二分查找用法

```
#include<iostream>
#include<algorithm>
using namespace std;
int a[10]={1,2,3,3,4,4};
int cmd(int a,int b){
    return a>b;
}
int main()
{
    cout<<lower_bound(a,a+6,3)-a<<endl;//返回数组中第一个大于或等于被查数的值的位置，第一个位置为 0 ,2
    cout<<upper_bound(a,a+6,3)-a<<endl;//返回数组中第一个大于被查数的值的位置 ，如果查最后一次出现，就是比他大的减一,4
    if(upper_bound(a,a+6,4)-a==6) cout<<-1<<endl;
    else cout<<upper_bound(a,a+6,4)-a<<endl;//没找到返回 a 加 6 的位置指针
    sort(a,a+6,cmd);
    cout<<lower_bound(a,a+6,3,greater<int>())-a<<endl;//小于等于 ,2
    cout<<upper_bound(a,a+6,3,greater<int>())-a<<endl;//小于 ,4
}
```

KMP 算法，求字符串 b 在 a 中所有位置

```
#include<stdio.h>
#include<string.h>
int kmp[2000000];
int la,lb,j=0,i;
char a[2000000],b[2000000];
int main()
{
    scanf("%s%s",a+1,b+1);
    la=strlen(a+1);
    lb=strlen(b+1);
    for(i=2;i<=lb;i++)//给 kmp[i]赋值，kmp 的值也可理解为到此数前后缀相等的个数
    {
        while(j&&b[i]!=b[j+1]) j=kmp[j];
        if(b[j+1]==b[i]) j++;
        kmp[i]=j;
    }
    j=0;
    for(i=1;i<=la;i++)//比较 a 与 b 串
    {
        while(j>0&&b[j+1]!=a[i]) j=kmp[j];
        if(b[j+1]==a[i]) j++;
        if(j==lb)
        {
```

```
                printf("%d ",i-lb);//题目要求下标从 0 开始
                j=kmp[j];
            }
        }
        return 0;
}
```

用 kmp 算法前后缀和性质（kmp 数组）求长度为几的前后缀和相同

```
#include<stdio.h>
#include<string.h>
int kmp[2000000];
int la,lb,j=0,i;
char a[2000000];
int ans[2000000],cnt=0;
int main()
{
        scanf("%s",a+1);
        la=strlen(a+1);
        for(i=2;i<=la;i++)//给 kmp[i]赋值，kmp 的值也可理解为到此数前后缀相等的个数
        {
            while(j&&a[i]!=a[j+1]) j=kmp[j];
            if(a[j+1]==a[i]) j++;
            kmp[i]=j;
        }
        int x=kmp[la];
        ans[cnt++]=la;
        while(x!=0)
        {
            ans[cnt++]=x;
            x=kmp[x];
        }
        for(i=cnt-1;i>=0;i--) printf("%d\n",ans[i]);
        return 0;
}
```

01 背包问题空间最小解法

```
for(i=0;i<n;i++)
{
        for(j=k;j>=weight[i];j--)
        {
            dp[j]=max(dp[j],dp[j-weight[i]]+luck[i]);
        }
}
```

//f(v)=max(f(v),f(v-weight[i])+luck[i]) 而 原 来 应 该 是 f(v)(i)=max(f(v)(i-1),f(i-1)(v-

weight[i])+luck[i]) 倒序把从 i 个的 i 省略了
问题 1 为什么二维 J 要从后往前：因为从前往后同一个元素会被重复统计
二维解法

```
for(i=1;i<=n;i++)
{
    for(j=0;j<=k;j++)
    {
        if(j<weight[i-1])
        {
            dp[i][j]=dp[i-1][j];
        }
        else
        {
            dp[i][j]=max(dp[i-1][j],dp[i-1][j-weight[i]]+luck[i]);
        }
    }
}
```

