```
#include <iostream>
#include <map>
#include <climits>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>


typedef boost::adjacency_list_traits<boost::vecS, boost::vecS, boost::bidirectionalS
> Traits;

typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::bidirectionalS,
                  boost::no_property,
                  boost::property<boost::edge_capacity_t, long,
                          boost::property<boost::edge_residual_capacity_t, long,
                                  boost::property<boost::edge_reverse_t, Traits::edg
e_descriptor>>>> Graph;
typedef Graph::vertex_descriptor Vertex;
typedef Graph::edge_descriptor Edge;
typedef boost::property_map<Graph, boost::edge_capacity_t>::type EdgeCapacityMap;
typedef boost::property_map<Graph, boost::edge_residual_capacity_t>::type ResidualCa
pacityMap;
typedef boost::property_map<Graph, boost::edge_reverse_t>::type EdgeReverseMap;

class EdgeAdder {
  Graph &g;
  EdgeCapacityMap &capacity_map;
  EdgeReverseMap &reverse_map;

public:
  EdgeAdder(Graph &g,
        EdgeCapacityMap &capacity_map,
        EdgeReverseMap &reverse_map)
    :g(g), capacity_map(capacity_map), reverse_map(reverse_map) {}

  void add_edge(int u, int v, int cap) {
    Edge e, rev_e; bool s;
    boost::tie(e, s) = boost::add_edge(u, v, g);
    boost::tie(rev_e, s) = boost::add_edge(v, u, g);

    // Add capcities
    capacity_map[e] = cap;
    capacity_map[rev_e] = 0;

    // Add reverse edge
    reverse_map[e] = rev_e;
    reverse_map[rev_e] = e;
  }
};

void testcase() {
  // 0. Read in values
  int l, p; std::cin >> l >> p;

  // Read in locations + num garrisoned and num needed to defend
  std::vector<int> num_garrisoned(l), num_defenders(l);
  int total_num_defenders = 0;
  for(int i = 0; i < l; i++) {
    int g_i, d_i; std::cin >> g_i >> d_i;
    num_garrisoned[i] = g_i;
    num_defenders[i] = d_i;
    total_num_defenders += d_i;
  }

  // Track input graph, but keep weights seperate
  // Done so that it's easier to read incoming and outgoing edges for vertices
  Graph input_g(l);
  std::map<Edge, int> min_cap, max_cap;

  // Read in paths
  for(int i = 0; i < p; i++) {
    int f_i, t_i, min_i, max_i; std::cin >> f_i >> t_i >> min_i >> max_i;

    // Add edge to input graph
    Edge e; bool s;
    boost::tie(e, s) = boost::add_edge(f_i, t_i, input_g);

    // Record min and max capacities
    min_cap[e] = min_i;
    max_cap[e] = max_i;
  }

  // Create graph with sink and target
  Graph g(l + 2);
  int s = l;
  int t = s + 1;
  EdgeCapacityMap capacity_map = boost::get(boost::edge_capacity, g);
  EdgeReverseMap reverse_map = boost::get(boost::edge_reverse, g);

  // Create edge adder helper
  EdgeAdder ea(g, capacity_map, reverse_map);

  // 1. For every vertex v
  // Compute s -> v, with c(s -> v) = g_v + sum(min(in(v)) - sum(min(v))
  // Compute v -> t, with c(v -> t) = d_v
  for(Vertex v = 0; v < l; v++) {
    // 1.a Compute sum(min(in(v)))
    int sum_min_in = 0;
    Graph::in_edge_iterator in_itr, in_end;
    for(boost::tie(in_itr, in_end) = boost::in_edges(v, input_g); in_itr != in_end;
in_itr++) {
      sum_min_in += min_cap[*in_itr];
    }

    // 1.b Compute sum(min(out(v)))
    int sum_min_out = 0;
    Graph::out_edge_iterator out_itr, out_end;
    for(boost::tie(out_itr, out_end) = boost::out_edges(v, input_g); out_itr != out_
end; out_itr++) {
      sum_min_out += min_cap[*out_itr];
    }

    // 1.c add edge s -> v with capacity g_i + sum_min_in - sum_min_out
    int s_v_cap =  num_garrisoned[v] + sum_min_in - sum_min_out;
    if (s_v_cap > 0)
      ea.add_edge(s, v, s_v_cap);

    // 1.d add edge v -> with capacity d_i
    ea.add_edge(v, t, num_defenders[v]);
  }

  // 2. For every edge u -> v (excluding s and t)
  // Compute u -> v, with c(u -> v) = max(u -> v) - min(u -> v)
  Graph::edge_iterator e_itr, e_end;
  for(boost::tie(e_itr, e_end) = boost::edges(input_g); e_itr != e_end; e_itr++) {
    ea.add_edge(boost::source(*e_itr, input_g),
         boost::target(*e_itr, input_g),
         max_cap[*e_itr] - min_cap[*e_itr]);
  }

  // 3. Compute (s, t)-flow f
  long f = boost::push_relabel_max_flow(g, s, t);

  // 4. If (f < sum d_v) no else yes
  if (f < total_num_defenders) {
    std::cout << "no" << std::endl;
  } else {
    std::cout << "yes" << std::endl;
  }
}

int main() {
  std::ios_base::sync_with_stdio(false);
  int t; std::cin >> t;
  while(t--) testcase();
  return 0;
}
```