复习：
基础数据结构与二分

基础-提高衔接计划 B

览遍千秋

2025-07-14

洛谷

www.luogu.com.cn

课前提示

- 上课的时候专心听讲解，**不要跟着老师抄代码**，下课后独立完成。
- 不使用 AI 做题，AI 会做不等于自己会。
- 不抄袭题解（含对照题解抄一遍），抄对不等于会做。
- 看完题解后，关闭题解独立练习。
- 练习中途遇到问题，应当分析题目及自己的思路，而非回忆题解或再次参考题解。
- 做过的题在课后需要重新独立完成，不参考老师的课件、代码，不参考自己以前的代码。

什么是线性数据结构？

- 想象若干人正在排队打饭，每个人都一个挨一个站着，这就是线性数据结构的样子
- 线性数据结构就像是一条“排好队”的数据队伍，所有元素一个接一个地排列
- 不同的线性数据结构，会支持对队伍的多种多样的修改/查看
- 例如：在队伍头部塞入一个人，从队伍尾部拿出一个人
- 例如：让队伍中每个人只知道自己前面是谁、后面是谁

目录

队列 queue

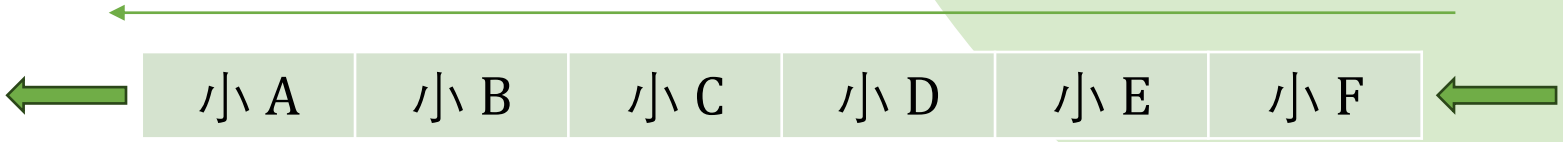
生活中的队列

- 食堂窗口：先来的人先打饭
- 车站检票口：先来的人先通过检票口

- 入队（队尾增加一个人）
- 出队（最前面的人离开队伍）
- 取队头（给队伍的第一个人打饭）

队列

- 先进先出（FIFO，First In First Out）
- 线性表



- 入队
- 出队
- 取队头
- 队列长度
- 判断队列为空

queue

- queue 是 STL 提供的队列容器
- 包含在头文件 queue 中
- 直接使用万能头亦可

queue

- `queue <T> QueueName;`

- 定义一个存储 int 类型的队列 Q
- `queue <int> Q;`

- 定义一个存储 double 类型的队列 w
- `queue <double> w;`

queue

```
struct Info {
    int x, y;
};
```

- 如何定义一个 Info 类型的队列 U?

成员函数

- 成员函数
- 变量名.函数名(参数);

```
struct MyType {
    int x;
    void plus(int k) { x += k; }
    int get() { return x; }
} A;
A.push(3);
A.get();
```

queue

- 入队
- q.push(x);
- x 为要插入的元素，需要与 queue 定义时类型一致
- 函数为 void 函数，即没有返回值

- 出队
- q.pop();
- 没有返回值
- 没有参数
- 只是将队头从队列里扔出去

queue

- 取队头
- `q.front()`
- 没有参数
- 有返回值 类型和 q 定义的类型相同

- 查询元素个数
- `q.size()`
- 没有参数
- 返回值为 unsigned int 类型
- 如何判断队列内是否有 5 个或以上元素?

queue

- 查询队列是否为空
- `q.empty()`
- 没有参数
- 返回值为 bool 类型
- 当队列为**空**时返回 true

- 如何判断队列是否为空?
- `if(q.empty())`
- `if(q.size() == 0)`

queue

- 不可以通过下标访问 queue 中的某一个元素

B3616 【模板】队列

- https://www.luogu.com.cn/problem/B3616

B3616 队列

```cpp
int T, op, x;
queue <int> Q;
int main() {
    cin >> T;
    while(T--) {
        cin >> op;
        if(op == 1) {
            cin >> x;
            Q.push(x);
        }
        else if(op == 2) {
            if(!Q.empty()) Q.pop();
            else cout << "ERR_CANNOT_POP" << endl;
        }
        else if(op == 3) {
            if(!Q.empty()) cout << Q.front() << endl;
            else cout << "ERR_CANNOT_QUERY" << endl;
        }
        else cout << Q.size() << endl;
    }
}
```

