南京工程学院

**实 验 报 告**

课程名称 人工智能2020

实验项目名称 A\*八数码实验

实验学生班级 数嵌 172

实验学生姓名 朱广锋

学　　　　号 202170638

同组学生姓名 无

实验时间 2020/5/30

实验地点

1. 实验主题和目的

熟悉和掌握启发式搜索的定义、估价函数和算法过程。

利用A\*算法求解N数码难题，理解求解流程和搜索顺序。

1. 实验准备和条件

Visual Studio Community 2019。

1. 实验原理和步骤

A\*算法是一种启发式图搜索算法，其特点在于对估价函数的定义上。对于 一般的启发式图搜索，总是选择估价函数 f 值最小的节点作为扩展节点。因此， f 是根据需要找到一条最小代价路径的观点来估算节点的，所以，可考虑每个节 点 n 的估价函数值为两个分量：从起始节点到节点 n 的实际代价以及从节点 n 到达目标节点的估价代价。

1. 实验任务和内容

1、参考 A\*算法核心代码，以 8 数码问题为例实现 A\*算法的求解程序（编程语言不限），要求设计两种不同的估价函数。

2、在求解8数码问题的A\*算法程序中，设置相同的初始状态和目标状态，针对不同的估价函数，求得问题的解，并比较它们对搜索算法性能的影响，包括扩展节点数、生成节点数等。

3、对于8数码问题，设置与上述2相同的初始状态和目标状态，用宽度优先搜索算法（即令估计代价 h(n)＝0 的 A\*算法）求得问题的解，以及搜索过程中的扩展节点数、生成节点数。

