NP задачи

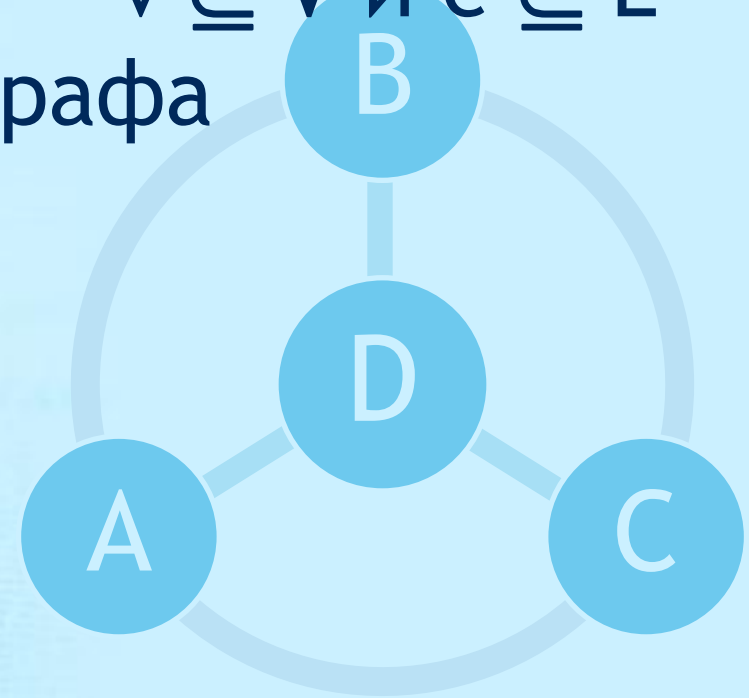
Алгоритмы на графах

Основные понятия

- G = (V,E)

  V – множество вершин {A, B, C, D, …}

  E – множество ребер {AB, BC, CD, …}
- (v,e) = g – подграф G: $v \subseteq V$ и $e \subseteq E$
- Число ребер полного графа с N вершинами равно $N(N-1)/2$

