2019NBUTACM校赛题解

A. 画展

解法一：

二分法，按艺术值排序，价值做前缀和

并计算以 i 为起点的艺术值为 $ai+2d$ 的区间总价值。

时间复杂度：$O(nlogn)$

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int N = 1e5+5;

int T;
ll n, d;
struct node {
  ll w, m;
  node(){}
  node(ll w,ll m):w(w),m(m){}
  bool operator < (const node m)const{ //定义比较方式，这一步很重要
    return w < m.w;
  }
}a[N];
ll sum[N];

int main() {
  scanf("%d", &T);
  while (T--) {
    scanf("%lld %lld", &n, &d);
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i].w);
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i].m);
    sort(a+1, a+1+n);

    sum[0] = 0;
    for (int i = 1; i <= n; i++) sum[i] = sum[i-1] + a[i].m;
    ll ans = 0;
    for (int i = 1; i <= n; i++) {
```

```
      int pos = upper_bound(a+1, a+1+n, node(a[i].w+2*d, 0)) - a - 1;
      ans = max(ans, sum[pos] - sum[i-1]);
    }
    cout << ans << endl;
  }
}
```

```
#include <iostream>
#include <cmath>
#include <cstring>
#include <algorithm>
#define ll long long
using namespace std;

int t, n, d;
ll sum[100005], Max;
struct node{
    int x, y;
}a[100005];

bool cmp(node a, node b){
    return a.x < b.x;
}

bool f(int l, int r){
    if(a[r].x - a[l].x <= 2 * d) return true;
    return false;
}

int main(){
    //scanf("%d", &t);
    cin >> t;
    while(t--){
        //scanf("%d%d", &n, &d);
        cin >> n >> d;
        for(int i = 1; i <= n; i++){
            //scanf("%d", &a[i].x);
            cin >> a[i].x;
        }
        for(int i = 1; i <= n; i++){
            //scanf("%d", &a[i].y);
            cin >> a[i].y;
        }

        sort(a + 1, a + n + 1, cmp);
```

```
        memset(sum, 0, sizeof(sum));
        Max = 0;

        for(int i = 1; i <= n; i++){
            sum[i] = sum[i-1] + a[i].y;
        }

        for(int i = 1; i <= n; i++){
            int l = i, r = n, R = i, mid;
            while(l <= r){
                mid = (l + r) / 2;
                if(f(i,mid)){
                    R = mid;
                    l = mid + 1;
                }
                else{
                    r = mid - 1;
                }
            }
            Max = max(Max, sum[R] - sum[i-1]);
        }
        cout << Max << endl;
    }
    return 0;
}
```

解法二：

尺取法，按艺术值排序，价值做前缀和

双指针，左指针代表合法区间的左端，右指针往右移动，找到满足题目条件的最右端，计算总价值。

因为每个位置最多只会被指针遍历2次，所以复杂度是$O(n)$级别的

时间复杂度：$O(nlogn(\text{排序}) + n(\text{尺取}))$

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int mod=1e9+7;
struct node{
    int w,m;
```

```
}a[100100];
bool cmp(node x,node y)
{
    if(x.w==y.w)return x.m>y.m;
    return x.w<y.w;
}

int main(){
    int t,n,d;

    cin>>t;
    while(t--)
    {
        cin>>n>>d;
        for(int i=0;i<n;i++)
        {
            cin>>a[i].w;
        }
        for(int i=0;i<n;i++)
        {
            cin>>a[i].m;
        }
        sort(a,a+n,cmp);
        int f=0;
        ll num=0;
        ll ans=0;
        for(int i=0;i<n;i++)
        {
            while(a[i].w+2*d>=a[f].w&&f<n)
            {
                num+=a[f++].m;
            }
            ans=max(ans,num);
            num-=a[i].m;
        }
        cout<<ans<<endl;
    }

    return 0;
}
```

B. 采红帽的小蘑菇

区间动态规划

$f(i,j,1)$ 表示从 i 到 j 的区间的蘑菇全都采完了，现在在j点，腐烂值最小是多少
$f(i,j,0)$ 表示从 i 到 j 的区间的蘑菇全都采完了，现在在i点，腐烂值最小是多少
$f(i,j,0)=min(f(i+1,j,0)+(s[i+1]-s[i])(n-j+i),f(i+1,j,1)+(s[j]-s[i])($
$f(i,j,1)=min(f(i,j-1,1)+(s[j]-s[j-1])(n-j+i),f(i,j-1,0)+(s[j]-s[i])$
最后$min(f(1,n,0),f(1,n,1))$就是答案啦

