

**哈尔滨工业大学**

**<<算法设计与分析>>**

**实验报告之二**

**(2015 年度秋季学期)**

姓名：	成坚
学号：	15S003005
学院：	计算机学院
教师：	高宏

## 实验二 搜索算法

### 一、实验目的

- 1、掌握搜索算法的基本设计思想与方法，
- 2、掌握分支界限搜索策略的设计思想与方法，
- 3、熟练使用高级编程语言实现搜索算法，
- 4、利用实验测试给出的搜索算法的性能。

### 二、实验内容

1、哈密顿环问题：输入是一个无向连通图  $G=(V,E)$ ；如果  $G$  中存在哈密顿环则输出该环，否则输出“否”。

2、最小哈密顿环问题：输入是一个无向连通图  $G=(V,E)$ ，每个节点都没有到自身的边，每对节点间都有一条非负加权边；输出一个权值代价和最小的哈密顿环。注意：事实上输入图是一个完全图，因此哈密顿环是一定存在的。

### 三、实验过程及结果

#### 1 SimAllHamilton 深度搜索输出图中所有的哈密顿环

```
27 bool HamiltonProcess(int (*graph)[MAX_SIZE], int depth, int hami[])
28 {
29     for(hami[depth] = 1; hami[depth] <= N; hami[depth]++) // 下一个顶点可能是1-N中的任何一个顶点
30     {
31         if(IsPathOk(graph, depth, hami) != 0) // 判断当前是否有边可达
32         {
33             ////////////////////////////////////////
34             visited[hami[depth]] = true; // 访问顶点k
35             ////////////////////////////////////////
36
37             //if(k == N || HamiltonProcess(graph, k+1, hami))
38             // 递归的进行下一个顶点，其实本质上也是一层相当于k++
39             // 外层循环遍历所有可能的第k的顶点的可能位置
40             // 没发现一个可能的顶点k,存入hami[k]中，
41             // 然后接着递归查找下一个顶点k+1，
42             HamiltonProcess(graph, depth + 1, hami); // 递归的进行下一个顶点
43         }
44         #ifdef DEBUG
45         if(depth == N)
46         {
47             printf("Case %2d: ", counter++);
48             for(int i = 1; i <= N + 1; i++)
49             {
50                 if(i == N + 1)
51                 {
52                     printf("%3d\n", hami[i]);
53                 }
54                 else
55                 {
56                     printf("%3d -> ", hami[i]);
57                 }
58             }
59             printf("=> %d\n", hami[1]);
60         }
61         #endif
62     }
63 }
```

## 2 BFSHamilton.cpp 广度优先搜索输出哈密顿环

```

27 void GenMinHamilton( )
28 {
29     m_depth = 1;
30     m_start = 1;
31     m_currCost = 0;
32     memset(m_visited, 0, MAX_VERTEX);
33     // 起始元素入队列
34     this->m_queue.push(m_start);
35
36     while(this->m_queue.empty() != true)
37     {
38         int curr = this->m_queue.front(); // 取出对头元素
39
40         //cout <<"Pop " <<curr <<endl;
41         this->m_queue.pop();
42
43         m_visited[curr] = true;
44         #ifdef DEBUG
45             cout <<"visited " <<curr <<endl;
46         #endif
47
48         this->hami[m_depth] = curr;
49         if(m_depth > 1)
50         {
51             #ifdef DEBUG
52                 cout <<" depth" <<m_depth <<" , add line" <<hami[m_depth - 1] <<" ->" <<hami[m_depth] <<" , Cost = " <<m_graph[hami[m_depth - 1]]
53             #endif
54             m_currCost += m_graph[hami[m_depth - 1]][hami[m_depth]];
55         }
56     }
57
58     if(N == m_depth)
59     {
60         #ifdef DEBUG
61             cout <<"add back line " <<curr <<" ->" <<START <<" , Cost = " << m_graph[curr][START]<<endl;
62         #endif
63         m_currCost += m_graph[curr][START];
64     }
65 }

```

## 3 ClimbeHiilHamilton 爬山法实现哈密顿环

```

27 void GenMinHamilton( )
28 {
29     m_depth = 1;
30     m_start = 1;
31     m_currCost = 0;
32     memset(m_visited, 0, MAX_VERTEX);
33     // 起始元素入队列
34     this->m_queue.push(m_start);
35
36     while(this->m_queue.empty() != true)
37     {
38         int curr = this->m_queue.top(); // 取出对头元素
39
40         //cout <<"Pop " <<curr <<endl;
41         this->m_queue.pop();
42
43         m_visited[curr] = true;
44         #ifdef DEBUG
45             cout <<"visited " <<curr <<endl;
46         #endif
47
48         this->hami[m_depth] = curr;
49         if(m_depth > 1)
50         {
51             #ifdef DEBUG
52                 cout <<" depth" <<m_depth <<" , add line" <<hami[m_depth - 1] <<" ->" <<hami[m_depth] <<" , Cost = " <<m_graph[hami[m_depth - 1]]
53             #endif
54             m_currCost += m_graph[hami[m_depth - 1]][hami[m_depth]];
55         }
56     }
57
58     if(N == m_depth)
59     {
60         #ifdef DEBUG
61             cout <<"add back line " <<curr <<" ->" <<START <<" , Cost = " << m_graph[curr][START]<<endl;
62         #endif
63         m_currCost += m_graph[curr][START];
64     }
65 }