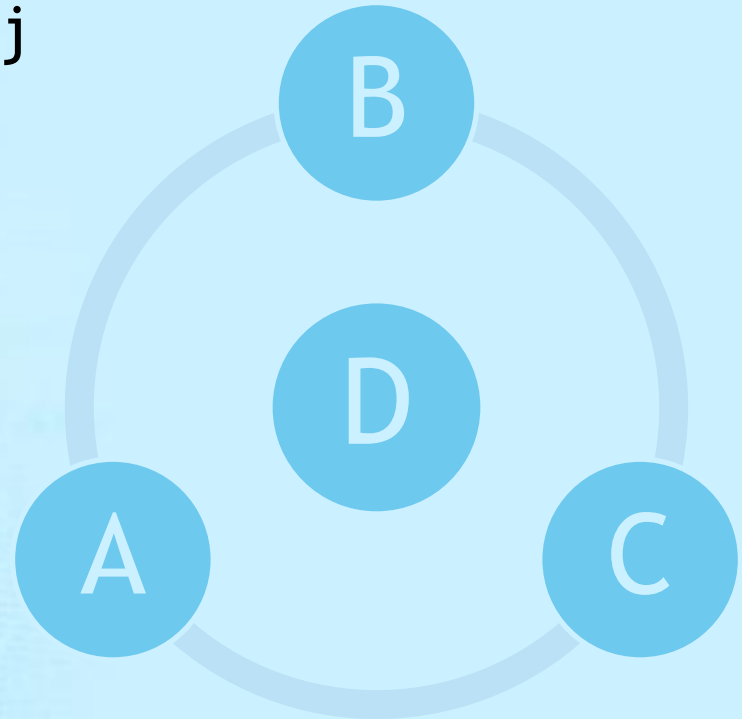


Путь P_{AC} из A в C это {{AB,BC},{AD,DC},{AB,BD,DC},{AD,DB,BC},{AC}}

Граф взвешенный: E → N

Связный: ∀ v_i, v_j ∃ Pv_iv_j

Цикл {AB, BC, CA}

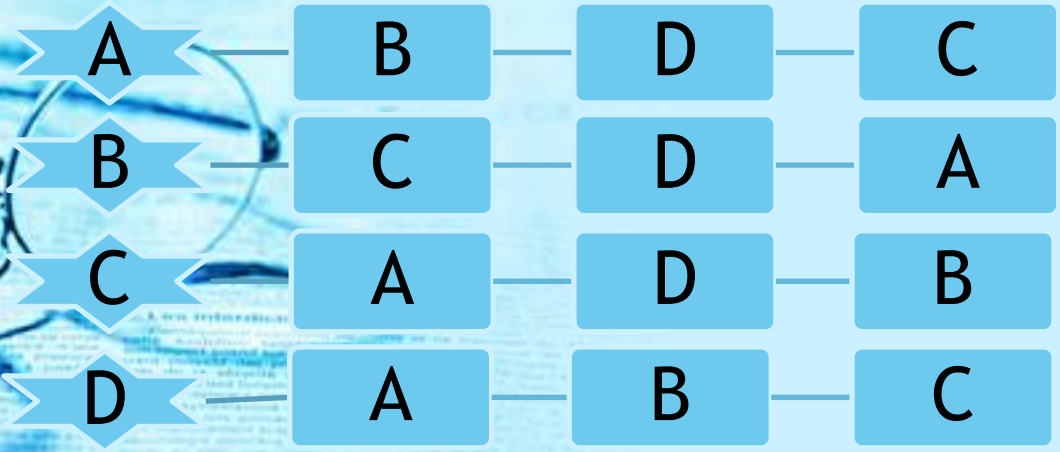


Структуры данных для представления графов

- Матрица смежности

	A	B	C	D
A		1	2	3
B			4	5
C				6
D				

- Список примыканий

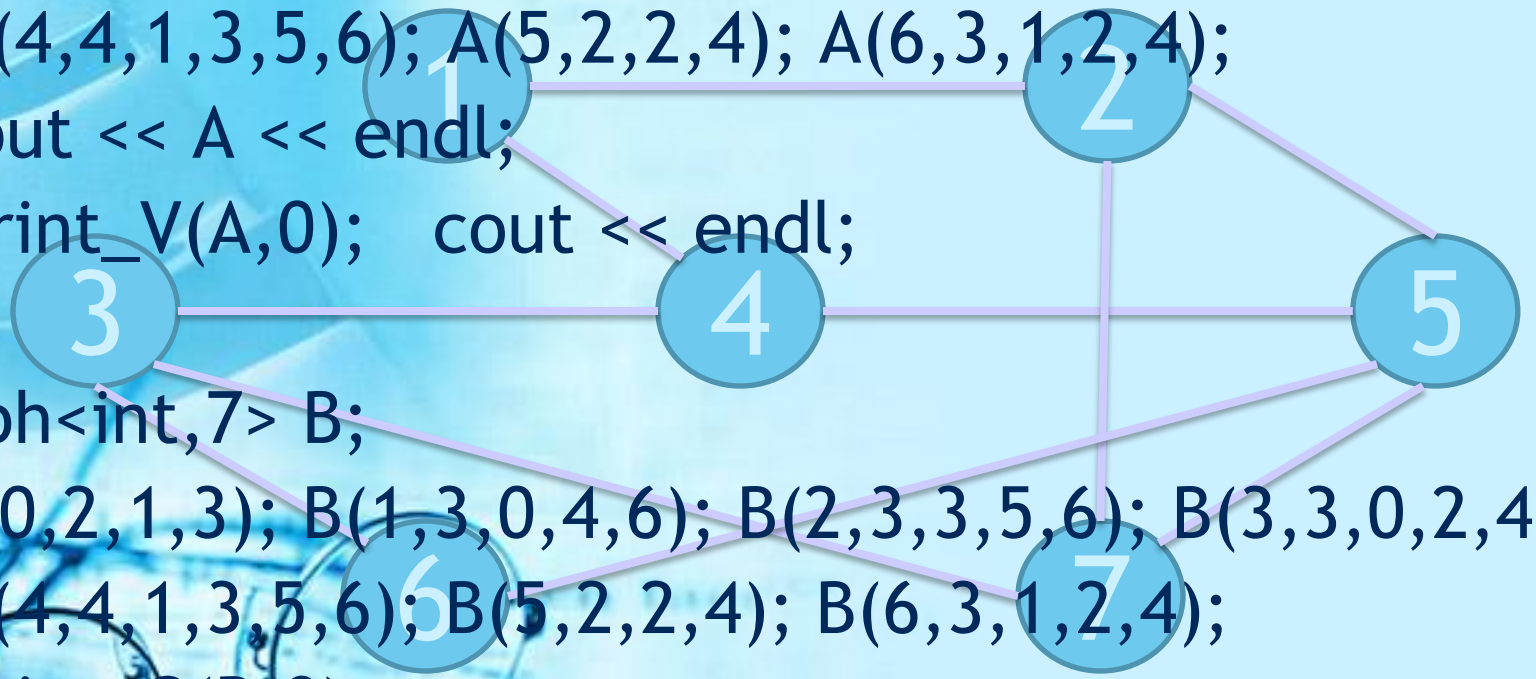


Описание шаблона типа Graph

```
template <typename T, int N> class Graph {
        List<T> S[N]; int B[N];
public:
Graph() { for(int i=0;i<N;i++) B[i]=0; }
List<T>& operator()(int n) { return S[n]; }
void operator()(int n,int m, T f, ...) {
   va_list pt; va_start(pt,f); T v=f;
