实验报告

数据结构

- 将每个字母作为顶点，单词作为边，构造一个图，其中点的结构是维护一个它指向的边vector和指向它的边的指针vector，边的结构除了长度和string之外维护一个star标识，用于搜索时的是否已经搜索过的指示。

最长单词链搜索

- 在小规模数据下的遍历搜索

  对于小规模数据（50个单词以内），可以采用深度优先搜索进行遍历，对某个点的所有未搜索过的边循环递归调用遍历搜索函数，将star标志置为true，并且在其递归调用结束后将该边的star标志清除，因为下一条边的搜索可能还会搜索到该边，由于递归调用的归纳证明可以得到改算法可以遍历所有单词链的组合。

- 遍历算法

```
void DSearch(vector<ArcNode> &qlist, int depth, Vertex &vex, ArcNode &node, WordMost *wm)
{
    int flag = 0;
    for(int i=0; i<vex.adjArcs().size(); i++)
    {
        if(vex.adjArcs()[i].isStar() == false)
        {
            /*cout << vex.adjArcs()[i].getName() << endl;*/
            flag = 1;
            vex.adjArcs()[i].setStar();
            node.depth = depth;
            node.value = vex.adjArcs()[i].getName();
            qlist.push_back(node);
            DSearch(qlist, depth+1, vex.adjArcs()[i].adjVex() , node, wm);
            vex.adjArcs()[i].clearStar();
            qlist.pop_back();
            /*for(int i=0; i<qlist.size() ; i++)
                cout << qlist[i].value << " ";
            cout << endl;*/
        }
    }
    if(flag == 0)
    {
        /*for (int i = 0; i < qlist.size(); i++)
            cout << qlist[i].value << " ";
        cout << endl;*/
        if(depth-1 > wm->getMaxlength())
        {
            wm->setMaxLength(depth-1);
            wm->setQlist(qlist);
        }
```

```
    }
}
```

- 大规模下的爬山算法

  对于大规模数据，搜索到精确最长链是不可能的，这里采用的是重新随机的爬山算法，规定一个timelimit表示最长时间，一共进行timelimit时间的爬山搜索。

  主要思想是随机对一个节点进行随机的深度搜索，直到搜到节点的所有临边均被搜索过，然后对这个随机出来的状态进行变换，得到相邻状态，变换方式如下：

  如果当前的链长小于20，则随机取后半部分的一个边，将其之后（包括其）的边全部删掉，从删掉边的前一个节点重新随机深度搜索，重新得到一个链，如果此链的长度大于原长，则采用并继续替换，如果没原长长，则不接受，继续进行上述变换；

  如果当前的链长大于20，则对后面的10个边中随机找一个进行上述变换，这样的好处是，选取后面的边进行变换大概率可以使得整个链的长度得到增长。

  当上述爬山法进行了整个时间限制的1/50时，重新随机一个初始边进行爬山，也就是随机重启的爬山法，这样不会由于特殊的前几个节点选取情况导致结果与最长链出现很大的偏差。

- 爬山法中的随机深度搜索

```
void randDSearch(vector<Arc *> &arclist,vector<ArcNode> &qlist, Vertex &vex, ArcNode &node)
{
    vector<int> arcArray;
    for (int i = 0; i < vex.adjArcs().size(); i++)
    {
        if (vex.adjArcs()[i].isStar() == false)
        {
            arcArray.push_back(i);
        }
    }
    /*cout <<"vex " << vex.getName() << " " <<  " arcArray = " << arcArray.size() <<
endl;*/
    if (arcArray.size() == 0)
    {
        /*for (int i = 0; i < qlist.size() ; i++)
            cout << qlist[i].value << "-";
        cout << endl;*/
        return;
    }
    else
    {
        int rnum = rand() % arcArray.size();
        vex.adjArcs()[arcArray[rnum]].setStar();
        arclist.push_back(&vex.adjArcs()[arcArray[rnum]]);
        node.value = vex.adjArcs()[arcArray[rnum]].getName();
        qlist.push_back(node);
        randDSearch(arclist, qlist, vex.adjArcs()[arcArray[rnum]].adjVex(), node);
    }
}
```

- 爬山算法

```
void WordMost::wordMostLargeScaleSearch()
{
    srand(time(0));
    setMaxLength(-1);
    graph.clearAllStar();
    vector<Arc *> arclist,lastarclist;
    vector<ArcNode> qlist,lastqlist;
    ArcNode node;
    vector<Vertex> &v = graph.v;
    time_t starttime = clock(),lasttime = clock();
    int vexnum;
    int arcnum;
    int curlength;
    while (clock() - starttime < timelimit)
    {
        do
        {
            graph.clearAllStar();
            qlist.clear();
            arclist.clear();
            vexnum = rand() % 26;
            randDSearch(arclist, qlist, v[vexnum], node);
            curlength = qlist.size();
            if (curlength > getMaxlength())
            {
                setMaxLength(curlength);
                setQlist(qlist);
            }
            lastarclist.assign(arclist.begin(), arclist.end());
            lastqlist.assign(qlist.begin(), qlist.end());
        } while (curlength == 0);
        lasttime = clock();

        while (clock() - lasttime < timelimit/50)
        {
            if (curlength > 20)
                arcnum = curlength - rand() % 10 - 1;
            else if (curlength == 1)
                arcnum = 0;
            else
                arcnum = curlength - rand() % (curlength / 2) - 1;
            qlist.erase(begin(qlist) + arcnum, end(qlist));
            Arc *removearc = arclist[arcnum];
            for (int j = arcnum; j < curlength; j++)
            {
                arclist[j]->clearStar();
            }
            arclist.erase(begin(arclist) + arcnum, end(arclist));

            randDSearch(arclist, qlist, removearc->revAdjVex(), node);
            if (qlist.size() > curlength)
            {
                curlength = qlist.size();
```

```
                lastarclist.assign(arclist.begin(), arclist.end());
                lastqlist.assign(qlist.begin(), qlist.end());
            }
            else
            {
                graph.clearAllStar();
                arclist.assign(lastarclist.begin(), lastarclist.end());
                qlist.assign(lastqlist.begin(), lastqlist.end());
                for (int j = 0; j < arclist.size(); j++)
                {
                    arclist[j]->setStar();
                }
            }
        }
        if (curlength > getMaxlength())
        {
            setMaxLength(curlength);
            setQlist(qlist);
        }
    }

}
```