

目录与绪论

OVERALL

本实验报告的基本结构如下（按逻辑顺序）：

模块	编号	内容
STL 风格的 泛型的 基础数据结构 容器实现	1	基于双向链表的 <code>linkedList</code>
	2	基于增长数组的 <code>vector</code>
	3	基于块状数组的 <code>dataBlock</code>
	4	实现基于循环增长数组的 <code>deque</code>
	5	基于 <code>vector</code> 实现 <code>stack</code>
	10	基于 R-BTree 实现 <code>set</code>
	11	基于 R-BTree 实现 <code>map</code>
	15	基于 Heap 实现 <code>priority_queue</code>
基础树/图结构	6	树上 <code>dfs</code> (基础信息)
	7	图上 <code>bfs</code> (最短路)
	9	R-BTree 的基本实现
特殊结构 及其应用	12	字典树 <code>Trie</code>
	13	线段树 <code>segTree</code>
	14	堆 <code>Heap</code>
	16	霍夫曼树 <code>Huffman-tree</code>
在算法中应用	17	算数表达式求值 (栈)
	18	括号匹配 (栈)
	19	高精度计算

本实验报告的所有代码文件、commit 记录等已经在 GITHUB 上同步了所有相关资料: <https://github.com/GYPpro/DS-Course-Report>

本 pdf 提供了完整的目录，强烈建议在 GITHUB 上下载阅读

使用 `typst` 渲染

仓库会在 7 月 10 日前保持 `private`

这两页只是用来大概描述用的，归档时可以撕去

容器设计原则

INTERFACE REFERENCE

Many containers have several member functions in common, and share functionalities. The decision of which type of container to use for a specific need does not generally depend only on the functionality offered by the container, but also on the efficiency of some of its members (complexity). This is especially true for sequence containers, which offer different trade-offs in complexity between inserting/removing elements and accessing them.

stack, queue and priority_queue are implemented as container adaptors. Container adaptors are not full container classes, but classes that provide a specific interface relying on an object of one of the container classes (such as deque or list) to handle the elements. The underlying container is encapsulated in such a way that its elements are accessed by the members of the container adaptor independently of the underlying container class used.

在模块 **STL** 风格的、泛型的、基础数据结构容器实现 中所有的数据结构，遵循以下规范：

- 提供增删改查的接口；
- 提供完整的泛型系统；
- 提供迭代器用于遍历；
- 按结构性质提供随机访问

为了更契合 **OOP** 中的 **SRP** 原则，对于原 **STL** 中的容器适配器，本实验报告中并不通过接口实现，而是另起一个类。

暨南大学本科实验报告专用纸

数据结构 课程实验项目目录

学生姓名：郭彦培 学号：2022101149 专业：信息管理与信息系统

序号	实验项目 编号	实验项目名称	*实验项 目类型	成绩	指导教师
1	1	基于双向链表的 linkedList	设计性		干晓聪
2	2	基于增长数组的 vector	设计性		干晓聪
3	3	基于块状数组的 dataBlock	设计性		干晓聪
4	4	实现基于循环增长数组的 deque	设计性		干晓聪
5	5	基于 vector 实现 stack	设计性		干晓聪
6	6	树上 dfs (基础信息)	设计性		干晓聪
7	7	图上 bfs (最短路)	设计性		干晓聪
8	9	R-BTree 的基本实现	设计性		干晓聪
9	10	基于 R-BTree 实现 set	设计性		干晓聪
10	11	基于 R-BTree 实现 map	设计性		干晓聪
11	12	字典树 Trie	设计性		干晓聪
12	13	线段树 segTree	设计性		干晓聪
13	14	堆 Heap	设计性		干晓聪
14	15	基于 Heap 实现 priority_queue	设计性		干晓聪
15	16	霍夫曼树 Huffman - tree	设计性		干晓聪
16	17	算数表达式求值 (栈)	设计性		干晓聪
17	18	括号匹配 (栈)	设计性		干晓聪
18	19	高精度计算	设计性		干晓聪

*实验项目类型：演示性、验证性、综合性、设计性实验。

*此表由学生按顺序填写。

暨南大学本科实验报告专用纸(附页)

基于双向链表的 `linkedList`

课程名称 数据结构 成绩评定

实验项目名称 基于双向链表的 `linkedList` 指导老师 干晓聪

实验项目编号 01 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现一个双向列表类，在类中实现增、删、改、查的方法并完成测试

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

暨南大学本科实验报告专用纸(附页)

3. 程序代码

3.1. linkedList.h

```
 1 // #define _PRIVATE_DEBUG
 2 #ifndef LINKED_LIST_HPP
 3 #define LINKED_LIST_HPP
 4
 5 #ifdef _PRIVATE_DEBUG
 6 #include <iostream>
 7 #endif
 8
 9 namespace myDS
10 {
11     template<typename VALUE_TYPE>
12     class linkedList{
13     protected:
14         class linkedNode {
15     public:
16             VALUE_TYPE data = VALUE_TYPE();
17             linkedNode * next = nullptr;
18             linkedNode * priv = nullptr;
19
20             linkedNode() { }
21
22             linkedNode(VALUE_TYPE _data){
23                 next = nullptr;
24                 priv = nullptr;
25                 data = _data;
26             }
27
28             linkedNode(VALUE_TYPE _data,linkedNode * priv)
29             {
30                 next = nullptr;
31                 priv = priv;
32                 data = _data;
33             }
34
35             ~linkedNode() {
36 #ifdef _PRIVATE_DEBUG
37                 // if(this->next != nullptr)
38                 //     std::cout << "Unexpected Delete at :" << this->data
39                 //         << " with next:" << this->next->data << "\n";
40             }
41
42             linkedNode * linkNext(linkedNode * _next)
43             {
44                 next = _next;
45                 _next->priv = this;
46                 return this->next;
47             }
48
49         };
50     };
51 }
```

暨南大学本科实验报告专用纸(附页)

```
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98

    }

    linkedNode * linkPriv(linkedNode *_priv)
    {
        priv = _priv;
        _priv->next = this;
        return this->priv;
    }

    void insertNext(linkedNode *_inst){
        if(_inst == nullptr) return;
        if(this->next == nullptr) linkNext(_inst);
        else {
            _inst->next = this->next;
            this->next->priv = _inst;
            _inst->priv = this;
            this->next = _inst;
        }
    }

    void deleteNext()
    {
        if(this->next == nullptr) return;
        else {
            linkedNode * tmp = this->next;
            this->next = this->next->next;
            this->next->priv = this;
            tmp->next = nullptr;
            delete tmp;
        }
    }
};

private:
    class _iterator
{
private:
    linkedNode *_ptr;

public:
    enum __iter_dest_type
    {
        front,
        back
    };
    __iter_dest_type _iter_dest;

    _iterator(linkedNode *_upper,__iter_dest_type _d)
    {
        _ptr = _upper;
        _iter_dest = _d;
    }
};
```

暨南大学本科实验报告专用纸(附页)

```
99
100    VALUE_TYPE & operator*()
101    {
102        return _ptr->data;
103    }
104
105    VALUE_TYPE *operator->()
106    {
107        return _ptr;
108    }
109
110    myDS::linkedList<VALUE_TYPE>::_iterator operator++()
111    {
112        if (_iter_dest == front)
113            _ptr = _ptr->next;
114        else
115            _ptr = _ptr->priv;
116        return *this;
117    }
118
119    myDS::linkedList<VALUE_TYPE>::_iterator operator++(int)
120    {
121        myDS::linkedList<VALUE_TYPE>::_iterator old = *this;
122        if (_iter_dest == front)
123            _ptr = _ptr->next;
124        else
125            _ptr = _ptr->priv;
126        return old;
127    }
128
129    // myDS::linkedList<VALUE_TYPE>::_iterator operator+(size_t
130    _n)
131    // {
132    //     if (_iter_dest == front)
133    //         _upper_idx += _n;
134    //     else
135    //         _upper_idx -= _n;
136    //     _ptr = &(*_upper_pointer)[_upper_idx];
137    //     return *this;
138    // }
139
140    bool operator==( myDS::linkedList<VALUE_TYPE>::_iterator _b)
141    {
142        if (&(*_b) == _ptr)
143            return 1;
144        else
145            return 0;
146    }
147
148    bool operator!=( myDS::linkedList<VALUE_TYPE>::_iterator _b)
149    {
150        if (&(*_b) == &(_ptr->data))
```

暨南大学本科实验报告专用纸(附页)

```
150         return 0;
151     else
152         return 1;
153     }
154 };
155
156 linkedNode * head = new linkedNode();
157 linkedNode * tail = new linkedNode();
158 int cap = 0;
159
160 public:
161     linkedList(){
162         head->linkNext(tail);
163     }
164
165     ~linkedList(){
166         clear();
167         delete head;
168         delete tail;
169     }
170
171     void push_back(VALUE_TYPE t) {
172         tail->data = t;
173         tail->linkNext(new linkedNode());
174         tail = tail->next;
175         cap++;
176     }
177
178     void push_frount(VALUE_TYPE t) {
179         head->data = t;
180         head = (head->linkPriv(new linkedNode()));
181         cap++;
182     }
183
184     void clear() {
185         linkedNode * deletingObject;
186         while(tail->priv != head) {
187             deletingObject = tail;
188             tail = tail->priv;
189             delete deletingObject;
190         }
191         cap = 0;
192         delete head;
193         delete tail;
194         tail = new linkedNode();
195         head = new linkedNode();
196         head->linkNext(tail);
197     }
198
199     std::size_t erase(VALUE_TYPE p) {
200         linkedNode * ptr = head;
201         int ttl = 0;
```

暨南大学本科实验报告专用纸(附页)

```
202     while(ptr->next != tail) {
203         if(ptr->next->data == p){
204             ptr->deleteNext();
205             ttl++;
206         } else ptr = ptr->next;
207     }
208     cap -= ttl;
209     return ttl;
210 }
211
212 std::size_t size() {return cap;}
213
214 bool erase(linkedList<VALUE_TYPE>::_iterator p) {
215     myDS::linkedList<VALUE_TYPE>::_iterator ptr = this-
216     >begin();
217     linkedNode * cur = head;
218     while(ptr != p) {
219         cur = cur->next;
220         ptr++;
221         if(cur == tail) return 0;
222     }
223     cur->deleteNext();
224     cap--;
225     return 1;
226 }
227
228     myDS::linkedList<VALUE_TYPE>::_iterator begin() {
229         enum
230             myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type _FRONT =
231                 myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type::front;
232             return myDS::linkedList<VALUE_TYPE>::_iterator(head-
233             >next,_FRONT);
234
235         myDS::linkedList<VALUE_TYPE>::_iterator rbegin() {
236             enum
237                 myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type _BACK =
238                     myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type::back;
239                     return myDS::linkedList<VALUE_TYPE>::_iterator(tail-
240             >priv,_BACK);
241
242         myDS::linkedList<VALUE_TYPE>::_iterator end() {
243             enum
244                 myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type _FRONT =
245                     myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type::front;
246                     return
247                         myDS::linkedList<VALUE_TYPE>::_iterator(tail,_FRONT);
248                 }
```

暨南大学本科实验报告专用纸(附页)

```
242     myDS::linkedList<VALUE_TYPE>::_iterator rend() {
243         enum
244             myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type _BACK =
245                 myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type::back;
246                     return myDS::linkedList<VALUE_TYPE>::_iterator(head,_BACK);
247     }
248
249     myDS::linkedList<VALUE_TYPE>::_iterator get(std::size_t p) {
250         linkedNode * ptr = head->next;
251         while(p --) ptr = ptr->next;
252         enum
253             myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type _FRONT =
254                 myDS::linkedList<VALUE_TYPE>::_iterator::_iter_dest_type::front;
255                     return myDS::linkedList<VALUE_TYPE>::_iterator(ptr,_FRONT);
256     }
257
258     VALUE_TYPE & operator[](std::size_t p) {
259         linkedNode * ptr = head;
260         while(p --) ptr = ptr->next;
261         return ptr->next->data;
262     }
263
264 #ifdef _PRIVATE_DEBUG
265     void innerPrint()
266     {
267         std::cout << "--Header[" << head << "]: " << head->data <<
268         "\n";
269         std::cout << "--Tail[" << tail << "]: " << tail->data <<
270         "\n";
271         std::cout << "-----\n";
272         std::cout << "cur:" << cap<< "\n";
273         auto ptr = head;
274         do {
275             std::cout << "[" << ptr << "] ->next:" << ptr->next <<
276             "->priv:" << ptr->priv << " |data:" << ptr->data << "\n";
277             ptr = ptr->next;
278         }while(ptr != nullptr);
279     }
280 #endif
281 };
282 }
283 #endif
```

3.2. _PRIV_TEST.cpp

```
1 #define DS_TOBE_TEST linkedList
2
3 #define _PRIVATE_DEBUG
```

暨南大学本科实验报告专用纸(附页)

```
4 #include "Dev\01\linkedList.h"
5
6
7 #include <iostream>
8 #include <math.h>
9 #include <vector>
10
11 using namespace std;
12
13 using TBT = myDS::DS_TOBE_TEST<int>;
14
15 void accuracyTest() {//结构正确性测试
16
17     TBT tc = TBT();
18     for(;;)
19     {
20         string op;
21         cin >> op;
22         if(op == "clr") { //清空
23             tc.clear();
24         } else if(op == "q") //退出测试
25         {
26             return;
27         } else if(op == "pb")//push_back
28         {
29             int c;
30             cin >> c;
31             tc.push_back(c);
32         } else if(op == "pf")//push_frount
33         {
34             int c;
35             cin >> c;
36             tc.push_frount(c);
37         } else if(op == "at")//随机访问
38         {
39             int p;
40             cin >> p;
41             cout << tc[p] << "\n";
42         } else if(op == "delEL")//删除所有等于某值元素
43         {
44             int p;
45             cin >> p;
46             cout << tc.erase(p) << "\n";
47         } else if(op == "delPS")//删除某位置上的元素
48         {
49             int p;
50             cin >> p;
51             cout << tc.erase(tc.get(p)) << "\n";
52         } else if(op == "iterF") //正序遍历
53         {
54             tc.innerPrint();
```

暨南大学本科实验报告专用纸(附页)

```
55         cout << "Iter with index:\n";
56         for(int i = 0;i < tc.size();i++) cout << tc[i] << " ";cout
57             << "\n";
58         cout << "Iter with begin end\n";
59         for(auto x = tc.begin();x != tc.end();x++) cout << (*x) <<
60             " ";cout << "\n";
61         cout << "Iter with AUTO&&\n";
62         for(auto x:tc) cout << x << " ";cout << "\n";
63     } else if(op == "iterB") //正序遍历
64     {
65         tc.innerPrint();
66         cout << "Iter with index:\n";
67         for(int i = 0;i < tc.size();i++) cout << tc[tc.size()-1-i]
68             << " ";cout << "\n";
69         cout << "Iter with begin end\n";
70         for(auto x = tc.rbegin();x != tc.rend();x++) cout << (*x)
71             << " ";cout << "\n";
72         // cout << "Iter with AUTO&&\n"; ."\n";
73     } else if(op == "mv")//单点修改
74     {
75         int p;
76         cin >> p;
77         int tr;
78         cin >> tr;
79         tc[p] = tr;
80     } else if(op == "") {
81         op.clear();
82     }
83 }
84
85
86
87
88 void memLeakTest() {//内存泄漏测试
89     TBT tc = TBT();
90     for(;){
91         tc.push_back(1);
92         tc.push_back(1);
93         tc.push_back(1);
94         tc.push_back(1);
95         tc.clear();
96     }
97 }
98
99 signed main()
100 {
101     // accuracyTest();
```

暨南大学本科实验报告专用纸(附页)

```
102     memLeakTest();  
103 }
```

4. 测试数据与运行结果

运行上述 _PRIV_TEST.cpp 测试代码中的正确性测试模块，得到以下内容：

```
pb 1  
pb 2  
pb 3  
pb 4  
pf 3  
pb 3  
iterF  
iterB  
delEL 3  
iterF  
delPS 1  
clr  
pb 1  
pb 2  
iterF  
delPS 0  
delEL 2  
iterF  
  
pb 1  
pb 2  
pb 3  
pb 4  
pf 3  
pb 3  
iterF  
--Header[0x662720]: 0  
--Tail[0x662770]: 0  
-----  
cur:6  
[0x662720] ->next:0x662540 ->priv:0 ||data:0  
[0x662540] ->next:0x662590 ->priv:0x662720 ||data:3  
[0x662590] ->next:0x6625e0 ->priv:0x662540 ||data:1  
[0x6625e0] ->next:0x662630 ->priv:0x662590 ||data:2  
[0x662630] ->next:0x662680 ->priv:0x6625e0 ||data:3
```

暨南大学本科实验报告专用纸(附页)

```
[0x662680] ->next:0x6626d0 ->priv:0x662630 ||data:4
[0x6626d0] ->next:0x662770 ->priv:0x662680 ||data:3
[0x662770] ->next:0 ->priv:0x6626d0 ||data:0
Iter with index:
3 1 2 3 4 3
Iter with begin end
3 1 2 3 4 3
Iter with AUTO&&
3 1 2 3 4 3
    iterB
--Header[0x662720]: 0
--Tail[0x662770]: 0
-----
cur:6
[0x662720] ->next:0x662540 ->priv:0 ||data:0
[0x662540] ->next:0x662590 ->priv:0x662720 ||data:3
[0x662590] ->next:0x6625e0 ->priv:0x662540 ||data:1
[0x6625e0] ->next:0x662630 ->priv:0x662590 ||data:2
[0x662630] ->next:0x662680 ->priv:0x6625e0 ||data:3
[0x662680] ->next:0x6626d0 ->priv:0x662630 ||data:4
[0x6626d0] ->next:0x662770 ->priv:0x662680 ||data:3
[0x662770] ->next:0 ->priv:0x6626d0 ||data:0
Iter with index:
3 4 3 2 1 3
Iter with begin end
3 4 3 2 1 3
    delEL 3
3
    iterF
--Header[0x662720]: 0
--Tail[0x662770]: 0
-----
cur:3
[0x662720] ->next:0x662590 ->priv:0 ||data:0
[0x662590] ->next:0x6625e0 ->priv:0x662720 ||data:1
[0x6625e0] ->next:0x662680 ->priv:0x662590 ||data:2
[0x662680] ->next:0x662770 ->priv:0x6625e0 ||data:4
[0x662770] ->next:0 ->priv:0x662680 ||data:0
Iter with index:
1 2 4
Iter with begin end
1 2 4
Iter with AUTO&&
1 2 4
```

暨南大学本科实验报告专用纸(附页)

```
delPS 1
1
clr
Unexpected Delete at :4 with next:16187728
pb 1
pb 2
iterF
--Header[0x6625e0]: 0
--Tail[0x662680]: 0
-----
cur:2
[0x6625e0] ->next:0x662540 ->priv:0 ||data:0
[0x662540] ->next:0x662630 ->priv:0x6625e0 ||data:1
[0x662630] ->next:0x662680 ->priv:0x662540 ||data:2
[0x662680] ->next:0 ->priv:0x662630 ||data:0
Iter with index:
1 2
Iter with begin end
1 2
Iter with AUTO&&
1 2
    delPS 0
1
    delEL 2
1
    iterF
--Header[0x6625e0]: 0
--Tail[0x662680]: 0
-----
cur:0
[0x6625e0] ->next:0x662680 ->priv:0 ||data:0
[0x662680] ->next:0 ->priv:0x6625e0 ||data:0
Iter with index:

Iter with begin end

Iter with AUTO&&
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

运行 `_PRIV_TEST.cpp` 中的内存测试模块，在保持 CPU 高占用率运行一段时间后内存变化符合预期，可以认为代码内存安全性良好。

暨南大学本科实验报告专用纸(附页)

后台进程 (145)					
	_PRIV_TEST.exe		9.2%	0.7 MB	
			0%	0.6 MB	

暨南大学本科实验报告专用纸(附页)

基于增长数组的 `vector`

课程名称 数据结构 成绩评定

实验项目名称 基于增长数组的 `vector` 指导老师 干晓聪

实验项目编号 02 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现基于增长数组的类 STL `vector` 类，提供尾端插入与随机访问和迭代器。

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

3. 程序原理

增长数组会周期性的申请连续的内存，并将以往的数据移动到新申请的内存中。其倍增的特性保证了其均摊的插入复杂度在 $\mathcal{O}(1)$ ，其连续的性质保证了随机访问的速度。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. memDeleteTest.cpp

```
1 #include <iostream>
2 #include <new>
3 #include <stdlib.h>
4 using namespace std;
5
6 class testClass{
7 public:
8     int a = 0;
9     testClass(){a=1;};
10    ~testClass(){cout << "Destroy TestClass\n";};
11 };
12 int main()
13 {
14     testClass * arr = new testClass[10];
15     cout << "Finish Alloc\n";
16     for(int i = 0;i < 10;i++)
17         arr[i].~testClass();
18     if(arr)
19         //delete[] arr;
20         ::operator delete[](arr);
21     else cout << "nullPtr\n";
22     cout << "Finish Delete\n";
23     return 0;
24 }
```

4.2. _PRIV_TEST.cpp

```
1 #define DS_TOBE_TEST vector
2
3 #define _PRIVATE_DEBUG
4
5 #include "Dev\02\myVector.h"
6
7 #include <iostream>
8 #include <math.h>
9 #include <vector>
10
11 using namespace std;
12
13 using TBT = myDS::DS_TOBE_TEST<int>;
14
15 void accuracyTest() {//结构正确性测试
16
17     TBT tc = TBT();
18     for(;;)
19     {
```

暨南大学本科实验报告专用纸(附页)

```
20     string op;
21     cin >> op;
22     if(op == "clr") { //清空
23         tc.clear();
24     } else if(op == "q") //退出测试
25     {
26         return;
27     } else if(op == "pb")//push_back
28     {
29         int c;
30         cin >> c;
31         tc.push_back(c);
32     } else if(op == "pf")//push_frount
33     {
34         int c;
35         cin >> c;
36         tc.push_frount(c);
37     } else if(op == "at")//随机访问
38     {
39         int p;
40         cin >> p;
41         cout << tc[p] << "\n";
42     } else if(op == "delEL")//删除所有等于某值元素
43     {
44         int p;
45         cin >> p;
46         cout << tc.erase(p) << "\n";
47     } else if(op == "delPS")//删除某位置上的元素
48     {
49         int p;
50         cin >> p;
51         cout << tc.erase(tc.get(p)) << "\n";
52     } else if(op == "iterF") //正序遍历
53     {
54         // tc.innerPrint();
55         cout << "Iter with index:\n";
56         for(int i = 0;i < tc.size();i++) cout << tc[i] << " ";cout
57         << "\n";
58         cout << "Iter with begin end\n";
59         for(auto x = tc.begin();x != tc.end();x++) cout << (*x) <<
60         " ";cout << "\n";
61         cout << "Iter with AUTO&&\n";
62         for(auto x:tc) cout << x << " ";cout << "\n";
63     } else if(op == "iterB") //正序遍历
64     {
65         // tc.innerPrint();
66         cout << "Iter with index:\n";
67         for(int i = 0;i < tc.size();i++) cout << tc[tc.size()-1-i]
68         << " ";cout << "\n";
```

暨南大学本科实验报告专用纸(附页)

