搜索 0921 补

目录

[前重复后精确覆盖 以及 精确覆盖 1](#_Toc304385268)

[IDA\* 版重复覆盖 5](#_Toc304385269)

[A\* 8](#_Toc304385270)

[IDA\* 12](#_Toc304385271)

前重复后精确覆盖 以及 精确覆盖

**const** **int** WID = 110;

**const** **int** HGT = 150;

**const** **int** SIZE = WID \* (HGT + 1) + 10;

**int** arr[WID][HGT], cnt[WID];

**int** best;

**int** N;

**const** **int** INF = 0x7FFFFFFF;

**struct** Dancer

{

**int** L[SIZE], R[SIZE], U[SIZE], D[SIZE], C[SIZE], Row[SIZE];

**int** S[WID + 10];

**int** width, height;

**void** **init**(**int** width, **int** height) //width列height行

{

best = INF;

**this**->width = width;

**this**->height = height;

**int** p, x, y, last, t;

**for** (x = 1; x <= width; x++)

{

L[x] = x - 1;

R[x] = x + 1;

U[x] = D[x] = x;

S[x] = 0;

}

R[width] = 0;

p = width + 1;

**for** (y = 1; y <= height; y++)

{

last = R[0] = L[0] = 0;

**for** (t = 1; t <= cnt[y]; t++)

{

**int** x = arr[y][t];

U[p] = U[x];

C[p] = D[p] = x;

L[p] = last;

S[x]++;

Row[p] = y;

last = R[last] = U[x] = D[U[x]] = p++;

}

R[last] = R[0];

L[R[0]] = last;

}

L[0] = width;

R[0] = 1;

S[0] = INF;

}

**void** **remove**(**const** **int** &c)

{

**int** i;

**for** (i = D[c]; i != c; i = D[i])

{

L[R[i]] = L[i];

R[L[i]] = R[i];

}

}

**void** **resume**(**const** **int** &c)

{

**int** i;

**for** (i = U[c]; i != c; i = U[i])

{

L[R[i]] = i;

R[L[i]] = i;

}

}

**void** **removeExact**(**int** c)

{

L[R[c]] = L[c];

R[L[c]] = R[c];

**for** (**int** i = D[c]; i != c; i = D[i])

{

**for** (**int** j = R[i]; j != i; j = R[j])

{

U[D[j]] = U[j];

D[U[j]] = D[j];

--S[C[j]];

}

}

}

**void** **resumeExact**(**int** c)

{

**for** (**int** i = U[c]; i != c; i = U[i])

{

**for** (**int** j = L[i]; j != i; j = L[j])

{

++S[C[j]];

U[D[j]] = j;

D[U[j]] = j;

}

}

L[R[c]] = c;

R[L[c]] = c;

}

**int** **h**(**int** bound)

{

**bool** hash[110];

**memset**(hash, **false**, **sizeof**(hash));

**int** ret = 0;

**for** (**int** c = R[0]; c != 0; c = R[c])

{

**if** (!hash[c] && c <= bound)

{

ret++;

hash[c] = **true**;

**for** (**int** i = D[c]; i != c; i = D[i])

{

**for** (**int** j = R[i]; j != i; j = R[j])

{

hash[C[j]] = **true**;

}

}

}

}

**return** ret;

}

**bool** **danceExact**()

{

**if** (R[0] == 0)

**return** **true**;

**int** c = 0, i, j;

**for** (i = R[0]; i; i = R[i])

**if** (S[i] < S[c])

c = i;

removeExact(c);

**for** (i = D[c]; i != c; i = D[i])

{

**for** (j = R[i]; j != i; j = R[j])

removeExact(C[j]);

**if** (danceExact())

**return** **true**;

**for** (j = L[i]; j != i; j = L[j])

resumeExact(C[j]);

}

resumeExact(c);

**return** **false**;

}

**bool** **dance**(**int** deep, **int** bound)

{

**if** (deep + h(bound) >= best)

**return** **false**;

**if** (R[0] == 0 || R[0] > bound)

{

best = min(best, deep);

**return** **true**;

}

**int** c = 0, i, j;

**bool** flag = **false**;

**for** (i = R[0]; i != 0; i = R[i])

**if** (S[i] < S[c] && i <= bound)

c = i;

**for** (i = D[c]; i != c; i = D[i])

{

remove(i);

**int** idx = -1;

**for** (j = R[i]; j != i; j = R[j])

{

**if** (C[j] <= bound)

remove(j);

**else**

idx = j;

}

**if** (idx != -1)

removeExact(C[idx]);

**if** (dance(deep + 1, bound))

flag = **true**;

**if** (idx != -1)

resumeExact(C[idx]);

