Problem
Last year, we asked you to help us convert expensive metals into lead. (You do not need to know anything about the revious problem to solve this one.) But your country's leader is still greedy for more lead!

There are M metals known in the world; lead is metal number 1 on your periodic table. Your country's leader has as ed you to use the metals in the treasury to make as much lead as possible.

For each metal (including lead), you know exactly one formula that lets you destroy one gram of that metal and creat one gram each of two metals. (It is best not to think too much about the principle of mass conservation!) Note that it is possible that the formula for the i-th metal might produce the i-th metal as one of the products. The formulas do n t work with partial grams. However, you can use each formula as often as you would like (or not at all), as long as y u have a gram of the required ingredient.

If you make optimal choices, what is the largest number of grams of lead you can end up with, or is it unbounded? If it is not unbounded: since the output can be a really big number, we only ask you to output the remainder of dividing the result by the prime 109+7 (that is, 1000000007).

Input
The first line of the input gives the number of test cases, T. T test cases follow. Each begins with one line with an int ger M: the number of metals known in the world. Then there are M more lines with two integers Ri1 and Ri2 each; t e i-th of these lines, counting starting from 1, indicates that you can destroy one gram of metal i to create one gram o metal Ri1 and one gram of metal Ri2. Finally, there is one line with M integers G1, G2, ..., GM; Gi is the number of grams of metal i in the treasury. Lead is metal 1.

Output
For each test case, output one line containing Case #x: y where x is the test case number (starting from 1). If there is o bound on the maximum amount of lead that can be produced, y must be UNBOUNDED. Otherwise, y must be the argest amount of lead, in grams, that you can end up with, modulo the prime 109+7 (that is, 1000000007).

Limits
$1 \leq Ri1 < Ri2 \leq M$, for all i.
Time limit: 20 seconds per test set.
Memory limit: 1GB.

Test set 1 (Visible)
$1 \leq T \leq 100$.
$2 \leq M \leq 10$.
$0 \leq Gi \leq 10$, for all i.

Test set 2 (Hidden)
$1 \leq T \leq 100$.
$2 \leq M \leq 100$.
$0 \leq Gi \leq 109$, for all i.

Test set 3 (Hidden)
$1 \leq T \leq 5$.
$2 \leq M \leq 105$.
$0 \leq Gi \leq 109$, for all i.

Sample

Input

Output

```
3
2
1 2
1 2
1 0
2
1 2
1 2
0 0
4
2 4
3 4
2 4
2 3
10 10 10 10
```

```
Case #1: UNBOUNDED
Case #2: 0
Case #3: 10
```

In Sample Case #1, you have one formula that turns 1 gram of lead into 1 gram of lead and 1 gram of the second met l, and another formula that turns 1 gram of the second metal into 1 gram of lead and 1 gram of the second metal. Yo can alternate between these formulas to produce as much of both metals as you want.

Sample Case #2 has the same formulas as Sample Case #1, but you have no metals to start with!

In Sample Case #3, none of the formulas help you produce more lead, so you cannot end up with more lead than you started with.

Solution:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
#define PB push_back
#define MP make_pair
#define LL long long
#define int LL
#define FOR(i,a,b) for(int i = (a); i <= (b); i++)
#define RE(i,n) FOR(i,1,n)
#define REP(i,n) FOR(i,0,(int)(n)-1)
#define R(i,n) REP(i,n)
#define VI vector<int>
#define PII pair<int,int>
#define LD long double
#define FI first
```

```
#define SE second
#define st FI
#define nd SE
#define ALL(x) (x).begin(), (x).end()
#define SZ(x) ((int)(x).size())

#define unordered_map __fast_unordered_map
template<class Key, class Value, class Hash = std::hash<Key>>
using unordered_map = __gnu_pbds::gp_hash_table<Key, Value, Hash>;

template<class C> void mini(C &a4, C b4) { a4 = min(a4, b4); }
template<class C> void maxi(C &a4, C b4) { a4 = max(a4, b4); }

template<class TH> void _dbg(const char *sdbg, TH h){ cerr<<sdbg<<'='<<h<<endl; }
template<class TH, class... TA> void _dbg(const char *sdbg, TH h, TA... a) {
  while(*sdbg!=',')cerr<<*sdbg++;
  cerr<<'='<<h<<','; _dbg(sdbg+1, a...);
}

template<class T> ostream &operator<<(ostream& os, vector<T> V) {
  os << "["; for (auto vv : V) os << vv << ","; return os << "]";
}
template<class L, class R> ostream &operator<<(ostream &os, pair<L,R> P) {
  return os << "(" << P.st << "," << P.nd << ")";
}

