ADS

Strs & Text proc

Pattern match Algorith:

Brute force pattern Matchi Algorith

```
def find_brute (T,P):
    n,m = len(T), len(P)
    for i in rge (n-m+1):
        k=0
        while k<m and T[i+k] = =P[k]
            k+=1
        if k==m:
            return i
    return -1
```

The Boye-Moo algori

```
def find_boy_moo (T,P)
    n,m =len(T), len(P)
    if m==0:
        return 0
    last = {}
    for k in rge (m):
        last[P(k)] = k
    i=m-1
    k=m-1
    while i<n:
        if T[i]==P[k]:
            if k==0:
                return i
            else:
```

```
        i -= 1
        k -= 1
    else:
        j = last.get(T[i], -1)
        i = m - min(k, j+1)
    k = m-1
return -1
```

The Knuth-Morris-Pratt Algorithm

```
def find_kmp(T, P):
    n, m = len(T), len(P)
    if m == 0:
        return 0
    fail = compute_kmp_fail(P)
    j = 0
    k = 0
    while j < n:
        if T[j] == P[k]:
            if k == m-1:
                return j - m + 1
            j += 1
            k += 1
        elif k > 0:
            k = fail[k-1]
        else:
            j += 1
    return -1

def compute_kmp_fail(P):
    m = len(P)
    fail = [0] * m
```

```
j-1
k=0
while j<m:
    if p[j]==p[k]:
        fail[j]=k+1
        j+=1
        k+=1
    elif k>0:
        k=fail[k-1]
    else:
        j+=1
return fail
```

Text comparison:-

The huffman codes Algorith

Algorithm huffman(x):-
Input: String x of length n with d distinct characters
output: Codes tree of x
Compute the frequn f(c) of each chords of x

Initialize a priority queue Q
create a single-Node binary tree T
ston c
Insert T into Q with key f(c)
while len(Q)>1 do
(f1, T1) = Q.remove_min()
(f1, T2) = Q.remove_min()

Date:
Page:

create a new binary tree T with left subtree T_1 and right subtree T_2

Insert T into Q with key $f_1 + f_2$

$(f, T) = Q.\text{remove\_min}()$

return tree T

The Greedy method for text compression

Algorithm Greedy(x):

Input: Original text to be compressed

output: Compressed binary sequence

Algorithm:

1) Create a compression table that maps characters to fixed length binary codes.
2) Initialize an empty string to store compressed binary sequence.
3) for each character in original text:
   a. look up the character in the compression table to find its binary code.
   b. Append the binary code to the compressed binary sequence.
4) Return the compressed binary sequence.

Steps this:

→ A standard tree is a tree like data structure where each node represents a single character.

→ The root node represents an empty string and each level of the tree corresponds to a character in the key

→ The edges between nodes are labeled with characters.

Compressed tries:

→ A compressed trie is an optimized version of the standard trie that aims to reduce memory usage.

→ In a compressed trie, instead of having a single character per node, multiple characters are combined into single nodes, reducing the number of nodes in the structure.

→ This compression is achieved by collapsing nodes with multiple children into a single branch.

Suffix tries:

→ A suffix trie is a specialized trie used to store all the suffixes of a given string or set of strings.

→ It is useful in string matching and substring search algorithms.

→ The root node represents the empty string and each edge represents a character in the strings.

→ Unlike standard & compressed tries, suffix tries do not represent complete words but rather substrings or suffixes.

→ They can be used to efficiently perform operations like substring search, longest common substring and pattern matching.