算法基础——递归分治stl

邓丝雨

牛客竞赛
AC.NOWCODER.COM

分治

- 天下大势，合久必分，分久必合
- 孙子曰：凡治众如治寡，分数是也；斗众如斗寡，形名是也；三军之众，可使必受敌而无败者，奇正是也；兵之所加，如以碫投卵者，虚实是也。——《孙子兵法》

- 当我们求解某些问题时，由于这些问题要处理的数据相当多，或求解过程相当复杂，使得直接求解在时间上相当长，或者根本无法直接求出。对于这类问题，我们往往先把它分解成几个子问题，找到求出这几个子问题的解法后，再找到合适的方法，把它们组合成求整个问题的解法。如果这些子问题还较大，难以解决，可以再把它们分成几个更小的子问题，以此类推，直至可以直接求出解为止。这就是分治策略的基本思想。

递归的概念：

- 递归的定义是：
- ——参见递归！
- 实际上，递归的意思就是，一个函数在执行时再次调用函数“本身”。

```
int f(int x)
{
    if(x==0) return 1;
    return x*f(x-1);
}
```

```
int fib(int x)
{
   return (x <= 1)? 1: fib(x - 1) + fib(x - 2);
}
```

牛客竞赛
AC.NOWCODER.COM

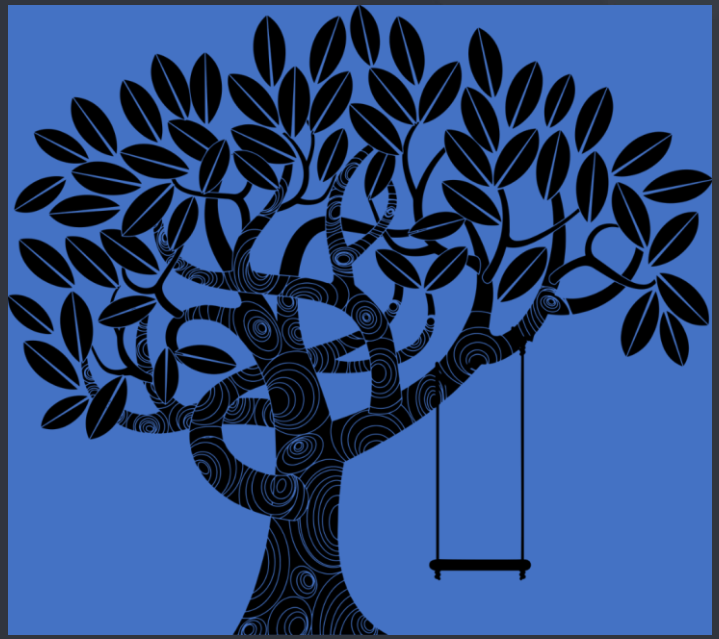
例1：汉诺塔问题

- 汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

- 求把所有圆盘从A柱移到C柱的最小步数 和 具体方案

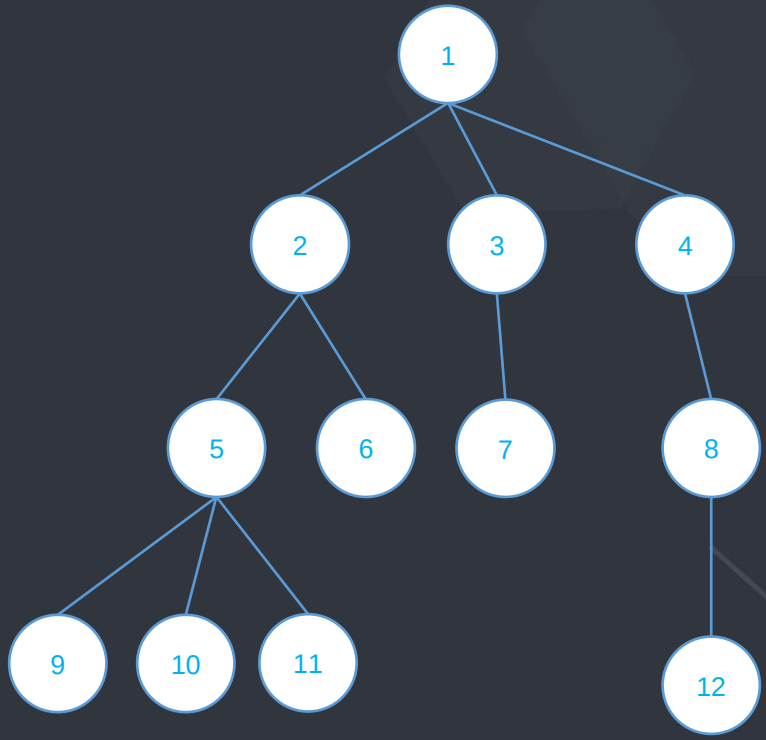
例2：给出二叉树的前序中序求后序

什么是树?



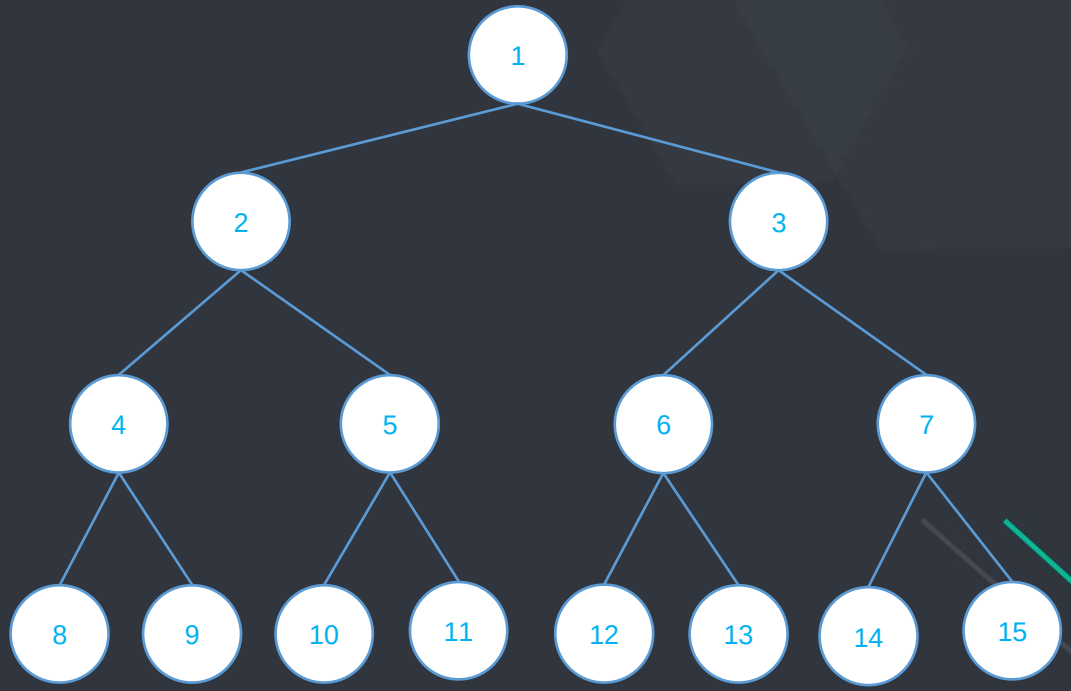
树

- 树是一种数据结构，它是由n（n>=1）个有限节点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。它具有以下的特点：
- 每个节点有零个或多个子节点
- 没有父节点的节点称为根节点
- 每一个非根节点有且只有一个父节点；除了叶子节点外，每个子节点可以分为多个不相交的子树



