第1题 文本对齐

- 定义dp[i]为i-n个单词所组成的文本最优答案
- 将dp[1]-dp[n]全部设置为正无穷，dp[n+1]设置为0
- 考虑状态转移，枚举第二行从哪个位置开始即可
- 转移方程为dp[i]=min(dp[i],dp[j]+badness(i, j-1));
- 状态数为n，每个状态转移的复杂度也是n次的，总时间复杂度为O(n^2)
- 对于单词的长度进行前缀和预处理，在求解badness时即可O(1)求出
- 转移时记录好从哪开始另起一行即可求出最终的方案

```
const int N=2e5+10;
int n,inf=1e9,width;
int dp[N],sum[N],brek[N];
string s[N];
void init(){
    for (int i=1; i<=n; i++) {
        sum[i]=sum[i-1]+s[i].size();
    }
}
int badness(int l,int r){
    int len=sum[r]-sum[l-1];
    if (len>width) {
        return inf;
    }
    return pow((width-len), 3);
}
void solve(){
    memset(dp, 0x3f, sizeof dp);
    dp[n+1]=0;
    for (int i=n; i>=1; i--) {
        for (int j=i+1; j<=n+1; j++) {
            if (dp[i]<dp[j]+badness(i, j-1)) {
                dp[i]=dp[j]+badness(i, j-1);
                brek[i]=j;
            }
        }
    }
    cout<<dp[1];
    int cur=1;
    while (brek[cur]) {
```

```
        for (int i=cur; i<brek[cur]; i++) {
            cout<<s[i];
        }
        cur=brek[cur];
        cout<<'\n';
    }
}
```

第2题 活动安排

- 对于活动开始的节点，给他赋值属性1，活动结束的节点，赋值属性0
- 对所有的时间节点排序，然后从小到大遍历所有时间节点
- 当该时间节点是一个事件的开始，则资源数量+1，否则资源数量-1
- 遍历的过程中更新答案即可
- 时间复杂度瓶颈在于对节点的排序，时间复杂度为O(nlogn)

```
const int N=2e5+10;
int n,s[N],f[N];
void solve(){
    cin>>n;
    for (int i=1; i<=n; i++) {
        cin>>s[i]>>f[i];
    }
    vector<pair<int, int>> v;
    for (int i=1; i<=n; i++) {
        v.push_back({s[i],1});
        v.push_back({f[i],0});
    }
    sort(v.begin(), v.end());
    int mx=0,cnt=0;
    for (auto [x,id]: v) {
        if (id==0) {
            cnt--;
        }
        else cnt++;
        mx=max(mx,cnt);
    }
    cout<<mx;
}
```

程序测试结果

```
4
1 3
2 5
2 4
5 6
3Program ended with exit code: 0
```

```
3
1 2
1 2
2 3
2Program ended with exit code: 0
```

第 3 题 单位时间任务安排问题

- 贪心地去安排任务
- 惩罚最大的优先处理，惩罚相等的，截止时间最早的优先处理
- 排序的复杂度是O(nlogn)
- 给每个任务分配做的时间时，是找到小于等于当前任务截止时间的最大的那个，这个过程用set<int>来维护，用upper_bound()找到大于当前数的第一个，在它之前的则为小于等于它最大的（若它前面已经没有元素，则代表分配失败）。时间复杂度O(nlogn)，所以总时间复杂度是O(nlogn)

```
const int N=2e5+10;
int n,pos[N];
```

```cpp
set<int> s;
struct node{
    int ddl;
    int penalty,id;
    bool operator < (const node &b)const{
        if (penalty==b.penalty) {
            return ddl<b.ddl;
        }
        return penalty>b.penalty;
    }
}task[N];
void align(int x){
    int dead=task[x].ddl;
    auto it=s.upper_bound(dead);
    if (it==s.begin()) {
        return;
    }
    it=prev(it);
    pos[x]=*it;
    s.erase(it);
}

void solve(){
    cin>>n;
    for (int i=1; i<=n; i++) {
        s.insert(i);
    }
    for (int i=1; i<=n; i++) {
        cin>>task[i].ddl;
        task[i].id=i;
    }
    for (int i=1; i<=n; i++) {
        cin>>task[i].penalty;
    }
    sort(task+1, task+1+n);
    memset(pos, -1, sizeof pos);
    for (int i=1; i<=n; i++) {
        align(i);
    }
    int ans=0;
    for (int i=1; i<=n; i++) {
        if (pos[i]==-1) {
```

```
45                ans+=task[i].penalty;
46                cout<<"第"<<task[i].id<<"个任务超时完成\n";
47            }
48            else cout<<"第"<<task[i].id<<"个任务时刻"<<pos[i]<<"完成\n";
49        }
50
51        cout<<"总误时惩罚: "<<ans;
52    }
```

程序测试结果

```
4
1 1 3 4
7 8 9 10
第4个任务时刻4完成
第3个任务时刻3完成
第2个任务时刻1完成
第1个任务超时完成
总误时惩罚: 7Program ended with exit code: 0
```

```
4
2 2 2 1
16 7 8 2
第1个任务时刻2完成
第3个任务时刻1完成
第2个任务超时完成
第4个任务超时完成
总误时惩罚: 9Program ended with exit code: 0
```