```

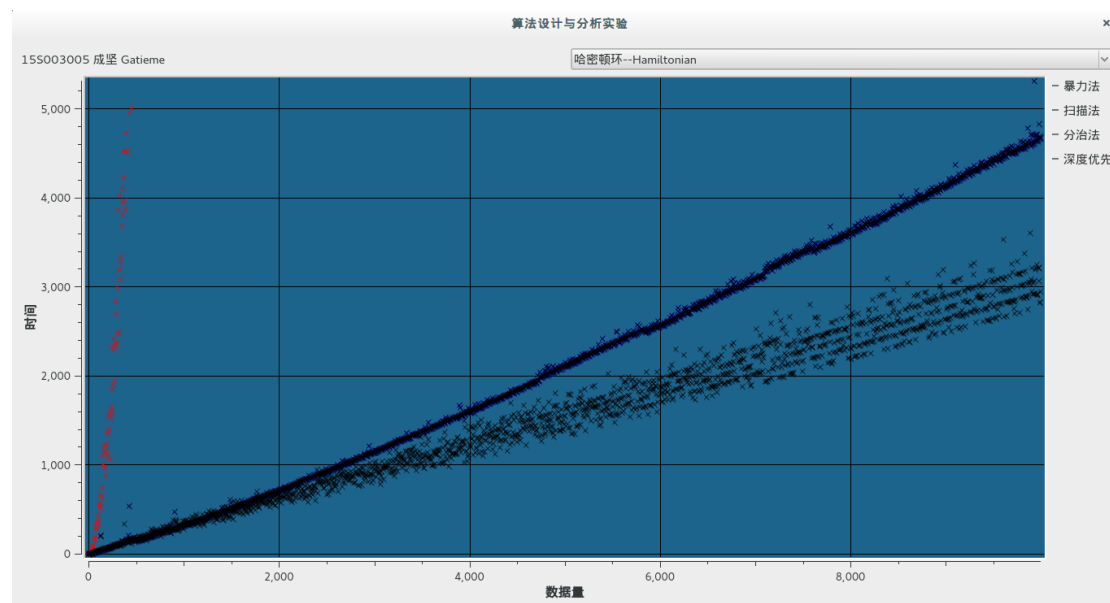
## 4 分支界限法输出最小哈密顿环

```

27 bool HamiltonProcess(int (*graph)[MAX_SIZE],int depth, int hami[])
28 {
29     for(hami[depth] = 1; hami[depth] <= N; hami[depth]++) // 下一个顶点可能是1-N中的任何一个顶点
30     {
31         // 此时hami[depth] == k进入栈hami[depth]中,
32         // 成为栈顶元素, 判断其是否可成为一条哈密顿的节点IsPathOk()
33         // 基于深度有限搜索策略, 如果期望使用爬山法, 可在此处进行一个权值判断
34         // 每次选择权值最小的那个进行扩展
35         if(IsPathOk(graph, depth, hami) != 0) // 判断当前是否有边可达, 即该节点hami[depth]可行
36         {
37             ////////////////////////////////////////////////////
38             visited[hami[depth]] = true; // 访问顶点K = hami[depth]
39             ////////////////////////////////////////////////////
40
41             // 哈密顿图的花费加上当前访问的边信息
42             //cost[counter] += graph[hami[depth - 1]][hami[depth]];
43             cost += graph[hami[depth - 1]][hami[depth]];
44
45             #ifdef DEBUG
46                 //printf("depth= %d, add line %d -> %d, cost = %d, total = %d\n", depth, hami[depth - 1], hami[depth], graph[hami[depth - 1]][hami[depth]], cost);
47                 printf("depth= %d, add line %d -> %d, cost = %d, total = %d\n", depth, hami[depth - 1], hami[depth], graph[hami[depth - 1]][hami[depth]], cost);
48             #endif
49
50             //if(k == N || HamiltonProcess(graph, k+1, hami))
51             // 递归的进行下一个顶点, 其实本质上也是一层相当于k++
52             // 外层循环遍历所有可能的第k的顶点的可能位置
53             // 没发现一个可能的顶点K, 存入hami[k]中,
54             // 然后接着递归查找下一个顶点K+1,
55             HamiltonProcess(graph, depth + 1, hami); // 递归的进行下一个顶点
56
57             #ifdef DEBUG
58                 if(depth == N)
59                 {
60                     // 哈密顿环是条回路, 花费加上回边的权值
61                     //cost[counter] += graph[hami[depth]][hami[depth - 1]];
62                     cost += graph[hami[depth]][hami[depth - 1]];
63                 }
64             #endif
65         }
66     }
67 }

```

## 5 绘制曲线



## 四、实验心得

- 1 对图的存储形式有了更深的体会
- 2 加强了对深度优先搜索和广度优先搜索的认识
- 3 体会了爬山法和分支界限深层次的精髓