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$\mathbb{O}(\log_2 n)$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`:$\mathbb{O}(1)$

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

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。

4. 程序代码

4.1. dataBlock.hpp

```cpp
// #define _PRIVATE_DEBUG
#ifndef DATA_BLOCK_HPP
#define DATA_BLOCK_HPP

#include <vector>
#include <map>

#define _PRIVATE_DEBUG

#ifdef _PRIVATE_DEBUG
#include <iostream>
#endif

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

    private:
        class _iterator
        {
        private:
            VALUE_TYPE *_ptr;
            std::pair<std::size_t,std::size_t> loc;
            dataBlock<VALUE_TYPE> * _upper_pointer;

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

            _iterator( myDS::dataBlock<VALUE_TYPE> *_upper,std::pair<std::size_t,std::size_t> _loc,__iter_dest_type _d)
            {
                _upper_pointer = _upper;
                loc = _loc;
                _ptr = &_upper_pointer->_indexs[loc.first][loc.second];
                _iter_dest = _d;
            }

            VALUE_TYPE & operator*()
            {
                return (*_ptr);
            }

            VALUE_TYPE *operator->()
            {
                return _ptr;
            }

             myDS::dataBlock<VALUE_TYPE>::_iterator operator++() {
                if(_iter_dest == front)
                {
                    loc = _upper_pointer->nextPII(loc);
                }
                else
                {
                    loc = _upper_pointer->prevPII(loc);
                }
                _ptr = &_upper_pointer->_indexs[loc.first][loc.second];
                return myDS::dataBlock<VALUE_TYPE>::_iterator(_upper_pointer,loc,_iter_dest);
            }

             myDS::dataBlock<VALUE_TYPE>::_iterator operator++(int) {
                 myDS::dataBlock<VALUE_TYPE>::_iterator old = *this;
                if(_iter_dest == front)
                {
                    loc = _upper_pointer->nextPII(loc);
                }
                else
                {
                    loc = _upper_pointer->prevPII(loc);
                }
                _ptr = &_upper_pointer->_indexs[loc.first][loc.second];
                return old;
            }

            bool operator==( myDS::dataBlock<VALUE_TYPE>::_iterator _b) {
                return _ptr == _b._ptr;
            }

            bool operator!=( myDS::dataBlock<VALUE_TYPE>::_iterator _b) {
                return _ptr != _b._ptr;
            }
        };

        std::vector<VALUE_TYPE *> _indexs;
        std::pair<std::size_t,std::size_t> _cap = {0,0};
        std::size_t consMEX = 1;
        std::size_t _size = 0;

        void _expension()
        {
            VALUE_TYPE *temp = new VALUE_TYPE[consMEX];
            _indexs.push_back(temp);
            consMEX *= 2;
            _cap.first++;
            _cap.second = 0;
        }

        std::size_t getMEX(std::int32_t p)
        {
            if(p <= 0) return p+1;
            return (1 << (p-1));
        }

        std::pair<std::size_t,std::size_t> nextPII(std::pair<std::size_t,std::size_t> p)
        {
            p.second++;
            if(p.second >= getMEX(p.first))
            {
                p.first++;
                p.second = 0;
            }
            return p;
        }

        std::pair<std::size_t,std::size_t> prevPII(std::pair<std::size_t,std::size_t> p)
        {
#ifdef __DETIL_DEBUG_OUTPUT
            std::cout << "{" << p.first << "," << p.second << "}'s prev is";
#endif

