**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC MÁY TÍNH**



**BÁO CÁO**

**XỬ LÝ NGÔN NGỮ TỰ NHIÊN**

**NÂNG CAO**

XÂY DỰNG MỘT MINI PARSER VỚI PCFG

Giáo viên hướng dẫn : TS. Nguyễn Tuấn Đăng

Sinh viên thực hiện : Nguyễn Kim

MSSV : 11520193

*TP Hồ Chí Minh, tháng 01 năm 2015*

MỤC LỤC

[I. Giới thiệu 1](#_Toc409190768)

[1. PCFG (Probabilistic Context Free Grammar) 1](#_Toc409190771)

[2. Nạp dữ liệu cho chương trình (TreeBanks) 1](#_Toc409190771)

[II. Các chức năng chính 2](#_Toc409190769)

[1. Xử lý TreeBanks thành văn phạm 2](#_Toc409190771)

[2. Chuyển văn phạm CFG thành CNF 4](#_Toc409190771)

[3. Sử dụng thuật toán CKY để phân tích câu 6](#_Toc409190771)

[4. Xuất văn phạm câu phân tích 8](#_Toc409190771)

[III. Chạy thử nghiệm chương trình 11](#_Toc409190770)

[1. Sử dụng chương trình 11](#_Toc409190771)

[2. Chạy chương trình 11](#_Toc409190771)

**I. Giới thiệu**

**1. Giới thiệu về PCFG**

Mô hình CFG là một mô hình sử dụng văn phạm phi ngữ cảnh để biểu diễn tập luật cú pháp. Đối với ngôn ngữ tự nhiên, mà đặc biệt là tiếng Việt, một loại ngôn ngữ có tính phức tạp rất cao về mặt ngữ pháp thì sự tự do và ít bị ràng buộc của văn phạm phi ngữ cảnh là một lựa chọn rất hợp lí.

Một mô hình xác suất văn phạm phi ngữ cảnh (PCFG) được định nghĩa đơn giản là mô hình CFG mà ở đó mỗi một luật sinh Nij sẽ được gán kèm thêm với một xác suất P(Nij) tương ứng. Và các xác suất này phải thỏa mãn điều kiện:

ij ) =1.

Cách tính xác suất của một cây phân tích trong mô hình PCFG: xác suất được tính bằng tích tất cả các xác suất của các luật đã được sử dụng trong cây phân tích đó.

**2. Nạp dữ liệu cho chương trình (TreeBanks)**

**Giới thiệu về Pentreebank**

Pentreebank Là Treebank được Đại học Pennsylvania xây dựng, gồm các tập nhãn ngữ pháp riêng, dược biểu diễn dưới bằng cây cú pháp trên văn bản.

**Các câu treebanks sử dụng :**

*Anh ấy đang học đại\_học.*

(S(NP(NN anh)(JJD ấy))(VP(RB đang)(VB học)(NN đại\_học)))

*Anh ấy học ngành hóa\_học.*

(S(NP(NN anh)(JJD ấy))(VP(VB học)(NP(NN ngành)(NN hóa\_học))))

*Nó học rất giỏi.*

(S(NP(PRP nó))(VP(VB học)(RB rất)(JJ giỏi)))

*Nó sắp đi học.*

(S(NP(PRP nó))(VP(RB sắp)(VB đi)(VB học)))

*Gia\_đình của nó sống ở thành\_phố.*

(S(NP(NN gia\_đình)(PP(IN của)(NP(PRP nó))))(VP(VB sống)(PP(IN ở)(NP(NN thành\_phố)))))

*Anh ấy cũng có một căn nhà ở quê.*

(S(NP(NN anh)(JJD ấy))(VP(RB cũng)(VB có)(NP(CD một)(NP(UN căn)(NN nhà))(PP(IN ở)(NN quê)))))

*Nó có một anh trai.*

(S(NP(PRP nó))(VP(VB có)(NP(CD một)(NP(NN anh)(JJ trai)))))

*Nó thích về quê vào chủ\_nhật.*

(S(NP(PRP nó))(VP(VB thích)(VP(VB về)(NP(NN quê)(PP(IN vào)(NN chủ\_nhật))))))

*Căn nhà ở quê rất đẹp.*

(S(NP(UN căn)(NN nhà)(PP(IN ở)(NN quê)))(VP(RB rất)(VB đẹp)))

*Nó đang làm bài.*

(S(NP(PRP nó))(VP(RB đang)(VB làm)(NP(NN bài))))

**II. Các chức năng chính**

**1. Xử lý treebanks thành văn phạm ( button LOAD).**

Button LOAD có chức năng đọc file text treebanks, xuất văn phạm và tính xác suất, sau đó load lên ListView.

