Skstronghold's Blog

Algorithms, Implementations, Programming and Linux

- Home
- Algorithms
- Data Structures
- Linux
- Mathematics
-

Type text to search here... □

Home > Greedy > Huffman coding and decoding

Huffman coding and decoding

January 10, 2012 skstronghold Leave a comment Go to comments

Huffman codes are a widely used and very effective technique for compressing data; savings of 20% to 90% are typical, depending on the characteristics of the data being compressed. We consider the data to be a sequence of characters. Huffman's greedy algorithm uses a table of the frequencies of occurrence of the characters to build up an optimal way of representing each character as a binary string.

We assume that C is a set of *n* characters and that each character *c* ∈▼ C is an object with a defined frequency f [c]. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with a set of |C| leaves and performs a sequence of |C| – 1 "merging" operations to create the final tree. A min-priority queue Q, keyed on f , is used to identify the two least-frequent objects to merge together. The result of the merger of two objects is a new object whose frequency is the sum of the frequencies of the two objects that were merged.

My C++ implementation of Huffman coding(according to CLRS) is:

```
#include <iostream>
#include <queue>
#include <string>
#include <algorithm>

```

```
    node* right;

    node( char c = ' ', int f = -1 )
    {
        ch = c;
        freq = f;
        left = NULL;
        right = NULL;
    }

    node( node* c1, node* c2 )
    {
        ch = ' ';
        freq = c1 -> freq + c2 -> freq;
        left = c1;
        right = c2;
    }

    bool operator ()( const node& l, const node &r)
    {
              return l.freq >r.freq;
        }

    void traverse( string code = "" ) const;

};

void node::traverse( string code ) const
{
    if( left ) {
        left->traverse( code + '0' );
        right->traverse( code + '1' );
    } else {
        cout << ch << "\t" << freq << "\t" << code << endl;
    }
}

int main() {
    string str;
    int cnt;
    char c;
    priority_queue< node, vector<node>, node > q;

    cin >> str;

    for( int i = 65; i <= 90 ; i++ ) {
        c = (char)i;
        cnt = (int)count( str.begin(), str.end(), c );
        if( cnt ) {
            q.push( node( c, cnt ) );
            cnt = 0;
        }
    }

    for( int i = 97; i <= 122 ; i++ ) {
```

```
                }
```

```
        }

    while( q.size() != 1 ) {
        node *left = new node( q.top() );
        q.pop();
        node *right = new node( q.top() );
        q.pop();
        q.push( node( left, right ) );
    }

    cout << "Encoding " << endl;
    cout << "Word" << "\t" << "freq" << "\t" << "code" << er
    q.top().traverse();
}
```

We can decode a coded string using simple string matching, below is a similar implementation:

```
#include<iostream>
#include<map>

using namespace std;

int main()
{
    int i,j,k,a,b,n;
    char c;
    string s,t,ans;
    map <string, char> mymap;
    map <string, char> :: iterator it;

    cout << "Enter the no. different types of characters: ";
    cin >> n;

    for( i = 0; i < n; i++ ) {
        cout << "Enter the " << i+1 << " character and its c
        cin >> c >> s;
        mymap[s] = c;
    }

    cout << "Enter the encoded string: ";
    cin >> s;

    t = "";
    ans = "";
    for( i = 0; i < s.size(); i++ ) {
        t += " ";
        t[t.size()-1] = s[i];
        it = mymap.find(t);
        if( it != mymap.end() ) {
            c = mymap[t];
            ans += " ";
            ans[ans.size()-1] = c;
            t = "";
```

```
    return 0;
```

```
}
```

Sample input for the above implementation of Huffman decoding can be:

```
10
A 00
E 010
F 011
B 100
C 101
D 110
G 1110
H 11110
I 111110
J 111111
00100100001000100001111001000110
```

Share this:

B

Related

Greedy algorithms
In "Algorithms"

Kruskal's minimum spanning tree
In "Graph Theory"

Prim's minimum spanning tree
In "Graph Theory"

Categories: Greedy Tags: c++, coding, decoding, greedy, huffman
Comments (0) Trackbacks (0) Leave a comment Trackback

1. No comments yet.

1. No trackbacks yet.

Leave a Reply

Enter your comment here...

Closest pair of points Depth first search
RSS feed

Sunil Kumar

Undergraduate Student
Indian Institute of Information Technology
Allahabad, India

Recent Posts

- Introduction to lex
- Downloading with wget
- Prim's minimum spanning tree
- Bucket sort
- Treaps
- Dijkstra's single source shortest path algorithm

Archives

- January 2012 (28)
- April 2011 (1)

Blog Hits

- 14,424

Subscribe

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 8 other followers

Enter your email address

Sign me up!

Categories

- Algorithms (18)
  - Computational Geometry (1)
  - Dynamic Programming (1)
  - Graph Theory (5)
  - Greedy (1)
  - Sorting (6)
  - String Matching (1)
- Data Structures (5)
- Linux (5)
  - Lex and Yacc (1)
  - Sudo (1)
  - Vim (2)
  - Wget (1)
- Mathematics (1)

Twitter Updates

Error: Twitter did not respond. Please wait a few minutes and refresh this page.

Top
Blog at WordPress.com.