第一周直播课（下午）
基础线性数据结构与 STL

洛谷网校
基础衔接提高计划

洛谷
www.luogu.com.cn

关于 STL 的声明

目前是否可以在正式比赛中使用 STL?

是。目前阶段（CSP-J/S），大多数情况下，可以放心使用 STL。

STL 会不会比我手写的要慢?

目前的大环境下（`-std=c++14`，`-O2`），绝大多数情况下不会。

指针

指针

概念

变量的在物理内存中的存放地址。

地址也是数据。存放地址所用的变量类型有一个特殊的名字，叫做「指针变量」，有时也简称做「指针」。

日常见到的 `int*`，`char*`，某个结构体* 都是指针。它们都指向某一个内存地址，只是后续有效的数据大小不同。

指针

引用

通过在变量名前加 & 获取变量地址。常见于 scanf。

解引用

访问指针变量地址所对应的空间（又称指针所指向的空间），使用 * 符号。

对结构体内变量访问也需要 *，不过有一种取巧的办法是直接使用 -> 符号。

指针

偏移量

数组是一块连续的空间。C/C++中直接访问数组名，得到的是数组的起始地址。

常用 [] 来访问数组中某个元素，这个过程间接施加了偏移量。指针允许使用 +/- 施加偏移量，一个偏移量就是一个指针对应的数据的大小。

对数组 a，a + 4 和 &a[4] 是等价的。

常见于 std::sort(a + 1, a + n + 1);。

空指针

C++11 之后用 nullptr / NULL 表示空指针，代表指针不指向任何有效内存地址。

指针

new

`type* t = new type();` 建立一个变量，并以指针的形式发送给 `t`。

也可以用于构建数组。`type* t = new type[12345];`

做了解即可。

delete / delete[]

释放一块内存，做了解即可。

string 类与 STL 算法模板库函数

类模板

概念复习

类模板（class template）本身不是一个类，而是可以根据 不同数据类型 产生 不同类 的「模板」。

在使用时，编译器会根据传入的数据类型产生对应的类，再创建对应实例。

模板属于 C++ 较为高级的语言特性，在信息学竞赛中几乎不会出现。如果对此感兴趣，可以进一步阅读《C++ Primer》以学习更深层次的 C++ 知识。

常常使用一对尖括号 <***> 来使用类模板，后面的例子中将会很常用这个部分。

C++ string 类

概念复习

方法	含义	例子
size / length	字符串长度	int len = s.length();
clear	清空字符串	s.clear();

+= / append / push_back	向后插入内容	s += "aaa";
		s.append("aaa");
		s.push_back('a');

insert	向指定位置插入字符串	s.insert(3, "aaa");
replace	替换指定位置的字符串	s.replace(3, 17, "aaa");
erase	删去指定区间的字符串	s.erase(3, 7);

C++ STL 算法模板库函数

概念复习

方法	含义	例子
min, max	最小/最大值	min(a, b); max(a, b);
swap	交换变量的值	swap(a, b);
sort	排序	sort(a + 1, a + n + 1);
reverse	翻转序列	reverse(a + 1, a + n + 1);
unique	去除相邻重复元素	unique(a + 1, a + n + 1);
nth_element	找出序列第 k 大元素	nth_element(a + 1, a + mid, a + n + 1);
lower_bound upper_bound	找到第一个大于（等于）指定元素的元素位置	lower_bound(a + 1, a + n + 1, k);
next_permutation	下一个排列	do while next_permutation