二叉树

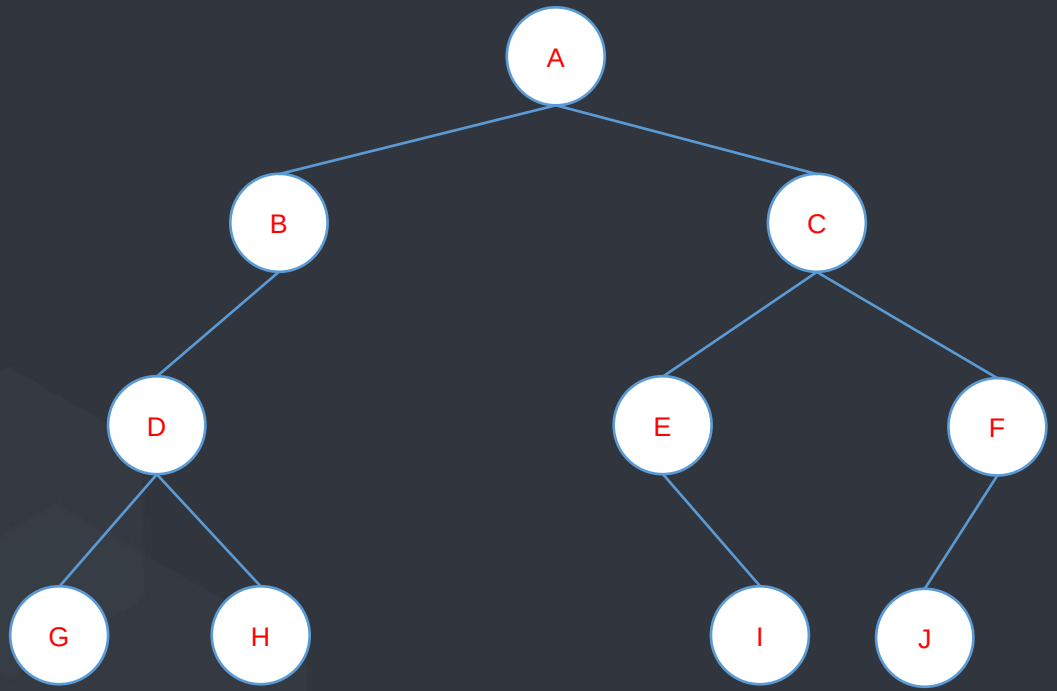
- 二叉树：每个节点最多含有两个子树的树称为二叉树；
- 满二叉树：除最后一层无任何子节点外，每一层上的所有结点都有两个子结点的二叉树。
- 完全二叉树：除最后一层外，每一层上的节点数均达到最大值；在最后一层上只缺少右边的若干结点。



二叉树的先序遍历、中序遍历、后序遍历和层序遍历

- 先序遍历也叫做先根遍历、前序遍历，首先访问根结点然后遍历左子树，最后遍历右子树。在遍历左、右子树时，仍然先访问根结点，然后遍历左子树，最后遍历右子树，如果二叉树为空则返回。
- 中序遍历首先遍历左子树，然后访问根结点，最后遍历右子树。在遍历左、右子树时，仍然先遍历左子树，再访问根结点，最后遍历右子树。
- 后序遍历首先遍历左子树，然后遍历右子树，最后访问根结点，在遍历左、右子树时，仍然先遍历左子树，然后遍历右子树，最后遍历根结点。
- 层序遍历就是按二叉树的每一层的顺序来遍历,也就是先访问根,然后访问第一层,接着访问第二层

二叉树的四种遍历



例2：已知前序中序求后序

Eg:前:ABCDFE 中:BADFCE

例3：归并排序&快速排序

- 三种O(n^2)的排序：冒泡、选择、插入
- 三种不基于比较的排序：桶、基数、计数

例3：归并排序&快速排序

- 归并排序：每次把待排序区间一分为二，将两个子区间排序，然后将两个已经排好序的序列合并
- 快速排序：选择一个基准，将小于基准的放在基准左边，大于基准的放在基准右边，然后对基准左右都继续执行如上操作直到全部有序。