双端队列 deque

双端队列

- 某医院诊室门口，按照先来后到顺序排队
- 危急症患者可以插队到最前面
- 最后面的人离开了队伍

- 从队尾入队（正常排队）
- 从队头入队（危重患者）
- 从队头出队（第一个人看病）
- 从队尾出队（最后一个人离开）
- 取队头
- 取队尾

deque

- deque 是 STL 提供的双端队列容器
- 包含在头文件 queue 中

deque

- 定义一个 deque
- 和 queue 类似
- `deque <T> DequeName;`

deque

- 入队
- 从队头入队：`dq.push_front(x)`
- 从队尾入队：`dq.push_back(x)`
- 都没有返回值，x 与 dq 定义类型相同

- 出队
- 从队头出队：`dq.pop_front()`
- 从队尾出队：`dq.pop_back()`
- 没有参数 没有返回值

deque

- 取队首
- `dq.front()`
- 没有参数
- 返回队首的元素

- 取队尾
- `dq.back()`
- 没有参数
- 返回队尾的元素

deque

- deque 的大小
- `dq.size()`
- 没有参数
- 返回值为 unsigned int 类型

- 判断 deque 是否为空
- `dq.empty()`
- 返回值为 bool 类型
- 当 deque 为___时返回 true?

deque 的内存占用

- 申请一个 deque 就会占用至少 512B 的空间
- 使用 deque 需要特别注意空间占用
- 申请 10^6 个空 deque 就会 MLE

P2952 Cow Line S

- https://www.luogu.com.cn/problem/P2952

P2952 Cow Line S

```
deque <int> dq;
int main() {
    cin >> T;
    while(T--) {
        cin >> op >> dir;
        if(op == "A") {
            ++cnt;
            if(dir == "L") dq.push_front(cnt);
            else dq.push_back(cnt);
        }
        else {
            cin >> k; while(k--) {
                if(dir == "L") dq.pop_front();
                else dq.pop_back();
            }
        }
    }
    while(dq.size()) {
        cout << dq.front() << endl;
        dq.pop_front();
    }
}
```

栈 stack

栈

- 吃完午饭后，将盘子一个个放进水池里，之后开始洗盘子
- 放入盘子时，只能放在之前盘子的最顶上
- 洗盘子时，每次只能拿盘子堆最顶上的盘子

- 先放进去的盘子被后洗
- 先进后出

- 入栈（放入新盘子）
- 出栈（最上面的盘子拿走）
- 取栈顶（洗最上面的盘子）

stack

- STL 提供的栈容器
- 包含在头文件 stack 中

stack

- stack 的定义
- `stack <T> StackName;`
- STL 的容器定义基本符合这个格式

stack

- 进栈
- `st.push(x);`
- 参数与栈定义的类型相同
- 没有返回值

- 出栈
- `st.pop();`
- 没有参数与返回值

stack

- 取栈顶
- `st.top();`
- 没有参数
- 返回值与 stack 定义类型相同

- 栈的大小
- `st.size();`
- 没有参数
- 返回值为 unsigned int 类型

stack

- 判断栈是否为空
- `st.empty()`
- 没有参数
- 返回值为 bool 类型

B3614 栈

- https://www.luogu.com.cn/problem/B3614

B3614 栈

```
int main() {
    cin >> T;
    while(T--) {
        while(s.size()) s.pop();
        cin >> n;
        while(n--) {
            string op; cin >> op;
            if(op == "push") {
                ull x; cin >> x;
                s.push(x);
            }
            else if(op == "pop")
                if(s.empty()) cout << "Empty" << endl;
                else s.pop();
            else if(op == "query")
                if(s.empty()) cout << "Anguei!" << endl;
                else cout << s.top() << endl;
            else cout << s.size() << endl;
        }
    }
}
```

