C++

day3 SH-B260103 集训-讲解

T1 投资方案

题 意

已知n个整数的序列，求最大子段和的方案计数。

部分分策略讨论

	测试点编号	特殊性质
特殊情况	1	所有x[i]均为正数
特殊情况	2	所有x[i]均为负数
特殊情况	3	所有x[i]均相同
$O(n^2)$暴力	4~6	n≤5000，x[i]的绝对值均不超过100
正解	7~10	无

数据分流

```
75  int main(){
76      freopen("invest.in","r",stdin);
77      freopen("invest.out","w",stdout);
78      input();
79      if(n<=5000)
80          solveBF();
81      else if(isPositive())
82          solvePositive();
83      else if(isNegative())
84          solveNegative();
85      else if(isIdentical())
86          solveIdentical();
87      else
88          solve();
89      return 0;
90  }
```

```
 3 typedef long long ll;
 4 const ll INF=1e16;
 5 const ll N=100009;
 6 ll x[N];
 7 ll n;
 8 void input(){
 9     cin>>n;
10     for(ll i=1;i<=n;++i)cin>>x[i];
11 }
```

暴力：枚举左右端点

```
12 void solveBF(){
13     ll ans=-INF;
14     ll cnt=0;
15     for(ll i=1;i<=n;++i){
16         ll sum=0;
17         for(ll j=i;j<=n;++j){
18             sum+=x[j];
19             if(sum==ans) ++cnt;
20 
21 
22 
23             }
24         }
25     }
26     cout<<ans<<" "<<cnt<<endl;
27 }
```

特判：全正数

```
28 bool isPositive(){
29     for(ll i=1;i<=n;++i)
30         if(x[i]<=0) return 0;
31     return 1;
32 }
33 void solvePositive(){
34     ll ans=0;
35     for(ll i=1;i<=n;++i) ans+=x[i];
36     cout<<ans<<" "<<1<<endl;
37 }
```

特判：全负数

```
38 bool isNegative(){
39     for(ll i=1;i<=n;++i)
40         if(x[i]>=0) return 0;
41     return 1;
42 }
43 void solveNegative(){
44 
45 
46 
47 
48 
49 }
```

特判：全相同

```
50 bool isIdentical(){
51     for(ll i=2;i<=n;++i)
52         if(x[i]!=x[i-1]) return 0;
53     return 1;
54 }
55 void solveIdentical(){
56     
57 }
```

排除全正和全负
只可能全0

正解

枚举两个端点，超时了怎么办？

考虑：减少枚举对象

只枚举右端点

快速计算合法的左端点

s[i]表示x数值前i个元素的总和
f[i]表示以i号结尾的最大子段和

输入
4
1 -1 2 3
输出
5 2

```
   i = 0  1  2  3  4
x[i] =    1 -1  2  3
f[i] =    1  0  2  5
s[i] = 0  1  0  2  5
```

输入
6
2 -3 1 -1 2 3
输出
5 2

```
   i = 0  1  2  3  4  5  6
x[i] =    2 -3  1 -1  2  3
f[i] =    2 -1  1  0  2  3
s[i] = 0  2 -1  0 -1  1  4
```

x数组中编号在[l,r]范围内的数值连续和 `s[r] - s[l-1]`

已知最大子段和 A，固定子段右端点r 思考：如何计算得到最大字段和?

跑路算法 f[i]：以i结尾的最大子段和 `f[i] = max(f[i-1],0) + x[i]`

最大子段和为 f[]数组的最大值，假设为**ans**

思考：如何确定符合条件的子区间?

`if(f[i] == ans)` 说明以i结尾的子区间是符合条件的

找到该子区间的左端点 `s[l] = s[i] - ans` 累加左端点(该前缀和数值)出现的数量

思考：如何记录?

```
map<ll,ll> valueCnt;
```

`valueCnt[s[i]]++`

所有可能成为符合条件的子区间的左端点之前位置的前缀和数值出现的次数。

也可以使用**unordered_map**

```
60 void solve(){
61     for(ll i=1;i<=n;++i)
62         f[i]=max(f[i-1],0LL)+x[i];
63     ll ans=*max_element(f+1,f+1+n);
64     for(ll i=1;i<=n;++i) s[i]=s[i-1]+x[i];
65     map<ll,ll> valueCnt;
66     
67     ll cnt=0;
68     for(ll r=1;r<=n;++r){
69         if(
70 
71 
72     }
73     cout<<ans<<" "<<cnt<<endl;
74 }
```

