```
#include <map>
#include <queue>
#include <limits>
#include "raceState.hpp"

const int searchDepth = 10;
const int MAX_DEPTH = 1;



struct PlayerState {
    //è\207ªæ©\237ã\201®ä½\215ç½®ã\201¨é\200\237å°¦
    Point position;
    IntVec velocity;

    //æ¯\224è¼\203ã\201\231ã\202\213ã\201\237ã\202\201ã\201®æ¯\224è¼\203æ¼\224ç®\227å-\220
    bool operator<(const PlayerState &ps) const {
        return
                position != ps.position ?
                position < ps.position :
                velocity < velocity;
    }

    bool operator !=(const PlayerState &ps)const{
        return position != ps.position;
    }

    int operator -(const PlayerState &ps)const{
        return position.y - ps.position.y;
    }
    int operator +(const PlayerState &ps)const{
        return position.y + ps.position.y;
    }

    PlayerState(Point p, IntVec v) : position(p), velocity(v){}
};

struct Candidate {
    int step;            // Steps needed to come here
    PlayerState state;         // State of the player
    Candidate *from;        // Came here from this place
    IntVec how;            //   with this acceleration
    Candidate(int t, PlayerState s, Candidate *f, IntVec h) :
            step(t), state(s), from(f), how(h) {}
    int cost;
};

// æ¯\224è¼\203é\226¢æ\225°ã\202\222å®\232ç¾©
bool GoodEvalOrder( const Candidate* left, const Candidate* right ) {
    return left->cost < right->cost;
}

//globalå®£è¨\200
//å\200\231è£\234ã\202\222æ ¼ç´\215ã\201\231ã\202\213queue
queue<Candidate *> candidates;
//ã\201\237ã\201©ã\202\212ç\235\200ã\201\221ã\202\213ã\201\213ã\202\222è¨\230é\214²ã\201\231ã\202\213map
map<PlayerState, Candidate *> reached;


//è©\225ä¾¡é\226¢æ\225° ä»\212ã\201¨æ¬¡ã\201®è¡\214ã\201\217å ´æ\211\200ã\200\201ç\233¤é\235¢ã\201®ç\212¶æ\205\213
int calcCost(Candidate* now,Point nextPos){
    int ret=0;
    //è©\225ä¾¡å\200¤
    //yæ\226¹å\220\221ã\201\214é«\230ã\201\221ã\202\214ã\201°é«\230ã\201\204æ\226¹ã\201\214è\211¯ã\201\204
    ret+=(nextPos.y)*3;
    //xæ\226¹å\220\221ã\201¸ã\201®ç§»å\213\225ã\201\214å¤§ã\201\215ã\201\221ã\202\214ã\201°å¤§ã\201\215ã\201\204ã\201»ã\201©è\211¯ã\201\204
    ret+=abs(now->state.position.x-nextPos.x);
    //å£\201ã\201\213ã\202\211é\233¢ã\202\214ã\201¦ã\201\204ã\201\237æ\226¹ã\201\214è\211¯ã\201\204
    //costX-=abs(nextPos.x-7);


    return ret;
}
//æ¬¡ã\201®å\200\231è£\234
//æ¬¡ã\201®9æ\226¹å\220\221ã\201«è¡\214ã\201£ã\201\237æ\231\202ã\201®stateã\201\214é\205\215å\210\227ã\201¨ã\201\227ã\201¦è¿\224ã\201\225ã\202\214ã\202\213
vector<Candidate*> generate_next_status(Candidate *ca,const Course &course){

//æ¬¡ã\201«ã\201\204ã\201\221ã\202\213ï¼\231^stepå\200\213(reachedã\201§ä¸\200å¿\234æ\236\235å\210\210ã\202\212ã\201\227ã\201¦ã\202\213)ã\201®å\200\231è£\234ã\202\222æ ¼ç´\215ã\201\231ã\202\213é\205\215å\210\227
    vector<Candidate*> ret;
    //å\210\235æ\234\237å\214\226
    while (!candidates.empty()) candidates.pop();
    reached.clear();

    candidates.push(ca);
    while(!candidates.empty()){
        Candidate *now = candidates.front();
        //cerr<<"here"<<endl;
        candidates.pop();
        //è¡\214ã\201\215å\205\210ã\202\2229ç¨®é¡\236å\205¨ã\201¦ã\203«ã\203¼ã\203\227
        for(int cay = 1; cay != -2;cay--){