```
66         cout << "Iter with begin end\n";
67         for(auto x = tc.rbegin();x != tc.rend();x++) cout << (*x)
68             << " ";cout << "\n";
69             // cout << "Iter with AUTO&&\n";.\n";
70         } else if(op == "mv")//单点修改
71         {
72             int p;
73             cin >> p;
74             int tr;
75             cin >> tr;
76             tc[p] = tr;
77         } else if(op == "")
78         {
79             } else {
80                 op.clear();
81             }
82         }
83     }
84
85
86
87
88 void memLeakTest() {//内存泄漏测试
89     TBT tc = TBT();
90     for(;){
91         tc.push_back(1);
92         tc.push_back(1);
93         tc.push_back(1);
94         tc.push_back(1);
95         tc.clear();
96     }
97 }
98
99 signed main()
100 {
101     accuracyTest();
102     // memLeakTest();
103 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
*
```

```
* @file myVector.h
* @brief A Memory-contiguous, variable-length array
* @details
```

暨南大学本科实验报告专用纸(附页)

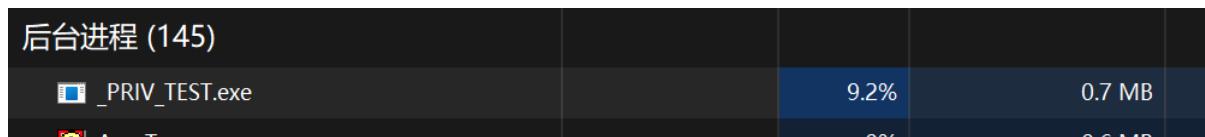
```
* 不知道该写什么，反正就是 vector，正常用就行了  
* @author github.com/GYPpro  
* @version 0.2.0
```

```
pb 1  
pb 2  
pb 3  
pb 4  
iterF  
pb 9  
iterB  
clr  
iterF  
pb 1  
iterF  
  
pb 1  
pb 2  
pb 3  
pb 4  
iterF  
Iter with index:  
1 2 3 4  
Iter with begin end  
1 2 3 4  
Iter with AUTO&&  
1 2 3 4  
pb 9  
iterB  
Iter with index:  
9 4 3 2 1  
Iter with begin end  
9 4 3 2  
clr  
iterF  
Iter with index:  
  
Iter with begin end  
  
Iter with AUTO&&
```

暨南大学本科实验报告专用纸(附页)

```
pb 1
iterF
Iter with index:
1
Iter with begin end
1
Iter with AUTO&&
1
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。



运行 `_PRIV_TEST.cpp` 中的内存测试模块，在保持 CPU 高占用率运行一段时间后内存变化符合预期，可以认为代码内存安全性良好。

暨南大学本科实验报告专用纸(附页)

基于块状数组的 `dataBlock`

课程名称 数据结构 成绩评定 _____

实验项目名称 基于块状数组的 `dataBlock` 指导老师 干晓聪

实验项目编号 03 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现基于 `vector` 的块状数组，针对插入场景进行特别优化。

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

3. 程序原理

在使用增长数组维护一个索引区域的基础上，使用不再进行移动的倍增数组维护动态扩容的数据。

具体的，每次扩容与 `vector` 类似，将新申请一个与当前内存相等大小的区域，将其索引插入索引区域，并保持原数组不变。

易得，本结构需要额外 $\mathcal{O}(\log_2 n)$ 的索引区域。

其申请与访问操作的复杂度分析大致如下：

`push_back` : $\mathcal{O}(1)$

`get_index` : $\log_{10}(\log_2(n)) \cdot n \rightarrow \mathcal{O}(\log(n))$

由于常数极小，在数据量小于 10^{20} 时可以认为 `get_index` 的复杂度为 1

特别的，在数据后半段，内存区间连续，依旧能享受到 CPU 分支优化。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. dataBlock.hpp

```
 1 // #define _PRIVATE_DEBUG
 2 #ifndef DATA_BLOCK_HPP
 3 #define DATA_BLOCK_HPP
 4
 5 #include <vector>
 6 #include <map>
 7
 8 #define _PRIVATE_DEBUG
 9
10 #ifdef _PRIVATE_DEBUG
11 #include <iostream>
12 #endif
13
14 namespace myDS
15 {
16     template<typename VALUE_TYPE>
17     class dataBlock{
18     protected:
19
20     private:
21         class _iterator
22         {
23             private:
24                 VALUE_TYPE *_ptr;
25                 std::pair<std::size_t, std::size_t> loc;
26                 dataBlock<VALUE_TYPE> *_upper_pointer;
27
28             public:
29                 enum __iter_dest_type
30                 {
31                     front,
32                     back
33                 };
34                 __iter_dest_type _iter_dest;
35
36             _iterator( myDS::dataBlock<VALUE_TYPE>
37             *_upper, std::pair<std::size_t, std::size_t> _loc, __iter_dest_type _d)
38             {
39                 _upper_pointer = _upper;
40                 loc = _loc;
41                 _ptr = &_upper_pointer->indexes[loc.first][loc.second];
42                 _iter_dest = _d;
43             }
44
45             VALUE_TYPE & operator*()
46             {
47                 return (*_ptr);
```

暨南大学本科实验报告专用纸(附页)

```
47 }
48
49     VALUE_TYPE *operator->()
50 {
51     return _ptr;
52 }
53
54     myDS::dataBlock<VALUE_TYPE>::_iterator operator++() {
55         if(_iter_dest == front)
56         {
57             loc = _upper_pointer->nextPII(loc);
58         }
59         else
60         {
61             loc = _upper_pointer->prevPII(loc);
62         }
63         _ptr = &_upper_pointer->_indexes[loc.first][loc.second];
64         return
65     myDS::dataBlock<VALUE_TYPE>::_iterator(_upper_pointer,loc,_iter_dest);
66 }
67
68     myDS::dataBlock<VALUE_TYPE>::_iterator operator++(int) {
69         myDS::dataBlock<VALUE_TYPE>::_iterator old = *this;
70         if(_iter_dest == front)
71         {
72             loc = _upper_pointer->nextPII(loc);
73         }
74         else
75         {
76             loc = _upper_pointer->prevPII(loc);
77         }
78         _ptr = &_upper_pointer->_indexes[loc.first][loc.second];
79         return old;
80     }
81
82     bool operator==( myDS::dataBlock<VALUE_TYPE>::_iterator _b)
83     {
84         return _ptr == _b._ptr;
85     }
86     bool operator!=( myDS::dataBlock<VALUE_TYPE>::_iterator _b)
87     {
88         return _ptr != _b._ptr;
89     };
90
91     std::vector<VALUE_TYPE *> _indexes;
92     std::pair<std::size_t,std::size_t> _cap = {0,0};
93     std::size_t consMEX = 1;
94     std::size_t _size = 0;
95 }
```

暨南大学本科实验报告专用纸(附页)

```
96
97
98
99
100
101
102
103
104
105     void _expension()
106     {
107         VALUE_TYPE *temp = new VALUE_TYPE[consMEX];
108         _indexs.push_back(temp);
109         consMEX *= 2;
110         _cap.first++;
111         _cap.second = 0;
112     }
113
114     std::size_t getMEX(std::int32_t p)
115     {
116         if(p <= 0) return p+1;
117         return (1 << (p-1));
118     }
119
120     std::pair<std::size_t, std::size_t>
121 nextPII(std::pair<std::size_t, std::size_t> p)
122     {
123         p.second++;
124         if(p.second >= getMEX(p.first))
125         {
126             p.first++;
127             p.second = 0;
128         }
129         return p;
130     }
131
132     std::pair<std::size_t, std::size_t>
133 prevPII(std::pair<std::size_t, std::size_t> p)
134     {
135 #ifdef __DETEIL_DEBUG_OUTPUT
136         std::cout << "{" << p.first << "," << p.second << "}'s prev
137 is";
138 #endif
139
140         std::int32_t tmp = p.second;
141         tmp--;
142         if(tmp < 0)
143         {
144             p.first--;
145             p.second = getMEX(p.first) - 1;
146         } else p.second--;
147 #ifdef __DETEIL_DEBUG_OUTPUT
148         std::cout << "{" << p.first << "," << p.second << "}\n";
149 #endif
150
151         return p;
152     }
153
154     public:
155     dataBlock(){
```

暨南大学本科实验报告专用纸(附页)

```
144     VALUE_TYPE *tmp = new VALUE_TYPE[1];
145     _indexs.push_back(tmp);
146 }
147
148 ~dataBlock(){
149     clear();
150     delete [] (_indexs[0]);
151 }
152
153 void push_back(VALUE_TYPE t) {
154     if(_cap.second >= getMEX(_cap.first)) {
155         _expension();
156     }
157     _indexs[_cap.first][_cap.second] = t;
158     _cap.second++;
159     _size++;
160 }
161
162 void clear() {
163     for(auto x:_indexs) delete [] x;
164     _indexs.clear();
165     VALUE_TYPE *tmp = new VALUE_TYPE[1];
166     _indexs.push_back(tmp);
167     consMEX = 1;
168     _size = 0;
169     _cap = {0,0};
170 }
171
172 std::size_t size() {
173     return _size;
174 }
175
176     myDS::dataBlock<VALUE_TYPE>::_iterator begin() {
177         return myDS::dataBlock<VALUE_TYPE>::_iterator(this,{0,0}),
178     myDS::dataBlock<VALUE_TYPE>::_iterator::front);
179     }
180
181     myDS::dataBlock<VALUE_TYPE>::_iterator rbegin() {
182         return
183     myDS::dataBlock<VALUE_TYPE>::_iterator(this,prevPII(_cap),
184     myDS::dataBlock<VALUE_TYPE>::_iterator::back);
185     }
186
187     myDS::dataBlock<VALUE_TYPE>::_iterator end() {
188         return
189     myDS::dataBlock<VALUE_TYPE>::_iterator(this,nextPII(prevPII(_cap)),
190     myDS::dataBlock<VALUE_TYPE>::_iterator::front);
191     }
192
193     myDS::dataBlock<VALUE_TYPE>::_iterator rend() {
```

暨南大学本科实验报告专用纸(附页)

```
        return  
189     myDS::dataBlock<VALUE_TYPE>::_iterator(this,prevPII({0,0}),  
190     myDS::dataBlock<VALUE_TYPE>::_iterator::back);  
191     }  
192 #ifdef _PRIVATE_DEBUG  
193     void innerPrint() {  
194         std::pair<std::size_t,std::size_t> p = {0,0};  
195         while(p.first <= _cap.first) {  
196             if(p.second == 0) std::cout << "\nBlock : [" << p.first  
197             << "] at:" << _indexs[p.first] << "\n";  
198             std::cout << _indexs[p.first][p.second] << " ";  
199             p = nextPII(p);  
200         }  
201         std::cout << "\n";  
202     }  
203 #endif  
204     VALUE_TYPE & operator[](std::size_t p) {  
205         if(p == 0) return _indexs[0][0];  
206         std::int32_t onord = 0;  
207         std::size_t tmp = p;  
208         while(tmp) {  
209             tmp >>= 1;  
210             onord++;  
211         }  
212 #ifdef __DETIL_DEBUG_OUTPUT  
213         std::cout << "onord:" << onord << " p:" << p << "  
213 GETMEX :"<< getMEX(onord) << " index:{" << onord << "," << p -  
213 getMEX(onord) << "}\n";  
214 #endif  
215         return _indexs[onord][p - getMEX(onord)];  
216     }  
217 }  
218 };  
219 #endif
```

4.2. _PRIV_TEST.cpp

```
#define DS_TOBE_TEST dataBlock  
  
#define _PRIVATE_DEBUG  
// #define __DETIL_DEBUG_OUTPUT  
  
#include "Dev\03\dataBlock.hpp"  
  
#include <time.h>
```

暨南大学本科实验报告专用纸(附页)

```
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

using TBT = myDS::dataBlock<int>;

void accuracyTest() {//结构正确性测试

    TBT tc = TBT();
    for(;;)
    {
        string op;
        cin >> op;
        if(op == "clr") { //清空
            tc.clear();
        } else if(op == "q") //退出测试
        {
            return;
        } else if(op == "pb")//push_back
        {
            int c;
            cin >> c;
            tc.push_back(c);
        } else if(op == "pf")//push_frount
        {
            int c;
            cin >> c;
            tc.push_frount(c);
        } else if(op == "at")//随机访问
        {
            int p;
            cin >> p;
            cout << tc[p] << "\n";
        } else if(op == "delEL")//删除所有等于某值元素
        {
            int p;
            cin >> p;
            cout << tc.erase(p) << "\n";
        } else if(op == "delPS")//删除某位置上的元素
        {
            int p;
            cin >> p;
```

暨南大学本科实验报告专用纸(附页)

```
//      cout << tc.erase(tc.get(p)) << "\n";
} else if(op == "iterF") //正序遍历
{
    tc.innerPrint();
    cout << "Iter with index:\n";
    for(int i = 0;i < tc.size();i++) cout << tc[i] << " ";cout <<
"\n";
    cout << "Iter with begin end\n";
    for(auto x = tc.begin();x != tc.end();x++) cout << (*x) << "
";cout << "\n";
    cout << "Iter with AUTO&&\n";
    for(auto x:tc) cout << x << " ";cout << "\n";
} else if(op == "iterB") //倒序遍历
{
    tc.innerPrint();
    cout << "Iter with index:\n";
    for(int i = 0;i < tc.size();i++) cout << tc[tc.size()-1-i] <<
" ";cout << "\n";
    cout << "Iter with begin end\n";
    for(auto x = tc.rbegin();x != tc.rend();x++) cout << (*x) << "
";cout << "\n";
    // cout << "Iter with AUTO&&\n";.\n";
} else if(op == "mv")//单点修改
{
    int p;
    cin >> p;
    int tr;
    cin >> tr;
    tc[p] = tr;
} else if(op == "") {
}
} else {
    op.clear();
}
}

void memLeakTest() {//内存泄漏测试
TBT tc = TBT();
for(;;){
    tc.push_back(1);
    tc.push_back(1);
    tc.push_back(1);
}
```

暨南大学本科实验报告专用纸(附页)

```
        tc.push_back(1);
        tc.clear();
    }
}

void speedTest()
{
    TBT tc = TBT();
    int begin = clock();
    int N = 1e8;
    for(int i = 0;i < N;i++)
    {
        tc.push_back(i);
    }
    cout << "myDS::dataBlock Push_back 10000000 elements cost:" << clock() -
begin << "ms\n";

    std::vector<int> tmp;
    begin = clock();
    for(int i = 0;i < N;i++)
    {
        tmp.push_back(i);
    }
    cout << "std::vector push_back 10000000 elements cost:" << clock() -
begin << "ms\n";
    system("pause");
}

signed main()
{
    // accuracyTest();
    // memLeakTest();
    speedTest();
}
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

暨南大学本科实验报告专用纸(附页)

```
pb 1
pb 2
pb 3
pb 4
iterF
iterB
clr
pb 0
pb 3
pb 1
pb 2
pb 3
pb 4
iterF
iterB
mv 0 3
iterF
pb 1
pb 2
pb 3
pb 4
iterF

Block : [0] at:0x722540
1
Block : [1] at:0x7225c0
2
Block : [2] at:0x722580
3 4
Iter with index:
1 2 3 4
Iter with begin end
1 2 3 4
Iter with AUTO&&
1 2 3 4
    iterB

Block : [0] at:0x722540
1
Block : [1] at:0x7225c0
2
Block : [2] at:0x722580
3 4
```

暨南大学本科实验报告专用纸(附页)

```
Iter with index:  
4 3 2 1  
Iter with begin end  
4 3 2 1  
clr  
pb 0  
pb 3  
pb 1  
pb 2  
pb 3  
pb 4  
iterF  
  
Block : [0] at:0x722540  
0  
Block : [1] at:0x722580  
3  
Block : [2] at:0x7225c0  
1 2  
Block : [3] at:0x722600  
3 4 -1163005939 -1163005939  
Iter with index:  
0 3 1 2 3 4  
Iter with begin end  
0 3 1 2 3 4  
Iter with AUTO&&  
0 3 1 2 3 4  
iterB  
  
Block : [0] at:0x722540  
0  
Block : [1] at:0x722580  
3  
Block : [2] at:0x7225c0  
1 2  
Block : [3] at:0x722600  
3 4 -1163005939 -1163005939  
Iter with index:  
4 3 2 1 3 0  
Iter with begin end  
4 3 2 1 3 0  
mv 0 3  
iterF
```

暨南大学本科实验报告专用纸(附页)

```
Block : [0] at:0x722540
3
Block : [1] at:0x722580
3
Block : [2] at:0x7225c0
1 2
Block : [3] at:0x722600
3 4 -1163005939 -1163005939
Iter with index:
3 3 1 2 3 4
Iter with begin end
3 3 1 2 3 4
Iter with AUTO&&
3 3 1 2 3 4
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

运行 `_PRIV_TEST.cpp` 中的内存测试模块，在保持 CPU 高占用率运行一段时间后内存变化符合预期，可以认为代码内存安全性良好。

状态	CPU	内存
_PRIV_TEST.exe	15.9%	0.6 MB

运行 `_PRIV_TEST.cpp` 中的性能测试模块，得到

```
myDS::dataBlock Push_back 10000000 elements cost:663ms
std::vector push_back 10000000 elements cost:1618ms
```

可以看到 `dataBlock` 在插入速度上较 `STL` 中的 `vector` 快 2-3 倍左右。

暨南大学本科实验报告专用纸(附页)

实现基于循环增长数组的 deque

课程名称 数据结构 成绩评定

实验项目名称 实现基于循环增长数组的 deque 指导老师 干晓聪

实验项目编号 04 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现基于循环增长数组的双向队列，保证在某一段重复添加弹出后实际内存占用规模符合理论占用，不会出现方向性泄漏。

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

3. 程序原理

在类 `deque` 中维护了两个指针与两个循环增长数组。如果某一端的长度偏差值大于 1，即某侧数据长小于同侧空白区域长度，则触发再分配。

可以证明，在数据规模极大时，再分配的均摊复杂度为 $O(1)$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. deque.h

```
1 // #define _PRIVATE_DEBUG
2 #ifndef PRVLIBCPP_DEQUE_HPP
3 #define PRVLIBCPP_DEQUE_HPP
4
5 #include <map>
6 #include <vector>
7
8 #ifdef _PRIVATE_DEBUG
9 #include <iostream>
10#endif
11
12 namespace myDS
13 {
14     template<typename VALUE_TYPE>
15     class deque{
16     protected:
17
18     private:
19         using coordinate = std::pair<std::int32_t, std::int32_t>;
20
21         // < L : 0 , R : 1 >
22         std::vector<std::vector<VALUE_TYPE>> _indexes;
23
24         std::int32_t _size = 0;
25         std::int32_t _L = 1;
26         std::int32_t _R = -1;
27
28         VALUE_TYPE & get(coordinate p) {
29             return _indexes[p.first][p.second];
30         }
31
32         coordinate index2coord(std::int32_t p) {
33             if(p+_L > 0) return coordinate(1,p+_L-1);
34             else return coordinate(0,-p-_L);
35         }
36
37         void _reDistribute() {
38             if(_L * _R <= 0) return;
39             if(abs(_L - _R) + 1 < std::min(abs(_L), abs(_R))) {
40                 if(_L > 0) { // < --- 0 : 0 --- L -- R --- >
41                     std::vector<VALUE_TYPE> N;
42                     for(int i = _L-1; i <= _R; i++)
43                         N.push_back(_indexes[1][i]);
44                     _indexes[1] = N;
45                     _L = 1;
46                     _R = N.size() - 1;
47                 } else { // < --- L(<0) -- R(<0) --- 0 : 0 --- >
48                     std::vector<VALUE_TYPE> N;
```

暨南大学本科实验报告专用纸(附页)

```
48     N.push_back(_indexs[0][i]);
49     _indexs[0] = N;
50     _L = -N.size() + 1;
51     _R = -1;
52     }
53 } else return;
54 }
55
56 public:
57     deque(){
58         _indexs.push_back(std::vector<VALUE_TYPE>());
59         _indexs.push_back(std::vector<VALUE_TYPE>());
60     }
61
62     void push_back(VALUE_TYPE t) {
63         _R++;
64         if(_R >= 0) {
65             _indexs[1].push_back(t);
66         } else {
67             _indexs[0][_R-1] = t;
68             _reDistribute();
69         }
70     }
71
72     void push_front(VALUE_TYPE t) {
73         _L--;
74         if(_L <= 0) {
75             _indexs[0].push_back(t);
76         } else {
77             _indexs[1][_L-1] = t;
78             _reDistribute();
79         }
80     }
81
82     VALUE_TYPE pop_back() {
83         if(!this->size()) throw std::out_of_range("Pop from empty
deque");
84         VALUE_TYPE t ;
85         if(_R >= 0) {
86             t = _indexs[1].back();
87             _indexs[1].pop_back();
88             _R--;
89         } else {
90             t = _indexs[0][-_R-1];
91             _R--;
92             _reDistribute();
93         }
94         return t;
95     }
96 }
```

暨南大学本科实验报告专用纸(附页)

```
97     VALUE_TYPE pop_frount() {
98         if(!this->size()) throw std::out_of_range("Pop from empty
99         deque");
100        VALUE_TYPE t;
101        if(_L <= 0) {
102            t = _indexs[0].back();
103            _indexs[0].pop_back();
104            _L++;
105        } else {
106            t = _indexs[1][_L-1];
107            _L++;
108            _reDistribute();
109        }
110        return t;
111    }
112
113    void clear() {
114        _indexs[0].clear();
115        _indexs[1].clear();
116        _L = 1;
117        _R = -1;
118    }
119    std::int32_t size() {
120        return _R - _L + 2;
121    }
122
123 #ifdef _PRIVATE_DEBUG
124     void innerPrint() {
125         std::cout << "L : " << _L << " R : " << _R << "\n";
126         std::cout << "L : ";
127         for(auto x:_indexs[0]) std::cout << x << " ";
128         std::cout << "\n";
129         std::cout << "R : ";
130         for(auto x:_indexs[1]) std::cout << x << " ";
131         std::cout << "\n";
132     }
133 #endif
134
135 // myDS::deque<VALUE_TYPE>::_iterator begin() { }
136
137 // myDS::deque<VALUE_TYPE>::_iterator rbegin() { }
138
139 // myDS::deque<VALUE_TYPE>::_iterator end() { }
140
141 // myDS::deque<VALUE_TYPE>::_iterator rend() { }
142
143 // myDS::deque<VALUE_TYPE>::_iterator get(std::int32_t p) { }
144
145 VALUE_TYPE & operator[](std::int32_t p) {
146     return get(index2cod(p));
147 }
```

