NOIP苏州强校邀请赛 （五）题解

T1 好数（number）

40%

$O(n^4)$枚举即可。

70%

$A[i] = A[x] + A[y] + A[z]$，$x < y < z < i$

维护小于i的$A[x] + A[y] + A[z]$的和即可，时间复杂度$O(n^3)$

100%

$A[x] + A[y] + A[z]$通过类似01背包的方式维护，维护两个数字构成的和的情况，三个数字构成和的情况，用 `bitset` 优化加速即可，时间复杂度$O(\frac{n^3}{64})$

```
#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
using namespace std;
int n, a[5005], ans;
bitset<600005> dp[4];
int main() {
    freopen("number.in", "r", stdin);
    freopen("number.out", "w", stdout);
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i], a[i] += 100000;
    }
    for (int i = 1; i <= n; i++) {
        if (dp[3][a[i] + 200000])
            ans++;
        dp[1][a[i]] = 1;
        dp[2] |= (dp[1] << a[i]);
        dp[3] |= (dp[2] << a[i]);
    }
    cout << ans << endl;
    return 0;
}
```

100%

表达式类问题可以通过移项优化枚举策略

$A[i]=A[x]+A[y]+A[z] \rightarrow A[i]-A[z]=A[x]+A[y]$，$x<y<z<i$

枚举i和z，维护小于i的$A[x]+A[y]$的和即可。

复杂度$O(n^2)$

```cpp
# include <iostream>
# include <unordered_set>
using namespace std;

const int N = 5e3+5;

int n, ans, a[N];
unordered_set<int> s;//sum of two numbers

int main() {
    freopen("number.in", "r", stdin);
    freopen("number.out", "w", stdout);
    std::ios_base::sync_with_stdio(0); std::cin.tie(0); std::cout.tie(0);
    cin >> n;
    for (int i=1; i<=n; ++i) cin >> a[i];
    for (int i=1; i<=n; ++i) {
      bool f = 0;
      for (int j=1; j<i; ++j) {
        if (s.count(a[i]-a[j])) f = 1;
      }
      ans += f;
      for (int j=1; j<=i; ++j) s.insert(a[i]+a[j]);
    }
    cout << ans << '\n';
    return 0;
}
```

考察知识点

枚举 桶 bitset

T2 字符串SOS（sos）

20%

$n \leq 12$，可以本地打表或者手算打表。

仔细想想可以发现只有三类字符，S，O与其余24种字符，依据三种字符枚举即可$O(3^n)$。

40%

考虑动态规划，$dp[i][j][k]$表示前$1-i$个字符已经出现了j个SOS，现在匹配到SOS的第k（$k \in 0,1,2$）个字符，从$dp[i-1][j][0/1]$（非$'S'$和$'O'$结尾,$'S'$结尾)和$dp[i-1][j-1][2]('SO'$结尾)转移即可

时间复杂度$O(n^2)$

60-100%

发现先前的动态规划中我们并不在乎$j>=2$的结果，这些转移都是非常简单的，因此我们不需要转移那么多，时间复杂度$O(n)$，根据写法的常数大小有概率TLE一些点。

100%

可以简化之前的转移，根据题意可知，本题其实关键可以是构成了几个“SOS”，以及构成到了第几个字母，因此可以长度为i的字符串构成到了第几个字母主要划分的阶段。

我们可以建立第几个字母对应的字符串类型

```
// 0: ''
// 1: '*S'
// 2: '*SO'
// 3: '*SOS*'
// 4: '*SOS*S'
// 5: '*SOS*SO'
// 6: '*SOS*SOS*'
// 7: '*SOS*SOS*S'
// 8: '*SOS*SOS*SO'
// 9: '*SOS*SOS*SOS*'
```

