به نام خدا

مخابرات دیجیتال

استاد بهنیا

تمرین سری ۱

نیما صمدی ـ ۹۷۱۰۲۰۱۱

۱۷ آبان ۱۴۰۰

---

۱ سوال اول

ابتدا مطابق توضیحات صورت سوال، توابع MyHuffman.m و MyLempelZiv.m را نوشتم. این تابع با کامنت‌گذاری مناسب و به صورت توابع مستقل از هم نوشته شده است تا قابلیت استفادهٔ مجدد داشته باشد و فهم آن راحت باشد.

۱) نرخ فشرده‌سازی بسته به تعداد سمبل‌ها متفاوت است و تغییر می‌کند. در اینجا خروجی یکی از دفعات اجرای کد دیده می‌شود:

```
Matlab Huffman coding compresion
    1.3683
Avg lenght: 8770

My Huffman coding compresion
    1.3683
Avg lenght: 8770

Lempel-Ziv coding compresion rate
    0.9494
Avg lenght: 12640
```

میزان فشرده‌سازی برای ۳ حالت بیان شده است. خروجی اول از توابع متلب برای کدگذاری هافمن است. خروجی دوم نتیجهٔ فشرده‌سازی تابع MyHuffman.m و خروجی آخر نتیجهٔ تابع MyLempelZiv.m است. البته در برخی مواقع نتیجهٔ کدگذاری Huffman تابع من و متلب اندکی با هم تفاوت دارد. به نظر من این به خاطر نحوهٔ نسبت دادن کدها می‌باشد. من هر بار دو سمبل با کمترین احتمال را انتخاب می‌کنم و آنها را ترکیب می‌کنم. سپس دوباره سمبل‌ها را مرتب کرده و این روند را ادامه می‌دهم. اما به نظر متلب به نحوهٔ دیگری این کار را انجام می‌دهد. به همین خاطر کدهای نسبت‌داده شده با هم تفاوت دارند هر چند این تفاوت در حد چند بیت است و گاهی نتیجهٔ حاصل از کد من از روش متلب بهتر است و برعکس.

همانطور که دیده می‌شود، میزان فشرده‌سازی کد هافمن من و متلب برابر 1.3713 است. میزان فشرده‌سازی کد Lempel-Ziv نیز برابر 0.9542 است. برای کد Lempel-Ziv برای نمایش اعداد سمبل‌ها نیاز به تعدادی بیت است. تعیین کردن این تعداد بیت‌ها از روی حداکثر شمارهٔ سمبل در رشتهٔ کد شده بدست می‌آید. تعداد کدها 756 تا بود که برای نمایش این تعداد نیاز به حداکثر 10 بیت است. البته ممکن است کمتر از این عدد مورد نیاز باشد. به همین خاطر تابع MyLempelZiv.m علاوه بر رشتهٔ کدشده، بیشینهٔ بیت‌های لازم برای نمایش عدد کنار هر سمبل را نیز خروجی می‌دهد. البته در عمدهٔ تعداد اجراها این عدد 10 بود.

۲) در مورد کدگذاری Lempel-Ziv صرفا دانستن طول بیت عدد کنار هر سمبل در رشتهٔ کدشده کافی است. برای 2000 کاراکتر تولیدی، باید 10 بیت برای آدرس جدا کرد. همچنین هر سمبل نیز با ۶ بیت نمایش داده می‌شود. بنابراین با دانستن این دو عدد و بدون اطلاعات دیگری می‌توان رشتهٔ دریافتی را دیکود کرد. البته باید درخت متناظر را ساخت که برای هر سمبل کدشده، حداکثر 16 بیت لازم است.

برای کد هافمن این تعداد اعضای دیکشنری بستگی به توزیع دارد. اگر رشتهٔ تصادفی کاملا از همان توزیع ورودی پیروی کند میزان بیت لازم برای دیکشنری هافمن ثابت خواهد بود. برای چندین اجرایی که انجام دادم به نظر این فرض صحیح است. میزان تعداد بیت لازم برای نمایش

دیکشنری برابر 197 bits می‌باشد. با اجرای کد سوال اول این مقدار در خروجی نوشته می‌شود. بنابراین اگر در گیرنده، 197 بیت را به ترتیب داشته باشیم، می‌توانیم کل رشتهٔ دریافتی را بازسازی کنیم.

۳) مطابق مباحث درس، می‌دانیم در صورتی که احتمال وقوع هر سمبل به صورت توانی از $\frac{1}{2}$ باشد، کد هافمن بهینه می‌شود و می‌تواند حد فشرده‌سازی به اندازهٔ آنتروپی منبع داشته باشد. بنابراین اگر توزیع احتمال به صورت توان‌هایی از $\frac{1}{2}$ تغییر کند، میزان فشرده سازی بیشینه می‌شود. بنابراین توزیع احتمال باید به صورت زیر شود:

$$P_X(i) = \begin{cases} (\dfrac{1}{2})^i & i = 1, 2, \cdots, 32 \\ (\dfrac{1}{2})^{32} & i = 33 \\ 0 & \text{O.W.} \end{cases}$$