暨南大学本科实验报告专用纸(附页)

```
148     };
149 }
150 #endif
```

4.2. _PRIV_TEST.cpp

```
1 #define DS_TOBE_TEST deque
2
3 #define _PRIVATE_DEBUG
4 // #define __DETIL_DEBUG_OUTPUT
5
6 #include "Dev\04\deque.h"
7
8 #include <time.h>
9 #include <iostream>
10 #include <math.h>
11 #include <vector>
12
13 using namespace std;
14
15 using TBT = myDS::deque<int>;
16
17 void accuracyTest() { //结构正确性测试
18
19     TBT tc = TBT();
20     for(;;)
21     {
22         string op;
23         cout << ">>>";
24         cin >> op;
25         if(op == "clr") { //清空
26             tc.clear();
27         } else if(op == "q") //退出测试
28         {
29             return;
30         } else if(op == "pb")//push_back
31         {
32             int c;
33             cin >> c;
34             tc.push_back(c);
35         } else if(op == "pf")//push_frount
36         {
37             int c;
38             cin >> c;
39             tc.push_frount(c);
40         } else if(op == "ob")//pop_back
41         {
42             cout << tc.pop_back() << "\n";
43         } else if(op == "of")//pop_frount
44         {
```

暨南大学本科实验报告专用纸(附页)

```
45     cout << tc.pop_frount() << "\n";
46 } else if(op == "at")//随机访问
47 {
48     int p;
49     cin >> p;
50     cout << tc[p] << "\n";
51 } else if(op == "at")//随机访问
52 {
53     int p;
54     cin >> p;
55     cout << tc[p] << "\n";
56 } else if(op == "of")//pop_frount
57 {
58
59 } else if(op == "at")//随机访问
60 {
61     int p;
62     cin >> p;
63     cout << tc[p] << "\n";
64 // } else if(op == "delEL")//删除所有等于某值元素
65 //
66 //     int p;
67 //     cin >> p;
68 //     cout << tc.erase(p) << "\n";
69 // } else if(op == "delPS")//删除某位置上的元素
70 //
71 //     int p;
72 //     cin >> p;
73 //     cout << tc.erase(tc.get(p)) << "\n";
74 } else if(op == "iterF") //正序遍历
75 {
76     tc.innerPrint();
77     cout << "Iter with index:\n";
78     for(int i = 0;i < tc.size();i++) cout << tc[i] << " ";cout
    << "\n";
79     // cout << "Iter with begin end\n";
80     // for(auto x = tc.begin();x != tc.end();x++) cout << (*x)
    << " ";cout << "\n";
81     // cout << "Iter with AUTO&&\n";
82     // for(auto x:tc) cout << x << " ";cout << "\n";
83 } else if(op == "iterB") //倒序遍历
84 {
85     tc.innerPrint();
86     cout << "Iter with index:\n";
87     for(int i = 0;i < tc.size();i++) cout << tc[tc.size()-1-i]
    << " ";cout << "\n";
88     // cout << "Iter with begin end\n";
89     // for(auto x = tc.rbegin();x != tc.rend();x++) cout <<
    (*x) << " ";cout << "\n";
```

暨南大学本科实验报告专用纸(附页)

```
90         // cout << "Iter with AUTO&&\n"; ."\n";
91     } else if(op == "mv")//单点修改
92     {
93         int p;
94         cin >> p;
95         int tr;
96         cin >> tr;
97         tc[p] = tr;
98     } else if(op == "")
99     {
100
101    } else {
102        op.clear();
103    }
104}
105}
106
107
108 void memLeakTest1() {//内存泄漏测试
109     TBT tc = TBT();
110     for(;;){
111         tc.push_back(1);
112         tc.push_back(1);
113         tc.push_back(1);
114         tc.push_back(1);
115         tc.clear();
116     }
117 }
118
119 void memLeakTest2() {//内存泄漏测试
120     TBT tc = TBT();
121     for(;;){
122         tc.push_back(1);
123         tc.pop_frount();
124     }
125 }
126
127 void speedTest()
128 {
129     TBT tc = TBT();
130     int begin = clock();
131     int N = 1e8;
132     for(int i = 0;i < sqrt(N/2);i++)
133     {
134         for(int j = 0;j < sqrt(N/2);j++)
135         {
136             tc.push_back(i);
137         }
138         for(int j = 0;j < sqrt(N/2);j++)
139         {
140             tc.pop_frount();
141         }
142     }
143 }
```

暨南大学本科实验报告专用纸(附页)

```
141     }
142 }
143 cout << "myDS::deque push_back then pop_frount sqrt(50000000)
144 elements for sqrt(5000000) times cost:" << clock() - begin << "ms\n";
145 std::vector<int> tmp;
146 begin = clock();
147 for(int i = 0;i < N;i++)
148 {
149     tmp.push_back(i);
150 }
151 cout << "std::vector push_back 10000000 elements cost:" << clock() -
152 begin << "ms\n";
153 system("pause");
154 }
155
156 signed main()
157 {
158     // accuracyTest();
159     // memLeakTest1();
160     // memLeakTest2();
161     speedTest();
162 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
pb 2
pb 3
pb 4
pf 1
pf 0
iterF

pb 5
pb 6
of
of
of
iterF
of
iterF
of
```

暨南大学本科实验报告专用纸(附页)

```
iterF

>>>pb 2
>>>pb 3
>>>pb 4
>>>pf 1
>>>pf 0
>>>iterF
L : -1 R : 2
L : 1 0
R : 2 3 4
Iter with index:
0 1 2 3 4
>>>
pb 5
>>>pb 6
>>>of
0
>>>of
1
>>>of
2
>>>iterF
L : 2 R : 4
L :
R : 2 3 4 5 6
Iter with index:
3 4 5 6
>>>of
3
>>>iterF
L : 1 R : 2
L :
R : 4 5 6
Iter with index:
4 5 6
>>>of
4
>>>iterF
L : 1 R : 1
L :
R : 5 6
Iter with index:
5 6
```

暨南大学本科实验报告专用纸(附页)

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

运行 `_PRIV_TEST.cpp` 中的内存测试模块与单向插入测试模块，在保持 CPU 高占用率运行一段时间后内存变化符合预期，可以认为代码内存安全性良好。

名称	状态	25% CPU	45% 内存
<code>_PRIV_TEST.exe</code>		20.7%	0.6 MB

运行 `_PRIV_TEST.cpp` 中的性能测试模块，结果为

```
myDS::deque push_back then pop_frount sqrt(5000000) elements for
sqrt(5000000) times cost:3964ms
std::vector push_back 10000000 elements cost:1528ms
```

可以认为在每轮中单向插入的复杂度符合预期。

暨南大学本科实验报告专用纸(附页)

基于 `vector` 实现 `stack`

课程名称 数据结构 成绩评定 _____

实验项目名称 基于 `vector` 实现 `stack` 指导老师 干晓聪

实验项目编号 05 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现 `stack`

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

利用 `vector` 维护动态增长的数据，并提供栈操作的 `pop` 和 `push` 函数

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. stack.h

```
1 // #define _PRIVATE_DEBUG
2 #ifndef INTERFACE_STACK_HPP
3 #define INTERFACE_STACK_HPP
4
5 #ifdef _PRIVATE_DEBUG
6 #include <iostream>
7 #endif
8
9 namespace myDS
10 {
11     template<typename VALUE_TYPE>
12     class stack{
13     protected:
14     private:
15         vector<VALUE_TYPE> _data;
16     public:
17         stack(){ }
18
19         void push(VALUE_TYPE t) {
20             _data.push_back(t);
21         }
22
23         VALUE_TYPE pop() {
24             VALUE_TYPE t = _data.back();
25             _data.pop_back();
26             return t;
27         }
28
29         VALUE_TYPE top() {
30             return _data.back();
31         }
32
33         bool empty() {
34             return _data.empty();
35         }
36
37         int size() {
38             return _data.size();
39         }
40
41         void clear() {
42             _data.clear();
43         }
44
45         ~stack() { }
46     };
47 }
48#endif
```

暨南大学本科实验报告专用纸(附页)

树上 dfs (基础信息)

课程名称 数据结构 成绩评定 _____

实验项目名称 树上 dfs (基础信息) 指导老师 干晓聪

实验项目编号 06 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

在树上进行 dfs

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

统计树每个节点的子树大小、节点深度与子树节点权值和。

对于节点 i , 其子节点列表 S_i , 父节点 p_i , 有状态转移:

$$\text{subTreeSize}_n = \sum_{\{\text{all } i \in S_n\}} \text{subTreeSize}_i \quad (1)$$

$$\text{depth}_n = \text{depth}_{p_n} + 1 \quad (2)$$

$$\text{subTreeSum}_n = \sum_{\{\text{all } i \in S_n\}} \text{subTreeSum}_i + \text{Wei}_n \quad (3)$$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. `dfs.cpp`

```
1 #include <iostream>
2 #include <vector>
3 #include <stdlib.h>
4 #include <map>
5 #include <algorithm>
6
7 using namespace std;
8 using pii = pair<int,int>;
9
10 #define int long long
11 #define pb push_back
12 #define F first
13 #define S second
14 #define all(x) x.begin(),x.end()
15 #define loop(i,n) for(int i = 0; i < n; i++)
16
17 const int mod = 1e9 + 7;
18 const int INF = 1e18;
19
20 // 子树大小 节点深度 子树节点值和
21 void solve()
22 {
23     int n;
24     cin >> n;
25     vector<vector<int>> cnj(n+1);
26     vector<int> wei(n+1);
27     vector<int> subTreeSize(n+1), depth(n+1), subTreeSum(n+1);
28     loop(i,n-1)
29     {
30         int u,v;
31         cin >> u >> v;
32         cnj[u].pb(v);
33         cnj[v].pb(u);
34     }
35     loop(i,n) cin >> wei[i+1];
36     auto dfs = [&](auto self, int p, int f) -> pii {
37         int size = 1;
38         int sum = wei[p];
39         depth[p] = depth[f] + 1;
40         for(auto &x : cnj[p])
41         {
42             if(x == f) continue;
43             auto [subSize,subSum] = self(self,x,p);
44             size += subSize;
45             sum += subSum;
46         }
47         subTreeSize[p] = size;
48     };
49 }
```

暨南大学本科实验报告专用纸(附页)

```
48     subTreeSum[p] = sum;
49     return {size,sum};
50 }
51 dfs(dfs,1,0);
52 cout << "subTreeSize \n";
53 loop(i,n) cout << subTreeSize[i+1] << " ";
54 cout << "\n";
55 cout << "depth \n";
56 loop(i,n) cout << depth[i+1] << " ";
57 cout << "\n";
58 cout << "subTreeSum \n";
59 loop(i,n) cout << subTreeSum[i+1] << " ";
60 cout << "\n";
61 }
62
63 signed main()
64 {
65 // std::ios::sync_with_stdio(false);
66 // std::cin.tie(nullptr);
67 // std::cout.tie(nullptr);
68
69 int T = 1;
70 cin >> T;
71 while(T--)
72     solve();
73
74 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
2
8
1 2
1 3
1 4
3 5
3 6
6 7
6 8
9 4 3 1 15 7 3 7
subTreeSize
8 1 5 1 1 3 1 1
depth
1 2 2 2 3 3 4 4
```

暨南大学本科实验报告专用纸(附页)

```
subTreeSum  
49 4 35 1 15 17 3 7  
6  
1 2  
2 3  
3 4  
4 5  
5 6  
1 1 4 5 1 4  
subTreeSize  
6 5 4 3 2 1  
depth  
1 2 3 4 5 6  
subTreeSum  
16 15 14 10 5 4
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

图上 bfs (最短路)

课程名称 数据结构 成绩评定 _____

实验项目名称 图上 bfs (最短路) 指导老师 干晓聪

实验项目编号 07 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024 年 6 月 13 日上午 ~ 2024 年 7 月 13 日中午

1. 实验目的

利用优先队列优化的 bfs 实现 Dijkstra 算法求最短路

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

已确定最短路的节点集合 S , 未确定的节点集合 T

对 T 中最小的节点 T 进行 BFS, 松弛其所有子节点后, 将 T 加入 S 中, 直到算法收敛。

对于松弛操作 $S \rightarrow u \rightarrow v$ 有 $\text{dis}(v) = \min(\text{dis}(v), \text{dis}(u) + w(u, v))$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. bfs.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <stdlib.h>
4 #include <map>
5 #include <set>
6 #include <algorithm>
7 #include <queue>
8
9 using namespace std;
10 using pii = pair<int, int>;
11
12 #define int long long
13 #define pb push_back
14 #define F first
15 #define S second
16 #define all(x) x.begin(), x.end()
17 #define loop(i, n) for (int i = 0; i < n; i++)
18
19 const int mod = 1e9 + 7;
20 const int INF = 1e18;
21
22 // 优先队列队列 BFS 求最短路
23 //
24 void solve()
25 {
26     int n, m;
27     cin >> n >> m;
28     vector<vector<pii>> cnj(n + 1);
29     vector<int> rcnj(n+1);
30     loop(i, m)
31     {
32         int u, v, w;
33         cin >> u >> v >> w;
34         cnj[u].pb({v, w});
35     }
36     priority_queue<pii, vector<pii>, greater<pii>> dfsOrder;
37     set<int> unReached;
38     loop(i, n) unReached.insert(i + 1);
39     vector<int> dis(n+1, INF);
40     vector<int> locked(n+1, 0);
41     int ori, tar;
42     cin >> ori >> tar;
43     dis[ori] = 0;
44     dfsOrder.push({0, ori});
45
46     auto release = [&](int _n) -> void
47     {
```

暨南大学本科实验报告专用纸(附页)

```
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
```

```
    for (auto x : cnj[_n])
    {
        // dis[x.first] = min(dis[x.first], dis[_n] + x.second);
        // dfsOrder.push({dis[x.first], x.first});
        if(locked[x.first]) continue;
        if(dis[x.first] > dis[_n] + x.second) {
            dis[x.first] = dis[_n] + x.second;
            rcnj[x.first] = _n;
            } dfsOrder.push({dis[x.first], x.first});
    }
};

while (unReached.size())
{
    auto u = dfsOrder.top();
    dfsOrder.pop();
    release(u.second);
    unReached.erase(u.second);
    locked[u.second] = 1;
}

vector<int> path;

int ptt = tar;
while(ptt != ori) {
    path.pb(ptt);
    ptt = rcnj[ptt];
} path.pb(ori);

loop(i,path.size()) cout << path[path.size()-1-i] << ( i != path.size()-1?" -> ":"\\n");
}

signed main()
{
    // std::ios::sync_with_stdio(false);
    // std::cin.tie(nullptr);
    // std::cout.tie(nullptr);

    int T = 1;
    cin >> T;
    while (T--)
        solve();
    system("pause");
    return 0;
}
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

暨南大学本科实验报告专用纸(附页)

```
1
8 13
1 2 2
1 3 14
1 4 15
2 3 2
3 4 3
2 5 15
3 5 15
4 6 1
6 5 3
5 6 3
5 7 2
5 8 14
8 7 3
1 7
1 -> 2 -> 3 -> 4 -> 6 -> 5 -> 7
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

R-BTree 的基本实现

课程名称 数据结构 成绩评定 _____

实验项目名称 R-BTree 的基本实现 指导老师 干晓聪

实验项目编号 08 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现 RB-tree 的基本结构

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

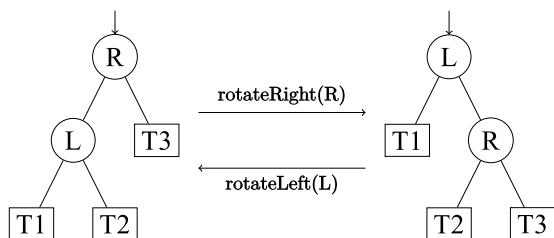
IDE: Visual Studio Code

3. 程序原理

对于一个二叉搜索树，标记所有叶节点为 `NIL`，并在路径上标记红黑节点，使得：

1. `NIL` 节点为黑色
2. 红色节点的子节点为黑色
3. 从根节点到 `NIL` 节点路径上的黑色节点数量相同

定义旋转



暨南大学本科实验报告专用纸(附页)

对于每次插入与删除，需要基于红黑节点性质进行平衡维护。具体实现见代码。

容易证明，满足红黑性质的红黑树，为近似平衡二叉搜索树。

可得插入复杂度为 $\Theta(\log_2 n)$ ，删除复杂度为 $\Theta(\log_2 n)$ ，随机访问复杂度为 $\Theta(\log_2 n)$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. memDeleteTest.cpp

```
1 #include <iostream>
2 #include <new>
3 #include <stdlib.h>
4 using namespace std;
5
6 class testClass{
7 public:
8     int a = 0;
9     testClass(){a=1;};
10    ~testClass(){cout << "Destroy TestClass\n";};
11 };
12 int main()
13 {
14     testClass * arr = new testClass[10];
15     cout << "Finish Alloc\n";
16     for(int i = 0;i < 10;i++)
17         arr[i].~testClass();
18     if(arr)
19         //delete[] arr;
20         ::operator delete[](arr);
21     else cout << "nullPtr\n";
22     cout << "Finish Delete\n";
23     return 0;
24 }
```

4.2. RB_Tree.h

```
1 #ifndef RBTREE_MAP_HPP
2 #define RBTREE_MAP_HPP
3
4 #ifdef __PRIVATE_DEBUG
5 #include <iostream>
6 #endif
7
8 #include <vector>
9 #include <stdlib.h>
10 #include "Dev\02\myVector.h"
11
12 using std::vector;
13
14
15 namespace myDS
16 {
17     template <typename VALUE_TYPE>
18     class RBtree{
19     private:
```

暨南大学本科实验报告专用纸(附页)

```
20 // using int size_t;
21 enum COLOR {RED,BLACK};
22
23 protected:
24 //节点类
25 class Node{
26 public:
27     VALUE_TYPE value;
28     COLOR color;
29     Node *leftSubTree, //左子树根节点指针
30             *rightSubTree, //右子树根节点指针
31             *parent; //父节点指针
32
33     explicit Node() :
34         value(VALUE_TYPE()),
35         color(COLOR::RED),
36         leftSubTree(nullptr),
37         rightSubTree(nullptr),
38         parent(nullptr) { };
39
40     //获取父节点指针
41     inline Node * getParent() {
42         return parent;
43     }
44
45     //获取祖父节点指针
46     inline Node * getGrandParent() {
47         if(parent == nullptr) return nullptr;
48         else return parent->parent;
49     }
50
51     //获取叔叔节点指针
52     inline Node * getUncle() {
53         Node* __gp = this->getGrandParent();
54         if(__gp == nullptr) return nullptr;
55         else if(parent == __gp->rightSubTree) return __gp-
>leftSubTree;
56         else return __gp->rightSubTree;
57     }
58
59     //获取兄弟节点指针
60     inline Node * getSibling(){
61         if(parent == nullptr) return nullptr;
62         else if(parent->leftSubTree == this) return parent-
>rightSubTree;
63         else return parent->leftSubTree;
64     }
65
66
67
68     class iterator{
```

暨南大学本科实验报告专用纸(附页)

```
69     friend RBtree;
70 protected:
71     Node * ptr;
72     Node * NIL;
73
74     void loop2Begin() {
75         if(ptr == NIL){ptr = nullptr;return;}
76         while(ptr->leftSubTree != NIL) ptr = ptr->leftSubTree;
77     }
78
79     void loop2End() {
80         if(ptr == NIL){ptr = nullptr;return;}
81         if(ptr->parent == nullptr){ ptr = nullptr;return;}
82         while(ptr->parent->leftSubTree != ptr) {
83             ptr = ptr->parent;
84             if(ptr->parent == nullptr){ ptr = nullptr;return;}
85         }
86         ptr = ptr->parent;
87     }
88
89     void getNextNode() {
90         if(ptr->rightSubTree != NIL){
91             ptr = ptr->rightSubTree;
92             loop2Begin();
93         } else {
94             loop2End();
95         }
96     }
97
98 public:
99     iterator(Node * _ptr,Node * _NIL) {
100        ptr = _ptr;
101        NIL = _NIL;
102    }
103
104     const VALUE_TYPE & operator*()
105     {
106         return ptr->value;
107     }
108
109     VALUE_TYPE *operator->() //?
110     {
111         return ptr;
112     }
113
114     myDS::RBtree<VALUE_TYPE>::iterator operator++() {
115         auto old = *this;
116         getNextNode();
117         return old;
118     }
119
120     myDS::RBtree<VALUE_TYPE>::iterator operator++(int) {
```

暨南大学本科实验报告专用纸(附页)

```
121     getNextNode();
122     return (*this);
123 }
124
125     bool operator==( myDS::RBtree<VALUE_TYPE>::iterator _b) {
126         return ptr == _b.ptr;
127     }
128
129     bool operator!=( myDS::RBtree<VALUE_TYPE>::iterator _b) {
130         return ptr != _b.ptr;
131     }
132 }
133
134 public:
135     //树结构
136     Node *root, *NIL;
137
138     RBtree() {
139         NIL = new Node();
140         NIL->color = COLOR::BLACK;
141         root = nullptr;
142     }
143
144     ~RBtree(){
145         auto DeleteSubTree = [&](auto self,Node *p) -> void{
146             if(p == nullptr || p == NIL) return;
147             self(self,p->leftSubTree);
148             self(self,p->rightSubTree);
149             delete p;
150             return;
151         };
152         if(!(root == nullptr)) DeleteSubTree(DeleteSubTree,root);
153         delete NIL;
154     }
155
156     void insert(VALUE_TYPE data) {
157         if(root == nullptr) {
158             root = new Node();
159             root->color = COLOR::BLACK;
160             root->leftSubTree = NIL;
161             root->rightSubTree = NIL;
162             root->value = data;
163         } else {
164             if(this->locate(data,root)) return;
165             subInsert(root,data);
166         }
167     }
168
169     VALUE_TYPE find(VALUE_TYPE tar) {
170         if(locate(tar,root) != nullptr) return locate(tar,root)-
>value;
```