可变长数组 vector

定长数组

- 在之前的课程中，我们要求同学：
- 全部数组定义在 main 函数外
- 定义常量大小的数组

错误的可变长数组

```
int main() {
    int n; cin >> n;
    int a[n + 1];
}
```

vector

- vector 是 STL 提供的可变长数组
- 包含在头文件 vector 中

vector

- 定义
- `vector <int> vec;`

- `int n; cin >> n;`
- `vector <int> vec(n + 1);`

- `int n; cin >> n;`
- `vector <int> vec(n + 1, 0);`

vector

- 获取大小
- `vec.size()`
- 返回值为 unsigned int

- 访问特定元素
- vec[x]（与数组类似）
- 下标范围为 0 至 vec.size()-1

vector

- 在末尾加入元素
- `vec.push_back(x)`
- 没有返回值
- 参数与定义相同

- 清空 vector
- `vec.clear()`
- 清空 vec 的元素，此后调用 size() 返回 0
- 但是不释放 vec 占用的内存
- 没有返回值 没有参数

vector

- 重新指定 vector 大小为 x
- `vec.resize(x)`
- 没有返回值
- 参数为 int 类型

vector

- `vec.begin()`
- `vec.end()`
- 返回指向 vec 开头和结尾的迭代器
- 在使用 sort、lower_bound 等 STL 函数时常见

vector

- 排序
- `sort(vec.begin(), vec.end());`

- 遍历
- `for(int i = 0; i < vec.size(); i++) {`
- `    cout << vec[i] << endl;`
- `}`

vector 数组

- `vector <int> vec[20];`
- `vector <vector<int> > E(20);`

- vec 是什么？
- vec[19] 是什么？
- vec[19][18] 是什么？
- E 是什么？
- E[19] 是什么？
- E[19][18] 是什么？

P3613 寄包柜

- https://www.luogu.com.cn/problem/P3613

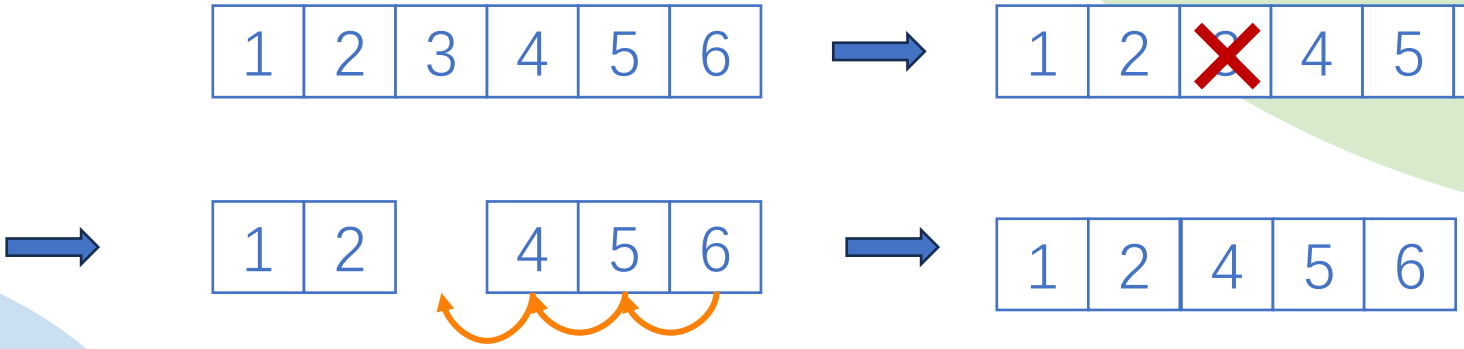
P3613 寄包柜

```cpp
int main() {
    cin >> n >> T;
    vector <vector<int> > vec(n + 1);
    while(T--) {
        int op, i, j, k;
        cin >> op >> i >> j;
        if(op == 1) {
            cin >> k;
            if(vec[i].size() < j)
                vec[i].resize(j + 1);
            vec[i][j] = k;
        }
        else cout << vec[i][j] << endl;
    }
}
```

链表 list

链表

- 可能会遇到一些情况，要求我们频繁地插入 / 删除一段序列中某两个元素之间的元素。
- 数组：（想要删除 3 号元素）非常多的时间用在了数据迁移上。



- 有没有一种方法可以快速的完成这样的操作?

