[17] 集合

深入浅出程序设计竞赛
第 3 部分 – 简单数据结构
V 2021-02

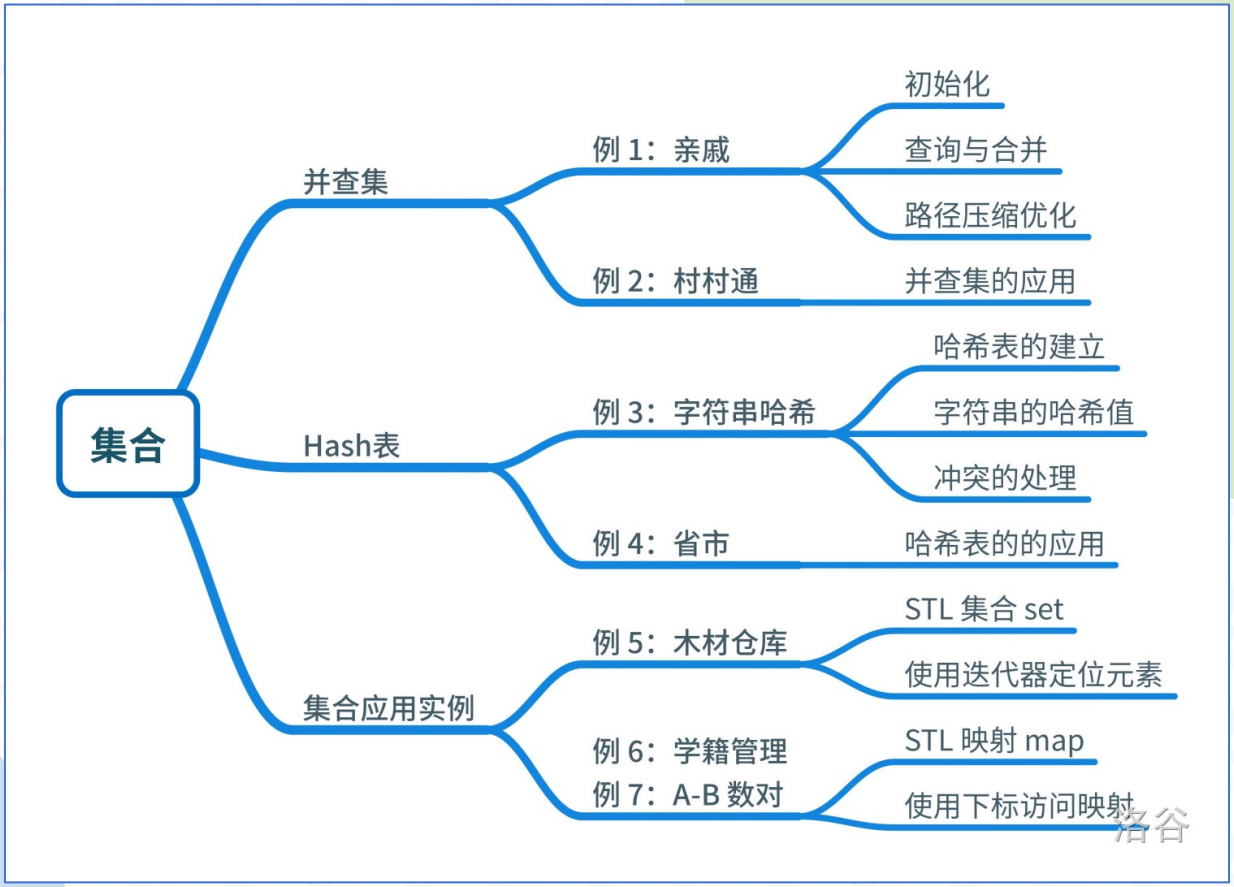
洛谷
www.luogu.com.cn

洛谷

版权声明

- 其它《深基》配套资源、购买本书等请参阅：https://www.luogu.com.cn/blog/kkksc03/IPC-resources
- 如果课件有任何错误，请在这里反馈 https://www.luogu.com.cn/discuss/show/296741