Ngoài ra có thể bổ sung treebanks thông qua button LOAD bằng một file text mới, chương trình sẽ tính toán xác xuất các quy tắc.

Với treebank : (S(NP(PRP nó)(VP(RB đang)(VB làm)(NP(NN bài)))).

Ta có được các văn phạm sau:

S 🡪 NP VP. NP 🡪 NN.

NP 🡪 PRP. NN 🡪 bài.

PRP 🡪 nó.

VP 🡪 RB VB NP.

RB 🡪 đang.

VB 🡪 làm.

Với mỗi chuỗi treebank ta duyệt các dấu đóng ngoặc “)”, khi gặp nó ta duyệt ngược lại cho đến khi gặp dấu đóng ngoặc “(“. Khi đó chuỗi trong cặp ngoặc “( )” là văn phạm ta cần lấy. Sau khi lấy được đoạn văn phạm ta thực hiện thay thế đoạn chuỗi trên kể cả cặp ngoặc bằng non-terminal vế trái.

Ví dụ: (S(NP(PRP nó)(VP(RB đang)(VB làm)(NP(NN bài)))).

Sau khi lấy được văn phạm PRP 🡪 nó. Ta thực hiện thay thế, đoạn treebank trở thành:

(S(NP PRP(VP(RB đang)(VB làm)(NP(NN bài)))).

Hàm layvanpham() thực hiện việc lấy văn phạm từ treebank.

private void layvanpham(string dulieu)

{

string[] mang = dulieu.Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

for (int i = 0; i < mang.Length; i++)

{

for (int k = 0; k < mang[i].Length; k++)

{

if ((mang[i][k] == ')') && (k == mang[i].Length - 1))

{

string[] hoanthanh = mang[i].Substring(1, mang[i].Length - 2).Split(' ');

string vpht = string.Format(" {0} -->", hoanthanh[0]);

for (int l = 1; l < hoanthanh.Length; l++)

{

vpht = vpht + " " + hoanthanh[l];

}

vanpham.Add(vpht);

break;

}

if (mang[i][k] == ')')

{

for (int j = k; j >= 0; j--)

{

if (mang[i][j] == '(')

{

string sub = mang[i].Substring(j + 1, k - j - 1);

string[] mangsub = sub.Split(' ');

string vp = string.Format(" {0} -->", mangsub[0]);

for (int m = 1; m < mangsub.Length; m++)

{

vp = vp + " " + mangsub[m];

}

vanpham.Add(vp);

mang[i] = mang[i].Replace(mang[i].Substring(j, k - j + 1), " " + mangsub[0]);

k = k - (sub.Length + 2 - mangsub[0].Length - 1);

break; }}}}}}

**2. Chuyển văn phạm CFG thành CNF ( button CNF).**

Sau khi load treebank, button CNF sẽ thực hiện chuyển văn phạm CFG thành CNF.

Trường hợp 1: VP 🡪 RB VB NN.

Khi gặp vế phải của văn phạm có 3 nhãn, chương trình sẽ tự động đặt biến phụ theo thứ tự X1. X2, X3.. để gán cho 2 nhãn đầu. Như vậy ta sẽ có 2 văn phạm:

VP 🡪 RB VB NN.

X1 🡪 RB VB.

Trong quá trình bổ sung ngữ liệu treebank, nếu gặp phải văn phạm trên, chương trình sẽ kiểm tra trong: List<string> cnf32 = new List<string>(); xem có biến phụ nào tương tự để tự động tăng tần số chứ không tạo ra biến phụ mới.

Trường hợp 2: Câu có văn phạm: NP 🡪 PRP, PRP 🡪 nó.

