不会dfs的先去网上学，第三题是dfs的应用模板题。

不同的和

> 给你 n 个数字，你可以从中取出任意多个数字进行求和，问：你最多可以求出多少不同的和。

解析

1. dfs搜索每一种不同的情况
2. 用last参数去维护搜索，可以使得数字不会重复，打个比方，现在有数字1，2，0，4，你每次只能按顺序递增的取数字，比如:1 2 , 1 0 , 1 4 , 2 0 ,2 4 ,0 ,4 , 1 2 0 ,1 2 4 ... 这样可以保证在每个深度不会重复搜索
3. 这里用到了set，set表示一个集合（不能拥有相同的元素），因此set的大小便是不同的和，不会set的百度自学，或者看注释。

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
int r[21];
int n;
set<int>s;//集合
void dfs(int step,int last,int sum ) {
    s.insert(sum);//把数字放到集合
    if (step == n) {
        return;
    }
    for (int i = last; i < n; i++) {
        dfs(step + 1, i + 1, sum + r[i]);
    }
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> r[i];
    }
    dfs(0, 0, 0);
    cout << s.size() << endl;//输出集合的大小
}
```

Singing and Dancing

> 歌舞会的精彩度 = 唱歌精彩度 * 跳舞精彩度。
>
> 唱歌精彩度 = a 个主歌手的唱歌能力之按位异或 + (x - a) 个副歌手的唱歌能力之按位或。

> 跳舞精彩度 = b 个主舞者的跳舞能力之按位异或 + (n - x - b) 个副舞者的跳舞能力之按位与。
>
> 如何进行角色分配才能使这场歌舞会尽可能精彩

解析

1. 对于每个人来说，只有4种情况，我们暴力搜索出每种情况，用参数统计目前的积累的结果，分别是

   > 歌手的唱歌能力之按位异或 r1
   >
   > 副歌手的唱歌能力之按位或 r2
   >
   > 舞者的跳舞能力之按位异或 r3
   >
   > 副舞者的跳舞能力之按位与 r4

2. 值得注意的是

   - 对于**异或运算**，可以用0去累加，因为任何数异或0都是其数字的本身
   - 对于或运算，可以用0去累加，因为任何和0做或运算都是其数字的本身
   - 对于**与运算**，我们用-1去累加，因为-1二进制是1111111111111..... ，任何数字与-1做与运算都是其本身

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
int a1, a2;//主 / 副歌手
int b1, b2;//主 / 副舞者
int a[11];//唱歌
int b[11];//跳舞
ll res = 0;//最大值
int n;
void dfs(int step, int start, int r1, int r2, int r3, int r4) {// r1 主歌， r2副歌
,r3主舞 ， r4副舞
    if (step == n + 1) {
        res = max(res, (ll)(r1 + r2)*(ll)(r3 + r4));
    }
    for (int i = start; i <= n; i++) {
        dfs(step + 1, i + 1, r1 ^ a[i], r2, r3, r4);
        dfs(step + 1, i + 1, r1, r2 | a[i], r3, r4);
        dfs(step + 1, i + 1, r1, r2, r3 ^ b[i], r4);
        if(b[i]>r4)//因为和比自己小的数字取异或，肯定会变小，加这个判断语句可以提高效率，也可
以不加
            dfs(step + 1, i + 1, r1, r2, r3, r4 & b[i]);
    }
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    for (int i = 1; i <= n; i++) {
        cin >> b[i];
    }
    dfs(1, 1, 0, 0, 0, -1);
```

```
    cout << res << endl;
}
```

