**How to create Cut, Copy, Paste, Delete, Add using Ropes data structure:**

Before going into the tutorial you have to know several things before continuing

**1. What are Ropes**

* A rope is a binary tree, with each node having a maximum of 2 children.
* Each leaf holds a string and a length. Each node further up the tree holds the sum of the lengths of all the leaves in its left subtree.
* A node with 2 children divides the whole thing into two parts
* The left stores the first part of the string
* The right stores the second part of the string,
* The node length is the sum of the lengths of the leaf nodes of its left child

**Here’s a picture demonstrating how it works**

Hello World

[11]

/ \

[5] [6]

“Hello” / \

[1] [5]

“ ” “World”

Hello 5 characters

“ ” 1 character

World 5 characters

HelloWorld

[10]

/ \

[5] [5]

“Hello” “World”

**2. Why Are Ropes Important?**

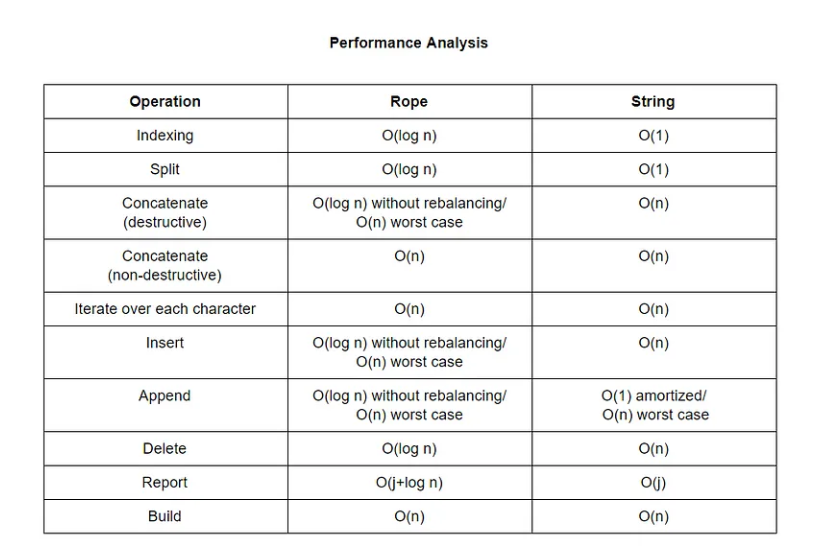
Ropes are very commonly used in text editors,

for example, Gmail, outlook, and even compilers

Because they deal with manipulations of large texts that need frequent edits and deletions, they are valuable in scenarios where regular string operations like arrays, and vectors would be too slow or large in memory.

Complexity comparison

(Rope vs Array)



As you may see concatenation and split operations in our case (cut) are faster on small datasets. However, when array-based strings are used for longer strings, time complexity and memory used for inserting and deleting characters becomes unacceptably large.

**3. Disclaimer**

This program uses a rope to handle text efficiently. However, for commands like copy, cut, paste, add, and delete, the rope is converted into a string, and then I reconstruct it back into a rope to make the changes easier, which makes it O(n) instead of O(log n). Once the changes are made the string is turned back into a rope. This keeps things simple while still using the rope structure, plus it was easier and less of a headache for me to make.

*So its like this for example cut*

Hello World

[11]

/ \

[5] [6]

“Hello” / \

[1] [5]

“ ” “World”

*Converts it into a string*

“Hello World”

*Cut Hello*

Finds Hello

*Output:*

World

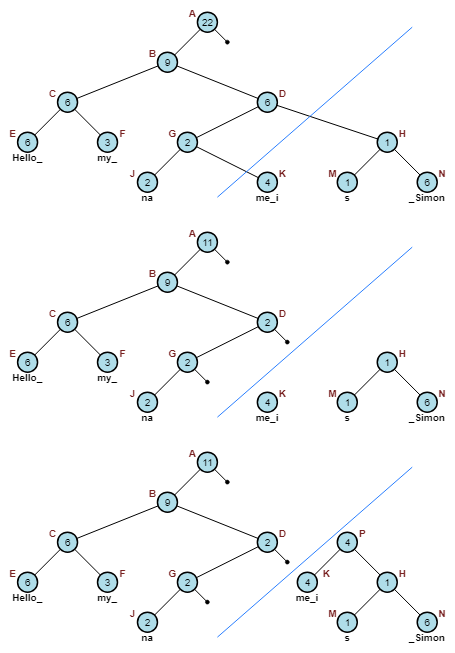
*Then it rebuilds it*

[5]

/ \

[2] [3]  
“Wo” “ld”

What it should look like



Since that's out of the way here's the tutorial…….

