95 | 1.37 | 1.37 |
| Chinese book (285 KB) | کتاب چینی | 2.32 | 1.31 | 1.31 |
| Average | میانگین | 1.71 | 1.49 | 1.49 |

جدول ۳. نتایج حاصل از پیاده سازی کدگذاری نویسی هافمن

جدول ۳ نشان دهنده نسبت فشرده سازی سه برنامه نویسی هافمن برای هر متن در مجموعه داده است. همانطور که جدول ۳ نشان می دهد، برای اکثر متون تفاوت چندانی بین پیاده سازی ها وجود ندارد. در واقع، دو پیاده سازی C نتایج دقیقا یکسانی را به دست آوردند، اگرچه دو پیاده سازی متفاوت توسط افراد مختلف است [۹۷] [۹۸]. پیاده سازی پایتون، Dahuffman، نسبت فشرده سازی کمی بهتر از دو پیاده سازی C برای کتاب انگلیسی و ایتالیایی، اما نسبت بهتری برای کتاب چینی دارد. در مورد سه متن دیگر، Dahuffman کم و بیش نتایج مشابهی با سایر پیاده سازی ها دارد. این می تواند به این دلیل باشد که این کتابها بیشتر از سایر متون دارای کاراکتر UTF-8 هستند، و Dahuffman در فشرده سازی کاراکترهای غیراسکی کارایی بهتری دارد. کتاب چینی به طور کامل از کاراکترهای UTF-8 تشکیل شده است، بنابراین این منطقی ترین نظریه به نظر می رسد. Dahuffman، دارای بالاترین نسبت فشرده سازی متوسط پیاده سازی ها است و بنابراین به عنوان ماژول کدگذاری هافمن برای TCS استفاده می شود.

### 5.4 ارزیابی ACM

برنامه فشرده سازی TCS شبیه به Shoco است. هر دو TCS و Shoco برنامه های فشرده سازی متن بدون اتلاف هستند که در فشرده سازی متن ASCII تخصص دارند [۱۵]. یک تفاوت اصلی این است که Shoco یک الگوریتم است در حالی که TCS یک راه حل فشرده سازی است که از ۳ الگوریتم کدگذاری LZW، ACM و Huffman تشکیل شده است. ACM عملکرد و راه حل بسیار مشابهی با Shoco دارد. به منظور مقایسه عادلانه، فقط ACM در این قسمت با Shoco مقایسه می شود.

مقاله مورد بحث در 2.4.2 یکی دیگر از برنامه های فشرده سازی ASCII است. با این حال، این راه حل در مقایسه با ACM به دلیل محدودیت های موجود نیست. راه حل پیشنهادی فقط قادر به فشرده سازی متن ASCII است و هیچ کدگذاری دیگری ندارد. از متون مورد استفاده در مجموعه داده ها، فقط فایل کد C با راه حل سازگار است، بنابراین نمی توان مقایسه عادلانه ای با ACM داشت.

ارزیابی ACM شامل اندازه گیری نسبت فشرده سازی آن برای همه متون مجموعه داده و مقایسه نتایج با Shoco است. مجموعه داده ها شامل ۶ متن با ویژگی های متفاوت است. برخی از متون به طور کامل یا عمدتا کاراکترهای ASCII هستند و متون دیگر عمدتا یا به طور کامل کاراکتر UTF-8 هستند. متون UTF-8 برای آزمایش اینکه چگونه ACM و Shoco می توانند فایل هایی را که مورد استفاده نیستند فشرده کنند، گنجانده شده است. هر آزمایش یکبار تکرار می شود تا از صحت اطلاعات اطمینان حاصل شود.

ACM از یک مدل فشرده سازی پیش فرض استفاده می کند که برای داده هایی که قرار است فشرده شود آموزش ندیده است. یک مدل فشرده سازی (همانطور که در بخش 4.4.1 ذکر شده است) لیستی از کاراکترها، نمادها یا بیگرام ها است که اغلب در یک متن استفاده می شود.

Shoco این قابلیت را دارد که یک مدل فشرده سازی را برای یک فایل متنی خاص آموزش دهد، به این معنی که نمادها و بیگرام های موجود در مدل اغلب در یک متن استفاده می شود. ACM این قابلیت را ندارد و از یک مدل فشرده سازی عمومی استفاده می کند که برای کلمات انگلیسی بهینه شده است. در مقایسه، شوکو هم با مدل پیش فرض خود و هم با یک مدل آموزش دیده آزمایش می شود.

### 5.4.1 نتایج حاصل از مقایسه ACM وShoco

| Data set | مجموعه داده ها | ACM | Shoco پیش فرض | Shoco آموزش دیده |
|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 1.19 | 1.23 | 1.4 |
| cpo37 encoded XML file (18.1 MB) | فایل XML کدگذاری cpo37 | 1 | 0.59 | 0.75 |
| C code file (64 KB) | فایل کد C | 1.21 | 1.17 | 1.35 |
| English book (680 KB) | کتاب انگلیسی | 1.46 | 1.32 | 1.56 |
| Italian book (626 KB) | کتاب ایتالیایی | 1.29 | 1.15 | 1.5 |
| Chinese book (285 KB) | کتاب چینی | 1 | 0.53 | 0.75 |
| Average | میانگین | 1.19 | 1 | 1.22 |

جدول ۴. نتایج حاصل از مقایسه ACM و Shoco

همانطور که جدول ۴ نشان می دهد، نسبت فشرده سازی متن های مختلف متفاوت است. این انتظار می رود، زیرا ACM و Shoco باید نسبت فشرده سازی بیشتری برای متون سنگین اسکی داشته باشند زیرا آنها فقط می توانند کاراکترهای ASCII را فشرده کنند. فایل XML با کد cpo37 و کتاب چینی دو متنی هستند که نسبت فشرده سازی ۱ برای هر دو الگوریتم دارند، به این معنی که حجم فایل فشرده کوچکتر نمی شود. این متون به ترتیب از نویسه cpo37 و UTF-8 تشکیل شده اند. ACM و Shoco هر دو فقط قادر به فشرده سازی کاراکترهای ASCII هستند، و بنابراین رمزگذاری های دیگر می توانند خروجی را بزرگتر کنند. برای Shoco، نسبت فشرده سازی کمتر از ۱ است، هر دو با یک مدل پیش فرض و آموزش دیده، به این معنی که فایل فشرده بزرگتر از فایل اصلی است. ACM دارای نسبت ۱ است، به این معنی که فایل فشرده دارای اندازه یکسان با فایل اصلی است. دلیل این امر این است که ACM آزمایشی را برای بررسی بزرگتر بودن فایل فشرده از نسخه اصلی انجام می دهد. اگر چنین باشد، فشرده سازی کنار گذاشته می شود و فایل فشرده بدون تغییر باقی می ماند. Shoco این آزمایش را ندارد و بنابراین یک فایل فشرده بزرگتر دریافت می کند.

