**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

****

**BÁO CÁO TỔNG KẾT**

**MỞ RỘNG TRUYỀN SỐ LIỆU VÀ MẠNG**

**Đề tài:**

Ứng dụng Huffman trong việc đọc mảng ký hiệu và xác suất, xuất ra cây Huffman, tính toán hiệu suất.

**Giáo viên hướng dẫn:** Thầy Phạm Quang Thái

**Thành viên:**

Trương Văn Lĩnh 1812840

Thái Quang Nguyên 1813294

Nguyễn Minh Thuận 1814232

Thành phố Hồ Chí Minh-Tháng 11-Năm 2020

Mục lục

[I. Cơ sở lý thuyết: 1](#_Toc59059288)

[1. Giới thiệu: 1](#_Toc59059289)

[2. Huffman coding làm việc thế nào? 1](#_Toc59059290)

[II. Tiến hành bài toán với ngôn ngữ Python. 3](#_Toc59059291)

[1. Cấu trúc thư mục & thư viện cần có: 3](#_Toc59059292)

[**a.** **Cấu trúc thư mục:** 4](#_Toc59059293)

[**b.** **Thư viện cần có:** 4](#_Toc59059294)

[2. Đọc symbol và xác suất tương ứng từ file EXCEL: 4](#_Toc59059295)

[3. Giải thuật Huffman: 5](#_Toc59059296)

[4. Xuất và lưu file LATEX: 9](#_Toc59059297)

[III. Tài liệu tham khảo: 15](#_Toc59059298)

1. **Cơ sở lý thuyết:**

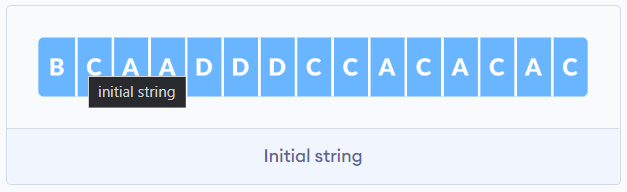
**Giới thiệu:**

Huffman coding là một kĩ thuật nén dữ liệu để giảm kích cỡ của nó mà không mất bất kì chi tiết nào. Nó được phát triển lần đầu bởi David Huffman.

Huffman coding rất hữu dụng để nén dữ liệu mà các kí tự đó thường xuyên xảy ra.

**Huffman coding làm việc thế nào?**

Giả sử ta có một chuỗi các kí tự như dưới đây được gửi qua mạng.



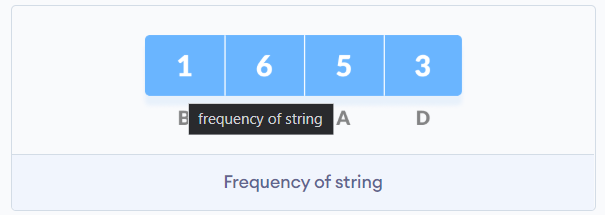
Mỗi ký tự có 8 bits. Có tổng cộng 15 kí tự, do đó, có 120 bit được yêu cầu để gửi chuỗi này đi.

Sử dụng Huffman Coding, ta có thể làm kích cỡ nó giảm đi.

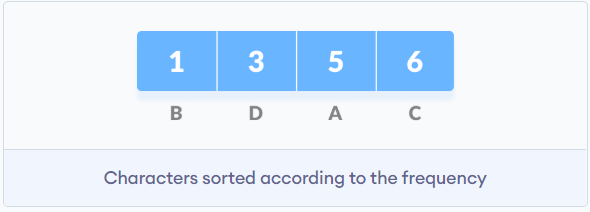
Huffman Coding đầu tiên tạo ra 1 cây sử dụng tần số của các kí tự và sau đó tạo ra code cho mỗi kí tự. Một khi dữ liệu được mã hóa, nó cần được giải mã, giải mã cũng sử dụng cây này.

