1. 基本功能测试： 文件的复制粘贴功能，首先关键字“文件”，文件有不同的分类（图片、视频、音频、文档等），每个分类又有不同的类型（文档类型：txt doc execl pdf等），每个文件又有不同的大小，而且文件还有很多权限，是不是隐藏，是不是只是管理员可执行。选择不同分类的不同类型，不同大小的文件做测试资源。比如：文档类型里面txt文件可以分为 1.KB的txt文件、1MB的txt文件、1GB的txt文件。。。。下一个关键字 复制粘贴 复制有多种方式 右击选择、Ctrl+C、 拖动复制，对应粘贴也有各种方式。然后从哪复制，粘贴到哪，比如 可以有本机硬盘、移动硬盘、优盘、内存卡、软盘、光盘、连接手机存储，复制到网络地址等等。复制粘贴后文件是不是可用，文件权限是不是有变化。复制过去容量不够怎么处理？复制过后有重名文件怎么处理？复制过程中取消、关机、拔优盘怎么处理？复制过程能不能执行文件？

2.性能测试：复制粘贴功能性能怎么样？复制文件的速度可不可以接受？同时复制多个文件是不是可以完成?复制文件过程中占用CPU资源大不大，耗电量大不大？

3.兼容性测试 Windows XP, Windows 7, Windows 8 , Windows 8.1, Windows 10等各种windos版本是不是都支持这个功能。

4.交互测试; 复制粘贴文件时，使用windows存储的其他功能是否有影响？比如播放本地的音频、视频、等同时复制文件是不是有影响。一边复制，一边粘贴是不是有影响

作者：broccoli\_fur  
链接：[https://www.nowcoder.com/discuss/226311?type=2](https://www.nowcoder.com/discuss/226311?type=2" \t "_blank)  
来源：牛客网

### 字符串压缩解压：

普通模拟，用栈存下来左括号，每次遇到一个右括号，就把中间的字符串分析一遍，解压

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | string s;  struct p{      int l;      int r;      p(int a,int b){          l = a;          r = b;      }  };  stack<int> kuo;    int main(){        cin>>s;      int n = s.size();      for(int i=0;i<n;++i){            if(s[i] == '['){              kuo.push(i);          }          if(s[i] == ']'){              int t = kuo.top();              kuo.pop();              string a = s.substr(t+1, i-t-1);                bool flag = 0;              int num = 0;              string str;              for(int j=0;j<a.size();++j){                  if(!flag && isdigit(a[j])){                      num+=a[j]-'0';                      num\*=10;                  }                  else if(a[j] == '|'){                      flag = true;                      str = a.substr(j+1, n-j-1);                      break;                  }              }              num/=10;              string replace;              for(int j=0;j<num;++j){                  replace +=  str;              }              s = s.substr(0,t)+ replace + s.substr(i+1, n-i-1);              n = s.size();              i = t+replace.size()-1;            }      }      cout<<s<<endl;  } |

### 放多少个xxx可以覆盖整条河面  最小区间覆盖问题

先按照左值排序，然后右值倒叙

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | struct p{      int l;      int r;      bool operator<(const p & a)const{          if(l == a.l) return r>a.r;          return l<a.l;      }  } arg[100005];    int main(){      cin>>n>>l;      for(int i=0;i<n;++i){          cin>>arg[i].l>>arg[i].r;      }      sort(arg, arg+n);      if(arg[0].l>0){          cout<<-1<<endl;          return 0;      }      int right = 0, pos = 0, max\_len = 0;      int ans=0;      while(true){          if(right >= l){              break;          }          for(int i=pos;i<n;++i){              if(arg[i].l<=right && arg[i].r > max\_len){                  pos = i;                  max\_len = arg[i].r;              }              if(arg[i].l>right){                  break;              }          }          if(max\_len == right){              cout<<-1<<endl;              return 0;          }          right = max\_len;          ++ans;      }        cout<<ans<<endl;    } |

### 反转区间之后问还有多少逆序对，没时间写。。。

前提条件，对于一个个数位n的数组，正序对个数+逆序对个数 = 总对数= n\*（n-1）

大概可能的思路是，归并排序的同时计算逆序对个数，计算到正好是我们需要翻转的那一层，记下一个个数k。

最后算出总的逆序对个数ans之后，之前算的那部分逆序对正好是完全算错的， ans  -  2^(m-n)\*(2^n\*(2^n-1))  +  2\*k，中间部分为所有的对数，

### 看得到多少楼

预处理左边能看到的楼。右边能看到的楼，总的看到的数量是左边可以看到的楼+右边可以看到的楼+1

预处理使用栈，如果发现现在需要入栈的楼比栈顶高了，就将它出栈。直到可以放进去位置，此时可以看到的楼的数量为栈的总大小

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | int n;  int w[100005];  int l[100005],r[100005];    stack<int> stk;    int main(){      cin>>n;      for(int i=0;i<n;++i){          cin>>w[i];      }      for(int i=0;i<n;++i){          if(stk.empty()){              stk.push(w[i]);          }          else{              while(!stk.empty() && stk.top()<=w[i]){                  stk.pop();              }              stk.push(w[i]);          }          l[i] = stk.size();      }        while(!stk.empty()) stk.pop();        for(int i=n-1;i>=0;--i){          if(stk.empty()){              stk.push(w[i]);          }          else{              while(!stk.empty() && stk.top()<=w[i]){                  stk.pop();              }              stk.push(w[i]);          }          r[i] = stk.size();      }        for(int i=0;i<n;++i){          int ans = 1;          if(i>0){              ans+=l[i-1];          }          if(i<n-1){              ans+=r[i+1];          }          cout<<ans<<" ";      }cout<<endl;    } |

### 最少可以休息几天

动态规划，分成三种情况，0上班，1运动，2休息，如果目前这个情况不合法，那么设置为-1

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | int dp[100005][3];  int seq[100005][2];  int n;  // 0 work 1 sport 2 rest    int main(){      cin>>n;      for(int t = 0;t<2;++t) {          for (int i = 0; i < n; ++i) {              cin >> seq[i][t];          }      }      dp[0][0] = (seq[0][0] == 1) ? 0:-1;      dp[0][1] = (seq[0][1] == 1) ? 0:-1;      dp[0][2] = 1;      for(int i=1;i<n;++i){          if(seq[i][0]) {// want work              dp[i][0] = min(dp[i-1][1], dp[i-1][2]);              if(dp[i][0] == -1){                  dp[i][0] = dp[i-1][2];              }          }          else{              dp[i][0] = -1;          }          if(seq[i][1]) {// want work              dp[i][1] = min(dp[i-1][0], dp[i-1][2]);              if(dp[i][1] == -1){                  dp[i][1] = dp[i-1][2];              }          }          else{              dp[i][1] = -1;          }          int m = 99999999;          if(seq[i-1][0]){              m = min(m,dp[i-1][0]);          }          if(seq[i-1][1]){              m = min(m,dp[i-1][1]);          }          m = min(dp[i-1][2]+1, m+1);          dp[i][2] = m;        }        // 最后在dp[n-1][0],dp[n-1][1],dp[n-1][2]之中选出最小的不为-1【合法】的结果    } |