堆与优先队列

基础知识

完全二叉树内容的回顾

- 父子节点的编号存在可计算的关系 因此不需要存储边的信息
- 可以用连续空间存储

堆的概念

- 一种基于完全二叉树的结构
- 大顶堆和小顶堆
  - 大顶堆 —— 任意的三元组，父节点都大于两个子节点
  - 小顶堆 —— 任意的三元组，父节点都小于两个子节点
  - 根节点为最大值或最小值
- 堆适合维护集合的最值

堆的基本操作（以大顶堆为例）

- 尾部插入
  - 比父节点大就和父节点交换 递归向上调整
  - 这个过程称为SIFT-UP
- 头部弹出
  - 用最后一个元素（叶子结点）补位 递归向下调整
  - 这个过程称为SIFT-DOWN

课上的堆代码实现1

```cpp
#define MAX_N 1000
int data[MAX_N + 5], cnt = 0;

int top() { return data[0]; }
int size() { return cnt; }

void shift_up(int ind) {
    while (ind && data[(ind - 1) / 2] < data[ind]) {
        swap(data[(ind - 1) / 2], data[ind]);
        ind = (ind - 1) / 2;
    }
    return ;
}

void output(int n) {
    printf("heap : ");
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
    printf("\n");
    return ;
}

void shift_down(int ind) {
    int n = cnt - 1;
    while (ind * 2 + 1 <= n) {
        int temp = ind;
        if (data[temp] < data[ind * 2 + 1]) temp = ind * 2 + 1;
        if (ind * 2 + 2 <= n && data[temp] < data[ind * 2 + 2]) temp = ind * 2
+ 2;
        if (temp == ind) break;
        swap(data[temp], data[ind]);
        ind = temp;
    }
    return ;
}

void push(int x) {
    data[cnt++] = x;
    shift_up(cnt - 1);
    return ;
}

void pop() {
    if (size() == 0) return ;
    swap(data[0], data[cnt - 1]);
    cnt -= 1;
    shift_down(0);
    return ;
}

int main() {
    int op, val;
    int max_n = 0;
    while (cin >> op) {
```

```
        switch (op) {
            case 0: {
                cin >> val;
                printf("push %d to heap\n", val);
                push(val);
            } break;
            case 1: {
                printf("pop %d from heap\n", top());
                pop();
            } break;
            case 2: {
                output(max_n);
            } break;
        }
        max_n = max(cnt, max_n);
        output(cnt);
    }
    return 0;
}
```

课上的堆实现代码2

```
template<typename T>
class Heap : public vector<T> {
public:
    template<typename Func_T>
    Heap(Func_T cmp) : cmp(cmp) {}

    void push(const T &a) {
        this->push_back(a);
        push_heap(this->begin(), this->end(), cmp);
        return;
    }

    void pop() {
        pop_heap(this->begin(), this->end(), cmp);
        this->pop_back();
        return;
    }

    T &top() { return this->at(0); }

private:
    function<bool(T, T)> cmp;

};
```

习题课

第一部分 堆的基础应用

最小的K个数

```cpp
class Solution {
public:
    vector<int> getLeastNumbers(vector<int> &arr, int k) {
        Heap<int> h{less<int>()};
        for (auto x: arr) {
            h.push(x);
            if (h.size() > k) { h.pop(); }
        }
        return h;
    }
};
```

最后一块石头的重量

```cpp
class Solution {
public:
    int lastStoneWeight(vector<int> &stones) {
        Heap<int> h{less<int>()};
        for (auto x: stones) {
            h.push(x);
        }
        while (h.size() > 1) {
            int y = h.top();
            h.pop();
            int x = h.top();
            h.pop();
            if (x == y) { continue; }
            h.push(y - x);
        }

        if (h.size() == 0) { return 0; }
        else { return h.top(); }
    }
};
```

数据流中的第K大元素

```cpp
class KthLargest {
public:
    Heap<int> h{greater<int>()};
    int k;

    KthLargest(int k, vector<int> &nums) : k(k) {
        for (auto x: nums) {
            add(x);
        }
    }

    int add(int val) {
        h.push(val);
        if (h.size() > k) { h.pop(); }
```

```
        return h.top();
    }
};
```

查找和最小的K对数字

```
struct CMP {
    bool operator()(vector<int> a, vector<int> b) {
        return a[0] + a[1] < b[0] + b[1];
    }
};


class Solution {
public:
    vector<vector<int>> kSmallestPairs(vector<int> &nums1, vector<int> &nums2, int k) {
        CMP less_than;
        Heap<vector<int>> h{less_than};
        vector<int> temp(2);
        for (auto x: nums1) {
            for (auto y: nums2) {
                temp[0] = x, temp[1] = y;
                if (h.size() < k || less_than(temp, h.top())) {
                    h.push(temp);
                    if (h.size() > k) { h.pop(); }
                }
                else { break; }
            }
        }
        return h;
    }
};
```

数组中的第K个最大元素

```
class Solution {
public:
    int findKthLargest(vector<int> &nums, int k) {
        Heap<int> h{greater<int>()};
        for (auto x: nums) {
            h.push(x);
            if (h.size() > k) { h.pop(); }
        }
        return h.top();
    }
};
```