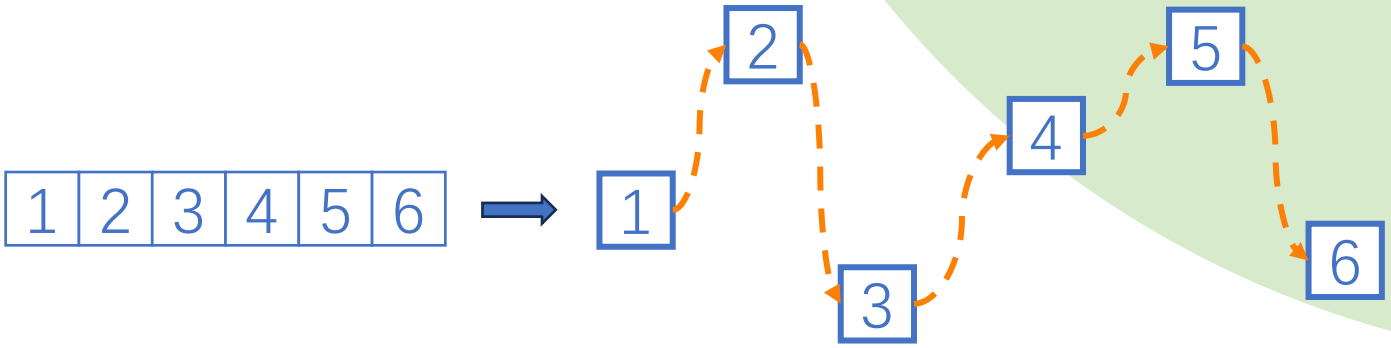
链表

- 不再关注元素的绝对位置（例如，其在数组中是第几位）
- 而只关心元素的相对位置

- 站队
- 一种方式是记住自己排在第几个
- 另一种方式是记住“自己前面是谁，后面是谁”，站队时直接找到自己前面后面的人，插入其中即可

链表

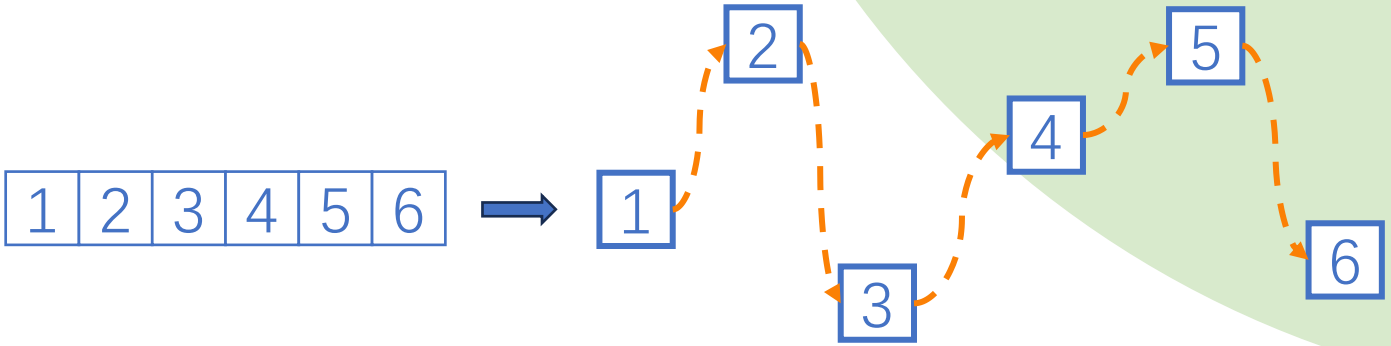
- 用途：可能会遇到一些情况，要求我们频繁地插入 / 删除一段序列中某两个元素之间的元素。放到链表上怎么做？



- 1 的下一个是 2，2 的下一个是 3，...