本章知识导图


集合
并查集
例 1：亲戚
初始化
查询与合并
路径压缩优化
例 2：村村通
并查集的应用
Hash表
例 3：字符串哈希
哈希表的建立
字符串的哈希值
冲突的处理
例 4：省市
哈希表的的应用
集合应用实例
例 5：木材仓库
STL 集合 set
使用迭代器定位元素
例 6：学籍管理
例 7：A-B 数对
STL 映射 map
使用下标访问映射

第 17 章 集合

- 并查集
- Hash表
- STL中的集合
- 课后习题与实验

集合：无序集

集合，数学中默认指无序集，用于表达元素的聚合关系。

两个元素只有属于同一个集合与不属于同一集合两种关系。

本章中将基于实例详细讲解。

常见应用场景：

- 两个元素是否属于同一个集合？
- 一个元素是否在集合中存在？

常见实现方式：

- std::unordered_set、std::unordered_map
- 并查集、哈希表
- 启发式可并堆

集合：偏序集

在实际应用中，可能需要关心集合元素顺序。

集合上定义偏序关系（即≤号），可构成一个偏序集。有序性作为重要规律，可引入算法（如二分法）提升运行效率。

将在未来的章节中介绍。

常见应用场景：

- 某个元素在的前驱/后继是什么？
- 插入/删除若干元素，使集合仍然有序？
- 某个元素是第几名？第几名是哪个元素？

小A 100分
八尾勇 98分
正妹 92分
Uim 59分

常见实现方式：

- std::set，std::map
- 二叉排序树（平衡二叉树）

并查集

第一个问题：“两个元素是否属于同一个集合？”

请翻至课本 P238

亲戚

例17.1 （洛谷 P1551）

如果 x 和 y 是亲戚，y 和 z 是亲戚，那么 x 和 z 也是亲戚。如果 x 和 y 是亲戚，那么 x 的亲戚都是 y 的亲戚，y 的亲戚也都是 x 的亲戚。现在给出某个亲戚关系图，求任意给出的两个人是否具有亲戚关系。

样例输入：

```
6 5 3
1 2
1 5
3 4
5 2
1 3
1 4
2 3
5 6
```

样例输出：

```
Yes
Yes
No
```

亲戚

观察样例的亲戚关系，1 的亲戚都可以通过某种方式追溯到 1。是否可以利用这种追溯关系，让 1 来代表 1 的整个家族呢？

将家族视为集合，若两人是亲戚，等价两个元素属于同一集合。

初始时，我们不知道任何人的亲属关系。可以认为每一个人单独属于一个集合，其代表为自己，如上图右。

每添加一组亲属关系，则等将于合并两者所属的集合。

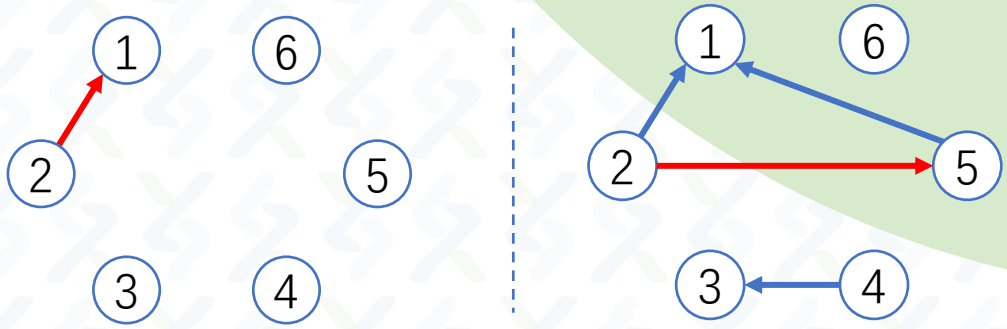
亲戚：“并”与“查”