空间消耗大，容易 MLE

完全背包问题空间最小解法//dp[i][j] = max(dp[i-1][j], dp[i][j-w[i]]+v[i]) (j >= w[i])

```
for(i=0;i<n;i++)
{
    for(j=w[i];j<=m;j++)
    {
        dp[j]=max(dp[j],dp[j-w[i]]+v[i]);//与 01 背包不同，此为正向
    }
}
```

```
void merge(int a[],int left,int mid,int right)
{
    int b[200000];
    int i=left,j=mid+1,k=0;
    while(i<=mid&&j<=right)
    {
        if(a[i]<=a[j])
        {
            b[k]=a[i];
            k++;i++;
        }
```

```
            else
            {
                b[k]=a[j];
                k++;j++;
            }
        }
        if(j==right+1)
        {
            while(i<=mid)
            {
                b[k]=a[i];
                i++;k++;
            }
        }
        if(i==mid+1)
        {
            while(j<=right)
            {
                b[k]=a[j];
                j++;k++;
            }
        }
        for (j=0,i=left;j<k;i++,j++)
        {
            a[i]=b[j];
        }
}
void mergesort(int a[],int left,int right)
{
        if(left>=right) return ;
        int mid;
        mid=(left+right)/2;
        mergesort(a,left,mid);
        mergesort(a,mid+1,right);
        merge(a,left,mid,right);
}
二分查找，找到一个比 key 大的数
int Bsearch(int left,int right,int key)
{
        int mid;
        while(left<=right)
        {
            mid=(left+right)/2;
            if(a[mid][0]==key) return mid+1;
```

```
            else if(a[mid][0]<key)
            {
                left=mid+1;
            }
            else
            {
                right=mid-1;
            }
        }
        return left;
}
```

哈希表

```
typedef struct node{
        int key;
        int val;
        struct node *next;
}Hash,*Hashlist;
Hashlist hash[50005];
//散列函数
int hashlize(int key)
{
        return fabs(key%50005);//自主设定
}
//向哈希表中插入
void insert(int key,int val)
{
        Hashlist tmp=(Hashlist)malloc(sizeof(Hash));
        tmp->key=key;
        tmp->val=val;
        tmp->next=NULL;
        int x=hashlize(key);
        if(hash[x]==NULL) hash[x]=tmp;
        else
        {
            tmp->next=hash[x];
            hash[x]=tmp;
        }
}
//哈希表中查找
```

```
Hashlist find(int key)
{
	int x=hashlize(key);
	Hashlist tmp=hash[x];
	while(tmp!=NULL)
	{
		if(tmp->key==key) return tmp;
		else tmp=tmp->next;
	}
	return NULL;
}
//学校不用，leetcode 不保证为 NULL，因此需要
void clear()
{
	int i;
	for(i=0;i<50005;i++) hash[i]=NULL;
}
```

小根堆

```
int heap[3000000],heap_size=0;
void insert(int d)//加入堆，下标从 1 开始
{
	heap_size++;
	heap[heap_size]=d;
	int now=heap_size,next,tmp;
	while(now>1)
	{
		next=now/2;
		if(heap[now]>=heap[next]) return;
		tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
		now=now/2;
	}
	return;
}
int del()//删除并返回堆顶元素
{
	int res=heap[1];
	heap[1]=heap[heap_size];
	heap_size--;
	int now=1,next,tmp;
```

```
    while(now*2<=heap_size)
    {
        next=now*2;
        if(next<heap_size&&heap[next+1]<heap[next]) next++;
        if(heap[now]<=heap[next]) return res;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=next;
    }
    return res;
}
```

堆操作++

```
typedef struct node{
    int height;
    int num;
}LNode,*linklist;
LNode heap[3000000];
int heap_size=0;
void insert(int height,int num)//加入堆，下标从 1 开始
{
    heap_size++;
    heap[heap_size].height=height;
    heap[heap_size].num=num;
    int now=heap_size,next;
    LNode tmp;
    while(now>1)
    {
        next=now/2;
        if(heap[now].height>=heap[next].height) return;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=now/2;
    }
    return;
}
void del()//删除堆顶元素
{
    heap[1]=heap[heap_size];
    heap_size--;
    int now=1,next;
```

```
    LNode tmp;
    while(now*2<=heap_size)
    {
        next=now*2;
        if(next<heap_size&&heap[next+1].height<heap[next].height) next++;
        if(heap[now].height<=heap[next].height) return;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=next;
    }
    return ;
}
void delk(int k)//删除第 k 个种植的绿藤
{
    int i,goal;
    for(i=1;i<=heap_size;i++)
    {
        if(heap[i].num==k)
        {
            goal=i;
        }
    }
    heap[goal]=heap[heap_size];
    heap_size--;
    int now=goal,next;
    LNode tmp;
    while(now>1)//往上比较
    {
        next=now/2;
        if(heap[now].height>=heap[next].height) break;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=now/2;
    }
    while(now*2<=heap_size)//往下比较
    {
        next=now*2;
        if(next<heap_size&&heap[next+1].height<heap[next].height) next++;
        if(heap[now].height<=heap[next].height) return;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=next;
    }
    return ;
}
void fchange(int k,int x)//修改第 k 个种植的绿藤的高度，将其变为 x
{
```

```
    int i,goal;
    for(i=1;i<=heap_size;i++)
    {
        if(heap[i].num==k)
        {
            goal=i;
        }
    }
    heap[goal].height=x;
    int now=goal,next;
    LNode tmp;
    while(now>1)//往上比较
    {
        next=now/2;
        if(heap[now].height>=heap[next].height) break;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=now/2;
    }
    while(now*2<=heap_size)//往下比较
    {
        next=now*2;
        if(next<heap_size&&heap[next+1].height<heap[next].height) next++;
        if(heap[now].height<=heap[next].height) return;
        tmp=heap[now];heap[now]=heap[next];heap[next]=tmp;
        now=next;
    }
    return ;
}
```

