TiongSun إضافة الملفات عن طريق الرفع

مساهم 1

297 297) خطًا sloc) | 6.53 كيلو بايت ...

Import libraries

In [69]:
```
from heapq import heappush, heappop, heapify
from collections import defaultdict
from bitarray import bitarray
```

Input data to be compressed

Can be a path to a text file. For simplicity, an example text is used here.

In [70]:
```
text = "HAPPY HAPPY"
```

Create a library with frequency of each symbols

In [71]:
```
freq_lib = defaultdict(int)    # generate a default library
for ch in text:                # count each letter and record into the freque
    freq_lib[ch] += 1

print(freq_lib)
```

```
defaultdict(<class 'int'>, {'H': 2, 'A': 2, 'P': 4, 'Y': 2, ' ': 1})
```

Create Huffman Tree

In [72]:
```
heap = [[fq, [sym, ""]] for sym, fq in freq_lib.items()]  # '' is for enterin
print(heap)
```

```
[[2, ['H', '']], [2, ['A', '']], [4, ['P', '']], [2, ['Y', '']], [1, [' ',
'']]]
```

In [73]:
```
heapify(heap) # transform the list into a heap tree structure
print(heap)
```

```
[[1, [' ', '']], [2, ['A', '']], [4, ['P', '']], [2, ['Y', '']], [2, ['H',
'']]]
```

In [74]:
```
while len(heap) > 1:
    right = heappop(heap)  # heappop - Pop and return the smallest item from
    print('right = ', right)
    left = heappop(heap)
    print('left = ', left)

    for pair in right[1:]:
        pair[1] = '0' + pair[1]   # add zero to all the right note
    for pair in left[1:]:
        pair[1] = '1' + pair[1]   # add one to all the left note
    heappush(heap, [right[0] + left[0]] + right[1:] + left[1:])  # add values
```

```
right =  [1, [' ', '']]
```

```
left =  [2, ['A', '']]
right =  [2, ['H', '']]
left =  [2, ['Y', '']]
right =  [3, [' ', '0'], ['A', '1']]
left =  [4, ['H', '0'], ['Y', '1']]
right =  [4, ['P', '']]
left =  [7, [' ', '00'], ['A', '01'], ['H', '10'], ['Y', '11']]
```

In [75]:

```
huffman_list = right[1:] + left[1:]
print(huffman_list)
huffman_dict = {a[0]:bitarray(str(a[1])) for a in huffman_list}
print(huffman_dict)
```

```
[['P', '0'], [' ', '100'], ['A', '101'], ['H', '110'], ['Y', '111']]
{'P': bitarray('0'), ' ': bitarray('100'), 'A': bitarray('101'), 'H': bitarra
y('110'), 'Y': bitarray('111')}
```

Huffman encoding

In [76]:

```
encoded_text = bitarray()
encoded_text.encode(huffman_dict, text)
print(encoded_text)
```

```
bitarray('110101001111100110101001111')
```

Padding

Because data is stored as bytes (8 bits) rather than bits, we need to record the "padding" added to the data in order to remove them during decoding

In [77]:

```
padding = 8 - (len(encoded_text) % 8)
```

Save encoded text as a binary file

In [78]:

```
with open('compressed_file.bin', 'wb') as w:
    encoded_text.tofile(w)
```

Decoding

In [79]:

```
decoded_text = bitarray()

with open('compressed_file.bin', 'rb') as r:
    decoded_text.fromfile(r)

decoded_text = decoded_text[:-padding] # remove padding

decoded_text = decoded_text.decode(huffman_dict)
decoded_text = ''.join(decoded_text)

print(decoded_text)
```

```
HAPPY HAPPY
```

Save an uncompressed file for comparison