f[i]：初始化，空前缀 s[0] = 0 出现1次（对应左端点 l=1 的情况）

枚举每一个区间的右端点

判断当前位置是否可能为满足要求的子区间右端点。

符合条件的子区间左端点出现的次数（前缀数值相等）

当前位置出现过(前缀和)，记录下来.
有可能是后续符合要求的区间左端点

复杂度分析
O(nlogV)

V为数值总和的范围

最大子段和问题
有O(n)解法

最大子段和方案计数问题
有没有O(n)解法?

```
ll f[N];    //以第 i 个元素结尾的最大子段和
ll cnt[N];  //以第 i 个元素结尾且和为 f[i] 的子数组个数
```

1、f[i-1] < 0 时 f[i] = x[i] 则 cnt[i]=1

2、f[i-1] > 0 时 f[i] = f[i-1] + x[i] 则 cnt[i]= cnt[i-1]

3、f[i-1] == 0 时

f[i] = f[i-1] + x[i] 和 f[i] = x[i] 是等效的，但多一种方案

cnt[i]= cnt[i-1] + 1

最终统计 根据最大子段和MAX，累加所有 f[i]==MAX时的 cnt[i]的和。

f[i]表示以i号结尾的最大子段和
c[i]表示以i号结尾的最大子段和的方案数

```
14  for(ll i=1;i<=n;++i){
15      if(f[i-1]<0){
16
17
18      }
19      else if(f[i-1]>0){
20
21
22      }
23      else{
24
25
26      }
27  }
```

f[i]表示以i号结尾的最大子段和
c[i]表示以i号结尾的最大子段和的方案数

```
28      ll ans=*max_element(f+1,f+1+n);
29      ll cnt=0;
30      for(ll r=1;r<=n;++r){
31          if(f[r]==ans)
32              cnt+=c[r];
33      }
34      cout<<ans<<" "<<cnt<<endl;
```

T2 记单词

理解题意

单词内部复杂度 每个子单词相邻字母ASCII码之差的绝对值之和

拆分带来的额外复杂度 每多拆出一个单词，就会增加当前单词数量的复杂度

即拆出第k+1个单词时，增加k点复杂度

如果拆分成k个单词，则额外的时间复杂度为：0+1+2+3+(k-1) = k*(k-1)/2

总复杂度 = 单词内部复杂度 + 拆分带来的复杂度

定义 原单词的复杂度为 **S** 拆分额外复杂度为：**k*(k-1)/2**

拆分后减少的差值为 **V** 则拆分后的单词内部复杂度为： **S - V**

总复杂度 = S – V + k*(k-1)/2

目标： **总复杂度最小** S – V + k*(k-1)/2 —> S + (k*(k-1)/2 – V

s为常数，则 **k*(k+1）/2 – V** 要最小

贪心策略

贪心策略

相邻字符差值绝对值

将每个可拆分位置的差值vi从大到小排序，依次考虑每个位置。

贪心：按照间隔从大到小插入空格

什么时候停止？

目标： 总复杂度最小， k*(k+1）/2 - V 要最小

vi是按从大到小排序后的差值

k*(k+1)/2 – V 可以拆分为： (1-v[1]) + (2- v[2]) +…+(k-v[k])

$\sum_{i=1}^{k}(i - v_i)$

所以：对于每一次拆分，要求净收益为(i-v[i])要小于0

```
5   struct letter
6   {
7       int v, id;
8   }a[maxn];
```

记录间隔的值v和位置id

```
bool cmp(letter a, letter b)
{
    return
}
```

```
return a.v > b.v || (a.v == b.v && a.id < b.id);
```

```
int n;
string s;
```

n表示间隔的数量

```
cin >> s;
n = s.size() - 1;
for (int i = 1; i <= n; i++)
    a[i].v = abs(s[i] - s[i - 1]), a[i].id = i;
sort(a + 1, a + n + 1, cmp);
```

