Stream Encoding

locked

In this challenge, you will implement Huffman encoding for data streams. Suppose you are receiving a stream of text which you would like to encode and store optimally. To perform Huffman encoding, you will need the frequencies of all characters in the entire stream, but you don't want to wait for the stream to end before you begin encoding.

One way to solve this problem is to have fixed size windows: after each window ends, you would update the encoding which you can then use to encode the most recently completed window, and repeat the process. To be more precise, let s denote the entire stream to be encoded, and let m be the window size. Then, s is broken down into $w = \lceil \frac{|s|}{m} \rceil$ windows $s_1, s_2, \ldots, s_w$, where each window (except possibly the last one) is of length m. Now, after reading and processing s_1 (to calculate the character frequencies), Huffman algorithm would be applied to produce the Huffman code, which would then be used for encoding s_1. In general, after processing s_i, the Huffman code would be reconstructed considering the cumulative frequencies in $s_1, \ldots, s_i$, and this code would be used to encode s_i.

Implement the above scheme in an efficient manner. In particular, use C++ STL Priority queue for storing and efficiently retrieving minimum frequency tree nodes while implementing Huffman's algorithm. While retreiving tree nodes with the same frequency, use the following strategy to break the tie:

- For external (i.e. leaf) nodes, pick the node based on the increasing order of the ASCII encoding of the characters corresponding to the nodes. That is, suppose characters `a` and `b` have the same frequency, then `a` should be selected first.
- If an external node and an internal node have the same frequency, you must first pick the external node.
- If two internal nodes have the same frequency, you must pick the internal node that was generated (and inserted into the prioirty queue) earlier.

Use C++ STL map (or unordered_map) to store frequencies of characters.

Input Format

- The first line will contain two numbers `n m` where `n` denotes the total size of the stream to be encoded, and `m` denotes the window size.
- The second line will contain the entire stream.

Constraints

$1 \leq m \leq n \leq 10^6$

Output Format

The output should contain on a single line, the entire (binary) huffman encoding of the input stream.

Sample Input 0

```
7 2
aKf$H#m
```

Sample Output 0

```
10110011010000
```

Sample Input 1

```
8 4
YqG&z@!*
```

Sample Output 1

```
10110100111011000010
```

Sample Input 2

```
6 2
r#Sx$F
```

Sample Output 2

```
100111101110
```

Sample Input 3

```
8 6
%m&iX^hZ
```

Sample Output 3

```
10001101001101111101011
```

Sample Input 4

```
7 2
$vX#jKg
```

Sample Output 4

```
01100000110110
```

Sample Input 5

```
9 3
H%l^&e!Qs
```

Sample Output 5

```
11100111101001110010110
```

Submissions: 88

Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

C++20

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <queue>
#include <unordered_map>
using namespace std;
class Code //Code represents a class corresponding to each character, which contains all its details like character value, its frequency, prev, left, right node when a tree would be constructed using it
{
    public:
      char cH;
      int fQ;
      Code* prev;
      Code* left;
      Code* right;
      int tM;
      Code(char ch, int time)
      {
          cH = ch;
          fQ = 1;
          prev = NULL;
          left = NULL;
          right = NULL;
          tM = time;
      }
      Code()
      {
          fQ=1;
          prev = NULL;
          left = NULL;
          right = NULL;
      }
      void changeLeft(Code* ptr){left = ptr;} //changing the left Code pointer to ptr;
      void changeRight(Code* ptr){right = ptr;} //changing the right Code pointer to ptr;
      void changePrev(Code* ptr){prev = ptr;} //changing the prev Code pointer to ptr;
      void incF(){fQ++;} //increasing the frequency by 1
      void changeF(int g){fQ=g;} //changing the frequency to a given value g
      void changeC(char chr){cH = chr;} //changing the character of the Code to chr
      void changeT(int time){tM = time;} //changing the time of insertion of the Code to time
};
class compare_u
{
    public:
    bool operator()(Code* c1, Code* c2)
    {
        if (c1->fQ!=c2->fQ)
        {
            return (c1->fQ>c2->fQ); //compares the frequency
        }
        else if ((c1->left==NULL&&c1->right==NULL)&&(c2->left!=NULL||c2->right!=NULL)) //gives priority to the external node when one is external and the other is internal
        {
            return false;
```