链表

- 在算法竞赛中往往使用数组下标表示“前面是谁，后面是谁”
- int nxt[XXX], pre[XXX]



- nxt[1] = 2, nxt[2] = 3, nxt[3] = 4, ...
- pre[6] = 5, pre[5] = 4, pre[4] = 3, ...

链表

- 如何修改链表？
- 添加、删除
- 稳妥方法：把要改的元素全部暂存起来，之后想怎么改怎么改

B3631 单向链表

- https://www.luogu.com.cn/problem/B3631

B3631 单向链表

```
for (int o = 1; o <= q; o++) {
    int op;
    cin >> op;
    if (op == 1) {
        int x, y; cin >> x >> y; nxt[y] = nxt[x]; nxt[x] = y;
    } else if (op == 2) {
        int x; cin >> x; cout << nxt[x] << endl;
    } else if (op == 3) {
        int x; cin >> x; nxt[x] = nxt[nxt[x]];
    }
}
```

B4324 【模板】双向链表

- https://www.luogu.com.cn/problem/B4324

B4324 【模板】双向链表

```
void remove(int x) {
    if (nxt[x] == -1 && pre[x] == -1) { return; }
    int a1 = pre[x], a2 = x, a3 = nxt[x]; // a1 -> a3
    nxt[a1] = a3; pre[a3] = a1;
    nxt[a2] = -1; pre[a2] = -1; // remove x
}
void iright(int x, int y) {
    remove(x); int a1 = y, a2 = x, a3 = nxt[y]; // a1 -> a2 -> a3
    nxt[a1] = a2; nxt[a2] = a3; pre[a3] = a2; pre[a2] = a1;
}
void ileft(int x, int y) {
    remove(x); int a1 = pre[y], a2 = x, a3 = y; // a1 -> a2 -> a3
    nxt[a1] = a2; nxt[a2] = a3; pre[a3] = a2; pre[a2] = a1;
}
```

B4324 【模板】双向链表

```
int main() {
    int n; cin >> n; int q; cin >> q;
    nxt[0] = 1; pre[n + 1] = n;
    for (int i = 1; i <= n; i++) { nxt[i] = i + 1; pre[i] = i - 1; }
    for (int o = 1; o <= q; o++) {
        int op; cin >> op;
        if (op == 1) {
            int x, y; cin >> x >> y; if (x == y) continue; ileft(x, y);
        } else if (op == 2) {
            int x, y; cin >> x >> y; if (x == y) continue; iright(x, y);
        } else if (op == 3) {
            int x; cin >> x; remove(x);
        }
    }
...
```

B4324 【模板】双向链表

```
…
    if (nxt[0] == n + 1) {
        cout << "Empty!\n";
    } else {
        int p = nxt[0];
        while (p != n + 1) {
            cout << p << " "; p = nxt[p];
        }
    }
}
```

list

- STL 提供的链表容器
- 包含在头文件 list 中
- `list <T> listName;`
- 但是非常非常难用，在正式比赛中，很少见到使用这种方式完成链表的

- 可以用来当一个双端队列

list

- 入队
- 从队头入队：`l.push_front(x)`
- 从队尾入队：`l.push_back(x)`
- 都没有返回值，x 与 l 定义类型相同

- 出队
- 从队头出队：`l.pop_front()`
- 从队尾出队：`l.pop_back()`
- 没有参数 没有返回值

list

- 取队首
- `l.front()`
- 没有参数
- 返回队首的元素

- 取队尾
- `l.back()`
- 没有参数
- 返回队尾的元素

list

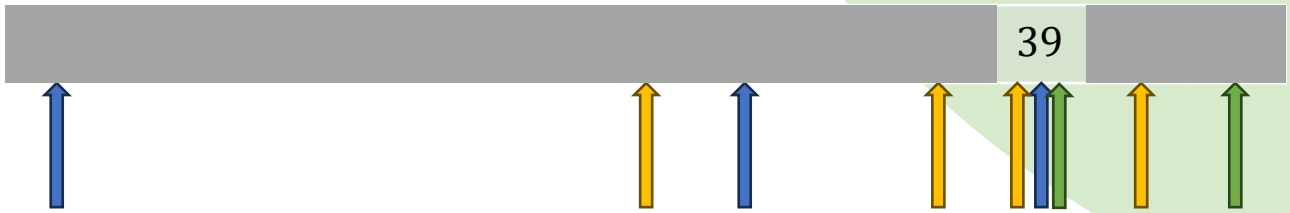
- list 的大小
- `l.size()`
- 没有参数
- 返回值为 unsigned int 类型

