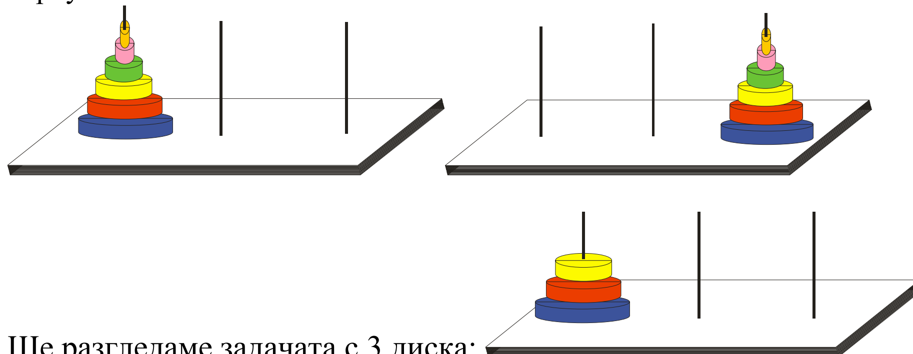
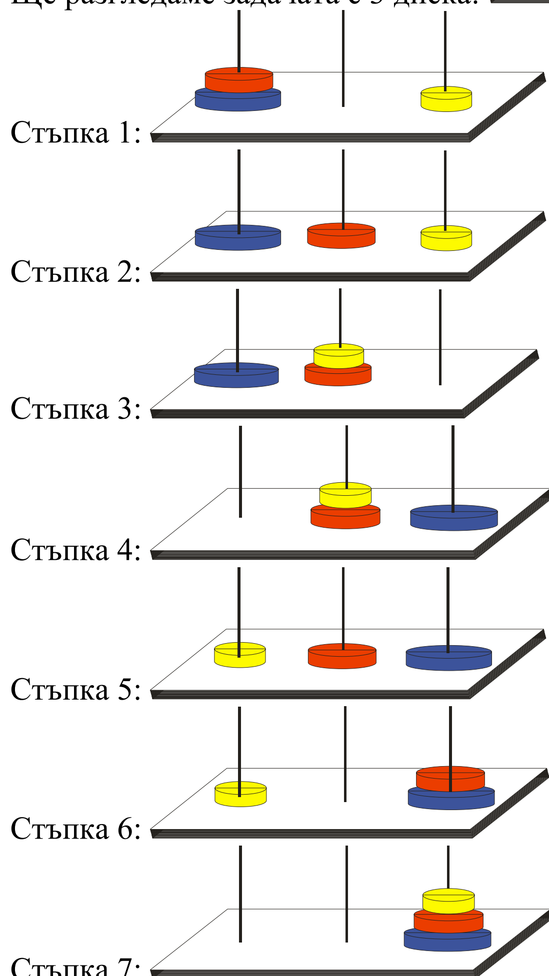
Въведение

Задачата за Ханойските кули:
Дадени са три стълба. На първият са поставени n диска с различен диаметър, наредени един върху друг от най-големия към най-малкия диск. Задачата е да се преместят всички дискове на третия стълб, като се запази подредбата им и при разместванията се спазва правилото винаги да се поставя по-малък диск върху по-голям.



Ще разгледаме задачата с 3 диска:



Броят на стъпките за решаване на задачата е 2^n-1 = 2^3-1=8-1=7.
Когато дисковете са 4 на брой, целта ни е да поставим най-големия диск на мястото му на третия пилон, после разиграването на останалите 3 диска ще се сведе до решавана вече задача.
След това вече, когато дисковете станат 5, отново търсим начин на поставим най-големия на мястото му на третия пилон, и после следва решаваната задача с 4 диска и т.н.
Т.е. разбиваме общата задача на по-малки задачи от същия тип.
Този подход се нарича рекурсия и се използва много често при задачи, при които имаме повтарящи се действия.

Рекурсия

1. Определение на понятията рекурсия и рекурсивна зависимост

Рекурсията е метод за дефиниране на алгоритми, множества, понятия, рецици и други чрез самите тях. Рекурсията може да се приложи в случаите, когато дадена задача може да се разбие на по-малки подзадачи от същия тип. В този случай казваме, че има рекурентна зависимост.

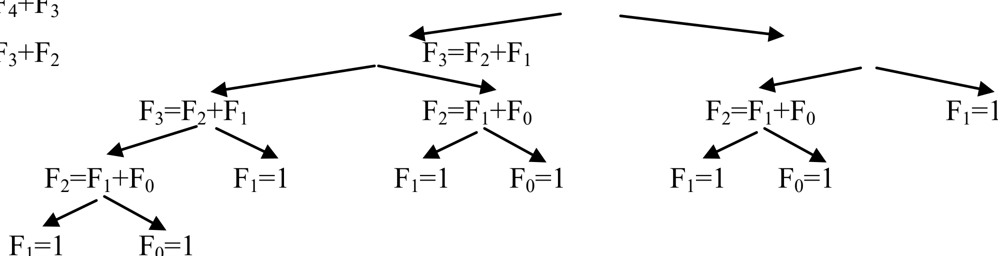
Примери за задачи, в които има рекурентна зависимост и могат да се решат с рекурсия:

Пример 1: Ханойски кули: Дадени са три стълба. На първият са поставени n диска с различен диаметър, наредени един върху друг от най-големия към най-малкия диск. Да се преместят всички дискове на третия стълб, като се запази подредбата им и при разместванията се спазва правилото винаги да се поставя по-малък диск върху по-голям.