简单线性数据结构与 STL 容器

简单线性数据结构概念

栈：先进后出（FILO）的线性数据结构。

队列：先进先出（FIFO）的线性数据结构。

链表：一些结点穿成的链，每个结点存储了其下一个结点。

双向链表：在链表基础上每个结点存储了其上一个链表结点。

循环链表：首尾相连的链表。

栈、队列

栈：先进后出（FILO）的线性数据结构。

队列：先进先出（FIFO）的线性数据结构。

链表

概念

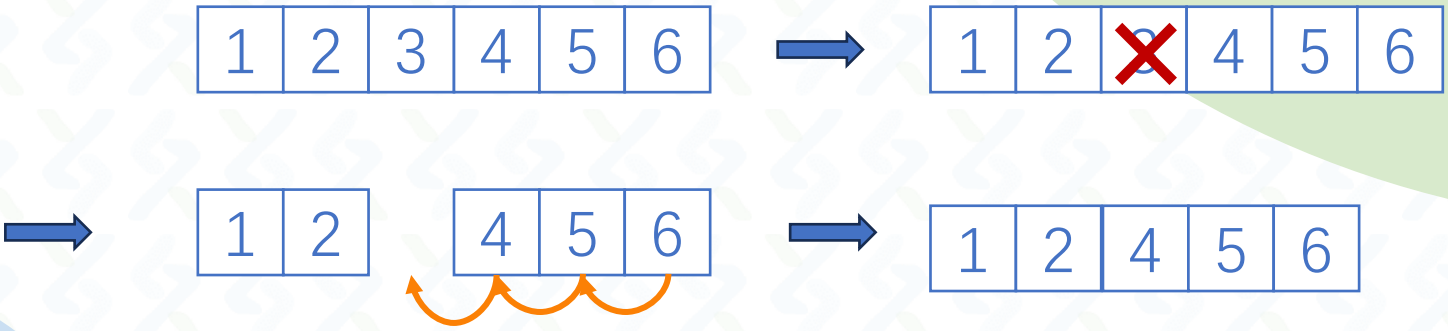
一种用于存储数据的数据结构，通过如链条一般的指针来连接元素。

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。

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

为什么数组会慢?

在数组中，定位元素的方式是使用下标。

例如：a[3] 代表 a 数组的第 4 个元素。这是一个完全绝对的关系，不受数组内元素之间关系的影响。

我们需要一个元素间的相对的关系。

链表借助的是元素间的相对的关系。

链表

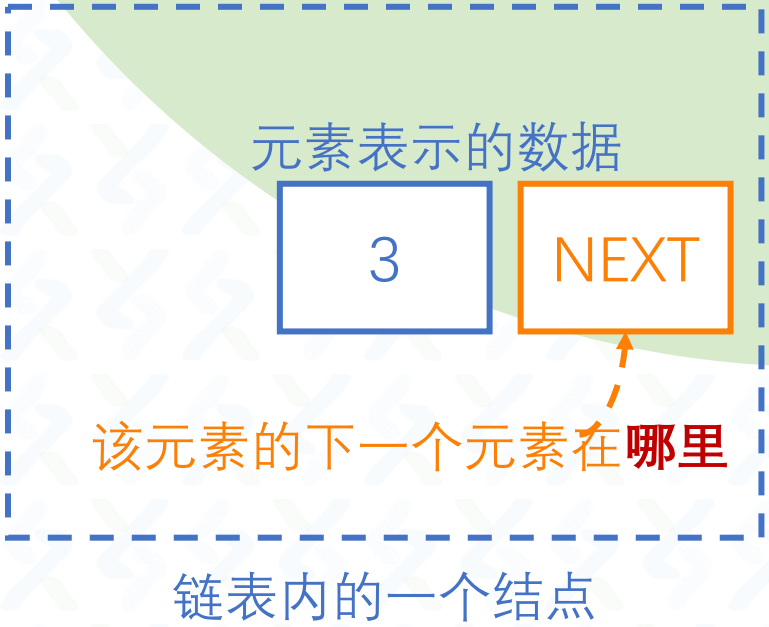
实现

算法竞赛中，一般另开数组来储存元素的相对关系。

如图。对于每一个元素，我们使用 NEXT 数组（一般写作 nxt）来存储某个元素的下一个元素。

上一个元素同理，一般用 pre。

插入，删除元素时，需要注意时序关系。



例题 – 单向链表

题目链接 https://www.luogu.com.cn/problem/B3631

实现一个数据结构，维护一张表（最初只有一个元素 1）。需要支持下面的操作，其中 x 和 y 都是 1 到 10^6 范围内的正整数，且保证任何时间表中所有数字均不相同，操作数量不多于 10^5：

$1\ x\ y$：将元素 y 插入到 x 后面；

$2\ x$：询问 x 后面的元素是什么。如果 x 是最后一个元素，则输出 0；

$3\ x$：从表中删除元素 x 后面的那个元素，不改变其他元素的先后顺序。

例题 – 单向链表

思路

直接采用一个 int 类型的 nxt 数组就可以。nxt[i] 表示数据为 i 的结点的 NEXT。

操作 1：nxt[y] = nxt[x]; nxt[x] = y;

操作 2：直接输出 nxt[x];

操作 3：nxt[x] = nxt[nxt[x]];

例题 – 单向链表

代码实现

```
cin >> q;
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]];
    }
}
```

