Lab 4-2 Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <time.h>

#define MAX\_DIM 3

struct kd\_node\_t {

double x[MAX\_DIM];

struct kd\_node\_t\* left, \* right;

};

struct range {

double x;

double y;

double w;

double h;

};

inline double

dist(struct kd\_node\_t\* a, struct kd\_node\_t\* b, int dim)

{

double t, d = 0;

while (dim--) {

t = a->x[dim] - b->x[dim];

d += t \* t;

}

return d;

}

inline void swap(struct kd\_node\_t\* x, struct kd\_node\_t\* y) {

double tmp[MAX\_DIM];

memcpy(tmp, x->x, sizeof(tmp));

memcpy(x->x, y->x, sizeof(tmp));

memcpy(y->x, tmp, sizeof(tmp));

}

/\* see quickselect method \*/

struct kd\_node\_t\*

find\_median(struct kd\_node\_t\* start, struct kd\_node\_t\* end, int idx)

{

if (end <= start) return NULL;

if (end == start + 1)

return start;

struct kd\_node\_t\* p, \* store, \* md = start + (end - start) / 2;

double pivot;

while (1) {

pivot = md->x[idx];

swap(md, end - 1);

for (store = p = start; p < end; p++) {

if (p->x[idx] < pivot) {

if (p != store)

swap(p, store);

store++;

}

}

swap(store, end - 1);

/\* median has duplicate values \*/

if (store->x[idx] == md->x[idx])

return md;

if (store > md) end = store;

else start = store;

}

}

struct kd\_node\_t\*

make\_tree(struct kd\_node\_t\* t, int len, int i, int dim)

{

struct kd\_node\_t\* n;

if (!len) return 0;

if ((n = find\_median(t, t + len, i))) {

i = (i + 1) % dim;

n->left = make\_tree(t, n - t, i, dim);

n->right = make\_tree(n + 1, t + len - (n + 1), i, dim);

}

return n;

}

/\* global variable, so sue me \*/

int visited;

void nearest(struct kd\_node\_t\* root, struct kd\_node\_t\* nd, int i, int dim,

struct kd\_node\_t\*\* best, double\* best\_dist)

{

double d, dx, dx2;

if (!root) return;

d = dist(root, nd, dim);

dx = root->x[i] - nd->x[i];

dx2 = dx \* dx;

visited++;

if (!\*best || d < \*best\_dist) {

\*best\_dist = d;

\*best = root;

}

/\* if chance of exact match is high \*/

if (!\*best\_dist) return;

if (++i >= dim) i = 0;

nearest(dx > 0 ? root->left : root->right, nd, i, dim, best, best\_dist);

if (dx2 >= \*best\_dist) return;

nearest(dx > 0 ? root->right : root->left, nd, i, dim, best, best\_dist);

}

void range\_search(struct kd\_node\_t\* root, struct range\* r) {

for (int i = 0;i < 6;i++) {

for (int j = 0;j < 6;j++) {

if (root[j].x[0] >= r[i].x && root[j].x[0] <= (r[i].x + r[i].w) &&

root[j].x[1] >= r[i].y && root[j].x[1] <= (r[i].y + r[i].h)) {

printf("range :(%d, %d, %d, %d), point :(%d, %d)\n", (int)r[i].x, (int)r[i].y, (int)r[i].w, (int)r[i].h, (int)root[j].x[0], (int)root[j].x[1]);

}

}

}

}

#define N 1000000

#define rand1() (rand() / (double)RAND\_MAX)

#define rand\_pt(v) { v.x[0] = rand1(); v.x[1] = rand1(); v.x[2] = rand1(); }

int main(void)

{

int i;

struct kd\_node\_t wp[] = {

{{2, 3}}, {{5, 4}}, {{9, 6}}, {{4, 7}}, {{8, 1}}, {{7, 2}}

};

struct range arr\_range[] = {

{1, 1, 5, 7}, {8, 1, 2, 7}, {3, 2, 4, 6}, {0, 0, 10, 10}, {1, 2, 2, 2}, {8, 7, 1, 2}

};

struct kd\_node\_t testNode = { {9, 2} };

struct kd\_node\_t\* root, \* found, \* million;

double best\_dist;

root = make\_tree(wp, sizeof(wp) / sizeof(wp[1]), 0, 2);

visited = 0;

found = 0;

nearest(root, &testNode, 0, 2, &found, &best\_dist);

printf(">> WP tree\nsearching for (%g, %g)\n"

"found (%g, %g) dist %g\nseen %d nodes\n\n",

testNode.x[0], testNode.x[1],

found->x[0], found->x[1], sqrt(best\_dist), visited);

range\_search(wp, arr\_range);

return 0;

}

Lab 4-2 Result

