Homework 4.1 Report

算法框架实现解决Romania和HIT问题

BFS

```
vector<string> BFS(string question, string start, string end)
{
    vector<string> solution;
    map<pair<string, string>, int> pathMap;
    pathMap = chooseMap(question);
    vector<string> frontier;
    frontier.push_back(start);
    vector<string> explored;
    while(1)
    {
      if(frontier.empty())
        return solution;
        string current = *(frontier.begin());
        frontier.erase(frontier.begin());
        explored.push_back(current);
        solution.push_back(current);
        if(current == end)
            return solution;
        for(map<pair<string, string>, int>::iterator iter = pathMap.begin(); iter !=
pathMap.end(); iter++)
          if(iter->first.first == current)
            if(find(frontier.begin(), frontier.end(), iter->first.second) == frontier.end() &&
find(explored.begin(), explored.end(), iter->first.second) == explored.end())
              frontier.push_back(iter->first.second);
    }
}
```

该算法具有如下的输入

- string question：解决的问题，本例中为"Romania"或"HIT"
- string start：搜索的起始点
- string end：搜索的结束点

该算法返回一个vector

- 向量内包含BFS搜索过程中每步寻找到的点

DFS

```
bool R_DLS(string node, map<pair<string, string>, int> pathMap, vector<string> &solution, int
limit, string end, vector<string> explored)
{
    solution.push_back(node);
    explored.push_back(node);
    if(node == end)
        return true;
    else if (limit == 0)
        return false;
    else
    {
        bool flag;
        for(map<pair<string, string>, int>::iterator iter = pathMap.begin(); iter !=
pathMap.end(); iter++)
            if(iter->first.first == node && find(explored.begin(), explored.end(), iter-
>first.second) == explored.end())
            {
                flag = R_DLS(iter->first.second, pathMap, solution, limit - 1, end, explored);
                if(flag == true)
                    return true;
            }
        if(flag == false)
            return false;
    }
    return false;
}

vector<string> DFS(string question, string start, string end, int limit)
{
    vector<string> solution;
    vector<string> explored;
    map<pair<string, string>, int> pathMap;
    pathMap = chooseMap(question);
    R_DLS(start, pathMap, solution, limit, end, explored);
    return solution;
}
```

该算法具有如下的输入

- string question：解决的问题，本例中为"Romania"或"HIT"
- string start：搜索的起始点
- string end：搜索的结束点
- int limit：DFS搜索限制层数

该算法返回一个vector

- 向量内包含DFS搜索过程得到的路径

IDS

```
bool R_DLS(string node, map<pair<string, string>, int> pathMap, vector<string> &solution, int
limit, string end, vector<string> explored)
{
    solution.push_back(node);
    explored.push_back(node);
    if(node == end)
        return true;
    else if (limit == 0)
        return false;
    else
    {
        bool flag;
        for(map<pair<string, string>, int>::iterator iter = pathMap.begin(); iter !=
pathMap.end(); iter++)
            if(iter->first.first == node && find(explored.begin(), explored.end(), iter-
>first.second) == explored.end())
            {
                flag = R_DLS(iter->first.second, pathMap, solution, limit - 1, end, explored);
                if(flag == true)
                    return true;
            }
        if(flag == false)
            return false;
    }
    return false;
}

vector<string> IDS(string question, string start, string end)
{
    map<pair<string, string>, int> pathMap;
    pathMap = chooseMap(question);
    for(int d = 1; d < MAXINT; d++)
    {
        vector<string> solution;
        solution.clear();
        vector<string> explored;
        explored.clear();
        bool success = false;
        success = R_DLS(start, pathMap, solution, d, end, explored);
        if(success == true)
            return solution;
    }
}
```

该算法具有如下的输入

- string question：解决的问题，本例中为"Romania"或"HIT"
- string start：搜索的起始点
- string end：搜索的结束点

该算法返回一个vector

- 向量内包含IDS搜索过程得到的路径

三种算法比较

	BFS	DFS	IDS
完备性	完备	图搜索在有限状态空间完备，无限空间不完备	完备
最优性	耗散随节点深度的非递减则有最优解	非最优	非最优
时间复杂性	O(b^d)	O(b^m)，m：最大深度	O(b^d)，d：深度
空间复杂性	O(b^(d-1))	bm+1	O(bd)，d：深度