如何插入空格？

假设要在abcde的bc和de之间插入空格

```
s.insert(2, " ");
s.insert(4, " ");
```

结果为？

ab c de	为什么？

如何修改？

假设要在abcde的bc和de之间插入空格

```
s.insert(4, " ");
s.insert(2, " ");
```

从后往前处理，依次插入

```
vector<int> b; //用于记录需要加空格的位置

for (int i = 1; i <= n; i++)
{
    if (                ) break;
    b.push_back(a[i].id);
}
sort(b.begin(), b.end());
reverse(b.begin(), b.end()); //从大到小排序插入空格的位置
for (int i = 0; i < b.size(); i++) s.insert(b[i], " ");
cout << s;
return 0;
```

T3 新生入学

请同学写出题目大意
已知什么求什么

共n人，i号人停留时间[Li,Ri]，每次最多接走k人，要全部接走，求最少接几次?

部分分策略讨论

【数据规模与约定】

送分

1号数据：k=1

简化问题启发思路

2号数据：k=2

送分

3号数据：对于所有i：L[i]=R[i]

简化问题启发思路

4号数据：对于所有i：R[i]-L[i]均为10

5-10号数据：k<=n<=100000，L[i]和R[i]均不超过1000000000

数据分治
剥离出小型数据
剥离出特色数据

```
65  int main(){
66      freopen("shuttle.in","r",stdin);
67      freopen("shuttle.out","w",stdout);
68      cin>>n>>k;
69      for(int i=1;i<=n;i++)cin>>L[i]>>R[i];
70      if(k==1)
71          cout<<n<<endl;
72      else if(isLReq())
73          solveLReq();
74      else
75          solve();
76      return 0;
77  }
```

对于所有i：L[i]=R[i]

```
46 bool isLReq(){
47     for(int i=1;i<=n;i++)
48         if(L[i]!=R[i]) return 0;
49     return 1;
50 }
```

对于所有i：L[i]=R[i]

```
51  void solveLReq(){
52      sort(L+1,L+1+n);
53      int ans=0;
54      int cnt=1;
55      for(                              ){
56          if(L[i]==L[i-1]){
57                                        
58              continue;
59          }
60                                        
61                                        
62      }
63      cout<<ans<<endl;
64  }
```

正解：贪心

每次干什么？

每次挑选

已经出现的人里截止日期最早的k个人

安排接走

可能不足k人

思考：如何管理到达和离开的数据？

01信息的标记

```
#define ARRIVAL 0
#define DEPARTURE 1
```

到达

离开

按照事件发展的先后排序

```
6  struct Event{
7      int t,id;
8      bool tag;
9  } events[N*2];
```

到达/离开

```
10 bool cmp(const Event&a,const Event&b){
11     return a.t<b.t||a.t==b.t&&a.tag<b.tag;
12 }
```

按时间顺序从小到大

时间相同时，**到达**的事件排在前面

按照事件发展的先后顺序扫描

```
24      for(int i=1;i<=n;i++){
25          events[i]=(Event){L[i],i,ARRIVAL};
26          events[i+n]=(Event){R[i],i,DEPARTURE};
27      }
28      sort(events+1,events+1+n*2,cmp);
```

同一个人的到达和离开为两个独立的事件

排序的目的是什么？

后续如何处理数据？逻辑是什么？

贪心算法中要找截止日期最早的人

```
13 struct Departure{
14     int t,id;
   bool operator<(const Departure&a)const{
16     return t>a.t;
17     }
18 };
19 priority_queue<Departure> q;
```

管理离开事件

重载小于号

离开时早的排在前面

优先队列　小根堆

```
29      int ans=0;
30      for(int i=1;i<=n*2;++i){
31          if(events[i].tag==ARRIVAL){ 到达事件
32              int id=events[i].id;
33                  [                                                ]  将对应的离开事件入队
34                  [              ]
35          }
36          if([                    ])continue;  如果已经离开，则跳过
37          ++ans;  增加一次发车
38          [                                                ]  最多一次转运k个人
39          [                                                ]
40          [                                                ]
41          [                                                ]
42          }
43      }
```