归并

```
6   void _merge(int l, int mid, int r)
7   {
8       int p1 = l, p2 = mid + 1;
9       for (int i = l; i <= r; i++)
10      {
11          if ((p1 <= mid) && ((p2 > r) || (a[p1]) <= a[p2]))
12          {
13              b[i] = a[p1];
14              p1++;
15          }
16          else {
17              b[i] = a[p2];
18              p2++;
19          }
20      }
21      for (i = l; i <= r; i++) a[i] = b[i];
22  }
```

```
24  void erfen(int l , int r)
25  {
26      int mid = (l + r) >> 1;
27      if (l < r)
28      {
29          erfen(l , mid);
30          erfen(mid + 1, r);
31      }
32      _merge(l, mid, r);
33  }
```

快排

```
10  void quick_sort(int l, int r)
11  {
12      int i = l, j = r;
13      int mid = (l + r) / 2;
14      int x = a[mid];
15      while (i <= j)
16      {
17
18          while (a[i] < x) i++;
19          while (a[j] > x) j--;
20          if (i <= j)
21          {
22              swap(a[i],a[j]);
23              i++; j--;
24          }
25      }
26      if (l < j) quick_sort(l, j);
27      if (i < r) quick_sort(i, r);
28  }
```

求逆序对个数

- 给你一个序列，求出这个序列的逆序对个数
- 逆序对：对于一个包含N个非负整数的数组A[1..n]，如果有i < j，且A[i] > A[j]，则称(A[i],A[j])为数组A中的一个逆序对。

求一个序列的第k大数

例4：求最大子串和

- 给你一个序列，求其最大连续的一段的和
- （有动态规划解法，这里暂时不讨论）

例4：求最大子串和

- 把序列分成两部分：
- 左边的最大和
- 右边的最大和
- 左边包含最右点的最大和 + 右边包含最左点的最大和

```
10  int maxs(int n, int l, int r)
11  {
12      int mid = (l+r) / 2;
13      if(l == r) return a[l];
14      int ans = max(maxs(n, l, mid), maxs(n, mid+1, r));
15      int ll = 0, rr = 0, tmp = 0;
16      for(int i = mid + 1 ; i <= r; i++)
17      {
18          tmp += a[i];
19          rr = max(rr, tmp);
20      }
21      tmp = 0;
22      for(int i = mid ;i >= l; i--)
23      {
24          tmp += a[i];
25          ll = max(ll, tmp);
26      }
27      ans = max(ans, ll+rr);
28      return ans;
29  }
```

补充：倍增

- 倍增，从字面的上意思看就是成倍的增长 。指我们在处理一个问题的时候，如果问题的规模很大，线性递推无法满足时间和空间复杂度的要求 ，就可以通过成倍的增长，先求解状态空间中在 2 的整数次幂位置上的值作为代表，再对答案进行求解拼接。

求a的b次方模p的值

- **思路：** $a^{10} = a^{(1010)_2} = a^{1\times2^3+0\times2^2+1\times2^1+1\times2^0} = a^{1\times2^3} \times a^{1\times2^1}$
- **我们可以通过倍增的方法先计算模p意义下的a^1, a^2, a^4 ,a^8, a^{16}, a^{32}再把它们根据b的二进制拆分乘起来。**

STL

- 标准库STL，就是C++自身携带的，国际通用的，没有BUG的，老少皆宜的，令一众算法竞赛选手喜闻乐见的C++的附加产品。
- 主要分两大类：算法类（又称为操作类）和容器类。

算法类

- 最大值最小值（别看简单，C语言自身却没有这个）
  - template <class _Tp>
  - _Tp min(_Tp a, _Tp b)
  - _Tp max(_Tp a, _Tp b)
- 快速排序 sort(a, a + 10);
- 数据交换 swap(a, b);
- 求下一个排列 next_permutation(a,a+n)

容器类

- 字符串string
- 不限制长度的数组，vector
- 队列queue，堆栈stack，双端队列deque
- 优先队列（堆）priority_queue
- 简单红黑树（一种较高效率的平衡二叉树）set
- 通过键值来建立的平衡二叉树map
- 允许多个相同值的上述两种结构multiset和multimap
- 位集bitset
- http://www.cplusplus.com/

