E 妮妮的约会大作战

> 21351023 李永琦

题目描述

妮妮的社团正在举办联谊活动！目前报名的有 n 个男生和 n 个女生。其中第 i 个男生魅力值为 a_i，只喜欢魅力值不小于 p_i 的女生；第 i 个女生魅力值为 b_i，只喜欢魅力值不小于 q_i 的男生。

男生与女生能约会当且仅当两人相互喜欢。妮妮希望让尽可能多的人能约会成功，那么最多有多少对情侣能约会成功呢？

输入格式

第一行一个正整数 n（$1 \leq n \leq 400$），表示男生和女生的数量。

第二行 n 个非负整数 $a_1, a_2, \ldots, a_n$（$0 \leq a_i \leq 10^9$），表示男生的魅力值。

第三行 n 个非负整数 $p_1, p_2, \ldots, p_n$（$0 \leq p_i \leq 10^9$），表示男生喜欢的女生魅力值最小值。

第四行 n 个非负整数 $b_1, b_2, \ldots, b_n$（$0 \leq b_i \leq 10^9$），表示女生的魅力值。

第五行 n 个非负整数 $q_1, q_2, \ldots, q_n$（$0 \leq q_i \leq 10^9$）表示女生喜欢的男生魅力值最小值。

输出格式

一行一个非负整数，表示最多能有几对情侣约会成功。

题目分析

本题首先根据条件建图，然后即为无权二分图最大匹配问题：

> 给定一个二分图 G，即分左右两部分，各部分之间的点没有边连接，要求选出一些边，使得这些边没有公共顶点，且边的数量最大。

匈牙利算法

求最大匹配的核心思路：枚举所有未匹配点的出边的终点，如果终点未匹配，那么进行匹配，并返回匹配成功；

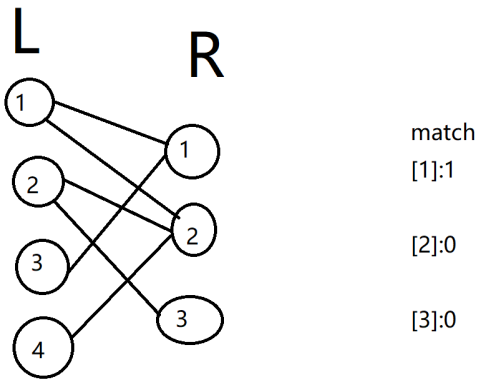
如果该终点已经被匹配过了，那么递归该点匹配的左边的点，查看这个点是否能换一个点来匹配，如果可以，返回匹配成功，否则返回匹配失败。

核心代码如下：

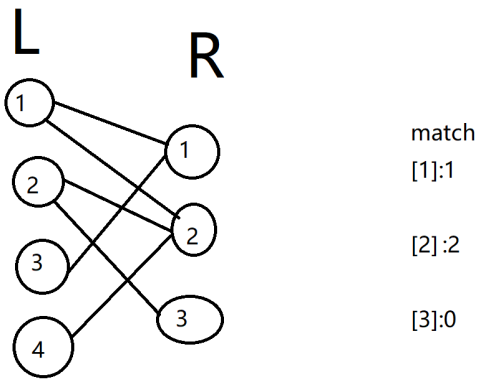
```
bool find(int x)
{
    for (int i : g[x]) // x 的所有出边的终点
    {
        if (!vis[i])
        {
            vis[i] = true;
            if (!match[i] || find(match[i])) // 该点未匹配 或 匹配的左边的点可以换一个
点
            {
                match[i] = x;
```

```
                return true;
            }
        }
    }
    return false;
}
int solve()
{
    int res = 0;
    for (int i = 1; i <= n; i++)
    {
        fill(vis.begin(), vis.end(), false);
        if (find(i))
        {
            ++res;
        }
    }
    return res;
}
```

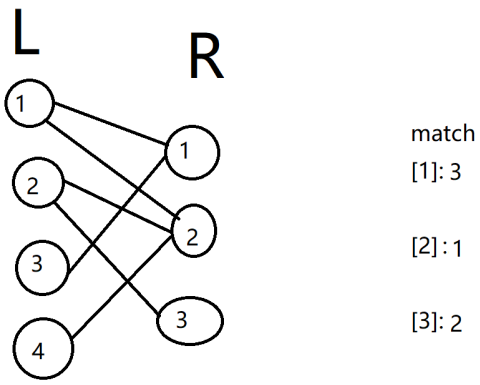
下图演示了该过程，其中 `match[y] = x` 表示当前左边第 `x` 个点匹配到了右边第 `y` 个点。对每个点执行 `find` 函数，首先寻找左侧点 1 的匹配对象，匹配到了右侧点 1：



接着找点 2，它的第一条出边终点（右侧点 2）未匹配，两者可匹配：



接着找点 3，它的第一条出边终点（右侧点 1）已经和左侧点 1 匹配了。此时为尽量增大匹配数，考虑让左侧点 1 重新寻找匹配对象，即递归调用 `find(1)`。发现左侧点 1 的第二条出边终点（右侧点 2）也已被匹配至左侧点 2，那么再调用 `find(2)`，最终发现左侧点 2 可匹配右侧点 3。返回匹配成功后，函数可递归修改匹配的值，使左侧点 3 匹配上右侧点 1。