همانطور که جدول ۴ نشان می دهد، کتاب انگلیسی بالاترین نسبت فشرده سازی را برای هر دو الگوریتم دارد. مدلهای پیش فرض که دو الگوریتم استفاده می کنند برای کلمات انگلیسی بهینه شده اند و بنابراین نسبت بالایی برای کتاب انگلیسی به دست می آورند. این نسبت هنگام استفاده از یک مدل آموزش دیده برای شوکو حتی بیشتر است زیرا مدل برای آن متن خاص سفارشی شده است. دلیل اینکه ACM و نسخه پیش فرض Shoco برای متون مشابه نتایج متفاوتی می گیرند این است که مدل های فشرده سازی پیش فرض آنها متفاوت است [۹۹]. مدل ACM نسبتاً ساده است. این لیستی از ۱۵ کاراکتر پرکاربرد در زبان انگلیسی و نماد فاصله است. شوکو از یک لیست مشابه استفاده می کند، اما همچنین فهرستی بسیار بزرگتر از کاراکترهایی دارد که به احتمال زیاد از کاراکترهای خاصی پیروی می کنند [۹۹].

### 5.5 ارزیابی TCS

ارزیابی TCSشامل اندازه گیری نسبت فشرده سازی آن برای همه متون موجود در مجموعه داده ها و سپس مقایسه نتایج با سایر برنامه های فشرده سازی عمومی است. وقتی TCS ارزیابی می شوند، از هر سه ماژول استفاده می شود. این بدان معناست که خروجی از یک ماژول به ماژول بعدی ارسال می شود، نسبت فشرده سازی برای خروجی ماژول قبلی اندازه گیری می شود. این آگاهی را می دهد که هر ماژول چقدر از فشرده سازی را بر عهده دارد. در نهایت، نسبت تراکم کل اندازه گیری می شود. این نسبت اندازه فایل اصلی در مقایسه با اندازه نهایی فایل فشرده است. همانطور که در بخش 4.4 ذکر شد. ابتدا ACM، سپس کدگذاری دیکشنری (LZW) و در نهایت کدگذاری هافمن استفاده می شود.

برنامه های فشرده سازی مورد استفاده برای مقایسه از الگوریتم های فشرده سازی بدون اتلاف استفاده می کنند که می تواند در انواع داده ها، هم در فایل های متنی و هم در فایل های باینری مانند تصویر، صدا یا ویدئو استفاده شود. این برنامه ها برای مقایسه استفاده می شوند زیرا عموماً برای همان وظایفی که TCS در نظر گرفته است استفاده می شوند. به عنوان مثال، ویکی پدیا سرویسی برای بارگیری همه مقالات پایگاه داده خود در قالب فقط متن دارد. این فایل های متنی با Bzip2، ZipیاZip-7 فشرده شده اند [۱۰۰]. TCS با برنامه های فوق مقایسه می شود. Gzip همچنین در مقایسه استفاده می شود زیرا یک برنامه فشرده سازی محبوب است که امروزه مورد استفاده قرار می گیرد [۱۰۱] [۱۰۲].

#### 5.5.1 نتایج حاصل از ارزیابیTCS

| Data set | مجموعه داده ها | ACM | LZW | Huffman coding | Total ratio |
|---|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 1.19 | 1.6 | 1.01 | 1.93 |
| cpo37 encoded XML file (18.1 MB) | فایل XML کدگذاری cpo37 | 1 | 1.6 | 1.01 | 1.62 |
| C code file (64 KB) | فایل کد C | 1.21 | 1.83 | 1.01 | 2.23 |
| English book (680 KB) | کتاب انگلیسی | 1.46 | 1.36 | 1 | 1.99 |
| Italian book (626 KB) | کتاب ایتالیایی | 1.29 | 1.57 | 1 | 2.02 |
| Chinese book (285 KB) | کتاب چینی | 1 | 1.59 | 1 | 1.59 |
| Average | میانگین | 1.19 | 1.59 | 1.01 | 1.9 |

جدول ۵. نتایج حاصل از ارزیابی TCS

همانطور که گفته شد، نسبت فشرده سازی اندازه گیری شده برای هر ماژول بر اساس خروجی ماژول قبلی است. "نسبت کل" نسبت فشرده سازی است که از TCS (هر سه ماژول) برای متون موجود در مجموعه داده به دست می آید. نسبت کل نتیجه ای است که در ارزیابی با سایر برنامه های فشرده سازی در بخش بعدی مورد استفاده قرار می گیرد.

همانطور که جدول ۵ نشان می دهد، برخی از فایل ها بیشتر از ACM و برخی دیگر از LZW سود بیشتری می برند. به عنوان مثال فایل کد C کتاب انگلیسی از ACM نسبت کوچکتری از LZW دارد. به این دلیل است که کتاب انگلیسی از کاراکترهای مکرر زبان انگلیسی بیشتری استفاده می کند و بنابراین نسبت بهتری از ACM دریافت می کنند، در حالی که فایل کد C دارای کلمات و زیر رشته های تکراری تر است و بنابراین نسبت بهتری از LZW دریافت می کند.

جدول ۵ همچنین نشان می دهد که کدگذاری هافمن تقریبا هیچ فشرده سازی را برای هیچ یک از فایل ها بدست نمی آورد. این می تواند به این دلیل باشد که برنامه نویسی هافمن برای فشرده سازی داده های باینری طراحی نشده است. همچنین ممکن است خروجی دودویی از LZW بسیار تصادفی و نامنظم باشد تا کدگذاری هافمن نتواند به فشرده سازی برسد. خروجی LZW قبلاً توسط ACM فشرده شده است و ممکن است ترکیب این دو ماژول نیاز به کدگذاری هافمن را برطرف کند. نتایج حاصل از این ارزیابی نیاز به ارزیابی جدیدی دارد که فقط از کدگذاری LZW و هافمن استفاده می شود، تا به پاسخ این سوال برسیم که آیا ACM قابلیت های ماژول کدگذاری هافمن را مختل می کند یا این ماژول قادر به فشرده سازی داده های باینری نیست.

| Data set | مجموعه داده ها | LZW | Huffman coding | Total ratio |
|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 1.92 | 1 | 1.92 |
| C code file (64 KB) | فایل کد C | 2.13 | 1.01 | 2.14 |
| English book (680 KB) | کتاب انگلیسی | 1.96 | 1 | 1.96 |
| Italian book (626 KB) | کتاب ایتالیایی | 2.13 | 1 | 2.13 |
| Average | میانگین | 2.04 | 1 | 2.04 |

جدول ۶. نتایج حاصل از ارزیابی ماژول کدگذاری LZW و هافمن

فایل XML کد شده cpo37 و کتاب چینی در این ارزیابی گنجانده نشده اند زیرا ACM هیچ تاثیری بر روی این فایل ها نخواهد داشت و هدف از این ارزیابی مقایسه با نتایجی است که ACM روی آن تأثیر گذاشته است. جدول ۶ نشان می دهد که حتی وقتی از ACM استفاده نمی شود، ماژول کدگذاری هافمن عملاً قادر به فشرده سازی داده های باینری نیست.