Sau khi tách văn phạm, chương trình đổ danh sách văn phạm vào trong List: List<string> vanpham = new List<string>();

Ta thực hiện duyệt các văn phạm bắt đầu bằng NP, VP, PP… và có vế phải là một nhãn. Như vậy văn phạm tìm được và văn phạm tiếp theo trong List là vi phạm CNF, ta bổ sung văn phạm với vế trái là nhãn của văn phạm tìm được và vế phải là nhãn vế phải của văn phạm tiếp theo. Sau đó ta xóa bỏ 2 văn phạm trên ra khỏi List.

Với trường hợp trên, ta có văn phạm bổ sung là : NP 🡪 nó.

Xử lý sự kiện button CNF

private void btnCNF\_Click(object sender, EventArgs e)

{

int n = cnf32.Count + 1;

for (int i = 0; i < vanpham.Count; i++)

{

string[] m = vanpham[i].Substring(vanpham[i].IndexOf('>') + 2).Split(' ');

string nw = vanpham[i].Substring(1, vanpham[i].IndexOf('-') - 2);

if (m.Length > 2)

{

int vt = kiemtra(cnf32, m[0] + " " + m[1]);

if (vt >= 0)

{

vanpham[i] = vanpham[i].Replace(m[0] + " " + m[1], cnf32[vt].Substring(1, cnf32[vt].IndexOf('-') - 2));

vanpham.Add(cnf32[vt]);

}

else

{

string s = string.Format(" {0} --> {1} {2}", "X" + n.ToString(), m[0], m[1]);

vanpham[i] = vanpham[i].Replace(m[0] + " " + m[1], "X" + n.ToString());

n = n + 1;

vanpham.Add(s);

cnf32.Add(s);

}

}

if ((m.Length == 1) && ((nw == "NP") || (nw == "PP") || (nw == "VP")))

{

vanpham[i - 1] = vanpham[i - 1].Replace(m[0], nw);

vanpham.RemoveAt(i);

i = i - 1;

}

}

for (int i = 0; i < vanpham.Count; i++)

{

cnf.Add(vanpham[i]);

}

vanpham.Clear();

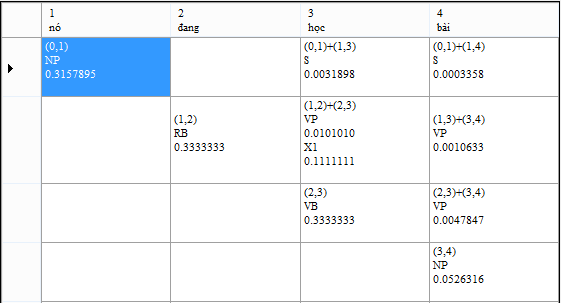
LoadListView(); }

**3. Sử dụng thuật toán CKY để phân tích câu (button Calculate)**

Button Calculate thực hiện thuật toán CKY trên mảng pcky = new string[n, n]; và xuất kết quả lên datagirdview.

Ví dụ:

Phân tích câu: Nó đang học bài



Thuật toán sử dụng 3 vòng lặp :

for (int i = 2; i <= n; i++)

{

for (int j = 1; j <= n - i + 1; j++)

{

for (int k = 1; k <= i - 1; k++)

{ Tìm văn phạm có vế trái là 2 nhãn của 2 ô:

[j – 1 , j – 1 + k – 1]

[j – 1 + k, j – 1 + k + i – k – 1]

Gán nhãn vế trái của văn phạm tìm được vào ô:

[j – 1, j – 1 + k + i – k – 1] }

Vòng lặp thứ nhất xác định độ dài kết hợp các từ trong câu:

Với i = 2 thì kết hợp các cặp 2 từ như ( nó đang, đang học, học bài).

Với i = 3 thì kết hợp các cặp 3 từ như ( nó đang học, đang học bài)

…

Vòng lặp thứ hai xác đinh chỉ số chạy j dựa vào độ dài kết hợp từ i:

Với i = 2 thì chỉ số chạy từ 1 đến 3 (nó, đang, bài).

Với i = 3 thì chỉ số chạy từ 1 đến 2 ( nó, đang).

…

Vòng lặp thứ ba xác định chỉ số chạy k trong độ dài i:

Với i = 3 thì k chạy từ 1 đến 2, duyệt k từ với i – k từ còn lại:

Khi duyệt k = 1 với ô có từ “nó” thuật toán sẽ kết hợp với ô kết hợp bởi 2 từ “đang học”.

