**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.\*

**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:**

***Задание 1.***

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

***Задание 2.***

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

***Задание 3.***

Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

***Задание 4.***

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

***Задание 5.***

**Нечетные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять в соответствии с вариантом. Построить графики зависимости времени вычисления от . **Отобразить ход решения в отчете**(по примеру из лекции) + код и копии экрана.

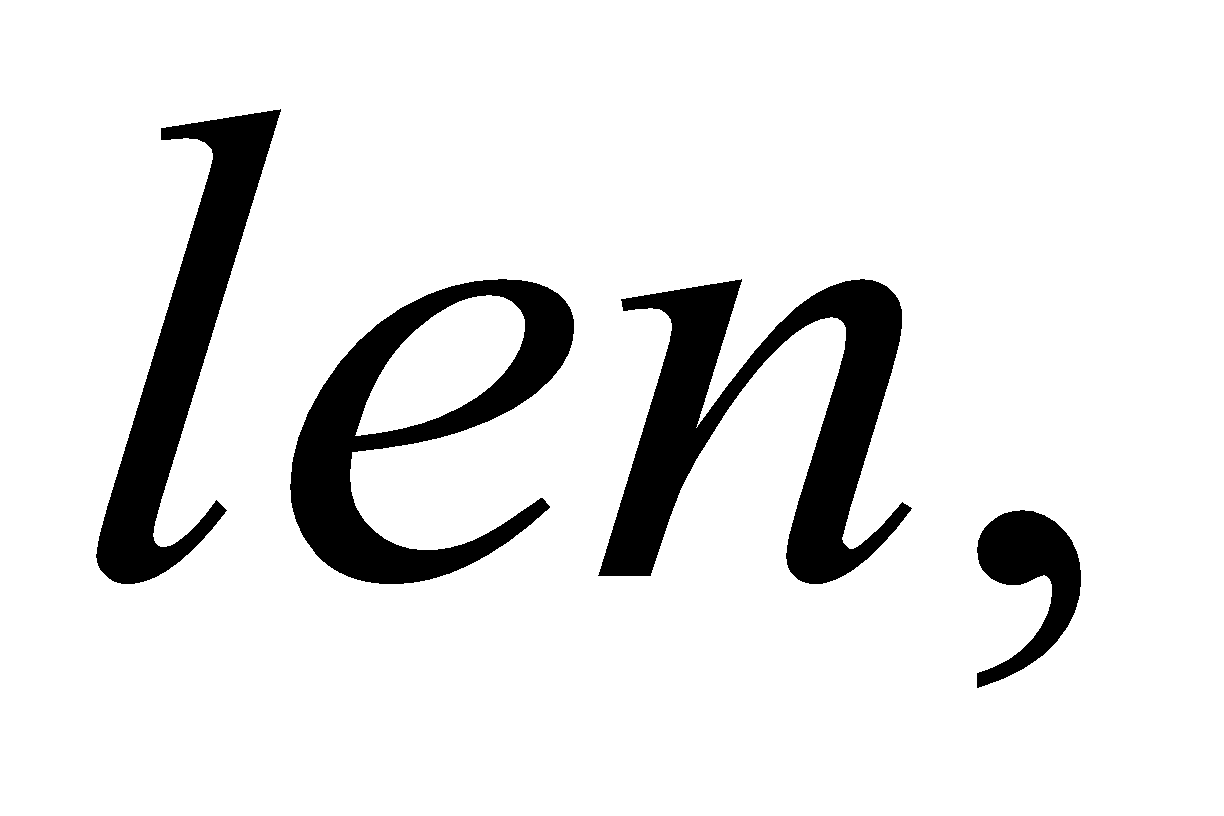
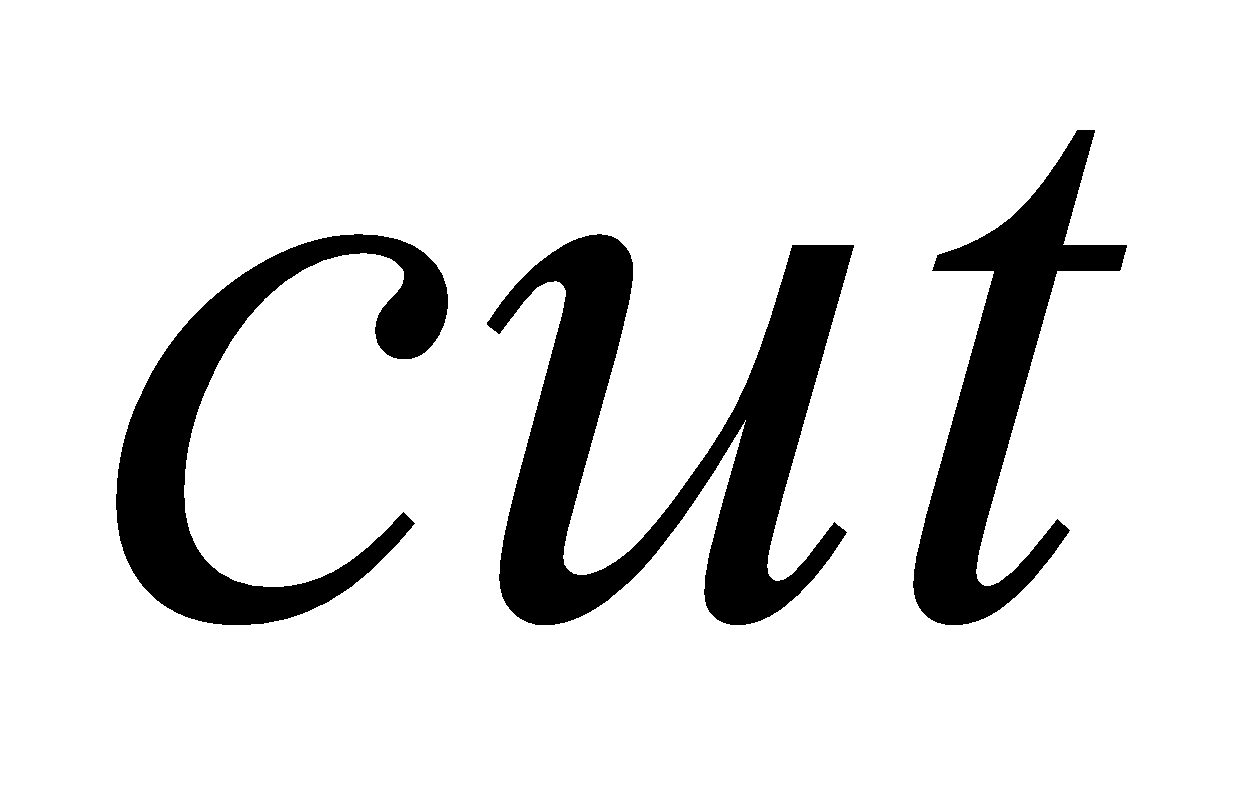
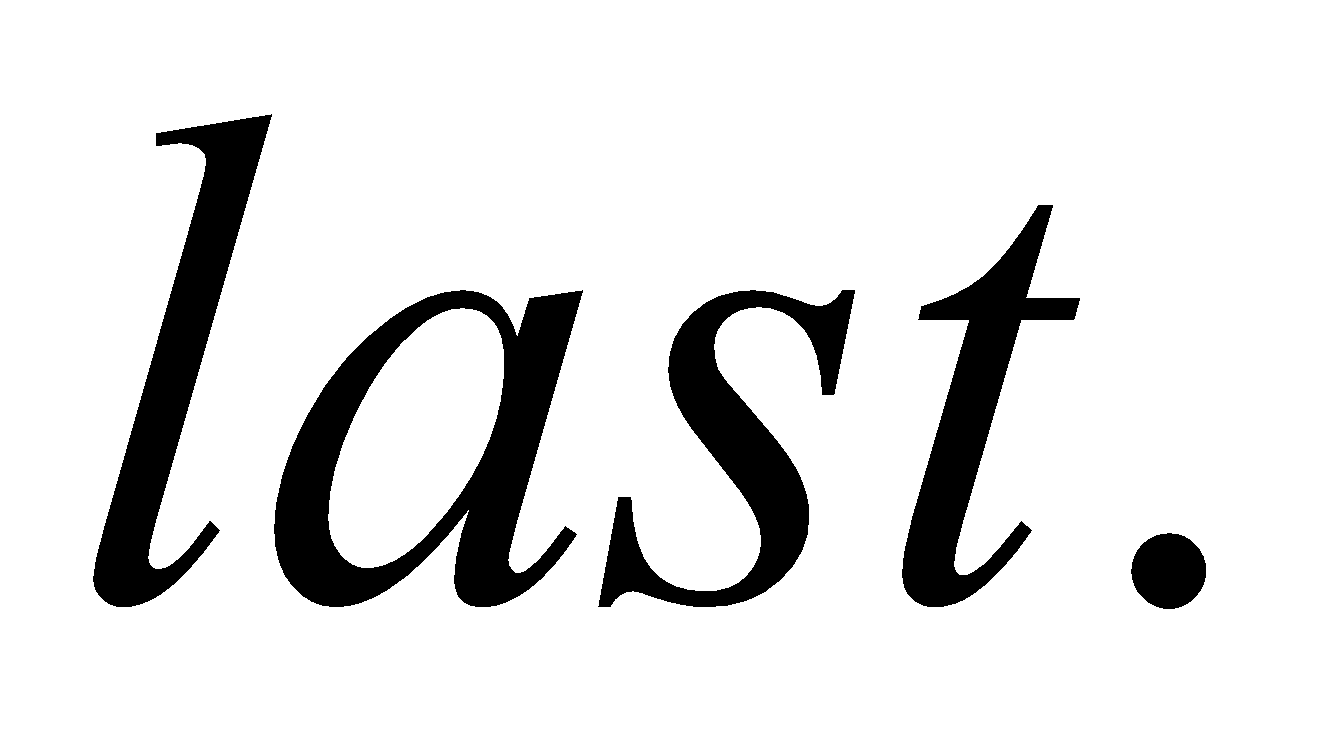
**Четные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | Задание 4 | | Задание 5 | |
|  | Бес | Блеск | ALBDACD | CDLDCA |
|  | Том | Исток | 7\*10, 10\*18, 18\*21, 21\*28, 28\*38, 38\*49 | |
|  | Лом | Гомон | ABCDFGI | EATUFI |
|  | Акр | Якорь | 20\*15, 15\*30, 30\*53, 53\*10, 10\*20, 20\*11 | |
|  | Сон | Донор | HRWCYUJ | WLPCAU |
|  | Сом | Домик | 9\*12, 12\*20, 20\*23, 23\*30, 30\*40, 40\*51 | |
|  | Кол | Столб | QVTWNHO | RQTWYK |
|  | Вар | Баран | 10\*15, 15\*80, 80\*23, 23\*50, 50\*40, 40\*71 | |
|  | Вол | Колун | TBHDSAV | KIBYSV |
|  | Ель | Дрель | 100\*15, 15\*20, 20\*43, 43\*70, 70\*40, 40\*71 | |
|  | Лот | Полет | TOUEXAZ | HIEHXZ |
|  | Эхо | Хорек | 8\*11, 11\*19, 19\*22, 22\*29, 29\*39, 39\*50 | |
|  | Раб | Барка | BXWAFRE | XCDUFR |
|  | Сан | Сонар | 60\*15, 15\*50, 50\*53, 53\*10, 10\*30, 30\*11 | |
|  | Сын | Фасон | MIOPLKJ | GUIOLW |
|  | Кон | Локон | 10\*52, 52\*100, 100\*33, 33\*50, 50\*40, 40\*21 | |

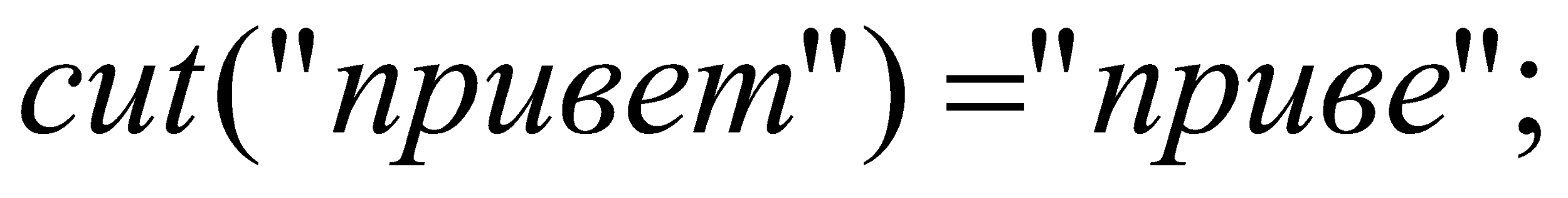
**ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ**

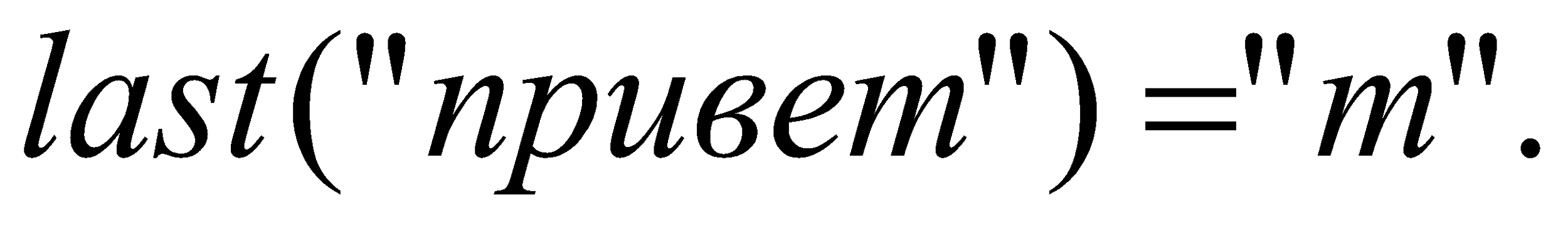
1. **Вычисление дистанции Левенштейна**



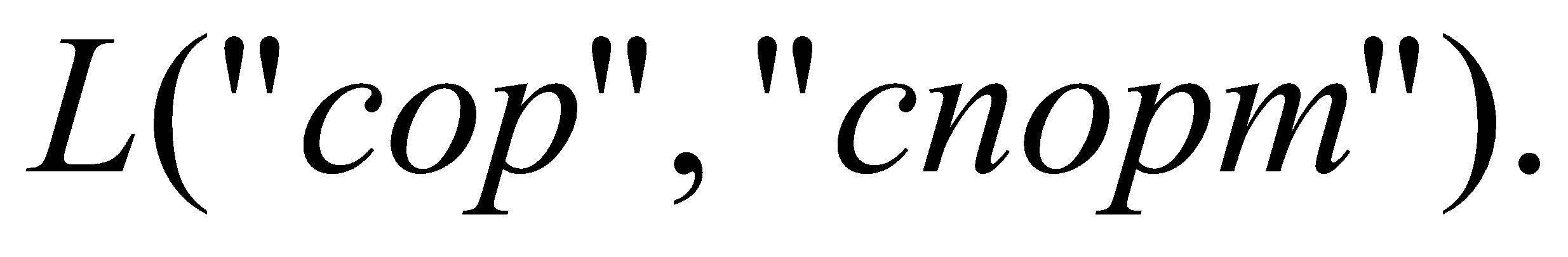
В предыдущем выражении используются символы   и  Разъясним их смысл:

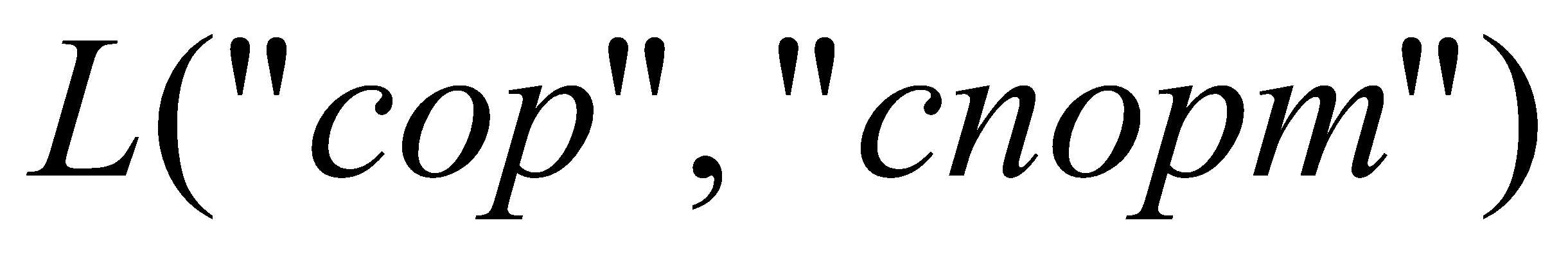
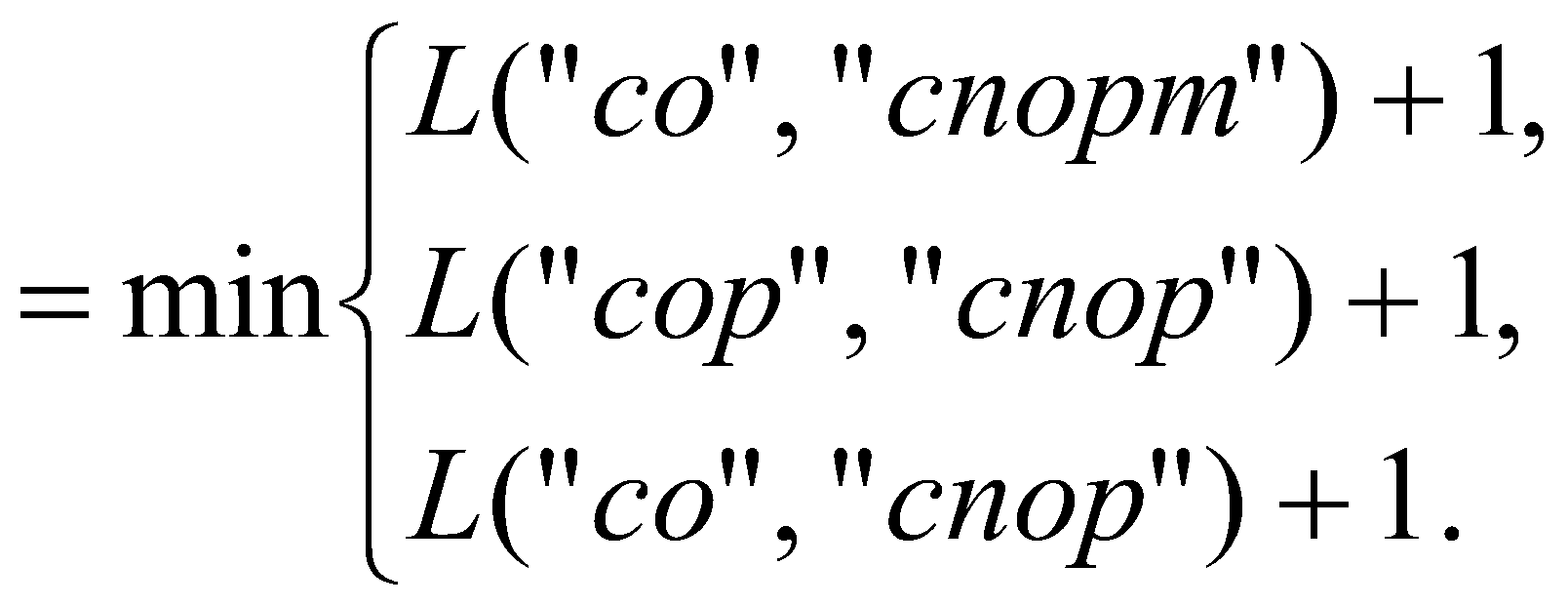
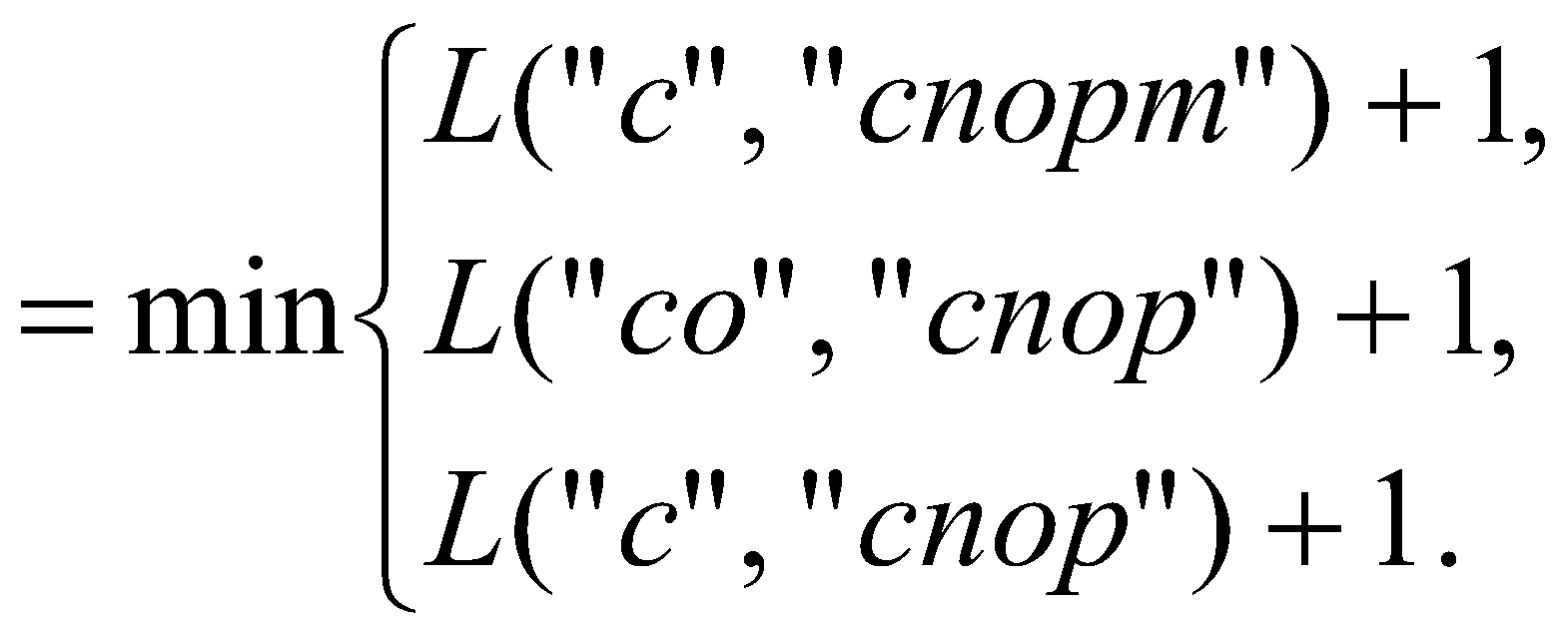
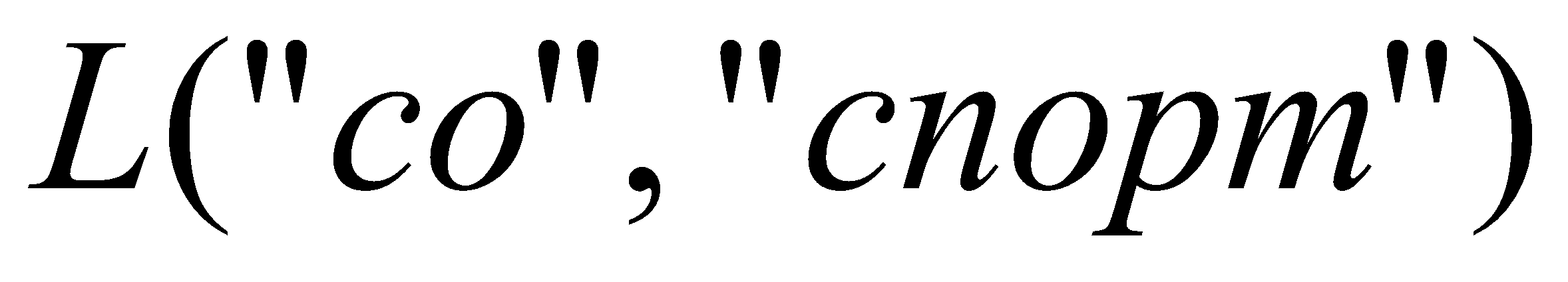
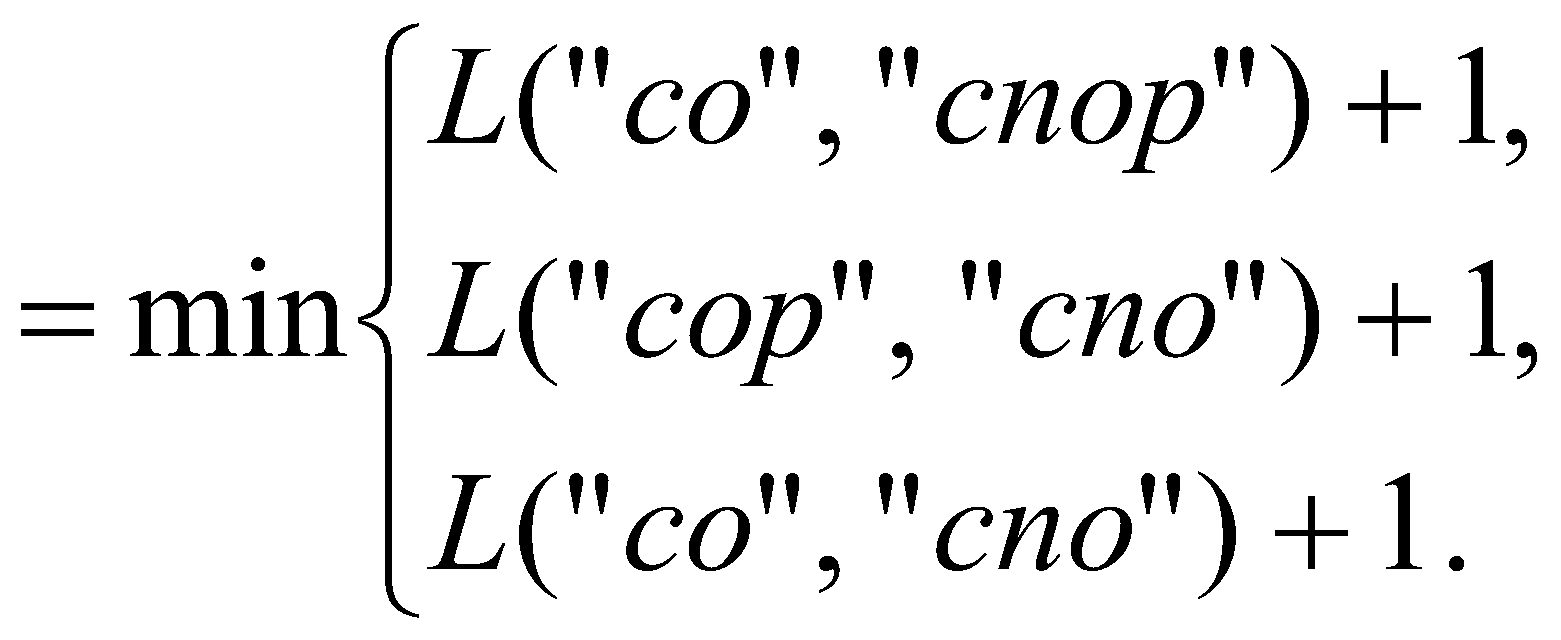
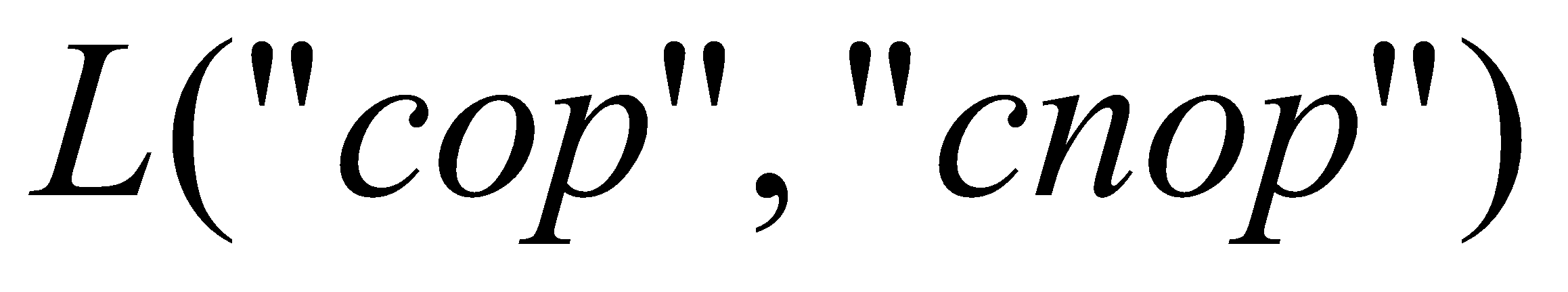
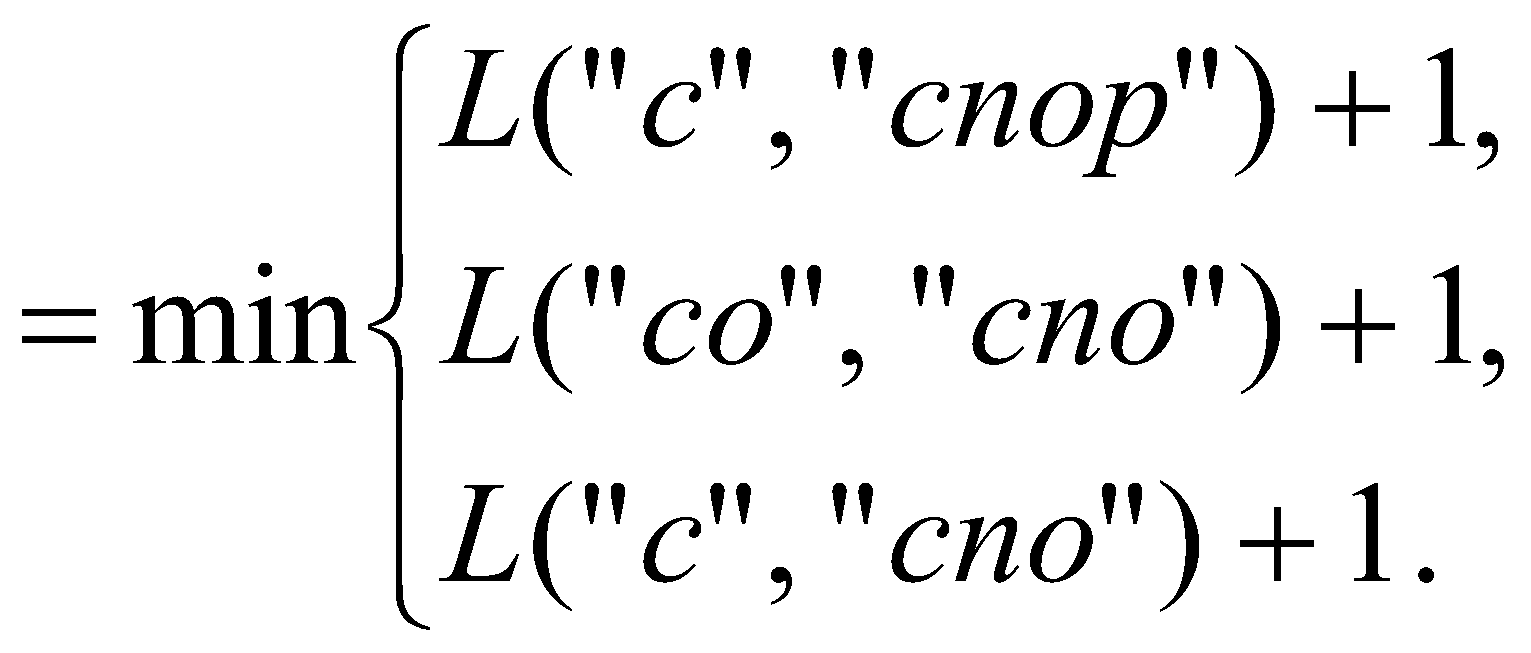
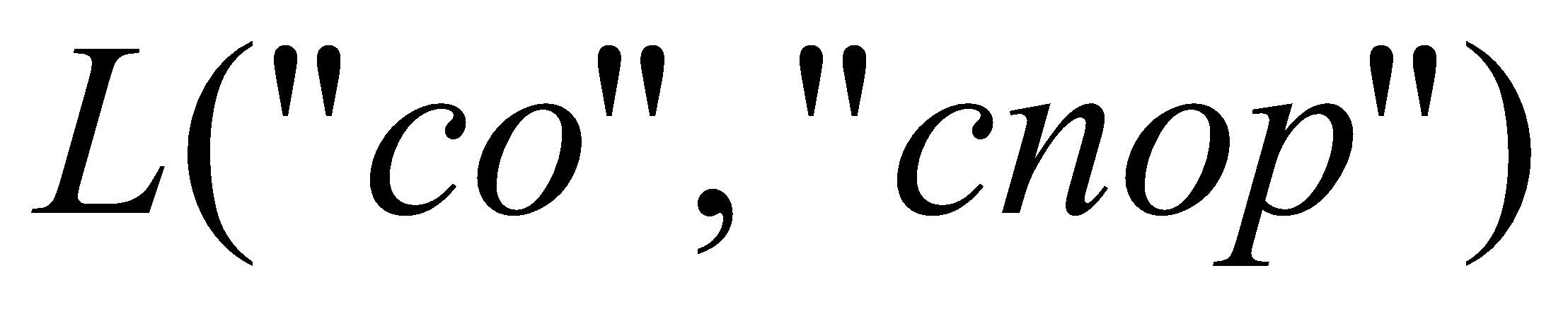
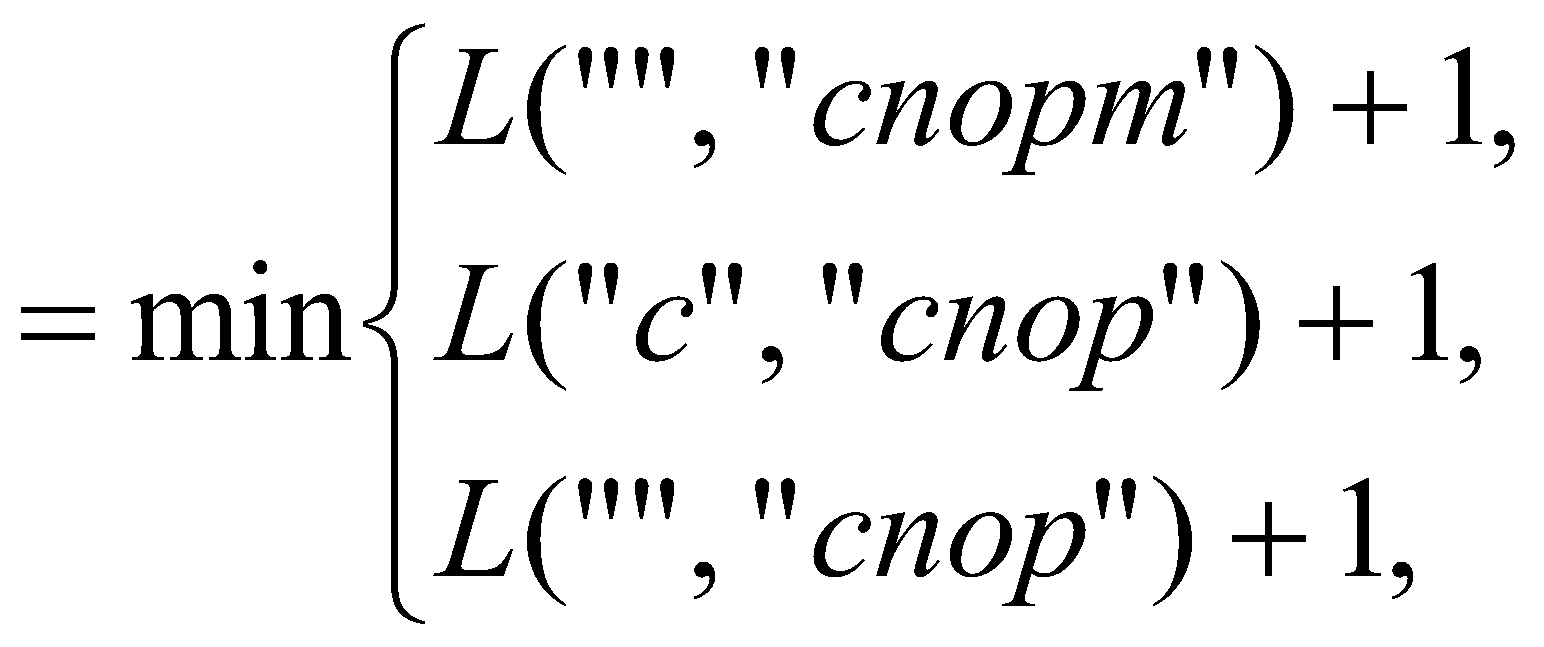
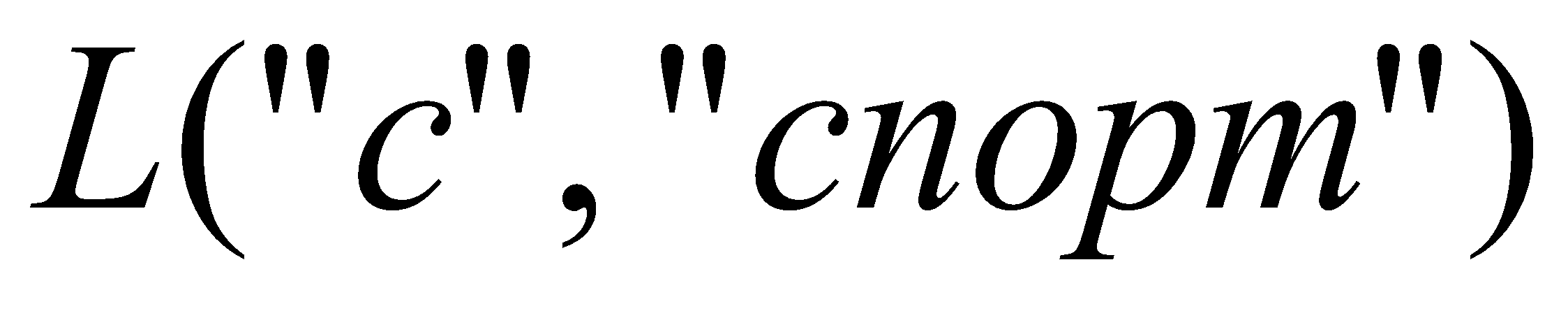
 – количество символов в заданной строке. Например, 

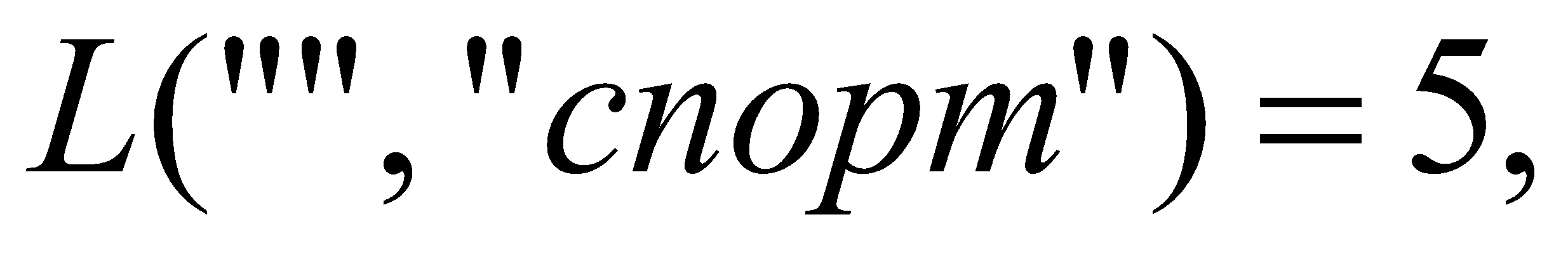
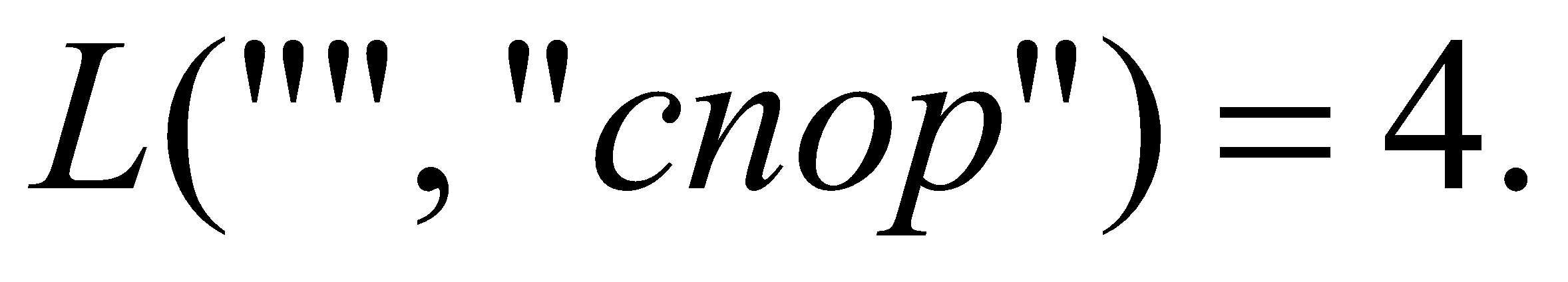
 – заданная строка без последнего символа. Например, 