暨南大学本科实验报告专用纸(附页)

```
171         else return -1;
172     }
173
174     bool erase(VALUE_TYPE data) {
175         return subDelete(root,data);
176     }
177
178     myDS::RBtree<VALUE_TYPE>::iterator begin(){
179         auto rt = iterator(root,NIL);
180         rt.loop2Begin();
181         return rt;
182     }
183
184     myDS::RBtree<VALUE_TYPE>::iterator end(){
185         return iterator(nullptr,NIL);
186     }
187
188 #ifdef __PRIVATE_DEBUGE
189     void printDfsOrder()
190     {
191         auto dfs = [&](auto self,Node * p ) -> void {
192             if(p == nullptr){ std::cout << "ED\n";return;}
193             if(p->leftSubTree == nullptr && p->rightSubTree ==
194             nullptr) {std::cout << "[NIL] \n";return;}
195             std::cout << "["<< p->value << " : " << (p->color ==
196             COLOR::BLACK ?"BLACK": "RED") << "] ";
197             self(self,p->leftSubTree);
198             self(self,p->rightSubTree);
199             return;
200         };
201         dfs(dfs,root);
202     }
203
204     vector<int> printList;
205     void printIterOrder()
206     {
207         auto dfs = [&](auto self,Node * p) -> void{
208             if(p->leftSubTree == nullptr && p->rightSubTree ==
209             nullptr) {std::cout << "[NIL] \n";return;}
210             self(self,p->leftSubTree);
211             std::cout << "["<< p->value << " : " << (p->color ==
212             COLOR::BLACK ?"BLACK": "RED") << "] ";
213             self(self,p->rightSubTree);
214         };
215         dfs(dfs,root);
216     }
217 #endif
218     private:
219     Node * locate(VALUE_TYPE t,Node * p) {
220         if(p == NIL) return nullptr;
221         else if(p->value == t) return p;
```

暨南大学本科实验报告专用纸(附页)

```
218     else if(p->value > t) return locate(t,p->leftSubTree);
219     else return locate(t,p->rightSubTree);
220 }
221
222 //右旋某个节点
223 void rotateRight(Node *p)
224 {
225     Node *_gp = p->getGrandParent();
226     Node *_pa = p->getParent();
227     Node *_rotY = p->rightSubTree;
228     _pa->leftSubTree = _rotY;
229     if(_rotY != NIL) _rotY->parent = _pa;
230     p->rightSubTree = _pa;
231     _pa->parent = p;
232
233     if(root == _pa) root = p;
234     p->parent = _gp;
235     if(_gp != nullptr) if(_gp->leftSubTree == _pa) _gp-
236 >leftSubTree = p;
237         else _gp->rightSubTree = p;
238     return;
239 }
240
241 //左旋某个节点
242 void rotateLeft(Node *p)
243 {
244     if(p->parent == nullptr){
245         root = p;
246         return;
247     }
248     Node *_gp = p->getGrandParent();
249     Node *_pa = p->parent;
250     Node *_rotX = p->leftSubTree;
251
252 #ifdef __DETIL_DEBUG_OUTPUT
253     printIterOrder();
254 #endif
255     _pa->rightSubTree = _rotX;
256
257 #ifdef __DETIL_DEBUG_OUTPUT
258     printIterOrder();
259 #endif
260
261     if(_rotX != NIL)
262         _rotX->parent = _pa;
263
264 #ifdef __DETIL_DEBUG_OUTPUT
265     printIterOrder();
266 #endif
267     p->leftSubTree = _pa;
268
269 #ifdef __DETIL_DEBUG_OUTPUT
```

暨南大学本科实验报告专用纸(附页)

```
268         printIterOrder();
269 #endif
270         _pa->parent = p;
271 #ifdef __DETIL_DEBUG_OUTPUT
272         printIterOrder();
273 #endif
274         if(root == _pa)
275             root = p;
276         p->parent = _gp;
277
278 #ifdef __DETIL_DEBUG_OUTPUT
279         printIterOrder();
280 #endif
281         if(_gp != nullptr){
282             if(_gp->leftSubTree == _pa)
283                 _gp->leftSubTree = p;
284             else
285                 _gp->rightSubTree = p; //?!
286         }
287 #ifdef __DETIL_DEBUG_OUTPUT
288         printIterOrder();
289 #endif
290     }
291
292     //插入节点递归部分
293     void subInsert(Node *p, VALUE_TYPE data)
294     {
295         if(p->value >= data){ //1 2
296             if(p->leftSubTree != NIL) //3
297                 subInsert(p->leftSubTree, data);
298             else {
299                 Node *tmp = new Node(); //3
300                 tmp->value = data;
301                 tmp->leftSubTree = tmp->rightSubTree = NIL;
302                 tmp->parent = p;
303                 p->leftSubTree = tmp;
304                 resetStatus_forInsert(tmp);
305             }
306         } else {
307             if(p->rightSubTree != NIL) //1 2
308                 subInsert(p->rightSubTree, data);
309             else {
310                 Node *tmp = new Node();
311                 tmp->value = data;
312                 tmp->leftSubTree = tmp->rightSubTree = NIL;
313                 tmp->parent = p;
314                 p->rightSubTree = tmp;
315                 resetStatus_forInsert(tmp);
316             }
317         }
318     }
```

暨南大学本科实验报告专用纸(附页)

```
319 //插入后的平衡维护
320
321 void resetStatus_forInsert(Node *p) {
322     //case 1:
323     if(p->parent == nullptr){
324         root = p;
325         p->color = COLOR::BLACK;
326         return;
327     }
328     //case 2-6:
329     if(p->parent->color == COLOR::RED){
330         //case 2: pass
331         if(p->getUncle()->color == COLOR::RED) {
332             p->parent->color = p->getUncle()->color =
333             COLOR::BLACK;
334             p->getGrandParent()->color = COLOR::RED;
335             resetStatus_forInsert(p->getGrandParent());
336         } else {
337             if(p->parent->rightSubTree == p && p-
338             >getGrandParent()->leftSubTree == p->parent) {
339                 //case 3:
340                 rotateLeft(p);
341                 p->color = COLOR::BLACK;
342                 p->parent->color = COLOR::RED;
343                 rotateRight(p);
344             } else if(p->parent->leftSubTree == p && p-
345             >getGrandParent()->rightSubTree == p->parent) { //this
346                 //case 4:
347                 rotateRight(p);
348                 p->color = COLOR::BLACK;
349                 p->parent->color = COLOR::RED;
350                 rotateLeft(p);
351             } else if(p->parent->leftSubTree == p && p-
352             >getGrandParent()->leftSubTree == p->parent) {
353                 //case 5:
354                 p->parent->color = COLOR::BLACK;
355                 p->getGrandParent()->color = COLOR::RED;
356                 rotateRight(p->parent);
357             } else if(p->parent->rightSubTree == p && p-
358             >getGrandParent()->rightSubTree == p->parent) {
359                 //case 6: BUG HERE
360                 p->parent->color = COLOR::BLACK;
361                 p->getGrandParent()->color = COLOR::RED;
362                 rotateLeft(p->parent);
363             }
364         }
365     }
366 }
367
368 //删除时的递归部分
```

暨南大学本科实验报告专用纸(附页)

```
364     bool subDelete(Node *p, VALUE_TYPE data){  
365  
366         //获取最接近叶节点的儿子  
367         auto getLowwestChild = [&](auto self,Node *p) -> Node*{  
368             if(p->leftSubTree == NIL) return p;  
369             return self(self,p->leftSubTree);  
370         };  
371  
372         if(p->value > data){  
373             if(p->leftSubTree == NIL){  
374                 return false;  
375             }  
376             return subDelete(p->leftSubTree, data);  
377         } else if(p->value < data){  
378             if(p->rightSubTree == NIL){  
379                 return false;  
380             }  
381             return subDelete(p->rightSubTree, data);  
382         } else if(p->value == data){  
383             if(p->rightSubTree == NIL){  
384                 deleteChild(p);  
385                 return true;  
386             }  
387             Node *smallChild = getLowwestChild(getLowwestChild,p->rightSubTree);  
388             std::swap(p->value, smallChild->value);  
389             deleteChild(smallChild);  
390  
391             return true;  
392         }else{  
393             return false;  
394         }  
395     }  
396  
397     //    //删除入口  
398     //    bool deleteChild(Node *p, int data){  
399     //        if(p->value > data){  
400     //            if(p->leftSubTree == NIL){  
401     //                return false;  
402     //            }  
403     //            return deleteChild(p->leftSubTree, data);  
404     //        } else if(p->value < data){  
405     //            if(p->rightSubTree == NIL){  
406     //                return false;  
407     //            }  
408     //            return deleteChild(p->rightSubTree, data);  
409     //        } else if(p->value == data){  
410     //            if(p->rightSubTree == NIL){  
411     //                delete_one_child (p);  
412     //                return true;  
413     //            }  
414 }
```

暨南大学本科实验报告专用纸(附页)

```
414     //           Node *smallest = getSmallestChild(p->rightTree);
415     //           swap(p->value, smallest->value);
416     //           delete_one_child (smallest);
417
418     //           return true;
419     // }else{
420     //           return false;
421     // }
422 }
423
424 //删除处理: 删除某个儿子
425 void deleteChild(Node *p){
426     Node *child = p->leftSubTree == NIL ? p->rightSubTree : p-
>leftSubTree;
427     if(p->parent == nullptr && p->leftSubTree == NIL && p-
>rightSubTree == NIL){
428         p = nullptr;
429         root = p;
430         return;
431     }
432     if(p->parent == nullptr){
433         delete p;
434         child->parent = nullptr;
435         root = child;
436         root->color = COLOR::BLACK;
437         return;
438     }
439     if(p->parent->leftSubTree == p) p->parent->leftSubTree =
child;
440     else p->parent->rightSubTree = child;
441
442     child->parent = p->parent;
443     if(p->color == COLOR::BLACK){
444         if(child->color == COLOR::RED){
445             child->color = COLOR::BLACK;
446         } else
447             resetStatus_forDelete(child);
448     }
449
450     delete p;
451 }
452
453 //删除后的平衡维护
454 void resetStatus_forDelete(Node *p){
455     if(p->parent == nullptr){
456         //case 0-0:
457         p->color = COLOR::BLACK;
458         return;
459     }
460     if(p->getSibling()->color == COLOR::RED) {
461         //case 0-1:
```

暨南大学本科实验报告专用纸(附页)

```
462     p->parent->color = COLOR::RED;
463     p->getSibling()->color = COLOR::BLACK;
464     if(p == p->parent->leftSubTree) rotateLeft(p->parent);
465     else rotateRight(p->parent);
466 }
467 if( p->parent->color == COLOR::BLACK &&
468     p->getSibling()->color == COLOR::BLACK &&
469     p->getSibling()->leftSubTree->color == COLOR::BLACK &&
470     p->getSibling()->rightSubTree->color == COLOR::BLACK) {
471     //case 1-1:
472     p->getSibling()->color = COLOR::RED;
473     resetStatus_forDelete(p->parent);
474 } else if(p->parent->color == COLOR::RED && p->getSibling()-
475 >color == COLOR::BLACK&& p->getSibling()->leftSubTree->color ==
476 COLOR::BLACK && p->getSibling()->rightSubTree->color == COLOR::BLACK) {
477     //case 1-2:
478     p->getSibling()->color = COLOR::RED;
479     p->parent->color = COLOR::BLACK;
480 } else {
481     if(p->getSibling()->color == COLOR::BLACK) {
482         if(p == p->parent->leftSubTree && p->getSibling()-
483 >leftSubTree->color == COLOR::RED && p->getSibling()->rightSubTree-
484 >color == COLOR::BLACK) {
485             //case 1-3:
486             p->getSibling()->color = COLOR::RED;
487             p->getSibling()->leftSubTree->color =
488 COLOR::BLACK;
489             rotateRight(p->getSibling()->leftSubTree);
490 } else if(p == p->parent->rightSubTree && p-
491 >getSibling()->leftSubTree->color == COLOR::BLACK && p->getSibling()-
492 >rightSubTree->color == COLOR::RED) {
493             //case 1-4:
494             p->getSibling()->color = COLOR::RED;
495             p->getSibling()->rightSubTree->color =
496 COLOR::BLACK;
497             rotateLeft(p->getSibling()->rightSubTree);
498         }
499     }
500     p->getSibling()->color = p->parent->color;
501     p->parent->color = COLOR::BLACK;
502     //case 1-5:
503     if(p == p->parent->leftSubTree){
504         //case 0-3
505         p->getSibling()->rightSubTree->color = COLOR::BLACK;
506         rotateLeft(p->getSibling());
507     } else {
508         //case 0-4
509         p->getSibling()->leftSubTree->color = COLOR::BLACK;
510         rotateRight(p->getSibling());
511     }
512 }
```

暨南大学本科实验报告专用纸(附页)

```
503         }
504     }
505 }
506
507 }
508 };
509 } // namespace myDS
510 #endif
```

4.3. _PRIV_TEST.cpp

```
1 // #include <d:\Desktop\Document\Coding\C++\ProjectC\myDS\myVector.h>
2 // #include "myVector.h"
3 #define __PRIVATE_DEBUG
4 // #define __DETIL_DEBUG_OUTPUT
5 #include "Dev\08\RB_Tree.h"
6 #include "Dev\08\eg2.h"
7 #include <iostream>
8 #include <vector>
9
10
11 using namespace myDS;
12
13 int main()
14 {
15     // testingVector tc;
16     int i = 0;
17     // bst rbt;
18     RBtree<int> rbt;
19     while (1)
20     {
21         // i++;
22         char q;
23         std::cin >> q;
24         switch (q)
25         {
26             case 'i':
27             {
28                 int t;
29                 std::cin >> t;
30                 rbt.insert(t);
31             }
32             break;
33             case 'p':
34                 std::cout << "====DFS Order====\n";
35                 rbt.printDfsOrder();
36                 std::cout << "====Iter Order====\n";
37                 rbt.printIterOrder();
38                 // std::cout << "====Use Itera====\n";
39                 // for(auto x:rbt) std::cout << x << " ";
```

暨南大学本科实验报告专用纸(附页)

```
40 // cout << "\n";
41 std::cout << "====Use Bg Ed====\n";
42 for(auto x = rbt.begin();x != rbt.end();x++)
43 {
44     auto & y = *x;
45     std::cout << y << " ";
46 }cout << "\n";
47
48 std::cout << "\n";
49 // rbt.inorder();
50 break;
51 case 'd':
52 {
53     int t;
54     std::cin >> t;
55     // rbt.delete_value(t);
56     rbt.erase(t);
57 }
58
59
60 }
```

5. 测试数据与运行结果

运行上述 _PRIV_TEST.cpp 测试代码中的正确性测试模块，得到以下内容：

```
i 1
i 2
i 3
i 4
i 5
i 6

p
====DFS Order====
[2 : BLACK] [1 : BLACK] [NIL]
[NIL]
[4 : RED] [3 : BLACK] [NIL]
[NIL]
[5 : BLACK] [NIL]
[6 : RED] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : BLACK] [NIL]
[2 : BLACK] [NIL]
[3 : BLACK] [NIL]
```

暨南大学本科实验报告专用纸(附页)

```
[4 : RED] [NIL]
[5 : BLACK] [NIL]
[6 : RED] [NIL]
d 2
p
====DFS Order====
[3 : BLACK] [1 : BLACK] [NIL]
[NIL]
[5 : RED] [4 : BLACK] [NIL]
[NIL]
[6 : BLACK] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : BLACK] [NIL]
[3 : BLACK] [NIL]
[4 : BLACK] [NIL]
[5 : RED] [NIL]
[6 : BLACK] [NIL]
d 5
p
====DFS Order====
[3 : BLACK] [1 : BLACK] [NIL]
[NIL]
[6 : BLACK] [4 : RED] [NIL]
[NIL]
[NIL]
====Iter Order====
[NIL]
[1 : BLACK] [NIL]
[3 : BLACK] [NIL]
[4 : RED] [NIL]
[6 : BLACK] [NIL]
d 3
p
====DFS Order====
[4 : BLACK] [1 : BLACK] [NIL]
[NIL]
[6 : BLACK] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : BLACK] [NIL]
[4 : BLACK] [NIL]
```

暨南大学本科实验报告专用纸(附页)

```
[6 : BLACK] [NIL]
d 2
p
====DFS Order====
[4 : BLACK] [1 : BLACK] [NIL]
[NIL]
[6 : BLACK] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : BLACK] [NIL]
[4 : BLACK] [NIL]
[6 : BLACK] [NIL]
i 2
p
====DFS Order====
[4 : BLACK] [1 : BLACK] [NIL]
[2 : RED] [NIL]
[NIL]
[6 : BLACK] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : BLACK] [NIL]
[2 : RED] [NIL]
[4 : BLACK] [NIL]
[6 : BLACK] [NIL]

i 1
i 2
i 3
i 5
i 4
p
====DFS Order====
[2 : BLACK] [1 : BLACK] [NIL]
[NIL]
[4 : BLACK] [3 : RED] [NIL]
[NIL]
[5 : RED] [NIL]
[NIL]
====Iter Order====
[NIL]
```

暨南大学本科实验报告专用纸(附页)

```
[1 : BLACK] [NIL]  
[2 : BLACK] [NIL]  
[3 : RED] [NIL]  
[4 : BLACK] [NIL]  
[5 : RED] [NIL]  
====Use Bg Ed====  
1 2 3 4 5
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

运行 `_PRIV_TEST.cpp` 中的内存测试模块，在保持 CPU 高占用率运行一段时间后内存变化符合预期，可以认为代码内存安全性良好。

名称	状态	25% CPU	45% 内存
<code>_PRIV_TEST.exe</code>		20.7%	0.6 MB

暨南大学本科实验报告专用纸(附页)

基于 R-BTree 实现 set

课程名称 数据结构 成绩评定 _____

实验项目名称 基于 R-BTree 实现 set 指导老师 干晓聪

实验项目编号 10 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024 年 6 月 13 日上午 ~ 2024 年 7 月 13 日中午

1. 实验目的

基于已有的 RB_Tree 结构实现 set

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

set 要求元素不重合，因此选择直接使用红黑树维护，提供 insert 和 erase 操作。

由红黑树性质易得，插入复杂度 $\mathcal{O}(\log_2 n)$ ，删除复杂度 $\mathcal{O}(\log_2 n)$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. mySet.h

```
1 #ifndef PRVLIBCPP_SET_HPP
2 #define PRVLIBCPP_SET_HPP
3
4 #include <Dev\08\RB_Tree.h>
5
6 namespace myDS
7 {
8     template<typename VALUE_TYPE>
9     class set{
10     private:
11         myDS::RBtree<VALUE_TYPE> dataST;
12
13     public:
14         set(){ }
15
16         ~set(){ }
17
18         void insert(VALUE_TYPE _d) {dataST.insert(_d);}
19         bool erase(VALUE_TYPE _d) {return dataST.erase(_d);}
20         bool find(VALUE_TYPE _d) {return dataST.find(_d);}
21         auto begin() {return dataST.begin();}
22         auto end() {return dataST.end();}
23     };
24 } // namespace myDS
25 #endif
```

4.2. _PRIV_TEST.cpp

```
1 #include <iostream>
2 #include <Dev\10\mySet.h>
3
4 using namespace std;
5
6 int main()
7 {
8     myDS::set<int> testST;
9     while(1)
10    {
11        string s;
12        cin >> s;
13        if(s == "i") {
14            int t;
15            cin >> t;
16            testST.insert(t);
17        } else if(s == "p") {
18            for(auto x:testST) cout << x << " ";
```

暨南大学本科实验报告专用纸(附页)

```
19         cout << "\n";
20     } else if(s == "d") {
21         int t;
22         cin >> t;
23         cout << testST.erase(t) << "\n";
24     } else if(s == "f") {
25         int t;
26         cin >> t;
27         cout << testST.find(t) << "\n";
28     }
29 }
30 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
i 1
i 2
i 3
i 6
i 5
p
1 2 3 5 6
i 6
i 5
p
1 2 3 5 6
d 3
1
d 4
0
p
1 2 5 6
f 2
1
f 3
0
f 4
0
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

基于 R-BTree 实现 map

课程名称 数据结构 成绩评定 _____

实验项目名称 基于 R-BTree 实现 map 指导老师 干晓聪

实验项目编号 11 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024 年 6 月 13 日上午 ~ 2024 年 7 月 13 日中午

1. 实验目的

基于 R-BTree 实现 map

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

map 维护一个 $S \rightarrow \mathbb{R}$ 的映射, 对于 \mathbb{R} 的规模 n 保证:

总体空间复杂度 $\mathcal{O}(\log_2 n)$, 访问时间复杂度 $\mathcal{O}(\log_2 n)$, 插入删除复杂度 $\mathcal{O}(\log_2 n)$ 。

在代码中, 每个映射体现为 `std::pair<KEY_TYPE, VALUE_TYPE>`

在 `RB_Tree` 的基础上, 在每个节点上添加数据项 `VALUE`, 并且对应的修改泛型系统。

额外实现一个基于节点推断后续节点的函数, 以提供对迭代器的支持, 具体实现见代码。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. map.hpp

```
1 #ifndef RBTREE_MAP_HPP
2 #define RBTREE_MAP_HPP
3
4 #ifdef __PRIVATE_DEBUGE
5 #include <iostream>
6 #endif
7
8 #include <vector>
9 #include <stdlib.h>
10 #include "Dev\02\myVector.h"
11
12 using std::vector;
13
14
15 namespace myDS
16 {
17     template <typename KEY,typename VALUE>
18     class map{
19     private:
20         // using int size_t;
21         enum COLOR {RED,BLACK};
22
23     protected:
24         //节点类
25         class Node{
26     public:
27             KEY key;
28             VALUE value;
29             COLOR color;
30             Node *leftSubTree, //左子树根节点指针
31                 *rightSubTree, //右子树根节点指针
32                 *parent; //父节点指针
33
34             explicit Node() : //构造函数
35                 key(KEY()),
36                 color(COLOR::RED),
37                 leftSubTree(nullptr),
38                 rightSubTree(nullptr),
39                 parent(nullptr),
40                 value(VALUE()) { };
41
42             //获取父节点指针
43             inline Node * getParent() {
44                 return parent;
45             }
46
47             //获取祖父节点指针
48 }
```

暨南大学本科实验报告专用纸(附页)

```
48     inline Node * getGrandParent() {
49         if(parent == nullptr) return nullptr;
50         else return parent->parent;
51     }
52
53     //获取叔叔节点指针
54     inline Node * getUncle() {
55         Node* __gp = this->getGrandParent();
56         if(__gp == nullptr) return nullptr;
57         else if(parent == __gp->rightSubTree) return __gp-
58             >leftSubTree;
59         else return __gp->rightSubTree;
60     }
61
62     //获取兄弟节点指针
63     inline Node * getSibling(){
64         if(parent == nullptr) return nullptr;
65         else if(parent->leftSubTree == this) return parent-
66             >rightSubTree;
67         else return parent->leftSubTree;
68     }
69
70     class iterator{
71         friend map;
72         protected:
73             Node * ptr;
74             Node * NIL;
75
76             void loop2Begin() {
77                 if(ptr == NIL){ptr = nullptr;return;}
78                 while(ptr->leftSubTree != NIL) ptr = ptr->leftSubTree;
79             }
80
81             void loop2End() {
82                 if(ptr == NIL){ptr = nullptr;return;}
83                 if(ptr->parent == nullptr){ ptr = nullptr;return;}
84                 while(ptr->parent->leftSubTree != ptr) {
85                     ptr = ptr->parent;
86                     if(ptr->parent == nullptr){ ptr = nullptr;return;}
87                 }
88                 ptr = ptr->parent;
89             }
90
91             void getNextNode() {
92                 if(ptr->rightSubTree != NIL){
93                     ptr = ptr->rightSubTree;
94                     loop2Begin();
95                 } else {
96                     loop2End();
```

