### star最短路径搜索

A\*算法，A\*（A-Star)算法是一种静态路网中求解最短路径最有效的直接搜索方法，也是解决许多搜索问题的有效算法。算法中的距离估算值与实际值越接近，最终搜索速度越快。

公式表示为： f(n)=g(n)+h(n),

其中， f(n) 是从初始状态经由状态n到目标状态的代价估计，

g(n) 是在状态空间中从初始状态到状态n的实际代价，

h(n) 是从状态n到目标状态的最佳路径的估计代价。

（对于路径搜索问题，状态就是图中的节点，代价就是距离）

h(n)的选取

保证找到最短路径（最优解的）条件，关键在于估价函数f(n)的选取（或者说h(n)的选取）。

我们以d(n)表达状态n到目标状态的距离，那么h(n)的选取大致有如下三种情况：

* 如果h(n)< d(n)到目标状态的实际距离，这种情况下，搜索的点数多，搜索范围大，效率低。但能得到最优解。
* 如果h(n)=d(n)，即距离估计h(n)等于最短距离，那么搜索将严格沿着最短路径进行， 此时的搜索效率是最高的。
* 如果 h(n)>d(n)，搜索的点数少，搜索范围小，效率高，但不能保证得到最优解。

**距离估计与实际值越接近，估价函数取得就越好**

例如对于几何路网来说，可以取两节点间曼哈顿距离做为距离估计，即f=g(n) + (abs(dx - nx) + abs(dy - ny))；这样估价函数f(n)在g(n)一定的情况下，会或多或少的受距离估计值h(n)的制约，节点距目标点近，h值小，f值相对就小，能保证最短路的搜索向终点的方向进行。明显优于Dijkstra算法的毫无方向的向四周搜索。

**算法实现（路径搜索）**

用C语言实现A\*最短路径搜索算法

#include <stdio.h>

#include <math.h>

#define MaxLength 100 //用于优先队列（Open表）的数组

#define Height 15 //地图高度

#define Width 20 //地图宽度

#define Reachable 0 //可以到达的结点

#define Bar 1 //障碍物

#define Pass 2 //需要走的步数

#define Source 3 //起点

#define Destination 4 //终点

#define Sequential 0 //顺序遍历

#define NoSolution 2 //无解决方案

#define Infinity 0xfffffff

#define East (1 << 0)

#define South\_East (1 << 1)

#define South (1 << 2)

#define South\_West (1 << 3)

#define West (1 << 4)

#define North\_West (1 << 5)

#define North (1 << 6)

#define North\_East (1 << 7)

typedef struct

{

signed char x, y;

} Point;

const Point dir[8] =

{

{0, 1}, // East

{1, 1}, // South\_East

{1, 0}, // South

{1, -1}, // South\_West

{0, -1}, // West

{-1, -1}, // North\_West

{-1, 0}, // North

{-1, 1} // North\_East

};

unsigned char within(int x, int y)

{

return (x >= 0 && y >= 0

&& x < Height && y < Width);

}

typedef struct

{

int x, y;

unsigned char reachable, sur, value;

} MapNode;

typedef struct Close

{

MapNode \*cur;

char vis;

struct Close \*from;

float F, G;

int H;

} Close;

typedef struct //优先队列（Open表）

{

int length; //当前队列的长度

Close\* Array[MaxLength]; //评价结点的指针

} Open;

static MapNode graph[Height][Width];

static int srcX, srcY, dstX, dstY; //起始点、终点

static Close close[Height][Width];

// 优先队列基本操作

void initOpen(Open \*q) //优先队列初始化

{

q->length = 0; // 队内元素数初始为0

}

void push(Open \*q, Close cls[Height][Width], int x, int y, float g)

{ //向优先队列（Open表）中添加元素

Close \*t;

int i, mintag;

cls[x][y].G = g; //所添加节点的坐标

cls[x][y].F = cls[x][y].G + cls[x][y].H;

q->Array[q->length++] = &(cls[x][y]);

mintag = q->length - 1;

for (i = 0; i < q->length - 1; i++)