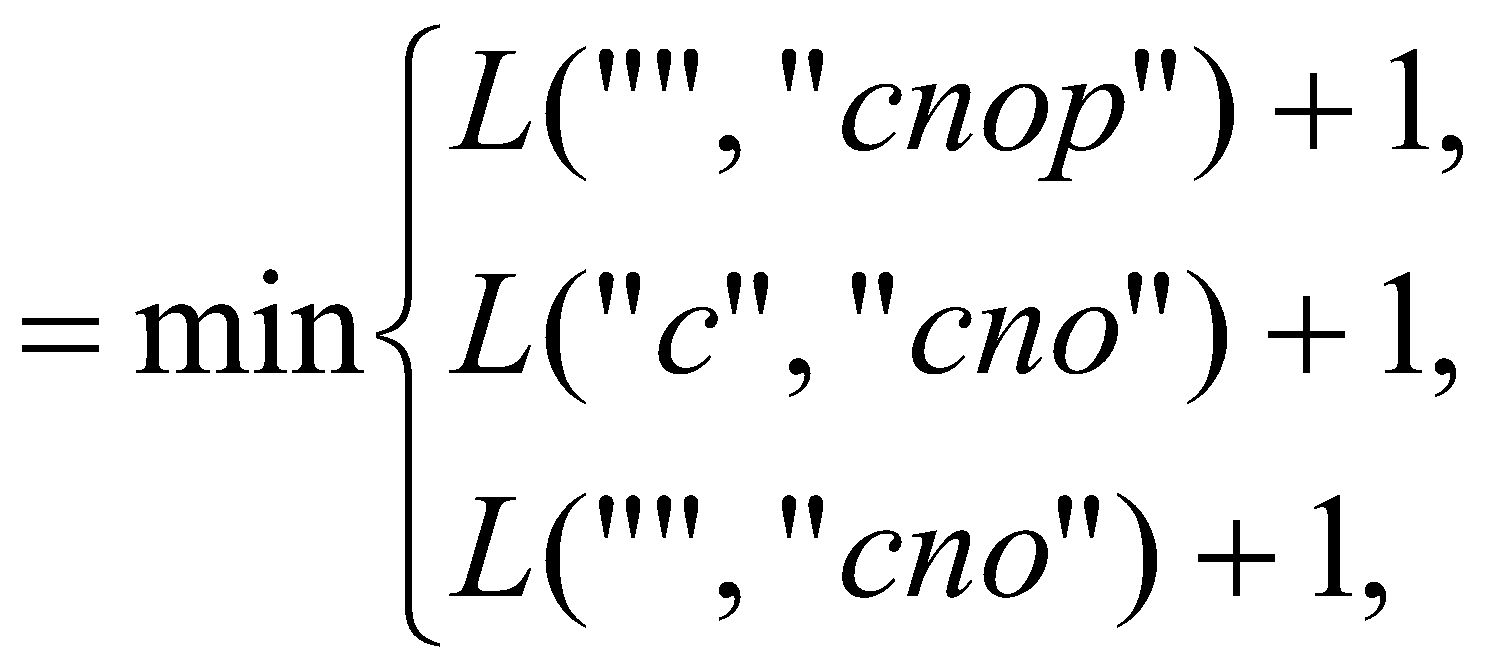
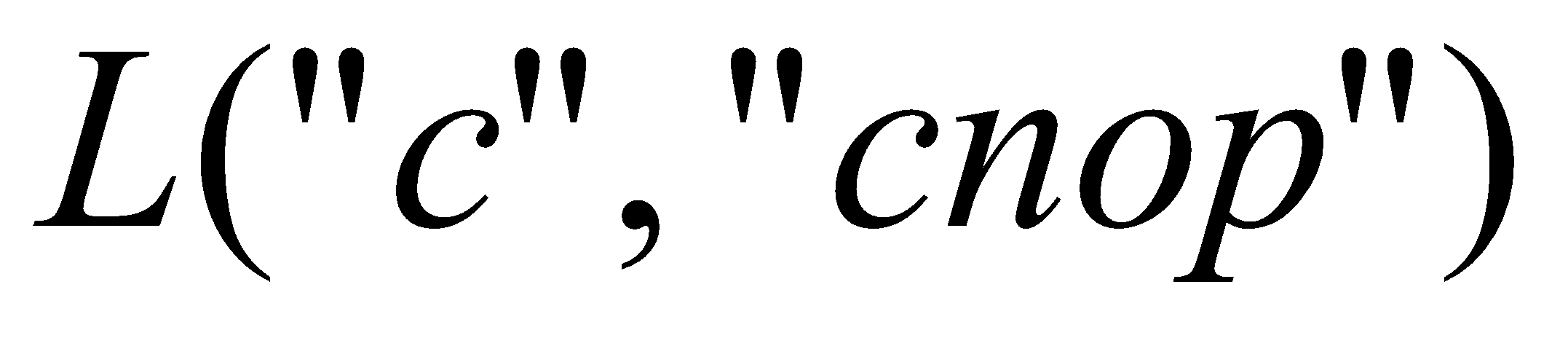
 – последний символ заданной строки. Например, 

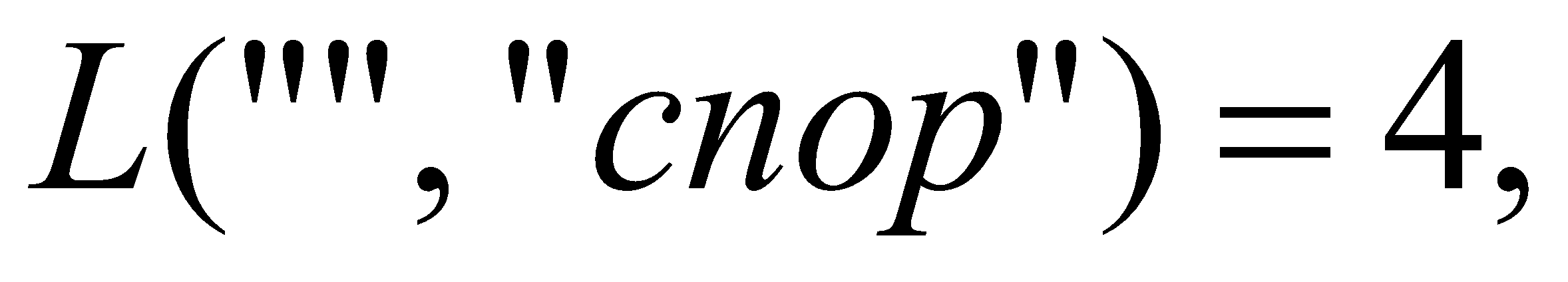
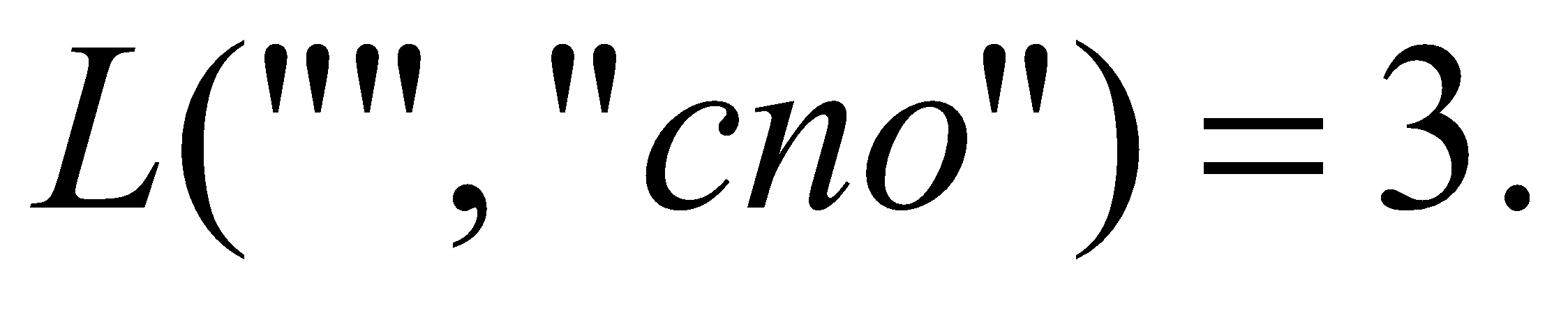
Поясним принцип применения этого рекуррентного соотношения на следующем примере.

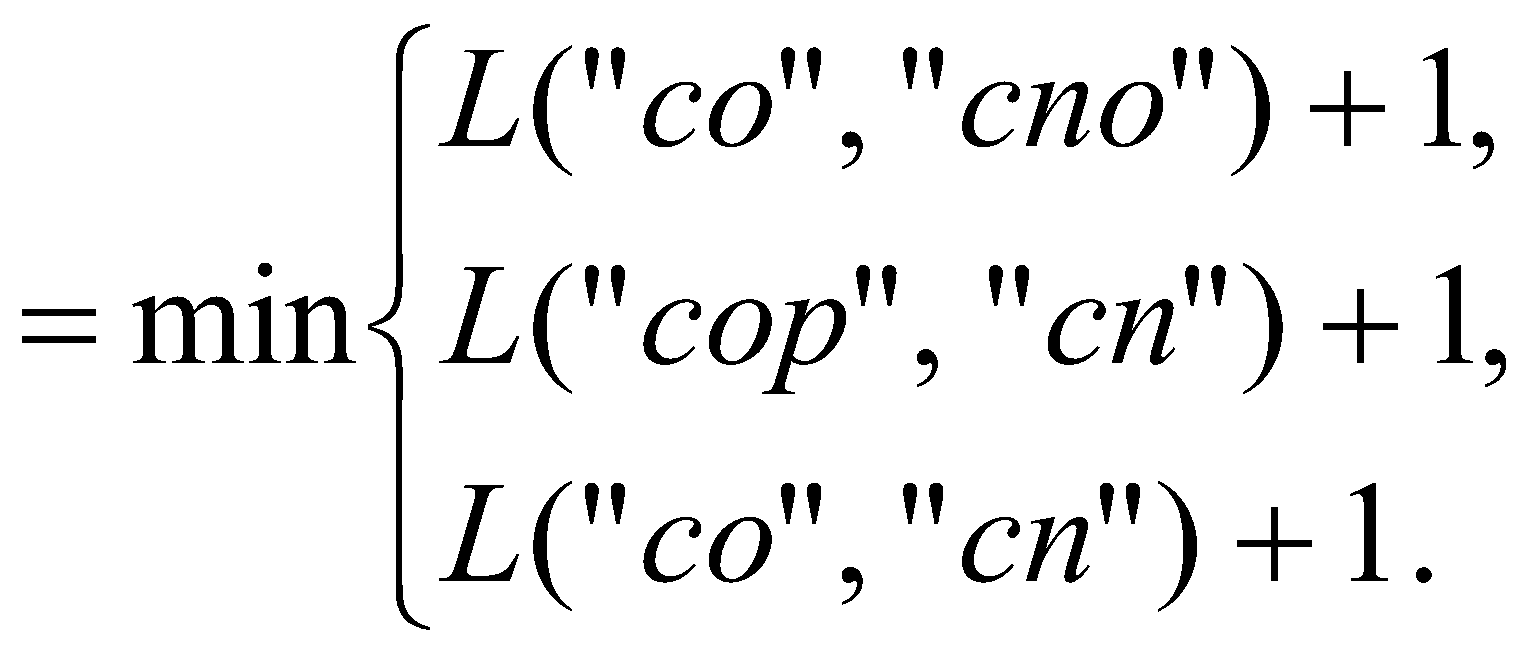
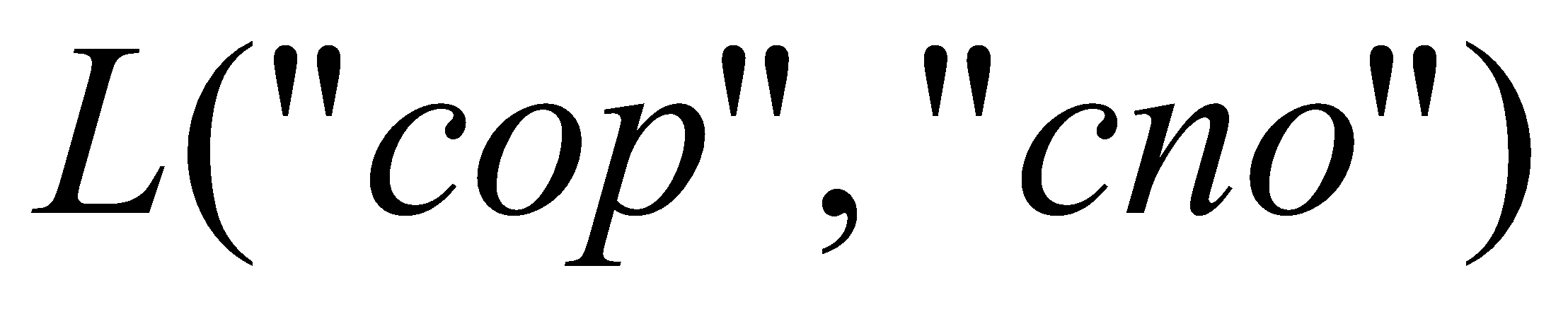
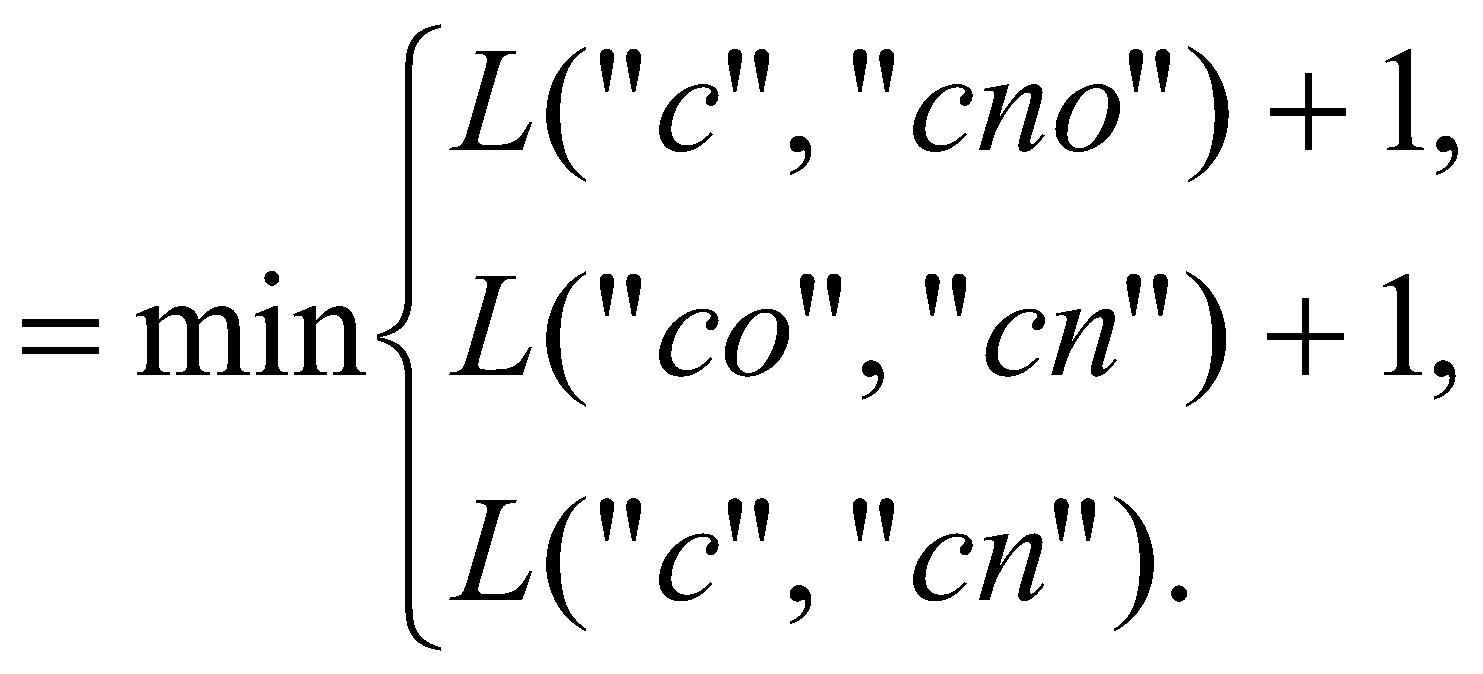
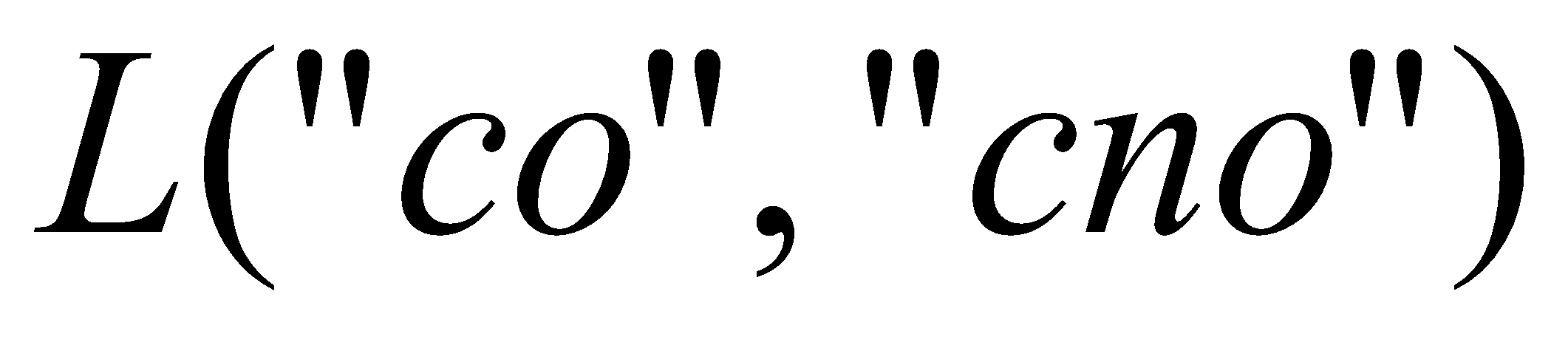
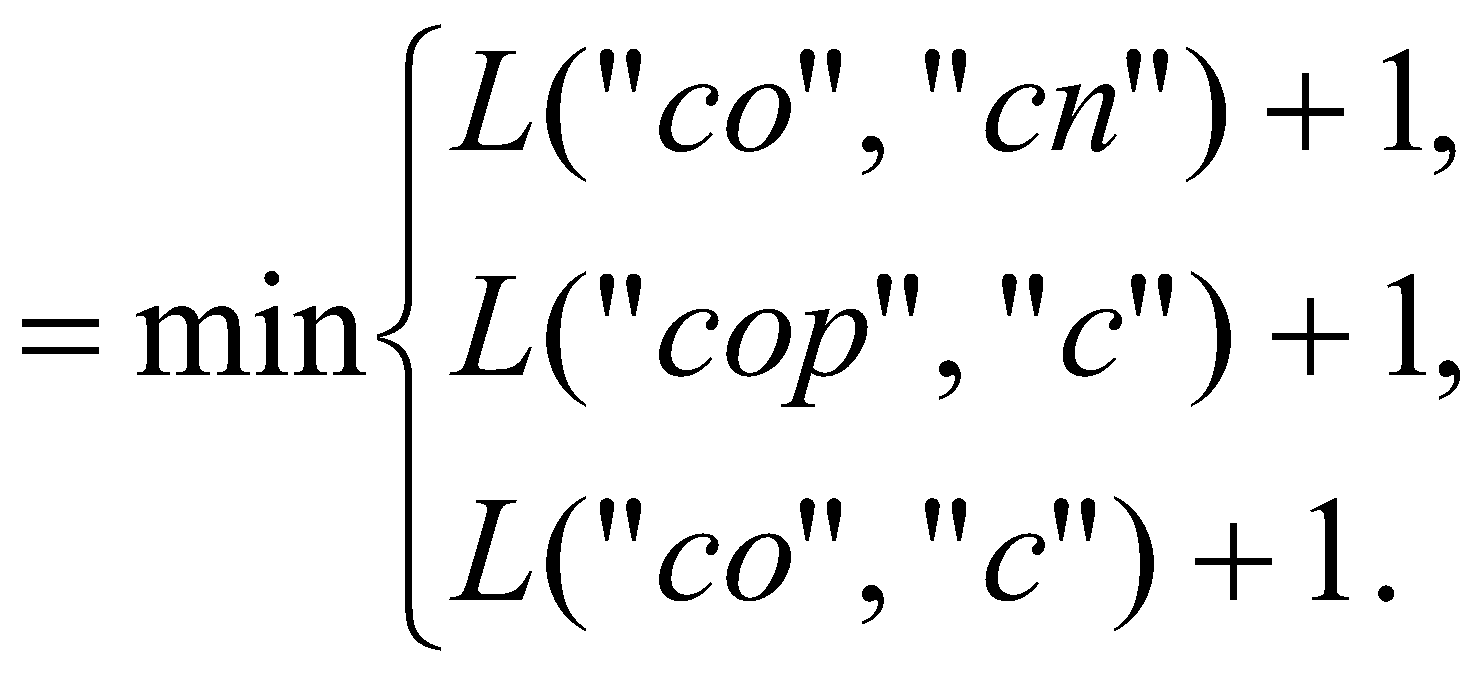
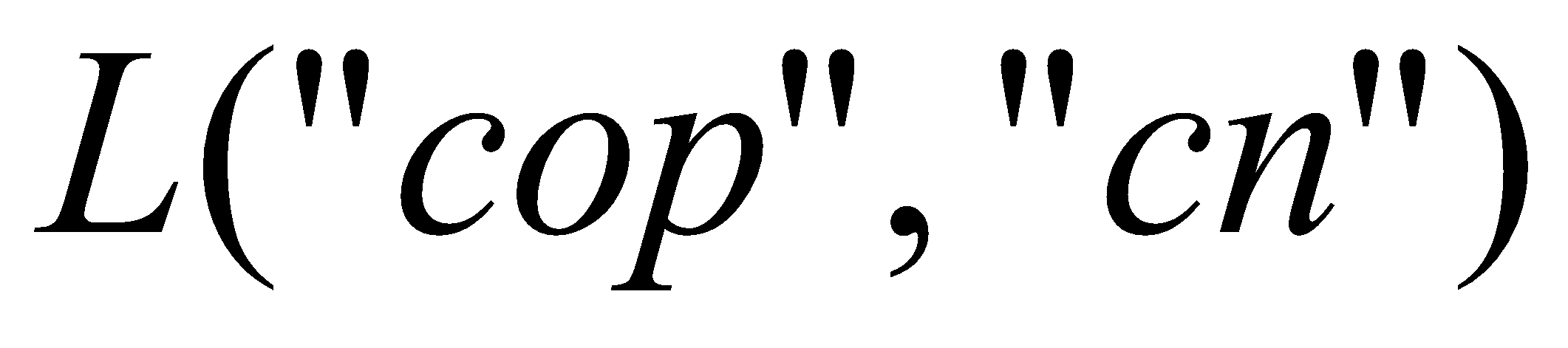
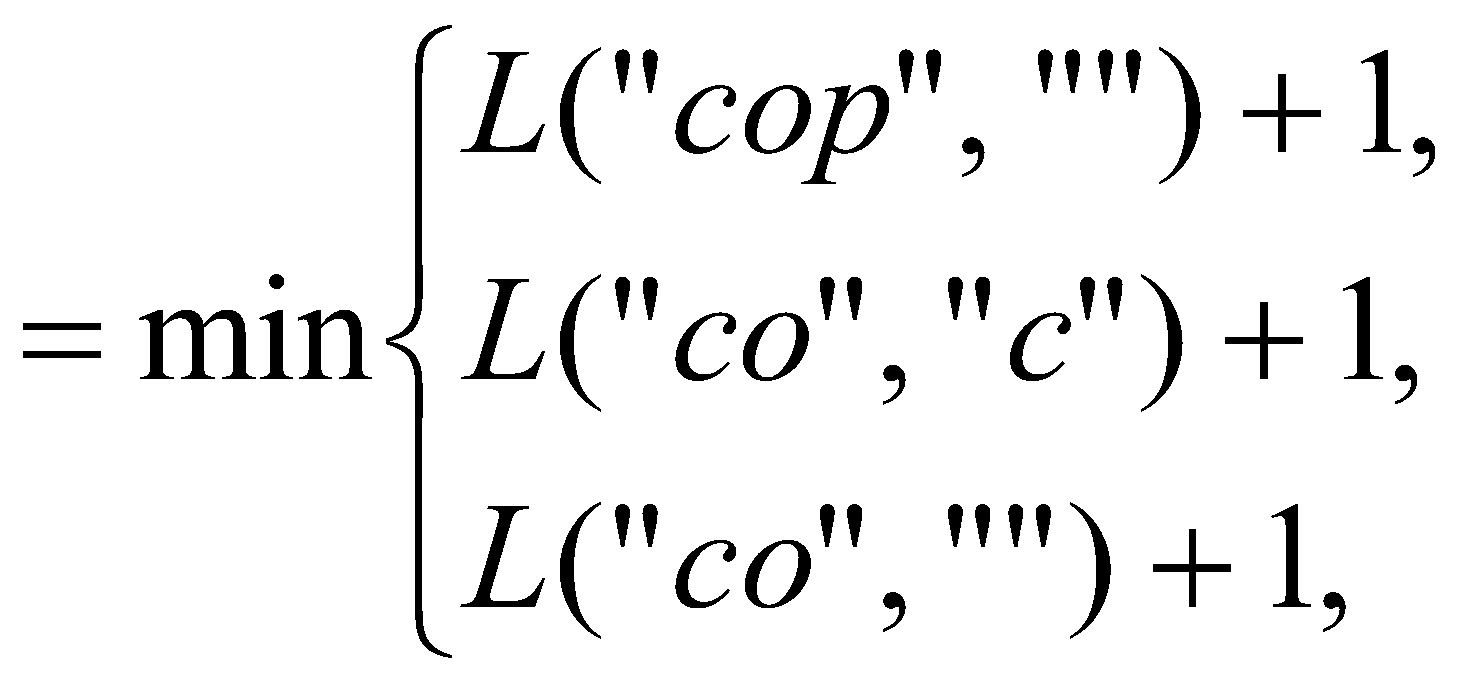
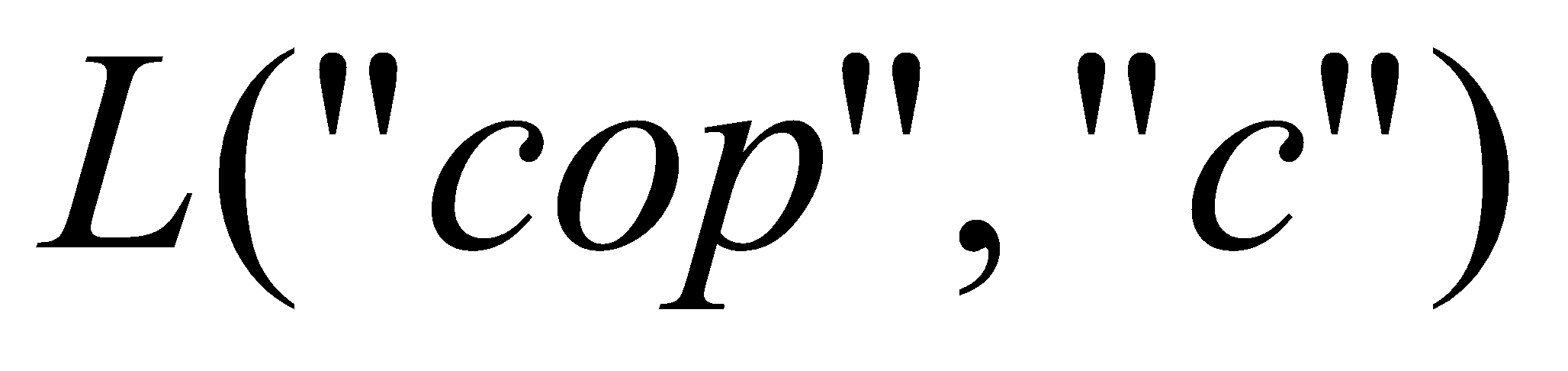
Пусть необходимо вычислить  Тогда имеем следующую последовательность шагов:

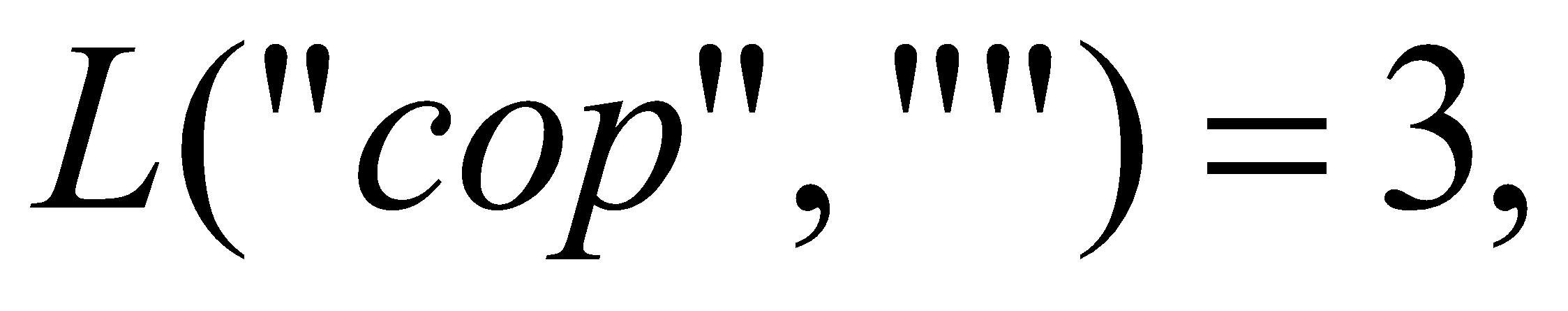
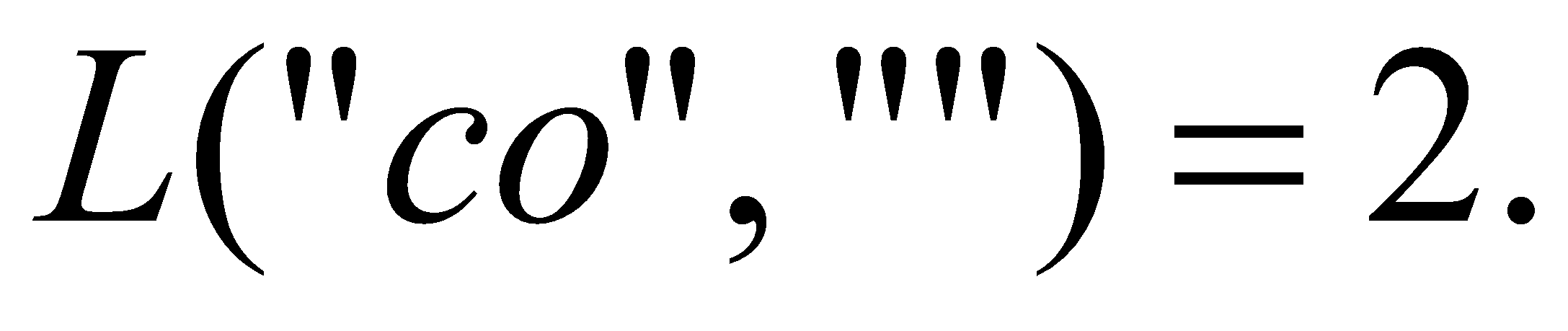
1.  
2. 
3. 
4. 
5. 

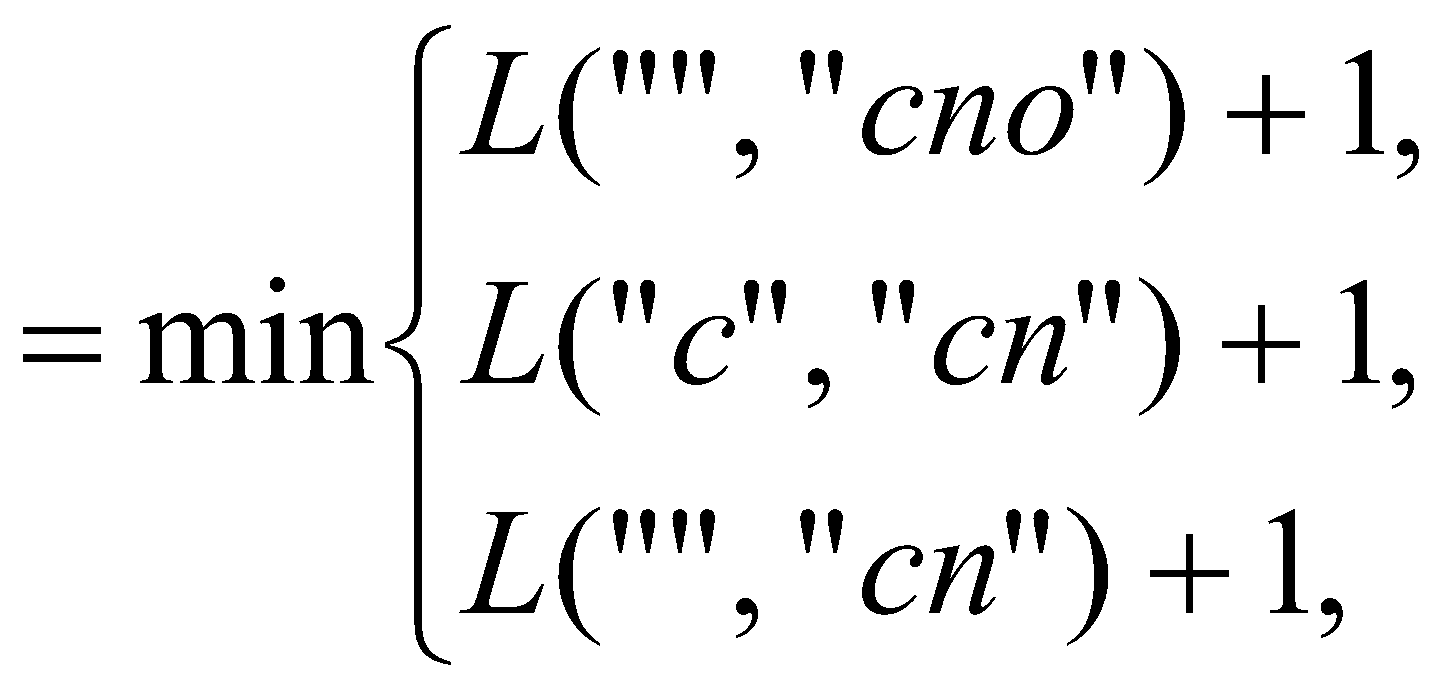
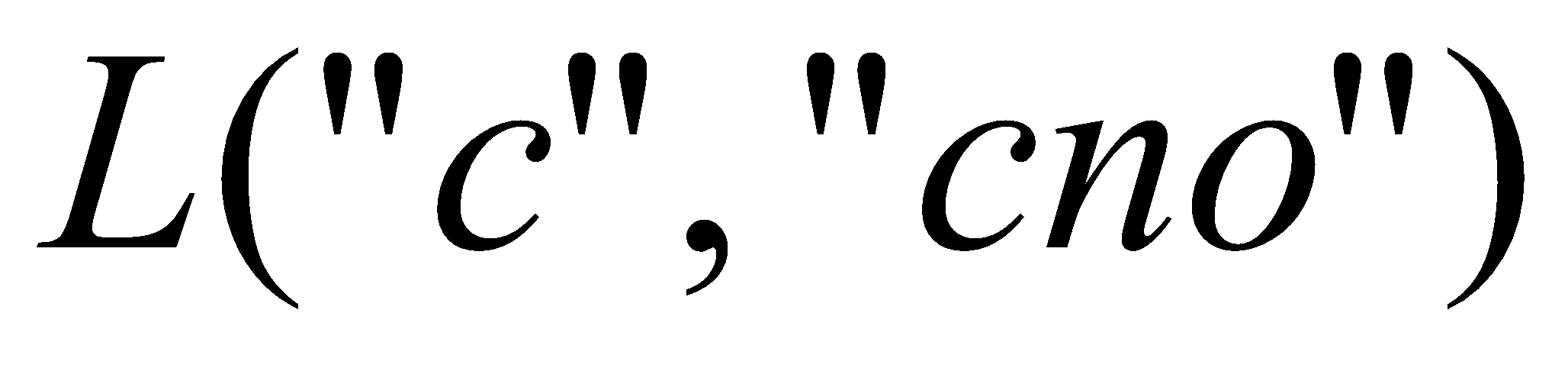
 

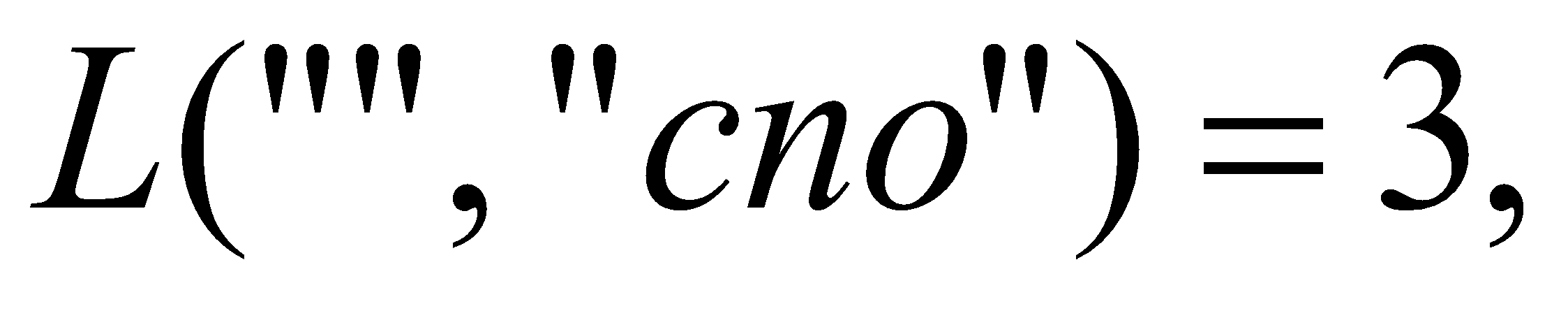
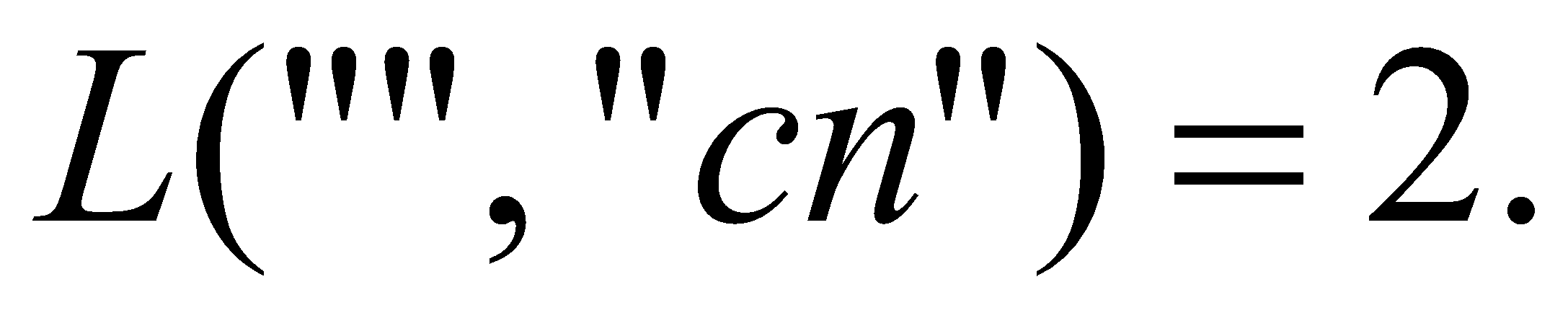
1. 