由此我们可以依据每次添加一个字符来实现0-9之间状态的转移，可以发现只有三类字符，S，O与其余24种字符，依据三种字符与这10种状态的关系转移即可。

时间复杂度$O(n)$，空间复杂度$O(n)$

```
#pragma comment(linker, "/STACK:102400000,102400000")
#include<cstdio>
#include<cstring>
#include<ctime>
#include<iostream>
#include<algorithm>
using namespace std;
typedef long long LL;
const int maxn=1e6+5;
const LL mod=1e9+7;
int n;
LL f[maxn][10];

inline LL dfs(int len, int k){
    if(f[len][k]!=-1)return f[len][k];
```

```
    if(len==0){
        if(k==0)return f[len][k]=1ll;
        return f[len][k]=0ll;
    }

    LL res = 0;
    if(k==9){//*SOS*SOS*SO
        res += dfs(len-1,8) % mod;//*SOS*SOS*SO +S
        res %= mod;
        res += dfs(len-1,9) * 26 % mod;//*SOS*SOS*SOS* +any
        res %= mod;
    }else if(k==8){
        res += dfs(len-1,7) % mod; // *SOS*SOS*S +O
        res %= mod;
    }else if(k==7){
        res += dfs(len-1,6) % mod; // *SOS*SOS*  +S
        res %= mod;
        res += dfs(len-1,7) % mod; // *SOS*SOS*S +S
        res %= mod;
        // res += dfs(len-1,8) % mod; // *SOS*SOS*SO +S
    }else if(k==6){ // *SOS*SOS*
        res += dfs(len-1,5) % mod; // *SOS*SO + S
        res %= mod;
        res += dfs(len-1,6) * 25 % mod; // *SOS*SOS* +a-z -S;
        res %= mod;
        res += dfs(len-1,7) * 24 % mod; // *SOS*SOS*S +a-z -S -O;
        res %= mod;
        res += dfs(len-1,8) * 25 % mod; // *SOS*SOS*SO +a-z -S;
        res %= mod;
    }else if(k==5){ // *SOS*SO
        res += dfs(len-1,4) % mod; //*SOS*S + O
        res %= mod;
    }else if(k==4){ // *SOS*S
        res += dfs(len-1,3) % mod; //*SOS* +S
        res %= mod;
        res += dfs(len-1,4) % mod; //*SOS*S +S
        res %= mod;
        // res += dfs(len-1,5) % mod; //*SOS*SO +S
    }else if(k==3){ // *SOS*
        res += dfs(len-1,2) % mod; //*SO + S
        res %= mod;
        res += dfs(len-1,3) * 25 % mod; //*SOS* +a-z -S
        res %= mod;
        res += dfs(len-1,4) * 24 % mod; //*SOS*S +a-z -O -S
        res %= mod;
        res += dfs(len-1,5) * 25 % mod; //*SOS*SH +a-z -S
        res %= mod;
        // res += dfs(len-1,8) % mod; // *SOS*SOS*SH +S
        // res %= mod;
```

```
    }else if(k==2){ // *SH
        res += dfs(len-1,1) % mod; //*S +H
        res %= mod;
    }else if(k==1){ // *s
        res += dfs(len-1,0) % mod; // '' + S
        res %= mod;
        res += dfs(len-1,1) % mod; // *S + S
        res %= mod;
        // res += dfs(len-1,2) % mod; // *SO + S
        // res += dfs(len-1,6) % mod; // *SOS*SOS* +S
        // res %= mod;
    }else if(k==0){
        res += dfs(len-1,0) * 25 % mod; // '' +a-z -S
        res %= mod;
        res += dfs(len-1,1) * 24 % mod; // *S +a-z -O -S
        res %= mod;
        res += dfs(len-1,2) * 25 % mod; // *SO +a-z -S
        res %= mod;
    }
    return f[len][k]=res%mod;
}
// 0: ''
// 1: '*S'
// 2: '*SO'
// 3: '*SOS*'
// 4: '*SOS*S'
// 5: '*SOS*SO'
// 6: '*SOS*SOS*'
// 7: '*SOS*SOS*S'
// 8: '*SOS*SOS*SO'
// 9: '*SOS*SOS*SOS*'

int main(){
    freopen("sos.in", "r", stdin);
    freopen("sos.out", "w", stdout);
    cin>>n;
    f[0][0]=1;
    memset(f,-1,sizeof(f));
    dfs(n,9);
    printf("%lld\n",f[n][9]%mod);
    return 0;
}

```

100%

更优秀的做法，可以发现每次是$dp[i][0-9]$和$dp[i-1][0-9]$之间的线性转移，因此可以构建矩阵描述两者间的转移，时间复杂度$O(9^3\log n)$

考察知识点

动态规划 组合计数

T3 集训营的气球（ballon）

30%

考虑不用修改该怎么获得答案，发现这就是01背包，$dp[i][j]$表示前i个人里有j个人选择了购买气球的方案数

有了修改呢?

