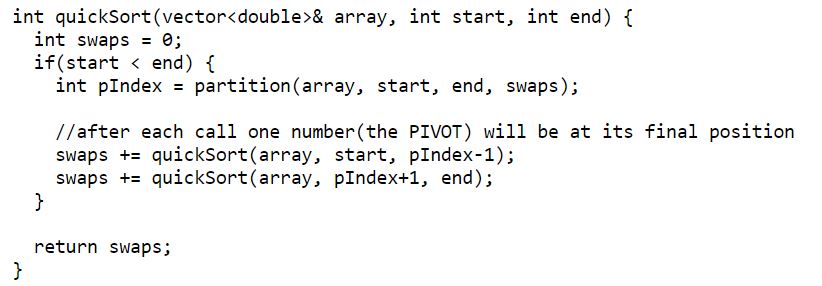
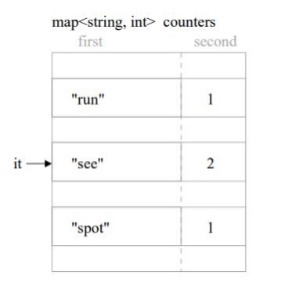
Quick sort:

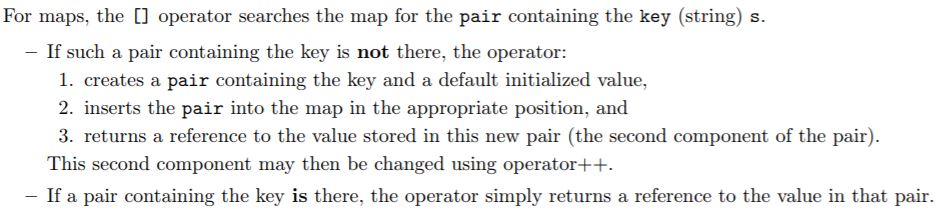


std::map<key\_type, value\_type> var\_name

std::pair< key\_type, value\_type>

Map search, insert and erase => 0(logn)

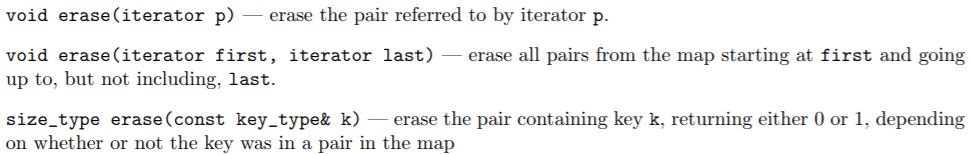




std::map<std::string, int>::const\_iterator it;

it->first it -> second

*m.find(key) m.insert(std::make\_pair(key, value));*

std::pair<map<key\_type, value\_type>::iterator, bool>

map < string, vector<int> > :: const\_iterator p;

typedef map < string, vector<int> > map\_vect;

map\_vect :: const\_iterator p;

set:

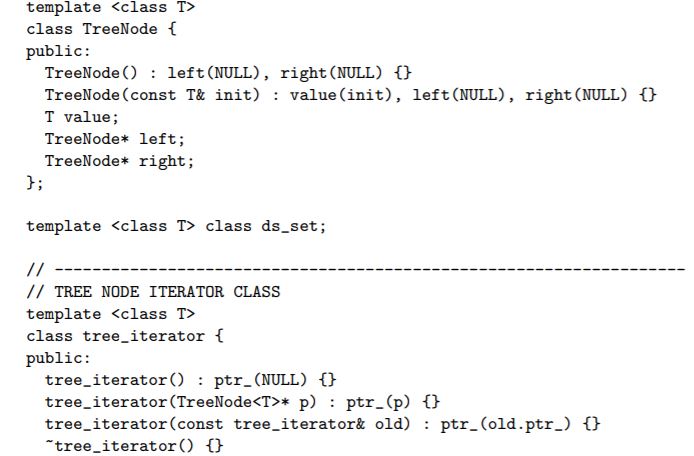
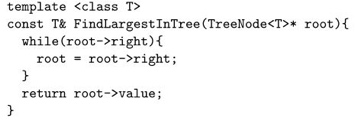
for (set<string>::iterator p = words.begin(); p!= words.end(); ++p) cout << \*p << endl; s.insert(\*it)