**Step-by-step Tutorial:**

1. **Class structure**

What you will need

RopeNode - node of the tree

Value - Actual text of the node (but only for leaf nodes)

Weight - Represents the length of the string in leaf nodes

Weight tells you the total length of the string represented by the left child or the subtree

Left and right - points to left and right child nodes

Here’s the full class structure

class RopeNode{

public:

int weight;

string value;

RopeNode\* left;

RopeNode\* right;

RopeNode(int w, const string& val = ""){ // referencing value making sure that it defaults to nothing if nothing is provided

weight = w;

value = val;

left = nullptr;

right = nullptr;

}

};

With RopeNode(int weight, const string& val = “”) being your constructor throughout the code

1. **Constructing a Rope**

I created recursive calls to build the rope

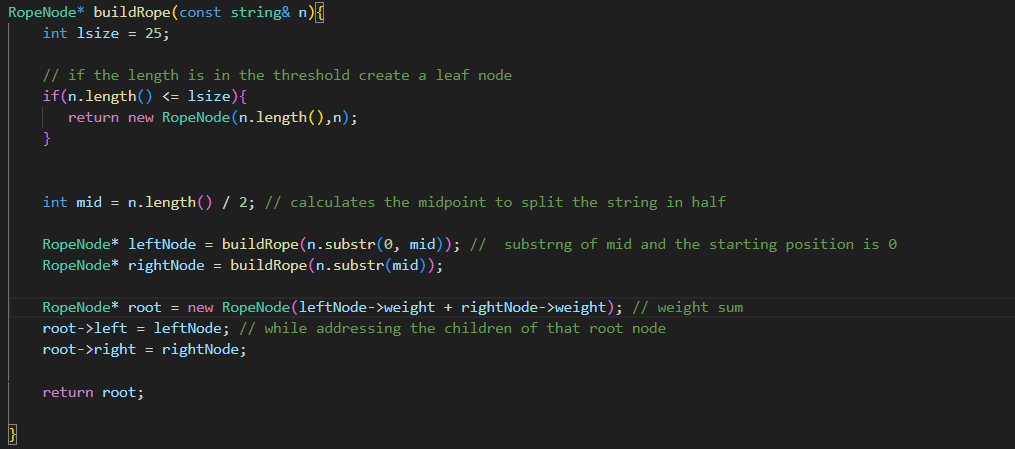
**buildRope function:**

Builds the rope from the string

*Base case*- its so the recursion stops

If the string length is less than equal to a threshold (lsize), it creates a leaf node with the string

*Recursive case*: Splits the string into two halves at the midpoint, and builds ropes for each half and connects them as left and right children of a new root node



What does new do? It allocates memory to the RopeNode which will have to be freed up by delete later.

*Breakdown:*

Lsize is a threshold on how long the rope should be

**Step1: Root Node creation**

Lets say that the input is “Hi I am a rope” which is a length of 14 characters lsize=14

Root created with the weight of 14

[14]

/ \

The middle or mid will calculate the split

**Step 2: Calculating the Midpoint**

The mid of the string is calculated: 14 / 2 = 7.

The string is split at position 7 into two substrings:

Left substring: (Hi I ), which is 5 characters long.

Right substring: (am a rope) which is 9 characters long.

[14]

/ \

[5] [9]

"Hi I " "am a rope"

**Step 3: Recusivly splitting substrings**

The left substring "Hi I " is 5 characters long which can be split again.

Left child: "Hi " = 3.

Right child: "I "= 2.

[14]

/ \

[5] [9]

/ \ "am a rope"

[3] [2]

"Hi " "I "

Same with the right it can be split further with this being the final rope

[14]

/ \

[5] [9]

/ \ / \

[3] [2] [3] [6]

"Hi " "I " “am ” / \

[2] [4]

“a ” “rope”

Why is there no 7 node in the split calculation? 7 is used during the mid calculation but it doesnt make a separate node of 7. If you remember the weights are determined by the length of the substrings

The tree reflects the lengths 5 and 9, not the midpoint itself.

1. **Displaying a rope:**

Once again I am using recursion for this one with a couple of simple if and else statements

void printRope(const RopeNode& node){

if(node.left == nullptr && node.right == nullptr){

cout << node.value;

}

else{

if(node.left != nullptr){

printRope(\*node.left);

}

if(node.right != nullptr){

printRope(\*node.right);

}

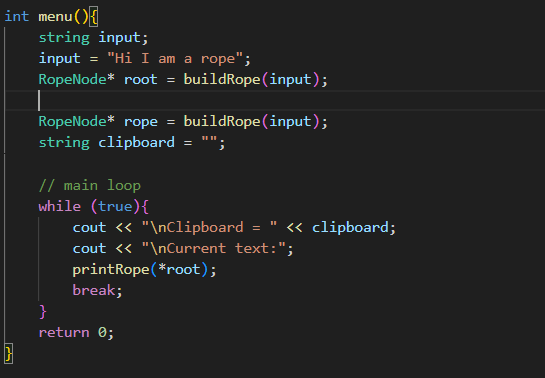
}

}

Mostly self-explanatory   
  
If both the left and right node reaches the nullptr print the nodes

And once again the base case that stops if it hits the nullptr.