Пример 2. Да се намери n-тия член на редицата на Фибоначи 1, 1, 2, 3, 5, 8, 13, 21, 34, ..., ако се знае, че $F_n=F_{n-1}+F_{n-2}$, $F_0=F_1=1$.

Например: Да се пресмятане F_5:

$F_5=F_4+F_3$



$F_4=F_3+F_2$ $F_3=F_2+F_1$

$F_3=F_2+F_1$ $F_2=F_1+F_0$ $F_2=F_1+F_0$ $F_1=1$

$F_2=F_1+F_0$ $F_1=1$ $F_1=1$ $F_0=1$ $F_1=1$ $F_0=1$

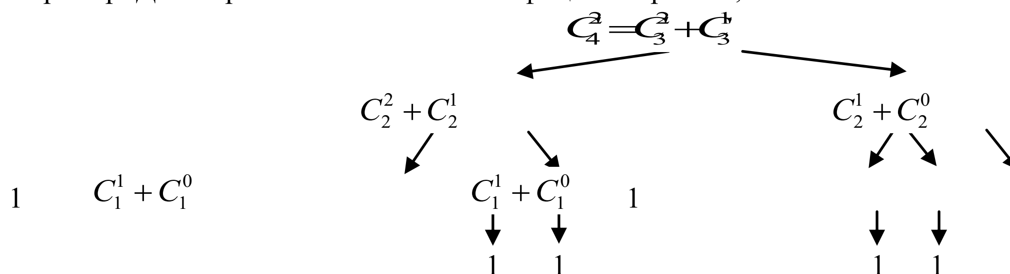
$F_1=1$ $F_0=1$

Следователно: $F_0=1$, $F_1=1$, $F_2=2$, $F_3=3$, $F_4=5$, $F_5=8$

Пример 3. Да се пресметне биномния коефициент C_n^m, при зададени n и m (0<=m<=n) и следната зависимост: $C_n^0=C_n^n=1$ и $C_n^m=C_{n-1}^m+C_{n-1}^{m-1}$.

Например: Да се пресметне биномния коефициент при n=4, m=2

$C_4^2=C_3^2+C_3^1$



$C_2^2+C_2^1$ $C_2^1+C_2^0$

1 $C_1^1+C_1^0$ $C_1^1+C_1^0$ 1

1 1 1 1

или

Има и други подобни примери. Общото при тях е, че рекурентната зависимост е явно зададена. В следващите примери също има рекурентна зависимост, но ние трябва сами да я установим. Този тип задачи ние сме решавали с помощта на цикъл, а сега ще кажем как става като използваме рекурсия. Кой от двата метода е по-ефективен ще обсъдим по-късно.

Пример 4. Пресмятане на n! = 1.2.3. ... .n

Знаем, че произведението на n числа се пресмята по формулата произведението на предходните (n-1) числа, умножено по n.

Това ние решавахме чрез цикъл и записвахме така:

```
P=1;
for (int i=1; i <=n; i++) P=P*i;
```

Но може да се запише и така n!=(n-1)! * n; за n>0 и 0!=1, т.е.

$$n! = \begin{cases} 1, & \text{за } n=0 \\ (n-1)! * n, & \text{за } n>0 \end{cases}$$

Пример 5. Пресмятане на сума на числата от 1 до n. Рекурсивната зависимост е $S_n = S_{n-1}+n$, като n>1 и $S_1=1$, т.е

$$S_n = \begin{cases} 1, & \text{за } n=1 \\ S_{n-1}+n, & \text{за } n>0 \end{cases}$$

Пример 6. Пресмятане на x^n, при n>=0

$$x^n = \begin{cases} 1, & \text{за } n=0; \\ x * x^{n-1}, & \text{за } n>0; \end{cases}$$

Пример 7. Освен тези две групи задачи, най-същественото приложение на рекурсията е за решаване на задачи, изискващи търсене с връщане назад – каквито са задачите за обхождане на шахматната дъска, намиране на път между два града и др.
Интересни приложения на рекурсията има в разработването на компилатори, при алгоритмите на изкуствения интелект и т.н.

2. Рекурсивни функции – това са функции, които извикват себе си за изпълнение, т.е. в описанието си съдържат обръщение към самата себе си.

При използването на рекурсивни функции се изисква дефинирането на един или няколко базови случая, към които се свеждат останалите. Базовият случай е условие за край на рекурсивното изпълнение.
Възможно е дадена функция да се извиква за изпълнение друга функция, която пък извиква отново първата. Това пак е рекурсия, но е косвена, а когато функцията вика себе си – пряка рекурсия.
Не всички компютърни езици поддържат рекурсия.

