

## 【可持久化线段树？！】rope史上最全详解

<https://www.luogu.org/problemnew/show/P3919>

看到上面链接中的题时，我在学会可持久化线段树的同时，第一次学会了一个非常屌（cai）的STL大法——**rope** **e!!!**

这是一个非标准的STL工具，一般情况下要支持c++11或更高才能用（上次去参加APIO时人家毛子评测用的是c++14啊喂！）

### 正题：

它的头文件是什么：#include<ext/rope>（注：你可以打开devcpp的目录去翻一翻rope这个头文件看看它的操作啊！）

除了头文件以外还需要什么：using namespace \_\_gnu\_cxx;（注：正是因为需要使用这种非std的标准命名空间，大部分竞赛才无法支持这个工具）

定义方法：rope<变量类型>变量名称;

或 crope 变量名称;

其中crope相当于定义成rope<char>，即定义为string类型

它到底是什么：

那得看你想听哪种解释了。

人话解释：超级string

算法解释：块状链表（即讲链表与数组的优势结合，形成分块思想）

用途解释：这本来是一个用于快速操作string的工具，却一般被定义成int，然后用作**可持久化线段树！**

### 它有哪些操作（重点）：

#### ●如果你把rope定义为string：

insert(int pos, string &s, int n) 将字符串s的前n位插入rope的下标pos处，如没有参数n则将字符串s的所有位都插入rope的下标pos处（补充地址知识：如果你不想从字符串下标为0（即第一个字符）的地址开始取n位，就将你想开始取的地址代入。如s+1表示从字符串下标为1（即第二个字符）的地址开始取n位。int、char等变量类型的数组都适用这种方法来更改数组操作的起始位置。）

示例代码：

```
1 char a[10];
2 for(int i=0;i<10;i++) a[i]=i+'0';
3 r.insert(0,a+1,8);
4 for(int i=0;i<10;i++) cout<<r.at(i);
```

append(string &s,int pos,int n) 把字符串s中从下标pos开始的n个字符连接到rope的结尾，如没有参数n则把字符串s中下标pos后的所有字符连接到rope的结尾，如没有参数pos则把整个字符串s连接到rope的结尾（这里已经给你起始位置参数pos了就没必要用上述的取地址方法了哈）

（insert和append的区别：insert能把字符串插入到rope中间，但append只能把字符串接到结尾）

substr(int pos, int len) 提取rope的从下标pos开始的len个字符

at(int x) 访问rope的下标为x的元素

erase(int pos, int num) 从rope的下标pos开始删除num个字符

copy(int pos, int len, string &s) 从rope的下标pos开始的len个字符用字符串s代替，如果pos后的位数不够就补足

replace(int pos, string &x); //从rope的下标pos开始替换成字符串x，x的长度为从pos开始替换的位数，如果pos后的位数不够就补足

以上是常用操作，不常用操作这里就不再赘述。

●如果你把rope定义为int（这里只是以int为例）：

insert(int pos, int \*s, int n) 将**int数组（以下的s都是int数组）**s的前n位插入rope的下标pos处，如没有参数n则将数组s的所有位都插入rope的下标pos处

append(int \*s, int pos, int n) 把数组s中从下标pos开始的n个数连接到rope的结尾，如没有参数n则把数组s中下标pos后的所有数连接到rope的结尾，如没有参数pos则把整个数组s连接到rope的结尾

substr(int pos, int len) 提取rope的从下标pos开始的len个数



at(int x) 访问rope的下标为x的元素

erase(int pos, int num) 从rope的下标pos开始删除num个数

copy(int pos, int len, int \*s) 从rope的下标pos开始的len个数用数组s代替，如果pos后的位数不够就补足

replace(int pos, int \*x); //从rope的下标pos开始替换成数组x，x的长度为从pos开始替换的位数，如果pos后的位数不够就补足

示例代码：

```

r.append(3);
r.append(1);
r.append(2);
r.append(1);
r=r.substr(1,3);
for(int i=0;i<r.size();i++) printf("%d ",r.at(i));

```

它有哪些好处：

**时间复杂度：** $O(n \cdot \sqrt{n})$ ，具体原理详见块状链表

**空间复杂度：** $O(\sqrt{n})$ （玄学），此处非常神奇，假如用rope存n个整数，它几乎只需要 $\sqrt{n}$ 的块空间加上一些链表指针的微小空间（个人猜测）。比如下面切的这道题就大方地开了100w个rope，每个rope都是一个存了100w个数的版本……我真是震惊了这东西真的这么省空间？

示范切题：以最上面那个链接中的题为例（可持久化线段树模板）