考察第一组亲属关系<1,2>。

将 1 所属的集合 {1} 与 2 所属的集合 {2} 合并为 {1,2}。

此处，我们假定以左侧元素 1 作为这个集合的代表。那么 1 的代表仍是 1，2 的代表变为 1。

```
1 2
1 5
3 4
5 2
1 3
```



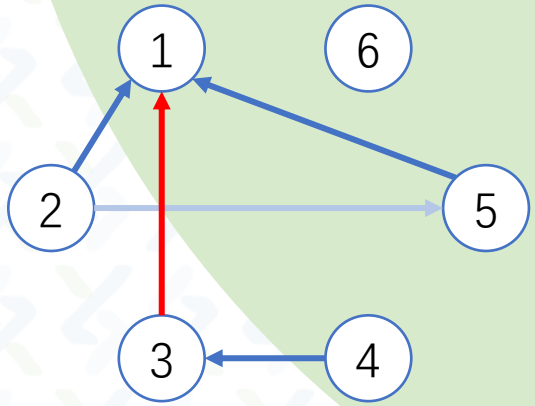
考察第四组亲属关系 <5,2>。

查询 2、5 的代表，发现代表均为1，意味他们已属于同一个集合。

因此，这一条关系并没有增加任何的信息，可以忽略。

亲戚：并查集

```
1 2
1 5
3 4
5 2
1 3
```



考察第五组关系 <1,3>。

此时 1、3 属于不同的集合 {1,2,5} 和 {3,4}，代表分别为 1、3。

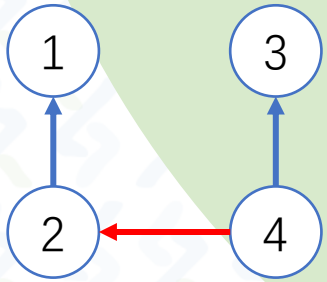
我们设置 3 的代表为 1。图中可以看到，代表关系具有传递性；4 的代表也随着 3 变为了 1。

这样一来，{3,4} 也并入了集合 {1,2,5}。

亲戚：细节

但是考虑下面一种情况：

```
1 2
3 4
2 4
```



我们期望的结果是 1、2、3、4 均有亲属关系。

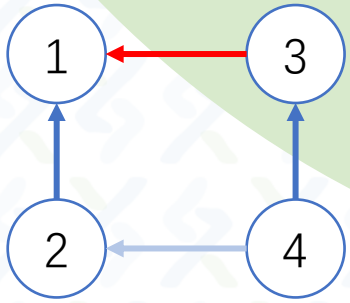
但若直接修改 4 的代表为 2，则丢失3、4 之间的关联，怎么办？

亲戚：细节

代表关系具有传递性。我们不需要直接修改2、4的代表，而是修改2和4的代表的代表，那么2、4的关系也会改变，同时所属集合中其他所有元素也随代表被更改！

因此，先查找 2 的代表为 1，4 的代表为 3。

```
1 2
3 4
2 4
```



然后合并，将3的代表设置为1。此时，4的代表也随着3变成了1。

这样，集合的合并与查询就都实现了，“并查集”因此得名。

亲戚

实现时，用数组 fa 来存储“代表”。

代表具有传递性。当查找代表时，需要递归地向上，直到代表为自身。

```
int find(int x) { // 查询集合的“代表”
    if (x == fa[x])return x;
    return find(fa[x]);
}
```

当合并两个元素时，需先判断两者是否属于同一集合。若否，则将其中一个集合的代表设置为另一方的代表。

```
void join(int c1, int c2) { // 合并两集合
    // f1为c1的代表，f2为c2的代表
    int f1 = find(c1), f2 = find(c2);
    if (f1 != f2) fa[f1] = f2;
}
```

本题为并查集最基础应用。套用模板即可解决。

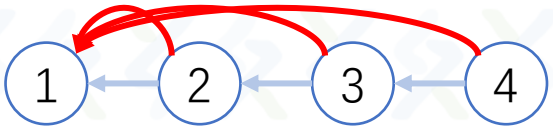
```
// 初始化
for (int i = 1; i <= n; ++i)
    fa[i] = i;
// 合并亲属关系
for (int i = 0; i < m; ++i) {
    cin >> x >> y;
    join(x, y);
}
// 根据代表是否相同，查询亲属关系
for (int i = 0; i < p; ++i) {
    cin >> x >> y;
    if (find(x) == find(y))
        cout << "Yes" << endl;
    else
        cout << "No" << endl;

}
```