Пример 8. Да се напише рекурсивна функция за пресмятане на n!
Решение:
Проблемът за изчислението на n! се свежда до пресмятането на (n-1)! (което се предполага, че е по-лесно) и умножението на резултата с n. Базовият случай е при n=0.

```
#include <iostream>
using namespace std;
int fact(int n)
{
        if (n==0) return 1;
        else return n*fact(n-1);
}
int main()
{
        int n;
        cout<<"n="; cin>>n;
        cout <<n<<"! = "<<fact(n)<<endl;
        return 0;
}
```

3. Механизъм на действие на рекурсията
Да разгледаме как работи тази функция за n=5.
fact(5) = 5 * fact (4)   // спира изпълнението на функцията fact за стойност n=5.

// извиква се за изпълнение функция fact за стойност n=4
fact (4) = 4 * fact(3) // спира изпълнението за n=4
// ново извикване на fact за n=3
fact(3) = 3 * fact(2) //спира изпълнението за n=3
// извиква се fact за n=2
fact(2) = 2 * fact(1) // спира изпълнението за n=2
//извиква се fact за n=1
fact(1)=1*fact(0) //спира изпълнението за n=1
//пресмята се fact (0)
1 //базов случай, дъно

Този процес до тук се нарича **разгъване на рекурсията (слизане към дъното)**. До тука умноженията се отлагат – не се изпълняват, тъй като не знаем текущата стойност на функцията fact.
Сега започва обратния процес – **свиване на рекурсията (нагоре)**. Сега се извършват изчисленията.
fact(2) = 2 * 1=2
fact(3) = 3 * fact(2) = 3 * 2 = 6
fact (4) = 4 * fact(3) = 4 * 6 = 24
fact(5) = 5 * fact (4) = 5 * 24 = 120

При разгъване на рекурсията извикваща функция, текущите стойности на параметрите й 5, 4, 3 и т.н. и дефинираните в нея променливи се поставят в стек, а при свиване на рекурсията се вземат от там. Всяко извикано копие на функцията има собствена памет, затова при много рекурсивни извиквания е възможно да настъпи грешка от препълване на стека и програмата прекъсва.

4. Рекурсия или итерация

Отговорът зависи от конкретната задача и от това какво целим. Доказано е, че всяка задача, която може да се реши рекурсивно, може да се реши итеративно (без рекурсия) и обратно. Използването на рекурсия не увеличава бързодействието на програмата, напротив, в много случаи е крайно неефективно.
Затова, ако задачата има лесно и кратко итеративно решение, то е за предпочитане пред рекурсивното.
А при задачи като търсене на път в лабиринт или ханойските кули, рекурсията е единствения разумен подход. Итеративното решение при тях би било сложно, трудно и дълго.

Задача 1. Да се напише рекурсивна функция за пресмятане сумата на числата от 1 до n.

```
#include <iostream>
using namespace std;
int sum(int n)
{
 if (n==1) return 1;
 else return n+sum(n-1);
}
int main()
{
	int n;
	cout<<"n="; cin>>n;
	cout <<"Sum = "<<sum(n)<<endl;
	return 0;
}
```

Задача 2. Да се напише рекурсивна функция за пресмятане x^n.

```
#include <iostream>
using namespace std;
int power(int x, int n)
{
 if (n==0) return 1;
 else return x*power(x, n-1);
}
int main()
{
	int x, n;
	cout<<"x="; cin>>x;
```

```
        cout<<"n="; cin>>n;
        cout <<"power("<<x<<","<<n<<") = "<<power(x,n)<<endl;
        return 0;
}
```

Задача 3. Да се напише рекурсивна функция за намиране на НОД на две числа по метода на Евклид.
Решение: Най-напред трябва да намерим рекурентната зависимост

$$НОД(A,B) = \begin{cases} A, & \text{ако } A=B \text{ /базов случай, гранично условие/} \\ НОД(A-B, B), & \text{ако } A>B) \\ НОД(A, B-A), & \text{ако } B>A \end{cases}$$

НОД(35,14) = НОД(21, 14)
НОД(21, 14) = НОД(7, 14)
НОД(7, 14) = НОД(7, 7)
НОД(7, 7) = 7 //базов случай
НОД(7, 14) = 7
НОД(21, 14) = 7
НОД(35,14) = 7

```
#include <iostream>
using namespace std;
int NOD(int a, int b)
{       if (a==b) return a;
 else
        if (a>b) return NOD(a-b, b);
        else return NOD(a, b-a);
}
int main()
{       int a,b;
        cout<<"a="; cin>>a;
        cout<<"b="; cin>>b;
        cout <<"NOD("<<a<<","<<b<<")="<<NOD(a,b)<<endl;
        return 0;
}
```

Задача 4. Да се напише рекурсивна функция за пресмятане на n-тия член на редицата на Фибоначи.

```
#include <iostream>
using namespace std;
int fib(int n)
{
 if ((n==0)||(n==1)) return 1;
 else return (fib(n-2)+fib(n-1));
}
int main()
{       int n;
        cout<<"n="; cin>>n;
        cout <<"fib("<<n<<") = "<<fib(n)<<endl;
        return 0;
}
```