T4 罗密欧与朱丽叶3

手算样例

输入
2 2
oJ
R#

输出多少
1.0

输入
4 5
ooo#J
o#o#o
o#o#o
R#ooo

输出多少
6.5

棋盘格里两人同时移动
每秒可以移动一格
求最少几秒相遇

固定一人不懂
只让另一人移动

起点和终点给定
求最短路长度
结果再除以2

单源单汇最短路问题

每一步耗时都是1

无权图最短路问题
用BFS求解

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

0				

dst[x][y]代表从起点到
(x,y)的最短路长度

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

1				
0				
1				

dst[x][y]代表从起点到
(x,y)的最短路长度

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

1	2			
0				
1	2			
2				

dst[x][y]代表从起点到
(x,y)的最短路长度

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

1	2	3		
0				
1	2	3		
2				
3				

dst[x][y]代表从起点到
(x,y)的最短路长度

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

1	2	3	4	
0		4		
1	2	3		
2		4		
3	4			

dst[x][y]代表从起点到
(x,y)的最短路长度

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

1	2	3	4	5
0		4		
1	2	3		
2		4	5	
3	4	5		

dst[x][y]代表从起点到
(x,y)的最短路长度

无权图最短路问题
用BFS求解

o	o	o	o	o
J	#	o	#	o
o	o	o	#	o
o	#	o	o	R
o	o	o	#	o

从J到R求最短路

BFS时第一次访问
就能确定最短路结果

1	2	3	4	5
0		4		6
1	2	3		
2		4	5	6
3	4	5		

dst[x][y]代表从起点到
(x,y)的最短路长度

```
31      cin>>n>>m;
32      for(int i=1;i<=n;i++)
33          for(int j=1;j<=m;j++){
34              cin>>d[i][j];
35              if(d[i][j]=='R')xR=i,yR=j;
36              else if(d[i][j]=='J')xJ=i,yJ=j;
37          }
38      int len=bfs();
39      if(len==-1)cout<<"forever"<<endl;
40      else cout<<fixed<<setprecision(1)<<len/2.0<<endl;
```

vst[x][y]代表第x行第y列是否访问过

dst[x][y]代表从起点到第x行第y列的最短路长度

```
int bfs(){
    queue<Node> q;
    vst[xR][yR]=1;
    dst[xR][yR]=0;
    
    while(!q.empty()){
        Node now=q.front(); q.pop();
        for(int k=0;k<4;k++){
            int nx=now.x+dx[k],ny=now.y+dy[k];
            if(nx>=1&&nx<=n&&ny>=1&&ny<=m&&d[nx][ny]!='#'&&!vst[nx][ny]){
                vst[nx][ny]=1;
                dst[nx][ny]=
                q.push((Node){nx,ny});
                if(nx==xJ&&ny==yJ)return 
            }
        }
    }
    return -1;
}
```

双向BFS

```
10      queue<Node> q[3];
11      vst[xR][yR]=1; vst[xJ][yJ]=2;
12      dst[xR][yR]=0; dst[xJ][yJ]=0;
13      q[1].push((Node){xR,yR}); q[2].push((Node){xJ,yJ});
14      while(!q[1].empty()&&!q[2].empty()){
15          int o=q[1].size()<q[2].size()?1:2;
16          Node now=q[o].front(); q[o].pop();
17          for(int k=0;k<4;k++){
18              int nx=now.x+dx[k],ny=now.y+dy[k];
19              if(nx>=1&&nx<=n&&ny>=1&&ny<=m){
20                  if(vst[nx][ny]+o==3)return dst[now.x][now.y]+dst[nx][ny]+1;
21                  if(!vst[nx][ny]&&d[nx][ny]!='#'){
22                      vst[nx][ny]=o;
23                      dst[nx][ny]=dst[now.x][now.y]+1;
24                      q[o].push((Node){nx,ny});
25                  }
26              }
27          }
28      }
```