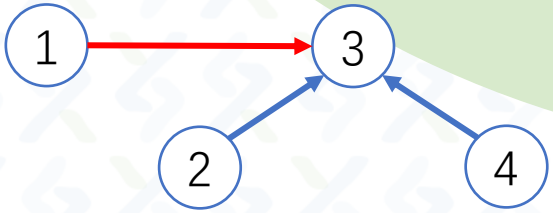
优化并查集

然而，寻找代表的过程可能需要多次溯源。随着集合合并的深度增加，溯源的次数会越来越多，严重影响效率。

这里，引入两种优化方式：路径压缩和启发式合并。

- 路径压缩（左图）
  即在查询完成后，将路径每一个元素的fa值直接更新为代表，使得下一次递归时，只需要一步即可找到代表。





- 启发式合并（右图）
  当合并两个集合时，选择较大的集合代表作为代表，即更改元素较少的集合的代表，可以减少路径压缩的次数。

优化并查集

经过优化后的并查集实现（可以用作模板）

其查询的时间复杂度接近常数。

```
// 一定不要忘了初始化，每个元素单独属于一个集合
void init() {
    for (int i = 1; i <= n; i++)
        f[i] = i;
}
int find(int x) { // 查询集合的“代表”
    if (x == fa[x])return x;
    return fa[x] = find(fa[x]); // 顺便【路径压缩】
}
void join(int c1, int c2) { // 合并两个集合
    // f1为c1的代表，f2为c2的代表
    int f1 = find(c1), f2 = find(c2);
    if (f1 != f2) {
        if (size[f1] < size[f2])  //  【取较大者】作为代表
            swap(f1, f2);
        fa[f2] = f1;
        size[f1] += size[f2];  // 只有“代表”的size是有效的
    }
}
```

洛谷

Hash表

第二个问题：“一个元素是否存在于这个集合中？”

请翻至课本 P42

字符串哈希

例 17.3 （洛谷 P3370）

给定 $N(N \leq 10000)$ 个字符串（第 i 个字符串长度为 $M_i(M_i \leq 1500)$，字符串内包含数字、大小写字母，大小写敏感），请求出 N 个字符串中共有多少个不同的字符串。

样例输入：

```
5
abc
aaaa
abc
abcc
12345
```

样例输出：

```
4
```

字符串哈希

任意两个字符串两两比较时间上必然是不可取的，时间复杂度。
时间只允许处理每一个字符串仅一次。

联想计数排序的思想，我们可以将每一个字符串装入一个投票箱，使得相同的投票箱里只有同一类字符串。

票箱#1	票箱#2	票箱#3	票箱#4	票箱#5
1	7	0	0	2

至于如何将字符串变为数字，就取决于Hash函数如何构造了。

先考虑一个简单的问题：

给定N个自然数，值域是 $[0, 10^{18}]$，求出这 N 个自然数中共有多少个不同的自然数。

简单的问题

将整数映射到较小的整数，不难想到一个经典的运算：取模。

原数	1	14	514	1919810	1145141919810
模 11	1	3	8	2	9

这里顺带介绍一下余和模的区别（选读）：

余数由除法定义：$r = q - \left[\frac{a}{q}\right] \times q$ 所得，符号与被除数相同。

模数由数轴划分：$m = \mathrm{q} - k\left[\frac{a}{q}\right]$ 所得，符号永远为正。

a	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4
余	-4	-3	-2	-1	0	-4	-3	-2	-1	0	1	2	3	4
模	1	2	3	4	0	1	2	3	4	0	1	2	3	4