VECTOR

- 一般翻译成向量——借用了数学中n维空间下的向量的定义，由于不能确定n到底是多少，所以：
- VECTOR是一个不限制数组长度的数组！（最简单的理解方式）
- 如果学习过链表，那么你也可以理解成一个链表（稍复杂的理解）
- 实际上，这是一个看上去很像链表的顺序容器。
- 出于实战需要，还是理解成一个数组比较好（常数比数组大）

VECTOR实现了什么

- 首先是最基础的数据访问功能

  operator[], top(), back()

- 其次是简单的数据操作:

  删除erase(int index, int size)

  插入insert(int index, int size)

  添加到数据后面push_back(int value)

  弹出最后一个数据pop_back()

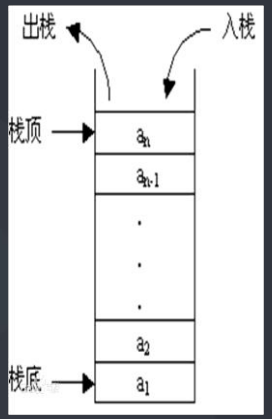
- 还有一些自身属性操作：数据大小和重新整理内存单元size(), resize(int size)
- 最后是整体的操作：清空clear(), 是否为空empty()

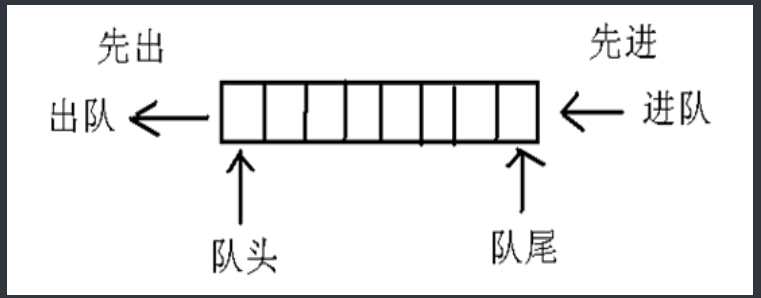
迭代器Iterator

- 定位：既能像指针一样访问数据，又能保证整体工程的稳定性
- 实际作用：用于访问一个容器内的数据的指针。
- 具体实现了什么?
- 返回第一个元素的迭代器begin()
- 返回容器末尾的迭代器end()（同样遵循左开右闭原则）

Stack 和Queue——栈和队列

- 栈的定义：栈是限定仅在表头进行插入和删除操作的线性表（先进后出）
- 队列的定义：队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。（先进先出）





- **stack常用函数：**
  - push() ——向栈顶压入元素
  - pop()——弹出栈顶元素
  - top()——访问栈顶元素
- **queue常用函数：**
  - front()——访问队首元素
  - back()——访问队尾元素
  - push_back()——向队尾插入元素
  - pop_front()——弹出队首元素

例：

- 有n个人，按照1,2,3,4……n的顺序依次进栈，判读能否以题目所给序列出栈。（n<=100000）
- Eg:
- 4 3 2 1
- 1 2 3 4
- 1 3 2 4
- 1 4 2 3

map

- map内部使用平衡二叉树实现
- 但是我们应用的时候可以把它理解成“自定义下标的数组”，或者是一个数对的集合。
- 而且它实际上是有序的

map

- map<string,int> mp;
- map<string,int>::iterator it;
- mp[“张三”]= 90;
- mp.insert(pair<string, int>(“李四”,95));
- mp.erase(“张三”);
- mp. clear(); mp.empty();
- mp.find(“张三”);
- mp.begin(); mp.end();