4 上交源程序。

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #include<math.h>  struct Node {  int s[3][3];  int f, g;  Node\* next;  Node\* previous;  };  int open\_N = 0;  int inital\_s[3][3] = {  2,8,3,  1,6,4,  7,0,5  };  int final\_s[3][3] = {  1,2,3,  8,0,4,  7,6,5  };  void insertNode(Node\* head, Node\* p)  {  Node\* q;  if (head->next)  {  q = head->next;  if (p->f < head->next->f) {  p->next = head->next;  head->next = p;  }  else {  while (q->next)  {  if ((q->f < p->f || q->f == p->f) && (q->next->f > p->f || q->next->f == p->f)) {  p->next = q->next;  q->next = p;  break;  }  q = q->next;  }  if (q->next == NULL)  q->next = p;  }  }  else head->next = p;  }  void removeNode(Node\* head, Node\* p)  {  Node\* q;  q = head;  while (q->next)  {  if (q->next == p) {  q->next = p->next;  p->next = NULL;  if (q->next == NULL) return;  }  q = q->next;  }  }  int equal(int s1[3][3], int s2[3][3])  {  int i, j, flag = 0;  for (i = 0; i < 3; i++)  for (j = 0; j < 3; j++)  if (s1[i][j] != s2[i][j]) { flag = 1; break; }  if (!flag)  return 1;  else return 0;  }  int existNode(Node\* head, int s[3][3], Node\* Old\_Node)  {  Node\* q = head->next;  int flag = 0;  while (q)  if (equal(q->s, s)) {  flag = 1;  Old\_Node->next = q;  return 1;  }  else q = q->next;  if (!flag) return 0;  }  int wrongCount(int s[3][3])  {  int i, j, fi, fj, sum = 0;  for (i = 0; i < 3; i++)  for (j = 0; j < 3; j++)  {  for (fi = 0; fi < 3; fi++)  for (fj = 0; fj < 3; fj++)  if ((final\_s[fi][fj] == s[i][j])) {  sum += fabs(i - fi) + fabs(j - fj);  break;  }  }  return sum;  }  int getSuccessor(Node\* BESTNODE, int direction, Node\* Successor)  {  int i, j, i\_0, j\_0, temp;  for (i = 0; i < 3; i++)  for (j = 0; j < 3; j++)  Successor->s[i][j] = BESTNODE->s[i][j];  for (i = 0; i < 3; i++)  for (j = 0; j < 3; j++)  if (BESTNODE->s[i][j] == 0) { i\_0 = i; j\_0 = j; break; }  switch (direction)  {  case 0: if ((i\_0 - 1) > -1) {  temp = Successor->s[i\_0][j\_0];  Successor->s[i\_0][j\_0] = Successor->s[i\_0 - 1][j\_0];  Successor->s[i\_0 - 1][j\_0] = temp;  return 1;  }  else return 0;  case 1: if ((j\_0 - 1) > -1) {  temp = Successor->s[i\_0][j\_0];  Successor->s[i\_0][j\_0] = Successor->s[i\_0][j\_0 - 1];  Successor->s[i\_0][j\_0 - 1] = temp;  return 1;  }  else return 0;  case 2: if ((j\_0 + 1) < 3) {  temp = Successor->s[i\_0][j\_0];  Successor->s[i\_0][j\_0] = Successor->s[i\_0][j\_0 + 1];  Successor->s[i\_0][j\_0 + 1] = temp;  return 1;  }  else return 0;  case 3: if ((i\_0 + 1) < 3) {  temp = Successor->s[i\_0][j\_0];  Successor->s[i\_0][j\_0] = Successor->s[i\_0 + 1][j\_0];  Successor->s[i\_0 + 1][j\_0] = temp;  return 1;  }  else return 0;  }  }  Node\* getBestNode(Node\* Open)  {  return Open->next;  }  void printPath(Node\* head)  {  Node\* q, \* q1, \* p;  int i, j, count = 1;  p = (Node\*)malloc(sizeof(Node));  p->previous = NULL;  q = head;  while (q)  {  q1 = q->previous;  q->previous = p->previous;  p->previous = q;  q = q1;  }  q = p->previous;  while (q)  {  if (q == p->previous)printf("Init: \n");  else if (q->previous == NULL)printf("Target: \n");  else printf("Process %d\n", count++);  for (i = 0; i < 3; i++)  for (j = 0; j < 3; j++)  {  printf("%4d", q->s[i][j]);  if (j == 2)printf("\n");  }  printf("f=%d, g=%d\n\n", q->f, q->g);  q = q->previous;  }  }  void subAStarAlgorithm(Node\* Open, Node\* BESTNODE, Node\* Closed, Node\* Successor)  {  Node\* Old\_Node = (Node\*)malloc(sizeof(Node));  Successor->previous = BESTNODE;  Successor->g = BESTNODE->g + 1;  if (existNode(Open, Successor->s, Old\_Node)) {  if (Successor->g < Old\_Node->g) {  Old\_Node->next->previous = BESTNODE;  Old\_Node->next->g = Successor->g;  Old\_Node->next->f = Old\_Node->g + wrongCount(Old\_Node->s);  removeNode(Open, Old\_Node);  insertNode(Open, Old\_Node);  }  }  else if (existNode(Closed, Successor->s, Old\_Node)) {  if (Successor->g < Old\_Node->g) {  Old\_Node->next->previous = BESTNODE;  Old\_Node->next->g = Successor->g;  Old\_Node->next->f = Old\_Node->g + wrongCount(Old\_Node->s);  removeNode(Closed, Old\_Node);  insertNode(Closed, Old\_Node);  }  }  else {  Successor->f = Successor->g + wrongCount(Successor->s);  insertNode(Open, Successor);  open\_N++;  }  }  void AStarAlgorithm(Node\* Open, Node\* Closed)  {  int i, j;  Node\* BESTNODE, \* inital, \* Successor;  inital = (Node\*)malloc(sizeof(Node));  for (i = 0; i < 3; i++)  for (j = 0; j < 3; j++)  inital->s[i][j] = inital\_s[i][j];  inital->f = wrongCount(inital\_s);  inital->g = 0;  inital->previous = NULL;  inital->next = NULL;  insertNode(Open, inital);  open\_N++;  while (1)  {  if (open\_N == 0) { printf("Failure!"); return; }  else {  BESTNODE = getBestNode(Open);  removeNode(Open, BESTNODE);  open\_N--;  insertNode(Closed, BESTNODE);  if (equal(BESTNODE->s, final\_s)) {  printf("Success!\n");  printPath(BESTNODE);  return;  }  else {  Successor = (Node\*)malloc(sizeof(Node)); Successor->next = NULL;  if (getSuccessor(BESTNODE, 0, Successor))subAStarAlgorithm(Open, BESTNODE, Closed, Successor);  Successor = (Node\*)malloc(sizeof(Node)); Successor->next = NULL;  if (getSuccessor(BESTNODE, 1, Successor))subAStarAlgorithm(Open, BESTNODE, Closed, Successor);  Successor = (Node\*)malloc(sizeof(Node)); Successor->next = NULL;  if (getSuccessor(BESTNODE, 2, Successor))subAStarAlgorithm(Open, BESTNODE, Closed, Successor);  Successor = (Node\*)malloc(sizeof(Node)); Successor->next = NULL;  if (getSuccessor(BESTNODE, 3, Successor))subAStarAlgorithm(Open, BESTNODE, Closed, Successor);  }  }  }  }  void main()  {  Node\* Open = (Node\*)malloc(sizeof(Node));  Node\* Closed = (Node\*)malloc(sizeof(Node));  Open->next = NULL; Open->previous = NULL;  Closed->next = NULL; Closed->previous = NULL;  AStarAlgorithm(Open, Closed);  } |