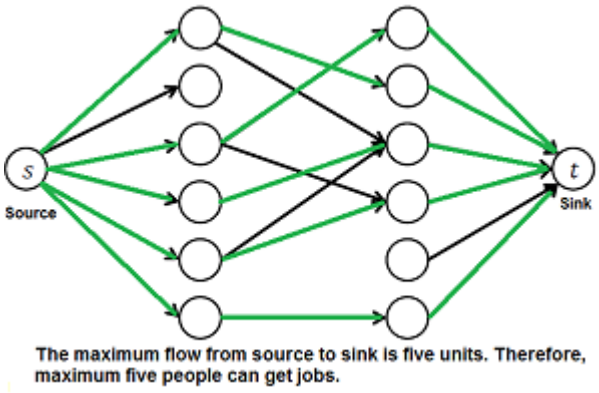
最后匹配点 4，发现无法匹配，算法结束。

每次调用时若匹配成功，则匹配数 +1；否则不变。因为要枚举 n 个点做 DFS，总复杂度为 $O(nm)$。

图片来源：zmza 的题解

最大流

二分图最大匹配可以转换成网络流模型：建立超级源点和超级汇点，将源点连上左边所有点，右边所有点连上汇点，原来的每条边从左往右连边，所有边的容量皆为 1，此时最大流即最大匹配。



如果使用 Dinic 算法，时间复杂度为 $O(\sqrt{n}m)$。Dinic 算法分成两部分，第一部分用 $O(m)$ 时间 BFS 建立网络流，第二步是 $O(nm)$ 时间 DFS 进行增广。但因为容量为 1，所以实际时间复杂度为 $O(m)$。循环 $O(\sqrt{n})$ 轮以后，每条增广路径长度至少 $\sqrt{n}$，而这样的路径不超过 $\sqrt{n}$，所以此时最多只需要跑 $\sqrt{n}$ 轮，整体复杂度为 $O(\sqrt{n}m)$。

算法的具体内容参考 C 题。

图片来源

模板

P3386 【模板】二分图最大匹配

B3605 [图论与代数结构 401] 二分图匹配

Hopcroft-Karp 算法也可解决该问题，复杂度为 $O(\sqrt{n}m)$。

代码

Kuhn (Hungarian)

```
#include <algorithm>
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
int n;
struct
{
    int first, second;
} male[405], female[405];
class Kuhn
{
  public:
    int n;
    vector<vector<int>> g;
    vector<int> match;
    vector<int> vis;
    Kuhn(int _n) : n(_n), g(_n + 1), match(_n + 1), vis(_n + 1)
    {
    }
    void add(int u, int v)
    {
        g[u].push_back(v);
    }
    bool find(int x)
    {
        for (int i : g[x])
        {
            if (!vis[i])
            {
                vis[i] = true;
                if (!match[i] || find(match[i]))
                {
                    match[i] = x;
                    return true;
                }
            }
        }
        return false;
    }
    int solve()
    {
        int res = 0;
        for (int i = 1; i <= n; i++)
        {
            fill(vis.begin(), vis.end(), false);
            if (find(i))
            {
                ++res;
```

```
            }
        }
        return res;
    }
};
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    Kuhn kuhn(n);
    for (int i = 1; i <= n; i++)
    {
        cin >> male[i].first;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> male[i].second;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> female[i].first;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> female[i].second;
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (male[i].first >= female[j].second && female[j].first >=
male[i].second)
            {
                kuhn.add(i, j);
            }
        }
    }
    cout << kuhn.solve() << endl;
    return 0;
}
```

Dinic

```
#include <cstring>
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
struct
{
    int first, second;
} male[405], female[405];
struct Edge
{
```

```
    int v, nxt, flow;
};
Edge edges[161000];
int head[805], dep[805], cur[805], cnt, T, n, m, s, t, u, v, w;
int maxflow;
void add(int u, int v, int w)
{
    edges[++cnt] = {v, head[u], w};
    head[u] = cnt;
}
void addTwoDirections(int u, int v, int w)
{
    add(u, v, w);
    add(v, u, 0);
}
bool bfs()
{
    queue<int> q;
    memset(dep, 0, sizeof(int) * (t + 1));
    dep[s] = 1;
    q.push(s);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int i = head[u]; ~i;)
        {
            auto [v, nxt, flow] = edges[i];
            if (!dep[v] && flow > 0)
            {
                dep[v] = dep[u] + 1;
                q.push(v);
                if (v == t)
                {
                    return true;
                }
            }
            i = nxt;
        }
    }
    return false;
}
int dfs(int u, int flow)
{
    if (u == t)
    {
        return flow;
    }
    int d, res = 0;
    for (int i = cur[u]; ~i && flow;)
    {
        cur[u] = i;
        auto &[v, nxt, f] = edges[i];
        if (f > 0 && dep[v] == dep[u] + 1)
        {
            d = dfs(v, min(flow, f));
```

```
            if (!d)
            {
                dep[v] = -1;
            }
            res += d;
            f -= d;
            edges[i ^ 1].flow += d;
            flow -= d;
        }
        i = nxt;
    }
    return res;
}
void dinic()
{
    while (bfs())
    {
        memcpy(cur, head, sizeof(int) * (t + 1));
        maxflow += dfs(s, 1e9);
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cnt = -1;
    maxflow = 0;
    cin >> n;
    s = 0;
    t = n * 2 + 1;
    for (int i = 0; i <= t; i++)
    {
        head[i] = -1;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> male[i].first;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> male[i].second;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> female[i].first;
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> female[i].second;
    }
    for (int i = 1; i <= n; i++)
    {
        addTwoDirections(0, i, 1);
        addTwoDirections(n + i, t, 1);
    }
    for (int i = 1; i <= n; i++)
```

```
    {
        for (int j = 1; j <= n; j++)
        {
            if (male[i].first >= female[j].second && female[j].first >=
male[i].second)
            {
                addTwoDirections(i, n + j, 1);
            }
        }
    }
    dinic();
    cout << maxflow << endl;
    return 0;
}
```