Thuật toán sẽ duyệt theo đường chéo chính cả mảng với các ô đường chéo chính lần lượt là các nhãn của các từ trong câu cần phân tích.

Cài đặt thuật toán trong chương trình:

for (int i = 2; i <= n; i++)

{

for (int j = 1; j <= n - i + 1; j++)

{

for (int k = 1; k <= i - 1; k++)

{

if (string.IsNullOrEmpty(pcky[j - 1, j - 1 + k - 1]) == false && string.IsNullOrEmpty(pcky[j - 1 + k, j - 1 + k + i - k - 1]) == false)

{

string[] m1 = pcky[j - 1, j - 1 + k - 1].Substring(pcky[j - 1, j - 1 + k - 1].IndexOf('\n') + 1).Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

string[] m2 = pcky[j - 1 + k, j - 1 + k + i - k - 1].Substring(pcky[j - 1 + k, j - 1 + k + i - k - 1].IndexOf('\n') + 1).Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

foreach (var vp in ts)

{

string sau = vp.Key.Substring(vp.Key.IndexOf('>') + 2);

string[] ms = sau.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if ( ms.Length > 1)

{

for (int l = 0; l < m1.Length; l += 2)

{

for (int t = 0; t < m2.Length; t += 2)

{

if (string.Compare(m1[l] + " " + m2[t], sau) == 0)

{

string dau = vp.Key.Substring(1, vp.Key.IndexOf('-') - 2);

double p = double.Parse(m1[l + 1]) \* double.Parse(m2[t + 1]) \* vp.Value / tsn[dau];

string td = (j - 1).ToString() + (j - 1 + k - 1).ToString() + (j - 1 + k).ToString() + (j - 1 + k + i - k - 1).ToString();

if (string.IsNullOrEmpty(pcky[j - 1, j - 1 + k + i - k - 1]) == true)

{

string s3 = string.Format("{0}\r\n{1}\r\n{2:0.0000000}\r\n",td, dau, p);

pcky[j - 1, j - 1 + k + i - k - 1] = pcky[j - 1, j - 1 + k + i - k - 1] + s3;

}

else

{

string s2 = string.Format("{0}\r\n{1:0.0000000}\r\n", dau, p);

pcky[j - 1, j - 1 + k + i - k - 1] = pcky[j - 1, j - 1 + k + i - k - 1] + s2;

}

}

}

}}}}}}}

**4. Xuất văn phạm câu phân tích (button Extract)**

Sau khi đã có kết quả từ thuật toán, button Extract duyệt đệ quy trên mảng pcky = new string[n, n]; bắt đầu từ vị trí [0 , n – 1] để xuất văn phạm câu tìm được.

Với tọa độ kết hợp trên mỗi ô dạng (i, j)+(j,k) ta duyệt đến vị trí các ô đó và xuất được văn phạm cần tìm. Mỗi ô là một chuỗi với định dạng như sau:

“{tọa độ kết hợp}\r\n{nhãn}\r\n{xác suất}”.

Hàm duyệt đệ quy:

private void dequy(string toado)

{

string toadocon = toado.Substring(0, toado.IndexOf('\r'));

if (toadocon.Length > 1)

{

int a = int.Parse(toadocon[0].ToString());

int b = int.Parse(toadocon[1].ToString());

int c = int.Parse(toadocon[2].ToString());

int d = int.Parse(toadocon[3].ToString());

string[] md = toado.Substring(toado.IndexOf('\n') + 1).Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

string[] ms1 = pcky[a, b].Substring(pcky[a, b].IndexOf('\n') + 1).Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

string[] ms2 = pcky[c, d].Substring(pcky[c, d].IndexOf('\n') + 1).Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

if(md.Length < 3)

{

for(int i = 0; i < ms1.Length; i +=2)

{

for(int k = 0; k < ms2.Length; k +=2)

{

string vp = string.Format(" {0} --> {1} {2}", md[0], ms1[i], ms2[k]);

if (cnf.Contains(vp) == true)

{

output.Add(vp);

}

}

}

if (ms1.Length > 2)