Задача 5. Да се напише рекурсивна функция за пресмятане на биномния коефициент C_3^1, като знаем следната зависимост: $C_n^0 = C_n^n = 1$ и $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$. Пресментете C_{10}^4.

```
#include <iostream>
using namespace std;
int C(int n, int m)
{
if ((n==m)||(m==0)) return 1;
else return C(n-1, m)+C(n-1, m-1);
}
int main()
{
```

```
int n,m;
        cout<<"n="; cin>>n;
        cout<<"m="; cin>>m;
        cout <<"C("<<n<<","<<m<<")="<<C(n,m)<<endl;
        return 0;
}
```

Задача 6. Да се напише рекурсивна функция, която намира стойността на функцията на Акерман Ack(m, n), дефинирана за m ≥ 0 и n ≥ 0 по следния начин:
Ack(0, n) = n+1
Ack(m, 0) = Ack(m-1, 1), m > 0
Ack(m, n) = Ack(m-1, Ack(m, n-1)), m > 0, n > 0.

Решение:

```
#include <iostream>
using namespace std;
int ack( int m, int n )
{
        if ( m == 0 ) return n + 1;
        else    if ( n == 0 ) return ack( m - 1, 1 );
                else  return ack( m - 1, ack( m, n - 1 ) );
}
int main( void )
{
        int n, m;
        do
        {       cout << "n = ";    cin >> n;
                cout << "m = ";   cin >> m;
        } while (n<0 || m<0);
        cout << "Ack(" << m << ',' << n << ") = " << ack( m, n ) << endl;
        return 0;
}
```

Зад. 7. Напишете рекурсивна функция, която проверява дали в записа на дадено число се съдържа дадена цифра.

I начин

```
#include <iostream>
#include <string>
using namespace std;
string DigitK( int n, int k )
{
        if (( n % 10 ) == k ) return "Yes";
        if ( n == 0 ) return "No";
        return DigitK( n / 10, k );
}

int main( void )
{       int n, k;
        cout << "n = "; cin >> n;
        do
        {
                cout << "k = ";
                cin >> k;
        } while ( (k < 0) || (k > 9) );
        cout << DigitK( n, k ) <<endl;
        return 0;
}
}
```

II начин

```
#include <iostream>
using namespace std;
void DigitK( int n, int k )
{
        if (( n % 10 ) == k ) cout<< "Yes";
        else
        if ( n == 0 ) cout<< "No";
        else DigitK( n / 10, k );
}

int main( void )
{       int n, k;
        cout << "n = "; cin >> n;
        do
        {
                cout << "k = ";
                cin >> k;
        } while ( (k < 0) || (k > 9) );
        DigitK( n, k ) ;
        return 0;
```

Зад. 8. Напишете рекурсивна функция, която:

а/ извежда цифрите на числото в обратен ред.

```
#include <iostream>
using namespace std;
void Digits (int n)
{           if (n>0)
            {    cout<<n%10;
                 Digits (n/10);
            }
}
int main()
{           int n;
            cout<<"n = "; cin>>n;
            Digits (n);
            return 0;
}
```

б/ извежда цифрите на числото отляво надясно.

```
#include <iostream>
using namespace std;
void Digits (int n)
{           if (n>0)
            {    Digits (n/10);
                 cout<<n%10;
            }
}
int main()
{           int n;
            cout<<"n = "; cin>>n;
            Digits (n);
            return 0;
}
```

в/ създава число със същите цифри в обратен ред

в/ създава число със същите цифри в обратен ред

```
#include <iostream>
using namespace std;
int m=0;
void Digits (int n)
{           if (n>0)
            {    m=m*10+(n%10);
                 Digits (n/10);
            }
}
int main()
{           int n;
            cout<<"n = "; cin>>n;
            Digits (n);
            cout<<m<<endl;
            return 0;
}
```

Зад. 9. Напишете рекурсивна функция за извеждане на двоичния запис на число.

Упътване: Трябва да се намерят всички остатъци при деление на 2 и да се изведат в обратен ред на получаване. За да ги изведем отзад напред, трябва извеждането да е при свиване на рекурсията.

```
#include <iostream>
using namespace std;
void Bin(int n)
{       if (n>0)
        {       Bin(n/2);
                cout<<n%2;
        }
}
int main()
{       int n;
        cout<<"n = "; cin>>n;
        Bin(n);
        return 0;
}
```

Зад. 10. Да се напише рекурсивна функция, която въвежда редица от знакове с произволна дължина, до въвеждане на ‘#’. Да се изведат знаковете в обратен ред.

```
#include <iostream>
using namespace std;
void Invert()
{       char x;
        cin>>x;
        if (x!='#') Invert();
        if (x!='#') cout<<x;
}
```

```
int main()
{	Invert();
	return 0;
}
```

Зад. 11. Какъв ще бъде резултатът от рекурсивната функция, ако въведем x=10?

```
#include <iostream>
using namespace std;
void Rec(int x)
{	x=x/2;
	cout<<x<<" "
	if (x>0) Rec(x);
	cout<<x<<" "
}
int main()
{	int x;
	cout<<"x="; cin>>x;
	Rec(x);
	return 0;
}
```