   for(int i=0;i<m;i++) { S[n].push_back(v); v=va_arg(pt,T); }
   va_end(pt); }
int& operator[](int n) { return B[n]; }
friend ostream& operator<<(ostream& a, Graph<T,N> b) {
for(int i=0;i<N;i++) a << i << ": " << b(i) << endl;  return a;}
};
```

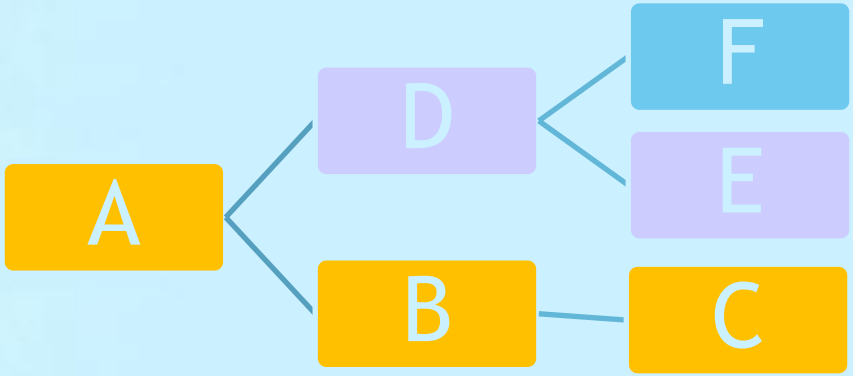
Основная программа

```
int main() {
Graph<int,7> A;
   A(0,2,1,3); A(1,3,0,4,6); A(2,3,3,5,6); A(3,3,0,2,4);
    A(4,4,1,3,5,6); A(5,2,2,4); A(6,3,1,2,4);
   cout << A << endl;
    print_V(A,0);    cout << endl;

Graph<int,7> B;
   B(0,2,1,3); B(1,3,0,4,6); B(2,3,3,5,6); B(3,3,0,2,4);
    B(4,4,1,3,5,6); B(5,2,2,4); B(6,3,1,2,4);
    print_G(B,0);
    return 0; }
```