***Các bước thực hiện:***

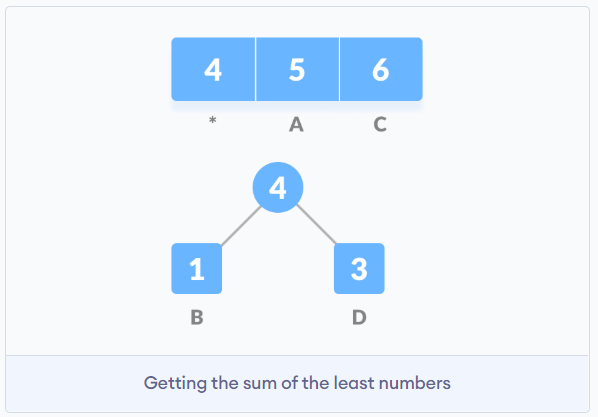
* + 1. Tính toán tần số của mỗi kí tự trong chuỗi.



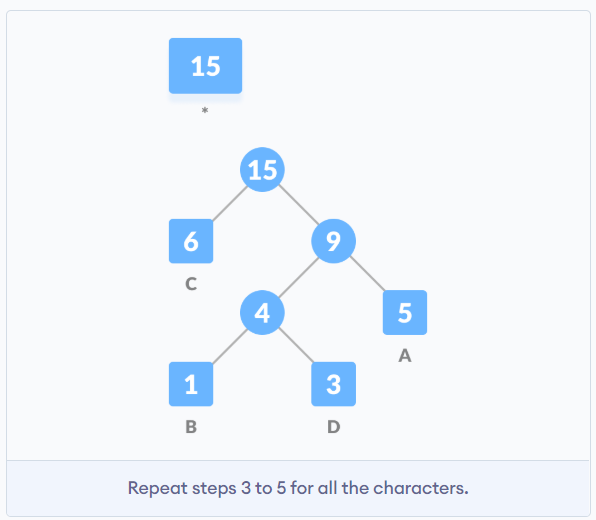
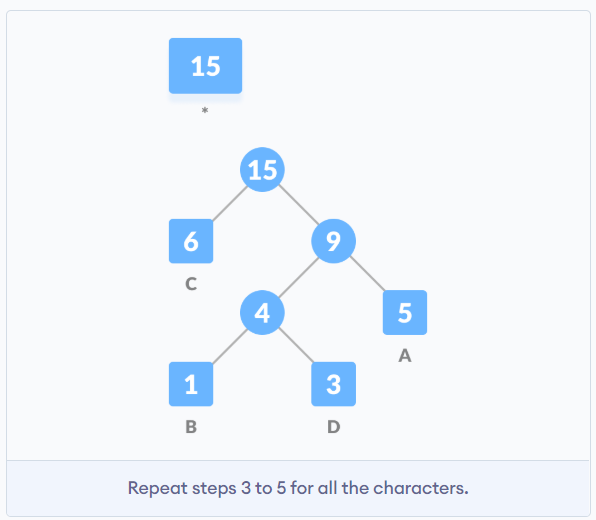
* + 1. Sắp xếp các kí tự tăng dần theo tần số. Nó được chứa trong priority Queue.



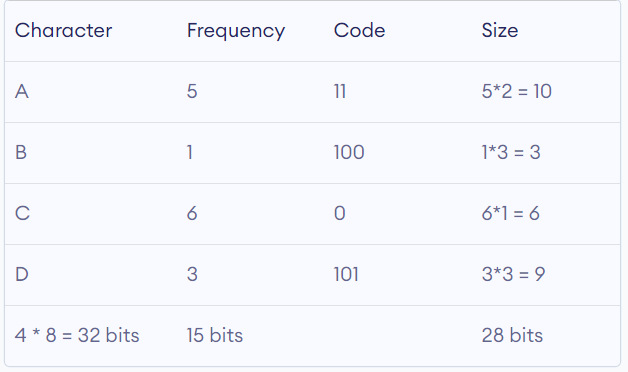
* + 1. Tạo ra một nút trống z. Để tần số nhỏ nhất ở bên trái của z và nhỏ thứ hai ở bên phải. Giá trị của z là tổng của 2 tần số này.