Отг. 5 2 1 0 0 1 2 5

```
#include <iostream>
using namespace std;
void Rec(int &x)
{	x=x/2;
	cout<<x<<" ";
	if (x>0) Rec(x);
	cout<<x<<" ";
}
int main()
{	int x;
	cout<<"x="; cin>>x;
	Rec(x);
	return 0;
}
```

Отг. 5 2 1 0 0 0 0 0

Зад. 12. Какъв ще бъде резултатът от рекурсивната функция, ако въведем 123456$?

```
#include <iostream>
using namespace std;
char x;
void Rec()
{	cin>>x;
	if (x!='$') Rec();
	cout<<x;
}
int main()
{	Rec();
	return 0;
}
```

Отг. $$$$$$$

```
#include <iostream>
using namespace std;
void Rec()
{	char x;
	cin>>x;
	if (x!='$') Rec();
	cout<<x;
}
int main()
{	Rec();
	return 0;
}
```

Отг. $654321

```
#include <iostream>
using namespace std;
void Rec()
{	char x;
	cin>>x;
	cout<<x;
	if (x!='$') Rec();
}
int main()
{	Rec();
	return 0;
}
```

Отг. 123456$

Зад. 13. Какъв ще бъде резултатът от рекурсивната функция, при n=3, n=6 и n=2?

```
#include <iostream>
using namespace std;
void Rec(int n)
{
	if (n==3)
		cout<<3<<endl;
	else
		Rec(n-1);
}
int main()
{
	int n;
	cout<<"n="; cin>>n;
	Rec(n);
	return 0;
}
```

Отг. При n=3 и n=6 извежда 3, при n=2 не извежда нищо

Зад. 14. Ханойски кули. Дадени са три стълба. На първият са поставени n диска с различен диаметър, наредени един върху друг от най-големия към най-малкия диск. Да се преместят всички дискове на третия стълб, като се запази подредбата им и при разместванията се спазва правилото винаги да се поставя по-малък диск върху по-голям.

Решение:

Разполагаме с 3 стълба, които ще означим с 1, 2 и 3. 1 е стълбът, от който вземаме дисковете, 3 – този, на който местим дисковете, а 2 е помощен, на който временно ги разполагаме.
Задачата за n диска се свежда до задача за n-1 диска по следния начин:

- премстваме горните (n-1) диска на стълб 2, като използваме стълб 3 за помощен
- преместваме n[тия] диск от стълб 1 на стълб 3
- преместваме пак (n-1)[те] диска от стълб 2 на стълб 3, като използваме стълб 1 за помощен

Ще напишем рекурсивна функция Hanoj, която ще приема 4 аргумента: I - броя на дисковете, II - от кой стълб вземаме, III - кой стълб е помощен и IV - на кой стълб поставяме диска.

```
#include <iostream>
using namespace std;
void Hanoj(int n, int a, int b, int c)
{
        if (n>0)
        {
                Hanoj(n-1, a,c,b);
                cout <<a<<"-->"<<c<<endl;
                Hanoj(n-1, b,a,c);
        }
}
int main ()
{
        int n;
        cout<< "Number of disks: "; cin>>n;
        int a=1, b=2, c=3;
        Hanoj(n, a,b,c);
        return 0;
}
```

Зад. 15. Квадратна мрежа с n реда и n стълба ($1 \leq n \leq 20$) има два вида квадратчета - бели и черни. В черно квадратче може да се влезе, но не може да се излезе. От бяло квадратче може да се премине във всяко от осемте му съседни, като се прекоси общата им страна или връх. Да се напише програма, която ако са дадени произволна мрежа с бели и черни квадратчета и две произволни квадратчета - начално и крайно, определя дали от началното квадратче може да се премине в крайното.
Анализ на алгоритъма:
а) Ако началното квадратче не е в мрежата, приемаме, че не може да се премине от началното до крайното квадратче.
б) Ако началното квадратче съвпада с крайното, приемаме, че може да се премине от началното до крайното квадратче.
в) Ако началното и крайното квадратчета са различни и началното квадратче е черно, не може да се премине от него до крайното квадратче.
г) Във всички останали случаи, от началното квадратче може да се премине до крайното тогава и само тогава, когато от някое от съседните му квадратчета (в хоризонтално, във вертикално или диагонално направление), може да се премине до крайното квадратче.
Мрежата ще реализираме чрез двумерен масив, в който a[i][j= 1, ако квадратче (i, j) е бяло, и a[i][j= 0, ако е квадратче (i, j) е черно. Попълването на масива с 0 и 1 става във функция init със случайни числа. Означаваме началното квадретче (x1, y1), а крайното (x2, y2). Пътят, ако има такъв, е само по клетките със стойност 1.
В програмата е дефинирана рекурсивната функция way. Тя връща стойност true, ако от квадратче (x1, y1) може да се премине до квадратче (x2, y2) и false - в противен случай. За да се избегне връщане в текущото квадратче от неговите съседни, преди рекурсивните обръщения към way го правим черно чрез a[x1][y1]=0.

```
#include <iostream>
#include <time.h>
using namespace std;

int a[20][20];
```

```
int n, x1, y1, x2, y2;
void init();
bool way(int x1, int y1);

int main()
{
	init();
	cout<<"Enter koord begins and end"<<endl;
	cout<<"x1 = "; cin >> x1;
	cout<<"y1 = "; cin >> y1;
	cout<<"x2 = "; cin >> x2;
	cout<<"y2 = "; cin >> y2;

	if (way(x1, y1))
		cout << "Yes " << endl;
	else
		cout << "No " << endl;
	return 0;
}

void init()
{
	srand(time(NULL));
	cout << "n = ";cin >> n;
	int i, j;
	for (i = 0; i<= n-1; i++)
		for (j = 0; j <= n-1; j++)
			a[i][j] = rand()%2;
	for (i = 0; i<= n-1; i++)
	{
		for (j = 0; j <= n-1; j++)
			cout << a[i][j]<<" ";
		cout << '\n';
	}
}

bool way(int x1, int y1)
{
	if (x1<0 || x1>n-1 || y1<0 || y1>n-1 || x2<0 || x2>n-1 || y2<0 || y2>n-1) return false;
	if (x1 == x2 && y1 == y2) return true;
	if (a[x1][y1] == 0) return false;
	a[x1][y1] = 0;
	bool b = way(x1+1,  y1) || way(x1-1, y1)  || way(x1,  y1+1) || way(x1, y1-1)  ||
	           way(x1-1,y1-1) || way(x1-1, y1+1)|| way(x1+1,y1-1) || way(x1+1, y1+1);
	a[x1][y1] = 1;
	return b;
}
```

