NOIPmoni2题解

[HNOI2005] 狡猾的商人

区间DP:状态表示f[l,r]表示这段时间的收入,枚举这段时间的所有断点k ， 可以把[l,r]区间切割成两部分[l,k],[k+1,r] 转移f[l][r] = f[l][k]+f[k+1][r]， 我们用INF表示区间营业额没有确定,如果区间营业额已经确定且与两个子区间的和不同，代表冲突，可以输出false

```cpp
const int N = 220, INF = 0x3f3f3f3f;
int f[N][N], n, m, T, ok;

int main()
{
    scanf("%d", &T);

    while (T -- )
    {
        scanf("%d%d", &n, &m);
        ok = 1;

        memset(f, 0x3f, sizeof f);
        while (m -- )
        {
            int s, t, v;
            scanf("%d%d%d", &s, &t, &v);
            if (f[s][t] == INF) f[s][t] = v;
            else if (f[s][t] != v) ok = 0;
        }

        if (!ok)
        {
            puts("false");
            continue ;
        }

        for (int len = 1; len <= n && ok; len ++ )
        {
            for (int l = 1, r = l + len - 1; r <= n; l ++, r ++ )
            {
                for (int k = l; k < r; k ++ )
                {
                    if (f[l][k] != INF && f[k + 1][r] != INF)
                    {
                        if (f[l][r] == INF) f[l][r] = f[l][k] + f[k + 1][r];
                        else if (f[l][r] != f[l][k] + f[k +1][r]) ok = 0;
                    }
                }
            }
        }

        puts(ok ? "true" : "false");
    }

    return 0;
}
```

带权并查集:首先我们研究并查集的本质，发现其实并查集是一棵树，最开始我们的并查集都是一个单点，然后慢慢地联通的，于是我们维护一个神奇的值是从这个点到根节点的差值，我们一会儿再来讨论维护的这个值，现在我们先看这个： 输入了 l,r,num，显然我们现在是l到r的值之和为num，如果我们把一个点一个点的前缀和表示为s，则s[r]−s[l−1]=num，因此我们可以维护一个值表示这个值到他的根的差值，设这个差值数组是sum，而如果l−1这个点和r这个点是联通的，那么显然有num=sum[l−1]−sum[r]因为我们的sum数组表示的是差值，也就是说从根节点到"最左边的节点"的距离减去从l−1到"最左边的节点"的距离的值，那么l−1的这段值如果减去r的这段值如果不等于输入的num，也就是之前输出的内容有冲突，则说明false，如果没有冲突则在这条账目上是对的 如果两个点的fa不同，则需要新维护 首先将v的父亲合并到u的父亲上 f[tv] = tu, 再更新tv的到tu的距离 sum[tv] = sum[u] + w - sum[v] 在find函数中进行合并即可

```cpp
int n, m;
int fa[MAXN], dis[MAXN];
inline int find(int x) {
  if (fa[x] == x)
    return x;
  int t = find(fa[x]);
  dis[x] += dis[fa[x]];
  fa[x] = t;
  return t;
}
inline void work(signed CASE = 1, bool FINAL_CASE = false) {
  cin >> n >> m;
  for (int i = 0; i <= n; i++) {
    fa[i] = i;
    dis[i] = 0;
  }
  bool ok = true;
  for (int i = 1; i <= m; i++) {
    int l, r, w;
    cin >> l >> r >> w;
    l--;
    int rr = find(r), ll = find(l);
    int b = dis[r], c = dis[l];
    if (rr != ll) {
      fa[ll] = rr;
      dis[ll] = b - c + w;
    } else {
      if (-w != b - c) {
        ok = false;
        break;
      }
    }
  }
  if (ok)
    cout << "true" << endl;
  else
    cout << "false" << endl;
}
```

差分约束：自己了解

最澄澈的空与海

结论当一个二分图有且仅有一种解时，必定有节点的入度为 1

```cpp
#include<bits/stdc++.h>
#define N 2000005
#define ll long long
using namespace std;

int t,n,m,cnt;
vector<int> to[N];
int in[N];
bool del[N];

void solve(){
    cin>>n>>m;
    for(int i=1,u,v;i<=m;i++){
        cin>>u>>v; v+=n; //右边的点重新编号
        to[u].push_back(v); in[v]++;
        to[v].push_back(u); in[u]++;
    }
    queue<int> q; cnt=0;
    for(int i=1;i<=2*n;i++)
        if(in[i]==1)q.push(i); //将入度为1的点入队
    while(!q.empty()){
        int now=q.front(),buf=0; q.pop();
        if(del[now]||in[now]!=1)continue; //如果已经没删掉了，或者入度不为1了，那就跳过。已经入队的点的状态也可能被改变，因为这里是左右两边一起迭代的。
        del[now]=true; cnt++; //删掉这个点，删掉的数量加1
        while(del[to[now][buf]])buf++; //找到仅存的那一条边
        del[to[now][buf]]=true; cnt++; //顺着仅存的边找到另一个点，删掉
        for(int i=0;i<to[to[now][buf]].size();i++) //删边，没有真的删，只是减少目标节点的入度，这也是为什么要用while找仅存的边。
            if(!del[to[to[now][buf]][i]]&&(--in[to[to[now][buf]][i]])==1)
                q.push(to[to[now][buf]][i]); //如果还没被删，并且入度减少为1了就入队
    }
    if(cnt==n*2)cout<<"Renko"<<endl; //全删了
    else cout<<"Merry"<<endl;  //没全删
    for(int i=1;i<=n*2;i++)in[i]=del[i]=0,to[i].clear(); //多组数据，要初始化
}

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

[NOI Online #1 提高组] 冒泡排序

假设原本在位置 i 上的逆序对数为 c_i ，冒泡排序一轮后，位置 i 上的逆序对数变为 $max(0, c_{i+1} - 1)$。 相当于一个左移，减一，与 0 取 max 的过程。 结果为$\sum(a[i] - k) * b[i]$

,b[i]是值等于a[i]的个数

