```cpp
class Ancestor {
private:
  int n;
  vvi lst;
  vvi table;
  vi from;
  vi visited, departed;
  deque<int> deq;
  void dfs(int a, int &t) {
    for (int i = 0; i < deq.size(); i = i * 2 + 1) {
      table[a].push_back(deq[i]);
    }
    visited[a] = t++;
    deq.push_front(a);
    Foreach(b, lst[a]) {
      if (from[b] == INT_MIN) {
        from[b] = a;
        dfs(b, t);
      }
    }
    deq.pop_front();
    departed[a] = t++;
  }
public:
  Ancestor(const vvi &lst, vi roots = { 0 }) {
    n = lst.size();
    this->lst = lst;
    table = vvi(n);
    from = vi(n, INT_MIN);
    visited.resize(n);
    departed.resize(n);
    int t = 0;
    Foreach(root, roots) {
      from[root] = -1;
      dfs(root, t);
    }
  }
  bool is_ancestor(int des, int anc) {
    return visited[anc] < visited[des]
      && departed[des] < departed[anc];
  }
  int lowest_common_ancestor(int x, int y) {
    if (x == y) return x;
    if (is_ancestor(x, y)) return y;
    if (is_ancestor(y, x)) return x;
    Loop1(i, table[x].size() - 1) {
      if (is_ancestor(y, table[x][i])) {
        return lowest_common_ancestor(table[x][i - 1], y);
      }
    }
    return lowest_common_ancestor(table[x].back(), y);
  }
  int get_ancestor(int des, int k) {
    if (k == 0) return des;
    int l = int(log2(k));
    if (l >= table[des].size()) return -1;
    else return get_ancestor(table[des][l], k - (1 << l));
  }
  // return first value causing "t" in evalfunc that returns descendant->[f,...,f,t,...,t]->root
  // NOTE: if [f,...,f] then return -1
  template<typename bsargv_t>
  int upper_bsearch(int des, const bsargv_t &v, bool(*evalfunc)(int, const bsargv_t&)) {
    if (evalfunc(des, v)) return des;
    if (table[des].size() == 0) return -1;
    Loop1(i, table[des].size() - 1) {
      if (evalfunc(table[des][i], v)) {
        return upper_bsearch(table[des][i - 1], v, evalfunc);
      }
    }
    return upper_bsearch(table[des].back(), v, evalfunc);
  }
  // return last value causing "t" in evalfunc that returns descendant->[t,...,t,f,...,f]->root
  // NOTE: if [f,...,f] then return -1
  template<typename bsargv_t>
  int lower_bsearch(int des, const bsargv_t &v, bool(*evalfunc)(int, const bsargv_t&)) {
    if (!evalfunc(des, v)) return -1;
    if (table[des].size() == 0) return des;
    Loop(i, table[des].size()) {
      if (!evalfunc(table[des][i], v)) {
        if (i == 0) return des;
        else return lower_bsearch(table[des][i - 1], v, evalfunc);
      }
    }
    return lower_bsearch(table[des].back(), v, evalfunc);
  }
};

```