### 5.5.2 نتایج حاصل از مقایسه TCS

| Data set | مجموعه داده ها | TCS | Bzip2 | Zip | 7-Zip | Gzip |
|---|---|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 1.93 | 3.79 | 2.96 | 4.24 | 2.96 |
| cpo37 encoded XML file (18.1 MB) | فایل XML کدگذاری cpo37 | 1.62 | 3.12 | 2.45 | 3.48 | 2.45 |
| C code file (64 KB) | فایل کد C | 2.23 | 4.22 | 3.88 | 4.25 | 3.91 |
| English book (680 KB) | کتاب انگلیسی | 1.99 | 3.52 | 2.65 | 3.18 | 2.65 |
| Italian book (626 KB) | کتاب ایتالیایی | 2.02 | 3.54 | 2.66 | 3.14 | 2.66 |
| Chinese book (285 KB) | کتاب چینی | 1.59 | 2.59 | 2.01 | 2.35 | 2.01 |
| Average | میانگین | 1.9 | 3.46 | 2.77 | 3.44 | 2.77 |

جدول ۷. نتایج حاصل از مقایسه TCS

جدول ۷ نشان می دهد که TCS نسبت به سایر برنامه ها برای هر فایل در مجموعه داده نسبت فشرده سازی کمتری دارد. Bzip2 و7-Zip بالاترین میانگین نسبت فشرده سازی را دارند زیرا از الگوریتم های متفاوتی نسبت به Zip و Gzip استفاده می کنند که بر اساس الگوریتم DEFLATE است [۴و۵و۴۷و۱۰۳]. TCS از راه حلی مشابه DEFLATE، اما با ACM به عنوان پیش پردازنده استفاده می کند. الگوریتم DEFLATE ترکیبی از رمزگذار دیکشنری LZ77 و کدگذاری هافمن [۱۱] است، در حالی که TCS از رمزگذار دیکشنری LZW و کدگذاری هافمن استفاده می کند. جدول ۶ نشان می دهد که ماژول کدگذاری هافمن مورد استفاده برای TCS با ماژول LZW سازگار نیست و بنابراین TCS جایگزینی برای الگوریتم DEFLATE ندارد. برای بررسی اینکه آیا TCS در صورت جایگزینی ماژول کدگذاری LZW و Huffman که در TCS با یک الگوریتم DEFLATE خالص جایگزین شده است، نتایج بهتری خواهد گرفت یا خیر، باید ارزیابی جدیدی انجام شود. در این ارزیابی، از برنامه های Zip و Gzip با ACM به عنوان پیش پردازنده استفاده می شود. این ارزیابی به این سوال پاسخ می دهد که آیا استفاده از ACM به عنوان پیش پردازنده باعث افزایش نسبت فشرده سازی الگوریتم DEFLATE می شود یا خیر.

| Data set | مجموعه داده ها | Zip | ACM + Zip | Gzip | ACM + Gzip |
|---|---|---|---|---|---|
| XML file (21.6 MB) | فایل XML | 2.96 | 2.92 | 2.96 | 2.92 |
| C code file (64 KB) | فایل کد C | 3.88 | 3.77 | 3.91 | 3.82 |
| English book (680 KB) | کتاب انگلیسی | 2.65 | 2.66 | 2.65 | 2.66 |
| Italian book (626 KB) | کتاب ایتالیایی | 2.66 | 2.58 | 2.66 | 2.58 |
| Average | میانگین | 3.04 | 2.98 | 3.05 | 3 |

جدول ۸. نتایج حاصل از مقایسه ACM + DEFLATE

مانند جدول ۶، فایل XML کد شده cpo37 و کتاب چینی در این ارزیابی گنجانده نشده اند زیرا ACM هیچ تاثیری بر روی این فایل ها نخواهد داشت. همانطور که جدول ۸ نشان می دهد، هنگامی که ACM به عنوان پیش پردازنده Zip یا Gzip استفاده می شود، نسبت فشرده سازی کمی پایین تر است. ACM فقط نسبت به کتاب انگلیسی نسبت فشرده سازی را تا حدی بیشتر نشان می دهد، اما افزایش ۰,۰۱ نسبت فشرده سازی ناچیز است. جدول ۸ همچنین نشان می دهد که استفاده از ACM به علاوه الگوریتم DEFLATE که Zip و Gzip استفاده می کند نسبت فشرده سازی بسیار بالاتری نسبت به TCSدارد.

# 6 بررسی

این فصل بررسی نتایج حاصل از ارزیابی ها را پوشش می دهد. همچنین در مورد کاستی های راه حل، مشارکت این پایان نامه و این که در چه شرایطی می تواند مورد استفاده قرار گیرد.

## 6.1 کارآییTCS

TCS مدعی الگوریتم DEFLATE بود، اما با تمرکز بر فشرده سازی متن ASCII. از نظر مفهومی، TCS از تکنیک های مشابه DEFLATE (کدگذاری دیکشنری و کدگذاری هافمن) استفاده می کند اما ACM را به عنوان یک گام جدید در کانال فشرده سازی معرفی می کند. تفاوت بین DEFLATE و TCS در این است که DEFLATE از رمزگذار دیکشنری LZ77 [۱۱] استفاده می کند در حالی که TCS از رمزگذار دیکشنری LZW استفاده می کند. ارزیابی TCS نشان می دهد که جایگزین DEFLATE که TCS از آن استفاده می کند ناقص است. بنابراین، استفاده از TCS (در حالت فعلی آن) به عنوان جایگزینی برای الگوریتم DEFLATE گزینه مناسبی نیست. مقایسه برنامه های فشرده سازی در جدول ۷ همچنین نشان داد که برنامه هایی که از راه حل های جایگزین برای الگوریتم DEFLATE استفاده می کردند نسبت فشرده سازی بالاتری داشتند. بنابراین، حتی اگر TCS دارای یک الگوریتم کارکرد DEFLATE باشد، باز هم نمی تواند با نسبت های فشرده سازی Bzip2 و7-Zip رقابت کند. در عوض، سهم این پایان نامه باید ACM باشد و نه TCS.