上面的代码实际上**不用循环**，因为每个人**有且只有4种**选择（感谢ist20013 yjy的指正

```
#include<bitsdc++.h>
using namespace  std;
typedef long long ll;
ll n, ans=INT_MIN;
ll sing[15], dance[15];
void dfs(int step, ll a, ll b, ll c, ll d)
{
    if (step == n+1)
    {
        ans = max(ans, (a + b) * (c + d));
        return;
    }

    dfs(step + 1, a ^ sing[step], b, c, d);
    dfs(step + 1, a, b|sing[step], c, d);
    dfs(step + 1, a, b, c ^ dance[step], d);
    dfs(step + 1, a, b, c, d&dance[step]);
}
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> s[i];
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> dance[i];
    }
    dfs(1, 0, 0, 0, -1);
    cout << ans << endl;
}
```

全排列

解析

dfs的模板应用，对于dfs不熟悉的同学，网上有很多详细的教学。

下面有两种写法

1. dfs

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
int ans[8];
bool book[9] = {};//标记数组
void dfs(int step) {
    if (step == 8) {
        for (int i = 0; i < 8; i++) {
            cout << ans[i] << " ";
        }
        cout << endl;
    }
    for (int i = 1; i <= 8; i++) {
        if (book[i] == 0) {
            ans[step] = i;
            book[i] = 1;//标记 表示数字i已经使用过
            dfs(step + 1);
            book[i] = 0;
        }
    }
}
int main() {
    dfs(0);
}
```

2. STL全排列（使用前需要对数组排序)

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
int r[8] = {1,2,3,4,5,6,7,8 };
int main() {
    do {
        for (int i = 0; i < 8 ; i++) {
            cout << r[i] << " ";
        }
        cout << endl;
    } while (next_permutation(r, r + 8));
}
```

派件问题

解析

我们要明确的是，从原点p_0对于快递点$[p_1,p_2,p_3,p_4,p_i](i <= 8)$ 我们的轨迹的可能是p_0 + 快递点全排列，上一题也写了如何全排列了，和上题本质上是一样的。我们只需要计算全排列后的距离之和为多少，然后维护一个最小的距离之和就是最后的答案。

```
#include<bits/stdc++.h>
```

```
using namespace  std;
typedef long long ll;
int n;
bool book[11] = {};
int res;
struct point {
    int x, y;
};
point pr2[11];//用来全排列临时存放的
point pr[10];//主函数输入的
void dfs(int step) {
    if (step == n) {
        int temp = 0;
        for (int i = 0; i < n; i++) {
            if (i == 0) {//从原点触发
                temp += abs(pr2[i].x)+abs(pr2[i].y);
            }
            else {
                temp += abs(pr2[i - 1].x - pr2[i].x) + abs(pr2[i].y - pr2[i -
1].y);//和上一个点的距离
            }
        }
        res = min(res, temp);
    }
    for (int i = 0; i < n; i++) {
        if (book[i] == 0) {
            pr2[step] = pr[i];
            book[i] = 1;
            dfs(step + 1);
            book[i] = 0;
        }
    }
}
int main() {
    int t;
    cin >> t;
    while (t--) {
        cin >> n;
        for (int i = 0; i < n; i++) {
            point p;
            cin >> p.x;
            cin >> p.y;
            pr[i] = p;
        }
        res = 0x3f3f3f3f;//INF
        dfs(0);
        cout << res << endl;
    }
}
```

走方格-4

解析

1. 因为方格很小，所以用dfs暴力跑出两个点之间的路径可能即可
2. 这边用到了方向数组，因为在我们的dfs之中，每一层需要**面临的选择**有两种，就是不同的四个不同的方向。
3. 用了set集合去存储不同的路径