            std::int32_t tmp = p.second;
            tmp --;
            if(tmp < 0)
            {
                p.first--;
                p.second = getMEX(p.first) - 1;
            } else p.second --;
#ifdef __DETIL_DEBUG_OUTPUT
            std::cout << "{" << p.first << "," << p.second << "}\n";
#endif

            return p;
        }

    public:
        dataBlock(){
```

```cpp
            VALUE_TYPE *tmp = new VALUE_TYPE[1];
            _indexs.push_back(tmp);
        }

        ~dataBlock(){
            clear();
            delete [] (_indexs[0]);
        }

        void push_back(VALUE_TYPE t) {
            if(_cap.second >= getMEX(_cap.first)) {
                _expension();
            }
            _indexs[_cap.first][_cap.second] = t;
            _cap.second++;
            _size++;
        }

        void clear() {
            for(auto x:_indexs) delete [] x;
            _indexs.clear();
            VALUE_TYPE *tmp = new VALUE_TYPE[1];
            _indexs.push_back(tmp);
            consMEX = 1;
            _size = 0;
            _cap = {0,0};
        }

        std::size_t size() {
            return _size;
        }

         myDS::dataBlock<VALUE_TYPE>::_iterator begin() {
            return  myDS::dataBlock<VALUE_TYPE>::_iterator(this,{0,0}, myDS::dataBlock<VALUE_TYPE>::_iterator::front);
        }

         myDS::dataBlock<VALUE_TYPE>::_iterator rbegin() {
            return myDS::dataBlock<VALUE_TYPE>::_iterator(this,prevPII(_cap)), myDS::dataBlock<VALUE_TYPE>::_iterator::back);
        }

         myDS::dataBlock<VALUE_TYPE>::_iterator end() {
            return myDS::dataBlock<VALUE_TYPE>::_iterator(this,nextPII(prevPII(_cap)), myDS::dataBlock<VALUE_TYPE>::_iterator::front);
        }

         myDS::dataBlock<VALUE_TYPE>::_iterator rend() {
            return myDS::dataBlock<VALUE_TYPE>::_iterator(this,prevPII({0,0}), myDS::dataBlock<VALUE_TYPE>::_iterator::back);
        }

#ifdef _PRIVATE_DEBUG
        void innerPrint() {
            std::pair<std::size_t,std::size_t> p = {0,0};
            while(p.first <= _cap.first) {
                if(p.second == 0) std::cout << "\nBlock : [" << p.first << "] at:" << _indexs[p.first] << "\n";
                std::cout << _indexs[p.first][p.second] << " ";
                p = nextPII(p);
            }
            std::cout << "\n";
        }
#endif

        VALUE_TYPE & operator[](std::size_t p) {
            if(p == 0) return _indexs[0][0];
            std::int32_t onord = 0;
            std::size_t tmp = p;
            while(tmp) {
                tmp >>= 1;
                onord++;
            }
#ifdef __DETIL_DEBUG_OUTPUT
            std::cout << "onord:" << onord << " p:" << p << " GETMEX :"<< getMEX(onord) << " index:{" << onord << "," << p - getMEX(onord) << "}\n";
#endif
            return _indexs[onord][p - getMEX(onord)];

        }
    };
}
#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>
```

```
#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;
```

```cpp
        //     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);
        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：

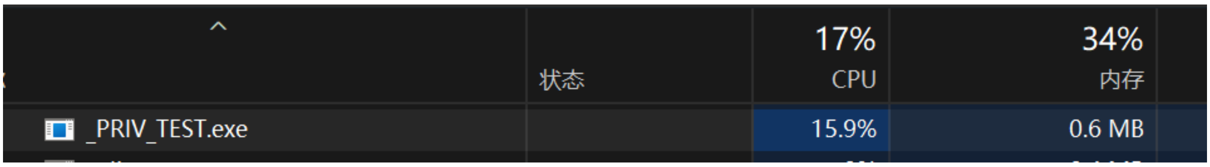
```
  pb 1
  pb 2
  pb 3
  pb 4
  iterF
  iterB
  clr
  pb 0
  pb 3
  pb 1
  pb 2
  pb 3
  pb 4
  iterF
  iterB
  mv 0 3
  iterF
  pb 1
  pb 2
  pb 3
  pb 4
  iterF

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
```

```
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
```

```
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	17% CPU	34% 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。