{

if (q->Array[i]->F < q->Array[mintag]->F)

{

mintag = i;

}

}

t = q->Array[q->length - 1];

q->Array[q->length - 1] = q->Array[mintag];

q->Array[mintag] = t; //将评价函数值最小节点置于队头

}

Close\* shift(Open \*q)

{

return q->Array[--q->length];

}

// 地图初始化操作

void initClose(Close cls[Height][Width], int sx, int sy, int dx, int dy)

{ // 地图Close表初始化配置

int i, j;

for (i = 0; i < Height; i++)

{

for (j = 0; j < Width; j++)

{

cls[i][j].cur = &graph[i][j]; // Close表所指节点

cls[i][j].vis = !graph[i][j].reachable; // 是否被访问

cls[i][j].from = NULL; // 所来节点

cls[i][j].G = cls[i][j].F = 0;

cls[i][j].H = abs(dx - i) + abs(dy - j); // 评价函数值

}

}

cls[sx][sy].F = cls[sx][sy].H; //起始点评价初始值

// cls[sy][sy].G = 0; //移步花费代价值

cls[dx][dy].G = Infinity;

}

void initGraph(const int map[Height][Width], int sx, int sy, int dx, int dy)

{ //地图发生变化时重新构造地

int i, j;

srcX = sx; //起点X坐标

srcY = sy; //起点Y坐标

dstX = dx; //终点X坐标

dstY = dy; //终点Y坐标

for (i = 0; i < Height; i++)

{

for (j = 0; j < Width; j++)

{

graph[i][j].x = i; //地图坐标X

graph[i][j].y = j; //地图坐标Y

graph[i][j].value = map[i][j];

graph[i][j].reachable = (graph[i][j].value == Reachable); // 节点可到达性

graph[i][j].sur = 0; //邻接节点个数

if (!graph[i][j].reachable)

{

continue;

}

if (j > 0)

{

if (graph[i][j - 1].reachable) // left节点可以到达

{

graph[i][j].sur |= West;

graph[i][j - 1].sur |= East;

}

if (i > 0)

{

if (graph[i - 1][j - 1].reachable

&& graph[i - 1][j].reachable

&& graph[i][j - 1].reachable) // up-left节点可以到达

{

graph[i][j].sur |= North\_West;

graph[i - 1][j - 1].sur |= South\_East;

}

}

}

if (i > 0)

{

if (graph[i - 1][j].reachable) // up节点可以到达

{

graph[i][j].sur |= North;

graph[i - 1][j].sur |= South;

}

if (j < Width - 1)

{

if (graph[i - 1][j + 1].reachable

&& graph[i - 1][j].reachable

&& map[i][j + 1] == Reachable) // up-right节点可以到达

{

graph[i][j].sur |= North\_East;

graph[i - 1][j + 1].sur |= South\_West;

}

}

}

}

}

}

int bfs()

{

int times = 0;

int i, curX, curY, surX, surY;

unsigned char f = 0, r = 1;

Close \*p;

Close\* q[MaxLength] = { &close[srcX][srcY] };

initClose(close, srcX, srcY, dstX, dstY);

close[srcX][srcY].vis = 1;

while (r != f)

{

p = q[f];

f = (f + 1) % MaxLength;

curX = p->cur->x;

curY = p->cur->y;

for (i = 0; i < 8; i++)

{

if (! (p->cur->sur & (1 << i)))

{

continue;

}

surX = curX + dir[i].x;

surY = curY + dir[i].y;

if (! close[surX][surY].vis)

{

close[surX][surY].from = p;

close[surX][surY].vis = 1;

close[surX][surY].G = p->G + 1;

q[r] = &close[surX][surY];

r = (r + 1) % MaxLength;

}

}

times++;

}

return times;

}

int astar()

{ // A\*算法遍历

//int times = 0;

int i, curX, curY, surX, surY;

float surG;

Open q; //Open表

Close \*p;

initOpen(&q);

initClose(close, srcX, srcY, dstX, dstY);

close[srcX][srcY].vis = 1;

push(&q, close, srcX, srcY, 0);

while (q.length)