## 6.2 مقایسه ACM وShoco

شوکو مدعی نزدیک ACM است و کار مرتبطی است که بیشتر در این پایان نامه مورد بحث قرار گرفته است. هر دو الگوریتم دارای پیش فرض یکسان و راه حل های فنی مشابه هستند، اما یکسان نیستند. یک تفاوت پیچیدگی مدل های فشرده سازی است که دو الگوریتم از آن استفاده می کنند. تفاوت دیگر کارایی الگوریتم ها است. الگوریتم شوکو به عنوان "کمپرسور سریع برای رشته های کوتاه" توصیف می شود و به نظر می رسد نقطه اصلی فروش شوکو سرعت آن باشد [۱۵]. ACM برای تأخیر اندازه گیری نشده است، اما آزمایش نشان داده است که Shoco (با مدل فشرده سازی پیش فرض) سریعتر از ACM است. دلیل اینکه ACM برای تأخیر اندازه گیری نشده این است که تمرکز ACM بر قابلیت فشرده سازی آن است و نه بر کارایی آن. ACM به عنوان اثبات این مفهوم بود که الگوریتم فشرده سازی ASCII از لحاظ نظری امکان پذیر است. این قبل از اینکه شوکو برای نویسنده شناخته شود، ثابت شد که فشرده سازی ASCII امکان پذیر است.

### 6.2.1 مدلهای آموزش دیده

در مقایسه بین ACM و Shoco (جدول ۴)، Shoco هم با مدل فشرده سازی پیش فرض و هم با یک مدل فشرده سازی آموزش دیده استفاده شد. ACM قابلیت ساخت یک مدل آموزش دیده را ندارد و بنابراین از مدل پیش فرض خود استفاده کرد. ساده ترین روش ساخت یک مدل آموزش دیده برای شوکو در سه مرحله انجام می شود. ابتدا، یک اسکریپت پایتون یک فایل متنی را به عنوان ورودی می گیرد و مدل فشرده سازی را به عنوان یک فایل هدر C خروجی می دهد. سپس، برنامه Shoco، که با C نوشته شده است، باید با مدل جدید کامپایل شود. در

نهایت، برنامه کامپایل شده با فایل متنی مورد نظر به عنوان ورودی اجرا می شود. همچنین ممکن است چندین جنبه در مورد نحوه ساخت مدل را سفارشی کرد، به عنوان مثال آیا علامت های نگارشی در آن گنجانده می شود یا خیر، اما این امر نیازمند دانش در مورد فایلی است که قرار است فشرده شود، می باشد. ایجاد مدلهای آموزش دیده برای شوکو ممکن است نسبت فشرده سازی را افزایش دهد (جدول ۴ را ببینید)، اما این یک کار خسته کننده و وقت گیر است. همانطور که گفته شد، نقطه اصلی فروش Shocoسرعت آن است و ایجاد یک مدل آموزش دیده در هر بار استفاده از برنامه، روند فشرده سازی را کندتر می کند. مدلهای آموزش دیده می توانند برای کار فشرده سازی یکبار مفید باشند، به عنوان مثال اگر یک فایل واحد و بزرگ نیاز به فشرده سازی دارد. با این حال، به عنوان یک برنامه فشرده سازی متن عمومی، باید از مدل پیش فرض استفاده کرد.

### 6.2.2 بحث در مورد نتایج حاصل از مقایسه

نتایج جدول ۴ نشان می دهد که ACM (با استفاده از مدل پیش فرض آن) نسبت فشرده سازی بیشتر از نسخه پیش فرض Shoco برای هر متن به جز یک متن دارد. با وجود این، نتایج برای متونی که عمدتاً دارای نویسه های ASCII هستند، تقریباً مشابه است. دو متنی که هیچ کاراکتر ASCII ندارند (فایل XML کد شده cpo37 و کتاب چینی) در Shoco نسبت فشرده سازی کمتر از ۱ دارند، این بدان معناست که حجم فایل فشرده بزرگتر می شود. همانطور که گفته شد، ACM آزمایشی را بررسی می کند که آیا فایل فشرده بزرگتر از فایل اصلی است و در صورت وجود، فشرده سازی را کنار می گذارد. Shoco این آزمایش را انجام نمی دهد و بنابراین می تواند با یک فایل فشرده بزرگتر به پایان برسد. این آزمون برای یک برنامه فشرده سازی متن عمومی مفید است زیرا به دانش قبلی در مورد رمزگذاری فایل متنی نیاز ندارد. بنابراین Shoco فقط در صورتی مورد استفاده قرار می گیرد که کاربر بداند که یک متن بیشتر دارای کاراکترهای ASCII است، در حالی که ACM را می توان در هر متنی بدون اطلاع از رمزگذاری آن استفاده کرد.

### 6.2.3 کاستی هایACM

تکنیک فشرده سازی که ACM برای کاراکترهای غیر ASCII استفاده می کند بهینه نیست. برخی از کاراکترهای UTF-8 مانند ایموجی ها می توانند تا ۴ بایت در یک فایل فشرده نشده استفاده کنند. این کاراکترها در صورت فشرده سازی از ۹ بایت استفاده می کنند. فشرده سازی غیر بهینه با تبدیل کاراکتر به عددی که آن را نشان می دهد و نوشتن رقم عدد برای رقم ایجاد می شود. این می تواند بهینه سازی شود، اما محدودیت زمانی منجر به این مسئله می شود که هنوز باید برطرف شود. فایلی که با ACM فشرده می شود دارای یک خط اضافه شده یا ۱ یا o می باشد. این به این معنی است که آیا فایل تغییر کرده است یا نه، و هنگامی که فایل از حالت فشرده خارج می شود، حذف می شود. مشکلی که ممکن است رخ دهد این است که اگر یک فایل دارای خط 1 یا o به عنوان آخرین خط باشد، و مشخص نیست که فایل فشرده است یا نه. اگر کاربر این فایل فشرده نشده را با یک فایل فشرده اشتباه کند، پس از فشرده سازی تغییر می کند. یک فایل فشرده به عنوان یک فایل متنی (.txt) ذخیره می شود و بنابراین احتمالاً دارای همان نوع فایل با یک فایل فشرده نشده است، بنابراین نمی توان گفت آیا فایل فشرده است یا نه.

ACM به زبان برنامه نویسی پایتون نوشته شده است، که در زمان اجرا به طور قابل توجهی برای مثال کندتر از زبان Cاست [۱۰۴و۱۰۵]. زبان برنامه نویسی C به طور کلی سریعتر از پایتون است، به همین دلیل برنامه های پایتون اغلب به دلایل کارآیی ماژول های C را ادغام می کنند [۱۰۶]. اگر ACM به زبان C نوشته می شد، به طور بالقوه می توانست سریعتر باشد.