```
int n, m;
int p[MAXN];
struct BIT {
  int n;
  int tr[MAXN];
  inline void init(int _n) {
    n = _n;
    mmst0(tr);
  }
  inline void add(int pos, int val) {
    pos++;
    for (; pos <= n; pos += pos & -pos)
      tr[pos] += val;
  }
  inline int query(int pos) {
    pos++;
    int ans = 0;
    for (; pos; pos -= pos & -pos)
      ans += tr[pos];
    return ans;
  }
  inline void set(int pos, int val) {
    pos++;
    int ori = query(pos) - query(pos - 1);
    add(pos, -ori);
    add(pos, val);
    // cout << pos << ":"
    //      << "from " << ori << "to" << val << "\n";
  }
  inline void out(int pos) {
    pos++;
    cout << query(pos) - query(pos - 1);
  }
} tr1, tr2, tr3; // tr2维护s[i],tr3维护s[i]*i;
int s[MAXN];
int cnt[MAXN];
inline void update(int c) {
  if (p[c] < p[c + 1]) {
    // cout << "op1"
    //      << " ";
    s[cnt[p[c]]] -= 1;
    tr2.add(cnt[p[c]], -1);
    tr3.set(cnt[p[c]] - 1, cnt[p[c]] * s[cnt[p[c]]]);
    cnt[p[c]] += 1;
    s[cnt[p[c]]] += 1;
    tr2.add(cnt[p[c]], +1);
    tr3.set(cnt[p[c]] - 1, cnt[p[c]] * s[cnt[p[c]]]);
  } else {
    // cout << "op2"
    //      << " ";
```

```
    s[cnt[p[c + 1]]] -= 1;
    tr2.add(cnt[p[c + 1]], -1);
    tr3.set(cnt[p[c + 1]] - 1, cnt[p[c + 1]] * s[cnt[p[c + 1]]]);
    cnt[p[c + 1]] -= 1;
    s[cnt[p[c + 1]]] += 1;
    tr2.add(cnt[p[c + 1]], +1);
    tr3.set(cnt[p[c + 1]] - 1, cnt[p[c + 1]] * s[cnt[p[c + 1]]]);
  }
  swap(p[c], p[c + 1]);
}
inline int query(int k) {
  int ans = 0;
  if (n <= k)
    return 0;
  ans = (tr3.query(n - 1) - tr3.query(k - 1)) -
        (tr2.query(n - 1) - tr2.query(k - 1)) * k;
  return ans;
}
inline void work(signed CASE = 1, bool FINAL_CASE = false) {
  cin >> n >> m;
  for (int i = 1; i <= n; i++)
    cin >> p[i];
  tr1.init(MAXN - 1);
  tr2.init(MAXN - 1);
  tr3.init(MAXN - 1);
  for (int i = 1, t = 0; i <= n; i++) {
    t = tr1.query(p[i]);
    s[i - t - 1] += 1, cnt[p[i]] = i - t - 1;
    tr1.add(p[i], 1);
  }
  for (int i = 0; i <= n; i++)
    tr2.add(i, s[i]), tr3.add(i, s[i] * i);
  for (int i = 1, t, c; i <= m; i++) {
    cin >> t >> c;
    if (t & 1)
      update(c);
    else
      cout << query(c) << "\n";
  }
}
```

小清新签到题

```
  dp[1][0] = 1; // 长度为1的序列只有一对逆序对
  for (int i = 1; i <= n; i++) {
    for (int j = 0; j <= x; j++) {
      for (int p = 0; p <= i - 1; p++) { // 枚举插入n的位置[0,i-1]
        if (j - (i - 1 - p) >= 0)
          dp[i][j] += dp[i - 1][j - (i - 1 - p)];
        if (dp[i][j] > k) {
          dp[i][j] = k + 1;
          break;
        }
      }
    }
  }
  for (int i = 1; i <= n; i++) {
    int now = 0;
    for (int j = 1; j <= n; j++) { // 选a[i]的值
      if (vis[j])
        continue;
      int c1 = j - 1;               // 比j小的数
      for (int t = 1; t < j; t++) { // 减去在j前比j小的数
        if (vis[t])
          c1 -= 1;
      }
      // 此刻c1表示j与后面元素形成的逆序对对数
      if (x - c1 >= 0 && dp[n - i][x - c1] + now >= k) {
        x -= c1, k -= now, ans[i] = j, vis[j] = true;
        break;
      }
      now += dp[n - i][x - c1]; // 满足n,x限制的序列但是字典序是第k之前
    }
  }
  for (int i = 1; i <= n; i++)
    if (!vis[i]) {
      ans[n] = i;
      break;
    }
  for (int i = 1; i <= n; i++)
    cout << ans[i] << " ";
```

https://www.luogu.com.cn/problem/solution/P3672