时间复杂度：$O(n^2)$

```
#include <cstdio>
#include <algorithm>
using namespace std;
int t,n,l,rec,s[1005],f[1005][1005][2];
int main(){

    scanf("%d",&t);
    while(t--)
    {
        scanf("%d%d",&n,&l);
        for(int i=1;i<=n;i++)scanf("%d",&s[i]);
        s[++n]=l;
        sort(s+1,s+1+n);
        for(int i=1;i<=n;i++)
            if(s[i]==l)rec=l,f[i][i][0]=f[i][i][1]=0;
            else f[i][i][0]=f[i][i][1]=0x3fffffff;
        for(int i=rec;i;i--)
            for(int j=i+1;j<=n;j++){
                f[i][j][0]=min(f[i+1][j][0]+(s[i+1]-s[i])*(n-j+i),f[i+1][j][1
]+(s[j]-s[i])*(n-j+i));
                f[i][j][1]=min(f[i][j-1][1]+(s[j]-s[j-1])*(n-j+i),f[i][j-1][0
]+(s[j]-s[i])*(n-j+i));
            }
        printf("%d\n",min(f[1][n][0],f[1][n][1]));
    }


}
```

C. 鲜花物语2

首先知道一个公式：

$C_{a_i}^0+\ldots+C_{a_i}^{a_i}=2^{a_i}$

然后贪心，要得到最大值，因此x必须得最小，因此$x = 0$

所以最后就变成了

$b_i * (2^{a_i} - 1)$

最后可以用快速幂求解或者预处理2的幂

时间复杂度：$O(n)$

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int N = 1e5+5;
const ll MOD = 1e9+7;
int T;
int n, m;
ll a[N], b[N];
ll ans;
ll _2[N];

int main() {
  _2[0] = 1;
  for (int i = 1; i < N; i++) {
    _2[i] = (_2[i-1] * 2) % MOD;
  }
  scanf("%d", &T);
  while (T--) {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i]);
    for (int i = 1; i <= n; i++) scanf("%lld", &b[i]);

    ans = 0;
    for (int i = 1; i <= n; i++) {
      ans = (ans + ((_2[a[i]]-1+MOD)%MOD * b[i]) % MOD) % MOD;
    }
    printf("%lld\n", ans);
  }
}
```

D. 鲜花物语1

贪心，对亲密度进行排序，按照从大到小排序，每种花只取一朵

然后扫描一遍1到n，如果花园可放的花的数量小于 a_i ，那么亲密度加上 $b_i * $ 花园可放花的数量

如果可放花的数量大于a_i ,那么亲密度加上 $a_i * b_i$,花园可放花的数量减去 a_i

时间复杂度：$O(n)$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll MOD = 1e9 + 7;
const int N = 1e5 + 5;
struct point{
    ll a,b;
}a[N];
int cmp(point x,point y){
    return x.b > y.b;
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        int n,m;
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++) scanf("%lld",&a[i].a);
        for(int i=1;i<=n;i++) scanf("%lld",&a[i].b);
        sort(a+1,a+1+n,cmp);
        ll ans = 0;
        ll left = m;
        for(int i=1;i<=n;i++){
            ans = ans + min(a[i].a,left)*a[i].b;
            left = left - min(a[i].a,left);
            if(left == 0) break;
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

E. 吃肉肉不吃肉肉！

考虑递推 设 $f[i]$ 代表长为 i 的01串合法情况(包括全是0的)

如果第 i 位是 0，那么前面不论是什么都可以，即有 $f[i-1]$种

如果第 i 位是 1，那么第 i - 1 位必须是 0，参考上一条，所以有 $f[i-2]$ 种

所以 $f[i]=f[i-1]+f[i-2]$，初始值 $f[1]=2$, $f[2]=3$，最后的输出要减一

时间复杂度：$O(n)$