* + 1. Thêm node z vào cây.
    2. Lặp lại cho tất cả các kí tự.

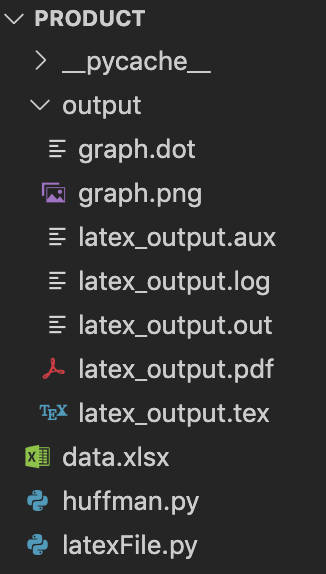


Để gửi chuỗi đó qua mạng, chúng ta phải gửi cây cũng như mã được nén đi. Kích cỡ tổng cộng cho ở bảng dưới.



Như vậy, kích cỡ đã giảm đi 75 bit.

1. **Tiến hành bài toán với ngôn ngữ Python.**
   1. **Cấu trúc thư mục & thư viện cần có:**

**Cấu trúc thư mục:**

Data sẽ được trích xuất từ file EXCEL ở thư mục ngoài cùng từ file ***data.xlsx***, sau đó thông qua các hàm xử lý chính nằm trong ***huffman.py***. Ngoài ra còn có một file thư viện ***latexFile.py*** gồm các hàm để làm việc với định dạng file .tex.

Tất cả các output ra bao gồm đồ thị cây Huffman ***graph.png*** hay file ***latex\_output.pdf*** sinh ra từ ***latex\_output.tex*** được lưu trong folder con ***output***.

*Hình 1.1 Cấu trúc thư mục của project*

**Thư viện cần có:**

* queue: dùng để sắp xếp các symbol dựa theo xác suất của chúng.
* math: dùng để thực hiện các lệnh tính toán toán học cần thiết
* os và subprocess: dùng để chạy các command tương tự như chạy trên terminal
* openpyxl: dùng để đọc các giá trị từ file excel
* sympy: dùng để xuất ra các hàm LATEX phục vụ cho file .tex
* graphviz: dùng để vẽ cây Huffman
  1. **Đọc symbol và xác suất tương ứng từ file EXCEL:**

Dữ liệu đầu vào:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Symbol** | A | B | C | D | E | F | G | H | I | J | K |
| **Probability** | 0.21 | 0.17 | 0.15 | 0.12 | 0.1 | 0.06 | 0.049 | 0.051 | 0.04 | 0.03 | 0.02 |

Ta sử dụng openpyxl để lấy dữ liệu từ file excel. Sử dụng vòng lặp và các hàm tương ứng để chuyển dữ liệu trong file excel thành mảng để ta xử lý.

probability\_CS = 0

for row in ws.iter\_rows(min\_col=1, max\_col=2, min\_row=2):

row = [cell.value for cell in row]

freq.append((row[0], str(row[1])))

probability\_CS += row[0]

if round(probability\_CS, 5) == 1:

print("Data satisfied")

else:

print("Data not satisfied")

os.\_exit(0)

Biến ***probability\_CS*** là biến để ta kiểm tra bộ dữ liệu có phù hợp hay không bằng cơ chế tổng tất cả các xác xuất của các symbol nếu bằng 1 thì bộ dữ liệu này phù hợp, nếu khác 1 thì bộ dữ liệu này không phù hợp.

Trong đó, freq là một mảng 2 chiều gồm có 2 cột, mỗi lần quét qua từng xác suất và symbol tương ứng thì ta sẽ thêm 1 hàng vào mảng freq. Ở đây để chắc chắn symbol là một chuỗi string, ta sử dụng hàm str().

