信息学奥赛笔记03

优先级队列（堆）

优先级队列

队列是一种先进先出(First In First Out，FIFO)，的数据类型，每次元素入队只能添加到队列的尾部，出队时从队列的头部开始出。

优先级队列其实它并不满足**先进先出**的条件，每次在出队时会优先出**优先级最高**的那个元素，而不是真正的队头元素，这个优先级可以通过程序员的手动设置来更改调换。

定义方式

```
priority_queue<typename, container, compare>
```

其中优先级队列的尖括号内要写3个元素。

- **typename**表示元素的类型
- **container**是容器类型，可以是vector，queue等用数组实现的容器，默认是vector
- **compare**是比较方式，默认情况是大根堆（优先出队元素值最大的那一个）；如果使用的是C++基本的数据类型，可以使用**greater**和**less**这两个仿函数。

大根堆

默认的情况，优先级队列会优先出队最大的那个元素，定义方式有两种。

```
priority_queue<typename>;
priority_queue<typename, vector<typename>, less<typename> >;
```

typename可以填入任何一个类型，也就是说，默认情况，优先级队列是一个大根堆。

小根堆

在小根堆的情况下，优先级队列会优先出队最小的那个元素，定义方式为

```
priority_queue<typename, vector<typename>, greater<typename> >;
```

使用方式

1) 插入函数push();

```
priority_queue<int> pq;
pq.push(3);
pq.push(1);
// 将3 1 入队后，队头元素为3
```

2) 出队函数pop();

```
priority_queue<int> pq;
pq.push(3);
pq.pop();
pq.push(1);
// 3入队后出队，1入队，此时队列元素长度为1
```

3) 长度函数size();

```
priority_queue<int> pq;
pq.push(3);
pq.push(5);
cout << pq.size() << endl;
// 输出 2
```

4) 判空函数empty();

```
priority_queue<int> pq;
cout << pq.empty();//结果为1，为真
pq.push(3);
cout << pq.empty();//结果为0，为假
```

5) 队头函数top();

```
priority_queue<int> pq;
pq.push(3);
cout << pq.top();//3
pq.push(1);
pq.push(5);
cout << pq.top();//5
pq.pop();
cout << pq.top();//3
```

[P3378] 堆https://www.luogu.com.cn/problem/P3378

题目描述

给定一个数列，初始为空，请支持下面三种操作：

1. 给定一个整数 x，请将 x 加入到数列中。
2. 输出数列中最小的数。
3. 删除数列中最小的数（如果有多个数最小，只删除 1 个）。

输入格式

第一行是一个整数，表示操作的次数 n。
接下来 n 行，每行表示一次操作。每行首先有一个整数 op 表示操作类型。

- 若 $op = 1$，则后面有一个整数 x，表示要将 x 加入数列。
- 若 $op = 2$，则表示要求输出数列中的最小数。
- 若 $op = 3$，则表示删除数列中的最小数。如果有多个数最小，只删除 1 个。

输出格式

对于每个操作 2，输出一行一个整数表示答案。

样例 #1

样例输入 #1

```
5
1 2
1 5
2
3
2
```

样例输出 #1

```
2
5
```

提示

【数据规模与约定】

- 对于 30% 的数据，保证 $n \leq 15$。
- 对于 70% 的数据，保证 $n \leq 10^4$。
- 对于 100% 的数据，保证 $1 \leq n \leq 10^6$，$1 \leq x < 2^{31}$，$op \in \{1, 2, 3\}$。

思路分析

> 这道题没有涉及到任何的算法，就只是对堆最基本的使用。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, op, x;
    cin >> n;
    priority_queue<int, vector<int>, greater<int> >pq;
```

```
    while (n--) {
        cin >> op;
        if (op == 1) {
            cin >> x;
            pq.push(x);
        } else if (op == 2) {
            cout << pq.top() << endl;
        } else {
            pq.pop();
        }
    }
    return 0;
}
```

[P1090] 合并果子https://www.luogu.com.cn/problem/P1090

题目描述

在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。

每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 $n-1$ 次合并之后， 就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 1 ，并且已知果子的种类 数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例如有 3 种果子，数目依次为 1 ， 2 ， 9 。可以先将 1 、 2 堆合并，新堆数目为 3 ，耗费体力为 3 。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为 12 ，耗费体力为 12 。所以多多总共耗费体力 $=3+12=15$ 。可以证明 15 为最小的体力耗费值。

输入格式

共两行。
第一行是一个整数 $n(1 \leq n \leq 10000)$ ，表示果子的种类数。

第二行包含 n 个整数，用空格分隔，第 i 个整数 $a_i(1 \leq a_i \leq 20000)$ 是第 i 种果子的数目。

输出格式

一个整数，也就是最小的体力耗费值。输入数据保证这个值小于 2^{31} 。

样例 #1

样例输入 #1