{

stringleft = output[output.Count - 1].Substring(output[output.Count - 1].IndexOf('-'));

}

if(ms2.Length > 2)

{

stringdown = output[output.Count - 1].Substring(output[output.Count - 1].IndexOf('-'));

}

dequy(pcky[a, b]);

dequy(pcky[c, d]);

}

else

{

int vt = 0;

for(int l = 0; l < md.Length; l +=2)

{

if ((stringleft.Contains(md[l]) == true) || (stringdown.Contains(md[l]) == true))

{

vt = l;

break;

}

}

for (int i = 0; i < ms1.Length; i += 2)

{

for (int k = 0; k < ms2.Length; k += 2)

{

string vp = string.Format(" {0} --> {1} {2}", md[vt], ms1[i], ms2[k]);

if (cnf.Contains(vp) == true)

{

output.Add(vp);

}

}

}

if (ms1.Length > 2)

{

stringleft = output[output.Count - 1].Substring(output[output.Count - 1].IndexOf('-'));

}

if (ms2.Length > 2)

{

stringdown = output[output.Count - 1].Substring(output[output.Count - 1].IndexOf('-'));

}

dequy(pcky[a, b]);

dequy(pcky[c, d]);

}

}

else

{

int a = int.Parse(toadocon);

string[] md = toado.Substring(toado.IndexOf('\n') + 1).Split(new char[] { '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

string vp = string.Format(" {0} --> {1}", md[0], m[a]);

output.Add(vp);

}

}

**III. Chạy thử nghiệm chương trình**

**1. Sử dụng chương trình.**

Nhấn button LOAD để nạp file Treebanks.txt vào chương trình.

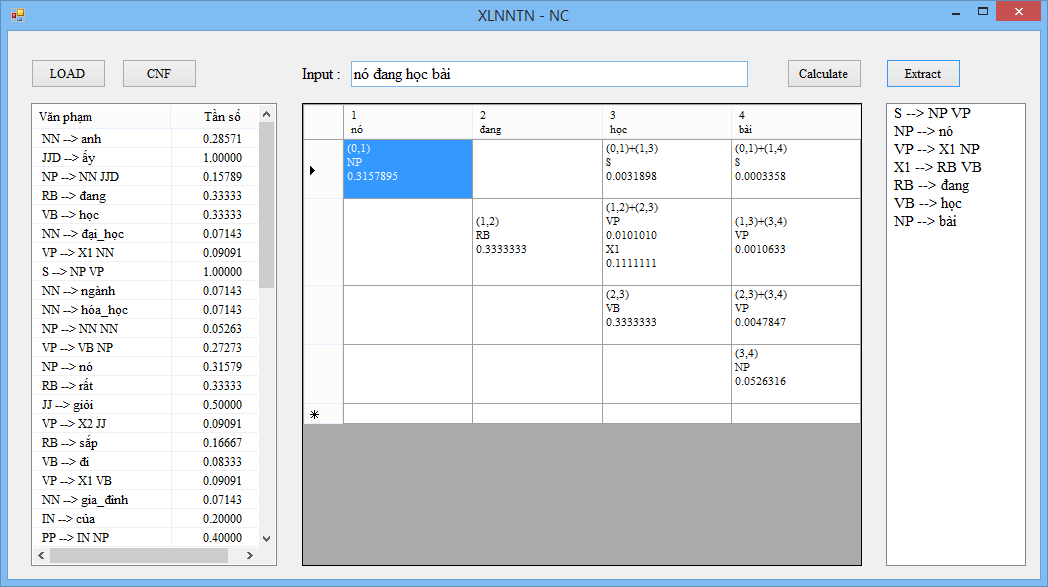
Nhấn button CNF để chuyển văn phạm trên ListView thành CNF (khi bổ sung treebanks bằng một file mới cần nhấn lại button CNF để chuyển văn phạm vừa bổ sung thành CNF và tính toán lại xác suất).

Sau khi ListView đã có văn phạm CNF, nhập câu cần phân tích vào Textbox Input và nhấn button Calculate để chạy thuật toán CKY và kết quả được thể hiện datagirdview.