Зад. 16. Дадена е квадратна мрежа от клетки, всяка от които е празна или запълнена. Запълнените клетки, които са свързани, т.е. имат съседни в хоризонтално, вертикално или диагонално направление, образуват област. Да се напише програма, която намира броя на областите и размера (в брой клетки) на всяка област.

Анализ на алгоритъма:

Мрежата ще представим чрез таблица и ще реализираме чрез двумерен масив. При това a[i][j]=1 ако квадратче (i, j) е запълнено, и a[i][j]=0 – в противен случай. Попълването на масива с 0 и 1 става във функция init със случайни числа.

Ще дефинираме функция Size, която преброява клетките в областта, съдържаща дадена клетка (x, y). Функцията има два параметъра x и y - координатите на точката и реализира следния алгоритъм:

а) Ако клетката с координати (x, y) е извън мрежата, приемаме, че броят на клетките в областта е равен на 0.

б) В противен случай, ако клетката с координати (x, y) е празна, приемаме, че броят е равен на 0.
в) В останалите случаи, броят на клетките в областта е равен на сумата от 1 и броя на клетките на всяка област, на която принадлежат осемте съседни клетки на клетката (x, y).
От последния случай се вижда, че функция Size е рекурсивна. За да избегнем зацикляне и многократно преброяване, трябва преди рекурсивното обръщение на направим клетката (x, y) празна.

```
#include <iostream>
#include <time.h>
using namespace std;

int a[20][20];
int n, x, y;
void init();
int Size(int x, int y);

int main()
{
        init();
        int br = 0;
        int i, j;
        for (i = 0; i <= n-1; i++)
                for (j = 0; j <= n-1; j++)
                        if (a[i][j] == 1)
                        {
                                br++;
                                cout << "Location Nr " <<  br << " have a size " << Size(i, j) << endl;
                        }
        return 0;
}

void init()
{
        srand(time(NULL));
        cout << "n = ";cin >> n;
        int i, j;
        for (i = 0; i<= n-1; i++)
                for (j = 0; j <= n-1; j++)
                        a[i][j] = rand()%2;
        for (i = 0; i<= n-1; i++)
        {
                for (j = 0; j <= n-1; j++)
                        cout << a[i][j]<<" ";
                cout << '\n';
        }
}

int Size(int x, int y)
{
        if (x<0 || x>n-1 || y<0 || y>n-1) return 0;
        if (a[x][y] == 0) return 0;
        a[x][y] = 0;
        return    1 + Size(x-1, y+1) + Size(x, y+1) + Size(x+1, y+1) + Size(x+1,y) +
                        Size(x+1, y-1) + Size(x, y-1) + Size(x-1, y-1) + Size(x-1, y);
}
```

Зад. 17. Дадени са n града ($1 \leq n \leq 20$) и целочислена матрица Anxn, така че a_{ij} е равно на 1, ако има пряк път от град i до град j и е 0 в противен случай ($0 \leq i, j \leq n-1$). Да се напише програма, която установява дали съществува път между два произволно зададени града (Приемаме, че ако от град i до град j има път, то има път и от град j до град i).

Анализ на задачата:

Ще дефинираме рекурсивната булева функция way, която зависи от два параметъра i и j, показващи номерата на градовете, между които се проверява дали съществува път. Функцията реализира следния алгоритъм:
а) Ако $i = j$, съществува път от град i до град j.
б) Ако $i \neq j$ и има пряк път от град i до град j, има път между двата града.
в) В останалите случаи има път от град i до град j, тогава и само тогава, когато съществува град k, с който град i е свързан с пряк път и от който до град j има път.