暨南大学本科实验报告专用纸(附页)

```
97     }
98 }
99
100
101     public:
102         iterator(Node * _ptr,Node * _NIL) {
103             ptr = _ptr;
104             NIL = _NIL;
105         }
106
107         const std::pair<KEY,VALUE> operator*()
108     {
109         return {ptr->key,ptr->value};
110     }
111
112         KEY *operator->() //?
113     {
114         return ptr;
115     }
116
117         myDS::map<KEY,VALUE>::iterator operator++()
118     {
119         auto old = *this;
120         getNextNode();
121         return old;
122     }
123
124         myDS::map<KEY,VALUE>::iterator operator++(int)
125     {
126         getNextNode();
127         return (*this);
128     }
129
130         bool operator==( myDS::map<KEY,VALUE>::iterator _b) {
131             return ptr == _b.ptr;
132         }
133
134         bool operator!=( myDS::map<KEY,VALUE>::iterator _b) {
135             return ptr != _b.ptr;
136         }
137     };
138
139     //树结构
140     Node *root, *NIL;
141
142     public:
143         map() {
144             NIL = new Node();
145             NIL->color = COLOR::BLACK;
146             root = nullptr;
147         }
148
149         ~map(){
```

暨南大学本科实验报告专用纸(附页)

```
149     auto DeleteSubTree = [&](auto self,Node *p) -> void{
150         if(p == nullptr || p == NIL) return;
151         self(self,p->leftSubTree);
152         self(self,p->rightSubTree);
153         delete p;
154         return;
155     };
156     if(!(root == nullptr)) DeleteSubTree(DeleteSubTree,root);
157     delete NIL;
158 }
159
160     void insert(KEY key) {
161         if(root == nullptr) {
162             root = new Node();
163             root->color = COLOR::BLACK;
164             root->leftSubTree = NIL;
165             root->rightSubTree = NIL;
166             root->key = key;
167         } else {
168             if(this->locate(key,root)) return;
169             subInsert(root,key);
170         }
171     }
172
173     void insert(KEY key,VALUE value) {
174         if(root == nullptr) {
175             root = new Node();
176             root->color = COLOR::BLACK;
177             root->leftSubTree = NIL;
178             root->rightSubTree = NIL;
179             root->key = key;
180             root->value = value;
181         } else {
182             if(this->locate(key,root)) return;
183             insert(key);
184             (*this)[key] = value;
185         }
186     }
187
188     myDS::map<KEY,VALUE>::iterator find(KEY tar) {
189         if(this->locate(tar,root) != nullptr) return
myDS::map<KEY,VALUE>::iterator(this->locate(tar,root));
190         else return this->end();
191     }
192
193     bool erase(KEY key) {
194         return subDelete(root,key);
195     }
196
197     myDS::map<KEY,VALUE>::iterator begin(){
198         auto rt = iterator(root,NIL);
```

暨南大学本科实验报告专用纸(附页)

```
199         rt.loop2Begin();
200         return rt;
201     }
202
203     myDS::map<KEY,VALUE>::iterator end(){
204         return iterator(nullptr,NIL);
205     }
206
207     VALUE & operator[] (std::size_t key) {
208         if(root == nullptr) {
209             root = new Node();
210             root->color = COLOR::BLACK;
211             root->leftSubTree = NIL;
212             root->rightSubTree = NIL;
213             root->key = key;
214             return locate(key,root)->value;
215         } else {
216             if(locate(key,root) == nullptr) this->insert(key);
217             return locate(key,root)->value;
218         }
219     }
220
221 #ifdef __PRIVATE_DEBUGE
222     void printDfsOrder()
223     {
224         auto dfs = [&](auto self,Node * p ) -> void {
225             if(p == nullptr){ std::cout << "ED\n";return;}
226             if(p->leftSubTree == nullptr && p->rightSubTree ==
227                 nullptr) {std::cout << "[NIL] \n";return;}
228             std::cout << "[" << p->key << " : " << p->value << "] ";
229             self(self,p->leftSubTree);
230             self(self,p->rightSubTree);
231             return;
232         };
233         dfs(dfs,root);
234     }
235
236     vector<int> printList;
237     void printIterOrder()
238     {
239         auto dfs = [&](auto self,Node * p) -> void{
240             if(p->leftSubTree == nullptr && p->rightSubTree ==
241                 nullptr) {std::cout << "[NIL] \n";return;}
242             self(self,p->leftSubTree);
243             std::cout << "[" << p->key << " : " << p->value << "] ";
244             self(self,p->rightSubTree);
245         };
246         dfs(dfs,root);
247     }
248 #endif
249     private:
```

暨南大学本科实验报告专用纸(附页)

```
248     Node * locate(KEY t,Node * p) {
249         if(p == NIL) return nullptr;
250         else if(p->key == t) return p;
251         else if(p->key > t) return locate(t,p->leftSubTree);
252         else return locate(t,p->rightSubTree);
253     }
254
255     //右旋某个节点
256     void rotateRight(Node *p)
257     {
258         Node * _gp = p->getGrandParent();
259         Node * _pa = p->getParent();
260         Node * _rotY = p->rightSubTree;
261         _pa->leftSubTree = _rotY;
262         if(_rotY != NIL) _rotY->parent = _pa;
263         p->rightSubTree = _pa;
264         _pa->parent = p;
265
266         if(root == _pa) root = p;
267         p->parent = _gp;
268         if(_gp != nullptr) if(_gp->leftSubTree == _pa) _gp-
>leftSubTree = p;
269             else _gp->rightSubTree = p;
270         return;
271     }
272
273     //左旋某个节点
274     void rotateLeft(Node *p)
275     {
276         if(p->parent == nullptr){
277             root = p;
278             return;
279         }
280         Node * _gp = p->getGrandParent();
281         Node * _pa = p->parent;
282         Node * _rotX = p->leftSubTree;
283
284 #ifdef __DETIL_DEBUG_OUTPUT
285         printIterOrder();
286 #endif
287         _pa->rightSubTree = _rotX;
288
289 #ifdef __DETIL_DEBUG_OUTPUT
290         printIterOrder();
291 #endif
292
293         if(_rotX != NIL)
294             _rotX->parent = _pa;
295
296 #ifdef __DETIL_DEBUG_OUTPUT
297         printIterOrder();
298 #endif
```

暨南大学本科实验报告专用纸(附页)

```
298 #endif
299     p->leftSubTree = _pa;
300 #ifdef __DETIL_DEBUG_OUTPUT
301     printIterOrder();
302 #endif
303     _pa->parent = p;
304 #ifdef __DETIL_DEBUG_OUTPUT
305     printIterOrder();
306 #endif
307     if(root == _pa)
308         root = p;
309     p->parent = _gp;
310
311 #ifdef __DETIL_DEBUG_OUTPUT
312     printIterOrder();
313 #endif
314     if(_gp != nullptr){
315         if(_gp->leftSubTree == _pa)
316             _gp->leftSubTree = p;
317         else
318             _gp->rightSubTree = p; //?!
319     }
320 #ifdef __DETIL_DEBUG_OUTPUT
321     printIterOrder();
322 #endif
323 }
324
325 //插入节点递归部分
326 void subInsert(Node *p, KEY key)
327 {
328     if(p->key >= key){ //1 2
329         if(p->leftSubTree != NIL) //3
330             subInsert(p->leftSubTree, key);
331         else {
332             Node *tmp = new Node(); //3
333             tmp->key = key;
334             tmp->leftSubTree = tmp->rightSubTree = NIL;
335             tmp->parent = p;
336             p->leftSubTree = tmp;
337             resetStatus_forInsert(tmp);
338         }
339     } else {
340         if(p->rightSubTree != NIL) //1 2
341             subInsert(p->rightSubTree, key);
342         else {
343             Node *tmp = new Node();
344             tmp->key = key;
345             tmp->leftSubTree = tmp->rightSubTree = NIL;
346             tmp->parent = p;
347             p->rightSubTree = tmp;
348             resetStatus_forInsert(tmp);
349 }
```

暨南大学本科实验报告专用纸(附页)

```
349         }
350     }
351 }
352 }
353 //插入后的平衡维护
354 void resetStatus_forInsert(Node *p) {
355     //case 1:
356     if(p->parent == nullptr){
357         root = p;
358         p->color = COLOR::BLACK;
359         return;
360     }
361     //case 2-6:
362     if(p->parent->color == COLOR::RED){
363         //case 2: pass
364         if(p->getUncle()->color == COLOR::RED) {
365             p->parent->color = p->getUncle()->color =
366             COLOR::BLACK;
367             p->getGrandParent()->color = COLOR::RED;
368             resetStatus_forInsert(p->getGrandParent());
369         } else {
370             if(p->parent->rightSubTree == p && p-
371             >getGrandParent()->leftSubTree == p->parent) {
372                 //case 3:
373                 rotateLeft(p);
374                 p->color = COLOR::BLACK;
375                 p->parent->color = COLOR::RED;
376                 rotateRight(p);
377             } else if(p->parent->leftSubTree == p && p-
378             >getGrandParent()->rightSubTree == p->parent) { //this
379                 //case 4:
380                 rotateRight(p);
381                 p->color = COLOR::BLACK;
382                 p->parent->color = COLOR::RED;
383                 rotateLeft(p);
384             } else if(p->parent->leftSubTree == p && p-
385             >getGrandParent()->leftSubTree == p->parent) {
386                 //case 5:
387                 p->parent->color = COLOR::BLACK;
388                 p->getGrandParent()->color = COLOR::RED;
389                 rotateRight(p->parent);
390             } else if(p->parent->rightSubTree == p && p-
391             >getGrandParent()->rightSubTree == p->parent) {
392                 //case 6: BUG HERE
393                 p->parent->color = COLOR::BLACK;
394                 p->getGrandParent()->color = COLOR::RED;
395                 rotateLeft(p->parent);
396             }
397         }
398     }
399 }
```

暨南大学本科实验报告专用纸(附页)

```
394 }
395
396 //删除时的递归部分
397 bool subDelete(Node *p, KEY key){
398
399     //获取最接近叶节点的儿子
400     auto getLowwestChild = [&](auto self,Node *p) -> Node*{
401         if(p->leftSubTree == NIL) return p;
402         return self(self,p->leftSubTree);
403     };
404
405     if(p->key > key){
406         if(p->leftSubTree == NIL){
407             return false;
408         }
409         return subDelete(p->leftSubTree, key);
410     } else if(p->key < key){
411         if(p->rightSubTree == NIL){
412             return false;
413         }
414         return subDelete(p->rightSubTree, key);
415     } else if(p->key == key){
416         if(p->rightSubTree == NIL){
417             deleteChild(p);
418             return true;
419         }
420         Node *smallChild = getLowwestChild(getLowwestChild,p-
421             >rightSubTree);
422         std::swap(p->key, smallChild->key);
423         std::swap(p->value,smallChild->value);
424         deleteChild(smallChild);
425
426         return true;
427     } else{
428         return false;
429     }
430
431     // //删除入口
432     // bool deleteChild(Node *p, int key){
433     //     if(p->value > key){
434     //         if(p->leftSubTree == NIL){
435     //             return false;
436     //         }
437     //         return deleteChild(p->leftSubTree, key);
438     //     } else if(p->value < key){
439     //         if(p->rightSubTree == NIL){
440     //             return false;
441     //         }
442     //         return deleteChild(p->rightSubTree, key);
443     //     } else if(p->value == key){
```

暨南大学本科实验报告专用纸(附页)

```
444 // if(p->rightSubTree == NIL){
445 //     delete_one_child (p);
446 //     return true;
447 // }
448 // Node *smallest = getSmallestChild(p->rightTree);
449 // swap(p->value, smallest->value);
450 // delete_one_child (smallest);
451 //
452 //     return true;
453 // }else{
454 //     return false;
455 // }
456 // }
457
458 //删除处理: 删除某个儿子
459 void deleteChild(Node *p){
460     Node *child = p->leftSubTree == NIL ? p->rightSubTree : p-
>leftSubTree;
461     if(p->parent == nullptr && p->leftSubTree == NIL && p-
>rightSubTree == NIL){
462         p = nullptr;
463         root = p;
464         return;
465     }
466     if(p->parent == nullptr){
467         delete p;
468         child->parent = nullptr;
469         root = child;
470         root->color = COLOR::BLACK;
471         return;
472     }
473     if(p->parent->leftSubTree == p) p->parent->leftSubTree =
child;
474     else p->parent->rightSubTree = child;
475
476     child->parent = p->parent;
477     if(p->color == COLOR::BLACK){
478         if(child->color == COLOR::RED){
479             child->color = COLOR::BLACK;
480         } else
481             resetStatus_forDelete(child);
482     }
483
484     delete p;
485 }
486
487 //删除后的平衡维护
488 void resetStatus_forDelete(Node *p){
489     if(p->parent == nullptr){
490         //case 0-0:
491         p->color = COLOR::BLACK;
```

暨南大学本科实验报告专用纸(附页)

```
492         return;
493     }
494     if(p->getSibling()->color == COLOR::RED) {
495         //case 0-1:
496         p->parent->color = COLOR::RED;
497         p->getSibling()->color = COLOR::BLACK;
498         if(p == p->parent->leftSubTree) rotateLeft(p->parent);
499         else rotateRight(p->parent);
500     }
501     if( p->parent->color == COLOR::BLACK &&
502         p->getSibling()->color == COLOR::BLACK &&
503         p->getSibling()->leftSubTree->color == COLOR::BLACK &&
504         p->getSibling()->rightSubTree->color == COLOR::BLACK) {
505         //case 1-1:
506         p->getSibling()->color = COLOR::RED;
507         resetStatus_forDelete(p->parent);
508     } else if(p->parent->color == COLOR::RED && p->getSibling()-
509     >color == COLOR::BLACK&& p->getSibling()->leftSubTree->color ==
510     COLOR::BLACK && p->getSibling()->rightSubTree->color == COLOR::BLACK) {
511         //case 1-2:
512         p->getSibling()->color = COLOR::RED;
513         p->parent->color = COLOR::BLACK;
514     } else {
515         if(p->getSibling()->color == COLOR::BLACK) {
516             if(p == p->parent->leftSubTree && p->getSibling()-
517             >leftSubTree->color == COLOR::RED && p->getSibling()->rightSubTree-
518             >color == COLOR::BLACK) {
519                 //case 1-3:
520                 p->getSibling()->color = COLOR::RED;
521                 p->getSibling()->leftSubTree->color =
522                 COLOR::BLACK;
523                 rotateRight(p->getSibling()->leftSubTree);
524             } else if(p == p->parent->rightSubTree && p-
525             >getSibling()->leftSubTree->color == COLOR::BLACK && p->getSibling()-
526             >rightSubTree->color == COLOR::RED) {
527                 //case 1-4:
528                 p->getSibling()->color = COLOR::RED;
529                 p->getSibling()->rightSubTree->color =
530                 COLOR::BLACK;
531                 rotateLeft(p->getSibling()->rightSubTree);
532             }
533         }
534         p->getSibling()->color = p->parent->color;
535         p->parent->color = COLOR::BLACK;
536         //case 1-5:
537         if(p == p->parent->leftSubTree){
538             //case 0-3
539             p->getSibling()->rightSubTree->color = COLOR::BLACK;
540             rotateLeft(p->getSibling());
541         }
542     }
543 }
```

暨南大学本科实验报告专用纸(附页)

```
533     } else {
534         //case 0-4
535         p->getSibling()->leftSubTree->color = COLOR::BLACK;
536         rotateRight(p->getSibling());
537     }
538 }
539 }
540
541
542 };
543 } // namespace myDS
544 #endif
```

4.2. _PRIV_TEST.cpp

```
1 #include <iostream>
2 #define __PRIVATE_DEBUGE
3 #include <Dev\11\map.hpp>
4
5 using namespace std;
6
7 int main()
8 {
9     myDS::map<int,int> testMP;
10    while(1)
11    {
12        cout << "">>>>";
13        string s;
14        cin >> s;
15        if(s == "im") {
16            int t;
17            cin >> t;
18            testMP.insert(t);
19        } else if(s == "is") {
20            int t,v;
21            cin >> t >> v;
22            testMP.insert(t,v);
23        } else if(s == "p") {
24            std::cout << "==DFS Order=="<< endl;
25            testMP.printDfsOrder();
26            std::cout << "==Iter Order=="<< endl;
27            testMP.printIterOrder();
28            for(auto x:testMP) cout << "[" << x.first << " " << x.second
29            << "] ";
30            cout << "\n";
31        } else if(s == "d") {
32            int t;
33            cin >> t;
34            cout << testMP.erase(t) << "\n";
35        } else if(s == "f") {
```

暨南大学本科实验报告专用纸(附页)

```
35     int t;
36     cin >> t;
37     cout << testMP[t] << "\n";
38 } else if(s == "m") {
39     int t,v;
40     cin >> t >> v;
41     testMP[t] = v;
42 }
43 }
44 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
>>>is 1 3
>>>is 2 5
>>>is 3 7
>>>im 4
>>>p
====DFS Order====
[2 : 5] [1 : 3] [NIL]
[NIL]
[3 : 7] [NIL]
[4 : 0] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : 3] [NIL]
[2 : 5] [NIL]
[3 : 7] [NIL]
[4 : 0] [NIL]
[1 3] [2 5] [3 7] [4 0]
>>>f 2
5
>>>m 2 1145
>>>p
====DFS Order====
[2 : 1145] [1 : 3] [NIL]
[NIL]
[3 : 7] [NIL]
[4 : 0] [NIL]
[NIL]
```

暨南大学本科实验报告专用纸(附页)

```
====Iter Order====
[NIL]
[1 : 3] [NIL]
[2 : 1145] [NIL]
[3 : 7] [NIL]
[4 : 0] [NIL]
[1 3] [2 1145] [3 7] [4 0]
>>>d 2
1
>>>p
====DFS Order====
[3 : 7] [1 : 3] [NIL]
[NIL]
[4 : 0] [NIL]
[NIL]
====Iter Order====
[NIL]
[1 : 3] [NIL]
[3 : 7] [NIL]
[4 : 0] [NIL]
[1 3] [3 7] [4 0]
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

字典树 Trie

课程名称 数据结构 成绩评定

实验项目名称 字典树 Trie 指导老师 干晓聪

实验项目编号 12 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现一个字典树

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

基于一棵每层至多 26 分支的搜索树，实现在已有字典的基础上，查询字符串 s 在字典中是否出现。

规定每个节点 t 一一对应一个字母 c_t , t 的子节点列表 S_t

则 $\exists t' \in S_t \Leftrightarrow$ 在字典中有 c_t 的后缀为 $c_{t'}$

因此如果存在路径 $t_0 \rightarrow t_1 \dots t_{|s|} s.t. \forall i \in [1, |s|] \text{ 有 } t_i = s[i]$, 则 s 在字典中出现过。

容易证明，查询复杂度为 $\mathcal{O}(|s|)$ ，与字典大小无关。

对于字典规模 n , 建立时的时间复杂度为 $\mathcal{O}(n)$, 字典树空间复杂度为 $\mathcal{O}(n \log_{26} n)$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. template_overAll.h

```
1 #include <vector>
2 #include <map>
3 #include <string>
4 #include <string.h>
5 #include <math.h>
6 #include <set>
7 #include <algorithm>
8 #include <iostream>
9 #include <queue>
10
11 using namespace std;
12
13 #define ll long long
14 #define pb push_back
15 #define ld long double
16 const ll int maxn = 1E5+10;
17 const ll int mod1 = 998244353;
18 const ll int mod2 = 1E9+7;
19
20 #define _IN_TEMPLATE_
21
22 ll int str2int(string s)
23 {
24     ll int rec = 0;
25     ll int pw = 1;
26     for(int i = s.length()-1;i >= 0;i --)
27     {
28         int gt = s[i] - '0';
29         if(gt < 0 || gt > 9) return INT64_MAX;
30         rec += gt * pw;
31         pw *= 10;
32     }
33     return rec;
34 }
35
36 vector<ll int> testReadLine()
37 {
38     string s;
39     getline(cin,s);
40     s.push_back(' ');
41     vector<ll int> rearr;
42     vector<string> substring;
43     string ts;
44     for(int i = 0;i < s.size();i++)
45     {
46         if(s[i] == ' ')
47             substring.push_back(ts);
```

暨南大学本科实验报告专用纸(附页)

```
48         ts.clear();
49     } else ts.push_back(s[i]);
50 }
51     for(int i = 0;i < substring.size();i +
52 +)rearr.push_back(str2int(substring[i]));
52     return rearr;
53 }
```

4.2. trie_Tree.h

```
1 #include <template_overAll.h>
2
3 class Trie//AC
4 {
5 public:
6     vector<map<char, int>> t;
7     int root = 0;
8     Trie()
9     {
10         t.resize(1);
11     }
12     void addedge(string _s)
13     {
14         int pvidx = root;
15         _s.push_back('-');
16         for (int i = 0; i < _s.size(); i++)
17         {
18             if (t[pvidx].find(_s[i]) != t[pvidx].end())
19             {
20                 pvidx = t[pvidx][_s[i]];
21             }
22             else
23             {
24                 t[pvidx][_s[i]] = t.size();
25                 t.push_back(map<char, int>());
26                 pvidx = t[pvidx][_s[i]];
27             }
28         }
29     }
30     bool ifcmp(string &s)
31     {
32         int pvidx = root;
33         for(int i = 0;i < s.size();i++)
34         {
35             if(t[pvidx].find(s[i]) != t[pvidx].end()) pvidx = t[pvidx]
36 [s[i]];
37             else return 0;
38         }
39         return t[pvidx].find(' ') != t[pvidx].end();
```

暨南大学本科实验报告专用纸(附页)