## 7 نتیجه گیری و کارهای آینده

این پایان نامه راه حلی برای TCS برای فشرده سازی متن با تمرکز ویژه بر فشرده سازی متن ASCII ارائه می دهد. TCS مشکلات ذخیره و انتقال زمان زیادی از داده های متنی را برطرف می کند. TCS حجم فایل ها را کاهش می دهد بنابراین به فضای ذخیره سازی کمتری نیاز دارند. علاوه بر این، TCS هنگام بارگیری یا انتقال فایل فشرده از طریق اینترنت، پهنای باند را کاهش می دهد. TCS برای متن ASCII سفارشی شده است با این حال برای کار بر روی هر شکل از متن طراحی شده است. جاه طلبی TCS فشرده سازی فایل هایی برای کاربرانی است که با مقادیر زیادی متون ASCII یا ASCII سنگین سروکار دارند. TCS از سه ماژول ACM، ماژول کدگذاری دیکشنری و ماژول کدگذاری هافمن تشکیل شده است. TCS مبتنی بر الگوریتم DEFLATE است که همچنین کدگذاری دیکشنری و کدگذاری هافمن را ترکیب می کند. TCS از ACM به عنوان پیش پردازنده جایگزین DEFLATE برای افزایش فشرده سازی برای آزمایش ASCII استفاده می کند. ACM توسط نویسنده به طور خاص برای این پروژه توسعه یافته است، در حالی که دیکشنری و ماژول های کدگذاری هافمن پیاده سازی شده توسط افراد دیگر است [۹۱] [۹۶]. ماژول های خاص مورد ارزیابی قرار گرفته اند، و TCS ارزیابی شده و با برنامه های فشرده سازی عمومی مورد مقایسه قرار گرفته است.

نتایج حاصل از ارزیابی TCS نشان می دهد که در حالت فعلی، مدعی سایر برنامه های فشرده سازی عمومی نیست. مقایسه فشرده سازی اسکی با الگوریتم Shoco نشان می دهد که هنگام استفاده از مدل های فشرده سازی عمومی، ACM نسبت فشرده سازی متوسط بهتری دارد. بنابراین ACM می تواند مدعی رشته تخصصی تر فشرده سازی ASCII باشد. ارزیابی ACM (جدول ۴ را ببینید) نشان می دهد که می توان آن را در هر شکلی از متن بدون افزایش اندازه فشرده استفاده کرد، اما نسبت فشرده سازی قابل توجهی فقط در متون سنگین ASCII به دست می آید. با این وجود، نسبت فشرده سازی ACM در مقایسه با برنامه های فشرده سازی عمومی بسیار کوچک است (جدول ۷ را ببینید). ترکیب ACM با برنامه های فشرده سازی مبتنی بر DEFLATE (جدول ۸ را ببینید) نسبت فشرده سازی متوسط را فقط تحت DEFLATE نشان می دهد، با یک مثال از مزیت جزئی ACM. ممکن است بهبود الگوریتم فشرده سازی ACM نسبت فشرده سازی آن را افزایش دهد و حتی ممکن است از DEFLATE فراتر رود (در صورتی که ACM به عنوان پیش پردازنده DEFLATE استفاده شود).

### 7.1 پاسخ به سوالات تحقیق

- "چه ترکیبی از تکنیک ها می تواند نسبت فشرده سازی متن ASCII را بهبود بخشد؟"

بدیهی است که ACM برای فایل های ASCII به فشرده سازی دست می یابد، اگرچه برخی از فایل های متنی نسبت فشرده سازی بیشتری نسبت به سایرین دارند (جدول ۴ را ببینید). علیرغم شایستگی ACM، استفاده از آن در ترکیب با کدگذاری دیکشنری و ماژول کدگذاری هافمن (و همچنین برنامه های فشرده سازی مبتنی بر DEFLATE) نسبت فشرده سازی متن ASCII را بهبود نداده است. دلیل این امر این است که خروجی ACM کمتر از متن ساده قابل فشرده سازی است. ACMفقط به عنوان پیش پردازنده برنامه های فشرده سازی مبتنی بر DEFLATE آزمایش شده است، زیرا TCS جایگزین DEFLATE بود. ممکن است ترکیب الگوریتم های دیگر (مانند الگوریتم های مورد استفاده در Bzip2 و 7-Zip) با ACM به عنوان پیش پردازنده بتواند نسبت فشرده سازی متن ASCII را افزایش دهد. این کار آینده TCS است.

- "آیا استفاده از فشرده سازی ASCII در ترکیب با کدگذاری دیکشنری و کدگذاری هافمن از نسبت فشرده سازی DEFLATE فراتر می رود؟"

در حالت فعلی، TCS (ACM، ماژول کدگذاری دیکشنری و ماژول کدگذاری هافمن) از نسبت فشرده سازی DEFLATE فراتر نمی رود. همانطور که گفته شد، ماژول برنامه نویسی هافمن TCS با ماژول برنامه نویسی دیکشنری سازگار نیست و بنابراین TCS جایگزین DEFLATE نیست. در عوض، ACM در ترکیب با برنامه نویسی دیکشنری و برنامه نویسی هافمن برنامه های فشرده سازی مبتنی بر DEFLATE در جدول ۸ آزمایش شده است. بر اساس میانگین نسبت فشرده سازی از ۴ متن مجموعه داده، با استفاده از ACM در ترکیب با رمزگذاری دیکشنری و هافمن کدگذاری از نسبت فشرده سازی DEFLATE فراتر نمی رود.

- "آیا این ترکیب فشرده سازی متن نسبت فشرده سازی بالاتری نسبت به برنامه های فشرده سازی عمومی برای متن ASCII به دست می آورد؟"

در مقایسه بین TCS و ۴ برنامه فشرده سازی عمومی در جدول ۷، ۴ متون بیشتر شامل کاراکترهای ASCII هستند. همانطور که گفته شد، نه ترکیب TCS و نه ACM + DEFLATE نسبت فشرده سازی بالاتری نسبت به برنامه های فشرده سازی مبتنی بر DEFLATE برای متون ASCII ندارند. علاوه بر این، Bzip2 و ۷-Zip از هر دو برنامه فشرده سازی مبتنی بر DEFLATE و TCS برای هر متنی در مورد استفاده بهتر هستند. در نتیجه، TCS نسبت فشرده سازی بیشتر از برنامه های فشرده سازی عمومی برای متن ASCII به دست نمی آورد.