```
#include <iostream>
#include <cstdio>
using namespace std;

long long a[55];
int T, n;

void f(){
    a[1] = 2;
    a[2] = 3;
    for(int i = 3; i <= 50; i++){
        a[i] = a[i-1] + a[i-2];
    }
}

int main(){
    f();
    scanf("%d", &T);
    while(T--){
        scanf("%d", &n);
        printf("%lld\n", a[n] - 1);
    }
    return 0;
}
```

F. 数星星

勾股定理计算一下距离，比较大小即可

```
#include <iostream>
#include <cstdio>
using namespace std;
int T, n, x, y, s, r, S, X, Y;
int sum;
bool check(int x, int y){
    return r * r >= (X - x) * (X - x) + (Y - y) * (Y - y);
}
```

```
int main(){
    scanf("%d", &T);
    while(T--){
        scanf("%d%d", &n, &r);
        sum = 0;
        scanf("%d%d%d", &S, &X, &Y);
        for(int i = 1; i < n; i++){
            scanf("%d%d%d", &s, &x, &y);
            if(s < S && check(x, y)){
                sum++;
            }
        }
        printf("%d\n", sum);
    }
    return 0;
}
```

G. 期末考

解法一：

男女分两组，结构体按分数排序

时间复杂度：$O(nlog)$

```
#include <iostream>
#include <string>
#include <string.h>
#include <algorithm>
using namespace std;

struct studentM{
    string name,id;
    char gender;
    int grade;
}male[50005];
struct studentF{
    string name,id;
    char gender;
    int grade;
}female[50005];
bool cmp1(studentM a,studentM b){
    return a.grade<b.grade;
```

```
}
bool cmp2(studentF a,studentF b){
    return a.grade>b.grade;
}

int main(){
    int t;
    cin>>t;
    while(t--){
        int n,grade,m=0,f=0;
        string name,id;
        char gender;
        cin>>n;
        for(int i=0;i<n;i++){
            cin>>name>>gender>>id>>grade;
            if(gender=='M'){
                m++;
                male[m].name=name;
                male[m].gender=gender;
                male[m].id=id;
                male[m].grade=grade;
            }
            else{
                f++;
                female[f].name=name;
                female[f].gender=gender;
                female[f].id=id;
                female[f].grade=grade;
            }
        }
        sort(male+1,male+m+1,cmp1);
        sort(female+1,female+f+1,cmp2);
        if(f==0) cout<<"Miss"<<endl;
        else cout<<female[1].name<<" "<<female[1].id<<endl;
        if(m==0) cout<<"Miss"<<endl;
        else cout<<male[1].name<<" "<<male[1].id<<endl;
        if(m==0||f==0) cout<<"Null"<<endl;
        else cout<<female[1].grade-male[1].grade<<endl;
    }

    return 0;
}
```

解法二：

循环遍历数组模拟，挑出最大最小，记录信息

时间复杂度：$O(n)$

```
#include <iostream>
#include <string>
using namespace std;

int t, n, Max, Min, num;
string m, mm, f, ff, x, y, z;
int main(){
    cin >> t;
    while(t--){
        Max = -1, Min = 101;
        cin >> n;
        while(n--){
            cin >> x >> y >> z >> num;
            if(y == "M"){
                if(num < Min){
                    Min = num;
                    m = x;
                    mm = z;
                }
            }
            else{
                if(num > Max){
                    Max = num;
                    f = x;
                    ff = z;
                }
            }

        }
        if(Max == -1)
            cout << "Miss" << endl;
        else cout << f << " " << ff << endl;

        if(Min == 101)
            cout << "Miss" << endl;
        else cout << m << " " << mm << endl;

        if(Max != -1 && Min != 101) cout << Max - Min << endl;
        else cout << "Null" << endl;