می‌توانید مشاهده کنید که در این حالت جمع احتمال‌ها 1 است پس توزیع احتمال معتبری است. اگر آنتروپی را برای این توزیع احتمال حساب کنید:

$$H_X = 2\,(\frac{\text{bits}}{\text{symbol}})$$

اگر کد هافمن را استفاده کنیم، می‌توانیم به جای 6 بیت برای هر سمبل، به میزان فشرده‌سازی 2 بیت برای هر سمبل برسیم. یعنی میزان فشرده‌سازی برابر 3 خواهد شد. البته این محاسبات از نظر تئوری صحیح است. اما در موقع پیاده‌سازی نتیجهٔ متلب می‌تواند متفاوت باشد. علت این است که متغیرهای تصادفی در کامپیوتر، کاملا تصادفی نیستند. بنابراین توزیع احتمالی که بدست می‌آید کاملا منطبق بر این توزیعی که بیان کردم نیست.

نتیجهٔ حاصل از متلب برای این توزیع احتمال منبع به صورت زیر است:

```
Matlab Huffman coding compresion
    2.9903
Avg lenght: 4013

My Huffman coding compresion
    2.9903
Avg lenght: 4013

Lempel-Ziv coding compresion rate
    1.7354
Avg lenght: 6915
```

البته گاهی میزان فشرده‌سازی عددی بیشتر از 3 می‌شود که به خاطر خطای محاسبات است. همانطور که دیده می‌شود نتیجهٔ حاصل از کدینگ هافمن بسیار بهتر است. علت این موضوع بهینه‌بودن کدینگ هافمن نسبت به سایر کدینگ‌ها، از جمله Lempel-Ziv است.

۴) مشابه کاری که در بخش قبل انجام دادم اینجا عمل می‌کنم. البته در اینجا ابتدا توزیع احتمال رشتهٔ ورودی را پیدا می‌کنم. در بررسی متن داده شده متوجه شدم که مقادیر سمبل‌ها از 32 تا 121 عوض می‌شود. بنابراین تعداد کل سمبل‌ها 99 تا است. البته تعدادی از این سمبل‌ها اصلا رخ نمی‌دهند و احتمال صفر دارند اما در فرایند کد هافمن و Lempel-Ziv مشکلی ایجاد نمی‌کنند. توابعی که من نوشتم، در آنها فرض شده که تعداد رقم تمام سمبل‌ها یکسان است. مثلا تمام سمبل‌ها عدد دو رقمی هستند. به همین خاطر صرفا برای انجام کدینگ، ابتدا سمبل‌ها را در فایل file2.txt را به محدودهٔ $[100, 188]$ مپ می‌کنم. توجه کنید که این کار هیچ تاثیری در کدگذاری ندارد و صرفا برای این انجام شده است که تغییری در توابع انجام ندهم. برای دیکود کردن صرفا کافی است یک mapping (که صرفا یک شیفت است) را انجام دهیم. پس از mapping، احتمال رخداد هر سمبل را با تابع CalDist.m محاسبه می‌کنم. در نهایت سمبل‌ها و توزیع آنها را به توابع محاسبهٔ کدینگ هافمن و Lempel-Ziv ارسال می‌کنم و نتایج را در اینجا بیان می‌کنم. توجه کنید که دوباره برای مقایسهٔ نتیجهٔ هافمن، از خود توابع متلب نیز جداگانه استفاده کرده‌ام تا نتایج را با هم مقایسه کنم.

```
Matlab Huffman coding compresion
    1.4121
Avg lenght: 19860

My Huffman coding compresion
    1.4121
Avg lenght: 19860

Lempel-Ziv coding compresion rate
    1.1775
Avg lenght: 23817
```

همانطور که دیده می‌شود، نتیجهٔ هافمن متلب و تابع خودم کاملا یکسان است.
همانطور که از مقایسهٔ نرخ فشرده‌سازی دیده می‌شود، کدینگ هافمن عملکرد بهتری دارد. علت آن بهینه بودن این روش کدینگ است. به علاوه در روش Lempel-Ziv مزیت اصلی عدم نیاز به محاسبهٔ توزیع احتمال و ذخیرهٔ دیکشنری در سمت گیرنده است. همچنین میزان فشرده‌سازی این روش کدینگ، کاملا به طول رشتهٔ ورودی بستگی دارد. به عبارت دیگر هر چقدر طول رشتهٔ ورودی بزرگتر شود، این کدینگ بهتر عمل می‌کند چون به جای ارسال سمبل‌های طولانی‌تر، صرفا کافی است یک عدد (معادل آدرس سمبل‌های قبلی) ارسال کنیم که مرتبهٔ تعداد بیت این عدد با لگاریتم طول رشته زیاد می‌شود. در حالی که اگر خود رشتهٔ ورودی را ارسال کنیم، باید به اندازهٔ خود آن بیت استفاده کنیم. پس در واقع یک مقایسه بین رشد تابع لگاریتمی و خطی داریم که برای مقادیر بزرگ، تابع خطی بسیار بزرگتر از لگاریتم می‌شود و این ایدهٔ اصلی روش Lempel-Ziv است.

۲ سوال دوم