T5 杂草种子

题 意

01二维表格里,若矩形3个角已有1,剩下角会自动补1,
求人工加几个1可以自动填满表格

请同学阅读[数据规模和约定]
识别部分得分点

【数据规模与约定】
10%数据：n<=2,m<=2
50%数据：n<=20, m<=20
所有数据：保证n,m,q<=200000。

输入
2 3 2 1 1 2 1

输出?
3

发现+猜想

自动填充完毕后
0格必须花费1次人工填充

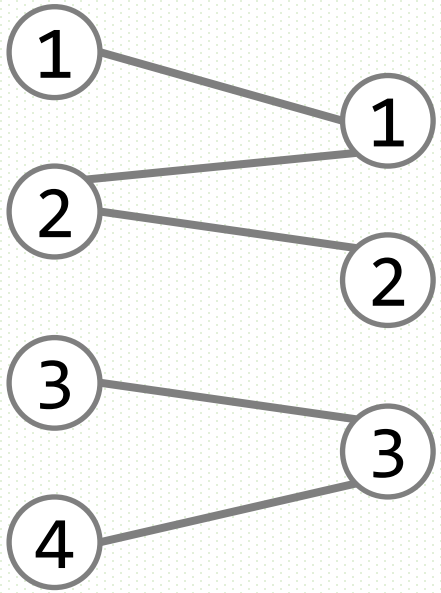
人工填充后再自动填充,不断重复

贪心法为什么正确?

二维01表对应图的连通信息

二分图建模
每行对应左侧节点
每列对应右侧节点

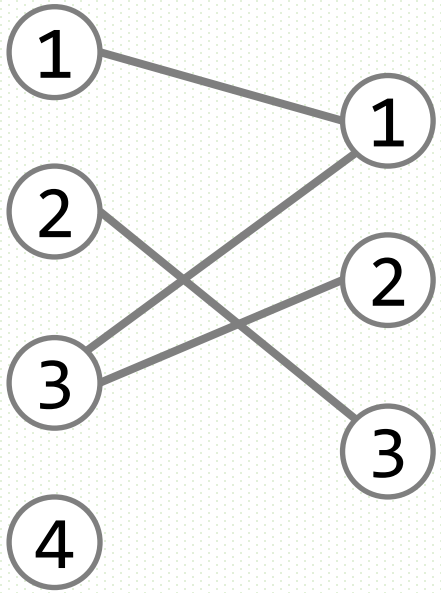
1	0	0
1	1	0
0	0	1
0	0	1



二维01表对应图的连通信息

二分图建模
每行对应左侧节点
每列对应右侧节点

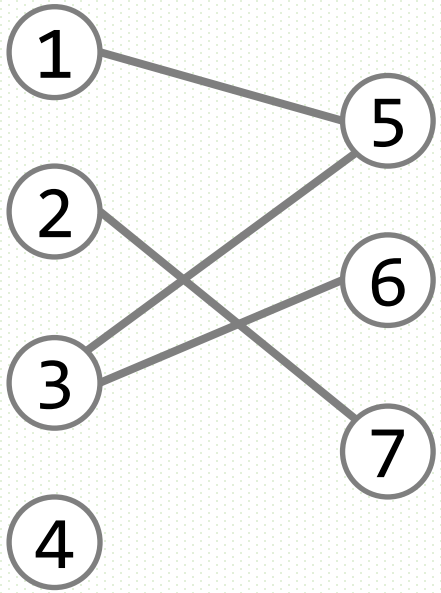
1	0	0
0	0	1
1	1	0
0	0	0



二维01表对应图的连通信息

二分图建模
每行对应左侧节点
每列对应右侧节点

1	0	0
0	0	1
1	1	0
0	0	0



重新编号

答案是 连通块数-1

```
3 const int N=400009;
4 bool vst[N];
5 vector<int> to[N];
6 void add(int u,int v){
7     to[u].push_back(v);
8     to[v].push_back(u);
9 }
```

```
20      int r,c,q;
21      cin>>r>>c>>q;
22      for(int i=1;i<=q;i++){
23          int x,y;
24          cin>>x>>y;
25          add(x,r+y);
26      }

27      int nV=r+c;
28      int cnt=0;
29      for(int u=1;u<=nV;u++)
30          if(!vst[u]){
31              dfs(u);
32              cnt++;
33          }
34      cout<<cnt-1<<endl;
```

```cpp
void dfs(int u){
    vst[u]=1;
    for(int i=0;i<to[u].size();i++){
        int v=to[u][i];
        if(!vst[v]) dfs(v);
    }
}
```

SH-B260103 订正题号

T1　3465 投资方案

T2　3355 记单词

T3　3403 新生入学

T4　941　罗密欧与朱丽叶3

T5　2739 杂草种子