第二部分 堆的进阶应用

设计推特

无

前K个高频单词

```cpp
class Solution {
public:
    vector<string> topKFrequent(vector<string> &words, int k) {
        unordered_map<string, int> freq;
        for (auto x: words) { freq[x] += 1; }
        auto cmp = [&freq](string a, string b) -> bool {
            if (freq[a] - freq[b]) { return freq[a] > freq[b]; }
            return a < b;
        };
        Heap<string> h{cmp};
        for (auto x: freq) {
            h.push(x.first);
            if (h.size() > k) { h.pop(); }
        }
        sort(h.begin(), h.end(), cmp);
        return h;
    }
};

```

连续中值

数据流的中位数

使用两个堆，前半部分为大顶堆，后半部分为小顶堆，维护中位性质即可。中位性质的维护思路很接近之前做过的“前中后队列”一题。

```cpp
template<typename T>
class Heap : public vector<T> {
public:
    template<typename Func_T>
    Heap(Func_T cmp) : cmp(cmp) {}

    void push(const T &a) {
        this->push_back(a);
        push_heap(this->begin(), this->end(), cmp);
        return;
    }

    void pop() {
        pop_heap(this->begin(), this->end(), cmp);
        this->pop_back();
        return;
    }

    T &top() { return this->at(0); }

private:
```

```
    function<bool(T, T)> cmp;
};

class MedianFinder {
public:
    Heap<int> h1, h2;

    MedianFinder() : h1{less<int>()}, h2{greater<int>()} {}

    void addNum(int num) {
        if (h1.size() == 0 || num <= h1.top()) { h1.push(num); }
        else { h2.push(num); }

        if (h2.size() > h1.size()) {
            h1.push(h2.top());
            h2.pop();
        }
        if (h1.size() == h2.size() + 2) {
            h2.push(h1.top());
            h1.pop();
        }
        return;
    }

    double findMedian() {
        int n = h1.size() + h2.size();
        if (n % 2 == 1) { return h1.top(); }
        return (h1.top() + h2.top()) / 2.0;
    }
};
```

积压订单中的订单总数

```
struct CMP1 {
    bool operator()(vector<int> a, vector<int> b) { return a[0] < b[0]; }
};

struct CMP2 {
    bool operator()(vector<int> a, vector<int> b) { return a[0] > b[0]; }

};

class Solution {
public:
    int getNumberOfBacklogOrders(vector<vector<int>> &orders) {
        Heap<vector<int>> buy{CMP1()}, sell{CMP2()};
        vector<int> temp(2);
        for (auto x: orders) {
            if (x[2] == 0) {
                while (x[1] != 0 && sell.size() && sell.top()[0] <= x[0]) {
                    int cnt = min(x[0], sell.top()[1]);
                    x[1] -= cnt;
                    sell.top()[1] -= cnt;
                    if (sell.top()[1] == 0) { sell.pop(); }
                }
                if (x[1] != 0) { buy.push(x); }
```

```
            }
            else {
                while (x[1] != 0 && buy.size() && buy.top()[0] >= x[0]) {
                    int cnt = min(x[0], buy.top()[1]);
                    x[1] -= cnt;
                    buy.top()[1] -= cnt;
                    if (buy.top()[1] == 0) { buy.pop(); }
                }
                if (x[1] != 0) { sell.push(x); }
            }
        }
        int sum = 0;
        int mod = 1000000007;

        for (auto x: buy) {
            sum = (sum + x[1]) % mod;
        }
        for (auto x: sell) {
            sum = (sum + x[1]) % mod;
        }
        return sum;
    }
};
```

第三部分 智力发散题

丑数 II

```
class Solution {
public:
    int nthUglyNumber(int n) {
        Heap<long long> h{greater<long long>()};
        long long ans = 0;
        h.push(1);
        while (n--) {
            ans = h.top();
            h.pop();
            if (ans % 5 == 0) { h.push(ans * 5); }
            else if (ans % 3 == 0) {
                h.push(ans * 5);
                h.push(ans * 3);
            }
            else {
                h.push(ans * 5);
                h.push(ans * 3);
                h.push(ans * 2);
            }
        }
        return ans;
    }
};
```

超级丑数

```cpp
class Solution {
public:
    int nthSuperUglyNumber(int n, vector<int> &primes) {
        vector<int> p(primes.size());
        vector<int> data;
        data.push_back(1);
        int ans = 1;
        while (data.size() != n) {
            ans = primes[0] * data[p[0]];
            for (int i = 1; i < primes.size(); ++i) {
                ans = min(ans, primes[i] * data[p[i]]);
            }
            for (int i = 0; i < primes.size(); ++i) {
                if (primes[i] * data[p[i]] == ans) { p[i]++; }
            }
            data.push_back(ans);
        }
        return ans;
    }
};
```

移除石子的最大得分

```cpp
class Solution {
public:
    int maximumScore(int a, int b, int c) {
        if (a > b) { swap(a, b); }
        if (a > c) { swap(a, c); }
        if (b > c) { swap(b, c); }
        int ans = 0;
        int cnt1 = min(c - b, a);
        a -= cnt1;
        c -= cnt1;
        ans += cnt1;
        if (a != 0) {
            if (a % 2 == 1) { a -= 1; }
            b -= a / 2;
            c -= a / 2;
            ans += a;
        }
        ans += b;
        return ans;
    }
};
```