1. 实验结果和总结

|  |  |
| --- | --- |
| 起始 0,1,2,7,8,9,6,5,4  目标 1,2,3,8,0,4,7,6,5 | 起始 2,8,3,1,6,4,7,0,5  目标 1,2,3,8,0,4,7,6,5 |
|  |  |

把S放入OPEN表，记f=h

OPEN=NULL?

是

失败

扩展BESTNODE,产生其后继结点SUCCESSOR

选取OPEN表上未设置过的具有最小f值的节点

BESTNODE,放入CLOSED表

BESTNODE是目标节点

建立从SUCCESSOR返回BESTNODE的指针

计算g(SUC)=g(BES)+k(BES,SUC)

SUC∈OPEN

开始

g(SUC)<g(OLD)

SUC=OLD,把它添加到BESTNDOE的后继结点表中

重新确定OLD的父辈节点为BESTNODE，并修正父辈节点的g值和f值，记下g(OLD)

是

成功

SUC∈CLOSED

把SUCCESSOR放入OPEN表，添进BESTNODE的后裔表

计算f值

是

否

是

否

是

否

否

否

把S放入OPEN表，记f=h

OPEN=NULL?

是

失败

扩展BESTNODE,产生其后继结点SUCCESSOR

选取OPEN表上未设置过的具有最小f值的节点

BESTNODE,放入CLOSED表

BESTNODE是目标节点

建立从SUCCESSOR返回BESTNODE的指针

计算g(SUC)=g(BES)+k(BES,SUC)

SUC∈OPEN

开始

根据A\*算法分析启发式搜索的特点：

启发式搜索是一种试探性的查询过程，为了减少搜索的盲目性引，增加试探的准确性，就要采用启发式搜索了。所谓启发式搜索就是在搜索中对每一个搜索的位置进行评估，从中选择最好、可能最容易达到目标的位置，再从这个位置向前进行搜索，这样就可以再搜索中省略大量无关的结点，提高了效率。

通过这次实验，我对启发式搜索算法有了更进一步的理解，特别是估计函数h(n)所起到的巨大重用。一个好的估计函数对于启发式搜索算法来说是十分关键的。教师评阅：

|  |  |
| --- | --- |
| 评阅项目及内容 | 得分 |
| 1．考勤（10分） |  |
| 2．实验完成情况（50分） |  |
| 3．报告撰写内容（40分） |  |
| 合 计 |  |
| 成绩评定 |  |