Tom and Jerry

*Giải mã văn bản mật sau*

🐈🖲🖲🐈🖲🐈🐈🖲🐈🐈🖲🖲🖲🖲🖲🐈🐈🐈🐈🖲🖲🖲🐈🐈🖲🐈🖲🐈🖲🖲🐈🖲🖲🖲🐈🐈🖲🐈🖲🐈

*Biết*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | C | E | H | M | O | T | U | V | \* |
| 🐈🐈🐈 | 🐈🐈🖲 | 🐈🖲🐈🐈 | 🐈🖲🐈🖲 | 🐈🖲🖲 | 🖲🐈🐈 | 🖲🐈🖲🐈 | 🖲🐈🖲🖲 | 🖲🖲🐈 | 🖲🖲🖲 |

Algorithm

Việc đầu tiên là số hóa hai ký tự 🐈🖲 thành 0 và 1 như trong input file TOMJERRY.INP.

### Code

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | C | E | H | M | O | T | U | V | \* |
| 🐈🐈🐈 | 🐈🐈🖲 | 🐈🖲🐈🐈 | 🐈🖲🐈🖲 | 🐈🖲🖲 | 🖲🐈🐈 | 🖲🐈🖲🐈 | 🖲🐈🖲🖲 | 🖲🖲🐈 | 🖲🖲🖲 |
| 000 | 001 | 0100 | 0101 | 011 | 100 | 1010 | 1011 | 110 | 111 |

input file

|  |  |
| --- | --- |
| TOMJERRY.INP | Giải thích |
| 10  A 000  C 001  E 0100  H 0101  M 011  O 100  T 1010  U 1011  V 110  \* 111 | 10 ký tự  mã số của A  mã số của C  mã số của E  mã số của H  mã số của M  mã số của O  mã số của T  mã số của U  mã số của V  mã số của \* |
| 0110100100111110000111001010110111001010 | văn bản cần giải mã |

Gọi s là string cần giải mã.

s = '0110100100111110000111001010110111001010'

d là từ điển chứa mã của các ký tự cho trong input file.

d = ['A' : 000, 'C' : 001, 'E' : 0100, 'H' : 0101, 'M' : 011, 'O' : 100, 'T' : 1010, 'U' : 1011, 'V' : 10, '\*' : 111]

Chương trình sẽ gồm hai phần: đọc dữ liệu và giải mã.

Phần thứ nhất: Đọc dữ liệu

Mở input file f với tên 'TOMJERRY.INP'. Chỉ thị 'r' cho biết file f được mở để đọc.

f = open('TOMJERRY.INP', 'r') # (1)

Đọc dòng đầu tiên từ file f và chuyển đổi sang số nguyên: đó là số ký tự được mã hóa.

n = int(f.readline()) # (2)

Trong dòng lệnh trên, f.readline() đọc một dòng (đầu tiên) từ file f và return string line. Hàm int chuyển đổi string sang số integer.

Khởi tạo từ điển d:

d = dict()

Tiếp đến đọc n dòng lệnh của f và nạp trị vào từ điển d.

for i in range(n):

line = f.readline().replace(' ', '').replace('\n', '') # (5)

d[line[0]] = line[1::] # (6)

Các lệnh replace(' ', '').replace('\n', '') làm nhiệm vụ xóa các dấu cách và dấu kết dòng trong string.

Sau các dòng lệnh này ta thu được từ điển:

# d['A':'000', 'C':'001', 'E':0100,...]

Cuối cùng đọc văn bản cần giải mã vào biến s để chuẩn bị cho các bước tiếp theo, rồi đóng file:

s = f.readline().replace(' ', '').replace('\n', '')

f.close() # (7)

Đoạn chương trình cho Phần thứ nhất sẽ được viết như sau:

# Tom And Jerry

# Brute Force

f = open('TOMJERRY.INP', 'r') # (1)

n = int(f.readline()) # (2)

print(n, ' ky tu') # (3)

d = dict() # d['A':'000', 'C':'001', 'E':0100,...] (4)

for i in range(n):

line = f.readline().replace(' ', '').replace('\n', '') # (5)

d[line[0]] = line[1::] # (6)

s = f.readline().replace(' ', '').replace('\n', '')

f.close() # (7)

print(d)

print(' Decoding ', s, '...\n ', end='')

Phần thứ hai: Giải mã

Phương án đơn giản đầu tiên là vét cạn:

Lần lượt so sánh mã của từng ký tự trong bảng mã với đoạn đầu của s. Nếu khớp thì cắt đầu của string s và xuất ra ký tự tương ứng vào biến kết quả result . Khi nào s trở thành string rỗng thì kết thúc.

result = ''

while s != '':

for x in d:

m = len(d[x])

if d[x] == s[:m]: # (8)

result += x # (9)

s = s[m::] # (10)

print(result)

Dòng lệnh (8)

if d[x] == s[:m]: # (8)

so sánh giá trị d[x] trong từ điển có khớp với tiền tố gồm m bit của s.

Dòng lệnh (10)

s = s[m::] # (10)

đặt lại trị cho string s bằng hậu tố của s tính từ ký tự thứ m đến cuối string.

Độ phức tạp tính toán theo phương án này sẽ là n×k×m phép so sánh bit, trong đó n là số bit của s, k là số ký tự được mã hóa, m là chiều dài trung bình của mỗi mã.

### Program of Version 1

# Tom And Jerry

# Ver. 1: Brute Force

f = open('TOMJERRY.INP', 'r') # (1)

n = int(f.readline()) # (2)

print(n, ' ky tu') # (3)

d = dict() # d['A':'000', 'C':'001', 'E':0100,...] (4)

for i in range(n):

line = f.readline().replace(' ', '').replace('\n', '') # (5)

d[line[0]] = line[1::] # (6)

s = f.readline().replace(' ', '').replace('\n', '')

f.close() # (7)

print(d)

print(' Decoding ', s, '...\n ', end='')

result = ''

while s != '':

for x in d:

m = len(d[x])

if d[x] == s[:m]: # (8)

result += x # (9)

s = s[m::] # (10)

print(result)

Cây tìm kiếm nhị phân

Ta nên tối ưu hóa chương trình bằng phương pháp sử dụng *cây tìm kiếm nhị phân*. Đó là một đồ thị có lá là các ký tự, mỗi đỉnh có không quá hai đỉnh con: con trái và con phải và được tổ chức như sau.

Cây tìm kiếm nhị phân

A diagram of a tree

Description automatically generated

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | C | E | H | M | O | T | U | V | \* |
| 🐈🐈🐈 | 🐈🐈🖲 | 🐈🖲🐈🐈 | 🐈🖲🐈🖲 | 🐈🖲🖲 | 🖲🐈🐈 | 🖲🐈🖲🐈 | 🖲🐈🖲🖲 | 🖲🖲🐈 | 🖲🖲🖲 |
| 000 | 001 | 0100 | 0101 | 011 | 100 | 1010 | 1011 | 110 | 111 |

Các đỉnh của cây được đánh số lần lượt từ trên xuống

A diagram of a diagram

Description automatically generatedDòng thứ nhất: 1 đỉnh, số hiệu 1

Dòng thứ hai: 2 đỉnh, số hiệu 2 và 3

Dòng thứ ba: 4 đỉnh, số hiệu 4, 5, 6, và 7

…

Dòng thứ m: đỉnh, số hiệu từ đến - 1

Đỉnh mang số hiệu i sẽ có hai cây con ứng với hai cạnh là

* con trái mang số hiệu 2i với nhãn trên cạnh là 0, và
* con phải mang số hiệu 2i + 1 với nhãn trên cạnh là 1.

Nếu ta đi từ gốc cây (từ đỉnh 1) xuống đến lá thì sẽ thu được các nhãn theo thứ tự từ trên xuống là mã của ký tự treo trên lá tương ứng.

Điều thú vị là có một sự tương ứng giữa số hiệu của lá và mã số của ký tự treo trên lá đó.

