علی آراسته ۹۶۱۰۱۱۶۵

تئوری اطلاعات و کدینگ - گزارش پروژه

سوال ۱: کد هافمن

آ) ابتدا با استفاده از تابع ()ord، کد اسکی مربوط به هر کاراکتر را بدست می‌آوریم و سپس با استفاده از توابع ()bin و ()zfill، کد اسکی هر کاراکتر را به یک دنباله باینری ۷ بیتی تبدیل می‌کنیم.

خروجی به صورت زیر بدست می‌آید:

[۱۰۳, ۱۱۱, ۳۲, ۱۰۳, ۱۱۱, ۳۲, ۱۰۳, ۱۱۱, ۱۰۰, ۱۲۲, ۱۰۵, ۱۰۸, ۹۷]

۱۱۰۰۱۱۱۱۱۰۱۱۱۱۰۱۰۰۰۰۰۱۱۰۰۱۱۱۱۱۰۱۱۱۱۰۱۰۰۰۰۰۱۱۰۰۱۱۱۱۱۰۱۱۱۱۱۱۱۰۰۱۰۰۱۱۱۱۰۱۰۱۱۰۱۰۰۱۱۱۰۱۱۰۰۱۱۰۰۰۰۱

ب) با محاسبه تعداد دفعات تکرار هر حرف وتقسیم آن بر تعداد کل حروف (۱۳)، فرکانس تکرار هر حرف را بدست می‌آوریم.

خروجی به صورت زیر بدست می‌آید:

{'a': 0.07692307692307693, 'b': 0.0, 'c': 0.0, 'd': 0.07692307692307693, 'e': 0.0, 'f': 0.0, 'g': 0.23076923076923078, 'h': 0.0, 'i': 0.07692307692307693, 'j': 0.0, 'k': 0.0, 'l': 0.07692307692307693, 'm': 0.0, 'n': 0.0, 'o': 0.23076923076923078, 'p': 0.0, 'q': 0.0, 'r': 0.0, 's': 0.0, 't': 0.0, 'u': 0.0, 'v': 0.0, 'w': 0.0, 'x': 0.0, 'y': 0.0, 'z': 0.07692307692307693, ' ': 0.15384615384615385}

ج) با استفاده از دو تابع ()encoding_huffman_tree و ()decoding_huffman_tree، کد هافمن مناسب را تولید می‌کنیم.

خروجی به صورت زیر بدست می‌آید:

{'a': '11010', 'b': '11011111111111111111110', 'c': '11011111111111111111111', 'd': '000', 'e': '1101111111111111111110', 'f': '11011111110', 'g': '01', 'h': '11011110', 'i': '1100', 'j': '110110', 'k': '1101110', 'l': '0011', 'm': '110111110', 'n': '1101111110', 'o': '10', 'p': '110111111110', 'q': '1101111111110', 'r': '11011111111110', 's': '110111111111110', 't': '1101111111111110', 'u': '11011111111111110', 'v': '110111111111111110', 'w': '1101111111111111110', 'x': '11011111111111111110', 'y': '11011111111111111111110', 'z': '0010', ' ': '111'}

د) با استفاده از تابع ()huffman_compressor، دنباله مورد نظر را با استفاده از کد هافمن فشرده می‌کنیم.

خروجی به صورت زیر بدست می‌آید:

۰۱۱۰۱۱۱۰۱۱۰۱۱۱۰۱۱۰۰۰۰۰۰۱۰۱۱۰۰۰۰۱۱۱۱۰۱۰

- در صورتی که می‌دانستیم هر ۷ بیت، معرف یک کاراکتر است، این کار امکان‌پذیر بود. در واقع استفاده از کد هافمن روی کاراکترها، حالت خاصی از کد هافمن قالبی است که در آن طول قالب برابر ۷ است.
- خیر این چنین نیست. اگرچه طول دنباله معرف جمله مورد نظر، در حالت استفاده از کد هافمن کوتاه‌تر است، حجم دیتای ذخیره شده زیاده شده است. علت این است که در حالت استفاده از کد هافمن، گیرنده و فرستنده باید جدولی

حاوی کد معرف هر کاراکتر را ذخیره کنند و در نتیجه در نتیجه استفاده از کد هافمن برای فشرده‌کردن جملات و دنباله‌های کوتاه بهینه نیست. البته در صورت مد نظر قرار دادن فضای لازم برای ذخیره کدهای اسکی، باید بررسی دقیق‌تری صورت گیرد.

ه) با استفاده از تابع n_p_sequences()، یک دیکشنری حاوی تمام دنباله‌های ممکن به طول n به عنوان key و احتمال وقوع هر یک به عنوان value می‌سازیم.