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
bool book[11][11];
int r[11][11];
int n,m;
int dx[] = { 1,0 };//只能往右 or 下走
int dy[] = { 0,1 };
set<int>s;//set集合 不允许有相同的元素
void dfs(int x,int y,int sum) {
    if (x== n-1 && y==m-1) {//判断是否到终点
        s.insert(sum);//加入到集合set 之中，由于集合的不可重复性质，因此这个集合的大小就是不同路线的之和。
    }
    for (int i = 0; i < 2; i++) {
        int tx = x + dx[i];//tx的意思是 try x 尝试这个tx ty能不能走
        int ty = y + dy[i];
        if (tx < n && ty < m && tx >= 0 && ty >= 0 && book[tx][ty] == 0 && r[tx][ty]!=0)  {
            book[tx][ty] = 1;
            dfs(tx, ty, sum+r[tx][ty]);
            book[tx][ty] = 0;
        }
    }
}

int main() {
    cin >> n >> m;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> r[i][j];
        }
    }
    dfs(0,0,r[0][0]);
    cout << s.size()<<endl;//输出集合的大小
}
```

班长请客

> 假设班上有n名同学，网上有m家店铺可以提供送餐服务。每家店铺由于规模大小不同，最多可提供的套餐数量也有所不同，分别为a1、a2、…、am份。每家店铺为套餐定的单价也有所不同，有的店铺卖得贵，有的店铺卖得便宜，分别为每份c1、c2、…、cm元。班长可以任意地从其中的某些店铺点任意份套餐（但在某家店铺点的套餐数量，不能超过该家店铺最多可提供的套餐数量）。每家店铺在提供送餐服务时，除了按份数和单价的乘积收取套餐费用外，还要收取一笔送餐费用t（固定值，不管送多少份都一样，但是如果从多家店铺订餐，则需要收取多笔送餐费）。

解析一（深度优先搜索

1. 由于配送费的存在，我们要尽可能在**尽量少**的商家购买，因此对于每个商家，策略是把商品**买断**或是**买到我们人数足够为止。**
2. 观察数据量，只有10个商家，每种商家就两种可能——**买或不买**，因此我们可以用O(n!)的深度优先搜索，全排列出我们购买商家的可能，如果最后这些商家可以使得我们的人数为0，对答案取一个min。

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
struct node {
    int a, c;//a 是商品数 c 是价格
};
node r[15];
bool book[15];
int n, m, t;
ll res;
void dfs(int left, ll sum) {//剩余学生人数   总花费
    if (left == 0) {
        res = min(res, sum);
        return;
    }
    for (int i =0; i < m; i++) {
        ll j = min(r[i].a, left);//要么买断，要么买够
        if (book[i] == 0) {
            book[i] = 1;
            dfs(left - j, sum + r[i].c * j + t);
            book[i] = 0;//回溯
        }
    }
}
int main() {
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m;
        res = LLONG_MAX;
        for (int i = 0; i < m; i++) {
            cin >> r[i].a;
        }
        for (int i = 0; i < m; i++) {
            cin >> r[i].c;
        }
        cin >> t;
        dfs(n, 0);
        cout << res << endl;
    }
}
```

- **优化：**可以观察到，对于我们要购买的商家是有存在**顺序问题**的，假设我们排列出要购买的商家为，a,b商家。如果b商家最贵，我们**对于b先进行买断操作**，然后对a买够剩余的学生数为止，就会出现并不是最优的情况，因此我们要优先对**便宜的商家进行购买**，
- **思路：**对整个结构体数组进行排序，然后在**dfs里面维护一个start参数**，从而保证每次都从小买到大。

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
struct node {
    int a, c;
};
node r[15];
int n, m;
ll res;
int t;