```
3
1 2 9
```

样例输出 #1

```
15
```

提示

对于 30% 的数据，保证有 $n \leq 1000$:

对于 50% 的数据，保证有 $n \leq 5000$;

对于全部的数据，保证有 $n \leq 10000$。

思路分析

> 这道题当中，我们需要明确，既然所有的果子都需要合并成一堆，也就是说过程最小才能保证总和最小，既然最终都只剩1堆果子，所以当前的情况对之后的情况不会造成屏蔽和影响，所以每次合并的时候只需要选择最轻的两堆进行合并即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, x, ans = 0;
    cin >> n;
    priority_queue<int, vector<int>, greater<int> > pq; // 创建小根堆
    for (int i = 0; i < n; i++) {
        cin >> x;
        pq.push(x); // 每输入一个数，入直接入堆
    }
    for (int i = 1; i < n; i++) {
        int merge = 0;
        merge += pq.top(); pq.pop(); // 每次进行2次出队，计算好出队的结果
        merge += pq.top(); pq.pop();
        pq.push(merge); // 再将刚刚的结果入队
        ans += merge; // 对取出的最小的两堆果子合并后的结果进行求和
    }
    cout << ans << endl;
    return 0;
}
```

[P2085]最小函数值

题目描述

有 n 个函数，分别为 $F_1, F_2, \ldots, F_n$。定义 $F_i(x) = A_i x^2 + B_i x + C_i (x \in \mathbb{N}*)$。给定这些 A_i、B_i 和 C_i，请求出所有函数的所有函数值中最小的 m 个（如有重复的要输出多个）。

输入格式

第一行输入两个正整数 n 和 m。

以下 n 行每行三个正整数，其中第 i 行的三个数分别为 A_i、B_i 和 C_i。

输出格式

输出将这 n 个函数所有可以生成的函数值排序后的前 m 个元素。这 m 个数应该输出到一行，用空格隔开。

样例 #1

样例输入 #1

```
3 10
4 5 3
3 4 5
1 7 1
```

样例输出 #1

```
9 12 12 19 25 29 31 44 45 54
```

提示

数据规模与约定

对于全部的测试点，保证 $1 \leq n, m \leq 10000$，$1 \leq A_i \leq 10, B_i \leq 100, C_i \leq 10^4$。

思路分析

> 对于每个函数，我们可以算出这个函数的前m小的数，那么一共有n个函数，最多就会算出n * m种答案，我们取这些答案当中的最小的m个值，就是结果。
>
> 但是，这道题如果数据拉满的话，$10^4 * 10^4 * log10^4 > 10^8$会导致超时的。所以咱们肯定不能把所有的结果都算出来，那其实咱们可以控制，只要队列中一直去维护m小的数就行了，每次发现新的数，push进来，pop一个最大的，如果当前的需要push的数比最大的还大，那从当前这个接过来看，根本没必要继续搜索下去了。直接进行下一个函数就好。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, m, a, b, c;
    cin >> n >> m;
    priority_queue<int> pq;
    while (n--) {
        cin >> a >> b >> c; // ax^2 + bx + c
        for (int x = 1; x <= m; x++) {
            int ret = a * x * x + b * x  + c;
            if (pq.size() == m && ret >= pq.top()) break; // 剪枝
            pq.push(ret);
            if (pq.size() > m) {  // 如果队列的元素超过m个，出去一个
                pq.pop();
            }
        }
    }
    vector<int> ans; // 把从大到小的结果转成从小到大的结果
```

```
    while (!pq.empty()) {
        ans.push_back(pq.top()); pq.pop();
    }
    for (int i = m - 1; i >= 0; i--) {
        cout << ans[i] << " ";
    }
    return 0;
}
```