堆+++ 返回第 K 小，建两个堆（前 K 个大顶堆，后面小顶堆））

求两数是否互质（返回最大公因数）

```
int judge(int a,int b)
{
    int tmp,c;
    if(a<b)
    {
        tmp=a;
        a=b;
        b=tmp;
    }
```

```
    while(b!=0)
    {
        c=a;
        a=b;
        b=c%b;
    }
    return a;//a 为最大公约数
}
```

一串数组分 k 段，各段最大值的最大值 最小是多少（二分法，给出答案区间分治）

```
#include<stdio.h>
int a[200000];
int judge(int mid,int divided_num,int n)
{
    int i,num=1,sum=0;
    for(i=0;i<n;i++)
    {
        sum=a[i]+sum;
        if(sum>mid)                    //从左到右将数组元素之和与 mid 比较，如是大于则再起一段，最后看段的大小
        {
            sum=a[i];
            num++;
        }
    }
    if(num>divided_num)                //若是段大于段数，则必然不和条件
        return 0;
    else
        return 1;

}
int merge(int left,int right,int divided_num,int n)//divided_num 为分段个数，n 为数组个数
{
    if(left>=right) return left;
    else
    {
        int mid=left+(right-left)/2;
        if(judge(mid,divided_num,n)==1) return merge(left,mid,divided_num,n);
        else    return merge(mid+1,right,divided_num,n);
    }
}
int main()
```

```
{
    int n,divided_num,i;
    int max=0,min=0;
    scanf("%d%d",&n,&divided_num);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
        max=max+a[i];
        if(a[i]>min) min=a[i];//这里的 Min 指的是单个数最大值
    }
    int ans=merge(min,max,divided_num,n);
    printf("%d",ans);
    return 0;
}
```

流水线调度问题 做 bi 前必须先做完 ai

Marvolo 和他的室友自然不会错过这个拿奖的机会。他们研究了一下规则后发现，一共有 n 个专业参加了活动，也就是有 2n 个外场摊位。

每个专业的两个摊位各持有印章的半个部分，路北的摊位有印章的左半部分，路南的有右半部分。

印章的获取顺序必须是先获得左半部分印章，再获得右半部分印章。Marvolo 和室友商量好，他们一个人只逛路北的摊位，一个人只逛路南的摊位。

一个人每次只能在一个摊位玩游戏。通过调查，他提前知道了玩完每个摊位的小游戏需要多少时间，现在只需要规划好一个逛摊位的方案，使得兑换大奖的时间最短（去的晚可能没了）。

```
#include<stdio.h>
#include<string.h>
int a[2000][2],b[2000][2];
int num[2000];
int cmp(const int *p1,const int *p2)
{
    return p1[0]-p2[0];
}
int x[2000],y[2000];
int main()
{
    int n,i,j;
    long long int sum=0;
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i][0]);
        x[i]=a[i][0];
```

```
        a[i][1]=i;
    }
    for(i=0;i<n;i++)
    {
        scanf("%d",&b[i][0]);
        y[i]=b[i][0];
        b[i][1]=i;
    }
    qsort(a,n,sizeof(a[0]),cmp);
    qsort(b,n,sizeof(b[0]),cmp);
//排序 例如设 n=4,( a1,a2,a3,a4 ) =( 3,4,8,10 ) 和( b1,b2,b3,b4 ) =(6,2,9,15 ),对这些 a 和 b
分类后的序列是( b2,a1,a2,b1,a3,b3,a4,b4 ) =( 2,3,4,6,8,9,10,15),
    i=0;j=0;int left=0,right=n-1,k=0;
    while(i!=n&&j!=n)
    {
        if(a[i][0]<b[j][0]&&a[i][1]!=-1)
        {
            num[left]=a[i][1];
            for(k=0;k<n;k++)
            {
                if(b[k][1]==a[i][1]) b[k][1]=-1;
            }
            a[i][1]=-1;
            i++;left++;
        }
        else if(a[i][0]>=b[j][0]&&b[j][1]!=-1)
        {
            num[right]=b[j][1];
            for(k=0;k<n;k++)
            {
                if(a[k][1]==b[j][1]) a[k][1]=-1;
            }
            b[j][1]=-1;
            j++;right--;
        }
        else if(a[i][0]<b[j][0]&&a[i][1]==-1) i++;
        else if(a[i][0]>=b[j][0]&&b[j][1]==-1) j++;
    }//到此为止为 johnson 算法 若最小是 a，置于数组左边，若最小是 b，置于数组右边，
若已被调度，转向下一数
/*σ1,σ2,σ3,σ4 是最优调度。
最小数是 b2， 置 σ4 = 2。
下一个最小数是 a1， 置 σ1 = 1。
接着的最小数是 a2,由于作业 2 已被调度,转向再下一个数 b1 。
作业 1 已被调度,再转向下一个数 a3,置 σ2 = 3。
```

```
最后剩σ3 是空的,而作业 4 还没调度,从而 σ3= 4*/
    sum=x[num[0]];
    i=1;j=0;
    while(j!=n)
    {
        if(i!=n&&i==j+1&&x[num[i]]>=y[num[j]])
        {
            sum=sum+x[num[i]];
            i++;j++;
        }
        else if(i!=n&&i>j+1&&x[num[i]]>=y[num[j]])
        {
            while(j<i-1&&x[num[i]]>=y[num[j]])
            {
                sum=sum+y[num[j]];
                x[num[i]]=x[num[i]]-y[num[j]];
                j++;
            }
        }
        else if(i!=n&&i>j&&x[num[i]]<y[num[j]])
        {
            while(i!=n&&x[num[i]]<y[num[j]])
            {
                sum=sum+x[num[i]];
                y[num[j]]=y[num[j]]-x[num[i]];
                i++;
            }
        }
        else if(i==n)
        {
            sum=sum+y[num[j]];
            j++;
        }
    }//如下图计算时间
    // 0 a1 3 a3     11 a4       21 a2 25
    //          3 b1 9 11 b3 20 21 b4                36 b2 38
    printf("%lld",sum);
    return 0;
}
```