**Before We continue lets start the main function**

****

Or you can store it in another function like I did and I called it menu()

With this, it builds the rope and if you run it should work.

The reason why there is a while true loop is so you can easily copy paste, cut at will

1. **Count the helper function and getTxt function**

count()

You may wonder why would you need a count function well it would be nice to know the final length of the string so you can correctly edit it.

To do so you need to basically return the weight pointer

Then you recursively return the left and right nodes

getTxt()

Similarly getting the full text of a rope and converting it into a string

The only thing different is returning a blank string and the value pointer

The reason why I did this is that it is easier to do string manipulation with text.

// tracks the length of the characters

int count(const RopeNode\* node) {

if (!node) {

return 0;

}

if (!node->left && !node->right) {

return node->weight;

}

return count(node->left) + count(node->right);

}

// full text of the rope which converts into a string

string getTxt(RopeNode\* node){

if(!node){

return "";

}

if(!node->left && !node->right){

return node->value;

}

return getTxt(node->left) + getTxt(node->right);

}

**Main setup**

In your main or menu

Get rid of break you wont need it at the moment

Add-in

The count and the command setup that we will need for the next step

cout << "\nText length: " << count(root) << " characters\n";

cout << "a = Add\n d = Delete\n x = Cut\n c = Copy\n p = Paste\n ";

char command;

cin >> command;

cin.ignore();

Here is the full menu structure so far….



1. **Cut, and Delete**

We will start with cut once you understand it you will be able to do delete very easily.

* You will need the root, clipboard, and substring as parameters
* Use a string to determine the text which I used getTxt to store the text itself

And position to find the substring of the text

* So to actually cut you will need to put the substring to the clipboard
* Then erase it from the text
* Then rebuild the rope

Which will get rid of the text and copy that text to the clipboard

Heres what that looks like

**// cut logic**

**RopeNode\* cutRope(RopeNode\* root, string& clipboard, const string& substring){**

**string txt = getTxt(root);**

**int pos = txt.find(substring);**

**if(pos == -1){**

**cout << "Substring not found\n";**

**return root;**

**}**

**clipboard = substring;**

**txt.erase(pos,substring.length());**

**delete root;**

**return buildRope(txt);**

**}**

**Then for delete, it's the same thing just don't copy the substring to the clipboard**

Now lets update the menu or main

***Cut***

if (command == 'x'){

cout << "Type what you would like to cut?\n";

string cut;

getline(cin,cut);

root = cutRope(root,clipboard,cut);

}

***And delete***

else if(command == 'd'){

cout << "Delete\n";

cout << "Type something to Delete:\n";

string del;

getline(cin,del);

root = deleteRope(root,del);

cout << "Deleted " << del << endl;

}

1. **Paste**

Similar structure

In the parameter Have a root, clipboard, and position

Insert the text, from the position and clipboard delete the previous, and rebuild the rope

RopeNode\* pasteRope(RopeNode\* root, string& clipboard, int pos){

string txt = getTxt(root);

if(pos > txt.length()){

cout << "bad position";

return root;

}

txt.insert(pos,clipboard);

delete root;

return buildRope(txt);

}

***Menu***

else if (command == 'p'){

cout << "Paste from clipboard\n";

if(clipboard.empty()){

cout << "Clipboard is empty\n";

continue;

}

cout << "Enter the position to paste the text\n";

int pos;

cin >> pos;

root = pasteRope(root,clipboard,pos);

}

1. **Add**

Very similar to paste instead of inserting the clipboard you insert the position, and the substring instead

RopeNode\* addRope(RopeNode\* root, const string& substring, int pos){

string txt = getTxt(root);

if(pos > txt.length()){

cout << "Invalid position " << txt.length() <<" "<<endl;

return root;

}

txt.insert(pos,substring);

delete root;

return buildRope(txt);

}

***Menu***

else if(command == 'a'){

cout << "Add\n";

cout << "Type something to Add:\n";

string add;

getline(cin,add);

cout << "Enter the position to insert the text\n";

int pos;

cin >> pos;

cin.ignore();

root = addRope(root,add,pos);

cout << "Added " << add << " at position" << pos << endl;