每次暴力重新跑01背包维护即可。

复杂度$O(qnc)$

50～100%

可以发现这就是个带修改的DP，考虑到这是动态DP，因此我们可以直接按照套路使用用线段树维护

模拟赛的版本用线段树能拿50分+（前三个样例的n,m没有保证是$\le 10^6$的！）

可以发现用递归线段树+DP+快读+$O2$是能过的。

不开$O2$就需要用到一个小技巧（容斥）优化常数，总的方案减去不满c个人买一血气球的方案（**正难则反的思想在后文还会用上**）， 这样每层循环的次数从就少了$(c+1)^2-c^2=2c+1$次。最终总方案减去不满c个人一血气球的方案。

复杂度为$O(nc^2+qc^2\log n)$

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

const int N = 1e5+5, C = 21, P = 1e4+7;

void read(int &var) {
  var = 0;
  int f = 1;
  char c;
  do {
    c = getchar();
    if (c=='-') f=-1;
  } while (c<48 || c>57);
  while (c>=48 && c<=57) var = var*10+(c-'0'), c = getchar();
  var *= f;
};
void write(int x) {
```

```
  if (x<0) putchar('-'), x = -x;
  if (x>9) write(x/10);
  putchar(x%10+'0');
}

int sz=1;
struct node {
  int ls, rs, l, r;//左儿子和右儿子的节点编号
  int dp[C], tot;
} tr[N<<2];

int n, c, q, a[N], b[N];

void pu(int u) {
  //拿左右两个儿子的dp数组做01背包
  memset(tr[u].dp, 0, sizeof(tr[u].dp));
  for (int i=0; i<c; ++i)
    for (int j=0; j+i<c; ++j) {
      tr[u].dp[i+j] = (1ll*tr[u].dp[i+j] + (1ll*tr[tr[u].ls].dp[i]*tr[tr[u].rs].dp[j])%P)%P;
    }
  tr[u].tot = (1ll*tr[tr[u].ls].tot * tr[tr[u].rs].tot)%P;
}
void build(int u, int l, int r) {
  tr[u].l = l, tr[u].r = r;
  if (l==r) {
    tr[u].dp[0] = b[l]%P;
    tr[u].dp[1] = a[l]%P;
    tr[u].tot = (a[l]+b[l])%P;
    return ;
  }
  tr[u].ls = ++sz;
  tr[u].rs = ++sz;
  int mid  = (l+r)>>1;
  build(tr[u].ls, l, mid);
  build(tr[u].rs, mid+1, r);
  pu(u);
}
void update(int u, int p, int x, int y) {
  if (tr[u].l==tr[u].r && tr[u].l==p) {
    tr[u].dp[0] = y;
    tr[u].dp[1] = x;
    tr[u].tot = (x+y)%P;
    return ;
  }
  int mid = (tr[u].l+tr[u].r)>>1;
  if (p<=mid) update(tr[u].ls, p, x, y);
  if (p>mid) update(tr[u].rs, p, x, y);
  pu(u);
}
```

```
}

int main() {
    freopen("balloon.in", "r", stdin);
    freopen("balloon.out", "w", stdout);
  read(n), read(c);
  for (int i=1; i<=n; ++i) read(a[i]);
  for (int i=1; i<=n; ++i) read(b[i]);
  build(1, 1, n);
  cin >> q;
  while (q--) {
    int p, x, y;
    read(p), read(x), read(y);
    update(1, p, x, y);
    int ans = tr[1].tot;
    for (int i=0; i<c; ++i) ans = (ans-tr[1].dp[i]+P)%P;
    write(ans), putchar('\n');
  }
  return 0;
}
/*
动态开点：不再用u<<1和u<<1|1作为下标，而是动态的节点编号
m次操作最多开m log n的节点 -> 2n-1

快读的优化
push_up的优化 -> 让DP[i]仅存下少于c的方案数量。用总方案数量减去不满c的方案数量
*/
```

100%

正难则反，考虑提前计算好不满足c的方案数，再用总方案数减去不满足c的方案数。

问题仍热是一个动态DP问题，但这是一个背包问题，换而言之，物品顺序没有影响，现在的线段树假设太强了，参考洛谷P4141消失之物，我们可以使用退背包的方式来撤销一个人对答案的贡献。

设tot为总方案数，x,y为修改后的第i个人的a_i,b_i。$inv(x)$为x的乘法逆元。

$$tot = \prod_{i=1}^{n} a_i + b_i \Rightarrow tot' = \frac{tot}{a_i + b_i} \times (x + y) = tot \times inv(a_i + b_i) \times (x + y) \tag{1}$$

设f为考虑该人前的dp数组，g为考虑该人之后的dp数组。

$$g_i = \begin{cases} f_i \times b, i = 0 \\ f_{i-1} \times a + f_i \times b, i > 0 \end{cases} \tag{2}$$

$$f_i = \begin{cases} g_i \times inv(b), i = 0 \\ (g_i - f_{i-1} \times a) \times inv(b), i > 0 \end{cases} \tag{3}$$

