数据结构课程项目

Data Structure 2020

2020 年 10 月 28 日

1 问题描述

设计一个合理的数据结构，使用尽量少的时间复杂度和空间复杂度，支持一些常见操作。此外，尽可能多的支持扩展操作。

设计对比实验和测试数据，在不同类型的操作上，比较你的设计与常用数据结构的性能差异，并进行分析。

允许使用 STL，但无必要不应使用 STL 以外的库。如有特殊需求，需事先询问助教。

在 OJ(http://10.177.31.231/) 上进行公开测试，并（可选地）提交自行构造的数据。

最终需要提交代码，测试数据（源数据，构造器，或获取途径），及完善的书面报告。每位同学都将在学期末单独向助教进行实验报告。

鼓励大家在最后一节课上进行课堂展示。

2 基础问题

在最低限度，你设计的数据结构应当是一个线性表（不妨称其 LinearTable），允许存储 int 整型元素，并支持以下操作：

1. 构造与析构，即能构造一个空表，并在其生命周期结束时正确释放所使用的内存。LinearTable(), ~LinearTable()
2. 表首单个元素查询 front()
3. 表首单个元素插入 push_front()
4. 表首单个元素删除 pop_front()
5. 表尾单个元素查询 back()
6. 表尾单个元素插入 push_back()
7. 表尾单个元素删除 pop_back()
8. 查询元素数量 size()
9. 判断是否为空 empty()
10. 随机位置访问与修改 operator[]
11. 交换 swap(B)
12. 清空数据 clear()

事实上，上述即是一个简化的 STL deque<int>。你的所有代码应被放置到 namespace DS 中。显然你设计的数据结构应保证正确性。此外，你还应当动态使用内存以节省空间，并尽量减少每种操作的耗时。如多种优化之间出现冲突，还应进行判断和取舍。

3 扩展-对基础操作的扩展

优化你设计的数据结构，使其基础操作（通用情况或在特定情况下）有更好的表现。请说明你实现的优越之处或困难之处，如果仅仅只是朴素地进行实现，那么不一定会视为你的额外贡献。

1. 构造，析构
   (a) LinearTable(n, val) 尝试构造长度 n，且初始值为 val 的线性表，LinearTable(first, last) 尝试从指针对/迭代器对 (first, last) 进行构造，等等。注意时空效率。
   (b) 对复杂数据类型进行析构时，可能需要递归调用析构函数以避免内存泄漏。而对简单数据类型进行析构时，可能可以批量处理以提升效率。
2. 实现随机访问迭代器。
3. 分类讨论并优化批量插入删除，如：
   (a) push_back(n, val)
   (b) push_front(first, last)
   (c) pop_front(pop_size)
   (d) insert(iterator, val) 在指定迭代器前插入元素
   (e) insert(iterator, n, val)
   (f) insert(iterator, first, last)
   (g) erase(iterator)
   (h) erase(first, last)

   等等
4. 设计常量对象并阻止各种操作对常量对象不必要的修改。
5. 实现深拷贝 deep_copy()，分析在你的设计上，深拷贝与浅拷贝的区别
6. 等等，可自行发挥。

4 扩展-区间最值查询

1. 增加额外的索引，以支持区间最值查询
2. 形式上，对于操作 max(first, last)，返回 [first, last) 区间内的最大值
3. 简化的操作为 max()，返回线性表内所有元素的最大值
4. 对于带修改和不带修改的情况，可以分别使用不同的索引以提升效率
5. 如你设计的索引与之前要求的操作有所冲突，可以作出取舍。
6. 同样可自由进行扩展，如 max(first, last, k) 返回 [first, last) 区间内的第 k 大值，max(first, last, lbound, rbound)，返回区间内第 [lbound, rbound) 大值，以及完成 min()，等等。
7. 鉴于这是一个被长久研究的问题，再次提醒不要拷贝他人的代码。

5 扩展-排序

1. 增加额外的索引，以支持对序列的排序 sort()
2. 可以采用多种方式实现对 sort 的扩展，如在插入新元素后使其依然有序 sorted_insert()，合并两个有序表 sorted_merge()，等等。
3. 对于不同情况，请分别说明设计的优越之处

6　扩展-复杂度分析

对你设计的数据结构的每个操作均进行时空复杂度分析。

通常而言，你需要同时考虑最坏复杂度和平均复杂度，如有必要还应举例说明。

如果大 O 表示法无法展现常数上的优越之处，则需要寻找其他更精细的比较方式。

此外，为了凸显你的设计的精妙之处，还应设计相应的对比实验以展现其价值。注意控制变量。

7 扩展-实验与测试数据

在 OJ(http://10.177.31.231/) 上预先提供了对基础操作的测试数据及实验代码，注意这些数据没有专门精心设计。测试代码以模板的形式提供，提交文件只需要补全其中 namespace DS 中的内容即可。

