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$\mathcal{O}(1)$

4. 程序代码

4.1. deque.h

```
// #define _PRIVATE_DEBUG
#ifndef PRVLIBCPP_DEQUE_HPP
#define PRVLIBCPP_DEQUE_HPP

#include <map>
#include <vector>

#ifdef _PRIVATE_DEBUG
#include <iostream>
#endif

namespace myDS
{
    template<typename VALUE_TYPE>
    class deque{
    protected:

    private:
        using coddinate = std::pair<std::int32_t,std::int32_t>;

        // < L : 0 , R : 1 >
        std::vector<std::vector<VALUE_TYPE>> _indexs;

        std::int32_t _size = 0;
        std::int32_t _L = 1;
        std::int32_t _R = -1;

        VALUE_TYPE & get(coddinate p) {
            return _indexs[p.first][p.second];
        }

        coddinate index2cod(std::int32_t p) {
            if(p+_L > 0) return coddinate(1,p+_L-1);
            else return coddinate(0,-p-_L);
        }

        void _reDistribute() {
            if(_L * _R <= 0) return;
            if(abs(_L - _R) + 1 < std::min(abs(_L),abs(_R))) {
                if(_L > 0) { // < --- 0 : 0 --- L -- R --- >
                    std::vector<VALUE_TYPE> N;
                    for(int i = _L-1;i <= _R;i ++)
N.push_back(_indexs[1][i]);
                    _indexs[1] = N;
                    _L = 1;
                    _R = N.size() - 1;
                } else { // < --- L(<0) -- R(<0) --- 0 : 0 --- >
                    std::vector<VALUE_TYPE> N;
```

```
                    for(int i = -_R-1;i <= -_L;i ++)
N.push_back(_indexs[0][i]);
                    _indexs[0] = N;
                    _L = -N.size()+1;
                    _R = -1;
                }
            } else return;
        }

    public:
        deque(){
            _indexs.push_back(std::vector<VALUE_TYPE>());
            _indexs.push_back(std::vector<VALUE_TYPE>());
        }

        void push_back(VALUE_TYPE t) {
            _R ++;
            if(_R >= 0) {
                _indexs[1].push_back(t);
            } else {
                _indexs[0][_R-1] = t;
                _reDistribute();
            }
        }

        void push_frount(VALUE_TYPE t) {
                _L --;
            if(_L <= 0) {
                _indexs[0].push_back(t);
            } else {
                _indexs[1][_L-1] = t;
                _reDistribute();
            }
        }

        VALUE_TYPE pop_back() {
            if(!this->size()) throw std::out_of_range("Pop from empty
deque");
            VALUE_TYPE t ;
            if(_R >= 0) {
                t = _indexs[1].back();
                _indexs[1].pop_back();
                _R --;
            } else {
                t = _indexs[0][-_R-1];
                _R --;
                _reDistribute();
            }
            return t;
        }

```

```
97          VALUE_TYPE pop_frount() {
98              if(!this->size()) throw std::out_of_range("Pop from empty deque");
99              VALUE_TYPE t;
100             if(_L <= 0) {
101                 t = _indexs[0].back();
102                 _indexs[0].pop_back();
103                 _L ++;
104             } else {
105                 t = _indexs[1][_L-1];
106                 _L ++;
107                 _reDistribute();
108             }
109             return t;
110         }
111
112         void clear() {
113             _indexs[0].clear();
114             _indexs[1].clear();
115             _L = 1;
116             _R = -1;
117         }
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         }
```

```
    };
}
#endif
```

4.2. `_PRIV_TEST.cpp`

```cpp
#define DS_TOBE_TEST deque

#define _PRIVATE_DEBUG
// #define __DETIL_DEBUG_OUTPUT

#include "Dev\04\deque.h"

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

using namespace std;

using TBT = myDS::deque<int>;

void accuracyTest() {//结构正确性测试

    TBT tc = TBT();
    for(;;)
    {
        string op;
        cout << ">>>";
        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 == "ob")//pop_back
        {
            cout << tc.pop_back() << "\n";
        } else if(op == "of")//pop_frount
        {
```

```
            cout << tc.pop_frount() << "\n";
        } else if(op == "at")//随机访问
        {
            int p;
            cin >> p;
            cout << tc[p] << "\n";
        } else if(op == "at")//随机访问
        {
            int p;
            cin >> p;
            cout << tc[p] << "\n";
        } else if(op == "of")//pop_frount
        {

        } 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;
        //     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 memLeakTest1() {//内存泄漏测试
    TBT tc = TBT();
    for(;;){
        tc.push_back(1);
        tc.push_back(1);
        tc.push_back(1);
        tc.push_back(1);
        tc.clear();
    }
}

void memLeakTest2() {//内存泄漏测试
    TBT tc = TBT();
    for(;;){
        tc.push_back(1);
        tc.pop_frount();
    }
}

void speedTest()
{
    TBT tc = TBT();
    int begin = clock();
    int N = 1e8;
    for(int i = 0;i < sqrt(N/2);i ++)
    {
        for(int j = 0;j < sqrt(N/2);j ++)
        {
            tc.push_back(i);
        }
        for(int j = 0;j < sqrt(N/2);j ++)
        {
            tc.pop_frount();
```

```
        }
    }
    cout << "myDS::deque push_back then pop_frount sqrt(5000000) elements for sqrt(5000000) times 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();
    // memLeakTest1();
    // memLeakTest2();
    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：

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

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

```
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
```

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。

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。