* 1. **Giải thuật Huffman:**

Đầu tiên, ta tạo một class với tên gọi HuffmanNode với mục đích để lưu các object là các node của cây Huffman như sau:

class HuffmanNode(object):

def \_\_init\_\_(self, left=None, right=None):

self.left = left

self.right = right

def children(self):

return (self.left, self.right)

Một object thuộc class *HuffmanNode* sẽ có hai thuộc tính ***self.left*** và ***self.right***, tương ứng với node con bên trái và bên phải của nó.

Tiếp đến, ta tạo hàm ***encode()*** như sau:

def encode(frequencies):

p = queue.PriorityQueue()

for item in frequencies:

p.put(item)

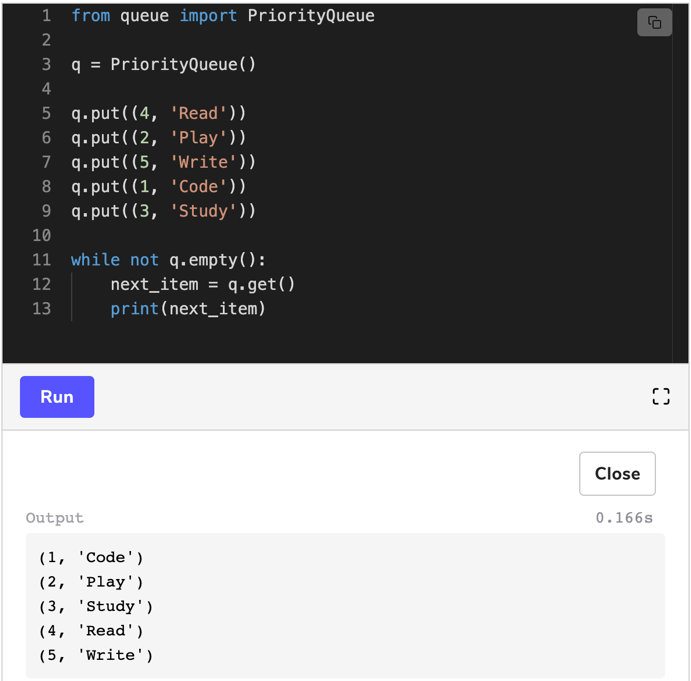
while p.qsize() > 1:

Left, Right = p.get(), p.get()

node = HuffmanNode(Left, Right)

p.put((Left[0] + Right[0], node))

return p.get()

Biến p được gán bằng ***queue.PriorityQueue()***, nhờ vậy khi gọi hàm ***p.put()*** các giá trị đặt vào p sẽ được sắp xếp theo thứ tự. Ví dụ đơn giản như hình dưới:

*Hình 3.1. Ví dụ về PriorityQueue*

Ở đây, đối số để ta sử dụng hàm ***p.put()*** chính là một vector chứa các symbols và xác suất tương ứng của chúng. Ví dụ như biến freq bên trên.

freq = [(0.21,'A'),(0.17,'B'),(0.15,'C'),(0.12,'D'),

(0.1,'E'),(0.06,'F'),(0.049,'G'),(0.051,'H'),

(0.04,'I'),(0.03,'J'),(0.02,'K')]

Các giá trị số chính là xác suất, và giá trị ***string*** chính là symbols tương ứng.

Trở lại với hàm ***encode()***, sau khi đã lưu được các symbols vào p theo thứ tự xác suất, ta tiến hành tạo các *HuffmanNode* (tức là một node được tạo thành từ 2 node con):

while p.qsize() > 1:

Left, Right = p.get(), p.get()

node = HuffmanNode(Left, Right)

p.put((Left[0] + Right[0], node))

return p.get()

Sử dụng hàm ***p.get()*** ta lấy ***(pop)*** được giá trị trên cùng của hàng ưu tiên, tức là giá trị với xác suất nhỏ nhất, lưu vào biến *Left*. Tương tự giá trị nhỏ nhất tiếp theo ta lưu vào biến *Right*. Sau đó tạo một object thuộc class *HuffmanNode* để lưu một node mới, với hai node con là Left và Right. Node mới tạo này được lưu trở lại vào hàng ưu tiên ***PriorityQueue***.