链表

总结

链表是一种存储数据的数据结构，依赖结点的前后关系存储数据。

特点：插入与删除数据十分方便，但寻找与读取数据的表现欠佳。

与数组的区别：链表是链状结构，数组将所有元素按次序依次存储。

- 随机访问数据中，链表删除、插入数据时间复杂度/操作次数 $O(1)$；寻找、读取数据时间复杂度/操作次数 $O(n)$。
- 随机访问数据中，数组寻找、读取数据时间复杂度 $O(1)$，删除、插入的操作次数/时间复杂度 $O(n)$。

其空间复杂度为 $O(n)$。

vector

概念复习

中文名为向量，但是实际上是一个动态数组。

运算符/方法	含义	例子
size	动态数组大小	v.size();
empty	判断数组是否为空	if (v.empty()) ...
push_back	向后插入内容	v.push_back(3);
operator[]	访问某位置元素	int a = v[3];
pop_back	弹出最后一个元素	v.pop_back();
clear	清空动态数组	v.clear();
insert	在某位置插入一个元素（时间复杂度是 $O(n)$ 的）	v.insert(3, 4);
erase	删除某个/某一些元素	
front, back, begin, end	迭代器	

stack

概念复习

中文名为“栈”，遵循先进后出（FILO）原则。

运算符/方法	含义	例子
push	向栈中推入内容	`s.push(3);`
pop	弹出最后一个元素	`s.pop();`
top	查询栈顶元素	`int x = s.top();`
迭代器		

queue

概念复习

中文名为“队列”，遵循先进先出（FIFO）原则。

运算符/方法	含义	例子
push	向队列中推入内容	q.push(3);
pop	弹出队列头元素	q.pop();
front	查询队列头元素	int x = q.front();
迭代器		

deque

概念复习

中文名为双端队列，顾名思义可以 $O(1)$ 向头尾插入元素。

运算符/方法	含义	例子
size	队列大小	q.size();
empty	判断队列是否为空	if (q.empty()) ...
push_back	向后插入内容	q.push_back(3);
pop_back	弹出最后一个元素	q.pop_back();
push_front pop_front	同理	
clear	清空队列	q.clear();
insert	在某位置前插入一个元素	q.insert(it, 4);
erase	删除某个/某些元素	
front, back, begin, end	迭代器	

list

概念复习

中文名为“链表”，$O(1)$ 添加元素，$O(n)$ 查询元素。

运算符/方法	含义	例子
push_back	向后插入内容	l.push_back(3);
pop_back	弹出最后一个元素	l.pop_back();
push_front pop_front	同理	
clear	清空链表	l.clear();
insert	在某位置前插入一个元素（时间复杂度是 $O(1)$ 的）	l.insert(it, 4);
erase	删除某个位置元素	
front, back, begin, end	迭代器	

map

概念复习

中文名“映射表”，$O(\log n)$ 添加元素，$O(\log n)$ 查询元素对应的值。

运算符/方法	含义	例子
operator[]	访问某位置元素	mp["XiaoMing"] = 114;
erase	删除某个元素	mp.erase("XiaoMing");
find	查找某个元素	auto it = mp.find("XiaoMing");

set

概念复习

中文名“集合”，$O(\log n)$ 添加元素，$O(\log n)$ 查询元素是否存在。

运算符/方法	含义	例子
insert	添加元素	st.insert(123);
erase	删除某个元素	st.erase(123);
find	查找某个元素	auto it = st.find(137);
count	计算某个元素出现了多少次	int x = st.count(136);
lower_bound upper_bound	查找第一个大于（等于）给定元素的元素位置	auto it = st.lower_bound(4);

map 与 set

以上两种 STL 均有两个分支：**unordered** 和 **multi-**。

unordered：无序的，使用哈希表结构存储，一般情况下可以 $O(1)$ 执行一次操作。

multi-：多个的，允许放置多个相同元素。

其中 multimap 不常用。

priority_queue

概念复习

中文名为优先队列，是一种堆。

支持在 $O(\log n)$ 时间复杂度内插入一个数、查询最大/最小值（默认最大）、删除最大/最小值。

运算符/方法	含义	例子
size	队列大小	q.size();
empty	判断队列是否为空	if (q.empty()) ...
push	向后插入内容	q.push(3);
top	访问最小/最大元素	q.top()
pop	弹出最小/最大元素	q.pop();
clear	清空队列	q.clear();

pair

概念复习

一个存储两个变量的「对」。常用于将关联数据捆绑存储、处理的场景。

可以简单地将 `pair<type1, type2> p;` 理解成

```
struct { type1 first; type2 second; } p;
```