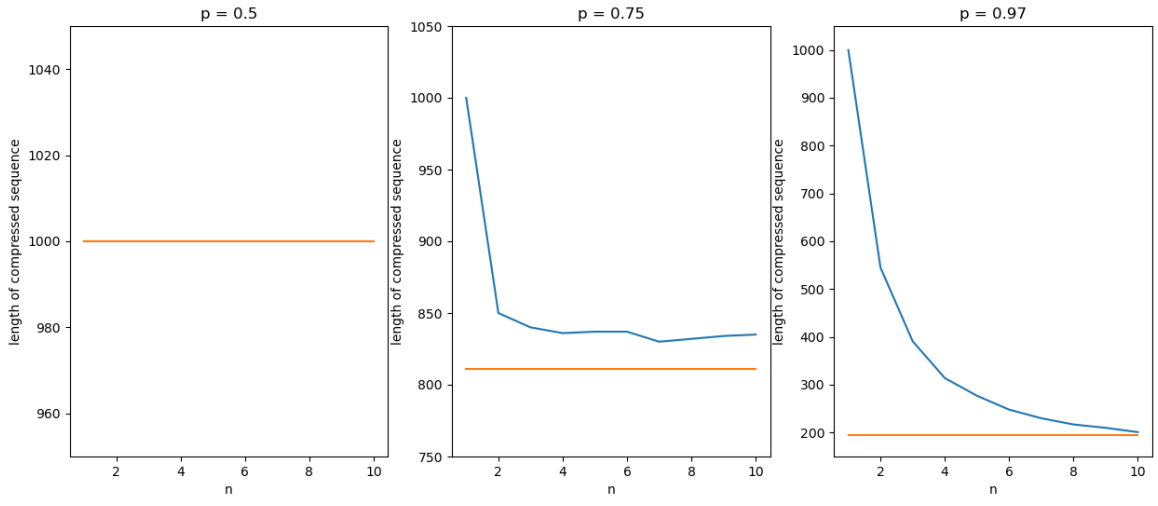
و) مانند قسمت ج، با استفاده از دو تابع encoding_huffman_tree() و decoding_huffman_tree()، کد هافمن مناسب را تولید می‌کنیم.

ز) ابتدا برای هر n، کوچک‌ترین عدد بزرگتر مساوی ۱۰۰۰ را که بر n بخش‌پذیر است، محاسبه می‌کنیم. سپس دنباله‌ای باینری به طول ماکسیمم مقادیر بدست آمده در قسمت قبل به صورت i.i.d. از توزیع برنولی با پارامتر p بدست می‌آوریم. در آخر با استفاده از کد هافمن تولید شده با پارامترهای n و p، دنباله تصادفی ایجاد شده را فشرده می‌کنیم.

لازم به ذکر است به دلیل محدودیت‌های سیستم در محاسبه توابع بازگشتی (وجود محدودیت در حداکثر تعداد بازگشت) محاسبه کد هافمن برای $n \geq 11$ امکان‌پذیر نبود.

خروجی به صورت زیر بدست می‌آید:

۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰	۱۰۰۰
۱۰۰۰	۸۵۰	۸۴۰	۸۳۶	۸۳۷	۸۳۷	۸۳۰	۸۳۲	۸۳۴	۸۳۵
۱۰۰۰	۵۴۵	۳۹۱	۳۱۴	۲۷۷	۲۴۸	۲۳۰	۲۱۷	۲۱۰	۲۰۱



- از آنجایی که دنباله مورد نظر به صورت تصادفی تولید می‌شود، رفتار کاهشی طول دنباله فشرده‌شده با افزایش n، یکنوا نیست؛ اما به طور کلیمی‌توان گفت هر چقدر n بزرگ‌تر باشد، طول کد فشرده شده کمتر می‌شود.

ح) با توجه به مفاهیم فراگرفته شده، باند پایین تئوری اطلاعاتی برابر $H_b(p)$ است. این مقدار در نمودارهای قسمت قبل رسم شده است.

ط) میان این دو روش دو تفاوت وجود دارد:

- در روش اول از کد هافمن روی کاراکترها (به طور معادل قالب‌های ۷ بیتی) استفاده شد. حال آن که در روش دوم طول‌های مختلف قالب مورد بررسی قرار گرفت
- در روش اول در هنگام تولید کد هافمن، از توزیع تجربی استفاده شد. حال آن که در روش دوم، توزیع احتمال به طور دقیق محاسبه شد.

سوال ۲: الگوریتم Blahut_Arimoto

با استفاده از توضیحات داده، الگوریتم Blahut_Arimoto را در تابع Blahut_Arimoto() پیاده‌سازی می‌کنیم.

خروجی برای توزیع کانال داده شده به صورت زیر بدست می‌آید:

$$r(x) \cong [0.45, 0, 0.55]$$

$$c \cong 0.0905$$