{ //times++;

p = shift(&q);

curX = p->cur->x;

curY = p->cur->y;

if (!p->H)

{

return Sequential;

}

for (i = 0; i < 8; i++)

{

if (! (p->cur->sur & (1 << i)))

{

continue;

}

surX = curX + dir[i].x;

surY = curY + dir[i].y;

if (!close[surX][surY].vis)

{

close[surX][surY].vis = 1;

close[surX][surY].from = p;

surG = p->G + sqrt((curX - surX) \* (curX - surX) + (curY - surY) \* (curY - surY));

push(&q, close, surX, surY, surG);

}

}

}

//printf("times: %d\n", times);

return NoSolution; //无结果

}

const int map[Height][Width] = {

{0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,1},

{0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1},

{0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,1},

{0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,1},

{0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0},

{0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,0,1,0,1,0,0,0,1},

{0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0}

};

const char Symbol[5][3] = { "□", "▓", "▽", "☆", "◎" };

void printMap()

{

int i, j;

for (i = 0; i < Height; i++)

{

for (j = 0; j < Width; j++)

{

printf("%s", Symbol[graph[i][j].value]);

}

puts("");

}

puts("");

}

Close\* getShortest()

{ // 获取最短路径

int result = astar();

Close \*p, \*t, \*q = NULL;

switch(result)

{

case Sequential: //顺序最近

p = &(close[dstX][dstY]);

while (p) //转置路径

{

t = p->from;

p->from = q;

q = p;

p = t;

}

close[srcX][srcY].from = q->from;

return &(close[srcX][srcY]);

case NoSolution:

return NULL;

}

return NULL;

}

static Close \*start;

static int shortestep;

int printShortest()

{

Close \*p;

int step = 0;

p = getShortest();

start = p;

if (!p)

{

return 0;

}

else

{

while (p->from)

{

graph[p->cur->x][p->cur->y].value = Pass;

printf("（%d，%d）→\n", p->cur->x, p->cur->y);

p = p->from;

step++;

}

printf("（%d，%d）\n", p->cur->x, p->cur->y);

graph[srcX][srcY].value = Source;

graph[dstX][dstY].value = Destination;

return step;

}

}

void clearMap()

{ // Clear Map Marks of Steps

Close \*p = start;

while (p)

{

graph[p->cur->x][p->cur->y].value = Reachable;

p = p->from;

}

graph[srcX][srcY].value = map[srcX][srcY];

graph[dstX][dstY].value = map[dstX][dstY];

}

void printDepth()

{

int i, j;

for (i = 0; i < Height; i++)

{

for (j = 0; j < Width; j++)

{

if (map[i][j])

{

printf("%s ", Symbol[graph[i][j].value]);

}

else

{

printf("%2.0lf ", close[i][j].G);

}

}

puts("");

}

puts("");

}

void printSur()

{

int i, j;

for (i = 0; i < Height; i++)

{

for (j = 0; j < Width; j++)

{

printf("%02x ", graph[i][j].sur);

}

puts("");

}

puts("");

}

void printH()

{

int i, j;

for (i = 0; i < Height; i++)

{

for (j = 0; j < Width; j++)

{

printf("%02d ", close[i][j].H);

}

puts("");

}

puts("");

}

int main(int argc, const char \*\*argv)

{

initGraph(map, 0, 0, 0, 0);

printMap();

while (scanf("%d %d %d %d", &srcX, &srcY, &dstX, &dstY) != EOF)

{

if (within(srcX, srcY) && within(dstX, dstY))

{

if (shortestep = printShortest())

{

printf("从（%d，%d）到（%d，%d）的最短步数是: %d\n",

srcX, srcY, dstX, dstY, shortestep);

printMap();

clearMap();

bfs();

//printDepth();

puts((shortestep == close[dstX][dstY].G) ? "正确" : "错误");

clearMap();

}

else

{

printf("从（%d，%d）不可到达（%d，%d）\n",

srcX, srcY, dstX, dstY);

}

}

else

{

puts("输入错误！");

}

}

return (0);

}

**输出示例：**