    }
```

```
    return 0;
}
```

H. 造桥鬼才XOR

解法一：

BFS变形，使用双端队列，搜索到 'S' 时，将其放到队首，步数不增加，相当于优先搜索 'S‘，搜索到另一个 'I' 停止。

时间复杂度：$O(n*m)$

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1005;
const int dx[] = {-1, 1, 0, 0};
const int dy[] = {0, 0, 1, -1};

struct node{int x, y, d;};

int n, m;
char p[N][N];
int vis[N][N];
deque<node> dq;
node now, tmp;

bool check(int x, int y) {
    if (x < 1 || x > n || y < 1 || y > m) return 0;
    if (vis[x][y]) return 0;
    return 1;
}

int bfs(int x, int y) {
    dq.push_back(node({x, y, 0}));
    vis[x][y] = 1;
    while(dq.size()) {
        node now = dq.front(); dq.pop_front();
        for (int i = 0; i < 4; i++) {
            tmp.x = now.x + dx[i];
            tmp.y = now.y + dy[i];
            if (check(tmp.x, tmp.y)) {
                if (p[tmp.x][tmp.y] == 'I' && p[now.x][now.y] != 'I') return
now.d;
```

```
                if (p[tmp.x][tmp.y] == 'I') {
                    dq.push_front(node({tmp.x, tmp.y, 0}));
                }
                else if(p[tmp.x][tmp.y] == 'S') {
                    dq.push_front(node({tmp.x, tmp.y, now.d}));
                }
                else {
                    dq.push_back(node({tmp.x, tmp.y, now.d+1}));
                }
                vis[tmp.x][tmp.y] = 1;
            }
        }
    }
    return -1;
}

void init() {
    while(!dq.empty()) dq.pop_back();
    memset(vis, 0, sizeof vis);
}

void solve() {
    init();
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> (p[i] + 1);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (p[i][j] == 'I') {
                cout << bfs(i, j) << endl;
                return;
            }
        }
    }
}

int main() {
    int T;
    cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

解法二：

优先队列，按步数排序。优先搜索最近能走到的地方，搜索到另一个 'I' 停止。

时间复杂度：$O(n*m*log)$

```
#include <bits/stdc++.h>
using namespace std;

int T;
int n, m, vis[505][505];
char pic[505][505];
int dx[] = {1, -1, 0, 0};
int dy[] = {0, 0, 1, -1};
struct node {
  int x, y, step;
  node(int x, int y, int step) {
    this->x = x;
    this->y = y;
    this->step = step;
  }
  friend bool operator < (node a, node b) {
    return a.step > b.step; // step小的优先
  }
};
priority_queue<node>q;

int bfs(int sx, int sy) {
  vis[sx][sy] = 1;
  q.push(node(sx, sy, 0));
  while (!q.empty()) {
    node u = q.top();
    q.pop();
    for (int i = 0; i < 4; i++) {
      int x = u.x + dx[i];
      int y = u.y + dy[i];
      int s = u.step;

      if (x < 1 || x > n || y < 1 || y > m || vis[x][y] == 1) continue;
      //cout << x << ' ' << y << ' ' << s << endl;
      vis[x][y] = 1;
      if (pic[u.x][u.y] != 'I' && pic[x][y] == 'I') {
        return s;
      }
      else if (pic[x][y] == 'I' || pic[x][y] == 'S') {
        q.push(node(x, y, s));
      }
```

```
      else if (pic[x][y] == '.') {
        q.push(node(x, y, s+1));
      }
    }
  }
  return 0;
}

int main() {
  cin >> T;
  while (T--) {
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= m; j++) {
        vis[i][j] = 0;
        cin >> pic[i][j];
      }
    }
    while (!q.empty()) q.pop();

    int ans = 0;
    for (int i = 1; i <= n; i++) {
      int f = 0;
      for (int j = 1; j <= m; j++) {
        if (pic[i][j] == 'I') {
          ans = bfs(i, j);
          f = 1;
          break;
        }
      }
      if (f) break;
    }
    cout << ans << endl;