例1：

- 给你n个字符串，输出其中重复次数最多的字符串（n<=10000）

例2：

- 给你n个敌人的坐标，再给你m个炸弹和爆炸方向，每个炸弹可以炸横排或竖排的敌人，问你每个炸弹能炸死多少个人。(n<=100000,m<=100000)

```
7  typedef map<int,multiset<int> > line;
8  map<int,multiset<int> >mx;
9  map<int,multiset<int> >my;
10 int n, m;
11 int bomb(line &x, line &y, int pos)
12 {
13     int ans = x[pos].size();
14     multiset<int>::iterator it;
15     for(it = x[pos].begin(); it != x[pos].end(); it++)
16         y[*it].erase(pos);
17     x[pos].clear();
18     return ans;
19 }
```

```
22     while(scanf("%d%d", &n, &m) != EOF)
23     {
24         if(n == 0 && m == 0)break;
25         mx.clear();
26         my.clear();
27         for(int i = 0; i < n; i++)
28         {
29             int x, y;
30             scanf("%d%d", &x, &y);
31             mx[x].insert(y);
32             my[y].insert(x);
33         }
34         for(int i = 0; i < m; i++)
35         {
36             int x, y;
37             scanf("%d%d", &x, &y);
38             if(x == 0) ans = deal(mx, my, y);
39             else ans = deal(my, mx, y);
40             printf("%d\n", ans);
41         }
42         printf("\n");
43     }
```

例3：

- 有n个人，每个人有两个属性x，y，如果对于一个人P（x，y），不存在另外一个人(a，b)，使得a<x, b <= y 或者 a < = x, b < y, 则这个人是有优势的。
- 动态插入每个人，要求统计当前已经插入的人中， 有优势的人的个数。

- 先考虑静态的问题能不能解决

```
6   struct ty
7   {
8       int x,y;
9       bool operator < (const ty & a) const
10      {
11          return x < a.x || (x == a.x && y < a.y);
12      }
13  };
14  multiset <ty> st;
```

```
23      scanf("%d",&n);
24      st.clear();
25      for (int i = 1; i <= n; i++)
26      {
27          int x ,y;
28          scanf("%d%d", &x, &y);
29          ty p;
30          p.x = x;
31          p.y = y;
32          multiset <ty>::iterator it = st.lower_bound(p);
33          if(it == st.begin() || (--it) -> y > y)
34          {
35              st.insert(p);
36              it = st.upper_bound(p);
37              while(it != st.end() && it -> y >= y) st.erase(it++);
38          }
39          printf("%d\n", st.size());
40      }
```

例4：

- 我方和敌方的每一个军队都有一个攻击力属性和防御力属性，当一方军队的攻击力大于等于对方的防御力，就可以摧毁敌军（有可能双方同归于尽）。对战只允许单挑，且一个队伍只能上场一次，现给出我方和敌方所有军队的攻击力和防御力，问要摧毁敌方所有军队，最多能保留多少的存活的军队。(T<=100，n,m<=10^5)

- 按照敌方军队的防御从大到小排序。
- 这个时候肯定希望选一个能打死对方且方能存活能存活的军队去对阵，而且选能存活的里面防御最小的，防御更大的留给后面说不定会更好。如果我方不管上哪一个都不能存活，肯定是选一个防御最小的，把最容易死的用掉，存活的机会留给后面。

例5：滑动窗口（NC51001）

- 给定一个长度为n的数列，求长度为k的定长连续子区间 $\{a_1,a_2,a_3,a_4,\ldots,a_{k-1},a_k\}$ $\{a_2,a_3,\ldots,a_k,a_{k+1}\}$……中每个区间的最大值和最小值。

```
15      int l = 0;
16      int r = 1;
17      du[0] = 1;
18      if (m == 1) printf("%d ", a[1]);
19      for (int i = 2; i <= n; i++)
20      {
21          if (i - du[l] >= m && (l < r)) l++;
22
23          while (r > l && a[du[r - 1]] >= a[i]) r--;
24          du[r++] = i;
25
26          if (i >= m) printf("%d ", a[du[l]]);
27      }
```

Thanks

牛客竞赛
AC.NOWCODER.COM