Ví dụ

Ký tự E có mã số là 0100 đi qua các đỉnh (1)(2)(5)(10)(20) được đặt tại đỉnh có số hiệu 20.

Ta có thể viết giải trình cho lá E như sau:

E: (1)**0**(2)**1**(5)**0**(10)**0**(20) (\*)

trong đó số ngoài ngoặc thuộc về mã (nhãn trên cạnh: **0**/**1**) của đỉnh E, các số trong ngoặc là số hiệu đỉnh khi ta đi từ gốc (1) đến lá E.

Biểu thức (\*) cho ta thủ tục tính số hiệu đỉnh thông qua mã số.

Ta tuân thủ theo quy tắc sau:

* Đi từ đỉnh i theo cạnh trái mang nhãn **0** thì sẽ đến đỉnh 2i, theo cạnh phải mang nhãn **1** thì sẽ gặp đỉnh 2i + 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *mã số của ký tự E* |  | 0 |  | 1 |  | 0 |  | 0 |  |  |
| *số hiệu đỉnh* | (1) gốc | trái | (2) | phải | (5) | trái | (10) | trái | (20) |  |
|  |  |  |  |  |  |  |  |  |  |  |

Hàm code(s) dưới đây return số hiệu của đỉnh treo ký tự có mã ghi trong string s.

Ví dụ

Code('0100') = 20

def Code(s):

c = 1

for b in s:

c = c \* 2 + int(b)

return c

|  |
| --- |
| Thuật toán giải mã |
| Xuất phát từ gốc v = 1.  Mỗi lần xét một ký hiệu b trong văn bản mật s:  Nếu b = 0 rẽ trái đến đỉnh v = 2\*v  Nếu b = 1 rẽ phải đến đỉnh v = 2\*v + 1  Nếu gặp lá: xuất ra ký tự trên lá rồi  quay lại gốc: v = 1. |

Để tìm kiếm một phần tử nào trong cây, tại mỗi đỉnh, cụ thể là tại mỗi ngã ba, sau khi quyết định rẽ trái hay phải thì ta loại bỏ được một nửa khả năng tìm kiếm, do đó số phép so sánh sẽ chỉ còn log(k) với k là số node của cây. Thuật toán giải mã khi đó có độ phức tạp nlog(k) với n là số bit của văn bản mật, k là số ký tự được mã hóa.

Biểu diễn cây

Để biểu diễn cây ta chỉ cần dùng một từ điển tham chiếu d với hai thành phần:

key: code của ký tự

value: bản thân ký tự

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| key | 8 | 9 | 11 | 12 | 14 | 15 | 20 | 21 | 26 | 27 |
| value | A | C | M | O | V | \* | E | H | T | U |

Program

# Tom And Jerry

# Binary Search Tree

def Code(s):

c = 1

for b in s:

c = c \* 2 + int(b)

return c

f = open('TOMJERRY.INP', 'r')

n = int(f.readline())

print(n, ' ky tu')

d = dict() # d['A'] = 8, ...

for i in range(n):

line = f.readline().replace(' ', '').replace('\n', '')

d[Code(line[1::])] = line[0] # (1)

line = f.readline().replace(' ', '').replace('\n', '')

f.close()

print(d)

print(' Decoding ', line, '...\n ', end='')

result = ''

c = 1

for b in line:

c = c \* 2 + int(b)

if c in d:

result += d[c]

c = 1

print(result)

Giải thích

Dòng lệnn (1)

d[Code(line[1::])] = line[0] # (1)

gán trị cho từ điển d với key Code(line[1::]) và value line[0], trong đó line[1::] là hậu tố của string line sau khi bỏ qua ký tự đầu line[0].

Result

10 ky tu

{8:'A', 9:'C', 20:'E', 21:'H', 11:'M', 12:'O', 26:'T', 27:'U', 14:'V', 15:'\*'}

Decoding 0110100100111110000111001010110111001010 ...

MEO\*VA\*CHUOT