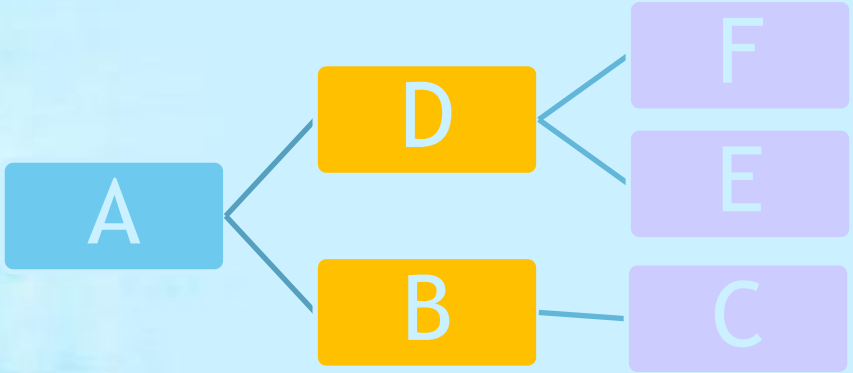


Алгоритмы обхода

- В глубину



- В ширину



Dev-C++ 4.9.9.2

Файл Правка Поиск Вид Проект Выполнить Отладка Сервис CVS Окно Справка

Создать Вставить Переключить Перейти

Проект Классы Отладка

[*] g1.cpp SQL.h

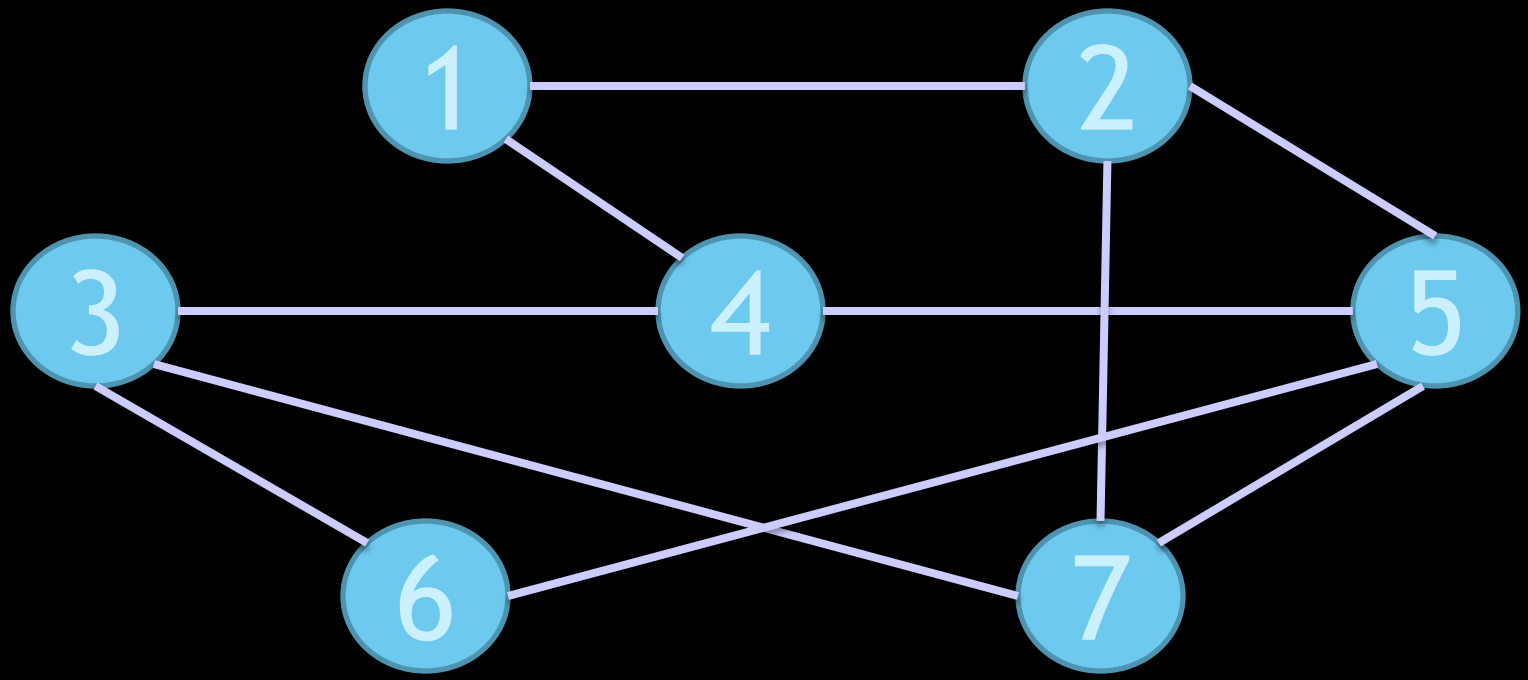
```
void print_V(Graph<int,7> A, int k) {
Stack<int> Q;
    Q.push(k); A[k]++;
    while(!Q.empty()) {
         while(!A(k).empty()) {
               int i=A(k).pop_front();
               if(!A[i]) { A[i]++; Q.push(i);
                           cout << '(' << k+1 << ',' << i+1 << ")\n"; k=i;}
         }
    k=Q.pop(); } // end while
}

void print_G(Graph<int,7> A, int k) {
Queue<int> Q;
    Q.put(k); A[k]++;
    while(!Q.empty()) {
          k=Q.get();
          while(!A(k).empty()) {
                int i=A(k).pop_front();
                if(!A[i]) { Q.put(i); A[i]++;
                            cout << '(' << k+1 << ',' << i+1 << ")\n"; }
          }
    }   // end while
}
```