```
tot = (1ll*tot*inv(a[i]+b[i])*(x+y))%P;
f[0] = f[0] * inv(a[i]+b[i]);
//背包的时候反着做，所以退包的时候正着做
for (int i=1; i<c; ++i) {
  f[i] = (1ll*(f[i]-f[i-1]*a[i]+P)*inv(b))%P;
}
```

代码

```
# include <iostream>

using namespace std;

const int P = 1e9+7, N = 1e6+5, C = 25;

void read(int &x) {
  int f = 1;
  char c = getchar();
  x = 0;
  while (c<48 || c>57) c = getchar();
  while (c>=48 && c<=57) x = x*10 + c-'0', c = getchar();
  x *= f;
}
void write(int x) {
  if (x<0) putchar('-'), x = -x;
  if (x>9) write(x/10);
  putchar(x%10+'0');
}
int pow(int x, int k) {
  int res = 1;
  while (k) {
    if (k&1) res = (1ll*res*x)%P;
    x = (1ll*x*x)%P;
    k >>= 1;
  }
  return res;
}

int a[N], b[N], n, c, q, f[C], tot;

int main() {
    freopen("balloon.in", "r", stdin);
    freopen("balloon.out", "w", stdout);
    read(n), read(c);
    for (int i=1; i<=n; ++i) read(a[i]);
    for (int i=1; i<=n; ++i) read(b[i]);
    tot = f[0] = 1;

    for (int i=1; i<=n; ++i) {
```

```
    for (int i=1; i<=n; ++i) {
      for (int j=c-1; j>0; --j) f[j] = (1ll*f[j-1]*a[i]%P + 1ll*f[j]*b[i]%P)%P;
      f[0] = (1ll*f[0]*b[i])%P;
      tot = (1ll * tot * (a[i]+b[i]))%P;
    }
    read(q);
    while (q--) {
      int p, x, y;
      read(p), read(x), read(y);
      //修改
      tot = 1ll*tot*pow(a[p]+b[p], P-2)%P * (x+y)%P;
      //退背包
      int invb = pow(b[p], P-2);
      f[0] = 1ll*f[0]*invb%P;
      for (int i=1; i<c; ++i) f[i] = 1ll*(f[i]-1ll*f[i-1]*a[p]%P+P)*invb%P;
      //重新做01背包
      for (int i=c-1; i>0; --i) f[i] = (1ll*f[i-1]*x%P + 1ll*f[i]*y%P)%P;
      f[0] = 1ll*f[0]*y%P;
      a[p] = x, b[p] = y;
      int ans = tot;
      for (int i=0; i<c; ++i) ans = (ans - f[i] + P)%P;
      write(ans), putchar('\n');
    }
  return 0;
}
```

考察知识点

动态DP 组合计数 背包 退背包

T4 连通子树与树的重心（tree）

50%

首先找出原树的重心。

重心的判则有很多，在此采用最简单的一条：以一个节点为根重构树，如果每个子树大小都不超过 $\frac{n}{2}$，那么它为重心。

```
int n,sz[N],w[N],c[2];
vector<int> e[N];
void dfs(int u,int fa)
{
    sz[u]=1; w[u]=0;
    for(int v:e[u])
        if(v!=fa)
        {
```