#ifdef LOCAL
#define debug(...) _dbg(#__VA_ARGS__, __VA_ARGS__)
#else
#define debug(...) (__VA_ARGS__)
#define cerr if(0)cout
#endif

const int Mod = 1e9 + 7;
const int Bound = 1e8;

struct ModMax {
  int mod_val, max_val;
  bool is_infinity;

  ModMax() : mod_val(0), max_val(0), is_infinity(false) {}
  ModMax(int x) : mod_val(x % Mod), max_val(min(x, Bound)), is_infinity(false) {}
  ModMax(int a, int b) : mod_val(a), max_val(b), is_infinity(false) {}

  static ModMax Inf() {
    ModMax m; m.is_infinity = true; return m;
  }

  ModMax operator+(const ModMax &other) const {
    if (is_infinity || other.is_infinity) { return Inf(); }
    return {(mod_val + other.mod_val) % Mod,
            min(max_val + other.max_val, Bound)};
  }
  ModMax operator*(int f) const {
    if (f && is_infinity) { return Inf(); }
    return {((LL)mod_val * f) % Mod, (int)min<LL>((LL)max_val * f, Bound)};
  }
};

struct Testcase {
  int test_idx_;
  Testcase(int tidx) : test_idx_(tidx) {}

  int N;
  vector<array<int, 2>> nexts;
  vector<int> starts;
  vector<VI> rev_graph;

  vector<bool> lead_visit;
  VI order;
  VI pos_in_order;
  vector<ModMax> coefs;

  void DfsLead(int v) {
    lead_visit[v] = true;
    for (int s : rev_graph[v]) {
      //if (!pos_visit[s]) { continue; }
      if (!lead_visit[s]) { DfsLead(s); }
    }
    pos_in_order[v] = SZ(order);
    order.PB(v);
  }

  /*&void DfsPos(int v) {
    pos_visit[v] = true;
    if (!v) { return; }
    for (int x : {0, 1}) {
      const int s = nexts[v][x];
      if (!pos_visit[s]) { DfsPos(s); }
    }
  }*/

  int Solve() {
    cin >> N;
    nexts.resize(N); starts.resize(N);
    for (int i = 0; i < N; ++i) {
      cin >> nexts[i][0] >> nexts[i][1];
      --nexts[i][0]; --nexts[i][1];
    }
    for (int i = 0; i < N; ++i) { cin >> starts[i]; }

    rev_graph.resize(N);

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

```
    for (int x : {0, 1}) {
      rev_graph[nexts[i][x]].PB(i);
    }
  }

  lead_visit.resize(N);
  pos_in_order.resize(N);

  DfsLead(0);
  coefs.resize(N);
  coefs[0] = ModMax{1};

  debug(pos_in_order);

  for (int i = 1; i < N; ++i) {
    if (!lead_visit[i]) { continue; }
    for (int x : {0, 1}) {
      const int j = nexts[i][x];
      if (!lead_visit[j]) { continue; }
      if (pos_in_order[j] <= pos_in_order[i]) {
        coefs[i] = ModMax::Inf();
      }
    }
  }

  auto rord = order;
  reverse(ALL(rord));
  for (int i : rord) {
    if (!i) { continue; }
    if (!lead_visit[i]) { continue; }
    for (int x : {0, 1}) {
      const int j = nexts[i][x];
      if (!lead_visit[j]) { continue; }
      if (pos_in_order[j] > pos_in_order[i]) {
        coefs[i] = coefs[i] + coefs[j];
      }
    }
  }

  ModMax answer;
  for (int i = 0; i < N; ++i) {
    if (!lead_visit[i]) { continue; }
    answer = answer + coefs[i] * starts[i];
  }

  if (answer.is_infinity) { return -1; }
  if (answer.max_val == 0) { return 0; }

  ModMax back_val;
  for (int x : {0, 1}) {
    const int s = nexts[0][x];
    if (lead_visit[s]) { back_val = back_val + coefs[s]; }
    }

    if (back_val.is_infinity || back_val.max_val >= 2) { return -1; }
    return answer.mod_val;
  }

  void Run() {
    const int ans = Solve();
    cout << "Case #" << test_idx_ << ": ";
    if (ans == -1) {
      cout << "UNBOUNDED\n";
    } else {
      cout << ans << "\n";
    }
  }
};


int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  cout << fixed << setprecision(11);
  cerr << fixed << setprecision(6);

  int T;
  cin >> T;
  for (int i = 1; i <= T; ++i) {
    Testcase(i).Run();
  }
}
```