Như vậy sau khi đã ***pop*** hai node có xác suất nhỏ nhất, ta lưu lại một node tổng. Số node trong hàng được giảm đi 1. Tiếp tục thực hiện tác vụ tạo *HuffmanNode* đến khi chỉ còn một node cuối cùng chính là node tổng (root).

Tiếp đến, ta bổ sung trong class *HuffmanNode* một hàm đệ quy để quét qua cây Huffman, đồng thời gán từ mã cho các lá cuối cùng của mỗi nhánh. Giải thuật được thực hiện như sau:

def preorder(self, path=None):

global average\_length

if path is None:

path = []

if self.left is not None:

if isinstance(self.left[1], HuffmanNode):

self.left[1].preorder(path+['0'])

else:

print(self.left, path+[0])

average\_length += self.left[0]\*(len(path)+1)

if self.right is not None:

if isinstance(self.right[1], HuffmanNode):

self.right[1].preorder(path+['1'])

else:

print(self.right, path+[1])

average\_length += self.right[0]\*(len(path)+1)

Giá trị trả về của hàm ***encode()*** chính là node tổng của cây Huffman, hiển nhiên sẽ có xác suất tương ứng bằng 1. Ta sẽ dùng hàm ***preorder()*** để bắt đầu quét từ node này.

Đầu tiên gọi hàm theo cú pháp:

node = encode(freq)

node[1].preorder()

Sau khi hàm ***preorder()*** được gọi đầu tiên từ node tổng, khởi tạo một vector một chiều ***path[]*** để chứa đựng đường đi trong lúc quét, phục vụ cho việc gán từ mã cho các lá cuối cùng mỗi nhánh.

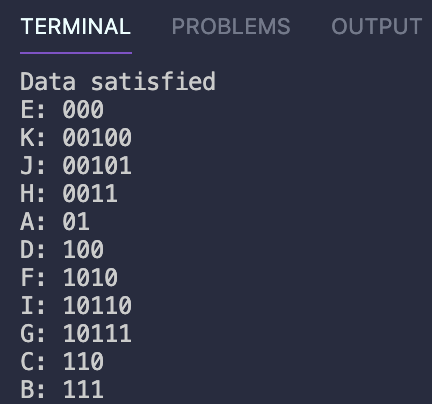
Lần lượt check xem node con bên trái (***self.left***) và bên phải (***self.right***) của node hiện tại có phải là object *HuffmanNode* không thông qua hàm ***isinstance()***. Lưu ý rằng các lá cuối cùng của mỗi nhánh không phải là object *HuffmanNode*, object này được tạo dựa trên sự kết hợp của 2 lá, hoặc 2 object con khác. ***self.left[0] là xác suất của node, self.left[1] là object HuffmanNode***.

Nếu node con bên trái hoặc phải là object HuffmanNode, tức là node đó là tạo thành từ hai node con, ta di chuyển đến node đó để xét tiếp, đồng thời cộng dồn vector path[] với [0] hoặc [1] ứng với bên trái hoặc phải. Thực hiện dựa vào hàm ***self.left[1].preorder(path+[0])***.

Quá trình đệ quy được tiếp tục cho đến khi node con bên trái (*self.left*) của node hiện tại không phải là *HuffmanNode*, tức nó là lá cuối cùng. Khi đó ta xuất ***path + [‘0’]*** ra màn hình, chính là từ mã của node con đó, đồng thời cộng dồn vào chiều dài trung bình từ mã một lượng bằng ***xác suất của lá (self.left[0]) nhân với chiều dài của từ mã (len(path)+1)***.

Do đã đến lá cuối cùng bên trái, ta không nhảy vào hàm ***preorder()*** khác mà xét tiếp node bên phải, một cách tương tự. Nếu node con bên phải (***self.right***) là một object HuffmanNode, ta lại nhảy vào hàm ***preorder()***, nếu ***self.right*** là một lá, xuất ***path+[‘1’]*** ra màn hình.