- 判断 list 是否为空
- `l.empty()`
- 返回值为 bool 类型

二分查找与二分答案

二分查找

- 假设 a 是长度为 n 的**递增**数列
- 尝试查找 39



二分查找

- 二分查找的基本格式
- 以递增数列为例

```
int solve(int x) {
    int l = 1, r = n, ans = -1;
    while(l <= r) {
        int mid = (l + r) / 2;
        if(a[mid] == x) ans = mid;
        if(a[mid] > x) r = mid - 1;
        else l = mid + 1;
    }
    return ans;
}
```

单调性

- 二分查找必须在单调的数列中应用
- 如果数列不单调，就无法根据 a[mid] 和 x 的大小关系，确定 x 在mid 左侧还是右侧

- 应用二分查找前必须考察单调性

非严格单调问题

- 严格单调增
- $a_1 < a_2 < \cdots < a_n$
- 非严格单调增（不减）
- $a_1 \leq a_2 \leq \cdots \leq a_n$
- 查找第一个 x
- 查找最后一个 x
- 查找有多少个 x

非严格单调问题

- 查找第一个 x
- 如果 a[mid] 等于 x
- 　记录当前位置
- 　第一个 x 的位置应该在左侧，即 r=mid-1
- 如果 a[mid]>x
- 　第一个 x 的位置应该在左侧
- 如果 a[mid]<x
- 　第一个 x 的位置应该在右侧

P2249 查找

- https://www.luogu.com.cn/problem/P2249

P2249 查找

```
int solve(int x) {
    int l = 1, r = n, ans = -1;
    while(l <= r) {
        int mid = (l + r) / 2;
        if(a[mid] == x) ans = mid, r = mid - 1;
        else if(a[mid] > x) r = mid - 1;
        else l = mid + 1;
    }
    return ans;
}
```

B3627 立方根

- https://www.luogu.com.cn/problem/B3627

B3627 立方根

```
ll l = 1, r = 100000, ans;
while(l <= r) {
    ll mid = (l + r) / 2;
    if(check(mid)) ans = mid, l = mid + 1;
    else r = mid - 1;
}
```

```
bool check(ll x) {
    if(x * x * x <= n) return true;
    return false;
}
```

二分答案

- 像立方根这种题目
- 如果对于要求的答案，p 能够满足/不满足条件，对于所有 >p 的答案都满足/不满足条件
- 存在临界点 p_0

- 求xx的最大值最小
- 求xx的最小值最大

二分答案

- 核心：check 的策略
- 上下界的确定

```
int l = ?, r = ?, ans;
while(l <= r) {
    int mid = (l + r) / 2;
    if(check(mid)) ans = mid, r = mid - 1;
    else l = mid + 1;
}
```

本质特征

- 最优化问题转换为判定性问题
- $\sqrt[3]{x}$ 的值是多少？ $\rightarrow$ 二分 k，k^3 是不是 x？

二分查找与二分答案的联系

- 二分查找是一类特殊的二分答案
- 二分的答案为下标

B3628 机器猫斗恶龙

- https://www.luogu.com.cn/problem/B3628

B3628 机器猫斗恶龙

```
int l = 1, r = 1000 * n, ans;
while(l <= r) {
    int mid = (l + r) / 2;
    if(check(mid)) ans = mid, r = mid - 1;
    else l = mid + 1;
}
```

```
bool check(int x) {
    for(int i = 1; i <= n; i++) {
        x += a[i];
        if(x <= 0) return false;
    }
    return true;
}
```