可以注意到余数在正负方向表现不一致。所以我们一般使用模数。

哈希函数

理论表明，并不是任意选择 Hash 函数都能取得同样的效果。

1. 使用较大的质数作为模数

模数越大，空间越多，越难以冲突。

同时，由于质数除了1和自身外没有其他因子，包含乘除运算的 Hash 函数不会因为有公因子而导致不必要的 Hash 冲突。

2. 使用复杂的 Hash 函数

直接取模是最简单的方式。

但复杂的 Hash 函数可使值域分布更均匀，降低冲突的可能。

请注意Hash函数的输入应只与对象本身有关，而与随机数等任何外界环境无关。

简单的问题（并不简单）

模运算可以将数值折叠到一个小区间内。

但是还有一个问题，折叠之后，不同的数可能映射到同一个区域，这一现象称为 Hash 冲突。例如下面的情况，如何区分？

原数	4	−1	9
模 5	4	4	4

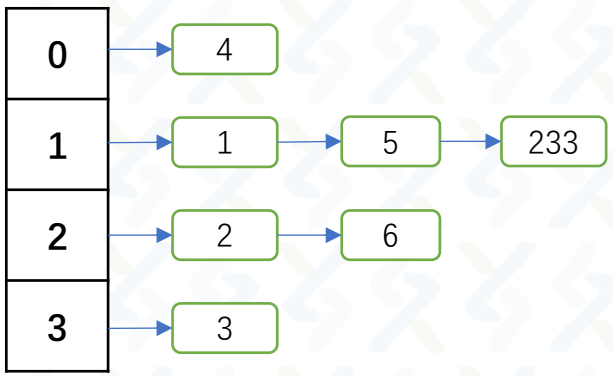
可以提出三种解决方法：

1. 使用稳健的 Hash 函数，效率最高，冲突率最高
2. 使用十字链表，完全解决冲突，效率较低
3. 使用 Multi-Hash，折中的方法

十字链表

使用链表（或 std::vector 等结构）将 Hash 冲突的元素保存起来。

这样，查找一个元素时只需要与 Hash 冲突的较少元素进行比较。




```
vector<long long> hash[maxh];

// 以下是插入集合的方式，查找也是类似
void int insert(x){
  int h = f(x);  //计算哈希值
  for (int i == 0, sz=hash[h].size(); i<sz; i++)
    if (x == hash[h][i]) // 从数组中找到了这一项
      return; // 什么都不做退出
  hash[h].push_back(x); // 插入这个元素
}
```

用这种方式可以完全解决 Hash 冲突问题。

但是查找元素的复杂度会有所上升（取决于 Hash 冲突的次数）。

Multi Hash （多哈希）

另一种解决方式是将映射f调整为高维，例如同时使用两个模数：

	0	1	2	3	4	5
0	1145141919810				1919810	
1		1				
2						
3						
4	14			514		

原数	1	14	514	1919810	1145141919810
模 5	1	4	4	0	0
模 7	1	0	3	4	0

此时，只有当多个Hash函数值同时相等才会导致Hash冲突。

冲突概率大幅降低。

注意Multi Hash对空间的开销较大，因为需要使用二维数组。

字符串哈希

该如何将字符串变成为整数编号呢？

回顾：字符在计算机中是以ASCII码（8位整数）的形式存储的。

所以可进行整数运算，把字符串视作超长的256进制高精度整数。

字符	L	u	o	g	u	4	!
ASCII	76	117	111	103	117	52	33

由于取模运算具有关于乘法的结合律和关于加法的分配率，可以构造出最简单的Hash函数：将字符串视作整数取模。

```
string s; // ......
int hash = 0;
for (int i = 0; s[i]; i++)
    // 计算base进制下模mod的值作为hash值
    hash = ((long long)hash * base + s[i]) % mod;
```

请同学回忆一下十六进制转十进制。

字符串哈希

计算哈希函数的 base 和 mod 根据经验选取。

- base 应当选择不小于字符集数的质数。例如，a-z 字符串为 26，任意 ASCII 字符串为 256。
- 而 mod 应该选取允许范围内尽可能大的质数。

```
int n, ans;
char s[MAXN];
vector <string> linker[mod + 2];

void insert();

int main() {
  cin >> n;
  for (int i = 1; i <= n; i++)
    cin >> s, insert();
  cout << ans << endl;
  return 0;
}
```

```
void insert() {
  int hash = 1;
  for (int i = 0; s[i]; i++)
    hash = (hash * 1ll * base + s[i]) % mod;
    //计算出字符串的hash值
  string t = s;
  for (int i = 0; i < linker[hash].size(); i++)
    //遍历hash值为该字符串hash值的链表，检查字符串是否存在
    if (linker[hash][i] == t)
      return; //如果找到同样的字符串，这个字符串不计答案
  linker[hash].push_back(t); //否则计入答案
  ans++;
}
```