- "آیا این ترکیب فشرده سازی متن نسبت فشرده سازی بالاتری نسبت به راه حل های فشرده سازی که روی متن ASCIIتخصص دارند، بدست می آورد؟

در این پایان نامه، ACM تنها با فشرده سازی اسکی توسط الگوریتم Shoco مقایسه شده است. تا آنجا که نویسنده مطلع است، Shoco تنها الگوریتم فشرده سازی اسکی دیگری است که برای فشرده سازی متن اسکی سفارشی شده است و همچنین قادر به فشرده سازی کدگذاری های دیگر است. راه حل های فشرده سازی ASCII که قادر به فشرده سازی سایر کدگذاری ها نیستند، مانند مقاله مورد بحث در 2.4.2، برای مقایسه در نظر گرفته نشده است، زیرا تعداد بسیار کمی فایلهای متنی امروزه فقط از کاراکترهای ASCII استفاده می کنند [۶۰]. در مقایسه ACM و Shoco، ACM نسبت فشرده سازی بالاتری نسبت به Shoco دارد – با استفاده از مدل فشرده سازی پیش فرض خود، اما ACM نسبت به Shocoنسبت کمی کمتر دارد – هنگام استفاده از یک مدل آموزش دیده. در بخش 6.2.1 ذکر شده است، مدلهای پیش فرض و آموزش دیده شوکو دارای مزایا و معایبی هستند. هر دو مدل قابل اجرا هستند و بسته به شرایط می توان از آنها استفاده کرد. بنابراین، این سوال تحقیق را نمی توان با بله یا خیر قطعی پاسخ داد.

### 7.2 کار آینده

کارهای آینده در TCS شامل یافتن یک ماژول کدگذاری هافمن است که می تواند خروجی دودویی را از ماژول کدگذاری دیکشنری فشرده کند. دستیابی به این امر می تواند قابلیت فشرده سازی TCS را با الگوریتم DEFLATE برابر کند. آزمایش الگوریتم هایی که توسط DEFLATE برای بهبود نسبت فشرده سازی استفاده نشده است، نیز کار آینده است.

برای افزایش نسبت فشرده سازی می توان پیشرفت هایی در ACM انجام داد. ACM می تواند فشرده سازی کاراکترهای غیر ASCII را بهبود بخشد، زیرا تکنیک فعلی بهینه نیست. افزودن قابلیت ایجاد مدل های آموزش دیده ممکن است نسبت فشرده سازی آن را نیز به میزان قابل توجهی افزایش دهد. با این پیشرفت ها، ممکن است ACM در صورت استفاده از پیش پردازنده، نسبت فشرده سازی برنامه های فشرده سازی عمومی را افزایش دهد.

همچنین می توان افزایش تأخیر ACM را بهبود داد. کد را می توان در C بازنویسی کرد و بهینه سازی را برای فرآیند فشرده سازی و خروج از فشرده سازی انجام داد. اگر ACM بیشتر بر زمان تأخیر تمرکز داشت تا نسبت فشرده سازی، استفاده از ACM در شرایطی که سرعت در اولویت بیشتری نسبت به ضریب فشرده سازی است، مفید خواهد بود. به عنوان مثال، می توان آن را در ترکیب با کمپرسور اسنپی مبتنی بر LZ77 [۱۰۷ و ۱۰۸] در زمینه و پس زمینه سرور استفاده کرد. Snappy نسبت فشرده سازی را از نظر سرعت به خطر می اندازد و در فایلهای متنی نسبت به فایلهای باینری به نسبت بهتری دست می یابد [۱۰۷]. نتایج جدول ۵ نشان می دهد که رمزگذار دیکشنری (LZW) می تواند نسبت فشرده سازی مناسبی را با ACM به عنوان پیش پردازنده بدست آورد، بنابراین استفاده از ACM به عنوان پیش پردازنده Snappy می تواند نسبت فشرده سازی را افزایش دهد.

**فهرست کتاب های مربوطه (کتابنامه)**


[1] K. Sayood, Introduction to data compression, 3rd ed. Amsterdam ; Boston: Elsevier, 2006.
[2] D. Salomon, Data compression the complete reference. New York: Springer, 2004.
[3] K. K. Shukla and M. V. Prasad, Lossy image compression: domain decomposition-based algorithms. London ; New York: Springer, 2011.
[4] «GNU Gzip». https://www.gnu.org/software/gzip/manual/gzip.html#Overview (opened Sep. 13th, 2020).
[5] «Untitled». PKWARE Inc., Opened: Sep. 13th, 2020. [Online]. Available at: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT.
[6] «WinRAR - WinRAR Documentation». https://documentation.help/WinRAR/ (opened Sep. 14th, 2020).
[7] D. Huffman, «A Method for the Construction of Minimum-Redundancy Codes», Proc. IRE, vol. 40, nr. 9, s. 1098–1101, sep. 1952, doi: 10.1109/JRPROC.1952.273898.
[8] S. Trost, «Alphabet and Character Frequency: English». https://www.sttmedia.com/characterfrequency-english (opened Sep. 15th, 2020).
[9] «Graphics Interchange Format (GIF) Specification», Opened: Sep. 14th, 2020. [Online]. Available at: https://www.w3.org/Graphics/GIF/specgif87.txt.
[10] J. Ziv and A. Lempel, «A universal algorithm for sequential data compression», IEEE Trans. Inform. Theory, vol. 23, nr. 3, pp. 337–343, May 1977, doi: 10.1109/TIT.1977.1055714.
[11] «RFC 1951 DEFLATE Compressed Data Format Specification ver 1.3». https://www.w3.org/Graphics/PNG/RFC-1951 (opened Sep. 15th, 2020).
[12] «Data Compression Conference - Home». https://www.cs.brandeis.edu/~dcc/index.html (opened Sep. 13th, 2020).
[13] A. Farina, G. Navarro, and J. R. Parama, "Boosting Text Compression with Word-Based Statistical Encoding", The Computer Journal, vol. 55, no. 1, pp. 111–131, Jan. 2012, doi: 10.1093/comjnl/bxr096.
[14] S. Wiedemann et al., «DeepCABAC: A Universal Compression Algorithm for Deep Neural Networks», IEEE J. Sel. Top. Signal Process., vol. 14, nr. 4, s. 700–714, May 2020, doi: 10.1109/JSTSP.2020.2969554.
[15] «Shoco - a fast compressor for short strings». https://ed-vonschleck.github.io/shoco/ (opened Sep. 14th, 2020).
[16] «Ed-von-Schleck/shoco», GitHub. https://github.com/Ed-vonSchleck/shoco/commits/master (opened Sep. 14th, 2020).
[17] N. Indurkhya and F. J. Damerau, Red., Handbook of natural language processing. Boca Raton, FL: Chapman & Hall/CRC, 2010.
[18] «torvalds/linux: Linux kernel source tree». https://github.com/torvalds/linux (opened Sep. 14th, 2020).
[19] «Untitled». https://api.github.com/repos/torvalds/linux (opened Sep. 14th, 2020).
[20] C. McAnlis and A. Haecky, Understanding compression: data compression for modern developers, First edition. Sebastopol, CA: O'Reilly, 2016.