Khi đã chạy hết dòng cuối của hàm ***preorder()***, con trỏ PC nhảy về dòng tiếp theo khi ta gọi hàm ***preorder()*** trước đó, tức là nhảy về các node gần root hơn và bắt đầu xét tiếp. Cứ như thế đến khi hết hàm ***preorder()*** đầu tiên mà ta gọi từ node tổng. Cuối cùng, ta xuất ra màn hình được từ mã của tất cả các lá, đồng thời tính ra được chiều dài từ mã trung bình. Kết quả chạy code được minh hoạ bằng hình sau:



*Hình 3.2. Kết quả code xuất từ mã dựa trên thuật toán hồi quy*

Ngoài ra, ta có thể dễ dàng tính được Entropy và hiệu suất như sau:

for item in freq:

entropy += item[0]\*math.log2(1/item[0])

efficiency = entropy/average\_length

* 1. **Xuất và lưu file LATEX:**

Để xuất được file .pdf từ file .tex, ta cần cài đặt TeX tuỳ theo hệ điều hành sử dụng. Ở đây tác giả sử dụng hệ điều hành MacOS nên tải MacTeX.

Trước hết ta code một thư viện chứa các hàm cần thiết để truy xuất file .tex như sau:

*class* latexClass(object):

def \_\_init\_\_(self, addr=None):

self.addr = addr

def init(self):

*with* open(self.addr,'w') *as* f:

f.write('\\documentclass{article}\n')

f.write('\\usepackage[utf8]{vietnam}\n')

f.write('\\usepackage[utf8]{inputenc}\n')

f.write('\\usepackage{amsmath}\n')

f.write('\\usepackage{graphicx}\n')

f.write('\\usepackage{tikz}\n')

f.write('\\usepackage{forest}\n')

f.write('\\usetikzlibrary{shapes}\n')

f.write('\\usepackage{hyperref}\n')

f.write('\\newcounter{figureCnt}\n\\newenvironment{figureCnt}[1][]')

f.write('{\\refstepcounter{figureCnt}\\par\\medskip\\textbf{Figure~\\thefigureCnt. #1} \\rmfamily}{\\medskip}')

f.write('\\title{TRUYEN SO LIEU VA MANG}\n\n')

f.write('\\begin{document}\n\n')

f.write('\\begin{center}\n')

f.write('\\Large\n\\textbf{Truyền Số Liệu và Mạng Mở Rộng}\n')

f.write('\\large\n\\end{center}\n\n')

def end(self):

*with* open(self.addr,'a') *as* f:

f.write('\\end{document}\n')

def init\_item(self):

*with* open(self.addr,'a') *as* f:

f.write('\\begin{itemize}\n')

def end\_item(self):

*with* open(self.addr,'a') *as* f:

f.write('\\end{itemize}\n')

def add\_table(self, \_array=None):

*with* open(self.addr,'a') *as* f:

f.write('\\begin{table}[h!]\n')

f.write('\\centering')

f.write('\\begin{tabular}{||c c c||}\n')

f.write('\\hline\n Symbol & Probability & Code \\\ [0.5ex]\n\\hline\n')

*for* row *in* \_array:

f.write('\\hline\n')

f.write(row[0])

f.write(' & ')

f.write(row[1])

f.write(' & ')

f.write(row[2])

f.write(' \\\ \n')

f.write('\\hline\n')

f.write('\\end{tabular}\n')

f.write('\\end{table}\n\n')

def init\_tree(self):

*with* open(self.addr,'a') *as* f:

f.write('\\begin{center}\n')

f.write('\\begin{forest}\n')

f.write('[Root ')

def addNode\_tree(self, \_text=None, \_type=None):

*with* open(self.addr,'a') *as* f:

f.write('[')

f.write(\_text)

*if* \_type == "NODE":

f.write(', for tree={circle,draw}')

*elif* \_type == "LEAF":

f.write(', for tree={rectangle,draw,fill=green}')

f.write(', rotate={-90}')

def closeNode\_tree(self):