```
        {
            dfs(v,u);
            sz[u]+=sz[v];
            w[u]=max(w[u],sz[v]);
        }
    w[u]=max(w[u],n-sz[u]);
    if(w[u]<=n/2) c[c[0]!=0]=u,++cnt;
}
```

这样，在整棵树遍历完以后，若`c[1]=0`，则重心仅有一个，反之则有两个。

接下来分类讨论：

1. 重心有两个a, b，那么它们之间必然有一条边。

将这条边割断，生成两棵树，根分别为a,b。设 $f[u][i]$ 表示以u为根的子树内包含点u且有i个结点的连通块有多少个，f数组可以通过树形背包得到。最后答案即为 $\sum_{i=1}^{\max(sz[u],sz[v])} f[u][i] \times f[v][i]$。两棵树可以共用一个f数组。时间复杂度 $O(n^2)$

2. 重心只有一个a。

类似刚刚的做法求出f数组后，可以设 $g[i][j]$ 表示以包含a且大小为i的一个连通块，其中最大子树大小为j的连通块个数，那么由重心的性质，答案即为 $\sum_{1\le i\le n, 2j\le i} g[i][j]$。

同样可以用树形背包解决，复杂度 $O(n^3)$。

100%

换一种方法，采用容斥。注意到 $f[a][i]$ 表示包含a且大小为i的连通块个数，那么只要求出这其中最大子树大小 $> i/2$ 的个数即可。

如果有一个子树t大小为 $k > i/2$，那么这样的t一定只有一个，所以可以枚举。这时不能直接把答案减去 $f[u][i-k] \times f[t][k]$，因为$f[u][i-k]$包含了取子树t的情况且这是一个背包问题，物品次序无关，因此需要进行一次"退背包"把t的贡献先删掉。

时间复杂度降至 $O(n^2)$.

```
#include<bits/stdc++.h>
using namespace std;
const int N=5e3+5;
const int MOD=10007;
const int INF=1e5;
int T,n,sz[N],w[N],c[2],f[N][N],g[N][N],cnt;
vector<int> e[N];
void add(int u,int v){e[u].push_back(v);e[v].push_back(u);}
void dfs(int u,int fa)
{
    sz[u]=1; w[u]=0;
    for(int v:e[u]) if(v!=fa)
    {
```

```
        {
            dfs(v,u);
            sz[u]+=sz[v];
            w[u]=max(w[u],sz[v]);
        }
        w[u]=max(w[u],n-sz[u]);
        if(w[u]<=n/2) c[c[0]!=0]=u,++cnt;
}
void dfs_dp(int u,int fa)
{
    sz[u]=1;
    f[u][0]=1;
    f[u][1]=1;
    for(int v:e[u]) if(v!=fa)
    {
        dfs_dp(v,u);
        sz[u]+=sz[v];
        for(int i=sz[u];i;--i)
            for(int j=1;j<=sz[v]&&j<=i-1;++j)
                (f[u][i]+=f[v][j]*f[u][i-j])%=MOD;

    }
}
void solve2()//重心有两个的情况
{
    dfs_dp(c[0],c[1]);//拆成两部分分别dp
    dfs_dp(c[1],c[0]);//拆成两部分分别dp
    int ans=0;
    for(int i=1;i<=n;++i)//合并计算答案
        (ans+=f[c[0]][i]*f[c[1]][i]%MOD)%=MOD;
    cout << ans << "\n";
}
void solve1()
{
    int u=c[0],ans=0;
    dfs_dp(u,0);//求出重心计算无约束dp结果
    for(int i=1;i<=n;++i) (ans+=f[u][i])%=MOD;
    for(int v:e[u])
    {
        for(int i=1;i<=n;++i)
        {
            g[v][i]=f[u][i];
            for(int k=1;k<=min(i,sz[v]);++k)//退背包，从u中删掉v的信息
                g[v][i]=(g[v][i]-g[v][i-k]*f[v][k]%MOD+MOD)%MOD;
        }
    }
    for(int i=1;i<=n;++i)
    {
        for(int v:e[u])
            for(int k=(i+1)/2;k<=i&&k<=sz[v];++k)//容斥，所有无约束答案-不合法的答案（v超过
```

```
n/2*其他方案)
                ans=(ans-f[v][k]*g[v][i-k]%MOD+MOD)%MOD;
    }
    cout << ans << "\n";
}
int main()
{
    freopen("tree.in", "r", stdin);
    freopen("tree.out", "w", stdout);
    T = 1;
    for(int kase=1;kase<=T;++kase)
    {
        cin>>n;
        cnt=0;
        memset(c,0,sizeof(c));
        memset(f,0,sizeof(f));
        for(int i=1;i<=n;++i) e[i].clear();
        for(int i=1,u,v;i<=n-1;++i)
        {
            cin>>u>>v;
            add(u,v);
        }
        dfs(1,0);
        if(c[1]) solve2();
        else solve1();
    }
    return 0;
}
```

考察知识点

动态规划 重心 退背包