近似查询实验报告

2011011258 计 13 谭志鹏

实验内容

尝试进行 Jaccard 和 ED 的近似搜索算法实现。算法上使用 q-gram 建立倒排列表，编辑距离计算使用动态规划算法。

算法实现

实现上主要使用了 stl 库的 vector 和 map 数据结构。

```
class SimSearcher
{
public:
    vector<string> vecstr;
    vector<int> gramcounts;
    int smin;
    int gramlen;

    map< string, vector<pair<int, int> > > mapidx;
    map< string, vector<pair<int, int> > >::iterator mapiter;
    int** distance;
```

其中 vecstr 记录所有的串。Mapidx 为倒排列表，用于存储 gram 到包含相应 gram 的编号。之所以使用 vector<pair<int, int> >记录是考虑到重复 gram 的出现，其中第一项记录编号，第二项记录该编号的串有几个相应 gram。这样在计算某一个 gram 匹配数时，应该取 query 串和数据串出现的最少次数。

索引建立：对于每一个数据串，调用 addIndex 函数往 mapidx 倒排表中添加。

```
void SimSearcher::addIndex(string data, int index)
{
    map<string, int> grams = splitgram(data);
    map<string, int>::iterator gramiter;

    for(gramiter = grams.begin(); gramiter != grams.end(); gramiter++)
    {
        string gram = gramiter->first;
        int repeat = gramiter->second;

        mapiter = mapidx.find(gram);
        if(mapiter == mapidx.end())
        {
            vector<pair<int, int> > item;
            item.push_back(pair<int, int>(index, repeat));
            mapidx.insert(pair<string, vector<pair<int, int> > >(gram, item));
        }
        else
        {
            mapidx[gram].push_back(pair<int, int>(index, repeat));
        }
    }
    int gramcount = data.length() - gramlen + 1;
    gramcounts.push_back(gramcount);
    if(gramcount < smin) smin = gramcount;
}
```

建立好倒排表后，之后查询串时先通过查询倒排列表筛选出 candidate。其中需要确定需要最少匹配几个 gram。

对于 jaccard,使用公式如下：

```
//|GQ∩GS| ≥ max (τ *|GQ|, (|GQ|+|GSmin|)*τ/(1+τ))
double t1 = threshold * gramcount;
double t2 = (gramcount + smin) * threshold / (1 + threshold);
int T = (int)max(t1, t2);
```

对于 ED 使用公式如下：

```
int T = querystr.length() - gramlen + 1 - gramlen * threshold;
if(T < 0 ) T = 0;
```

计算 jaccard 距离可以直接从到来表的匹配次数得出：

```
string candidate = vecstr[i];
double tjac = (double)(table[i]) / (gramcount + gramcounts[i] - table[i]);
```

对于编辑距离的调用课上说到的剪枝的动态规划算法，只搜索|I – j| < threshold 部分，同时某一行全部大于阀值时直接放回阀值+1：

```
for(i = 1; i <= source.length(); i++)
{
    int jbegin = i - edthre;
    int jend = i + edthre;
    if(jbegin < 1) jbegin = 1;
    if(jend >target.length()) jend = target.length();
    bool toobig = true;
    for(j = jbegin; j <= jend; j++)
    {
        int tmp = 1;
        if((source[i - 1] == target[j - 1]))
        {
            tmp = 0;
        }
        int edIns = distance[i][j - 1] + 1;
        int edDel = distance[i - 1][j] + 1;
        int edRep = distance[i - 1][j - 1] + tmp;
        distance[i][j] = min(min(edIns, edDel), edRep);
        if(distance[i][j] <= edthre) toobig = false;
    }
    if(toobig) return edthre + 1;

}
```

实验结果和总结

程序成功通过评测系统，耗时 40 多秒。制约速度的因素包括进行倒排列表计算时，自己只使用了简单的 scan count 方法，没有使用更加高效的 mergeheap 算法。还有其他一下因素可能包括一些字符串处理方面的细节问题有待优化。

通过这次试验，实现了课堂上说的 q-gram, 倒排列表进行近似匹配的算法， 对于大数据搜索方面有了更多的认识。