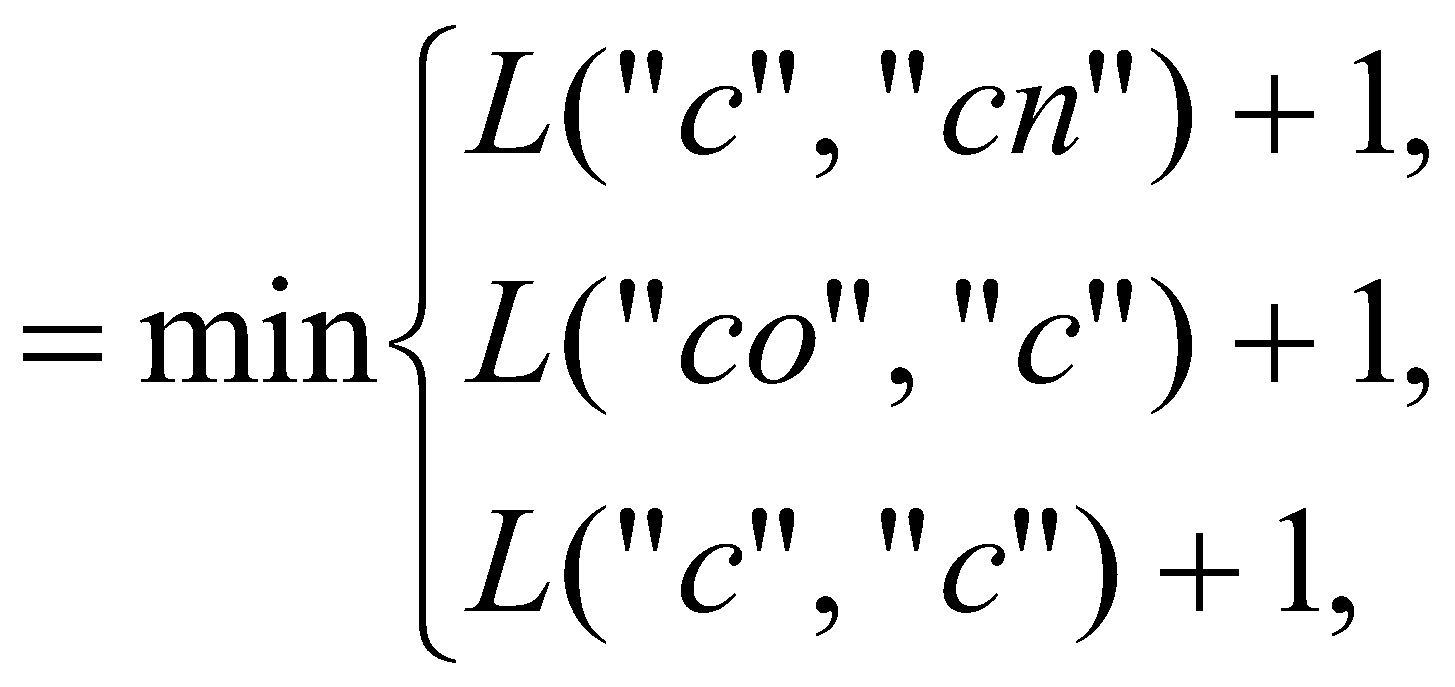
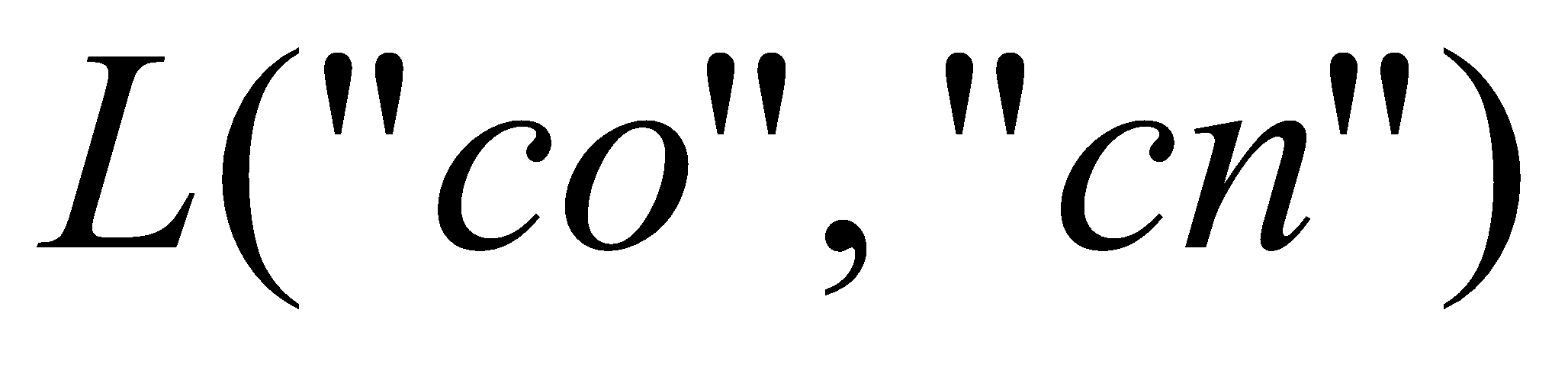
 

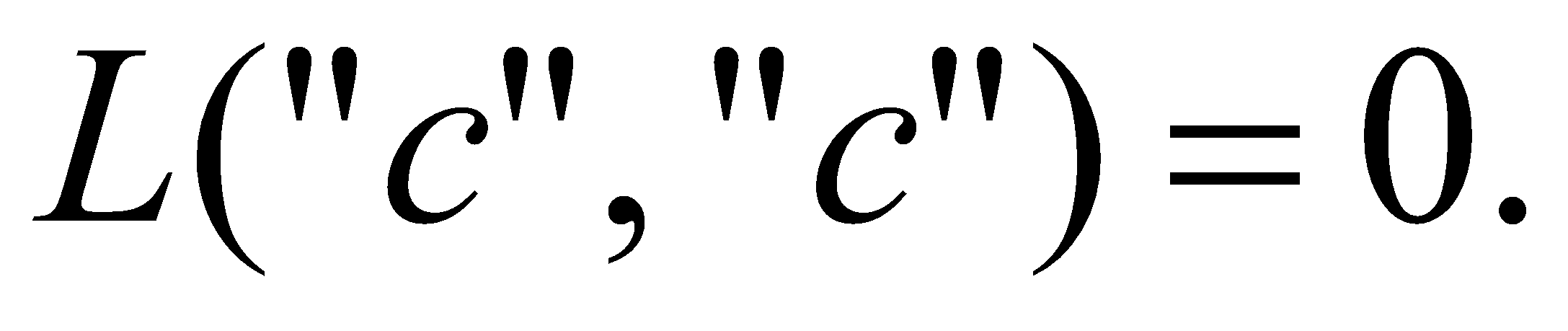
1. 
2. 
3. 
4. 

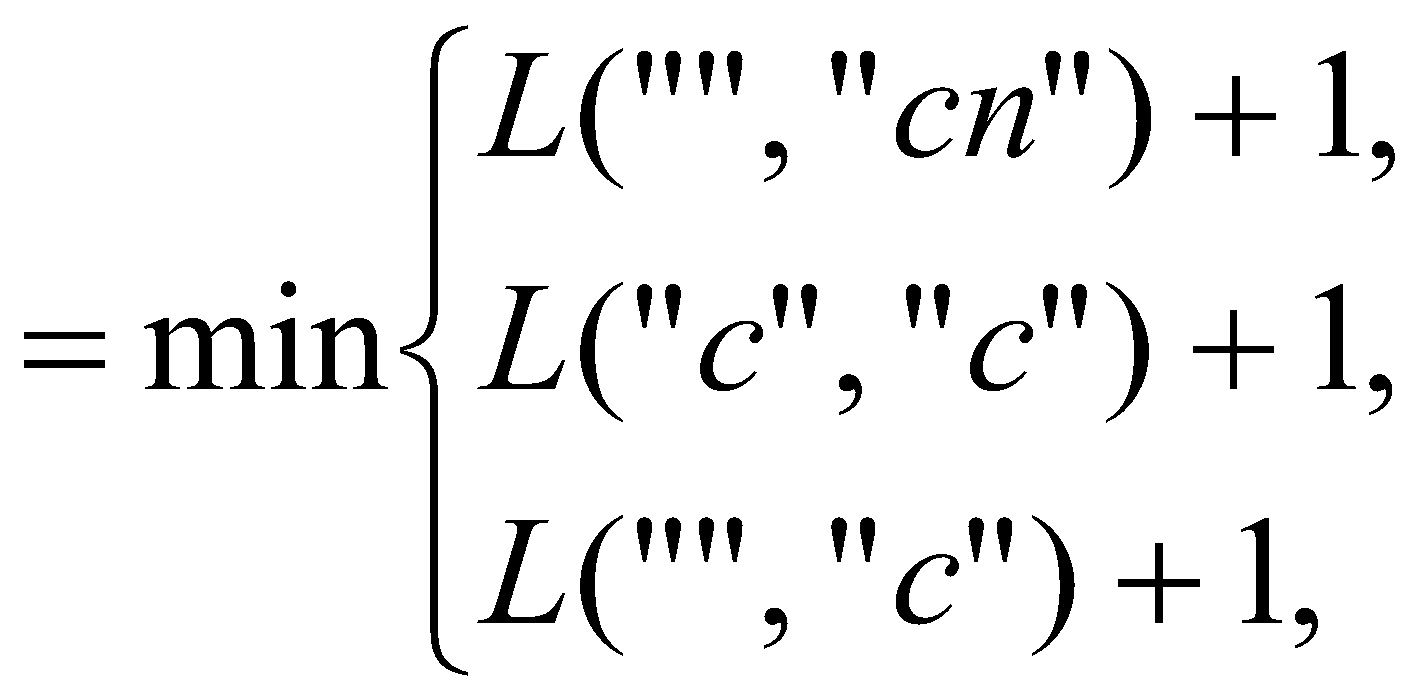
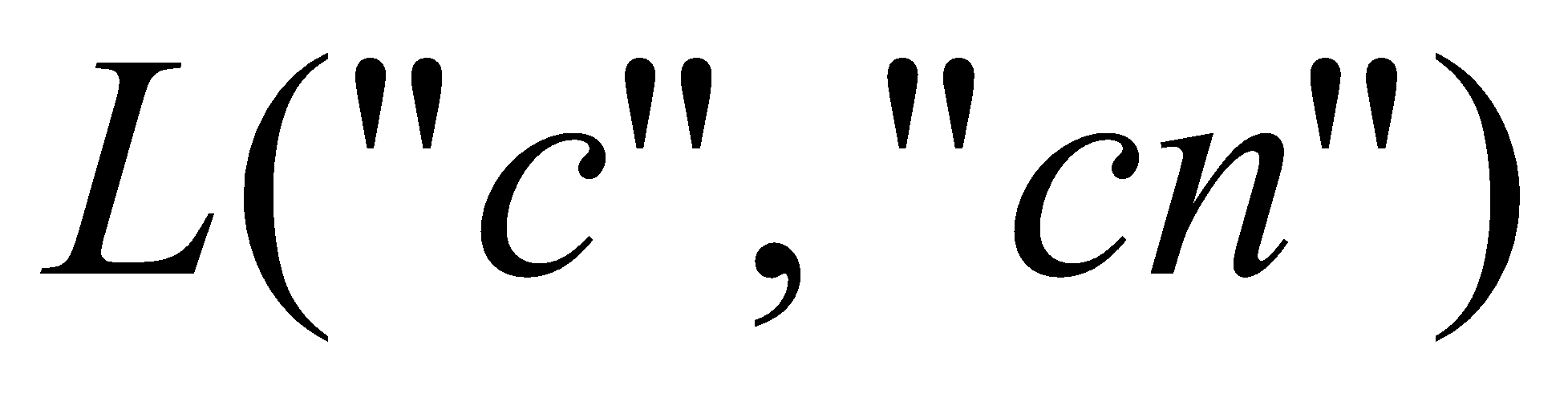
 

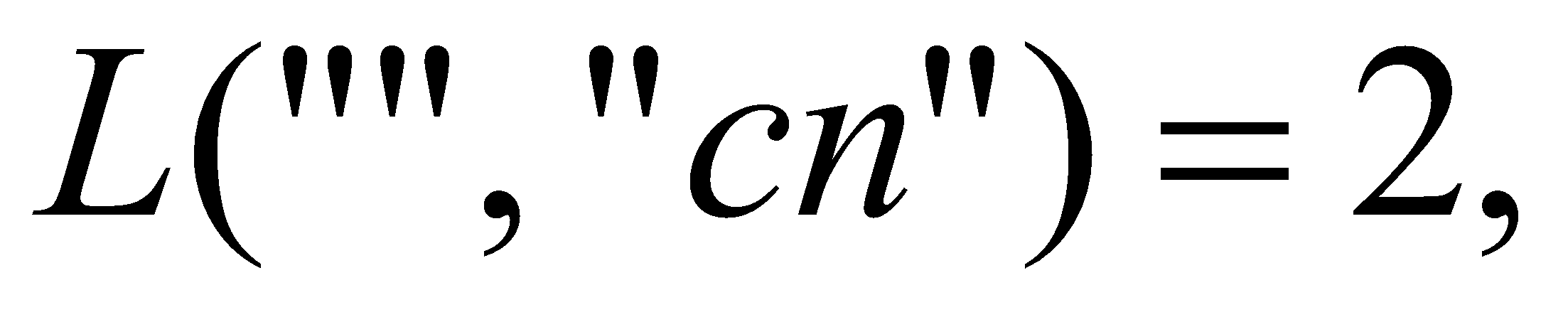
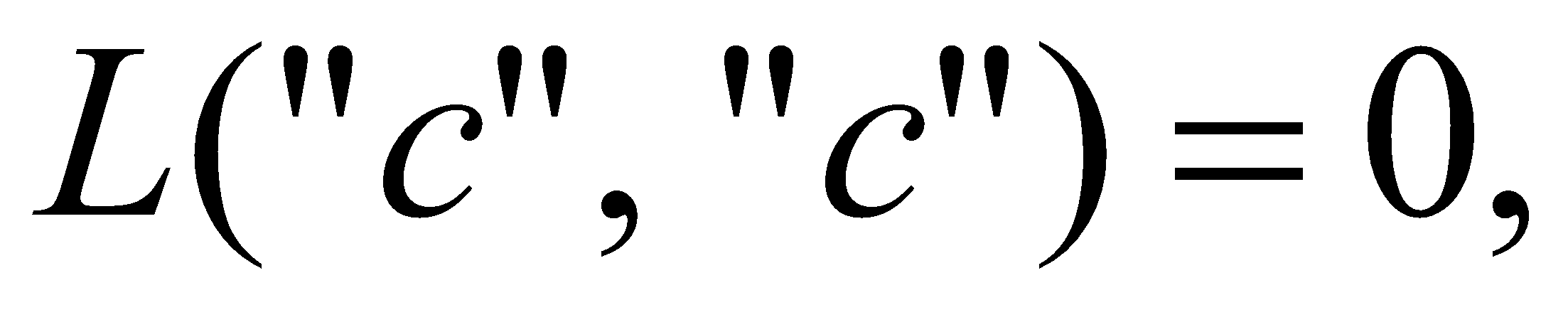
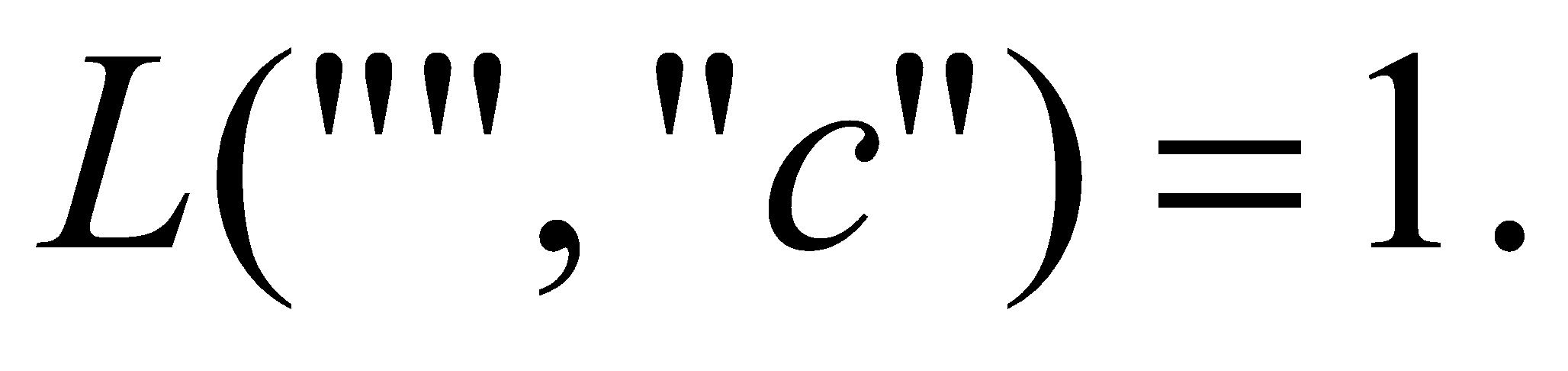
1. 

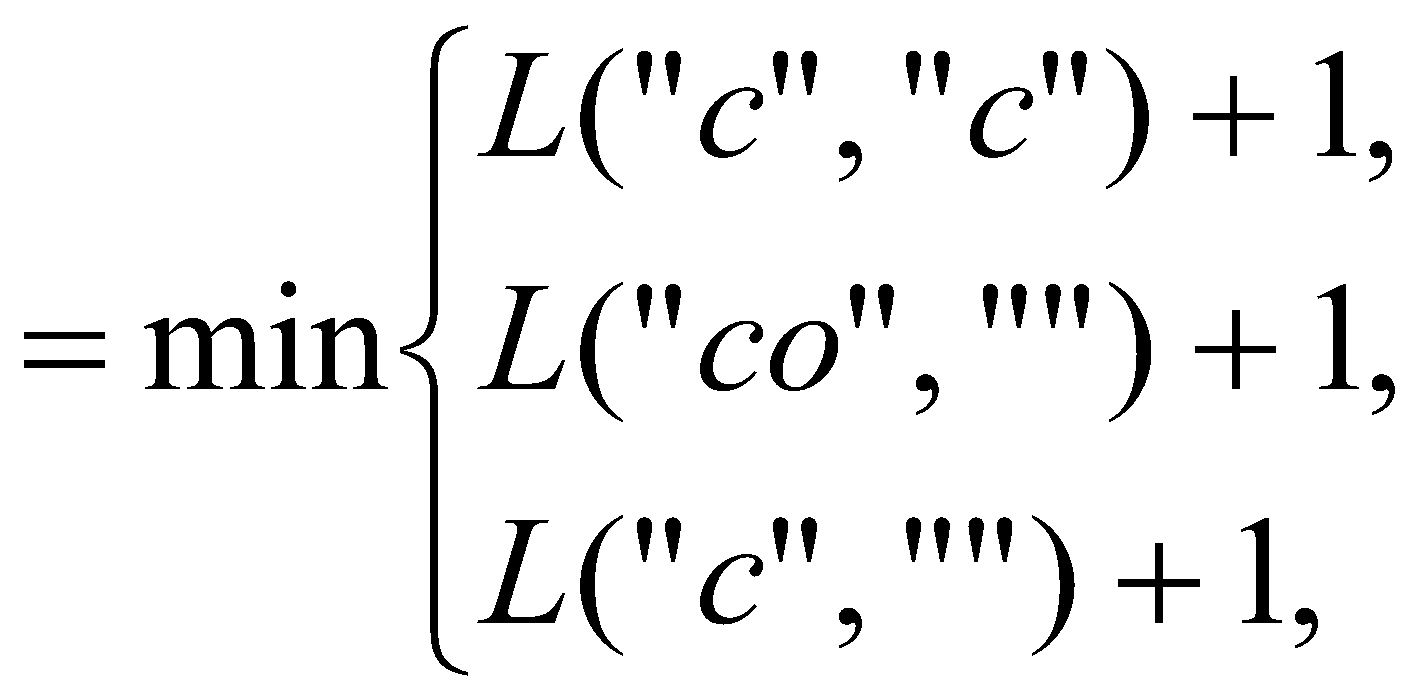
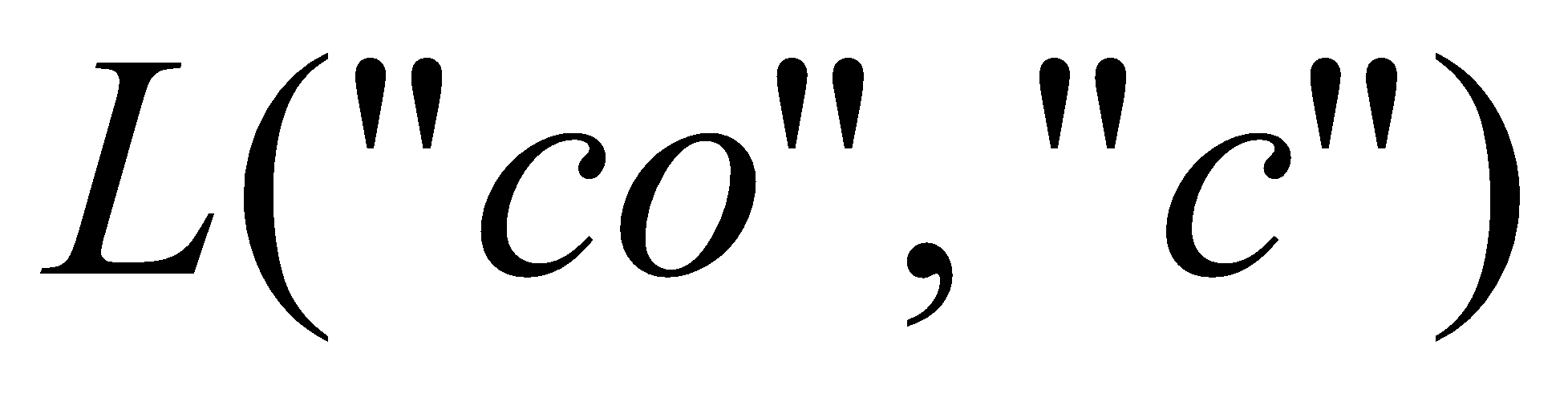
 