这里取 base=261，mod=23333。

STL中的集合

第三个问题：“某个元素在偏序关系上的后继（或前驱）是什么？”
以及，可以使用 STL 简化编程的过程吗？

请翻至课本 P246

STL中的集合与映射

之前已经提到过，STL中也有集合的实现，分为无序集和偏序集。
其中分为 集合(set) 和 映射(map)。

- 无序集在 STL 中是 unordered_set 和 unordered_map。
  其本质为 Hash表，因此增删改查均为 O(1)。
  对于复杂数据类型，需要手动实现 Hash函数。
- 偏序集在 STL 中是 set 和 map。
  本质为排序树（将来介绍），增删改查均为 O(logn)。
  对于复杂数据类型，需要手动实现偏序关系，即 < 运算符。

STL中的集合

集合在 STL 中有两种，分别是 有序集合 和 无序集合

需头文件 <unordered_set>

unordered_set 的行为（无序）：

```
unordered_set<Type> s; //创建Type类型的集合

s.insert(x);     // 插入元素x
s.erase(x);      // 删除值为x的元素
s.erase(it);     // 删除it所指的元素
s.end();         // 返回末位哨兵的迭代器
s.find(x);       // 查询x；不存在则返回s.end()
s.empty();       // 判断是否为空
s.size();        // 返回集合的大小
```

需头文件 <set>

set 的行为（有序）：

```
set<Type> s;     // 创建一个Type类型的集合

s.insert(x);     // 插入元素x
s.erase(x);      // 删除值为x的元素
s.erase(it);     // 删除it所指的元素
s.end();         // 返回末位哨兵的迭代器
s.find(x);       // 查询x；不存在则返回s.end()
s.empty();       // 判断是否为空
s.size();        // 返回集合的大小

s.upper_bound(x);    // 查询大于x的最小元素
s.lower_bound(y);    // 查询不小于x的最小元素
// 使用方法与二分查找一章所介绍的一致
```

功能上类似，但有序集可找前驱后继

STL中的映射

映射在 STL 中也有两种，分别是 有序映射 和 无序映射

需头文件 <unordered_map>　　需头文件 <map>

unordered_map 的行为（无序）：　map 的行为（有序）：

```
unordered_map <T1, T2> m;  // 创建T1到T2的映射
// 其中T1称为键key，T2称为值value

m.insert({a,b});// 创建映射a->b
m.erase(a);     // 删除key为a的映射
m.erase(it);    // 删除it所指的映射
m.end();        // 返回末位哨兵的迭代器
m.find(x);      // 寻找键x；若不存在则返回m.end()
m.empty();      // 判断是否为空
m.size();       // 返回映射数目
m[a] = b;       // 修改a映射为b；若不存在则创建
```

```
map<T1, T2> m;  // 创建一个T1到T2的映射
// 其中T1称为键key，T2称为值value

m.insert({a,b});// 创建映射a->b
m.erase(a);     // 删除key为a的映射
m.erase(it);    // 删除it所指的映射
m.end();        // 返回末位哨兵的迭代器
m.find(x);      // 寻找键x；若不存在则返回m.end()
m.empty();      // 判断是否为空
m.size();       // 返回映射数目
m[a] = b;       // 修改a映射为b；若不存在则创建

m.upper_bound(x);   // 查询大于x的最小键
m.lower_bound(x);   // 查询不小于x的最小键
// 使用方法与二分查找一章所介绍的一致
```

木材仓库

例17.5 （洛谷 P5250）

仓库里面可存储各种长度的木材，但保证所有长度均不同。

作为仓库负责人，你有时候会进货，有时候会出货，因此需要维护这个库存。有不超过 100000 条的操作：

1. 进货，格式 1 Length：在仓库中放入一根长度为 Length（不超过 10^9) 的木材。如果已有相同长度的木材输出Already Exist。

2. 出货，格式 2 Length：从仓库中取出长度为 Length 的木材。若无刚好长度的木材，取出在库的和要求长度最接近的木材。
   如有多根木材符合要求，取出比较短的一根。
   输出取出的木材长度。如果仓库是空的输出Empty。

