哈尔滨工程大学

实 验 报 告

实 验 名 称：­­

班 级：

学 号：

姓 名：

实 验 时 间：

成 绩：

指 导 教 师：

实验室名称：

**哈尔滨工程大学实验室与资产管理处 制**

|  |
| --- |
| 实验名称：修道士野人问题 |
| 一、问题描述  河的左岸有N 个野人和N 个修道士以及一条小船，修道士们想用这条小船把所有的人都运到河的右岸，但又受到以下限制：   ·修道士和野人都会划船，但船一次只能载C 人。   ·在任何岸边，为了防止野人侵犯修道士，野人数不能超过修道士数，否则修道士将会被野人吃掉。  假定野人愿意服从任何一种过河的安排，本设计的主要任务是规划出一种确保修道士安全的过河方案。  二、数据结构设计  typedef struct { //Boat  int Monk; //修道士的数量  int Savage; //野人的数量  bool state; //船的状态 1表示船在起始岸边 0表示在目的地  } Boat;  Boat Boatsafe[MAXN];  typedef struct Edge {  int adjvex; //该边所指向的顶点的位置  struct Edge\* nextedge; //next域 指向下一条边  } Edge;  typedef struct Vnode {  Boat data; //顶点上的数据信息  int number; //顶点编号  int pre; //前驱  Edge\* firstedge; //指向第一个依附该点的边  } Vnode;  三、算法设计    //安全状态：左、右岸和船上，传教士都在or都不在or传教士人数等于野人人数  bool isSafe(Boat B) { //检测当前修道士是否安全  if ((B.Monk >= B.Savage || B.Monk == 0) &&  ((N - B.Monk) >= (N - B.Savage) || N == B.Monk) && B.Monk >= 0 && B.Monk <= N && B.Savage >= 0 && B.Savage <= N)  return true;  else  return false;  }  //打印出过河状态，进行过河操作  if (!(T[i].Monk == 0 && T[i].Savage == 0 && T[i].state == 0))//该状态是否已经访问过  int vis(int state[3]){  for(vector<node>::iterator it = s.begin(); it != s.end(); it++)  if((\*it).q[0] == state[0] && (\*it).q[1] == state[1] && (\*it).q[2] == state[2])  return 1;  return 0;  }  四、界面设计    界面设计  五、运行测试与分析  1.人数=5,最大载人数=4时    2.人数=11,最大载人数=4时    六、实验收获与思考  修道士野人问题是一个很经典的算法问题，我最开始的想法是使用暴力搜索的方法，并进行逐一的比较，最终确定最短的移动方案，这种思路能确保找到的移动方法时正确的，但是当修道士和野人的数量足够多的时候，这种方法会非常慢，不适用。因此，我思考了离散数学中的二部图的思想，通过设置二元组和三元组表示状态量，进而设计函数使得遍历二部图的路径最短，可以达到本题的目的。 |