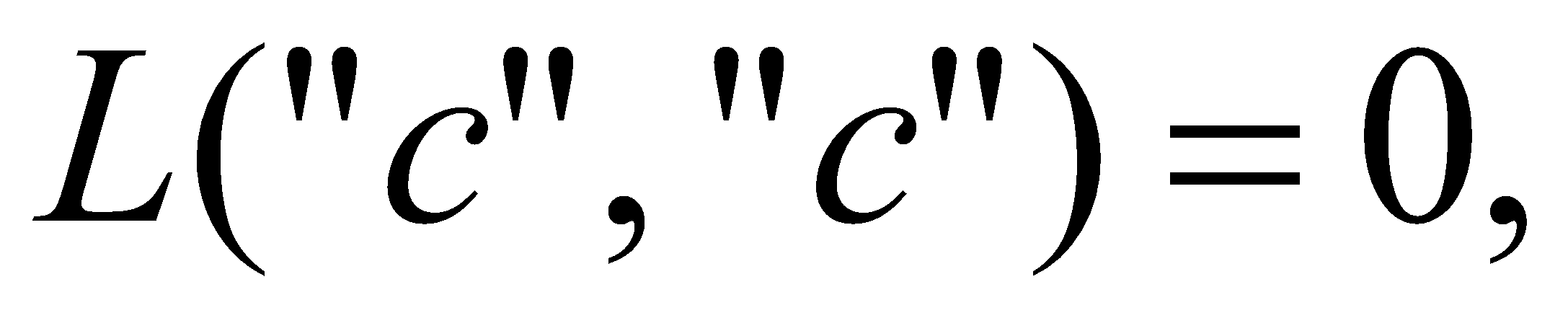
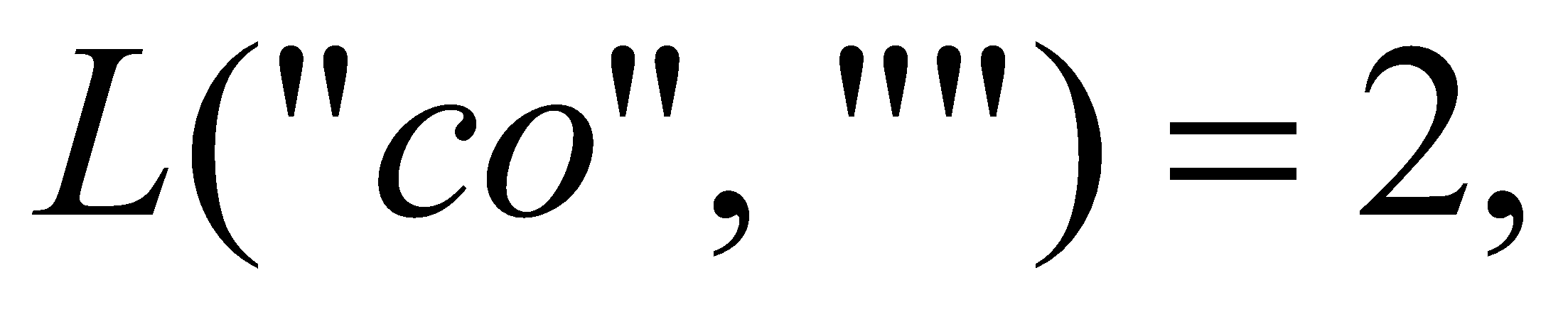
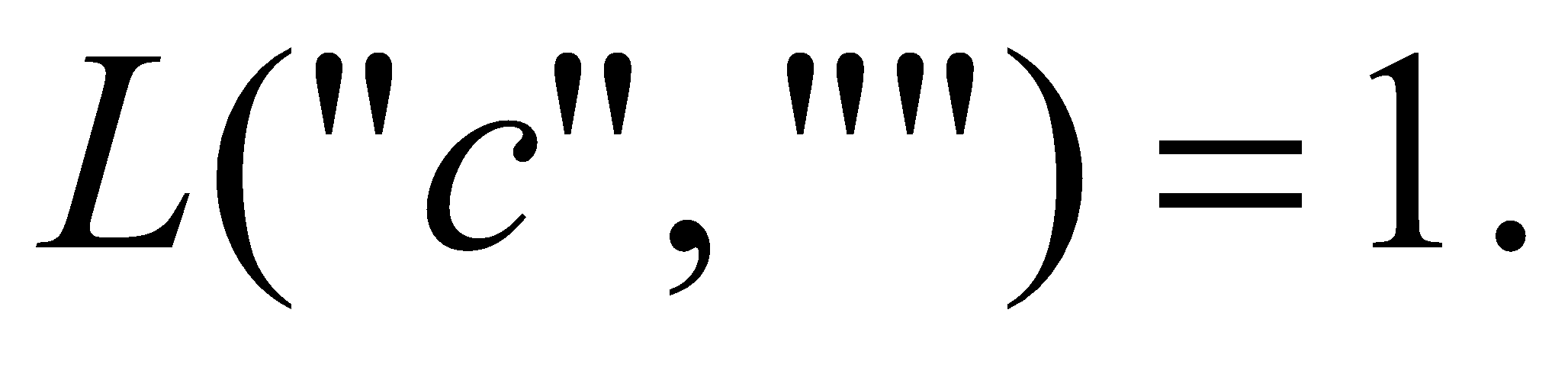
1. 

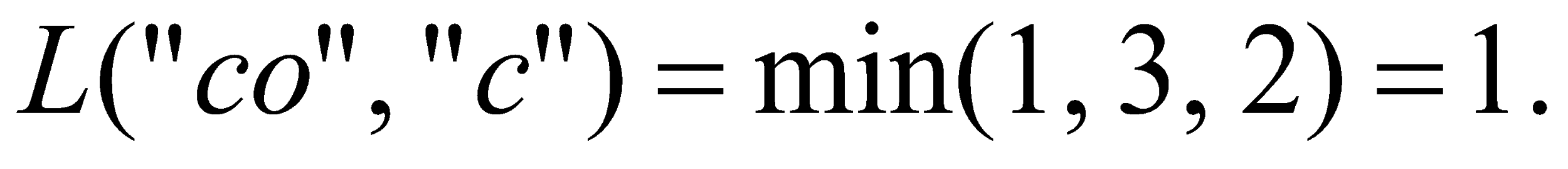
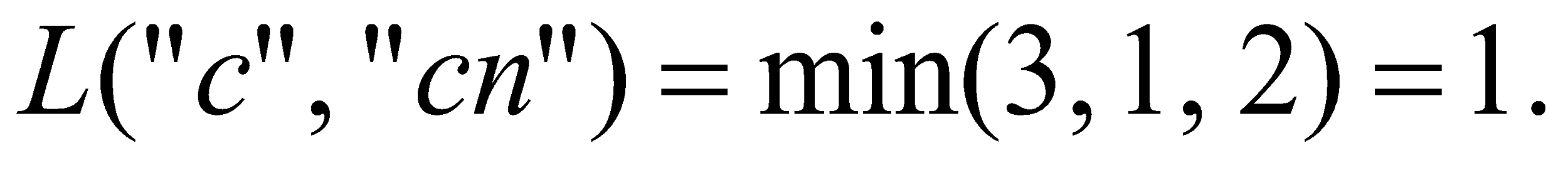
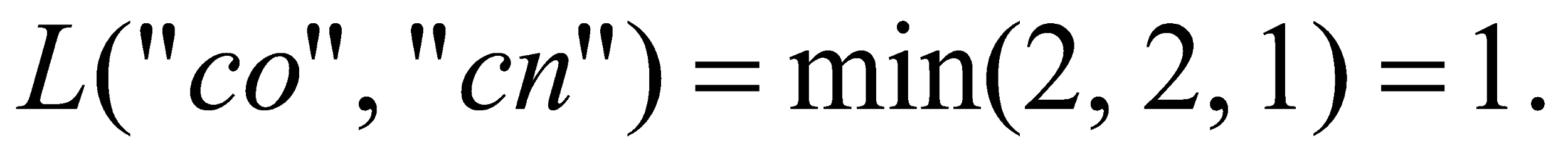
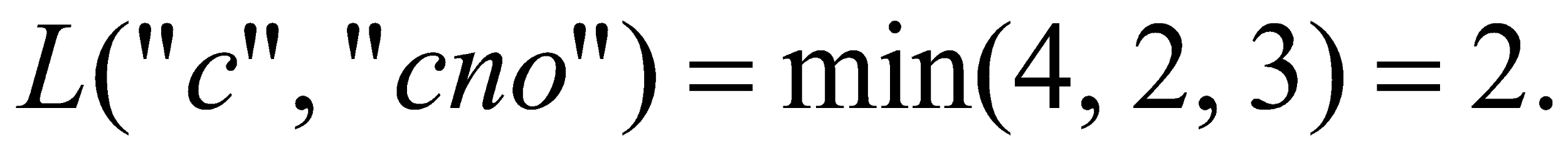
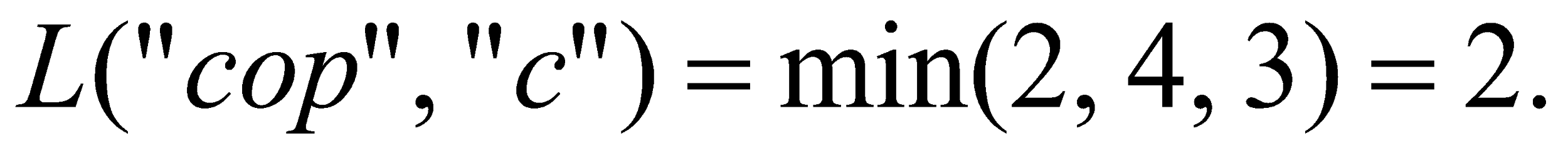
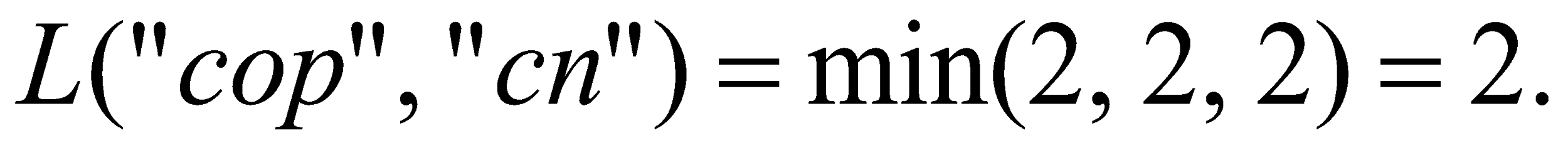
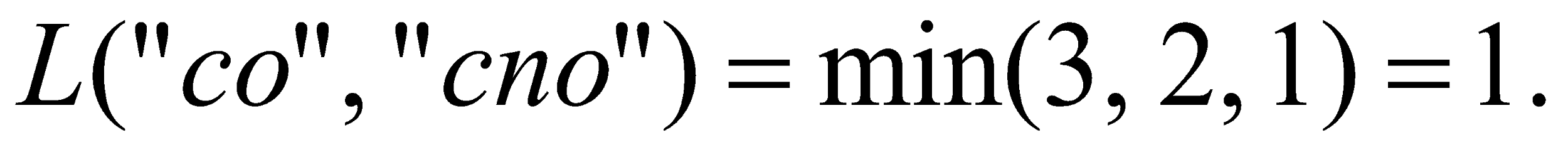
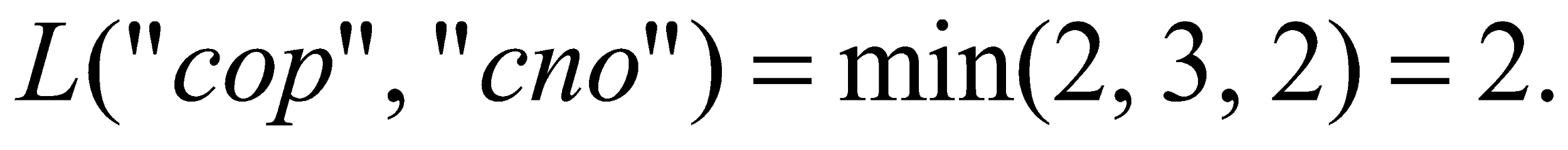
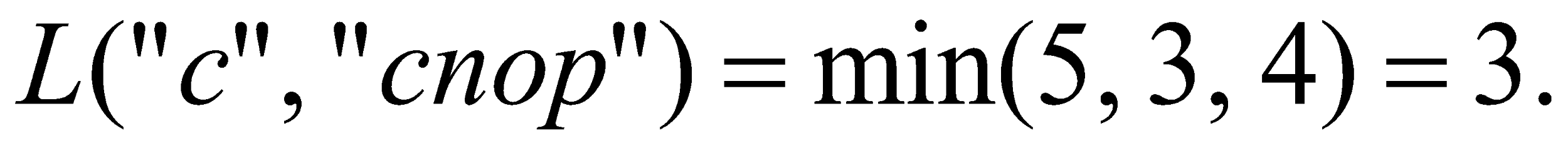
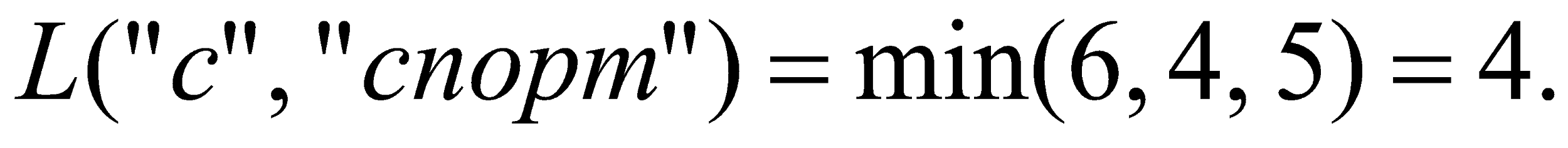
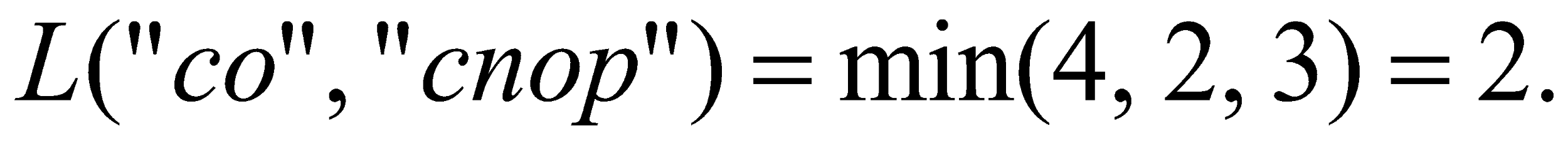
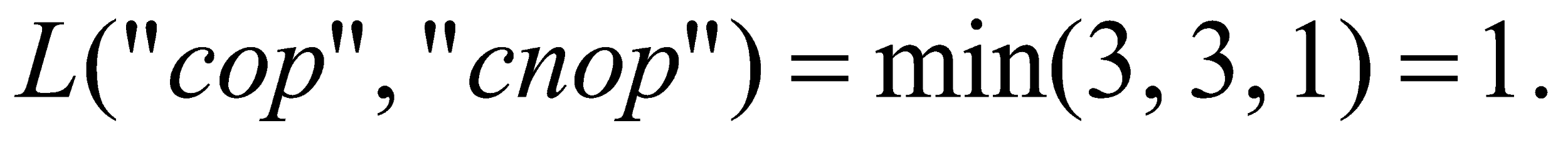
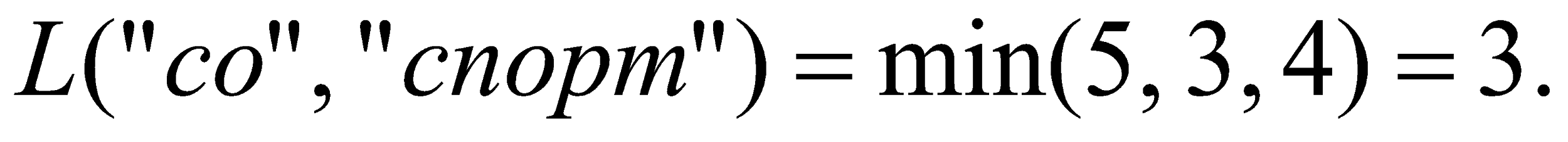
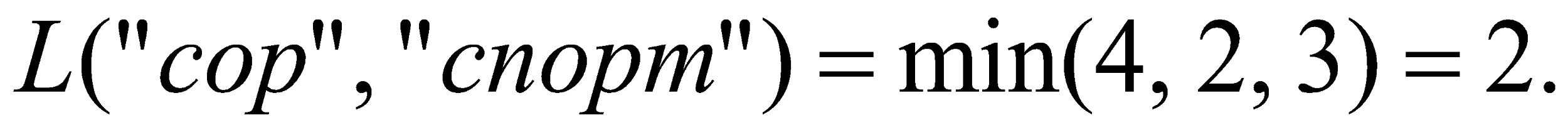


1. 

1. 