[21] Institute of Electrical and Electronics Engineers, and Amirkabir University of Technology, Red., 2011 19th Iranian Conference on Electrical Engineering (ICEE 2011): Tehran, Iran, 17 - 19 May 2011. Piscataway, NJ: IEEE, 2011.
[22] J. Fowers, J.-Y. Kim, D. Burger, and S. Hauck, «A Scalable HighBandwidth Architecture for Lossless Compression on FPGAs», in 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, Canada, May 2015, pp. 52–59, doi: 10.1109/FCCM.2015.46.
[23] Y. Q. Shi and H. Sun, Image and Video Compression for Multimedia Engineering, Boca Raton, FL: CRC Press, 1999, pp. 140–141.
[24] M. Aggarwal and A. Narayan, «Efficient Huffman decoding», in Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101), Vancouver, BC, Canada, 2000, vol. 1, pp. 936–939, doi: 10.1109/ICIP.2000.901114.
[25] S. T. Klein, «Parallel Huffman Decoding with Applications to JPEG Files», The Computer Journal, vol. 46, nr. 5, pp. 487–497, May 2003, doi: 10.1093/comjnl/46.5.487.
[26] A. Langiu, «On parsing optimality for dictionary-based text compression—the Zip case», Journal of Discrete Algorithms, vol. 20, pp. 65– 70, May 2013, doi: 10.1016/j.jda.2013.04.001.
[27] «bzip2 and libbzip2, version 1.0.8». https://www.sourceware.org/bzip2/manual/manual.html (opened Sep. 14th, 2020).
[28] D. Pickell, «Qualitative vs Quantitative Data – What's the Difference?» https://learn.g2.com/qualitative-vs-quantitative-data (opened Sep. 15th, 2020).
[29] C. Read, Deductive and Inductive. Outlook Verlag, 2018.
[30] Merriam-Webster, «'Deduction' vs. 'Induction' vs. 'Abduction'». https://www.merriam-webster.com/words-at-play/deduction-vs-inductionvs-abduction (opened Sep. 15th, 2020).
[31] D. Kleiman, Red., The official CHFI exam 312-49 study guide: for computer hacking forensics investigators. Burlington, MA: Syngress Pub, 2007.
[32] R. Togneri and C. J. S. DeSilva, Fundamentals of information theory and coding design. Boca Raton: Chapman & Hall/CRC, 2002.
[33] «PNG Documentation». http://www.libpng.org/pub/png/pngdocs.html (opened Sep. 15th, 2020).
[34] R. Hoffman, Data Compression in Digital Systems. Boston, MA: Springer US, 1997.
[35] D. R. Hankerson, G. A. Harris, and P. D. Johnson, Introduction to information theory and data compression, 2nd ed. Boca Raton, Fla: Chapman & Hall/CRC Press, 2003.
[36] S. Senthil and L. Robert, « Text Compression Algorithms - A Comparative Study», IJCT, vol. 02, nr. 04, pp. 444–451, Dec. 2011, doi: 10.21917/ijct.2011.0062.
[37] S. A. A. Taleb et al., «Improving LZW Image Compression», EJSR, vol. 44, pp. 502–509, Aug. 2010.
[38] J. Ziv and A. Lempel, «Compression of individual sequences via variablerate coding», IEEE Trans. Inform. Theory, vol. 24, nr. 5, pp. 530–536, Sep. 1978, doi: 10.1109/TIT.1978.1055934.
[39] M. Atwal and L. Bansal, «Fast Lempel-ZIV (LZ'78) Algorithm Using Codebook Hashing», International Journal of Engineering and Technical Research (IJETR), pp. 220–223, Mar 2015.
[40] N. J. Larsson and A. Moffat, «Offline Dictionary-Based Compression», Opened: Sep. 14th, 2020. [Online]. Available at: http://www.larsson.dogma.net/dcc99.pdf.
[41] R. Bose, Information theory, coding and cryptography. New Delhi: Tata McGraw-Hill, 2008.
[42] M. Burrows and D. J. Wheeler, «A block-sorting lossless data compression algorithm», 1994.
[43] D. C. Wyld, M. Wozniak, ACITY, and Academy & Industry Research Collaboration Center, Red., Advances in computing and information technology: first international conference, ACITY 2011, Chennai, India, July 15 - 17, 2011 ; [the First International Conference on Advances in Computing and Information Technology] ; proceedings. Berlin: Springer, 2011.
[44] M. Arregoces and M. Portolani, Data center fundamentals. Indianapolis, Ind: Cisco, 2004.

[45] A. Carlsson and A. B. Miller, «Future Potentials for ASCII art», PostDigital Art - Proceedings of the 3rd Computer Art Congress, pp. 13–24, Nov 2012.
[46] D. Oluwade, «A Comparative Analysis and Application of the Compression Properties of Two 7-Bit Subsets of Unicode», 2012, doi: 10.1.1.645.8168.
[47] J. K. Korpela, Unicode explained, 1st ed. Sebastopol, CA: O'Reilly, 2006.
[48] B. Ediger, Advanced rails, 1st ed. Sebastopol, CA: O'Reilly, 2008.
[49] G. A. V. Pai, Data structures and algorithms: concepts, techniques and applications. New Delhi: Tata McGraw-Hill, 2008.
[50] International Conference on Intelligent Computing, D.-S. Huang, X.-P. Zhang, and G.-B. Huang, Eds., Advances in intelligent computing: International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005 : proceedings. Berlin; New York: Springer, 2005.
[51] A. Desoky and M. Gregory, "Compression of text and binary files using adaptive Huffman coding techniques," in Conference Proceedings '88., IEEE Southeastcon, Knoxville, TN, USA, 1988, pp. 660–663, doi: 10.1109/SECON.1988.194940.
[52] A. Bookstein and S. T. Klein, "Is Huffman coding dead?" Computing, vol. 50, no. 4, pp. 279–296, Dec. 1993, doi: 10.1007/BF02243872.
[53] J. Duda, "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding," arXiv:1311.2540 [cs, math], Jan. 2014, Accessed: Oct. 03, 2020. [Online]. Available: http://arxiv.org/abs/1311.2540.
[54] Meteficha, https://commons.wikimedia.org/wiki/File:Huffman_tree_2.svg. 2007.
[55] «The Computer Journal». https://academic.oup.com/comjnl (opened Oct. 29th, 2020).
[56] A. Moffat, "Word-based text compression," Softw: Pract. Exper., vol. 19, no. 2, pp. 185–198, Feb. 1989, doi: 10.1002/spe.4380190207.
[57] E. S. De Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates, "Direct pattern matching on compressed text," in Proceedings. String Processing and Information Retrieval: A South American Symposium (Cat. No.98EX207), Santa Cruz de La Sierra, Bolivia, 1998, pp. 90–95, doi: 10.1109/SPIRE.1998.712987.
[58] E. Silva de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates, "Fast and flexible word searching on compressed text," ACM Trans. Inf. Syst., vol. 18, no. 2, pp. 113–139, Apr. 2000, doi: 10.1145/348751.348754.
[59] «Information Technology Coding and Computing – ITCC». http://www.itcc.info/ (opened Nov. 3rd, 2020).
[60] J. Istle, P. Mandelbaum, and E. Regentova, "Online compression of ASCII files", ITCC, vol. 2, Apr. 2004, doi: 10.1109/ITCC.2004.1286559
[61] «What is the difference between "Legacy" and "Best method" compression?». https://support.winzip.com/hc/enus/articles/115011643067-What-is-the-difference-between-Legacy-and-Bestmethod-compression- (opened Nov. 5th, 2020).
[62] A. Håkansson, «Portal of Research Methods and Methodologies for Research Projects and Degree Projects», 2013, [Online]. Available at: https://www.semanticscholar.org/paper/Portal-of-Research-Methods-andMethodologies-forH%C3%A5kansson/e591fa1db4a633cd956cf06e204f82cffdc02e3e.
[63] B. B. Agarwal, S. P. Tayal, and M. Gupta, Software engineering & testing: an introduction. Sudbury, Mass: Jones and Bartlett, 2010.
[64] G. Guida, G. Lamperti, and M. Zanella, «Software Prototyping in Data and Knowledge Engineering». Heidelberg, Germany: Springer Netherlands, 1999.
[65] J. A. Highsmith, Agile software development ecosystems. Boston: Addison-Wesley, 2002.
[66] I. Sommerville, Software engineering, 8th ed. Harlow, England; New York: Addison-Wesley, 2007.
[67] «UTF-8 and Unicode Standards». http://www.utf-8.com/ (opened Sep. 16th, 2020).
[68] A. M. McEnery and R. Z. Xiao, «Character encoding in corpus construction.», in Developing Linguistic Corpora : A Guide to Good Practice, M. Wynne, Red. Oxford, UK: AHDS, 2005.