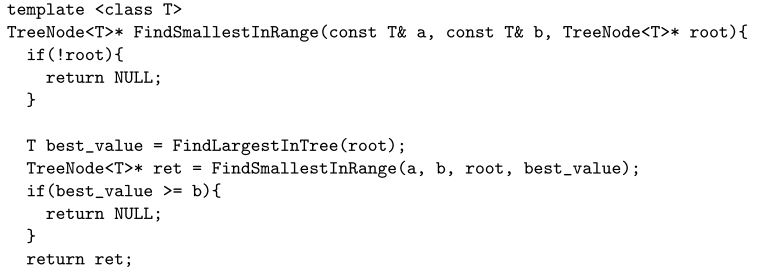
size\_type set<key>::erase(const Key& x);

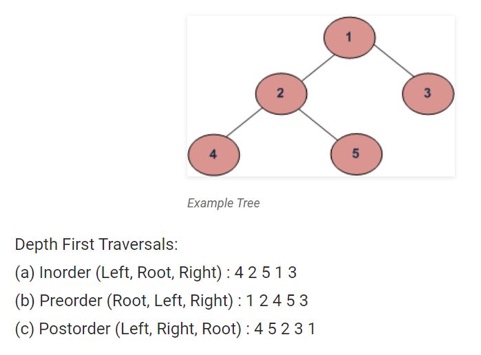
void set<key>::erase(iterator p);

void set<key>::erase(iterator first, iterator last);

insert find 复杂度同map







void Rope::destroy\_tree(Node\* p)

{ if (!p) return;

destroy\_tree(p->left);

//move the iterator before delete the Node

Node\* tmp = p->right;

delete p;

destroy\_tree(tmp);}

Node\* Rope::copy\_tree(Node\* p){

if (p == NULL) return NULL;

else{

Node\* copy = new Node;

copy -> weight = p->weight;

copy -> value = p->value;

copy -> left = copy\_tree(p->left);

if (copy->left != NULL){

copy->left->parent = copy;

}

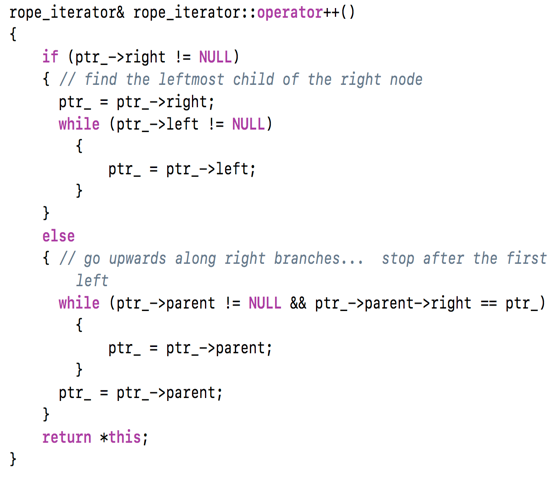
copy -> right = copy\_tree(p->right);

if (copy->right != NULL){

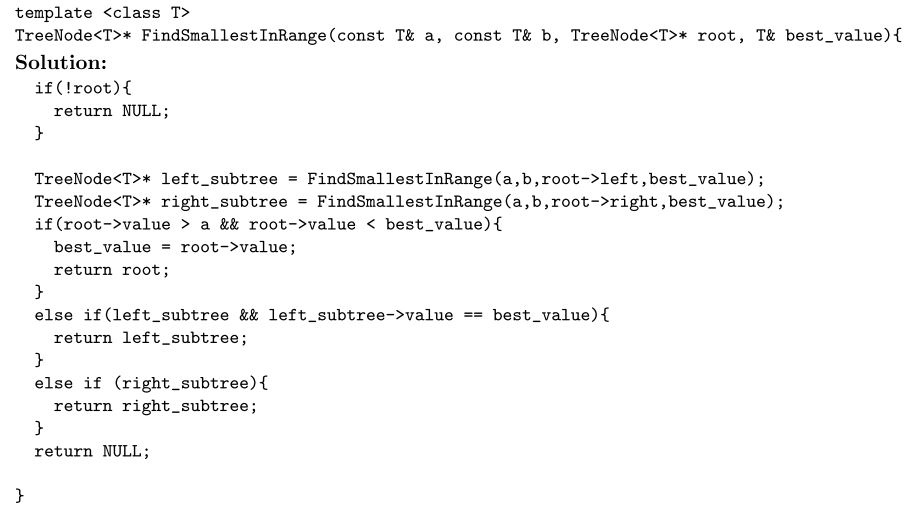
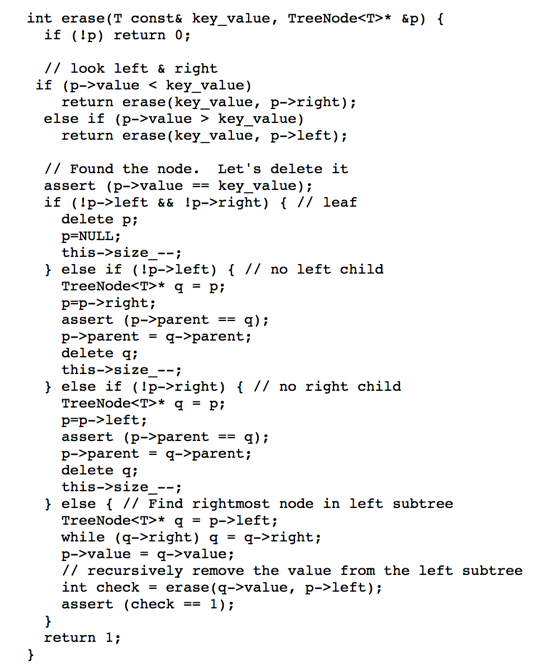
copy->right->parent = copy;