*with* open(self.addr,'a') *as* f:

f.write(']')

def end\_tree(self):

*with* open(self.addr,'a') *as* f:

f.write(']')

f.write('\\end{forest}\n')

f.write('\\end{center}\n')

def addString(self, input\_string=None, item=None, enter=0, math=None):

*with* open(self.addr,'a') *as* f:

*if* math is not None:

f.write('\\[\n')

*if* item is not None:

f.write('\\item ')

f.write(input\_string)

*if* math is not None:

f.write('\n\\]')

f.write('\n')

def add\_figure(self, \_caption=None):

*if* \_caption is not None:

*with* open(self.addr,'a') *as* f:

f.write('\\begin{center}')

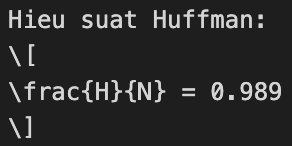
f.write('\\begin{figureCnt}\n')

f.write(\_caption)

f.write('\n\\end{figureCnt}\n')

f.write('\\end{center}')

Trong đó hàm ***init()*** để khởi tạo file, ***end()*** hay các hàm còn lại để thêm nội dung vào file với option *‘a’.* Sử dụng module **{amsmath}** trong latex để biểu diễn các biểu thức toán học. Ở hàm ***addString()*** phía trên, ta sẽ dùng để append thêm vào file .tex, có những option như *item* và *math* nếu ta muốn đưa dữ liệu vào theo kiểu gì. Cụ thể, các biểu thức toán học sẽ được đặt giữa hai dấu ngoặc vuông như sau:



Ngoài ra còn có hàm ***add\_table()*** để tạo bảng gồm 3 cột, đối số là một vector 2 chiều có sẵn. Hàm ***add\_figure()*** được dùng để thêm chú thích cho hình hoặc bảng… với số đi kèm tự động đếm lên 1 sau mỗi lần được gọi. Ta thực hiện được điều này bằng cách khai báo thêm một counter trong file latex theo cú pháp như sau:

f.write('\\usepackage[utf8]{inputenc}\n')

f.write('\\newcounter{figureCnt}\n\\newenvironment{figureCnt}[1][]')

f.write('{\\refstepcounter{figureCnt}\\par\\medskip\\textbf{Figure~\\thefigureCnt. #1} \\rmfamily}{\\medskip}')

Qua đó, ta có thể chỉnh sửa code cho phù hợp với việc xuất file .tex. Có thể search thư viện sympy để tìm ra hàm thích hợp để viết biểu thức, *sympy* hỗ trợ convert sang latex markup bằng hàm *latex()*.

#*##Write to tex file*

latexFile.init()

rootNODE = myMapFunc(node[1])

latexFile.init\_tree()

rootNODE.preorder()

latexFile.end\_tree()

latexFile.add\_figure("The Huffman Tree")

latexFile.add\_table(\_array=output\_freq)

latexFile.add\_figure("Bảng từ mã các nodes")

i, H, N = symbols('i H N')

latexFile.addString(input\_string="Entropy:")

entropy\_exp = Eq(Sum(Indexed('p',i)\*log(1/(Indexed('p',i)),2),(i,0,'k')), round(entropy,3))

latexFile.addString(input\_string=latex(entropy\_exp), math=True)

latexFile.addString(input\_string="Chiều dài TB từ mã:")

tuma\_exp = Eq(Sum(Indexed('p',i)\*Indexed('N',i),(i,0,'k')), round(average\_length,3))

latexFile.addString(input\_string='N = '+latex(tuma\_exp), math=True)

latexFile.addString(input\_string="Hiệu suất Huffman:")

efficiency = entropy/average\_length

efficiency\_exp = Eq((H/N), round(efficiency,3))

latexFile.addString(input\_string=latex(efficiency\_exp), math=True)

latexFile.end()