```
39 }  
40 };
```

5. 测试数据与运行结果

代码通过在线平台 luogu.org 正确性测试

测试点信息 源代码

测试点信息

Subtask #0

#2 AC 4ms/680.00KB	#3 AC 4ms/684.00KB	#4 AC 4ms/692.00KB	#5 AC 44ms/3.54MB	#6 AC 222ms/36.80MB	#7 AC 297ms/37.57MB	#8 AC 298ms/38.28MB
#9 AC 508ms/36.61MB	#10 AC 623ms/38.09MB	#11 AC 257ms/36.57MB				

Subtask #1

#1 AC 3ms/680.00KB

GYPplus

所属题目 P2580 于是他错误的点名开始了

评测状态 Accepted

评测分数 100

提交时间 2023-11-05 16:52:07

暨南大学本科实验报告专用纸(附页)

线段树 segTree

课程名称 数据结构 成绩评定 _____

实验项目名称 线段树 segTree 指导老师 干晓聪

实验项目编号 13 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现一个泛型线段树 segTree 库

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

线段树可以在 $\mathcal{O}(\log n)$ 的时间复杂度内实现单点修改、区间修改、区间查询等操作。

定义合并运算符“ \oplus ”及其高阶运算“ \otimes ”

对于一个连续的序列 a 递归地对连续两个区域进行合并，形成新序列 t

$$s.t. \forall i \in t, s_i = s_{i*2} \oplus s_{i*2+1} \quad (1)$$

其中

$$\forall i \in a, \exists f, s.t. s_{f+i} = a_i \quad (2)$$

则整个 s 序列形成二叉搜索树形结构。

定义运算符 a_i^T 为递归地向上访问所经过的所有节点集合， s_i^L, s_i^R 分别为：

暨南大学本科实验报告专用纸(附页)

$$s_i^L = \text{Val}^{a_i} \min(i \in \mathbb{N} s.t. s_i \in a_i^T) \quad (3)$$

$$s_i^R = \text{Val}^{a_i} \max(i \in \mathbb{N} s.t. s_i \in a_i^T) \quad (4)$$

则可以描述：

对于区间求合并值：给定区间 $[l, r]$

$$\sum_{i \in [l, r]} a_i \Leftrightarrow \sum_{k \in \{i \mid i \in a_i^T\}} (s_k^R - s_k^L) \otimes s_k \quad (5)$$

易得上式 k 的规模为 $\mathcal{O}(\log_2 \mathbb{N})$

对于区间合并修改，保留懒惰标记 L_i

对于修改：区间 $[l, r]$ 均 $\oplus c$

$$\begin{aligned} & \forall i \in [l, r], s_i \oplus c \\ \Leftrightarrow & \forall k \in \{a_i^T, i \in [l, r]\} s.t. [s_k^R, s_k^L], L_k \oplus (s_k^R - s_k^L) \otimes c \end{aligned} \quad (6)$$

则每次查询改为

$$\sum_{i \in [l, r]} a_i \Leftrightarrow \sum_{k \in \{i \mid i \in a_i^T\}} \left((s_k^R - s_k^L) \otimes s_k + \sum_{i \in a_k^T} L_i \right) \quad (7)$$

易得两者规模均为 $\mathcal{O}(\log_2 \mathbb{N})$

在每次搜索时按 $L_{i*2} \otimes (s_{i*2}^R - s_{i*2}^L) \oplus L_{i*2+1} \otimes (s_{i*2+1}^R - s_{i*2+1}^L) = (s_i^R - s_i^L)$ ，
则可以证明，维护懒惰标记的均摊复杂度为 $\mathcal{O}(1)$

具体实现参考代码，没这么复杂（只是用数学语言描述比较麻烦而已）

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. segTree.h

```
1 #include <template_overAll.h>
2
3 // AC 带懒惰标记线段树
4 template <class TYPE_NAME>
5 class lazyTree
6 {
7     /*
8     * TYPE_NAME 需要支持: + += != 和自定义的合并运算符
9     * 实现了懒惰标记, 仅支持区间批量增加
10    * 区间批量减需要 TYPE_NAME 支持-, 且有-a = 0 - a
11    * 额外处理了一个单点修改为对应值的函数, 非原生实现, 其复杂度为 4logn
12    * 不提供在线
13    * 不提供持久化
14    */
15 private:
16     vector<TYPE_NAME> d;
17     vector<TYPE_NAME> a;
18     vector<TYPE_NAME> b;
19     int n;
20     const TYPE_NAMEINI = 0; // 不会影响合并运算的初始值, 如 max 取 INF, min
取 0, mti 取 1
21
22     void subbuild(int s, int t, int p)
23     {
24         if (s == t)
25         {
26             d[p] = a[s];
27             return;
28         }
29         int m = s + ((t - s) >> 1); // (s+t)/2
30         subbuild(s, m, p * 2);
31         subbuild(m + 1, t, p * 2 + 1);
32         d[p] = d[p * 2] + d[p * 2 + 1];
33         // 合并运算符 ↑
34     }
35
36     TYPE_NAME subGetSum(int l, int r, int s, int t, int p)
37     {
38         if (l <= s && t <= r)
39             return d[p];
40         int m = s + ((t - s) >> 1);
41         if (b[p] != 0)
42         {
43             d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运
算为应用懒惰标记
```

暨南大学本科实验报告专用纸(附页)

```
44     d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运
算为应用懒惰标记
45     b[p * 2] += b[p];                // 下传标记, 无需修改
46     b[p * 2 + 1] += b[p];            // 下传标记, 无需修改
47     b[p] = 0;
48 }
49 TYPE_NAME ansL =INI;
50 TYPE_NAME ansR =INI;
51 if (l <= m)
52     ansL = subGetSum(l, r, s, m, p * 2);
53 if (r > m)
54     ansR = subGetSum(l, r, m + 1, t, p * 2 + 1);
55 return ansL + ansR;
56 // 合并运算符↑
57 }
58
59 void subUpdate(int l, int r, TYPE_NAME c, int s, int t, int p)
60 {
61     if (l <= s && t <= r)
62     {
63         d[p] += (t - s + 1) * c; // 合并运算符的高阶运算 此处运算为修改
整匹配区间值
64         b[p] += c;                // 记录懒惰标记, 无需修改
65         return;
66     }
67     int m = s + ((t - s) >> 1);
68     if (b[p] != 0 && s != t)
69     {
70         d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运
算为应用懒惰标记
71         d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运
算为应用懒惰标记
72         b[p * 2] += b[p];                // 下传标记, 无需修改
73         b[p * 2 + 1] += b[p];            // 下传标记, 无需修改
74         b[p] = 0;
75     }
76     if (l <= m)
77         subUpdate(l, r, c, s, m, p * 2);
78     if (r > m)
79         subUpdate(l, r, c, m + 1, t, p * 2 + 1);
80     d[p] = d[p * 2] + d[p * 2 + 1];
81     // 合并运算符 ↑
82 }
83
84 public:
85     lazyTree(TYPE_NAME _n)
86     {
87         n = _n;
88         d.resize(4 * n + 5);
```

暨南大学本科实验报告专用纸(附页)

```
89         a.resize(4 * n + 5);
90         b.resize(4 * n + 5);
91     }
92
93     void build(vector<TYPE_NAME> _a)
94     {
95         a = _a;
96         subbuild(1, n, 1);
97     }
98
99     TYPE_NAME getsum(int l, int r)
100    {
101        return subGetSum(l, r, 1, n, 1);
102    }
103
104    void update(int l, int r, TYPE_NAME c) // 修改区间
105    {
106        subUpdate(l, r, c, 1, n, 1);
107    }
108
109    void update(int idx, TYPE_NAME tar)
110    { // 修改单点, 未测试
111        TYPE_NAME tmp = getsum(idx, idx);
112        tar -= tmp;
113        subUpdate(idx, idx, tar, 1, n, 1);
114    }
115};
```

4.2. template_overAll.h

```
1 #include <vector>
2 #include <map>
3 #include <string>
4 #include <string.h>
5 #include <math.h>
6 #include <set>
7 #include <algorithm>
8 #include <iostream>
9 #include <queue>
10
11 using namespace std;
12
13 #define ll long long
14 #define pb push_back
15 #define ld long double
16 const ll int maxn = 1E5+10;
17 const ll int mod1 = 998244353;
18 const ll int mod2 = 1E9+7;
19
20 #define _IN_TEMPLATE_
```

暨南大学本科实验报告专用纸(附页)

```
21
22 ll int str2int(string s)
23 {
24     ll int rec = 0;
25     ll int pw = 1;
26     for(int i = s.length()-1;i >= 0;i --)
27     {
28         int gt = s[i] - '0';
29         if(gt < 0 || gt > 9) return INT64_MAX;
30         rec += gt * pw;
31         pw *= 10;
32     }
33     return rec;
34 }
35
36 vector<ll int> testReadLine()
37 {
38     string s;
39     getline(cin,s);
40     s.push_back(' ');
41     vector<ll int> rearr;
42     vector<string> substring;
43     string ts;
44     for(int i = 0;i < s.size();i++)
45     {
46         if(s[i] == ' ')
47             substring.push_back(ts);
48             ts.clear();
49         } else ts.push_back(s[i]);
50     }
51     for(int i = 0;i < substring.size();i +
52 +)rearr.push_back(str2int(substring[i]));
53 }
```

5. 测试数据与运行结果

代码通过在线平台 LUOGU.org 正确性测试

The screenshot shows a successful submission on the GYplus platform for problem P3372. The submission details are as follows:

- 编程语言: C++20
- 代码长度: 4.40KB
- 用时: 477ms
- 内存: 11.15MB

The "Test Point Information" section lists 10 test cases (#1 to #10), all of which were accepted (AC). The results are summarized in the table below:

Test Case	Status	Time	Memory
#1	AC	4ms/60.00KB	
#2	AC	3ms/96.00KB	
#3	AC	3ms/688.00KB	
#4	AC	10ms/60.00KB	
#5	AC	10ms/79.00KB	
#6	AC	10ms/680.00KB	
#7	AC	10ms/680.00KB	
#8	AC	144ms/11.11MB	
#9	AC	142ms/11.07MB	
#10	AC	141ms/11.15MB	

The "Problem Details" section shows the problem ID P3372, category [模板], and tree 1. The status is Accepted with a score of 100, and the submission time is 2023-11-05 23:58:45.

暨南大学本科实验报告专用纸(附页)

堆 Heap

课程名称 数据结构 成绩评定 _____

实验项目名称 堆 Heap 指导老师 干晓聪

实验项目编号 14 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现一个基础堆结构

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

堆是一个完全二叉树，满足任意节点 a_i ，其子节点值均小于它，这是小根堆。
大根堆反之。

插入时，在整棵树右下角插入，并且按照大小逐个上浮。可以证明，这样操作过后堆依然满足其性质。

删除时，将根与右下角对换，并按照左右大小下压根节点，最后删除右下角节点即可。

可以证明两者复杂度均为 $\mathcal{O}(\log_2 n)$

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. big_root_heap.h

```
1 #ifndef PRIORITY_QUEUE_HPP
2 #define PRIORITY_QUEUE_HPP
3
4 #include <vector>
5
6 #ifdef __PRIVATE_DEBUGE
7 #include <iostream>
8#endif
9
10 namespace myDS
11 {
12     template<typename VALUE_TYPE>
13     class big_root_heap{
14         private:
15             std::vector<VALUE_TYPE> h;
16
17             void floow(std::size_t x) {
18                 while(x > 1 && h[x] > h[x/2]) {
19                     std::swap(h[x],h[x/2]);
20                     x >>= 1;
21                 }
22             }
23
24             void drown(std::size_t x) {
25                 while(x * 2 <= h.size()-1) {
26                     int t = x * 2;
27                     if(t + 1 <= h.size()-1 && h[t + 1] > h[t]) t++;
28                     if(h[t] <= h[x]) break;
29                     std::swap(h[x],h[t]);
30                     x = t;
31                 }
32             }
33
34         public:
35             big_root_heap() {h.push_back(0);}
36
37             ~big_root_heap() { }
38
39             void push(VALUE_TYPE t)
40             {
41                 h.push_back(t);
42                 floow(h.size()-1);
43             }
44
45             VALUE_TYPE top(){
46                 return h[1];
47             }
48 }
```

暨南大学本科实验报告专用纸(附页)

```
49     VALUE_TYPE pop(){
50         auto t = this->top();
51         std::swap(h[1],h[h.size()-1]);
52         h.pop_back();
53         drown(1);
54         return t;
55     }
56
57 #ifdef __PRIVATE_DEBUGE
58     void innerPrint(){
59         for(auto x:h) std::cout << x << " ";
60         std::cout << "\n";
61     }
62 #endif
63 };
64 } // namespace myDS
65
66
67 #endif
```

4.2. _PRIV_TEST.cpp

```
1 #include <iostream>
2 #define __PRIVATE_DEBUGE
3 #include <Dev\14\big_root_heap.h>
4 using namespace std;
5
6 int main()
7 {
8     myDS::big_root_heap<int> piq;
9     while(1) {
10         string s;
11         cin >> s;
12         if(s == "push") {
13             int t;
14             cin >> t;
15             piq.push(t);
16         } else if(s == "pop") {
17             cout << piq.pop() << "\n";
18         } else if(s == "top") {
19             cout << piq.top() << "\n";
20         } else if(s == "p") {
21             piq.innerPrint();
22         }
23     }
24 }
```

暨南大学本科实验报告专用纸(附页)

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
push 5
push 4
push 1
push 3
push 2
p
0 5 4 1 3 2
pop
5
p
0 4 3 1 2
top
4
pop
4
pop
3
pop
2
pop
1
p
0
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

基于 Heap 实现 priority_queue

课程名称 数据结构 成绩评定 _____

实验项目名称 基于 Heap 实现 priority_queue 指导老师 干晓聪

实验项目编号 15 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

基于 Heap 实现 priority_queue

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

priority_queue 要求实时维护序列中最大（或最小）的值，刚好符合堆的性质。

代码实现的 priority_queue 中提供了可选模板参数

`typename Compare = std::less<VALUE_TYPE>`，若传入 `std::greater` 则为最大值优先，反之为最小值优先。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. priority_queue.h

```
1 #ifndef PRIORITY_QUEUE_HPP
2 #define PRIORITY_QUEUE_HPP
3
4 #include <vector>
5 #include <functional>
6
7 #ifdef __PRIVATE_DEBUGE
8 #include <iostream>
9 #endif
10
11 namespace myDS {
12     template<typename VALUE_TYPE, typename Compare = std::less<VALUE_TYPE>>
13     class priority_queue {
14     private:
15         std::vector<VALUE_TYPE> h;
16         Compare comp;
17
18         void floow(std::size_t x) {
19             while (x > 1 && comp(h[x / 2], h[x])) {
20                 std::swap(h[x], h[x / 2]);
21                 x >= 1;
22             }
23         }
24
25         void drown(std::size_t x) {
26             while (x * 2 <= h.size() - 1) {
27                 int t = x * 2;
28                 if (t + 1 <= h.size() - 1 && comp(h[t], h[t + 1])) t++;
29                 if (!comp(h[x], h[t])) break;
30                 std::swap(h[x], h[t]);
31                 x = t;
32             }
33         }
34
35     public:
36         explicit priority_queue(const Compare& comp = Compare()) :
37             comp(comp) { h.push_back(VALUE_TYPE()); }
38
39         ~priority_queue() { }
40
41         void push(const VALUE_TYPE& t) {
42             h.push_back(t);
43             floow(h.size() - 1);
44         }
45
46         const VALUE_TYPE& top() const {
```

暨南大学本科实验报告专用纸(附页)

```
46     return h[1];
47 }
48
49
50     VALUE_TYPE pop() {
51         auto t = this->top();
52         std::swap(h[1], h[h.size() - 1]);
53         h.pop_back();
54         drown(1);
55         return t;
56     }
57
58 #ifdef __PRIVATE_DEBUGE
59     void innerPrint() {
60         for (auto x : h) std::cout << x << " ";
61         std::cout << "\n";
62     }
63 #endif
64 };
65
66 } // namespace myDS
67
68 #endif
```

4.2. _PRIV_TEST.cpp

```
1 #include <iostream>
2 #define __PRIVATE_DEBUGE
3 #include <Dev\15\priority_queue.h>
4 using namespace std;
5
6 int main()
7 {
8     myDS::priority_queue<int, greater<int>> piq;
9     while(1) {
10         string s;
11         cin >> s;
12         if(s == "push") {
13             int t;
14             cin >> t;
15             piq.push(t);
16         } else if(s == "pop") {
17             cout << piq.pop() << "\n";
18         } else if(s == "top") {
19             cout << piq.top() << "\n";
20         } else if(s == "p") {
21             piq.innerPrint();
22         }
23     }
24 }
```

暨南大学本科实验报告专用纸(附页)

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
push 5
push 4
push 1
push 3
push 2
p
0 5 4 1 3 2
pop
5
p
0 4 3 1 2
top
4
pop
4
pop
3
pop
2
pop
1
p
0
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

霍夫曼树 Huffman-tree

课程名称 数据结构 成绩评定 _____

实验项目名称 霍夫曼树 Huffman-tree 指导老师 干晓聪

实验项目编号 16 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

实现一个霍夫曼树并提供初始化后的编解码

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

定义 w_i 为节点 i 的权值, l_i 为深度, 则有 Huffman tree s.t. $\min WPL = \sum_{i=1}^n w_i l_i$

构造时循环地将权值最小的两棵树连接到新的节点即可, 容易证明本贪心过程可以构造霍夫曼树。

将霍夫曼树视作只有 0, 1 的字典树, 并维护叶节点与原字符的对应关系即可进行编解码。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. Huffman_tree.hpp

```
 1 #ifndef _HUFFMAN_TREE_HPP
 2 #define _HUFFMAN_TREE_HPP
 3
 4 #include <functional>
 5 #include <string>
 6 #include <map>
 7 #include <vector>
 8 #include <queue>
 9
10 namespace myDS {
11     class huffmanTree {
12     protected:
13         std::vector<int> pa;
14         std::vector<int> wei;
15         std::vector<std::pair<int,int>> s2code;
16
17         bool comp(std::pair<char,int> a,std::pair<char,int> b)
18     {
19         return a > b;
20     }
21
22         std::size_t cap = 0;
23         std::size_t MEX = 0;
24
25         void link(int a,int b) {
26             pa[a] = cap,pa[b] = cap;
27             s2code[cap] = {a,b};
28         }
29
30     public:
31         huffmanTree() { }
32
33         huffmanTree(std::vector<int> _wei) {buildup(_wei);}
34
35         void buildup(std::vector<int> _wei) {
36             wei = _wei;
37             MEX = wei.size()-1;
38             pa.resize(MEX * 2 + 2);
39             s2code.resize(MEX * 2 + 2);
40
41             std::priority_queue<std::pair<int,int>,std::vector<std::pair<int,int>>,std::greater<std::pair<int,int>> values;
42                 cap = MEX;
43                 for(int i = 1;i <= MEX;i++) {
44                     values.push({wei[i],i});
45                 }
46                 while(values.size()) {
```

暨南大学本科实验报告专用纸(附页)

```
46     cap++;
47     auto a = values.top();
48     values.pop();
49     if (values.size() == 0)
50         break;
51     auto b = values.top();
52     values.pop();
53     link(a.second,b.second);
54     values.push({a.first + b.first,cap});
55 }
56 }
57
58 int getWPL() {
59     int t = 0;
60     for(int i = 1;i <= MEX;i++) t += wei[i] *
61     (getPath(i).size());
62 }
63
64 std::vector<char> getPath(std::size_t n) {
65     std::vector<char> rt;
66     int t = n;
67     while(pa[t]) {
68         rt.push_back(s2code[pa[t]].first == t);
69         t = pa[t];
70     }
71     std::vector<char> path;
72     for(int i = 0;i < rt.size();i++)
73         path.push_back(rt[rt.size()-1-i]);
74     return path;
75 }
76
77 std::vector<int> getC(std::vector<char> t) {
78     int ori = cap-1;
79     std::vector<int> rt;
80     for(auto x:t) {
81         if(s2code[ori] == std::pair<int,int>()){
82             rt.push_back(ori);
83             ori = cap-1;
84         }
85         if(x == 0) ori = s2code[ori].second;
86         else ori = s2code[ori].first;
87     }
88     if(s2code[ori] == std::pair<int,int>()){
89         rt.push_back(ori);
90         ori = cap;
91     }
92     return rt;
93 }
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
```

暨南大学本科实验报告专用纸(附页)

```
95 class huffmanEncoder : huffmanTree {
96 private:
97
98     std::map<char,int> wordCounter;
99     std::map<char,int> c2i;
100    std::map<int,char> i2c;
101    std::string init;
102
103 public:
104     huffmanEncoder(std::string _init) : huffmanTree(){
105         init = _init;
106         for(auto x:_init) wordCounter[x]++;
107         pa.resize(wordCounter.size()*2+1);
108         std::vector<std::pair<char,int>> gt(1);
109         for(auto x:wordCounter) gt.push_back(x);
110         for(int i = 1;i < gt.size();i++) c2i[gt[i].first] = i;
111         for(int i = 1;i < gt.size();i++) i2c[i] = gt[i].first;
112         std::vector<int> wei;
113         for(auto x:gt) wei.push_back(x.second);
114         buildup(wei);
115     };
116
117     std::vector<char> encode(std::string s) {
118         std::vector<char> rt;
119         auto add = [&](std::vector<char> addit) -> void{
120             for(int i = 0;i < addit.size();i++)
121                 rt.push_back(addit[i]);
122             };
123             for(auto x:s) add(this->getPath(c2i[x]));
124             return rt;
125         }
126
127         std::string decode(std::vector<char> r) {
128             std::string rt;
129             for(auto x:getC(r)) rt.push_back(i2c[x]);
130             return rt;
131         }
132     };
133
134 };
135
136 #endif
```

4.2. _PRIV_TEST.cpp

```
1 #include <iostream>
2 #define __PRIVATE_DEBUGE
3 #include <Dev\16\Huffman_tree.hpp>
4 using namespace std;
```

暨南大学本科实验报告专用纸(附页)

```
5 int main()
6 {
7     string s;
8     cin >> s;
9     myDS::huffmanEncoder hfe(s);
10    for(auto x:hfe.encode(s)) cout << (bool)x << " ";
11    cout << "\n";
12    for(auto x:hfe.decode(hfe.encode(s))) cout << x << " ";
13    cout << "\n";
14    system("pause");
15 }
16 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
aaaaaaabbcccd
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1
a a a a a b b b c c c d
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

算数表达式求值（栈）

课程名称 数据结构 成绩评定 ____

实验项目名称 算数表达式求值（栈） 指导老师 干晓聪

实验项目编号 17 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

基于逆波兰式求算术表达式值

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

逆波兰式指的是不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行的运算式。

将前缀表达式转换为逆波兰式，使用两个栈，按以下流程：

1. 数字：入临时栈
2. 左括号：入符号栈
3. 右括号：重复弹出符号栈内容插入临时栈，直到遇到左括号
4. 其他运算符：重复弹出栈内符号直到栈顶运算符优先级不低于自身，入符号栈