            for(int cax = -1;cax != 2; cax++){
                //cerr<<"search: "<<cay<<" "<<cax<<endl;
                //cerr<<"now->step: "<<now->step<<endl;
                //æ¬¡ã\201®é\200\237å°¦
                IntVec nextVelo = now->state.velocity + IntVec(cax,cay);
                //æ¬¡ã\201®ä¸\200
                Point nextPos = now->state.position + nextVelo;

                //ã\202²ã\203\206ã\203\203ã\203\227æ\225°ã\201\2140ã\201§ã\201ªã\201\217é\232\234å®³ç\211©ã\201«è¡\235çª\201ã\201\227ã\201ªã\201\204
                if(!course.obstacled(now->state.position,nextPos)){
                    //æ¬¡ã\201®ã\203\227ã\203¬ã\202¤ã\203¤ã\203¼ã\201®ä½\215ç½®ã\200\201é\200\237å°¦ã\202\222æ¬¡ã\201®å\200\231è£\234å¤\211æ\225°ã\201«æ ¼ç´\215
                    PlayerState next(nextPos,nextVelo);
                    //cerr<<"nextCand"<<endl;
                    Candidate *nextCand =
                            new Candidate(now->step + 1, next, now, IntVec(cax, cay));
                    if (reached.count(next) == 0) { //ã\201\235ã\201\223ã\201«ã\201\237ã\201©ã\202\212ç\235\200ã\201\217ã\201®ã\201\214æ\234\200å\210\235ã\201®å\200\231è£\234ã\201§ã\201\202ã\202\214ã\201°
                        //æ\216¢ç´¢æ•±ã\201\225ã\202\210ã\202\212ã\202\202æµ\205ã\201\217ã\200\200ã\201\213ã\201¤ã\200\200ã\202³ã\203¼ã\202¹ã\202\222ã\201¯ã\201¿å\207ºã\201\227ã\201¦ã\201\204ã\201ªã\201\221ã\202\214ã\201°
                        if (now->step < searchDepth && nextPos.y <= course.length && nextPos.x > 0) {


                            //è©\225ä¾¡å\200¤ã\201®è¨\210ç®\227
                            nextCand->cost=calcCost(now,nextPos);
                            //æ¬¡ã\201®å\200\231è£\234ã\201«è¿½å\212
```

```
                            candidates.push(nextCand);
                        }
                        //nextã\201«ã\201\237ã\201©ã\202\212ç\235\200ã\201\221ã\202\213ã\201\223ã\201¨ã\202\222è¨\230é\214²
                        reached[next] = nextCand;
                        //cerr<<"statusã\201«push?"<<endl;
                        //æ¬¡è¡\214ã\201\221ã\202\213çµ\220æ\236\234ã\202\222æ ¼ç´\215
                        ret.push_back(nextCand);
                    }


                }
            }
        }

    }

    return ret;
}

// rs.positon è\207ªæ©\237ã\201®ç\217¾å\234¨å\234°ã\200\200rs.velocity è\207ªæ©\237ã\201®ç\217¾å\234¨ã\201®é\200\237åº¦ course ã\202³ã\203¼ã\202¹ã\201®æ\203\205å ±
IntVec play(RaceState &rs, const Course &course) {

    cerr<<"s"<<endl;
    //å\210\235æ\234\237å\214\226
    while (!candidates.empty()) candidates.pop();
    reached.clear();

    cerr<<"t"<<endl;
    //initialã\201¯ã\203\227ã\203¬ã\202¤ã\203¤ã\203¼ã\201®ç\212¶æ\205\213 è\207ªæ©\237ã\201®ä½\215ç½®ã\200\201é\200\237åº¦ã\200\200ç\233¸æ\211\213ã\201®ä½\215ç½®ã\200\201é\200\237åº¦
    PlayerState initial(rs.position, rs.velocity);
    //step å\210\235æ\234\237ã\201®ã\203\227ã\203¬ã\202¤ã\203¤ã\203¼ã\201®ç\212¶æ\205\213ã\200\200æ¬¡ã\201®ã\203\227ã\203¬ã\202¤ã\203¤ã\203¼ã\201®ç\212¶æ\205\213ã\200\200é\200\237åº¦
    Candidate initialCand(0, initial, nullptr, IntVec(0, 0));
    //reached[initial]ã\201¯initialã\201«è¾¿ã\202\212ã\201¤ã\201\221ã\202\213ã\201\223ã\201¨ã\202\222ä¿\235å-\230ã\201\231ã\202\213map