```
        }
        else if ((c1->left!=NULL||c1->right!=NULL)&&(c2->left==NULL&&c2->right==NULL)) //gives priority to the external node when one is external and the other is internal
        {
            return true;
        }
        else if ((c1->left!=NULL||c1->right!=NULL)&&(c2->left!=NULL||c2->right!=NULL))
        {
            return (c1->tM>c2->tM);  //checks the time of insertion and compares on the basis of that
        }
        else if ((int)(c1->cH)>(int)(c2->cH))
        {
            return true; //compares the ASCII value of the characters and compares on the basis of that
        }
        else
        {
            return false;
        }
    }
};
void Construct(priority_queue<Code*, vector<Code*>, compare_u> pq, int* tt)
{
    priority_queue<Code*, vector<Code*>, compare_u> pp;
    if (pq.size()==1) //special case when the priority_queue has only one element
    {
        pp.push(pq.top()); pq.pop(); //the Code is being popped out
    }
    else
    {   // the loop below runs till all the Code pointers has been assigned a terminal node in the binary tree constructed
        while(pq.size()>1)
        {
          Code* aa = pq.top();
          if (aa->left==NULL&&aa->right==NULL)
          {
              pp.push(pq.top());    //as in this code we are popping from the priority_queue, we are maintaining a separate priority_queue pp to be again able to copy the elements from pp to pq
          }
          pq.pop();   // the most prior element has been popped
          Code* bb = pq.top();
          if (bb->left==NULL&&bb->right==NULL)
          {
              pp.push(pq.top());
          }
          pq.pop();   // the most prior element has been popped(the second most considering the former one too), basically we are popping two most prior elements and assigning them to a terminal node
          Code* cc = new Code();  // a new Code is created which will serve as a parent node of the two most prior elements of the former priority_queue

          // the defined above node is being made related to its children nodes(two most prior Codes of the former priority_queue)
          (*cc).changeC(aa->cH);
          (*cc).changeLeft(aa);
          (*cc).changeRight(bb);
          (*aa).changePrev(cc);
          (*bb).changePrev(cc);
          (*cc).changeF((aa->fQ) + (bb->fQ));
          (*cc).changeT(*tt);
          pq.push(cc);
          (*tt)++;
        }
        pq.pop();
    }
```

```
    //here we are again copying the elements of pp to pq itself for future use
    while(!pp.empty())
    {
        pq.push(pp.top());
        pp.pop();
    }
}
int main()
{
    int num, mm;
    cin>>num>>mm;
    Code* ch[num];   // an array of Codes will be maintained which shall
    int tt=0;
    unordered_map<char, int> mp;  // this map contains the frequency of all the characters
    priority_queue<Code*, vector<Code*>, compare_u> pq;
    char chr;
    for (int ii=0; ii<num; ii++)
    {
        int jj=0;
        //the below loop runs for each mm lenth window
        for (; jj<mm&&ii+jj<num; jj++)
        {
            cin>>chr;
            ch[ii+jj] = new Code(chr, tt);
            if (mp.count(chr)==1)
            {
                mp[chr]++;
                vector<Code*> vc;    // this vector is basically used to store the elements we would pop from the priority_queue till we reach the desired element and modify it, so that those elements can be reinserted into the priority_queue
                while(!pq.empty())
                {
                    if (pq.top()->cH==ch[ii+jj]->cH)
                    {
                        Code* gg = pq.top();
                        pq.pop();
                        gg->incF();
                        pq.push(gg);
                        ch[ii+jj] = gg;
                        break;
                    }
                    else
                    {
                        vc.push_back(pq.top());
                        pq.pop();
                    }
                }
                while(!vc.empty())
                {
                    pq.push(*(--vc.end()));
                    vc.pop_back();
                }
            }
            else
            {
                mp[chr]=1;
                pq.push(ch[ii+jj]);
            }
            tt++;
        }
        Construct(pq, &tt);  // calls the constructor funcion defined above
        // the below loop is for finding out the code value of all the characters of the window just read
        for (int kk=0; kk<mm&&ii + kk<num; kk++)
        {
            Code* cd = ch[ii + kk];
```

```
            vector<int> v1; // this vector contains the path we traverse from the terminal node(leaf) to the root node in the form of bits
            while(cd->prev!=NULL)
            {
                if (cd->prev->left==cd)
                {
                    v1.push_back(0);
                }
                else if (cd->prev->right==cd)
                {
                    v1.push_back(1);
                }
                cd=cd->prev;
            }
            if (pq.size()==1)
            {
                v1.push_back(0);
            }
            while(!v1.empty()) // the elements of the vectors are eliminated in the opposite order od insertion, as we need the path code from root to leaf
            {
                cout<<*(--v1.end());
                v1.pop_back();
            }
        }
        ii=ii + jj-1;
    }
    return 0;
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code