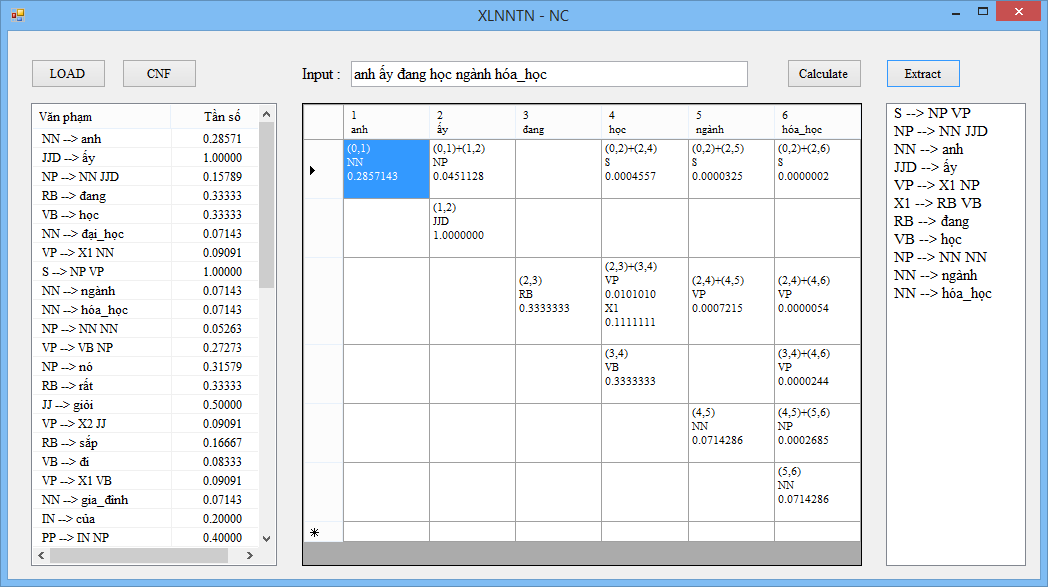
Nhấn button Extract để xuất cây văn phạm từ mảng pcky lên TextBox Extract.