**for** (j = L[i]; j != i; j = L[j])

{

**if** (C[j] <= bound)

resume(j);

}

resume(i);

}

**return** flag;

}

};

IDA\* 版重复覆盖

**const** **int** WID = 230;

**const** **int** HGT = 230;

**const** **int** SIZE = WID \* (HGT + 1) + 10;

**int** arr[WID][HGT], cnt[WID];

**int** best;

**const** **int** INF = 0x7FFFFFFF;

**struct** Dancer

{

**int** L[SIZE], R[SIZE], U[SIZE], D[SIZE], C[SIZE], Row[SIZE];

**int** S[WID + 10];

**int** width, height;

**void** init(**int** width, **int** height)//width列height行

{

best = INF;

**this**->width = width;

**this**->height = height;

**int** p, x, y, last, t;

**for** (x = 1; x <= width; x++)

{

L[x] = x - 1;

R[x] = x + 1;

U[x] = D[x] = x;

S[x] = 0;

}

R[width] = 0;

p = width + 1;

**for** (y = 1; y <= height; y++)

{

last = R[0] = L[0] = 0;

**for** (t = 1; t <= cnt[y]; t++)

{

**int** x = arr[y][t];

U[p] = U[x];

C[p] = D[p] = x;

L[p] = last;

S[x]++;

Row[p] = y;

last = R[last] = U[x] = D[U[x]] = p++;

}

R[last] = R[0];

L[R[0]] = last;

}

L[0] = width;

R[0] = 1;

S[0] = INF;

}

**void** remove(**const** **int** &c)

{

**int** i;

**for** (i = D[c]; i != c; i = D[i])

{

L[R[i]] = L[i];

R[L[i]] = R[i];

}

}

**void** resume(**const** **int** &c)

{

**int** i;

**for** (i = U[c]; i != c; i = U[i])

{

L[R[i]] = i;

R[L[i]] = i;

}

}

**bool** hs[260];

**int** h()

{

memset(hs, **false**, **sizeof**(hs));

**int** ret = 0;

**for** (**int** c = R[0]; c != 0; c = R[c])

{

**if** (!hs[c])

{

ret++;

hs[c] = **true**;

**for** (**int** i = D[c]; i != c; i = D[i])

{

**for** (**int** j = R[i]; j != i; j = R[j])

{

hs[C[j]] = **true**;

}

}

}

}

**return** ret;

}

**int** flg, lim;

**int** dance(**int** deep)

{

**int** tmp = h();

**if** (deep + tmp > lim)

**return** deep + tmp;

**if** (R[0] == 0)

{

flg = **true**;

**return** deep;

}

**int** c = 0, i, j;

**for** (i = R[0]; i != 0; i = R[i])

**if** (S[i] < S[c])

c = i;

**int** nxt = INF;

**for** (i = D[c]; i != c; i = D[i])

{

remove(i);

**for** (j = R[i]; j != i; j = R[j])

{

remove(j);

}

tmp = dance(deep + 1);

**if** (flg)

**return** tmp;

nxt = min(nxt, tmp);

**for** (j = L[i]; j != i; j = L[j])

{

resume(j);

}

resume(i);

}

**return** nxt;

}

**int** id\_astar()

{

lim = h();

flg = **false**;

**while** (!flg)

{

lim = dance(0);

}

**return** lim;

}

};

A\*

**const** **int** maxn = 60;

**int** limit, solved;

**int** M, N;

**int** mat[maxn][maxn];

**int** sx, sy;

**int** destx, desty;

**int** sd;

**int** dx[] =

{ -1, 0, 1, 0 };

**int** dy[] =

{ 0, 1, 0, -1 };

**void** init()

{

**for** (**int** i = 1; i <= M; i++)

{

**for** (**int** j = 1; j <= N; j++)

{

scanf("%d", mat[i] + j);

}

}

scanf("%d%d%d%d", &sx, &sy, &destx, &desty);

**char** tmp[10];

scanf("%s", tmp);

**if** (tmp[0] == 'n')

sd = 0;

**else** **if** (tmp[0] == 'e')

sd = 1;

**else** **if** (tmp[0] == 's')

sd = 2;

**else**

sd = 3;

}

**inline** **int** isok(**int** x, **int** y)

{

**return** x >= 1 && x <= M - 1 && y >= 1 && y <= N - 1 && !mat[x][y]

&& !mat[x][y + 1] && !mat[x + 1][y] && !mat[x + 1][y + 1];

}

**int** isok(**int** fx, **int** fy, **int** dir, **int** step)

{

**for** (**int** i = 0, x = fx, y = fy; i <= step; x += dx[dir], y += dy[dir], i++)

{

**if** (!isok(x, y))

**return** 0;

}

**return** 1;

}

**int** dec[] =

{ 3, 0, 1, 2 };