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. linnerCaculate.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <stdlib.h>
4 #include <map>
5 #include <algorithm>
6 #include <stack>
7 #include <math.h>
8
9 using namespace std;
10 using pii = pair<int,int>;
11
12 #define int long long
13 #define pb push_back
14 #define F first
15 #define S second
16 #define all(x) x.begin(),x.end()
17 #define loop(i,n) for(int i = 0; i < n; i++)
18
19 const int mod = 1e9 + 7;
20 const int INF = 1e18;
21
22 bool ifdigit(string &s){
23     if(s.size() >= 2) return 1;
24     else return (s[0] > '0' && s[0] < '9');
25 }
26
27 int str2int(string &s) {
28     int t = 0;
29     for(int i = 0;i < s.size();i++) {
30         t += pow(10LL,i) * (s[s.size()-1-i] - '0');
31     } return t;
32 }
33
34 int getPri(string s){
35     if(s[0] == '+' || s[0] == '-') return 0;
36     else return 1;
37 }
38
39 //
40 void solve()
41 {
42     string s;
43     cout << ">>>";
44     getline(cin,s);
45     // cin.get();
46     s.push_back('\n');
47     vector<string> RES;
```

暨南大学本科实验报告专用纸(附页)

```
48     stack<string> sign,tmpl,rbc;
49     string tmp;
50     for(auto x:s) {
51         if(x == ' ' || x == '\n'){
52             if(tmp.size()) RES.push_back(tmp);
53             tmp.clear();
54         } else tmp.push_back(x);
55     }
56     for(auto x:RES){
57         if(ifdigit(x)) tmpl.push(x);
58         else{
59             if(x[0] == '(') sign.push(x);
60             else if(x[0] == ')') {
61                 while(sign.top()[0] != '(' && sign.size()){
62                     tmpl.push(sign.top());
63                     sign.pop();
64                 }
65                 sign.pop();
66             } else {
67                 while(1) {
68                     if (sign.size() == 0 || sign.top()[0] == '(')
69                     { sign.push(x);break;}
70                     else {
71                         if(getPri(x) > getPri(sign.top()))
72                         { sign.push(x);break;}
73                         else {tmpl.push(sign.top());sign.pop();}
74                     }
75                 }
76             }
77             while(sign.size()) {
78                 tmpl.push(sign.top());
79                 sign.pop();
80             }
81             vector<string> gt;
82             while(tmpl.size()){
83                 gt.push_back(tmpl.top());
84                 tmpl.pop();
85             }
86             vector<string> cacu;
87             for(int i = 0;i < gt.size();i++)cacu.push_back(gt[gt.size()-1-i]);
88
89             stack<int> nums;
90             for(auto x:cacu) {
91                 if(ifdigit(x)) nums.push(str2int(x));
92                 else {
93                     int a = nums.top();
94                     nums.pop();
95                     int b = nums.top();
```

暨南大学本科实验报告专用纸(附页)

```
96     nums.pop();
97     char c = x[0];
98     switch (c)
99     {
100     case '+':
101         nums.push(a+b);
102         break;
103     case '-':
104         nums.push(a-b);
105         break;
106     case '*':
107         nums.push(a*b);
108         break;
109     case 'x':
110         nums.push(a*b);
111         break;
112     case '/':
113         nums.push(a/b);
114         break;
115     default:
116         break;
117     }
118 }
119 }
120 }
121 cout << nums.top() << "\n";
122 }
123 }
124 }
125
126 signed main()
127 {
128 // std::ios::sync_with_stdio(false);
129 // std::cin.tie(nullptr);
130 // std::cout.tie(nullptr);
131
132 // int T = 1;
133 // cin >> T;
134 while(1)
135     solve();
136 return 0;
137 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

```
>>>1 + 1
```

暨南大学本科实验报告专用纸(附页)

```
2
>>>2 * 3 + 1
7
>>>2 * ( 3 + 1 )
8
>>>115 * 514
59110
>>>( 1 + 1 ) / 2 * 2 + 2
4
>>>( 1 + 1 ) / 2 * ( 2 + 2 + 2 )
4
>>>( 1 + 1 ) / 2 * ( 2 + 2 + 2 )
6
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

括号匹配（栈）

课程名称 数据结构 成绩评定

实验项目名称 括号匹配（栈） 指导老师 干晓聪

实验项目编号 18 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

判断括号是否匹配

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

没啥好讲的... 左括号入栈，右括号能消就消，不能消不管，结束时栈空即合法。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. seq.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 using namespace std;
5
6 void solve()
7 {
8     string s;
9     cin >> s;
10    stack<char> c;
11    for(auto x:s) if(c.size()) if((x == ')') && c.top() == '(') c.pop();
12    else c.push(x); else c.push(x);
13    cout << (c.size() == 0 ? "yes" : "no") << "\n";
14 }
15 signed main()
16 {
17     while(1)
18         solve();
19     return 0;
20 }
```

5. 测试数据与运行结果

运行上述 _PRIV_TEST.cpp 测试代码中的正确性测试模块，得到以下内容：

```
()()
yes
()(
no
()())
no
(())
yes
())(
no
())((((())))
yes
```

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

暨南大学本科实验报告专用纸(附页)

高精度计算

课程名称 数据结构 成绩评定

实验项目名称 高精度计算 指导老师 干晓聪

实验项目编号 19 实验项目类型 设计性 实验地点 数学系机房

学生姓名 郭彦培 学号 2022101149

学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统

实验时间 2024年6月13日上午 ~ 2024年7月13日中午

1. 实验目的

利用扩增数组实现对大整数的四则运算

2. 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

3. 程序原理

模拟手动计算流程即可。

暨南大学本科实验报告专用纸(附页)

4. 程序代码

4.1. hghCacu.hpp

```
1 #include <cstdio>
2 #include <iostream>
3 #include <cmath>
4 #include <string>
5 #include <cstring>
6 #include <vector>
7 #include <algorithm>
8 using namespace std;
9 class BigInt {
10 public:
11     int sign;
12     std::vector<int> v;
13
14     BigInt() : sign(1) {}
15     BigInt(const std::string &s) { *this = s; }
16     BigInt(int v) {
17         char buf[21];
18         sprintf(buf, "%d", v);
19         *this = buf;
20     }
21     void zip(int unzip) {
22         if (unzip == 0) {
23             for (int i = 0; i < (int)v.size(); i++)
24                 v[i] = get_pos(i * 4) + get_pos(i * 4 + 1) * 10 +
25             get_pos(i * 4 + 2) * 100 + get_pos(i * 4 + 3) * 1000;
26         } else
27             for (int i = (v.resize(v.size() * 4), (int)v.size() - 1), a;
28 i >= 0; i--)
29                 a = (i % 4 >= 2) ? v[i / 4] / 100 : v[i / 4] % 100, v[i]
30 = (i & 1) ? a / 10 : a % 10;
31             setsign(1, 1);
32         int get_pos(unsigned pos) const { return pos >= v.size() ? 0 :
33 v[pos]; }
34         BigInt &setsign(int newsign, int rev) {
35             for (int i = (int)v.size() - 1; i > 0 && v[i] == 0; i--)
36                 v.erase(v.begin() + i);
37             sign = (v.size() == 0 || (v.size() == 1 && v[0] == 0)) ? 1 :
38 (rev ? newsign * sign : newsign);
39             return *this;
40         }
41         std::string to_str() const {
42             BigInt b = *this;
43             std::string s;
44             for (int i = (b.zip(1), 0); i < (int)b.v.size(); ++i)
```

暨南大学本科实验报告专用纸(附页)

```
41         s += char(*b.v.rbegin() + i) + '0');
42     return (sign < 0 ? "-" : "") + (s.empty() ? std::string("0") :
43 s);
44 }
45     bool absless(const BigInt &b) const {
46     if (v.size() != b.v.size()) return v.size() < b.v.size();
47     for (int i = (int)v.size() - 1; i >= 0; i--)
48         if (v[i] != b.v[i]) return v[i] < b.v[i];
49     return false;
50 }
51     BigInt operator-() const {
52     BigInt c = *this;
53     c.sign = (v.size() > 1 || v[0]) ? -c.sign : 1;
54     return c;
55 }
56     BigInt &operator=(const std::string &s) {
57     if (s[0] == '-')
58         *this = s.substr(1);
59     else {
60         for (int i = (v.clear(), 0); i < (int)s.size(); ++i)
61             v.push_back(*(s.rbegin() + i) - '0');
62         zip(0);
63     }
64     return setsign(s[0] == '-' ? -1 : 1, sign = 1);
65 }
66     bool operator<(const BigInt &b) const {
67     return sign != b.sign ? sign < b.sign : (sign == 1 ? absless(b) :
68 b.absless(*this));
69 }
70     bool operator==(const BigInt &b) const { return v == b.v && sign ==
71 b.sign; }
72     BigInt &operator+=(const BigInt &b) {
73     if (sign != b.sign) return *this = (*this) - -b;
74     v.resize(std::max(v.size(), b.v.size()) + 1);
75     for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
76         carry += v[i] + b.get_pos(i);
77         v[i] = carry % 10000, carry /= 10000;
78     }
79     return setsign(sign, 0);
80 }
81     BigInt operator+(const BigInt &b) const {
82     BigInt c = *this;
83     return c += b;
84 }
85     void add_mul(const BigInt &b, int mul) {
86     v.resize(std::max(v.size(), b.v.size()) + 2);
87     for (int i = 0, carry = 0; i < (int)b.v.size() || carry; i++) {
88         carry += v[i] + b.get_pos(i) * mul;
89         v[i] = carry % 10000, carry /= 10000;
90     }
91 }
```

暨南大学本科实验报告专用纸(附页)

```
88 }
89 BigInt operator-(const BigInt &b) const {
90     if (b.v.empty() || b.v.size() == 1 && b.v[0] == 0) return *this;
91     if (sign != b.sign) return (*this) + -b;
92     if (absless(b)) return -(b - *this);
93     BigInt c;
94     for (int i = 0, borrow = 0; i < (int)v.size(); i++) {
95         borrow += v[i] - b.get_pos(i);
96         c.v.push_back(borrow);
97         c.v.back() -= 10000 * (borrow >>= 31);
98     }
99     return c.setsign(sign, 0);
100 }
101 BigInt operator*(const BigInt &b) const {
102     if (b < *this) return b * *this;
103     BigInt c, d = b;
104     for (int i = 0; i < (int)v.size(); i++, d.v.insert(d.v.begin(),
105         0))
106         c.add_mul(d, v[i]);
107     return c.setsign(sign * b.sign, 0);
108 }
109 BigInt operator/(const BigInt &b) const {
110     BigInt c, d;
111     BigInt e=b;
112     e.sign=1;
113     d.v.resize(v.size());
114     double db = 1.0 / (b.v.back() + (b.get_pos((unsigned)b.v.size() -
115         2) / 1e4) +
116                         (b.get_pos((unsigned)b.v.size() - 3) + 1) /
117                         1e8);
118     for (int i = (int)v.size() - 1; i >= 0; i--) {
119         c.v.insert(c.v.begin(), v[i]);
120         int m = (int)((c.get_pos((int)e.v.size()) * 10000 +
121             c.get_pos((int)e.v.size() - 1)) * db);
122         c = c - e * m, c.setsign(c.sign, 0), d.v[i] += m;
123         while (!(c < e))
124             c = c - e, d.v[i] += 1;
125     }
126     return d.setsign(sign * b.sign, 0);
127 }
128 BigInt operator%(const BigInt &b) const { return *this - *this / b *
129 b; }
130 bool operator>(const BigInt &b) const { return b < *this; }
131 bool operator<=(const BigInt &b) const { return !(b < *this); }
132 bool operator>=(const BigInt &b) const { return !(*this < b); }
133 bool operator!=(const BigInt &b) const { return !(*this == b); }
134 };
```

暨南大学本科实验报告专用纸(附页)

4.2. _PRIV_TEST.cpp

```
1 #include <Dev\19\hghCacu.hpp>
2
3 int main()
4 {
5     string aa,bb;
6     cin >> aa >> bb;
7     BigInt a = BigInt(aa);
8     BigInt b = BigInt(bb);
9     cout << ( a + b ).to_str() << '\n';
10    cout << ( a - b ).to_str()<< '\n';
11    cout << ( a * b ).to_str()<< '\n';
12    system("pause");
13
14 }
```

5. 测试数据与运行结果

运行上述 `_PRIV_TEST.cpp` 测试代码中的正确性测试模块，得到以下内容：

11
22
33
-11
2469135802469135802469135802469135802468641975308641975308641975308641975308642

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

附录与参考资料

APPENDIX AND REFERENCES

暨南大学本科实验报告专用纸(附页)

生成每份报告基础格式的 python 代码：

```
import os

def load_typ_file(rep_dir):
    typ_file_path = os.path.join(rep_dir, 'template.typ')
    with open(typ_file_path, 'r') as file:
        typ_content = file.read()
    return typ_content

def read_dev_files(dev_dir):
    file_data = {}
    for subdir, _, files in os.walk(dev_dir):
        for file_name in files:
            if file_name.endswith('.cpp', '.h', '.hpp'):
                file_path = os.path.join(subdir, file_name)
                with open(file_path, 'r') as file:
                    content = file.read()
                file_data[file_path] = content
    return file_data

def read_dev_files(dev_dir):
    all_data = {}
    for subdir, _, files in os.walk(dev_dir):
        subdir_data = {}
        for file_name in files:
            if file_name.endswith('.cpp', '.h', '.hpp'):
                file_path = os.path.join(subdir, file_name)
                with open(file_path, 'r') as file:
                    content = file.read()
                subdir_data[file_name] = content
        if subdir_data:
            all_data[subdir] = subdir_data
    return all_data

import re

def split_text(text):
    pattern = r'/*.*\/'
    matches = re.findall(pattern, text, re.DOTALL)

    if matches:
        last_match = matches[-1]
        remaining_text = text.replace(last_match, '', 1)
        last_match = last_match.replace("/*", "")
        last_match = last_match.replace("*/", "")
    return remaining_text, last_match
```

暨南大学本科实验报告专用纸(附页)

```
else:
    return text,""
s = '''      rowspanx(8)[STL 风格的\ 泛型的\ 基础数据结构\ 容器实现],[1],[基于
双向链表的`linkedList`],
(),[2],[基于增长数组的`vector`],
(),[3],[基于块状数组的`dataBlock`],
(),[4],[实现基于循环增长数组的`deque`],
(),[5],[基于`vector`实现`stack`],
(),[10],[基于 R-BTree 实现`set`],
(),[11],[基于 R-BTree 实现`map`],
(),[15],[基于 Heap 实现`priority_queue`],
rowspanx(4)[基础树/图结构],[6],[树上 dfs (基础信息) ],
(),[7],[图上 bfs (最短路) ],
(),[8],[二叉树三序遍历],
(),[9],[R-BTree 的基本实现],
rowspanx(4)[特殊结构\ 及其应用],[12],[字典树`Trie`],
(),[13],[线段树`segTree`],
(),[14],[堆`Heap`],
(),[16],[霍夫曼树`Huffman-tree`],
rowspanx(3)[在算法中应用],[17],[算数表达式求值 (栈) ],
(),[18],[括号匹配 (栈) ],
(),[19],[高精度计算], ...
t = s.split("\n");
m = {};
for x in t:
    text = "rowspanx(8)[STL 风格的\\ 泛型的\\ 基础数据结构\\ 容器实现],[1],[基于
双向链表的`linkedList`]"  
  
# 定义正则表达式模式
pattern = re.compile(r'\[(\w+)\]+')
matches = pattern.findall(x)
m[int(matches[-2])] = matches[-1];
#     print(matches)
m
rep_dir = 'Rep'
dev_dir = 'Dev'  
  
# Load Rep/0.typ file
template = load_typ_file(rep_dir)
# print(f"Loaded 0.typ content:\n{typ_content}")  
  
# Read files in Dev subdirectories
file_data = read_dev_files(dev_dir)
```

暨南大学本科实验报告专用纸(附页)

```
# Print the loaded file names and content (for demonstration purposes)
for file_path, sub_paths in file_data.items():
    print(f"File: {file_path}")
    MAINCODE = ""
    COMMENTS = ""
    for sub_path, text in sub_paths.items():
        code, comment = split_text(text);
        MAINCODE += "==" ` + sub_path + "` \n"
        MAINCODE += "#sourcecode[
```

cppn" + code + "n

```
]` \n"
if len(comment) > 0 :
    COMMENTS += "
```

n" + comment + "n

```
\n"
subdir_name = os.path.basename(file_path);
nt = template
nt = nt.replace("MAINCODE",MAINCODE);
nt = nt.replace("TESTCASES",COMMENTS);
nt = nt.replace("maintitle",m[int(subdir_name)])
nt = nt.replace("INDEXS",subdir_name);
# print(m[int(subdir_name)])
# if(int(subdir_name) <= 18):
#     continue
rep_file_path = os.path.join(rep_dir, f"{subdir_name}.typ")
f = open(rep_file_path,'w')
f.write(nt)
# print(rep_file_path);
# print(f"Content:\n{content}")
# print('-' * 80)
```

渲染报告用 typst 模板：

```
#import "@preview/tablex:0.0.6": tablex, hlinex, vlinex, colspanx, rowspanx
#import "@preview/codelst:2.0.1": sourcecode
// Display inline code in a small box
// that retains the correct baseline.
```

暨南大学本科实验报告专用纸(附页)

```
#set text(font:"Times New Roman","Source Han Serif SC"))
#show raw: set text(
    font: ("consolas", "Source Han Serif SC")
)
#set page(
    paper: "a4",
)
#set text(
    font:("Times New Roman","Source Han Serif SC"),
    style:"normal",
    weight: "regular",
    size: 13pt,
)

#let nxtIdx(name) = box[ #counter(name).step()#counter(name).display()]
#set math.equation(numbering: "(1)")

#show raw.where(block: true): block.with(
    fill: luma(240),
    inset: 10pt,
    radius: 4pt,
)

#set math.equation(numbering: "(1)")

#set page(
    paper:"a4",
    number-align: right,
    margin: (x:2.54cm,y:4cm),
    header: [
        #set text(
            size: 25pt,
            font: "KaiTi",
        )
        #align(
            bottom + center,
            [ #strong[暨南大学本科实验报告专用纸(附页)] ]
        )
        #line(start: (0pt,-5pt),end:(453pt,-5pt))
    ]
)
/*
-----*/
```

暨南大学本科实验报告专用纸(附页)

```
= maintitle
\
#text(
    font: "KaiTi",
    size: 15pt
)[
课程名称 #underline[#text("      数据结构      ")]成绩评定 #underline[#text("")]
")]\ \
实验项目名称 #underline[#text(" ") maintitle #text(" ")]指导老师
#underline[#text("  干晓聪  ")]\ \
实验项目编号 #underline[#text("  INDEXS  ")])实验项目类型 #underline[#text(" "
设计性  ")])实验地点 #underline[#text(" 数学系机房 ")]\ \
学生姓名 #underline[#text("  郭彦培  ")])学号 #underline[#text("  2022101149
")]\ \
学院 #underline[#text(" 信息科学技术学院 ")]系 #underline[#text("  数学系  ")])专业
#underline[#text("  信息管理与信息系统  ")]\ \
实验时间 #underline[#text(" 2024 年 6 月 13 日上午
")]\#text("~")#underline[#text(" 2024 年 7 月 13 日中午  ")])\ \
]
#set heading(
    numbering: "1.1."
)
```

= 实验目的

= 实验环境

计算机: PC X64

操作系统: Windows + Ubuntu20.0LTS

编程语言: C++: GCC std20

IDE: Visual Studio Code

= 程序原理

\

#pagebreak()

暨南大学本科实验报告专用纸(附页)

= 程序代码

MAINCODE

= 测试数据与运行结果

运行上述`_PRIV_TEST.cpp`测试代码中的正确性测试模块，得到以下内容：

TESTCASES

可以看出，代码运行结果与预期相符，可以认为代码正确性无误。

用作参考的 RB_tree 实现 (CPP STL)

```
#ifndef RBTREE_MAP_HPP
#define RBTREE_MAP_HPP

#include <cassert>
#include <cstddef>
#include <cstdint>
#include <functional>
#include <memory>
#include <stack>
#include <utility>
#include <vector>

template <typename Key, typename Value, typename Compare = std::less<Key>>
class RBTreeMap {
private:
    using USize = size_t;

    Compare compare = Compare();

public:
    struct Entry {
        Key key;
        Value value;

        bool operator==(const Entry &rhs) const noexcept {

```

暨南大学本科实验报告专用纸(附页)

```
        return this->key == rhs.key && this->value == rhs.value;
    }

    bool operator!=(const Entry &rhs) const noexcept {
        return this->key != rhs.key || this->value != rhs.value;
    }
};

private:
    struct Node {
        using Ptr = std::shared_ptr<Node>;
        using Provider = const std::function<Ptr(void)> &;
        using Consumer = const std::function<void(const Ptr &)> &;

        enum { RED, BLACK } color = RED;

        enum Direction { LEFT = -1, ROOT = 0, RIGHT = 1 };

        Key key;
        Value value{};

        Ptr parent = nullptr;
        Ptr left = nullptr;
        Ptr right = nullptr;

        explicit Node(Key k) : key(std::move(k)) {}

        explicit Node(Key k, Value v) : key(std::move(k)),
value(std::move(v)) {}

        ~Node() = default;

        inline bool isLeaf() const noexcept {
            return this->left == nullptr && this->right == nullptr;
        }

        inline bool isRoot() const noexcept { return this->parent ==
nullptr; }

        inline bool isRed() const noexcept { return this->color == RED; }

        inline bool isBlack() const noexcept { return this->color ==
BLACK; }
    };
};
```

暨南大学本科实验报告专用纸(附页)

```
inline Direction direction() const noexcept {
    if (this->parent != nullptr) {
        if (this == this->parent->left.get()) {
            return Direction::LEFT;
        } else {
            return Direction::RIGHT;
        }
    } else {
        return Direction::ROOT;
    }
}

inline Ptr &sibling() const noexcept {
    assert(!this->isRoot());
    if (this->direction() == LEFT) {
        return this->parent->right;
    } else {
        return this->parent->left;
    }
}

inline bool hasSibling() const noexcept {
    return !this->isRoot() && this->sibling() != nullptr;
}

inline Ptr &uncle() const noexcept {
    assert(this->parent != nullptr);
    return parent->sibling();
}

inline bool hasUncle() const noexcept {
    return !this->isRoot() && this->parent->hasSibling();
}

inline Ptr &grandParent() const noexcept {
    assert(this->parent != nullptr);
    return this->parent->parent;
}

inline bool hasGrandParent() const noexcept {
    return !this->isRoot() && this->parent->parent != nullptr;
}

inline void release() noexcept {
```

暨南大学本科实验报告专用纸(附页)

```
// avoid memory leak caused by circular reference
this->parent = nullptr;
if (this->left != nullptr) {
    this->left->release();
}
if (this->right != nullptr) {
    this->right->release();
}
}

inline Entry entry() const { return Entry{key, value}; }

static Ptr from(const Key &k) { return
std::make_shared<Node>(Node(k)); }

static Ptr from(const Key &k, const Value &v) {
    return std::make_shared<Node>(Node(k, v));
}
};

using NodePtr = typename Node::Ptr;
using ConstNodePtr = const NodePtr &;
using Direction = typename Node::Direction;
using NodeProvider = typename Node::Provider;
using NodeConsumer = typename Node::Consumer;

NodePtr root = nullptr;
USize count = 0;

using K = const Key &;
using V = const Value &;

public:
    using EntryList = std::vector<Entry>;
    using KeyValueConsumer = const std::function<void(K, V)> &;
    using MutKeyValueConsumer = const std::function<void(K, Value &)> &;
    using KeyValueFilter = const std::function<bool(K, V)> &;

    class NoSuchMappingException : protected std::exception {
private:
    const char *message;

public:
    explicit NoSuchMappingException(const char *msg) : message(msg) {}
```