**2. Chạy chương trình**

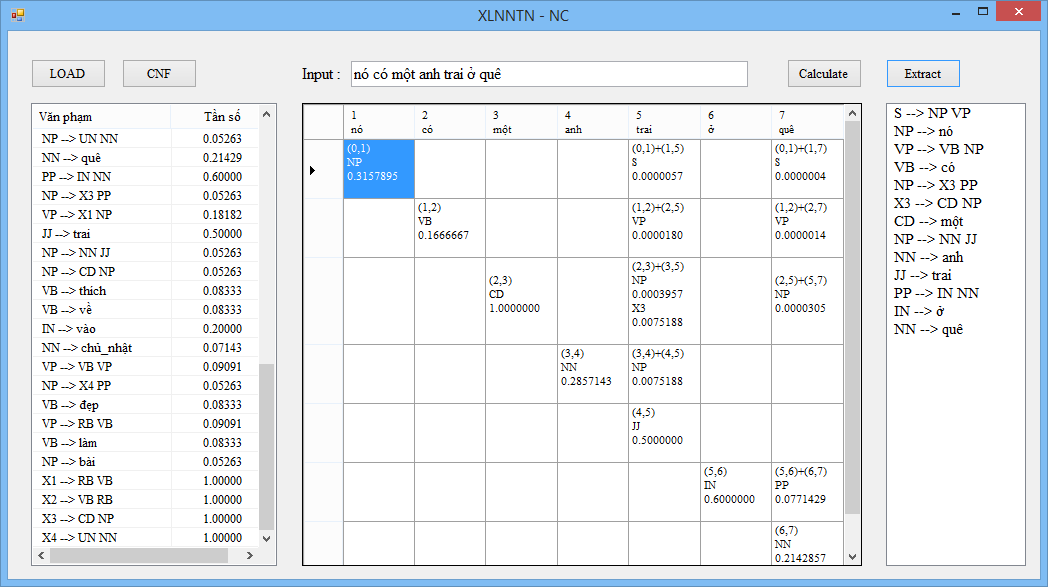
Câu 1: *Nó đang học bài.*



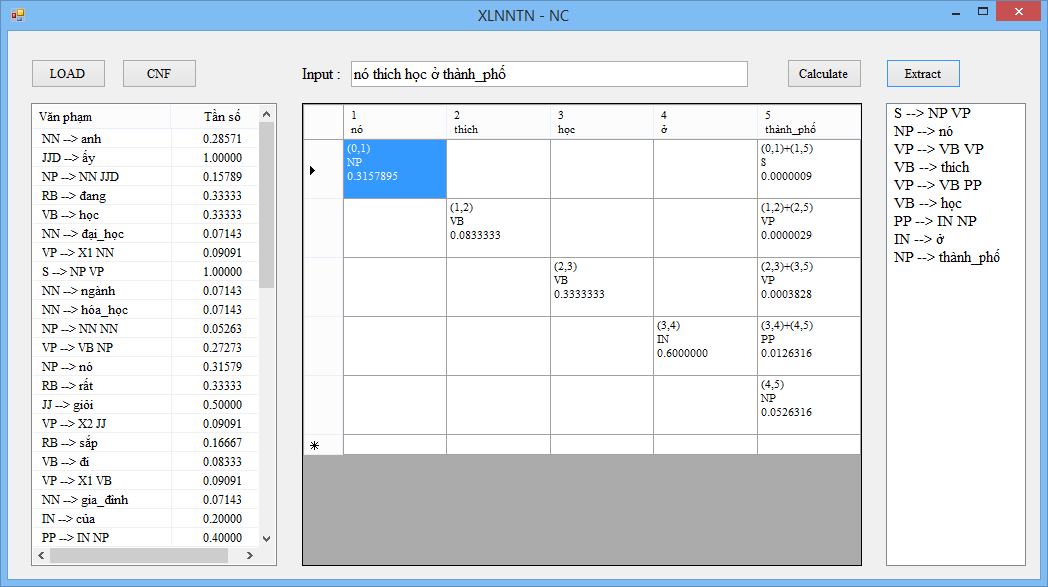
Câu 2 : *Anh ấy đang học ngành hóa\_học.*