矩阵连乘最多次/最少次

```
#include<stdio.h>
```

```
#include<string.h>
// 如果 i=j，m[i,j]=0
// 如果 i<j，m[i,j]=min{m[i,k]+m[k+1,j]+p(i-1)p(k)p(j)}  i<=k<j
long long int line[2000],row[2000];
long long int dp_min[2000][2000],dp_max[2000][2000];
int main()
{
    long long int n,i,x,j,k,min=9e18,max=-1;
    scanf("%lld",&n);
    for(i=0;i<=n;i++)//读取矩阵 n 个
    {
        scanf("%lld",&x);
        if(i==0) line[i]=x;
        else if(i==n) row[i-1]=x;
        else row[i-1]=line[i]=x;
    }
    for(x=1;x<n;x++)//动态规划
    {
        for(i=0;i<n-x;i++)
        {
            j=x+i;
            for(k=i;k<j;k++)
            {
                if(dp_min[i][k]+dp_min[k+1][j]+line[i]*row[j]*row[k]<min)
                {
                    min=dp_min[i][k]+dp_min[k+1][j]+line[i]*row[j]*row[k];
                }
                if(dp_max[i][k]+dp_max[k+1][j]+line[i]*row[j]*row[k]>max)
                {
                    max=dp_max[i][k]+dp_max[k+1][j]+line[i]*row[j]*row[k];
                }
            }
            dp_min[i][j]=min;
            dp_max[i][j]=max;
            min=9e18;
            max=-1;
        }
    }
    printf("%lld\n",dp_min[0][n-1]);
    printf("%lld\n",dp_max[0][n-1]);
    return 0;
}
```

最优二叉搜索树

```
#include<stdio.h>
long long int a[1000],b[1000];
long long int dp[2000][2000],w[2000][2000];
int main()
{
	long long int n,i,j,l,r,sum;
	scanf("%lld",&n);
	for(i=1;i<=n;i++) scanf("%lld",&a[i]);
	for(i=0;i<=n;i++) scanf("%lld",&b[i]);
	for(i=1;i<=n+1;i++) dp[i][i-1]=w[i][i-1]=b[i-1];
	for(l=1;l<=n;l++)
	{
		for(i=1;i<=n-l+1;i++)
		{
			j=i+l-1;
			dp[i][j]=9e18;
			w[i][j]=w[i][j-1]+b[j]+a[j];
			for(r=i;r<=j;r++)
			{
				sum=dp[i][r-1]+dp[r+1][j]+w[i][j];
				if(sum<dp[i][j]) dp[i][j]=sum;
			}
		}
	}
	for(i=0;i<=n;i++) dp[1][n]=dp[1][n]-b[i];//这一句不是最优二叉搜索树，其余完全一致
	printf("%lld",dp[1][n]);
	return 0;
}
```