}

1. **Menu and Copy**

Copy is indeed not a function it was easier to implement in the menu itself believe it or not

So we will go through the menu

else if (command == 'c'){

cout << "Type something to copy:\n";

string copy;

getline(cin,copy);

clipboard = copy;

cout << "Copied to clipboard " << clipboard << endl;

}

*Here the full menu*

int menu(){

string input;

input = "Hi I am a rope";

RopeNode\* root = buildRope(input);

RopeNode\* rope = buildRope(input);

string clipboard = "";

while (true){

cout << "\nClipboard = " << clipboard;

cout << "\nCurrent text:";

printRope(\*root);

cout << "\nText length: " << count(root) << " characters\n";

cout << "a = Add\n d = Delete\n x = Cut\n c = Copy\n p = Paste\n ";

char command;

cin >> command;

cin.ignore();

if (command == 'x'){

cout << "Type what you would like to cut?\n";

string cut;

getline(cin,cut);

root = cutRope(root,clipboard,cut);

}

else if (command == 'c'){

cout << "Type something to copy:\n";

string copy;

getline(cin,copy);

clipboard = copy;

cout << "Copied to clipboard " << clipboard << endl;

}

else if (command == 'p'){

cout << "Paste from clipboard\n";

if(clipboard.empty()){

cout << "Clipboard is empty\n";

continue;

}

cout << "Enter the position to paste the text\n";

int pos;

cin >> pos;

root = pasteRope(root,clipboard,pos);

}

else if(command == 'd'){

cout << "Delete\n";

cout << "Type something to Delete:\n";

string del;

getline(cin,del);

root = deleteRope(root,del);

cout << "Deleted " << del << endl;

}

else if(command == 'a'){

cout << "Add\n";

cout << "Type something to Add:\n";

string add;

getline(cin,add);

cout << "Enter the position to insert the text\n";

int pos;

cin >> pos;

cin.ignore();

root = addRope(root,add,pos);

cout << "Added " << add << " at position" << pos << endl;

}

else{

cout << "Invalid command";

}

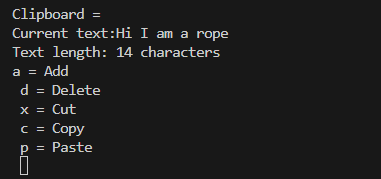
}

}

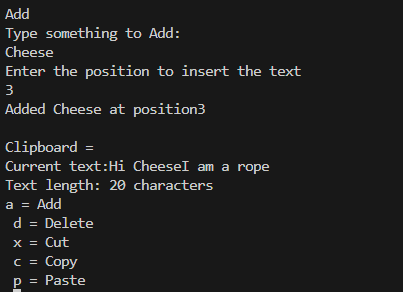
And then just call the menu you could create an exit if you so desire in the commands but it's up to you.

1. **Output**

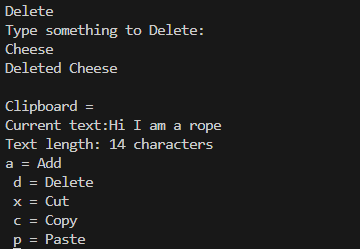
For the final output, it should look something like this



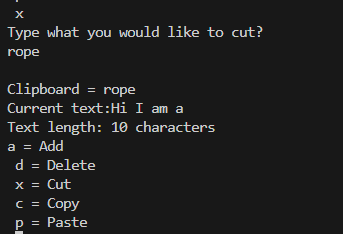
***Add***



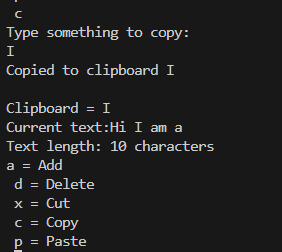
***Delete***



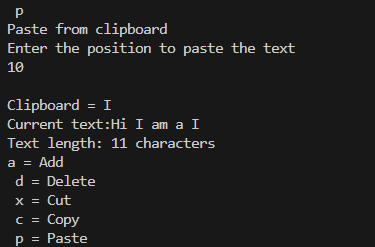
***Cut***



***Copy***



***Paste***



**Full Code:**[**https://github.com/Hubiejr/Rope-cut-copy-and-paste/blob/main/rope.cpp**](https://github.com/Hubiejr/Rope-cut-copy-and-paste/blob/main/rope.cpp)

**References:**

Wikipedia on ropes

<https://en.wikipedia.org/wiki/Rope_(data_structure)>

Stack Overflow

Mostly used to research syntax and built-in functions

<https://stackoverflow.com/>