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 

// - Levenshtein.h

// -- дистанции Левенштeйна (динамическое программирование)

int levenshtein(

int lx, // длина слова x

const char x[], // слово длиной lx

int ly, // длина слова y

const char y[] // слово y

);

// -- дистанции Левенштeйна (рекурсия)

int levenshtein\_r(

int lx, // длина строки x

const char x[], // строка длиной lx

int ly, // длина строки y

const char y[] // строка y

);

// - Levenshtein.cpp

#include "stdafx.h"

#include <iomanip>

#include <algorithm>

#include "Levenshtein.h"

#define DD(i,j) d[(i)\*(ly+1)+(j)]

int min3(int x1, int x2, int x3)

{ return std::min(std::min(x1,x2),x3); }

int levenshtein(int lx, const char x[],int ly, const char y[])

{

int \*d = new int[(lx+1)\*(ly+1)];

for(int i = 0; i <= lx; i++) DD(i, 0) = i;

for(int j = 0; j <= ly; j++) DD(0, j) = j;

for (int i = 1; i <= lx; i++)

for (int j = 1; j <= ly; j++)

{

DD(i,j) = min3(DD(i-1, j) + 1, DD(i, j-1) + 1,

DD(i-1, j-1) + (x[i-1]==y[j-1]?0:1));

}

return DD(lx,ly);

}

int levenshtein\_r(

int lx, const char x[],

int ly, const char y[]

)

{

int rc = 0;

if (lx == 0) rc = ly;

else if (ly == 0) rc = lx;

else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;

else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;

else rc = min3(

levenshtein\_r(lx-1, x, ly, y)+1,

levenshtein\_r(lx, x, ly-1, y)+1,

levenshtein\_r(lx-1, x, ly-1, y)+(x[lx-1] == y[ly-1]?0:1)

);

return rc;

};

// --- main

// вычисление дистанции (расстояния) Левенштейна

#include "stdafx.h"

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include "Levenshtein.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

clock\_t t1 = 0, t2 = 0, t3,t4;

char x[] = "abcdefghklmnoxm", y[] = "xyabcdefghomnkm";

int lx = sizeof(x)-1, ly = sizeof(y)-1;

std::cout<<std::endl;

std::cout<<std::endl<< "-- расстояние Левенштейна -----"<< std::endl;

std::cout<<std::endl<< "--длина --- рекурсия -- дин.програм. ---"

<<std::endl;

for (int i = 8; i < std::min(lx,ly); i++)

{

t1 = clock();levenshtein\_r(i,x,i-2, y); t2 = clock();

t3 = clock();levenshtein(i,x,i-2, y); t4 = clock();

std::cout<<std::right<<std::setw(2)<<i-2<<"/"<<std::setw(2)<<i

<< " "<<std::left<<std::setw(10)<<(t2-t1)

<<" "<<std::setw(10)<<(t4-t3)<<std::endl;

}

system("pause");

return 0;

}

1. **Решение задачи о расстановке скобок при перемножении матриц**

// --- MultyMatrix.h

// расстановка скобок

#pragma once

// расстановка скобок при умножении матриц

// функции возвращают минимальное количество операций умножения

#define OPTIMALM\_PARM(x) ((int\*)x) // для представления 2мерного массива

int OptimalM( // рекурсия

int i, // [in] номер первой матрицы

int j, // [in] номер последней матрицы

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

int OptimalMD( // динамическое программирование

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

// --- MultiMatrix.cpp

// расстановка скобок (рекурсия)

#include "stdafx.h"

#include <memory.h>

#include "MultiMatrix.h"

#define INFINITY 0x7fffffff

#define NINFINITY 0x80000000

int OptimalM(int i, int j, int n, const int c[], int \*s)

{

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

int o =INFINITY, bo = INFINITY;

if (i<j)

{

for (int k = i; k<j;k++)

{

bo = OptimalM(i,k, n, c, s)+

OptimalM(k+1,j,n, c, s)+ c[i- 1]\*c[k]\*c[j];

if (bo < o)

{

o = bo;

OPTIMALM\_S(i,j) = k;

}

}

}

else o = 0;

return o;

#undef OPTIMALM\_S

};

// --- MultyMatrix.cpp (продолжение)

// расстановка скобок (динамическое программирование)

int OptimalMD(int n, const int c[], int\* s)

{

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

#define OPTIMALM\_M(x1,x2) (M[(x1-1)\*n+x2-1])

int \*M = new int[n\*n], j = 0, q = 0;

for (int i = 1; i <= n; i++) OPTIMALM\_M(i,i) = 0;

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

{

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

{

j = i+l-1;

OPTIMALM\_M(i,j) = INFINITY;

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

{

q = OPTIMALM\_M(i,k) + OPTIMALM\_M(k+1,j)+c[i-1]\*c[k]\*c[j];

if (q < OPTIMALM\_M(i,j))

{

OPTIMALM\_M(i,j) = q; OPTIMALM\_S(i,j)= k;

}

}

}

}

return OPTIMALM\_M(1,n);

#undef OPTIMALM\_M

#undef OPTIMALM\_S

};

// --- main

// расстановка скобок

#include "stdafx.h"

#include <cmath>

#include <memory.h>

#include <iostream>

#include "MultiMatrix.h" // умножение матриц

#define N 6

int main()

{

int Mc[N+1] = {5,10,15,20,25,30,35}, Ms[N][N], r = 0, rd = 0;

memset(Ms,0,sizeof(int)\*N\*N);

r = OptimalM(1, N, N, Mc, OPTIMALM\_PARM(Ms));

setlocale(LC\_ALL, "rus");

std::cout<<std::endl;

std::cout<<std::endl<< "-- расстановка скобок (рекурсивное решение) "

<< std::endl;

std::cout<<std::endl<< "размерности матриц: ";

for (int i = 1; i <= N; i++) std::cout<<"("<<Mc[i-1]<<","<<Mc[i]<<") ";

std::cout<<std::endl<< "минимальное количество операций умножения: " << r;

std::cout<<std::endl<<std::endl<<"матрица S"<<std::endl;

for (int i = 0; i < N; i++)

{

std::cout<<std::endl;

for (int j = 0; j < N; j++) std::cout<<Ms[i][j]<< " " ;

}

std::cout<<std::endl;

memset(Ms,0,sizeof(int)\*N\*N);

rd = OptimalMD(N, Mc, OPTIMALM\_PARM(Ms));

std::cout<<std::endl

<< "-- расстановка скобок (динамичеое программирование) "<< std::endl;

std::cout<<std::endl<< "размерности матриц: ";

for (int i = 1; i <= N; i++)

std::cout<<"("<<Mc[i-1]<<","<<Mc[i]<<") ";

std::cout<<std::endl<< "минимальное количество операций умножения: "

<< rd;

std::cout<<std::endl<<std::endl<<"матрица S"<<std::endl;

for (int i = 0; i < N; i++)

{

std::cout<<std::endl;

for (int j = 0; j < N; j++) std::cout<<Ms[i][j]<< " " ;

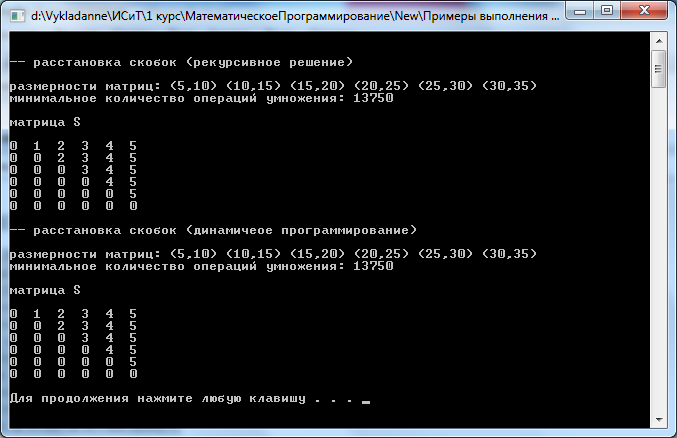
}

std::cout<<std::endl<<std::endl;

system("pause");

return 0;

}



**ОБЪЯСНИТЬ, КАК РАССТАВЛЯЮТСЯ СКОБКИ ПО ДАННЫМ МАТРИЦАМ!!!**

1. **Решение задачи вычисления длины наибольшей общей подпоследовательности**

// - LCS.h

// -- рекурсивное вычисление длины LCS

int lcs (

int lenx, // длина последовательности X

const char x[], // последовательность X

int leny, // длина последовательности Y

const char y[] // последовательность Y

);

// - LCS.cpp

// -- рекурсивное вычисление длины LCS

#include "stdafx.h"

#include <algorithm>

#include "LCS.h"

int lcs (int lenx, const char x[],

int leny, const char y[])

{

int rc = 0;

if (lenx > 0 && leny > 0)

{

if (x[lenx-1] == y[leny-1]) rc = 1 + lcs(lenx-1, x,leny-1, y);

else rc = std::max(lcs(lenx, x,leny-1, y), lcs(lenx-1, x,leny, y));

}

return rc; //длина LCS

}

// - main

// -- вычисления длины LCS

#include "stdafx.h"

#include <iostream>

#include "LCS.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

char X[]="ALDC", Y[]="LADCM";

std::cout<<std::endl<<"-- вычисление длины LCS для X и Y(рекурсия)";

std::cout<<std::endl<<"-- последовательность X: "<< X;

std::cout<<std::endl<<"-- последовательность Y: "<< Y;

int s = lcs(

sizeof(X)-1, // длина последовательности X

"ALDC", // последовательность X

sizeof(Y)-1, // длина последовательности Y

"LADCM" // последовательность Y

);

std::cout<<std::endl<< "-- длина LCS: "<<s<<std::endl;

system("pause");

return 0;

}

**Динамическое программирование**

//- LCH.h

int lcsd(

const char x[], // последовательность X

const char y[], // последовательность Y

char z[] // наибольшая общая подпоследовательность

);

//- LCS.cpp

#include "stdafx.h"

#include <cstring>

#include "LCS.h"

#define LCS\_C(x1,x2) (C[(x1)\*(leny+1)+(x2)])

#define LCS\_B(x1,x2) (B[(x1)\*(leny+1)+(x2)])

#define LCS\_X(i) (x[(i)-1])

#define LCS\_Y(i) (y[(i)-1])

#define LCS\_Z(i) (z[(i)-1])

enum Dart{TOP,LEFT,LEFTTOP};

void getLCScontent( int lenx, int leny, const char x[],

const Dart\* B,

int n, int i, int j, char z[])

{

if ((i > 0 && j > 0 && n > 0 ))

{

if (LCS\_B(i,j) == LEFTTOP)

{

getLCScontent(lenx, leny,x, B, n-1, i-1, j-1, z);

LCS\_Z(n) = LCS\_X(i);

LCS\_Z(n+1) = 0;

}

else if (LCS\_B(i,j)== TOP)

getLCScontent(lenx, leny,x, B, n, i-1, j, z);

else getLCScontent(lenx, leny,x, B, n, i, j-1, z);

}

};

int lcsd(const char x[], const char y[], char z[])

{

int n;

int lenx = strlen(x), leny = strlen(x),

\*C = new int[(lenx+1)\*(leny+1)];

Dart\* B = new Dart[(lenx+1)\*(leny+1)];

memset(C,0,sizeof(int)\*(lenx+1)\*(leny+1));

for (int i = 1; i <= lenx; i++)

for(int j = 1; j <= leny; j++)

if (LCS\_X(i) == LCS\_Y(j))

{LCS\_C(i,j) = LCS\_C(i-1,j-1)+1;

LCS\_B(i,j) = LEFTTOP;}

else if (LCS\_C(i-1,j) >= LCS\_C(i, j-1))

{

LCS\_C(i,j) = LCS\_C(i-1, j);

LCS\_B(i,j) = TOP;

}

else

{

LCS\_C(i,j) = LCS\_C(i, j-1);

LCS\_B(i,j) = LEFT;

}

getLCScontent(lenx, leny, x, B, LCS\_C(lenx,leny), lenx, leny, z);

return LCS\_C(lenx,leny);

}

#undef LCS\_Z

#undef LCS\_C

#undef LCS\_B

#undef LCS\_X

#undef LCS\_Y

// --- main

// наибольшая общая подпоследовательность

#include "stdafx.h"

#include <iostream>

#include "LCS.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

char z[100]="";

char x[] = "ABCBDAB",

y[] = "BDCABA" ;

int l = lcsd(x, y, z);

std::cout<<std::endl

<< "-- наибольшая общая подпоследовательость - LCS(динамическое"

<<"программирование)"<< std::endl;

std::cout<<std::endl<<"последовательость X: " << x;

std::cout<<std::endl<<"последовательость Y: " << x;

std::cout<<std::endl<<" LCS: " << z;

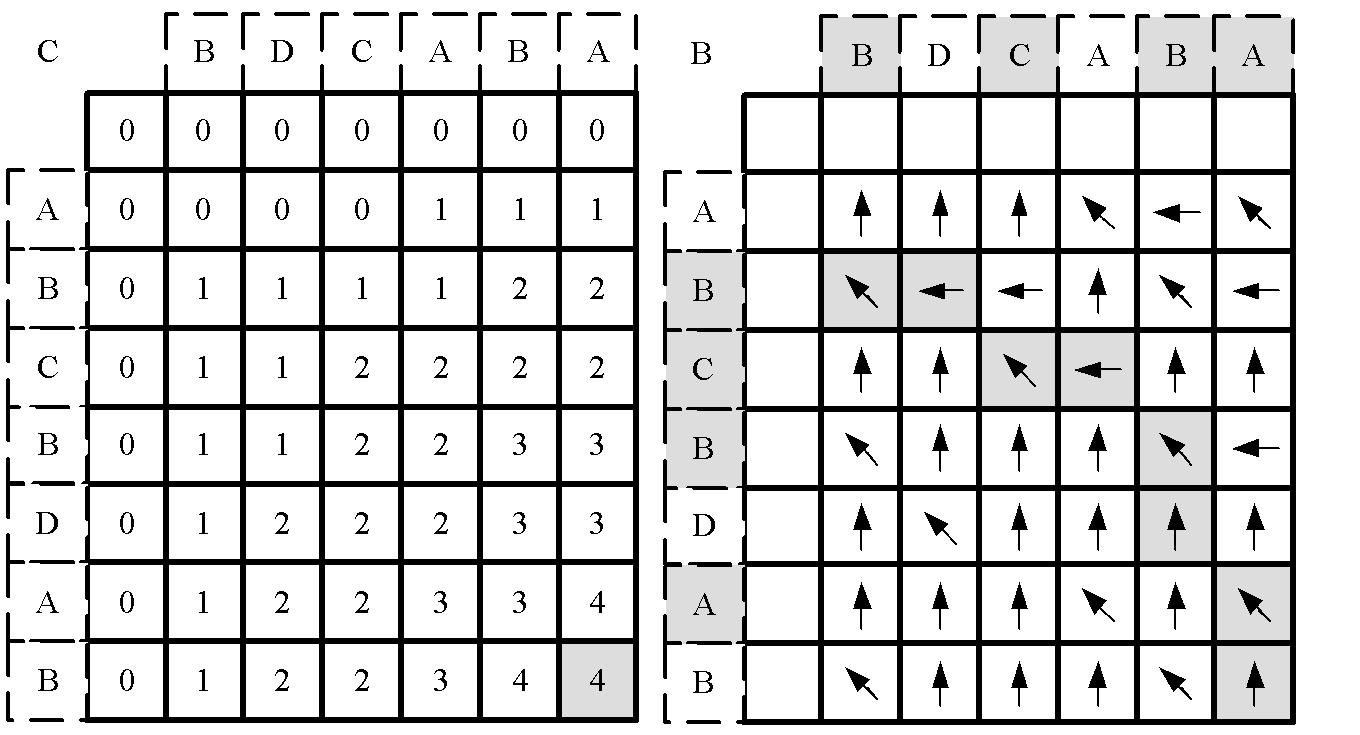
std::cout<<std::endl<<" длина LCS: " << l;

std::cout<<std::endl;

system("pause");

return 0;

}



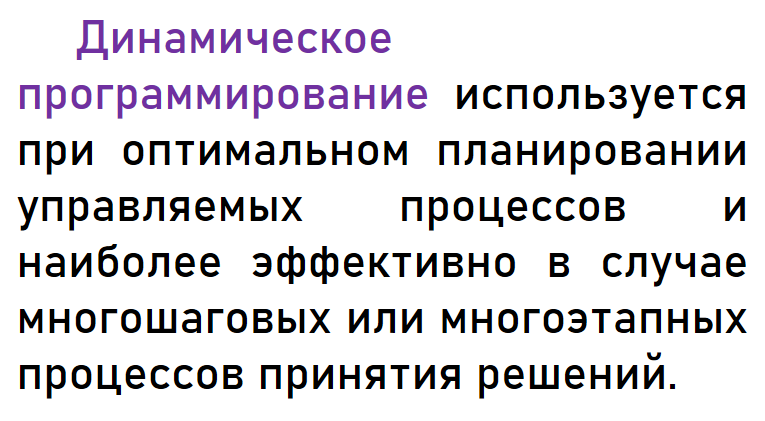
**ОБЪЯСНИТЬ, КАК ЗАПОЛНЯЮТСЯ МАТРИЦЫ И ЧТО ОНИ ОЗНАЧАЮТ!**

**Вопросы для защиты:**

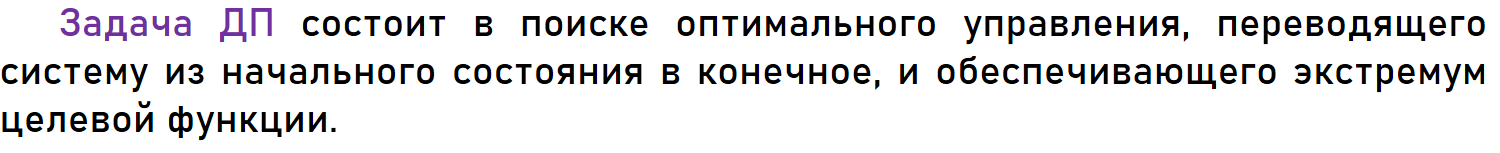
* + - 1. В каких областях используется динамическое программирование?
      2. В чем заключается задача динамического программирования?
      3. Чем аддитивная функция отличается от мультипликативной?
      4. Каков принцип оптимальности Беллмана?
      5. Что такое рекурсивный алгоритм?
      6. Что такое рекурсивная функция?
      7. Что такое системный стек?
      8. Объясните понятие «глубина рекурсии»
      9. Поясните своими словами схему решения задачи по принципу «разделяй и властвуй»
      10. Что такое редакционное расстояние?
      11. Что такое подпоследовательность и как её можно получить из последовательности?

**Вопросы для защиты:**

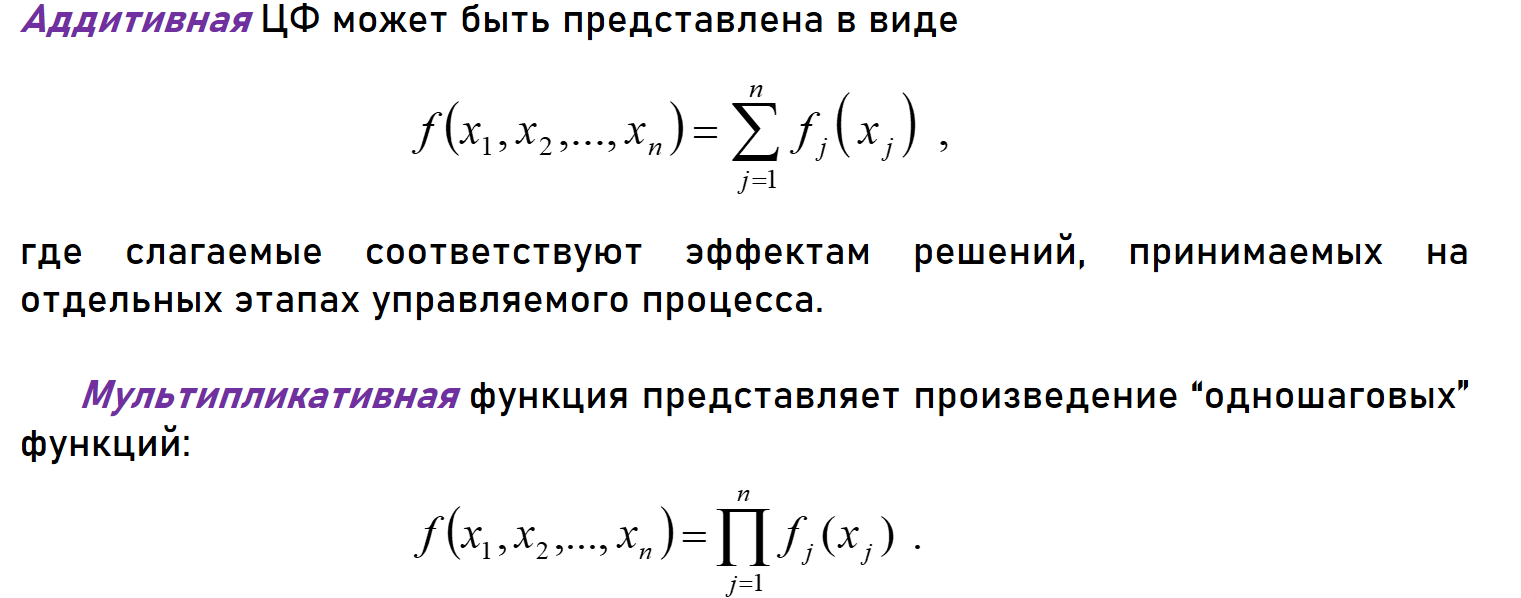
В каких областях используется динамическое программирование?