    //reached[initial] = &initialCand;
    //æ\234\200ã\202\202è\211¯ã\201\204å\200\231è£\234ã\202\222ä¿\235å-\230ã\201\231ã\202\213å¤\211æ\225°
    candidates.push(&initialCand);


    //æ\216¢ç´¢ã\201®æ•±ã\201\225ã\201\224ã\201¨ã\201®å\200\231è£\234ã\202\222æ ¼ç´\215ã\201\231ã\202\213é\205\215å\210\227
    vector<Candidate*> status[MAX_DEPTH+1];
    status[0].push_back(&initialCand);


    for(int depth=0;depth<MAX_DEPTH;depth++){
        //ã\203\223ã\203¼ã\203 ã\202µã\203¼ã\203\201ã\201®å\210\207ã\202\212æ\215¨ã\201¦é\203¨å\210\206
        //ã\201\223ã\201\223ã\201«æ\233¸ã\201\217
        const int BEAM_WIDTH = 50;
        //è©\225ä¾¡å\200¤ã\201\214è\211¯ã\201\204é \206ã\201§sort
        sort(status[depth].begin(), status[depth].end(), GoodEvalOrder);
        //ä¸ \212ä½\215BEAM_WIDTHã\201«å\205¥ã\202\211ã\201ªã\201\204ã\202\202ã\201®ã\202\222å\211\212é\231¤
        if(status[depth].size()>BEAM_WIDTH)
            status[depth].erase(status[depth].begin()+BEAM_WIDTH,status[depth].end());
        //ã\201\223ã\201\223ã\201¾ã\201§

        cerr<<"depth:"<<depth<<endl;

        //æ•±ã\201\225 depthã\201®ç\212¶æ\205\213ã\202\222å\210\227æ\214\231
        for(Candidate* state : status[depth]){
            //cerr<<"generate_next_status(state,course).size: "<<endl;
            for(Candidate* next_state: generate_next_status(state,course)) {
                //cerr<<"next_State";
                status[depth + 1].push_back(next_state);
                //cerr<<"push_Backã\201\227ã\201\237ï¼\201"<<endl;
            }
            //cerr<<"ä¸ -ã\201®forã\203«ã\203¼ã\203\227çµ\202º\206";
        }
        //cerr<<"depth serchå•¡å\233\236ä¸ -ã\200\234"<<endl;
    }

    //cerr<<"sizeè¨\210ç®\227ã\201\231ã\202\213ã\202\210"<<endl;

    int size = (int)status[MAX_DEPTH].size();
    cerr<<"size="<<size<<endl;
    int temp=-1;
    Candidate *best=&initialCand;
    for (int i=0 ;i<size;i++){
        if(temp<status[MAX_DEPTH][i]->state.position.y){
            temp=status[MAX_DEPTH][i]->state.position.y;
            best=status[MAX_DEPTH][i];
        }
    }
    //cerr<<"temp :" <<temp<<endl;
    cerr<<status[MAX_DEPTH].size()<<endl;

    if(status[MAX_DEPTH].size()==0){
        cerr<<"å\200\231è£\234ã\201\214ã\201ªã\201\204ã\201¿ã\201\237ã\201\204ã\201ã\201\213ã\202\211å\210\235æ\234\237å\200¤å\205¥ã\202\214ã\202\213ã\201-ï¼\201"<<endl;
        best = &initialCand;
    }


    cerr<<"best?"<<endl;

    if (best == &initialCand) {
        // No good move found
        // Slowing down for a while might be a good strategy
        int ax = 0, ay = 0;
        if (rs.velocity.x < 0) ax += 1;
        else if (rs.velocity.x > 0) ax -= 1;
        if (rs.velocity.y < 0) ay += 1;
        else if (rs.velocity.y > 0) ay -= 1;
        return IntVec(ax, ay);
    }
    Candidate *c = best;
    while (c->from != &initialCand) c = c->from;
    return c->how;
}

int main(int argc, char *argv[]) {
    Course course(cin);
    cout << 0 << endl;
    cout.flush();
    while (true) {
        RaceState rs(cin, course);
        IntVec accel = play(rs, course);
        cerr<<"accel.x"<<accel.x<<" "<<"accel.y"<<accel.y<<endl;
        cout << accel.x << ' ' << accel.y << endl;
    }
```

```
233: }
```