  }
}
```

I. 艾瑞II

使用线段树数据结构维护区间信息，典型的区间加以及区间求和，额外记录一个lazy值cnt，代表被覆盖次数，push_down的时候加上一个等差数列求和即可。

时间复杂度：$O(n + mlogn)$

空间复杂度：$O(n)$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 5;

struct segt {
    struct seg {
        int l, r;
        ll sum, lz, cnt;
        void update(ll val, ll cont) {
            sum += (r - l + 1) * val;
            sum += cont * (r - l) * (r - l + 1) / 2;
            lz += val;
            cnt += cont;
        }
    };

    #define lc x << 1
    #define rc x << 1 | 1
    #define mid ((t[x].l + t[x].r) / 2)

    ll * a;
    int left;
    seg t[N << 2];

    void modify(ll * arr) {a = arr;}
    void push_up(int x) {t[x].sum = t[lc].sum + t[rc].sum;}
    void push_down(int x) {
        ll lz = t[x].lz, cnt = t[x].cnt;
        t[lc].update(lz, cnt);
        t[rc].update(lz + cnt * (t[rc].l - t[lc].l), cnt);
        t[x].lz = t[x].cnt = 0;
    }
    void build(int x, int l, int r) {
        t[x] = seg({l, r, 0, 0, 0});
        if (l == r) {
            t[x].sum = a[l];
            return;
        }
        build(lc, l, mid);
        build(rc, mid + 1, r);
        push_up(x);
    }
```

```
    void update(int x, int l, int r) {
        int L(t[x].l), R(t[x].r);
        if (l <= L && R <= r) {
            t[x].update(t[x].l - left + 1, 1);
            return;
        }
        push_down(x);
        if (l <= mid) update(lc, l, r);
        if (r >  mid) update(rc, l, r);
        push_up(x);
    }
    ll query(int x, int l, int r) {
        int L(t[x].l), R(t[x].r);
        if (l <= L && R <= r) return t[x].sum;
        push_down(x);
        ll res = 0;
        if (l <= mid) res += query(lc, l, r);
        if (r >  mid) res += query(rc, l, r);
        return res;
    }
};

int n, m, op, l, r;
ll a[N];
segt tree;

void solve() {
    // cin >> n >> m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        // cin >> a[i];
        scanf("%lld", a+i);
    tree.modify(a);
    tree.build(1, 1, n);
    while (m--) {
        // cin >> op >> l >> r;
        scanf("%d%d%d", &op, &l, &r);
        if (op == 1) {
            tree.left = l;
            tree.update(1, l, r);
        }
        else{
            // cout << tree.query(1, l, r) << '\n';
            printf("%lld\n", tree.query(1, l, r));
        }
    }
```

```
}

int main() {
    int T;
    cin >> T;
    while(T--) solve();
    return 0;
}
```

J. 小柯的晚饭

解法一：

用数组模拟被覆盖的时间，统计被覆盖的时间数。

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int N = 1e5+5;
const ll MOD = 1e9+7;

int T;
int n;
int s, e;
int vis[1005];
int ans;

int main() {
  cin >> T;
  while (T--) {
    cin >> n;
    for (int i = 0; i <= 1000; i++) vis[i] = 0;
    while (n--) {
      cin >> s >> e;
      for (int i = s; i < e; i++) {
        vis[i] = 1;
      }
    }
    ans = 0;
    for (int i = 1; i <= 1000; i++) {
      if(vis[i] == 1) ans ++;
    }
    cout << ans << endl;
```

```
    }
}
```

解法二：

对区间排序，贪心求解

```
#include<algorithm>
#include <iostream>
#include<string.h>

#include<fstream>
using namespace std;
struct node{
    int start;
    int ending;
}l[1005];
bool cmp(node a,node b){
    if(a.start!=b.start)
        return a.start<b.start;
    return a.ending<b.ending;
}
int ans,n;
void init(){
    memset(l,0,sizeof(l));
    ans=0;
}
void solve(){ //贪心算法求得线段覆盖长度
    for(int i=1;i<n;i++){
        if( l[i].start >= l[i-1].start && l[i].start <= l[i-1].ending && l[i]
.ending > l[i-1].ending)//当前线段起点大于等于前一个线段的起点小于等于前一个线段的终点
并且终点大于前一个线段的终点
            ans += l[i].ending - l[i-1].ending; //总覆盖长度增加两线段终点只差
        else if( l[i].start >= l[i-1].ending)//当前线段的起点大于等于前一线段的终点
            ans += l[i].ending - l[i].start;//总覆盖长度增加此线段的长度
        else if(l[i].start >= l[i-1].start && l[i].ending < l[i-1].ending)//
当前线段在前一条线段的区间内
            l[i].ending = l[i-1].ending; //将当前线段的终点改为前一线段的终点
    }
}
int main(){
    int t;
    cin>>t;
    while(t--){
```

```
        init();
        cin>>n;
        for(int i=0;i<n;i++){
            cin>>l[i].start>>l[i].ending;
        }
        sort(l,l+n,cmp);
        ans= l[0].ending - l[0].start;
        solve();
        cout<<ans<<endl;
    }
    return 0;
}
```