不同的是，`pair` 提供了更健全的赋值、构造、比较大小的方法。一定情况下可以省力。

`map` 中存储的键值对通过 `pair` 向外暴露。

习题

P3613 【深基15.例2】寄包柜（vector）

B3614 【模板】栈（stack）

B3616 【模板】队列（queue）

B3656 【模板】双端队列 1（deque）

P2580 于是他错误的点名开始了（map）

P1090 [NOIP2004 提高组] 合并果子（优先队列）

P1160 队列安排（链表）

P2141 [NOIP2014 普及组] 珠心算测验（set 在这里大材小用，不过可以尝试一下）

补充内容：位运算

以下内容不是教学大纲内内容，课上不做教学，感兴趣的同学可以自行了解。

基本位运算

运算符	运算名称	含义
&	按位与	二进制两对应位均为 1 时才为 1
|	按位或	只要两个对应位中有一个 1 就为 1
^	按位异或	只有两个对应位不同时才为 1
~	取反	二进制补码中 0 和 1 全部变反
<<	二进制左移	`num << i` 表示将 `num` 的二进制左移 `i` 位
>>	二进制右移	`num >> i` 表示将 `num` 的二进制右移 `i` 位

补码：在二进制表示下，非负整数的补码为其本身，负数的补码是将其对应正数按位取反后加一。

C++ 运算符优先级：https://zh.cppreference.com/w/cpp/language/operator_precedence

位运算应用

1. 表示集合：二进制中的每一位 0/1 代表集合中某元素的存在情况。
2. 特殊题目要求的位运算

获取一个数二进制的某一位

```
int getBit(int a, int b) { return (a >> b) & 1; }
```

将一个数二进制设为 0/1 / 取反

```
int unsetBit(int a, int b) { return a & ~(1 << b); }
int setBit(int a, int b) { return a | (1 << b); }
int flapBit(int a, int b) { return a ^ (1 << b); }
```

bitset

std::bitset 是标准库中一个存储 0/1 的大小不可变的容器。

```cpp
#include <bitset>
#include <iostream>
using namespace std;
const int SIZE = 1024;
int main() {
    bitset<SIZE> set, new_set;
    //operators
    set[1] = 1;
    cout << set[1] << endl; // operator []
    cout << (set != new_set) << endl; // operator == / !=
    set |= new_set; // operator & / | / ^ / &= / |= / ^=
    // can only operate with a bitset
    set <<= 1; // operator << / >> / <<= / >>=
    // member functions
    cout << set.count() << endl; // count
    cout << set.size() << endl; // size
    cout << set.any() << set.none() << set.all(); // any none all
    set.set(1, true); set.reset(1); set.flip(1); // set reset flip
}
```

例题 – 高低位交换

题目链接 https://www.luogu.com.cn/problem/P1100

给定一个小于 2^{32} 的非负整数，求这个整数高低 16 位互换后的结果。

例题 – 高低位交换

思路

使用左右移运算。

(n >> 16) 代表 n 的左 16 位，(n << 16)（n 为 unsigned 类型）代表 n 的右 16 位移到左边的结果（左 16 位溢出）

最终答案为 (n >> 16) + (n << 16)（n 为 unsigned 类型）。

```
unsigned n;
int main(){
    cin >> n;
    cout << ((n >> 16) + (n << 16)) << endl;
    return 0;
}
```

例题 – 集合运算 1

题目链接 https://www.luogu.com.cn/problem/B3632

给定两个集合 A, B，求 $|A|$，$A \cap B$，$A \cup B$。

$$1 \leq |A|, |B| \leq 63$$

例题 – 集合运算 1

思路

使用位运算实现，可以使用 unsigned long long，这里介绍一个 std::bitset 做法。

使用两个 std::bitset，分别记录集合 A 和 B 的信息。

使用 count() 函数计算集合大小，使用 & | 运算符做集合的 $\cap$，$\cup$ 操作。

例题 – 集合运算 1

代码实现

```
bitset<64> a, b;
int x, y;
cin >> x;
for (int i = 1; i <= x; ++i) {
    int v; cin >> v; a.set(v, 1);
}
cin >> y;
for (int i = 1; i <= y; ++i) {
    int v; cin >> v; b.set(v, 1);
}
cout << a.count() << endl;
bitset<64> c = a & b;
for (int i = 0; i <= 63; ++i)
    if (c[i]) cout << i << " ";
cout << endl;
c = a | b;
for (int i = 0; i <= 63; ++i)
    if (c[i]) cout << i << " ";
cout << endl;
```