**int** inc[] =

{ 1, 2, 3, 0 };

**int** h(**int** x, **int** y)

{

**return** (abs(destx - x) + abs(desty - y) + 2) / 3;

}

**struct** Node

{

**int** x, y, dir;

**int** g, h, f;

Node()

{

}

**bool** **operator**<(**const** Node& p) **const**

{

**return** f > p.f;

}

**bool** **operator**==(**const** Node& p) **const**

{

**return** x == p.x && y == p.y && dir == p.dir;

}

};

priority\_queue<Node> Q;

**int** f[maxn][maxn][4];

**void** handle(**const** Node& from, **int** x, **int** y, **int** dir)

{

**int** tmp = h(x, y);

**if** (f[x][y][dir] == -1 || f[x][y][dir] > tmp + from.g)

{

Node next;

next.x = x;

next.y = y;

next.dir = dir;

next.g = from.g + 1;

next.h = tmp;

next.f = next.g + next.h;

f[x][y][dir] = next.f;

Q.push(next);

}

}

**int** AStar()

{

**while** (!Q.empty())

Q.pop();

memset(f, -1, **sizeof**(f));

Node begin;

begin.x = sx;

begin.y = sy;

begin.dir = sd;

begin.h = h(sx, sy);

begin.g = 0;

begin.f = begin.h;

f[sx][sy][sd] = begin.f;

Q.push(begin);

solved = 0;

**while** (!solved && !Q.empty())

{

Node current = Q.top();

Q.pop();

**if** (current.h == 0)

**return** current.g;

handle(current, current.x, current.y, dec[current.dir]);

handle(current, current.x, current.y, inc[current.dir]);

**for** (**int** i = 1; i <= 3; i++)

{

**int** nx = current.x + i \* dx[current.dir];

**int** ny = current.y + i \* dy[current.dir];

**if** (isok(nx, ny))

{

handle(current, nx, ny, current.dir);

}

**else**

**break**;

}

}

}

**int** vis[maxn][maxn];

**void** hasSolution(**int** x, **int** y)

{

**for** (**int** i = 0; i < 4; i++)

{

**int** nx = x + dx[i];

**int** ny = y + dy[i];

**if** (isok(nx, ny) && !vis[nx][ny])

{

vis[nx][ny] = 1;

hasSolution(nx, ny);

}

}

}

**int** hasSolution()

{

memset(vis, 0, **sizeof**(vis));

**if** (!isok(sx, sy))

**return** 0;

vis[sx][sy] = 1;

hasSolution(sx, sy);

**if** (!vis[destx][desty])

**return** 0;

**return** 1;

}

**void** work()

{

**if** (hasSolution())

{

printf("%d\n", AStar());

}

**else**

puts("-1");

}

IDA\*

//启发函数h带系数

**int** limit;

**int** solved;

**const** **int** maxn = 20;

**int** arr[maxn];

**void** init()

{

scanf("%d", &N);

**for** (**int** i = 1; i <= N; i++)

scanf("%d", arr + i);

}

**int** h(**int** arr[])

{

**int** result = 0;

**for** (**int** i = 0; i <= N - 1; i++)

{

**if** (arr[i + 1] != arr[i] + 1)

result++;

}

**return** (result + 2) / 3;

}

**int** DFS(**int** deep, **int** arr[])

{

**int** hvalue = h(arr);

**if** (hvalue + deep > limit)

{

**return** hvalue + deep;

}

**if** (hvalue == 0)

{

solved = 1;

**return** deep;

}

**int** next = 0x7FFFFFFF;

**int** tmp[maxn];

tmp[0] = 0;

**for** (**int** i = 1; i <= N; i++)

{

**for** (**int** j = i; j <= N; j++)

{

**for** (**int** k = 0; k <= N; k++)

{

**if** (k >= i - 1 && k <= j)

**continue**;

**int** cnt = 0;

**for** (**int** l = 1; l <= k; l++)

{

**if** (l >= i && l <= j)

**continue**;

tmp[++cnt] = arr[l];

}

**for** (**int** l = i; l <= j; l++)

tmp[++cnt] = arr[l];

**for** (**int** l = k + 1; l <= N; l++)

{

**if** (l >= i && l <= j)

**continue**;

tmp[++cnt] = arr[l];

}

**int** v = DFS(deep + 1, tmp);

**if** (solved)

**return** v;

next = min(next, v);

}

}

}

**return** next;

}

**int** IDAstar()

{

solved = 0;

limit = h(arr);

**while** (!solved && limit <= 4)

{

limit = DFS(0, arr);

}

**if** (solved)

**return** limit;

**return** -1;

}