Và cả ở hàm *preorder()* trong class *HuffmanNode*, ta sẽ khởi tạo và đóng các node trong cây ở file latex bằng 2 hàm ***addNode\_tree()*** và ***closeNode\_tree()****.*

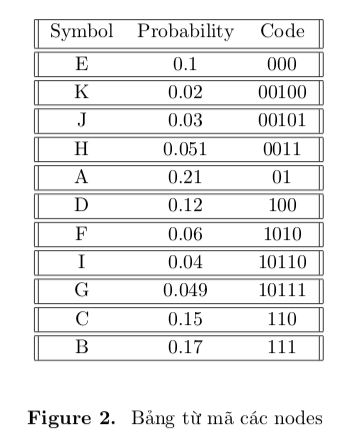
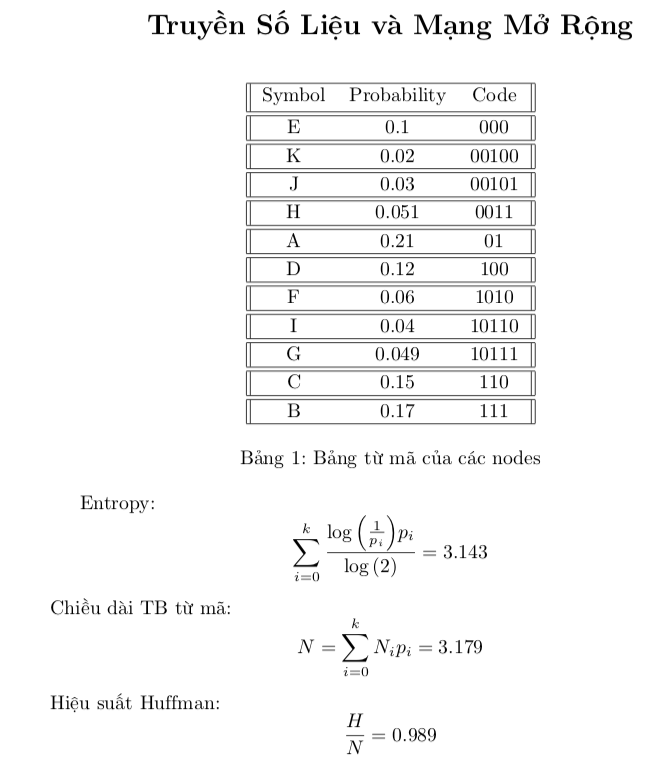
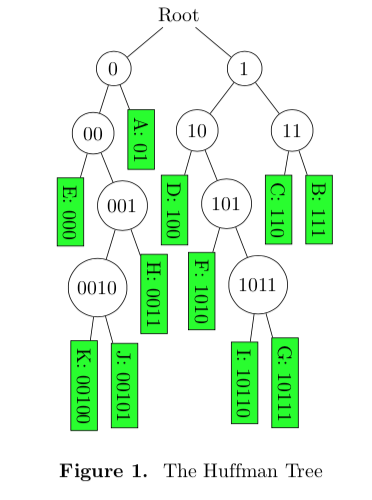


Cuối cùng ta dùng command *pdflatex* trên Terminal để xuất ra file .pdf:

#*##Convert from tex to pdf*

x = subprocess.call('pdflatex -output-directory=output output/latex\_output.tex', shell=True)

Ta được kết quả như hình sau ở file *output/latext\_output.pdf*:

**

Kết quả sau mã hóa:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Symbol** | A | B | C | D | E | F | G | H | I | J | K |
| **Probability** | 0.21 | 0.17 | 0.15 | 0.12 | 0.1 | 0.06 | 0.049 | 0.051 | 0.04 | 0.03 | 0.02 |
| **Code** | 01 | 111 | 110 | 100 | 000 | 1010 | 10111 | 0011 | 10110 | 00101 | 00100 |

1. **Tài liệu tham khảo:**

<https://stackoverflow.com/>

<https://tug.org/tug2019/slides/slides-ziegenhagen-python.pdf>

<https://en.wikibooks.org/wiki/LaTeX>