木材仓库

一句话题意：维护一个集合，支持插入、删除最接近元素。

最接近，即前驱、自身（若存在）或后继。可知应当使用偏序集。

- 使用 lower_bound 即可找自身或后继；
- 而其 prev 即为所求前驱。

```
set <int> ::iterator p, q;
q = s.lower_bound(length);
p = prev(q);  // 方法1
p = q; p--;   // 方法2；本质等价
```

iterator 是指集合元素的迭代器，参见课本例 15-13 P219。

```
set <int> ds;
set <int> ::iterator i, j;


i = ds.lower_bound(lenth);
j = i;
```

```
if (j != ds.begin()) --j;
// 需注意如果j是ds.begin()的话是不能--的
if (i != ds.end() && lenth - (*j) > (*i)-lenth)
    j = i;
//若i是end()则不能对i解引用
cout << (* j) << endl, ds.erase(j);
```

j 为前驱地址，i 为本身/后继地址，选择二者中与 length 接近者。

如果 i 是 s.begin()，则 prev() 或 -- 会得到错误的值，需特判。

学籍管理

例 17.6 （洛谷 P5266）

设计一个学籍管理系统，最开始数据为空，支持如下操作：

1. 插入与修改，格式 1 NAME SCORE：
   插入姓名为 NAME，分数为 SCORE 的学生。若已经有同名学生则更新他的成绩为 SCORE。如果成功插入或者修改输出OK。
2. 查询，格式 2 NAME：在系统中查询姓名为 NAME 的学生的成绩。若未找到这名学生则输出 Not found，否则输出该生成绩。
3. 删除，格式 3 NAME：在系统中删除姓名为 NAME 的学生信息。若未找到则输出Not found，否则输出 Deleted successfully。
4. 汇总，格式 4：输出系统中学生数量。

学籍管理

- 需要支持名字到分数的映射。使用 map；
- 不需取前驱后继，也可用 unordered_map。

```
map <string, int> ds;
string name;
```

系统自带字符串 string 类型 Hash 函数，无需自定义 Hash。需求 1-4 分别是 unordered_map 提供的 []、find、erase、size 。

```
if (opt == 1) {
    cin >> name >> num; ds[name] = num;
    cout << "OK" << endl;
} else if (opt == 2) {
    cin >> name;
    if (ds.find(name) != ds.end())
        cout << ds[name] << endl;
    else cout << "Not found" << endl;
}
```

```
else if (opt == 3) {
    cin >> name;
    if (ds.find(name) != ds.end()) {
        ds.erase(ds.find(name));
        cout << "Deleted successfully" << endl;
    }
    else cout << "Not found" << endl;
} else
    cout << ds.size() << endl;
```

在查找时如果直接用 m[a]，当 m[a] 不存在时，会自动创建为默认值（例如数值类型默认值是 0），不能“判断是否存在”。

必须要先使用 m.find(a) 确认不为 m.end() 后，方能返回结果。

学籍管理（不推荐的做法）

本题中由于分数不为0，可以直接用 if (m[a]) 判断。此时由于0项的存在，直接size是无法获得学生人数的，只能另开变量储存。

2020 年 1-2 次印刷教材的写法如下，答案正确但不推荐，将修正。

```
cin >> opt;
if (opt == 1) {
    cin >> name >> num;
    if (!ds[name]) ans++; // ans是当前学生个数
    ds[name] = num; // 这里对映射表name所对应的值修改为num
    cout << "OK" << endl;
} else if (opt == 2) {
    cin >> name;
    if (ds[name]) cout << ds[name] << endl;
    else cout << "Not found" << endl;
} else if (opt == 3) {
    cin >> name;
    if (ds[name]) {
        ans--;ds[name] = 0;
        cout << "Deleted successfully" << endl;
    }
    else cout << "Not found" << endl;
} else
    cout << ans << endl;
```

