**A\*算法**

米日阿地·买买提明 171491103

**一 算法介绍**

A\*算法常用于二维地图路径规划,算法所采用的启发式搜索可以利用实际问题所具备的启发式信息来指导搜索,从而减少搜索范围,控制搜索规模,降低实际问题的复杂度 。

## 二 算法原理

A\*算法的原理是设计一个代价估计函数:

其中评估函数 F(n)是从起始节点通过节点 n 的到达目标节点的最小代价路径的估计值,函数 G(n)是从起始节点到 n 节点的已走过路径的实际代价,函数H(n)是从 n 节点到目标节点可能的最优路径的估计代价 。函数 H(n)表明了算法使用的启发信息,它来源于人们对路径规划问题的认识,依赖某种经验估计。根据 F(n)可以计算出当前节点的代价,并可以对下一次能够到达的节点进行评估。采用每次搜索都找到代价值最小的点再继续往外搜索的过程,一步一步找到最优路径。

## 三 算法流程

(1)首先创建开始节点为 START,目标节点为 GOAL,创建开启列表OPEN 列表,关闭列表 CLOSED 列表,创建关闭列表时初始化为空。

(2)将开始节点 START 加入 OPEN 中。

(3)查询 OPEN 中的节点。如果 OPEN 为空,则退出并说明没有找到路径。

(4)如果 OPEN 不为空,从 OPEN 中选择 F(n)函数值最小的节点 n。

(5)把节点 n 从 OPEN 中去除,并将其添加到 CLOSED 中。

(6)判断节点 n 是否是目标节点 GOAL,如果节点 n 是目标节点 GOAL,则退出,并说明找到最优路径。如果节点 n 不是目标节点 GOAL,则转到(7)

(7)扩展节点 n,生成 n 节点的子节点集。设 n 节点的子节点为 m,对所有字节点 m 计算 F(m)。之后根据节点 m 的分类情况往下运行

1)若节点 m 既不在 OPEN 中也不在 CLOSED 中,将其加入 OPEN中,并为节点 m 分配一个指向其父节点 n 的指针。之后算法运行找到目标节点后根据该指针逐次返回,构成最优路径。

2)若节点 m 在 OPEN 中,则比较刚计算的 F(m)值和之前已存在的F(m)旧值。若 F(m)新值比旧值小,表明算法找到一条更好的路径,将 F(m)新值作为节点 m 的代价值;若 F(m)新值比旧值大,将 F(m)旧值作为节点 m 的代价值。修改节点 m 的指针,将指针指向它的父节点 n。

3)若节点 m 在 CLOSED 中,则忽略该节点并转到(7)。

(8)转到(3)继续运行,直至算法获得最优路径或无解退出。其中,在算法运行中创建的列表 OPEN 用于保存要搜索的节点,这些节点与算法运行的当前节点相邻并且不在列表 CLOSED 中。列表 CLOSED 用来储存算法获得最佳路径点。

## 四 算法实现

Main.cpp

#include "AStarAlgorithm.h"

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

/\*\*

\* Although this problem is better solved with dynamic programming,

\* it illustrates perfectly how to inherit from that template in order to solve a problem.

\*

\* Let's assume a currency composed of coins whose values are 2, 7, 8 and 19.

\* The problem consists of finding a minimal set of these coins whose total value equals a given amount.

\*/

class CCoinDistribution {

public:

size\_t coins2;

size\_t coins7;

size\_t coins8;

size\_t coins19;

CCoinDistribution():coins2(0),coins7(0),coins8(0),coins19(0) {}

size\_t money() const {

return 2\*coins2+7\*coins7+8\*coins8+19\*coins19;

}

inline bool operator==(const CCoinDistribution &mon) const {

return (coins2==mon.coins2)&&(coins7==mon.coins7)&&(coins8==mon.coins8)&&(coins19==mon.coins19);

}

};

class CAStarExample:public CAStarAlgorithm<CCoinDistribution> {

private:

const size\_t N;

public:

CAStarExample(size\_t goal):N(goal) {}

virtual bool isSolutionEnded(const CCoinDistribution &s) {

return s.money()==N;

}

virtual bool isSolutionValid(const CCoinDistribution &s) {

return s.money()<=N;