暨南大学本科实验报告专用纸(附页)

```
        const char *what() const noexcept override { return message; }

};

RBTreeMap() noexcept = default;

~RBTreeMap() noexcept {
    // Unlinking circular references to avoid memory leak
    this->clear();
}

/**
 * Returns the number of entries in this map.
 * @return size_t
 */
inline USize size() const noexcept { return this->count; }

/**
 * Returns true if this collection contains no elements.
 * @return bool
 */
inline bool empty() const noexcept { return this->count == 0; }

/**
 * Removes all of the elements from this map.
 */
void clear() noexcept {

    if (this->root != nullptr) {
        this->root->release();
        this->root = nullptr;
    }
    this->count = 0;
}

Value get(K key) const {
    if (this->root == nullptr) {
        throw NoSuchMappingException("Invalid key");
    } else {
        NodePtr node = this->getNode(this->root, key);
        if (node != nullptr) {
            return node->value;
        } else {
            throw NoSuchMappingException("Invalid key");
        }
    }
}
```

暨南大学本科实验报告专用纸(附页)

```
        }
    }
}

Value &getOrDefault(K key) {
    if (this->root == nullptr) {
        this->root = Node::from(key);
        this->root->color = Node::BLACK;
        this->count += 1;
        return this->root->value;
    } else {
        return this
            ->getNodeOrProvide(this->root, key,
                [&key]() { return
Node::from(key); })
            ->value;
    }
}

bool contains(K key) const {
    return this->getNode(this->root, key) != nullptr;
}

void insert(K key, V value) {
    if (this->root == nullptr) {
        this->root = Node::from(key, value);
        this->root->color = Node::BLACK;
        this->count += 1;
    } else {
        this->insert(this->root, key, value);
    }
}

bool insertIfAbsent(K key, V value) {
    USize sizeBeforeInsertion = this->size();
    if (this->root == nullptr) {
        this->root = Node::from(key, value);
        this->root->color = Node::BLACK;
        this->count += 1;
    } else {
        this->insert(this->root, key, value, false);
    }
    return this->size() > sizeBeforeInsertion;
}
```

暨南大学本科实验报告专用纸(附页)

```
Value &getOrInsert(K key, V value) {
    if (this->root == nullptr) {
        this->root = Node::from(key, value);
        this->root->color = Node::BLACK;
        this->count += 1;
        return root->value;
    } else {
        NodePtr node = getNodeOrProvide(this->root, key,
[&]() { return Node::from(key, value); });
        return node->value;
    }
}

Value operator[](K key) const { return this->get(key); }

Value &operator[](K key) { return this->getOrDefault(key); }

bool remove(K key) {
    if (this->root == nullptr) {
        return false;
    } else {
        return this->remove(this->root, key, [](){}(ConstNodePtr) {});
    }
}

Value getAndRemove(K key) {
    Value result;
    NodeConsumer action = [&](ConstNodePtr node) { result = node-
>value; };

    if (root == nullptr) {
        throw NoSuchMappingException("Invalid key");
    } else {
        if (remove(this->root, key, action)) {
            return result;
        } else {
            throw NoSuchMappingException("Invalid key");
        }
    }
}

Entry getCeilingEntry(K key) const {
```

暨南大学本科实验报告专用纸(附页)

```
if (this->root == nullptr) {
    throw NoSuchMappingException("No ceiling entry in this map");
}

NodePtr node = this->root;

while (node != nullptr) {
    if (key == node->key) {
        return node->entry();
    }

    if (compare(key, node->key)) {
        /* key < node->key */
        if (node->left != nullptr) {
            node = node->left;
        } else {
            return node->entry();
        }
    } else {
        /* key > node->key */
        if (node->right != nullptr) {
            node = node->right;
        } else {
            while (node->direction() == Direction::RIGHT) {
                if (node != nullptr) {
                    node = node->parent;
                } else {
                    throw NoSuchMappingException(
                        "No ceiling entry exists in this map");
                }
            }
            if (node->parent == nullptr) {
                throw NoSuchMappingException("No ceiling entry
exists in this map");
            }
            return node->parent->entry();
        }
    }
}

throw NoSuchMappingException("No ceiling entry in this map");
}

Entry getFloorEntry(K key) const {
```

暨南大学本科实验报告专用纸(附页)

```
    if (this->root == nullptr) {
        throw NoSuchMappingException("No floor entry exists in this
map");
    }

    NodePtr node = this->root;

    while (node != nullptr) {
        if (key == node->key) {
            return node->entry();
        }

        if (compare(key, node->key)) {
            /* key < node->key */
            if (node->left != nullptr) {
                node = node->left;
            } else {
                while (node->direction() == Direction::LEFT) {
                    if (node != nullptr) {
                        node = node->parent;
                    } else {
                        throw NoSuchMappingException("No floor entry
exists in this map");
                    }
                }
                if (node->parent == nullptr) {
                    throw NoSuchMappingException("No floor entry exists
in this map");
                }
                return node->parent->entry();
            }
        } else {
            /* key > node->key */
            if (node->right != nullptr) {
                node = node->right;
            } else {
                return node->entry();
            }
        }
    }

    throw NoSuchMappingException("No floor entry exists in this map");
}
```

暨南大学本科实验报告专用纸(附页)

```
Entry getHigherEntry(K key) {
    if (this->root == nullptr) {
        throw NoSuchMappingException("No higher entry exists in this
map");
    }

    NodePtr node = this->root;

    while (node != nullptr) {
        if (compare(key, node->key)) {
            /* key < node->key */
            if (node->left != nullptr) {
                node = node->left;
            } else {
                return node->entry();
            }
        } else {
            /* key >= node->key */
            if (node->right != nullptr) {
                node = node->right;
            } else {
                while (node->direction() == Direction::RIGHT) {
                    if (node != nullptr) {
                        node = node->parent;
                    } else {
                        throw NoSuchMappingException(
                            "No higher entry exists in this map");
                    }
                }
                if (node->parent == nullptr) {
                    throw NoSuchMappingException("No higher entry
exists in this map");
                }
                return node->parent->entry();
            }
        }
    }

    throw NoSuchMappingException("No higher entry exists in this map");
}

Entry getLowerEntry(K key) const {
    if (this->root == nullptr) {
        throw NoSuchMappingException("No lower entry exists in this
```

暨南大学本科实验报告专用纸(附页)

```
map");
}

NodePtr node = this->root;

while (node != nullptr) {
    if (compare(key, node->key) || key == node->key) {
        /* key <= node->key */
        if (node->left != nullptr) {
            node = node->left;
        } else {
            while (node->direction() == Direction::LEFT) {
                if (node != nullptr) {
                    node = node->parent;
                } else {
                    throw NoSuchMappingException("No lower entry
exists in this map");
                }
            }
            if (node->parent == nullptr) {
                throw NoSuchMappingException("No lower entry exists
in this map");
            }
            return node->parent->entry();
        }
    } else {
        /* key > node->key */
        if (node->right != nullptr) {
            node = node->right;
        } else {
            return node->entry();
        }
    }
}

throw NoSuchMappingException("No lower entry exists in this map");
}

void removeAll(KeyValueFilter filter) {
    std::vector<Key> keys;
    this->inorderTraversal([&](ConstNodePtr node) {
        if (filter(node->key, node->value)) {
            keys.push_back(node->key);
        }
    })
}
```

暨南大学本科实验报告专用纸(附页)

```
});  
    for (const Key &key : keys) {  
        this->remove(key);  
    }  
}  
  
void forEach(KeyValueConsumer action) const {  
    this->inorderTraversal(  
        [&](ConstNodePtr node) { action(node->key, node-  
>value); });  
}  
  
void forEachMut(MutKeyValueConsumer action) {  
    this->inorderTraversal(  
        [&](ConstNodePtr node) { action(node->key, node-  
>value); });  
}  
  
EntryList toEntryList() const {  
    EntryList entryList;  
    this->inorderTraversal(  
        [&](ConstNodePtr node) { entryList.push_back(node-  
>entry()); });  
    return entryList;  
}  
  
private:  
    static void maintainRelationship(ConstNodePtr node) {  
        if (node->left != nullptr) {  
            node->left->parent = node;  
        }  
        if (node->right != nullptr) {  
            node->right->parent = node;  
        }  
    }  
  
    static void swapNode(NodePtr &lhs, NodePtr &rhs) {  
        std::swap(lhs->key, rhs->key);  
        std::swap(lhs->value, rhs->value);  
        std::swap(lhs, rhs);  
    }  
  
    void rotateLeft(ConstNodePtr node) {  
        assert(node != nullptr && node->right != nullptr);  
    }  
}
```

暨南大学本科实验报告专用纸(附页)

```
// clang-format on
NodePtr parent = node->parent;
Direction direction = node->direction();

NodePtr successor = node->right;
node->right = successor->left;
successor->left = node;

maintainRelationship(node);
maintainRelationship(successor);

switch (direction) {
    case Direction::ROOT:
        this->root = successor;
        break;
    case Direction::LEFT:
        parent->left = successor;
        break;
    case Direction::RIGHT:
        parent->right = successor;
        break;
}
successor->parent = parent;
}

void rotateRight(ConstNodePtr node) {
    assert(node != nullptr && node->left != nullptr);
    // clang-format on

    NodePtr parent = node->parent;
    Direction direction = node->direction();

    NodePtr successor = node->left;
    node->left = successor->right;
    successor->right = node;

    maintainRelationship(node);
    maintainRelationship(successor);

    switch (direction) {
        case Direction::ROOT:
            this->root = successor;
            break;
```

暨南大学本科实验报告专用纸(附页)

```
        case Direction::LEFT:
            parent->left = successor;
            break;
        case Direction::RIGHT:
            parent->right = successor;
            break;
    }

    successor->parent = parent;
}

inline void rotateSameDirection(ConstNodePtr node, Direction direction)
{
    assert(direction != Direction::ROOT);
    if (direction == Direction::LEFT) {
        rotateLeft(node);
    } else {
        rotateRight(node);
    }
}

inline void rotateOppositeDirection(ConstNodePtr node, Direction direction) {
    assert(direction != Direction::ROOT);
    if (direction == Direction::LEFT) {
        rotateRight(node);
    } else {
        rotateLeft(node);
    }
}

void maintainAfterInsert(NodePtr node) {
    assert(node != nullptr);

    if (node->isRoot()) {
        // Case 1: Current node is root (RED)
        // No need to fix.
        assert(node->isRed());
        return;
    }

    if (node->parent->isBlack()) {
        // Case 2: Parent is BLACK
        // No need to fix.
    }
}
```

暨南大学本科实验报告专用纸(附页)

```
        return;
    }

    if (node->parent->isRoot()) {
        // clang-format off
        // Case 3: Parent is root and is RED
        // Paint parent to BLACK.
        //      <P>          [P]
        //      |      =====>  |
        //      <N>          <N>
        // p.s.
        //      `<X>' is a RED node;
        //      `[X]' is a BLACK node (or NIL);
        //      `{X}' is either a RED node or a BLACK node;
        // clang-format on
        assert(node->parent->isRed());
        node->parent->color = Node::BLACK;
        return;
    }

    if (node->hasUncle() && node->uncle()->isRed()) {
        // clang-format off
        // Case 4: Both parent and uncle are RED
        // Paint parent and uncle to BLACK;
        // Paint grandparent to RED.
        //      [G]          <G>
        //      / \           / \
        //      <P> <U>      =====> [P] [U]
        //      /           /
        //      <N>          <N>
        // clang-format on
        assert(node->parent->isRed());
        node->parent->color = Node::BLACK;
        node->uncle()->color = Node::BLACK;
        node->grandParent()->color = Node::RED;
        maintainAfterInsert(node->grandParent());
        return;
    }

    if (!node->hasUncle() || node->uncle()->isBlack()) {
        // Case 5 & 6: Parent is RED and Uncle is BLACK
        // p.s. NIL nodes are also considered BLACK
        assert(!node->isRoot());
```

暨南大学本科实验报告专用纸(附页)

```
if (node->direction() != node->parent->direction()) {
    // clang-format off
    // Case 5: Current node is the opposite direction as parent
    //     Step 1. If node is a LEFT child, perform l-rotate to
parent;
    //                         If node is a RIGHT child, perform r-
rotate to parent.
    //     Step 2. Goto Case 6.
    //           [G]                                [G]
    //           / \      rotate(P)      / \
    //           <P> [U]  ======>  <N> [U]
    //           \                               /
    //           <N>                           <P>
    // clang-format on

    // Step 1: Rotation
    NodePtr parent = node->parent;
    if (node->direction() == Direction::LEFT) {
        rotateRight(node->parent);
    } else /* node->direction() == Direction::RIGHT */ {
        rotateLeft(node->parent);
    }
    node = parent;
    // Step 2: vvv
}

// clang-format off
// Case 6: Current node is the same direction as parent
//     Step 1. If node is a LEFT child, perform r-rotate to
grandparent;
//                         If node is a RIGHT child, perform l-
rotate to grandparent.
//     Step 2. Paint parent (before rotate) to BLACK;
//             Paint grandparent (before rotate) to
RED.
[P]          //           [G]                                <P>
repaint      //           / \      rotate(G)      / \
              / \          <P> [U]  ======>  <N> [G]  =====>  <N>
<G>          //           /           \           <N>
\           //           <N>
```

暨南大学本科实验报告专用纸(附页)

```
[U] [U]
// clang-format on

assert(node->grandParent() != nullptr);

// Step 1
if (node->parent->direction() == Direction::LEFT) {
    rotateRight(node->grandParent());
} else {
    rotateLeft(node->grandParent());
}

// Step 2
node->parent->color = Node::BLACK;
node->sibling()->color = Node::RED;

return;
}

NodePtr getNodeOrProvide(NodePtr &node, K key, NodeProvider provide) {
    assert(node != nullptr);

    if (key == node->key) {
        return node;
    }

    assert(key != node->key);

    NodePtr result;

    if (compare(key, node->key)) {
        /* key < node->key */
        if (node->left == nullptr) {
            result = node->left = provide();
            node->left->parent = node;
            maintainAfterInsert(node->left);
            this->count += 1;
        } else {
            result = getNodeOrProvide(node->left, key, provide);
        }
    } else {
        /* key > node->key */
        if (node->right == nullptr) {
```

暨南大学本科实验报告专用纸(附页)

```
        result = node->right = provide();
        node->right->parent = node;
        maintainAfterInsert(node->right);
        this->count += 1;
    } else {
        result = getNodeOrProvide(node->right, key, provide);
    }
}

return result;
}

NodePtr getNode(ConstNodePtr node, K key) const {
    assert(node != nullptr);

    if (key == node->key) {
        return node;
    }

    if (compare(key, node->key)) {
        /* key < node->key */
        return node->left == nullptr ? nullptr : getNode(node->left,
key);
    } else {
        /* key > node->key */
        return node->right == nullptr ? nullptr : getNode(node->right,
key);
    }
}

void insert(NodePtr &node, K key, V value, bool replace = true) {
    assert(node != nullptr);

    if (key == node->key) {
        if (replace) {
            node->value = value;
        }
        return;
    }

    assert(key != node->key);

    if (compare(key, node->key)) {
        /* key < node->key */

```

暨南大学本科实验报告专用纸(附页)

```
if (node->left == nullptr) {
    node->left = Node::from(key, value);
    node->left->parent = node;
    maintainAfterInsert(node->left);
    this->count += 1;
} else {
    insert(node->left, key, value, replace);
}
} else {
    /* key > node->key */
    if (node->right == nullptr) {
        node->right = Node::from(key, value);
        node->right->parent = node;
        maintainAfterInsert(node->right);
        this->count += 1;
    } else {
        insert(node->right, key, value, replace);
    }
}
}

void maintainAfterRemove(ConstNodePtr node) {
    if (node->isRoot()) {
        return;
    }

    assert(node->isBlack() && node->hasSibling());

    Direction direction = node->direction();

    NodePtr sibling = node->sibling();
    if (sibling->isRed()) {
        ConstNodePtr parent = node->parent;
        assert(parent != nullptr && parent->isBlack());
        assert(sibling->left != nullptr && sibling->left->isBlack());
        assert(sibling->right != nullptr && sibling->right->isBlack());
        rotateSameDirection(node->parent, direction);
        sibling->color = Node::BLACK;
        parent->color = Node::RED;
        sibling = node->sibling();
    }

    NodePtr closeNephew =
        direction == Direction::LEFT ? sibling->left : sibling->
```

暨南大学本科实验报告专用纸(附页)

```
>right;
    NodePtr distantNephew =
        direction == Direction::LEFT ? sibling->right : sibling-
>left;

    bool closeNephewIsBlack = closeNephew == nullptr || closeNephew-
>isBlack();
    bool distantNephewIsBlack =
        distantNephew == nullptr || distantNephew->isBlack();

    assert(sibling->isBlack());

    if (closeNephewIsBlack && distantNephewIsBlack) {
        if (node->parent->isRed()) {
            // clang-format off
            // Case 2: Sibling and nephews are BLACK, parent is RED
            // Swap the color of P and S
            //           <P>                               [P]
            //           / \                               / \
            // [N] [S] =====> [N] <S>           / \
            //           / \                               [C] [D]
            //           [C] [D]                         [C] [D]
            // clang-format on
            sibling->color = Node::RED;
            node->parent->color = Node::BLACK;
            return;
        } else {
            // clang-format off
            // Case 3: Sibling, parent and nephews are all black
            // Step 1. Paint S to RED
            // Step 2. Recursively maintain P
            //           [P]                               [P]
            //           / \                               / \
            // [N] [S] =====> [N] <S>           / \
            //           / \                               [C] [D]
            //           [C] [D]                         [C] [D]
            // clang-format on
            sibling->color = Node::RED;
            maintainAfterRemove(node->parent);
            return;
        }
    } else {
        if (closeNephew != nullptr && closeNephew->isRed()) {
            // Step 1
```

暨南大学本科实验报告专用纸(附页)

```
rotateOppositeDirection(sibling, direction);
// Step 2
closeNephew->color = Node::BLACK;
sibling->color = Node::RED;
// Update sibling and nephews after rotation
sibling = node->sibling();
closeNephew =
    direction == Direction::LEFT ? sibling->left :
sibling->right;
distantNephew =
    direction == Direction::LEFT ? sibling->right :
sibling->left;
// Step 3: vvv
}

assert(closeNephew == nullptr || closeNephew->isBlack());
assert(distantNephew->isRed());
// Step 1
rotateSameDirection(node->parent, direction);
// Step 2
sibling->color = node->parent->color;
node->parent->color = Node::BLACK;
if (distantNephew != nullptr) {
    distantNephew->color = Node::BLACK;
}
return;
}
}

bool remove(NodePtr node, K key, NodeConsumer action) {
    assert(node != nullptr);

    if (key != node->key) {
        if (compare(key, node->key)) {
            /* key < node->key */
            NodePtr &left = node->left;
            if (left != nullptr && remove(left, key, action)) {
                maintainRelationship(node);
                return true;
            } else {
                return false;
            }
        } else {
            /* key > node->key */
```

暨南大学本科实验报告专用纸(附页)

```
        NodePtr &right = node->right;
        if (right != nullptr && remove(right, key, action)) {
            maintainRelationship(node);
            return true;
        } else {
            return false;
        }
    }
}

assert(key == node->key);
action(node);

if (this->size() == 1) {
    // Current node is the only node of the tree
    this->clear();
    return true;
}

if (node->left != nullptr && node->right != nullptr) {
    // clang-format off
    // Case 1: If the node is strictly internal
    // Step 1. Find the successor S with the smallest key
    //         and its parent P on the right subtree.
    // Step 2. Swap the data (key and value) of S and N,
    //         S is the node that will be deleted in
place of N.
    //     Step 3. N = S, goto Case 2, 3
    //           |
    //           N           S
    //           / \           / \
    //           L   ..      swap(N, S)   L   ..
    //           |           ======>   |
    //           P           N           / \
    //           / \           ..           / \
    //           S   ..       P           N   ..
    // clang-format on

    // Step 1
    NodePtr successor = node->right;
    NodePtr parent = node;
    while (successor->left != nullptr) {
        parent = successor;
        successor = parent->left;
    }
}
```

暨南大学本科实验报告专用纸(附页)

```
        }
        // Step 2
        swapNode(node, successor);
        maintainRelationship(parent);
        // Step 3: vvv
    }

    if (node->isLeaf()) {
        // Current node must not be the root
        assert(node->parent != nullptr);

        // Case 2: Current node is a leaf
        //     Step 1. Unlink and remove it.
        //     Step 2. If N is BLACK, maintain N;
        //             If N is RED, do nothing.

        // The maintain operation won't change the node itself,
        // so we can perform maintain operation before unlink the
node.

        if (node->isBlack()) {
            maintainAfterRemove(node);
        }
        if (node->direction() == Direction::LEFT) {
            node->parent->left = nullptr;
        } else /* node->direction() == Direction::RIGHT */ {
            node->parent->right = nullptr;
        }
    } else /* !node->isLeaf() */ {
        assert(node->left == nullptr || node->right == nullptr);
        // Case 3: Current node has a single left or right child
        //     Step 1. Replace N with its child
        //     Step 2. If N is BLACK, maintain N
        NodePtr parent = node->parent;
        NodePtr replacement = (node->left != nullptr ? node->left :
node->right);
        switch (node->direction()) {
            case Direction::ROOT:
                this->root = replacement;
                break;
            case Direction::LEFT:
                parent->left = replacement;
                break;
            case Direction::RIGHT:
                parent->right = replacement;
```

暨南大学本科实验报告专用纸(附页)

```
        break;
    }

    if (!node->isRoot()) {
        replacement->parent = parent;
    }

    if (node->isBlack()) {
        if (replacement->isRed()) {
            replacement->color = Node::BLACK;
        } else {
            maintainAfterRemove(replacement);
        }
    }
}

this->count -= 1;
return true;
}

void inorderTraversal(NodeConsumer action) const {
    if (this->root == nullptr) {
        return;
    }

    std::stack<NodePtr> stack;
    NodePtr node = this->root;

    while (node != nullptr || !stack.empty()) {
        while (node != nullptr) {
            stack.push(node);
            node = node->left;
        }
        if (!stack.empty()) {
            node = stack.top();
            stack.pop();
            action(node);
            node = node->right;
        }
    }
};

#endif // RBTREE_MAP_HPP
```