}

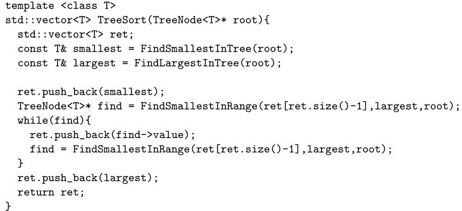
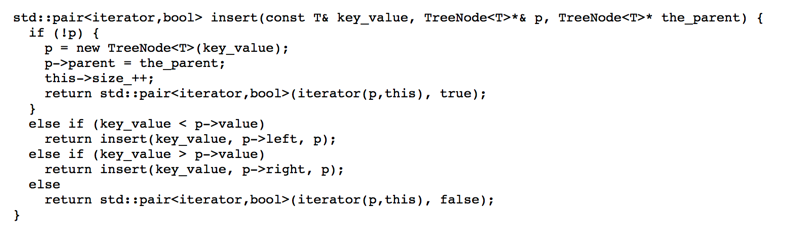
return copy;

 }

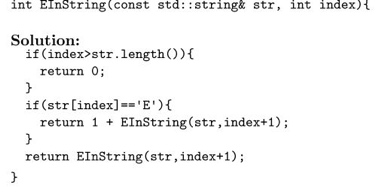
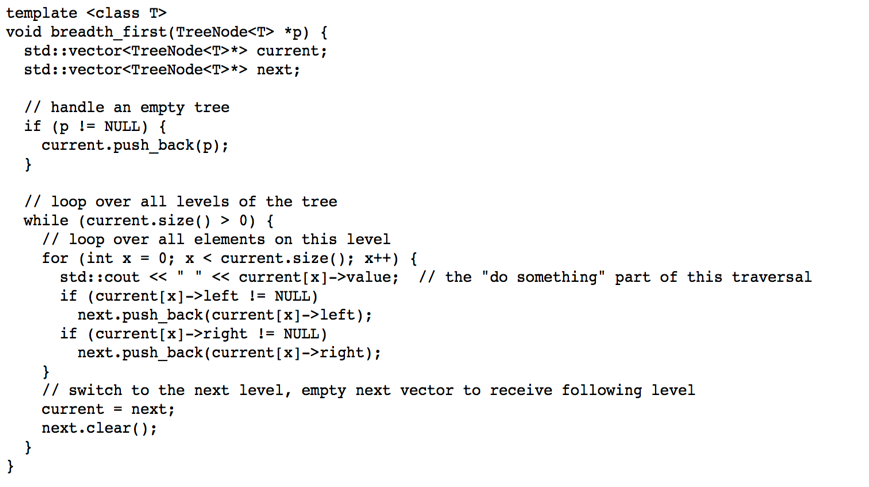
}



Tree sort:



在string查找char:



|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Singled-linked list | STL list |
| size | √ O(1) | √ O(n) | O(1) |
| push\_back | √ O(1) | √ O(n) | O(1) |
| push\_front |  | √ O(1) | O(1) |
| pop\_back | √ O(1) | √ O(n) | O(1) |
| pop\_front |  | √ O(1) | O(1) |
| erase | √ O(n) | √ O(1) | O(1) |
| insert | √ O(n) | √ O(1) | O(1) |