为了凸显你设计的优越之处，尤其是支持额外的操作时，应当寻找额外的测试数据及对比实验。测试数据与实验代码应完整提交。如构造的测试数据较大，则可以改为提交数据生成器。注意如需随机化请保留随机化种子（以及其他必需信息），以保证可以重新构造相同的数据。

如有意愿，鼓励大家在课堂展示前提前提交精心构造的测试代码（不妨称其为 hack 数据）。提交 hack 数据，并在 OJ 上被他人所使用，有助于从对比中展现你设计的数据结构的优越性。此外，使用 hack 数据评估你的设计，有助于检验其完备性，并提高设计的说服力。

提交 hack 数据前，请与助教单独交流，给出清晰的文字描述，说明设计思路，并确认想法合理后，再将对应的测试代码以模板的形式单独发给助教。经过检查无误后，助教会将其添加到 OJ 中。具体的数据不会公开，但会公开文字描述。

hack 数据不应包含过多的输出内容，也不要出现行末空格。如输出内容过多，请自行哈希。

受 OJ 功能所限，对每个数据提交不同代码的行为不会受到进制。但很显然的，如果在数据 A 上使用代码 X，在数据 B 上使用代码 Y，可以简单地构造数据 A+B，使得代码 X、代码 Y 与代码 X+Y 均无法通过。精妙的设计会比无脑的堆砌更有用。

此外，OJ 评测性能有限，因此请不要进行反复无意义的提交，尤其是单纯的刷性能刷榜。如因个人原因导致其他人无法评测，将会适度惩罚。

8 评价指标

最终需要在 elearning 上提交代码，自行构造的测试数据（源数据，构造器），及完善的实验报告。根据设计数据结构的理论性能，精巧程度，新颖程度，以及实验结果进行打分。扩展可以任意选做，使用精妙的设计支持越多的扩展操作，将会获得加分。如助教认为你自行设计的扩展操作有意义且实现精妙，那么也会获得加分。

在 OJ 的基础数据和 hack 数据提交你的代码，评测时运行时间和空间消耗将会成为有力证据。虽然你在测试时可以在不同数据上使用不同的代码，但最终提交时应使用同一份代码进行提交，并在实验报告中注明每个数据对应的提交 ID。

每位同学都将在学期末单独向助教进行面谈。面谈时请基于实验报告清晰准确地讲解你设计的优越之处，这会极大的影响你的得分。必要时，会要求具体讲解代码实现，此时优秀的注释和合理的组织会极大降低阐述的难度。

在最后一节课上，会组织进行课堂展示，鼓励大家积极参与，会根据展示获得加分。

ddl 为学期末，具体时间待定。

9 Test.cpp 示例

```
//PREPEND BEGIN
#include <bits/stdc++.h>
//PREPEND END

//TEMPLATE BEGIN
namespace DS {
    class LinearTable {
    private:
                //TODO
    public:
        LinearTable();
        ~LinearTable();
                //TODO
    };
        //TODO
}
//TEMPLATE END

//APPEND BEGIN
int main() {
    DS::LinearTable testLinearTable;
    std::cout << (testLinearTable.empty() ? 1 : 0) << std::endl;
    int n;
    std::cin >> n;
    testLinearTable.push_back(n);
    testLinearTable.push_front (2);
    testLinearTable.push_front (3);
    testLinearTable.push_back(4);
    std::cout << testLinearTable.size() << ' ' << testLinearTable.front()
    testLinearTable.pop_back();
    testLinearTable.pop_front();
```

```
    std::cout << testLinearTable.size() << ' ' << testLinearTable.front()
    testLinearTable[0] = 5;

    DS::LinearTable testLinearTable2;
    testLinearTable2.push_back(6);
    testLinearTable2.push_front (7);
    testLinearTable2.push_back(8);
    testLinearTable2.push_front (9);
    testLinearTable2[1] = 10;
    std::cout << (testLinearTable2.empty() ? 1 : 0) << std::endl;

    testLinearTable.swap(testLinearTable2);

    int len = testLinearTable.size();
    for (int x = 0; x < len; x++)
        std::cout << testLinearTable[x] << ' ';
    std::cout << std::endl;
    len = testLinearTable2.size();
    for (int x = 0; x < len; x++)
        std::cout << testLinearTable2[x] << ' ';
    std::cout << std::endl;

    return 0;
}
//APPEND END
```