\_RuneLinkNode\* CWordBase::Set\_HashMap\_Word\_Tree(\_RuneLinkNode\* pRuneNode, \_Rune\* pRune, int nLayer)

{

int nOfficeSet = pRuneNode->m\_hmapRuneNextMap.Get\_Hash\_Box\_Data((char\* )pRune->m\_szRune);

if(nOfficeSet > 0)

{

\_RuneLinkNode\* pCurrRuneNode = m\_objNodePool.Get\_NodeOffset\_Ptr(nOfficeSet);

if(pCurrRuneNode->m\_objRune == (\*pRune))

{

//如果找到了，则返回当前节点

return pCurrRuneNode;

}

}

//如果没找到，则创建新的

\_RuneLinkNode\* pNode = m\_objNodePool.Create(nLayer);

//printf("[CWordBase::Set\_HashMap\_Word\_Tree]pNode=0x%08x.\n", pNode);

if(NULL == pNode)

{

printf("[CWordBase::Set\_HashMap\_Word\_Tree]node pool is empty.\n");

return NULL;

}

int nNodeOffset = m\_objNodePool.Get\_Node\_Offset(pNode);

pNode->m\_objRune = (\*pRune);

int nPos = pRuneNode->m\_hmapRuneNextMap.Add\_Hash\_Data((char\* )pRune->m\_szRune, nNodeOffset);

if(-1 == nPos)

{

printf("[CWordBase::Set\_HashMap\_Word\_Tree]tree node child is full.\n");

m\_objNodePool.Delete(pNode);

return NULL;

}

return pNode;

}

void CHmmDict::Viterbi(const char\* pData, int nLen, vector<\_Rune>& objRuneList, vector& objResList)

{

//首先把句子拆成一个个字

Sentance\_To\_Rune(pData, nLen, objRuneList);

printf("[CHmmDict::Viterbi]objRuneList Count=%d.\n", objRuneList.size());

int nRowCount = (int)objRuneList.size();

int nColCount = (int)RUNE\_POS\_ALL;

int nMatrixCount = nRowCount \* nColCount;

//马尔科夫模型

//组建显式矩阵(字权重矩阵)

vector objWeightMatrix(nMatrixCount);

//组建隐式矩阵(字状态矩阵)

vector objStatusMatrix(nMatrixCount);

//显式矩阵第一行，第一个字初始状态的概率，对应BEMS

//第一个字的概率为 初始概率 + 字的初始概率

for (int nCol = 0; nCol < nColCount; nCol++)

{

if(nCol == RUNE\_POS\_B)

{

objWeightMatrix[0 + nCol \* nRowCount] = m\_dbStart[nCol] + Get\_Rune\_Prob(objRuneList[0], m\_hashMapB);

objStatusMatrix[0 + nCol \* nRowCount] = -1;

}

else if(nCol == RUNE\_POS\_E)

{

objWeightMatrix[0 + nCol \* nRowCount] = m\_dbStart[nCol] + Get\_Rune\_Prob(objRuneList[0], m\_hashMapE);

objStatusMatrix[0 + nCol \* nRowCount] = -1;

}

else if(nCol == RUNE\_POS\_M)

{

objWeightMatrix[0 + nCol \* nRowCount] = m\_dbStart[nCol] + Get\_Rune\_Prob(objRuneList[0], m\_hashMapM);

objStatusMatrix[0 + nCol \* nRowCount] = -1;

}

else

{

objWeightMatrix[0 + nCol \* nRowCount] = m\_dbStart[nCol] + Get\_Rune\_Prob(objRuneList[0], m\_hashMapS);

objStatusMatrix[0 + nCol \* nRowCount] = -1;

}

}

//填充矩阵的其余部分(按字的顺序)

for(int nRow = 1; nRow < nRowCount; nRow++)

{

for (int nCol = 0; nCol < nColCount; nCol++)

{

int nCurrPos = nRow + nCol\*nRowCount;

objWeightMatrix[nCurrPos] = MIN\_DOUBLE;

objStatusMatrix[nCurrPos] = RUNE\_POS\_E;

double dbCurrProb = MIN\_DOUBLE;

if(nCol == RUNE\_POS\_B)

{

dbCurrProb = Get\_Rune\_Prob(objRuneList[nRow], m\_hashMapB);

}

else if(nCol == RUNE\_POS\_E)

{

dbCurrProb = Get\_Rune\_Prob(objRuneList[nRow], m\_hashMapE);

}

else if(nCol == RUNE\_POS\_M)

{

dbCurrProb = Get\_Rune\_Prob(objRuneList[nRow], m\_hashMapM);

}

else

{

dbCurrProb = Get\_Rune\_Prob(objRuneList[nRow], m\_hashMapS);

}

//寻找和上一个字的对应关系，取概率最高的关系作为当前概率(算法核心)

for (int nPreCol = 0; nPreCol < nColCount; nPreCol++)

{

int nOldPos = nRow - 1 + nPreCol \* nRowCount;

double dbTemp = objWeightMatrix[nOldPos] + m\_dbTransProb[nCol + nPreCol\*nColCount] + dbCurrProb;

if (dbTemp > objWeightMatrix[nCurrPos])

{

objWeightMatrix[nCurrPos] = dbTemp;

objStatusMatrix[nCurrPos] = nPreCol;

}

}

}

}

//打印隐式矩阵

/\*

printf("=============(Status Matrix)=================\n");

for(int nRow = 0; nRow < nRowCount; nRow++)

{

for (int nCol = 0; nCol < nColCount; nCol++)

{

int nCurrPos = nRow + nCol\*nRowCount;

printf("%d ", objStatusMatrix[nCurrPos]);

}

printf("\n");

}

printf("=============(Status Matrix)=================\n");

\*/

//获得最后末尾的S和E，因为末尾的字只可能是这两个状态之一

double dbEndE = objWeightMatrix[nRowCount - 1 + RUNE\_POS\_E\*nRowCount];

double dbEndS = objWeightMatrix[nRowCount - 1 + RUNE\_POS\_S\*nRowCount];

short sStat = RUNE\_POS\_E;

if(dbEndE < dbEndS)

{

sStat = RUNE\_POS\_S;

}

objResList.resize(nRowCount);

for(int i = nRowCount - 1; i >= 0; i--)

{

objResList[i] = sStat;

sStat = objStatusMatrix[i + sStat\*nRowCount];

}

}