[69] «Extensible Markup Language (XML)». https://www.w3.org/XML/ (opened Sep. 16th, 2020).
[70] «JSON Syntax». https://www.w3schools.com/js/js_json_syntax.asp (opened Sep. 16th, 2020).
[71] «HTML Charset». https://www.w3schools.com/html/html_charset.asp (opened Sep. 16th, 2020).
[72] «Wikimedia Downloads». https://dumps.wikimedia.org/backupindex.html (opened Sep. 24th, 2020).
[73] «torvalds/linux». https://github.com/torvalds/linux/blob/master/kernel/sys.c (opened Sep. 24th, 2020).
[74] F. Nietzsche and T. Common, Thus spake Zarathustra. Ware: Wordsworth Editions, 1997.
[75] Dante Alighieri, B. Garavelli, and L. Magugliani, La divina commedia. Milano: BUR Rizzoli, 2012.
[76] Ainajushi, 豆棚閒話. Salt Lake City: Project Gutenberg, 2008.
[77] «Open license - Creative Commons». https://wiki.creativecommons.org/wiki/Open_license (opened Sep. 16th, 2020).
[78] D. Kaye, Loosely coupled: the missing pieces of Web services. Marin County, Calif: RDS Press, 2003.
[79] «Python 3.6.9». https://www.python.org/downloads/release/python369/ (opened Oct. 20th, 2020).
[80] «Standard encodings». https://docs.python.org/3/library/codecs.html#standard-encodings (opened Oct. 22nd, 2020).
[81] Welch, «A Technique for High-Performance Data Compression», Computer, vol. 17, nr. 6, pp. 8–19, Jun. 1984, doi: 10.1109/MC.1984.1659158.
[82] «meZip». https://github.com/aaronscode/meZip (opened Sep. 16th, 2020).
[83] «lzw-compression». https://github.com/biroeniko/lzw-compression (opened Sep. 16th, 2020).
[84] «python-lzw». https://github.com/joeatwork/python-lzw (opened Sep. 16th, 2020).
[85] «Rosetta Code». http://rosettacode.org/wiki/Rosetta_Code (opened Sep. 16th, 2020).
[86] «LZW compression». https://rosettacode.org/wiki/LZW_compression#Python (opened Sep. 16th, 2020).
[87] «gzip — Support for gzip files — Python 3.8.6rc1 documentation». https://docs.python.org/3/library/gzip.html (opened Sep. 16th, 2020).
[88] «bz2 — Support for bzip2 compression — Python 3.8.6rc1 documentation». https://docs.python.org/3/library/bz2.html (opened Sep. 16th, 2020).
[89] «zipfile — Work with ZIP archives — Python 3.8.6rc1 documentation». https://docs.python.org/3/library/zipfile.html (opened Sep. 16th, 2020).
[90] «LZ77-Compressor». https://github.com/manassra/LZ77-Compressor (opened Sep. 16th, 2020).
[91] «lzw-ab». https://github.com/dbry/lzw-ab (opened Sep. 16th, 2020).
[92] «ulz77». https://github.com/zooxyt/ulz77 (opened Sep. 16th, 2020).
[93] «huffman». https://github.com/emersonmde/huffman (opened Sep. 16th, 2020).
[94] «HuffmanEncoding». https://github.com/nrutkowski1/HuffmanEncoding (opened Sep. 16th, 2020).
[95] «hcomp». https://github.com/ronchauhan/hcomp (opened Sep. 16th, 2020).
[96] «dahuffman». https://github.com/soxofaan/dahuffman (opened Sep. 16th, 2020).
[97] «huffman». https://github.com/gyaikhom/huffman (opened Sep. 16th, 2020).
[98] «huffman». https://github.com/drichardson/huffman (opened Sep. 16th, 2020).
[99] «shoco». https://github.com/Ed-vonSchleck/shoco/blob/master/shoco_model.h (opened Oct. 9th, 2020).
[100] «Wikipedia:Database download». https://en.wikipedia.org/wiki/Wikipedia:Database_download#Dealing_with _compressed_files (opened Oct. 15th, 2020).
[101] T. Summers, “Hardware based GZIP compression, benefits and applications,” CORPUS, vol. 3, pp. 2–68, 2008.

[102] D. Belanger, K. Church, and A. Hume, “Virtual Data Warehousing, Data Publishing and Call Detail,” in Databases in Telecommunications, vol. 1819, W. Jonker, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 106–117.
[103] «7z Format». https://www.7-zip.org/7z.html (opened Oct. 18th, 2020).
[104] Li Jun and Li Ling, “Comparative research on Python speed optimization strategies” in 2010 International Conference on Intelligent Computing and Integrated Systems, Oct. 2010, pp. 57–59, doi: 10.1109/ICISS.2010.5655011.
[105] M. Salib, “Faster than C: Static type inference with Starkiller” in PyCon Proceedings, Mar. 2004, pp. 2–26.
[106] D. M. Beazley, Python essential reference, 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2009.
[107] «snappy». https://github.com/google/snappy (opened Nov. 9th, 2020).
[108] «format_description.txt». https://github.com/google/snappy/blob/master/format_description.txt (opened Nov. 9th, 2020).