I начин: използваме допълнителен масив, в който отбелязваме през кой град сме минали вече

```
#include <iostream>
using namespace std;

int a[20][20];
int n;
bool visited[20];
void init();
bool way( int first,  int second);

int main()
{
	init();
	int first, second;
	cout << "First town: "; cin >> first;
	cout << "Second town: "; cin >> second;
	if (way(first, second))
		cout << "yes \n";
	else
		cout << "no \n";
	return 0;
}

void init()
{
	cout << "n = ";cin >> n;
	int i, j;
	for (i=0; i<=n-1; i++)
	{
		a[i][i] = 1;
		for (j=i+1; j<=n-1; j++ )
		{
			cout << "a[" << i << "][" << j << "] = " << "a[" << j << "][" << i << "] = ";
			cin >> a[i][j];
			a[j][i] = a[i][j];
		}
	}
}

bool way( int first,  int second)
{
	int i;
	if ( first == second ) return true;
	visited[first] = true;
	for (i = 0; i < n; i++ )
		if ( a[first][i] && !visited[i] )
			if ( way(i, second) ) return true;
	return false;
}
```

II начин – използваме допълнителна променлива, която става истина, когато намерим път. За да избегнем зациклянето, използваме подхода в предходните задачи - премахваме пътя между града от който идваме и града, в който влизаме /a[first][k] = 0; a[k][first] = 0;/ и после го възстановяваме / a[first][k] = 1; a[k][first] = 1;/.

```
#include <iostream>
using namespace std;

int a[20][20];
int n;
void init();
bool way(int first, int second);

int main()
{
	init();
	int first, second;
	cout << "First town: "; cin >> first;
	cout << "Second town: "; cin >> second;
	if (way(first, second))
		cout << "yes \n";
	else
		cout << "no \n";
	return 0;
}
void init()
{	cout << "n = ";cin >> n;
	int i, j;
	for (i=0; i<=n-1; i++)
	{	a[i][i] = 1;
		for (j=i+1; j<=n-1; j++ )
		{	cout << "a[" << i << "][" << j << "] = " << "a[" << j << "][" << i << "] = ";
			cin >> a[i][j];
			a[j][i] = a[i][j];
		}
	}
}
bool way(int first, int second)
{	if (first == second) return true;
	if (a[first][second] == 1) return true;
	bool b = false;
	for (int k = 0;b==false && k <= n-2; k++)
		if (a[first][k] == 1)
		{	a[first][k] = 0; a[k][first] = 0;
			b = way(k, second);
			a[first][k] = 1; a[k][first] = 1;
		}
	return b;
}
```

Зад. 18. Дадени са n града ($1 \leq n \leq 10$) и целочислена матрица Anxn, така че a_{ij} е равно на 1, ако има пряк път от град i до град j и е 0 в противен случай ($0 \leq i, j \leq n-1$). Да се напише програма, която извежда пътя между два произволно зададени града в случай, че път между тях съществува.

Анализ на задачата:

Задачата е продължение на предходната, като запазваме пътя в масив path и го извеждаме.

```
#include <iostream>
using namespace std;

int a[20][20];
int n;
bool visited[20];
int path[20], br;
void init();
bool way( int first,  int second);
```

```
int main()
{
	init();
	int first, second;
	cout << "First town: "; cin >> first;
	cout << "Second town: "; cin >> second;
	br = 1;
	path[0] = first;
	if (way(first, second))
	{
		cout << "yes \n";
		for (int i = 0; i <= br - 1; i++ )
			cout << path[i] << " ";
		cout <<  endl;
	}
	else
		cout << "no \n";
	return 0;
}

void init()
{
	cout << "n = ";cin >> n;
	int i, j;
	for (i=0; i<=n-1; i++)
	{
		a[i][i] = 1;
		for (j=i+1; j<=n-1; j++ )
		{
			cout << "a[" << i << "][" << j << "] = " << "a[" << j << "][" << i << "] = ";
			cin >> a[i][j];
			a[j][i] = a[i][j];
		}
	}
}

bool way( int first,  int second)
{
	int i;
	if ( first == second ) return true;
	visited[first] = true;
	for (i = 0; i < n; i++ )
		if ( a[first][i] && !visited[i] )
		{
			path[br] = i;
			br++;
			if ( way(i, second) ) return true;
			br--;
		}
	return false;
}
```

Зад. 19. Дадени са n града (n е естествено число, $1 \leq n \leq 10$) и целочислена матрица Anxn, така че a_{ij} е равно на 1, ако има пряк път от град i до град j и е 0 в противен случай ($0 \leq i, j \leq n-1$). Да се напише булева функция fixway(int i, int j, int p), която установява дали съществува път от град i до град j с дължина p ($p \geq 1$).

Анализ на задачата:

Булевата функция fixway реализира следния алгоритъм:

а) Ако p = 1, има път от град i до град j с дължина p ако има пряк път между двата града.