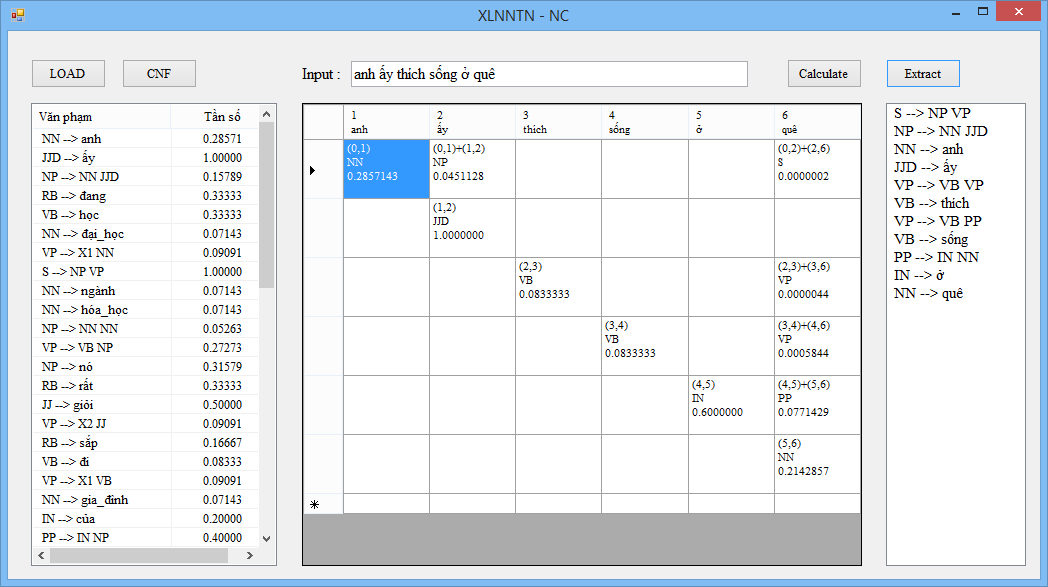
Câu 3: *Nó có một anh trai ở quê.*



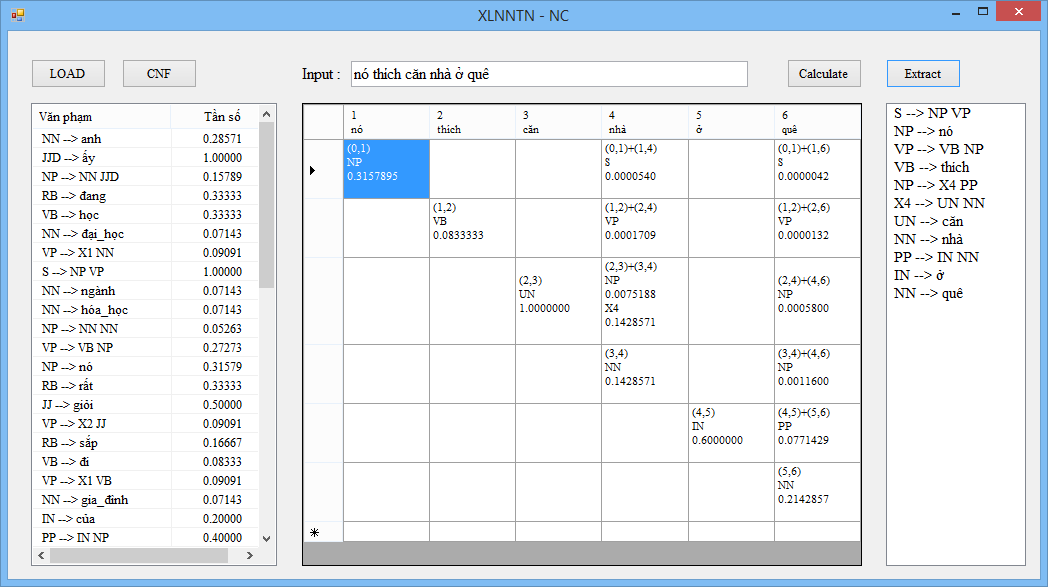
Câu 4: *Nó thích học ở thành\_phố.*



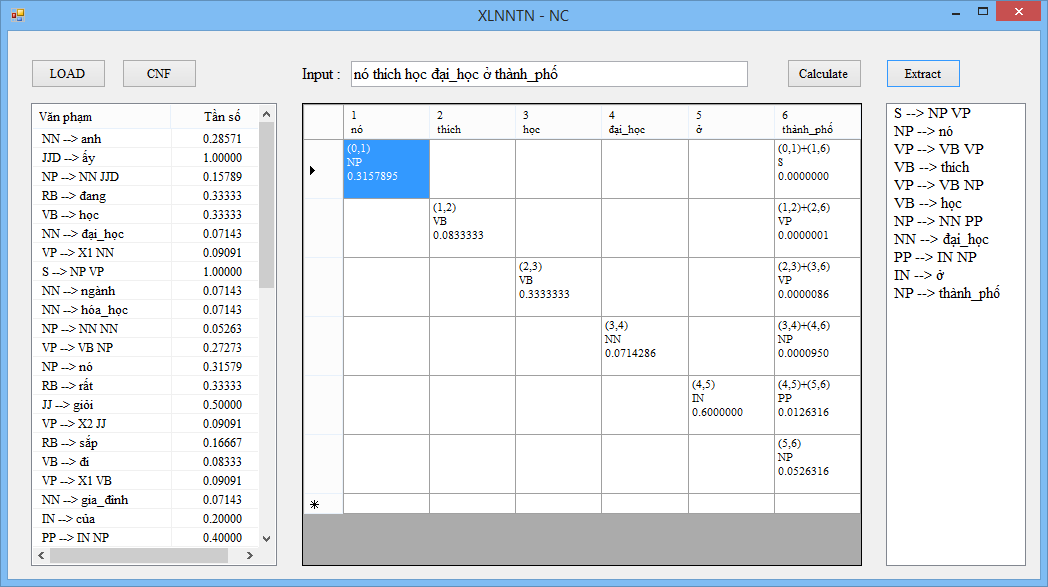
Câu 5: *Anh ấy thích sống ở quê.*



Câu 6: *Nó thích căn nhà ở quê.*



Câu 7: *Nó thích học đại\_học ở thành\_phố.*



Câu 8: *Anh ấy sắp có một căn nhà ở thành\_phố ( câu có 2 cách phân tích).*