小贴士：关于STL

使用STL可以方便快捷地解决问题。

（可以看到，这两题讲题的PPT都比前面的题短！）

在有需要的情况下，尽量优先使用STL！

1. STL经过验证，避免手写出错；
2. STL经过优化，一般性能更好；
3. STL使用方便，可以快速过题。

不过STL的问题是：常数大所以慢！开启 O2 优化可提升。

课后习题与实验

学而时习之，不亦说乎。学而不思则罔，思而不学则殆。——孔子

请翻至课本 P250

复习

并查集

解决集合的合并问题，判断两个元素是否在同一个集合中
别忘了记得初始化

Hash（哈希）

将很大的数组或者字符串等元素，映射为较小整数

哈希冲突的解决：设计哈希函数、多维模数、十字链表

set / map

维护偏序集，排序，快速查找前驱后继

集合、映射之间的区别和关系

unordered_set / unordered_map

维护无序集，快速查找、修改元素

作业

实验题

1. 给出 3 种通过例 17.7 的解法。分析它们的复杂度。
2. 学籍管理的例题中，如果要求判断不能有两名同学的成绩相同，否则插入或者修改失败，该如何实现呢？

习题 17.2 保龄球（洛谷 P1918）

DL 在打保龄球，可以看到 $n(n \le 10^5)$个球道中每个球道的瓶子数 $a_i(a_i \le 10^9)$，各不相同。

现在想一次打掉 $M(M \le 10^9)$个瓶子，请问应该在哪个球道发球？询问次数不超过 10^5。

作业

习题 17.3 关押罪犯（洛谷 P1525，NOIP2010）

有 $n(n \le 2\times10^4)$ 名罪犯，$n(n \le 10^5)$ 对关系 (a,b,c) 分别代表罪犯 a 和罪犯 b 有矛盾值 c，代表如果这两个罪犯在同一个监狱就会产生 c 的矛盾值，否则不产生。

现在将这些囚犯放到 2 个监狱中，问最大的矛盾值最小是多少？

习题 17.4 集合（洛谷 P1621）

给你从 $A(A \le 10^5)$到 $B(B \le 10^5)$的整数。

一开始每个整数都属于各自的集合，每次选择两个属于不同集合的整数，如果这两个整数拥有大于等于P的公共质因数，则把它们所在的集合合并；直到没有可以合并的集合为止。

求最后有多少个集合。

作业

习题 13.6 程序自动分析 （洛谷 P1955，NOI 2015）

假设 $x_1, x_2, x_3 \ldots$ 代表程序中出现的变量，给定 n 个形如 $x_i = x_j$ 或 $x_i \neq x_j$ 的变量相等/不等的约束条件，请判定是否可以分别为每一个变量赋予恰当的值，使得上述所有约束条件同时被满足。

（补充选做）狡猾的商人（洛谷 P2294，HNOI2005）

一本账本，给定 m 段时间内的总收入，判断账本是否作假。

提示：

1. 若存在时间点 a、b、c 使得 (c-a)≠(b-a)+(c-b)，则必然有假。
2. 如果还是想不到，可以看看补充阅读~

补充阅读：带路径长度的并查集

并查集除了可以维护元素之间的聚类关系外，还可以维护距离。
只需要额外添加一个距离记录项即可；路径压缩时动态更新。

```
int find(int x) { // 查询集合的“代表”
    if (x == fa[x])return x;
    fa[x] = find(fa[x]);
    dist[x] += dist[fa[x]];
    return fa[x];
}

void join(int c1, int c2, int d) { // 合并两个集合
    // f1为c1的代表，f2为c2的代表
    int f1 = find(c1), f2 = find(c2);
    if (f1 != f2)
    {
        if (size[f1] < size[f2])  // 取较大者作为代表
            swap(f1, f2);
        fa[f2] = f1;
        dist[f2] = d;
        size[f1] += size[f2];  // 只有“代表”的size是有效的
    }
}
```

参考阅读材料

以下的内容限于课件篇幅未能详细阐述。如果学有余力，可自行翻阅课本作为扩展学习。

- P242 例 17.2：简单的并查集应用
- P245 例 17.4：利用字符串拼接进行Hash
- 习题 17.5、17.7、17.8、17.9

（太水的STL题没有必要都做哦）