б) Ако p > 1, има път от град i до град j с дължина p тогава и само тогава, когато съществува град k, с който град i е свързан с пряк път и от който до град j има път с дължина p-1.
Следващият програмен фрагмент дефинира само функцията fixway.

```
bool fixway(int i, int j, int p)
{if (p == 1) return arr[i][j] == 1; else
 {bool b = false;
  int k = -1;
  do
   {k++;
    if (arr[i][k] == 1) b = fixway(k, j, p-1);
   } while (!b && k <= n-2);
  return b;
 }
}
```

Тази функция извършва проверка за съществуване на цикличен път от един до друг връх с указана дължина.
Пример: Ако имаме само два града, означени с 0 и 1 и те са свързни с пряк път, fixway(0, 1, 3) ще отговори с true.
Ако търсим съществуването само на ациклични пътища, ще използваме модификацията на горната функция, дадена по-долу.

```
bool fixway(int i, int j, int p)
{bool b;
 int k;
 if (p == 1) return arr[i][j] == 1;
 b = false;
 k = -1;
 do
 {k++;
  if (arr[i][k] == 1)
   {arr[i][k] = 0; arr[k][i] = 0;
    b = fixway(k, j, p-1);
    arr[i][k] = 1; arr[k][i] = 1;
   }
 } while (!b && k <= n-2);
 return b;
}
```

Зад. 20. Дадени са n града (n е естествено число, $1 \leq n \leq 10$) и целочислена матрица Anxn, така че a_{ij} е равно на 1, ако има пряк път от град i до град j и е 0 в противен случай ($0 \leq i, j \leq n-1$). Да се напише процедура foundfixway(int i, int j, int p, int s&, int* x), която намира пътя от град i до град j с дължина p в случай, че такъв съществува.
За краткост, дефиницията на функцията fixway е пропусната. Пътят е запомнен в масива x, а параметърът s съдържа текущата му дължина.

```
#include <iostream.h>
int arr[10][10];
int n;
bool fixway(int i, int j, int p)
...
void foundfixway(int i, int j, int p, int& s, int* x)
{s++;
 x[s] = i;
 if (p == 1)
  {s++;
   x[s] = j;
  }
 else
  {bool b = false;
   int k = -1;
   do
    {k++;
```

```
            if (arr[i][k] == 1) b = fixway(k, j, p-1);
  } while (!b);
  foundfixway(k, j, p-1, s, x);
 }
}
            int main()
{int p = -1;
 int a[100];
 do
 {cout << "n= ";
  cin >> n;
 } while (n < 1 || n > 10);
int i, j;
 for (i = 0; i <= n-1; i++)
   for (j = 0; j <= n-1; j++)
               arr[i][j] = 0;
 for (i = 0; i <= n-2; i++)
   for (j = i+1; j <= n-1; j++)
   {cout << "connection between " <<  i << " and "
                                 << j << " 0/1? ";
             cin >> arr[i][j];
             arr[j][i] = arr[i][j];
            }
 int l;
 do
 {cout << "start and final towns, and len between them: ";
  cin >> i >> j >> l;
 } while (i < 0 || i > 9 || j < 0 ||  j > 9);
 if (fixway (i, j, l))
 {foundfixway(i, j, l, p, a);
  for (int m = 0; m <= l; m++) cout << a[m] << "  " ;
  cout << endl;
 }
 else cout << "no \n";
 return 0;
}
```

Зад. 21. Дадена е мрежа от m x n квадратчета, като за всяко квадратче е определен цвят - бял или черен. Път ще наричаме редица от съседни във вертикално или хоризонтално направление квадратчета с един и същ цвят. Област ще наричаме множество от квадратчета с един и същ цвят между всеки две, от които има път. Дадено е квадратче. Да се определи:

а) броят на квадратчетата от областта, в която се съдържа даденото квадратче.
б) броят на областите с цвят, съвпадащ с цвета на даденото квадратче.
в) броят на областите с цвят, различен от цвета на даденото квадратче.
г) броят на квадратчетата с цвят, съвпадащ с цвета на даденото квадратче, които не са в една област с него.

Зад. 22. Дадено е множество от n града и за всеки два от тях е определено дали са свързани с път или не. Едно множество от градове ще наричаме пълно, ако всеки два различни града, принадлежащи на множеството, са свързани с път. Да се напише програма, която по дадено k, k < n, извежда всички пълни множества, състоящи се от k на брой града.

Зад. 23. Дадено е множество от n града и за всеки два от тях е определено дали са свързани с път или не. Едно множество от градове ще наричаме независимо, ако всеки два различни града, принадлежащи на множеството, не са свързани с път. Да се напише програма, която по дадено k, k < n, извежда всички независими множества, състоящи се от k на брой града.

Зад. 24. Да се състави рекурсивна функция, която генерира и извежда всички последователности от символите 0 и 1 с дължина n и несъдържащи две последователни нули.

Решението не е както трябва

```
#include <iostream>
using namespace std;
void Print(int n,int k)
{

        if (n==1)
                cout<<k<<endl;
        else
                if (k==0)
                {        cout<<0;Print (n-1,1);}
                else
                {
                cout<<1;Print (n-1, 1);
                cout<<1;Print (n-1, 0);
                }
}
int main ()
{
        int n;
        cout<< "n = "; cin>>n;
        Print (n, 1);
        Print (n, 0);
        return 0;
}
```

Зад. 25. Трябва да платим сума n и разполагаме с банкноти от по 2, 5 и 10 лв. По колко начина може да платим сумата?