bool cmp(node q, node p) {
    return q.c < p.c;
}

void dfs(int left, int start, ll sum) {//深度 剩余学生数 开始数 总和
    if (left == 0) {
        //如果人数剩余为0了
        res = min(res, sum);
        return;//一定要退出
    }
    for (int i = start; i < m; i++) {//学生的人数
        ll x = min(r[i].a, left);
        dfs(left - x, i + 1, sum + x * r[i].c + t);
    }
}
int main() {
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m;
        res = LLONG_MAX;
        for (int i = 0; i < m; i++) {
            cin >> r[i].a;
        }
        for (int i = 0; i < m; i++) {
            cin >> r[i].c;
        }
        sort(r, r + m, cmp);
        cin >> t;
        dfs(n, 0, 0);
        cout << res << endl;
    }
}
```

解析二（二进制枚举

- 我们观察到，对于每个商家只有两种可能，买或者是不买，而且我们对其排序优化以后，**购买顺序是固定的**，所以我们能用二进制枚举出所有的可能。
- **二进制枚举:**如果有10个商家，1代表我们在这个商家进行**买断或买够**的操作。假设二进制串1010100000代表我们在1,3,5商家里进行购买
    - 范围：从0000000000到1111111111$((1 << 10) - 1)$

基础知识

对二进制串的操作:

对于十进制下的341，二进制下的101010101,如果我们想知道其末尾是否是1，很简单对其取&1即可，但如果我们想知道如何去求出这个二进制1的个数，我们可以不断让这个数字右移，直到这个数字为0。

```
int a=341;
int cnt=0;
while(a!=0){
    if(a&1==1){
        cnt++
    }
    a>>=1;
}
cout<<cnt;//1的个数
```

- 看来你已经完全懂了，那么**练习一下:**练习一下吐泡泡

- 本题代码：

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
struct node {
    int a, c;
};
node r[15];
int n, m, t;
ll res;
bool cmp(node q, node p) {
    return q.c < p.c;
}
int main() {
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m;
        res = LLONG_MAX;
        for (int i = 0; i < m; i++) {
            cin >> r[i].a;
        }
        for (int i = 0; i < m; i++) {
            cin >> r[i].c;
        }
        cin >> t;
        sort(r, r + m, cmp);
        for (int i = 1; i < (1 <<m); i++) {
            int x = i;//代表这种枚举可能的二进制串
            int cnt = 0;//第几个商家
            ll sum = 0;//花钱总额
            int left = n;//学生人数
            while (x!=0) {//不断取末尾
                if (x & 1 == 1) {
```

```
                    int numb=min(r[cnt].a, left);//本次购买数量      要么买够，要么买断取最小
                    sum += numb * r[cnt].c + t;
                    left -= numb;
                }
                if (left == 0)break;//人数够了 就不必要枚举这种可能了。
                cnt++;//下一个商家
                x >>= 1;//右移
            }
            if (left == 0) {//如果人数够 就和答案比较
                res = min(res, sum);
            }
        }
        cout << res << endl;
    }
}
```

身高排序

> 涂涂想把参加竞赛的队员们按身高从低到高排序排成一列纵队，也就是任意一对相邻的队员，排前面的队员身高总是小于等于后面的队员。
>
> 但他不想把这个问题变得如此简单，于是他给了自己一个例外的特权，允许在排序时，可以最多存在一对相邻的队员，排在前面的队员反而比后面的队员高。
>
> 已知所有队员的身高，问涂涂有多少种合理的排队方法?
>
> 注意：如果两个队员身高相同，他们之间交换位置，也算作另一种排队方法。涂涂可以选择不使用特权，就按顺序排序。

解析

1. 观察数据量，m<=10，直接暴力全排列即可，然后每次检查排列后的人数出现前面比后面高的次数是否<=1即可

```
#include<bits/stdc++.h>
using namespace  std;
typedef long long ll;
int r[3];
int ans[11];
bool book[11];
int m;
ll res;
void dfs(int step) {
    if (step == m) {
        int f = 0;//允许交换的次数 <=1
        for (int i = 0; i < m-1; i++) {
            if (ans[i] > ans[i + 1]) {
                f++;
            }
```

```
        }
        res += (f <= 1);//如果f<=1 就累加进去
    }
    for (int i = 0; i < m; i++) {
        if (book[i]==0) {
            ans[step] = r[i];
            book[i] = 1;
            dfs(step + 1);
            book[i] = 0;//回溯
        }
    }
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        cin >> m;
        res = 0;
        for (int i = 0; i < m; i++) {
            cin >> r[i];
        }
        dfs(0);
        cout << res << endl;
    }
}
```