}

virtual void generateChildren(const CCoinDistribution &s,vector<CCoinDistribution> &sols) {

sols=vector<CCoinDistribution>(4,s);

sols[0].coins2++;

sols[1].coins7++;

sols[2].coins8++;

sols[3].coins19++;

}

virtual double getHeuristic(const CCoinDistribution &s) {

return static\_cast<double>(N-s.money())/19.0;

}

virtual double getCost(const CCoinDistribution &s) {

return s.coins2+s.coins7+s.coins8+s.coins19;

}

};

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

for (;;) {

char text[11];

printf("Input an integer number to solve a problem, or \"e\" to end.\n");

if (1!=scanf("%10s",text))

{

printf("Please, input a positive integer.\n\n");

continue;

}

if (strlen(text)==1&&(text[0]=='e'||text[0]=='E')) break;

int val=atoi(text);

if (val<=0) {

printf("Please, input a positive integer.\n\n");

continue;

}

CCoinDistribution solIni,solFin;

CAStarExample prob(static\_cast<size\_t>(val));

switch (prob.getOptimalSolution(solIni,solFin,HUGE\_VAL,15)) {

case 0:

printf("No solution has been found. Either the number is too small, or the time elapsed has exceeded 15 seconds.\n\n");

break;

case 1:

printf("An optimal solution has been found:\n");

printf("\t%u coins of 2 piastres.\n\t%u coins of 7 piastres.\n\t%u coins of 8 piastres.\n\t%u coins of 19 piastres.\n\n",(unsigned)solFin.coins2,(unsigned)solFin.coins7,(unsigned)solFin.coins8,(unsigned)solFin.coins19);

break;

case 2:

printf("A solution has been found, although it may not be optimal:\n");

printf("\t%u coins of 2 piastres.\n\t%u coins of 7 piastres.\n\t%u coins of 8 piastres.\n\t%u coins of 19 piastres.\n\n",(unsigned)solFin.coins2,(unsigned)solFin.coins7,(unsigned)solFin.coins8,(unsigned)solFin.coins19);

break;

}

}

return 0;

}

AStarAlgorithm.h

#ifndef CASTARALGORITHM\_H

#define CASTARALGORITHM\_H

#include <map>

#include <vector>

#include <cmath>

template<typename T> class CAStarAlgorithm {

public:

virtual bool isSolutionEnded(const T &sol)=0;

virtual bool isSolutionValid(const T &sol)=0;

virtual void generateChildren(const T &sol,std::vector<T> &sols)=0;

virtual double getHeuristic(const T &sol)=0;

virtual double getCost(const T &sol)=0;

private:

inline double getTotalCost(const T &sol) {

return getHeuristic(sol)+getCost(sol);

}

public:

int getOptimalSolution(const T &initialSol,T &finalSol,double upperLevel=HUGE\_VAL,double maxComputationTime=HUGE\_VAL) {

std::multimap<double,T> partialSols;

partialSols.insert(std::pair<double,T>(getTotalCost(initialSol),initialSol));

double currentOptimal=upperLevel;

bool found=false;

std::vector<T> children;

while (!partialSols.empty()) {

typename std::multimap<double,T>::iterator it=partialSols.begin();

double tempCost=it->first;

if (tempCost>=currentOptimal) return found?1:0;

T tempSol=it->second;

partialSols.erase(it);

if (isSolutionEnded(tempSol)) {

currentOptimal=tempCost;

finalSol=tempSol;

found=true;

continue;

}

generateChildren(tempSol,children);

for (typename std::vector<T>::const\_iterator it2=children.begin();it2!=children.end();++it2) if (isSolutionValid(\*it2)) {

bool alreadyPresent=false;

double cost=getTotalCost(\*it2);

typename std::pair<typename std::multimap<double,T>::const\_iterator,typename std::multimap<double,T>::const\_iterator> range = partialSols.equal\_range(cost);

for (typename std::multimap<double,T>::const\_iterator it3=range.first;it3!=range.second;++it3) if (it3->second==\*it2) {

alreadyPresent=true;

break;

}

if (!alreadyPresent) partialSols.insert(std::pair<double,T>(getTotalCost(\*it2),\*it2));

}

}

return found?1:0;

}

};

#endif