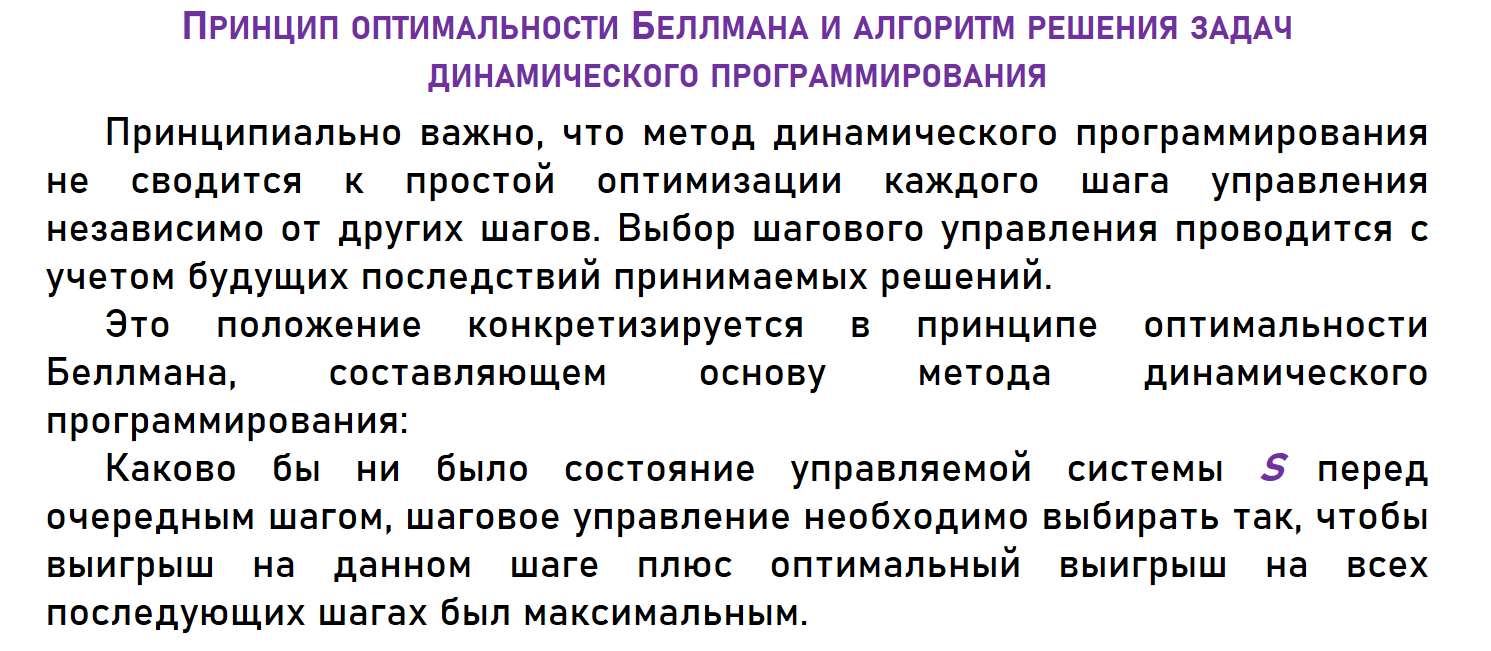
В чем заключается задача динамического программирования?

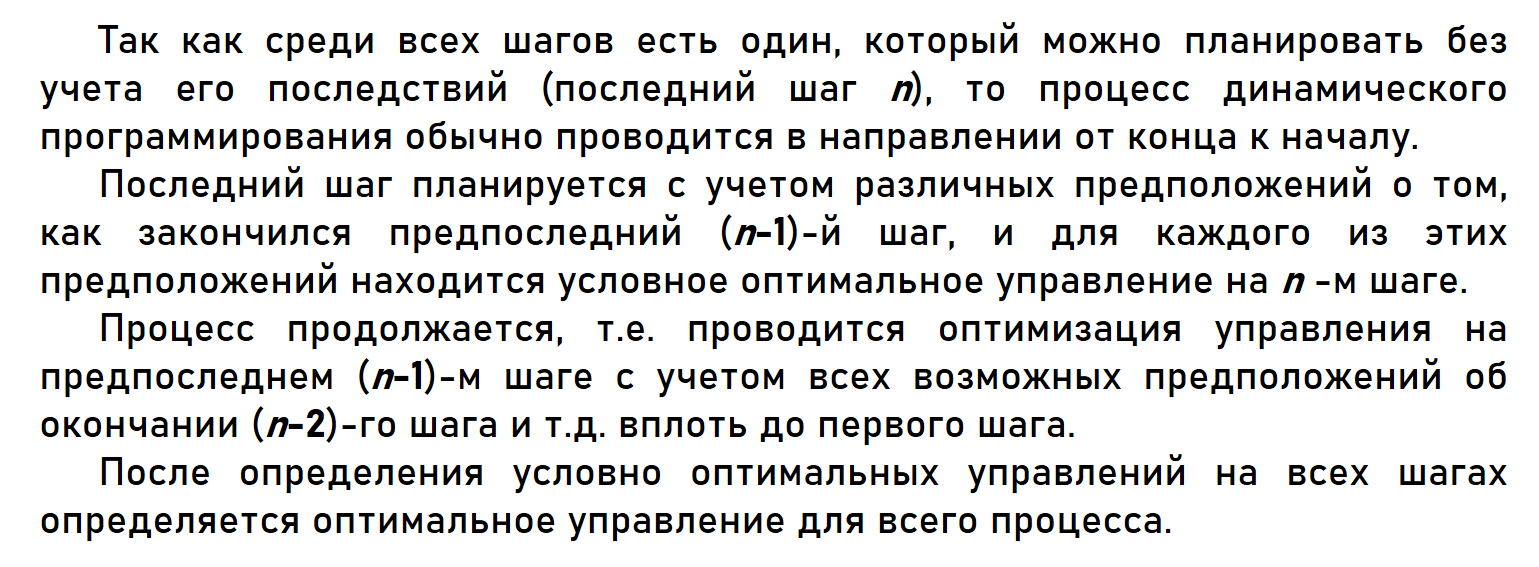


Чем аддитивная функция отличается от мультипликативной?

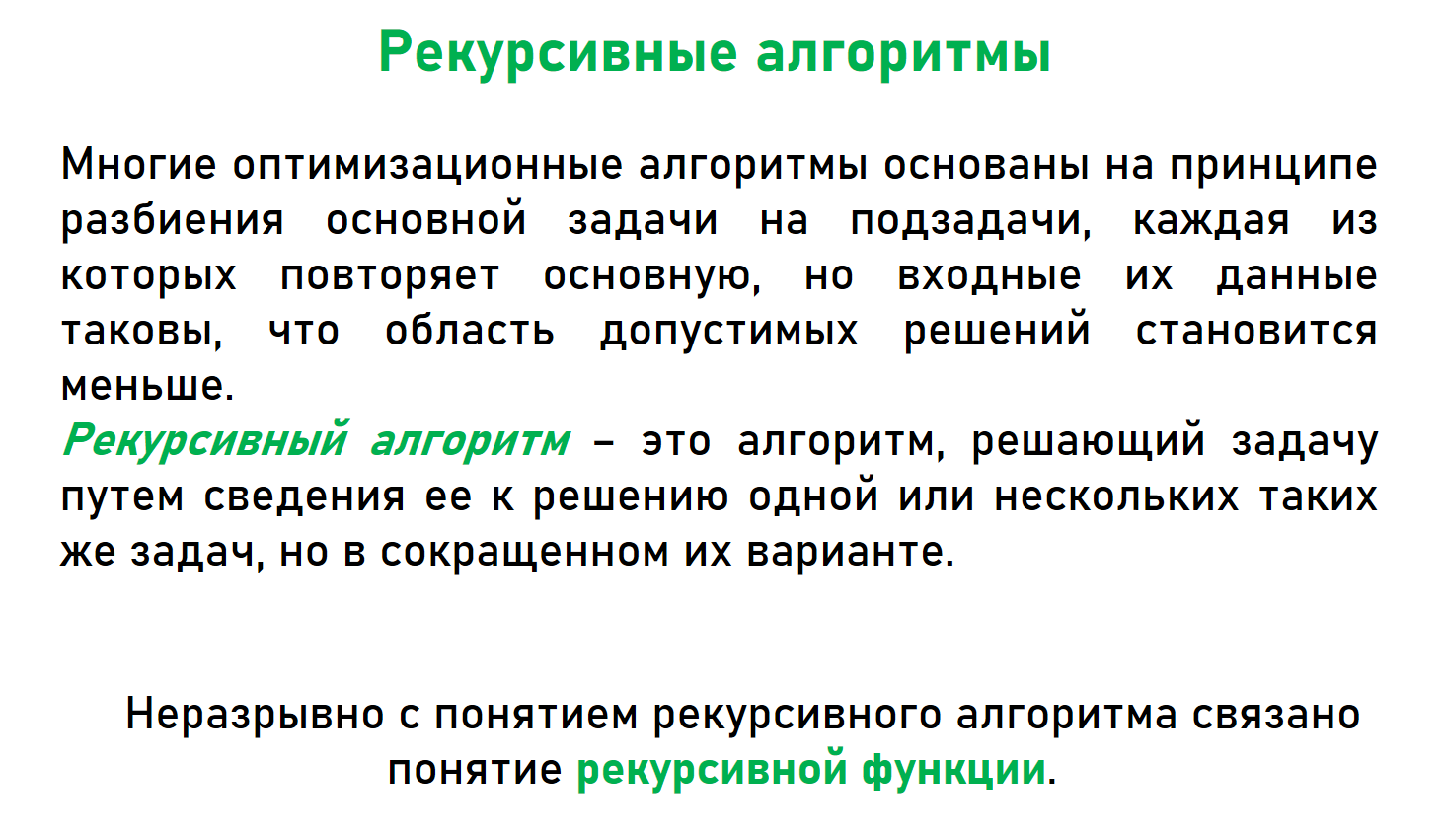


Каков принцип оптимальности Беллмана?

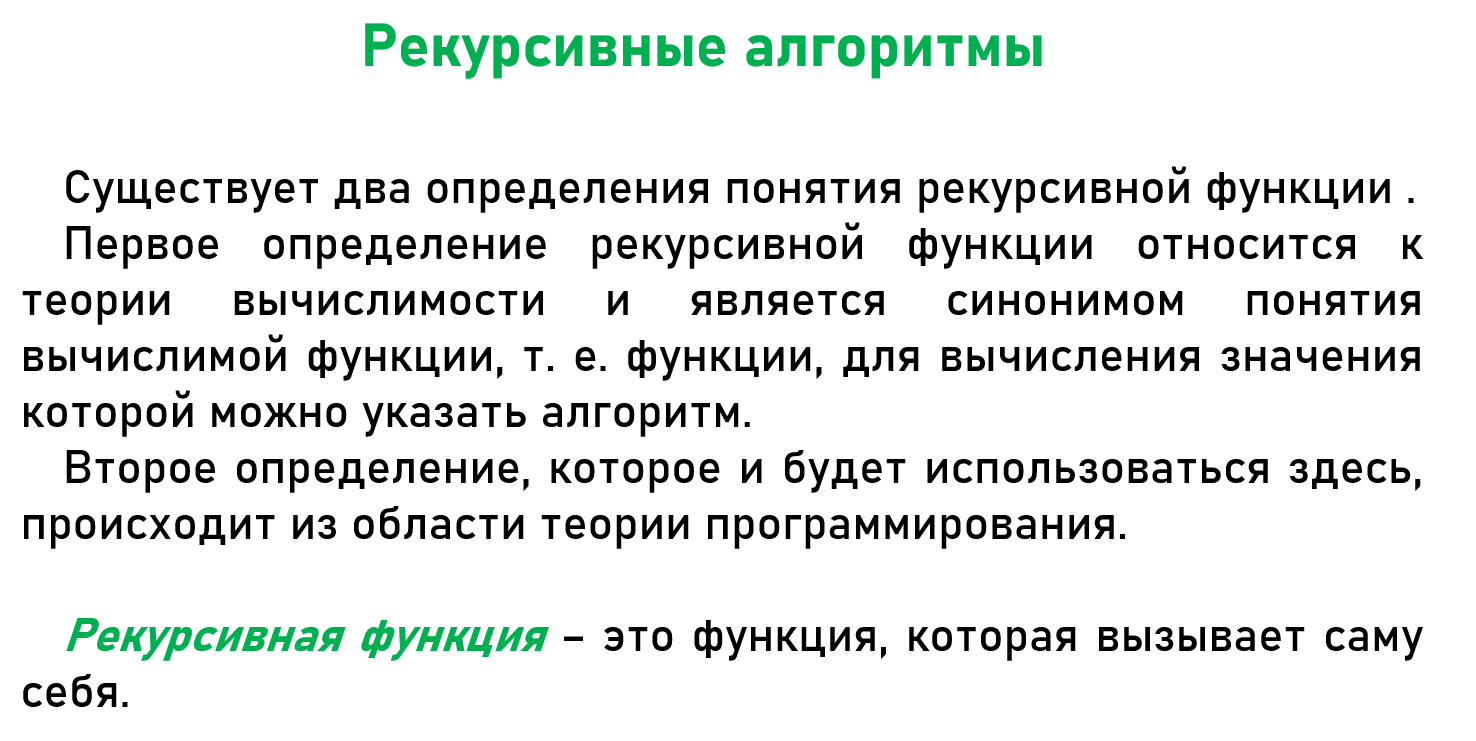




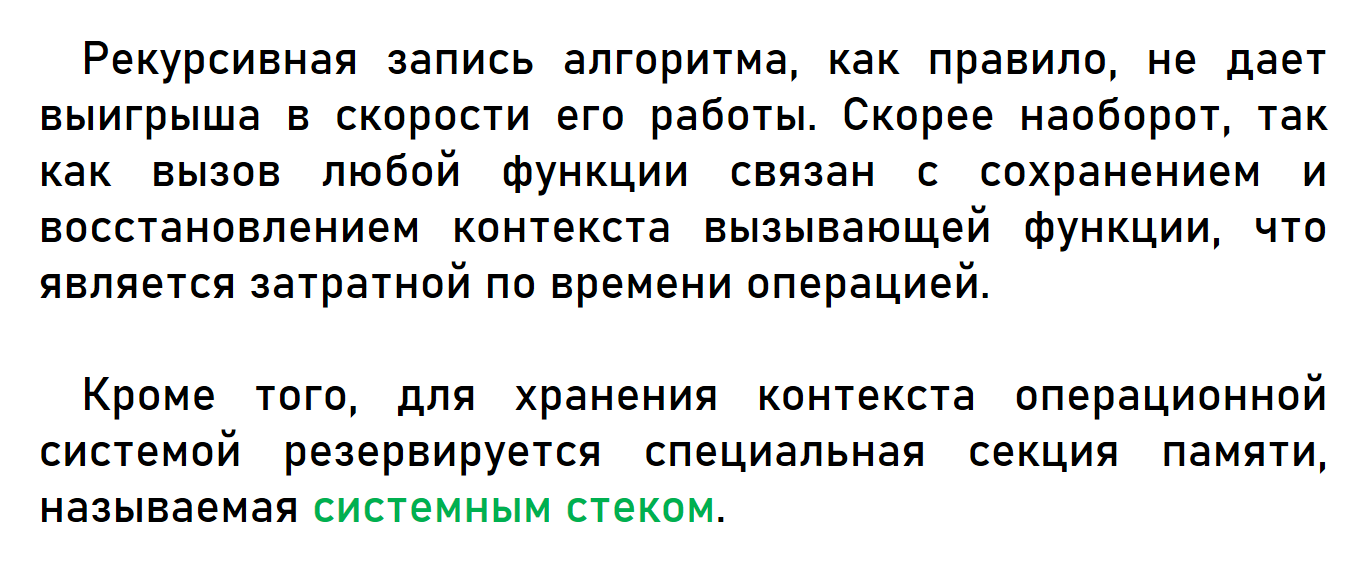
Что такое рекурсивный алгоритм?



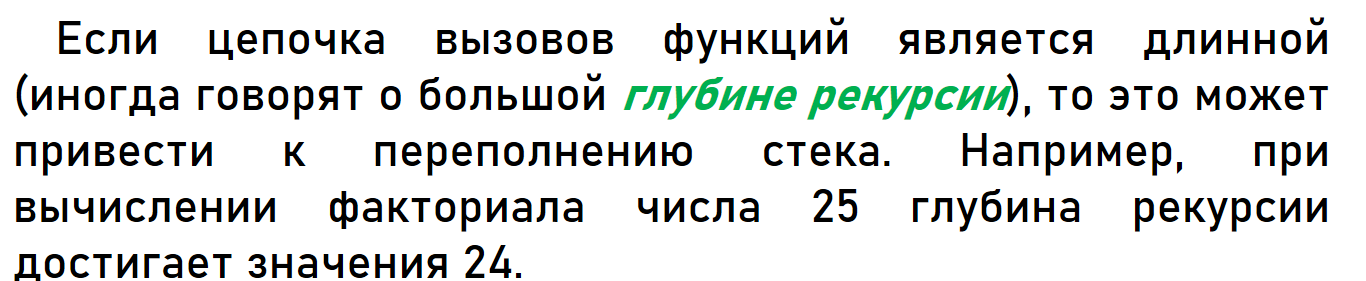
Что такое рекурсивная функция?



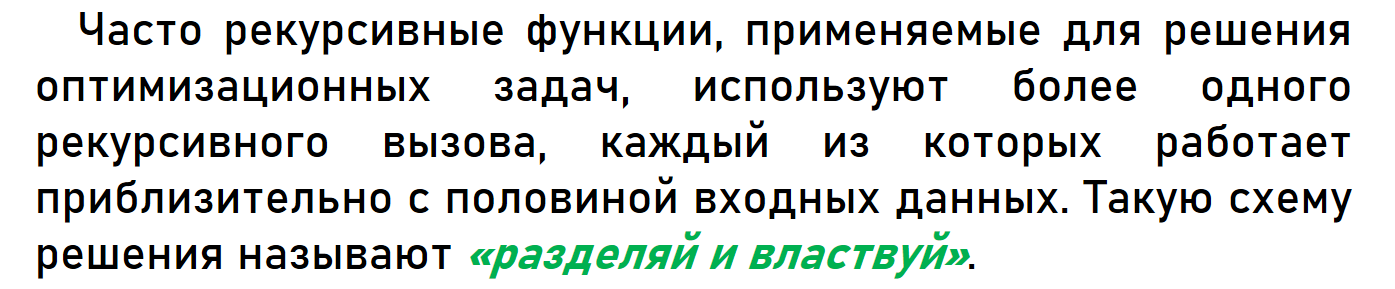
Что такое системный стек?



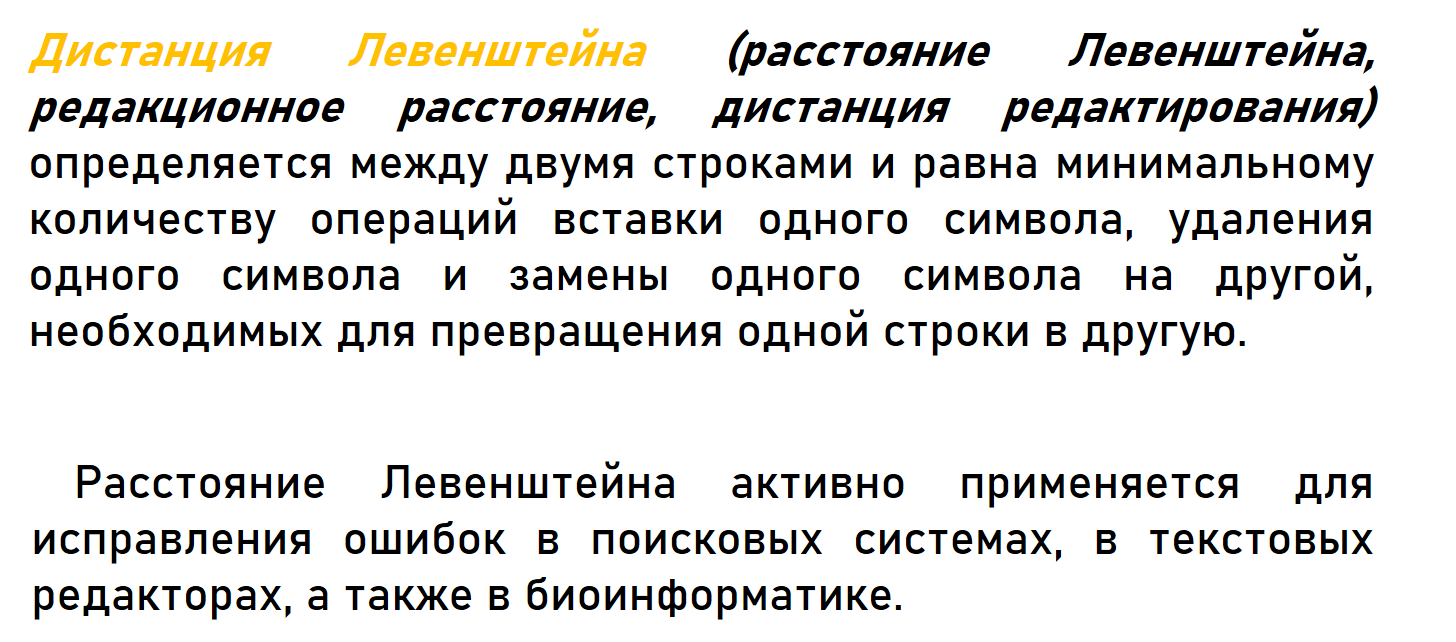
Объясните понятие «глубина рекурсии»



Поясните своими словами схему решения задачи по принципу «разделяй и властвуй»



Что такое редакционное расстояние?



Что такое подпоследовательность и как её можно получить из последовательности?