Компилятор Ресурсы Журнал компиляции Отладка Результаты поиска

42: 2 Изменен Вставка 60 строк в файле

D:\A1\Lect2013\Exmp\12\g1.exe

```
0: 1
3

1: 0
4
6

2: 3
5
6

3: 0
2
4

4: 1
3
5
6

5: 2
4

6: 1
2
4


(1,2)
(2,5)
(5,4)
(4,3)
(3,6)
(3,7)

(1,2)
(1,4)
(2,5)
(2,7)
(4,3)
(5,6)
```



Построение Минимального Остового Дерева

- МОД связного взвешенного графа – это его связный подграф, состоящий из всех вершин исходного графа и некоторых его ребер. Причем сумма весов ребер должна быть минимальной.
- Трудоемкость полного перебора O(exp(N))
- Попробуем построить МОД на основе «жадного» алгоритма: В каждый момент будем использовать лишь часть данных и выбирать наилучший вариант на основе этой части.

Алгоритм Дейкстры-Прима

$V = M \cup M' \cup R$

M – множество вершин включенных в МОД

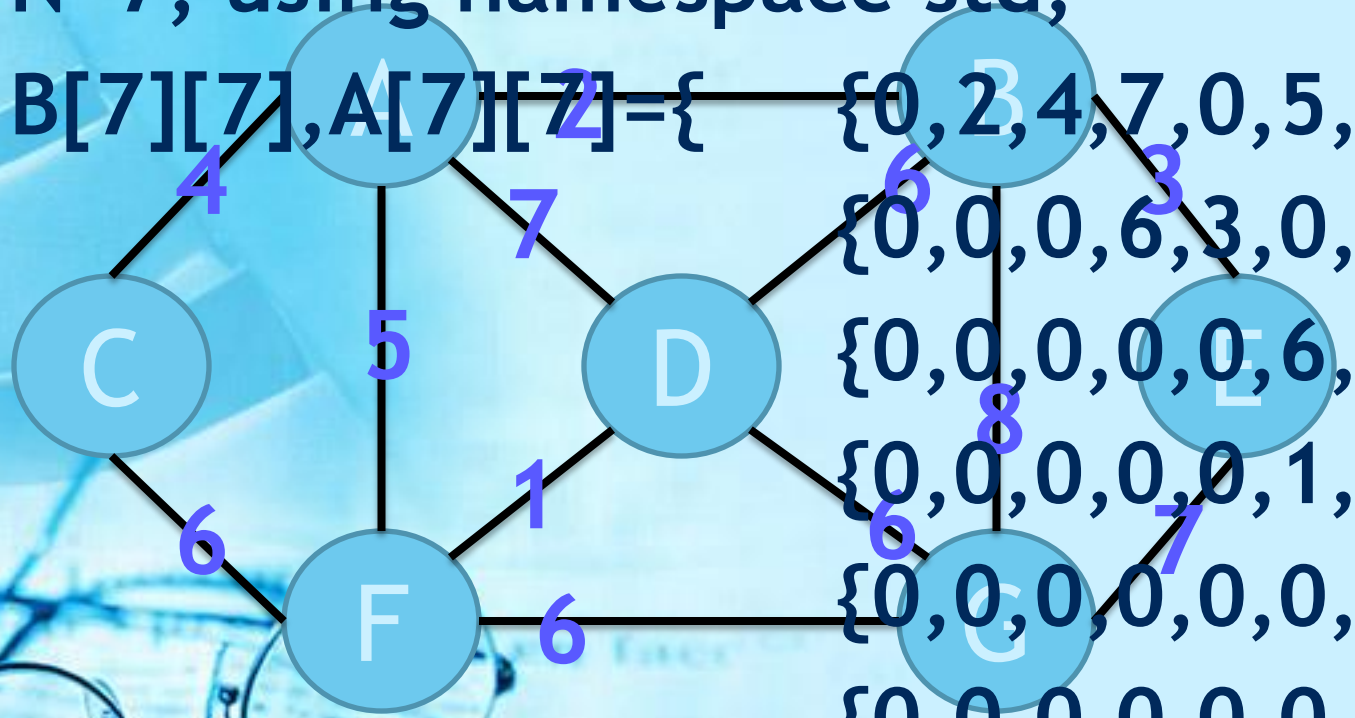
M’ – множество вершин окаймляющих M

R = V\M\M’ – не рассмотренные вершины

- **Выбрать начальный узел**
- **Сформировать начальную кайму**
- **While $M' \neq \varnothing$ do**
  - **Выбрать min mm’: $m \in M$, $m' \in M'$ и $M \leftarrow M \cup \{m'\}$**
  - **Дополнить кайму и скорректировать матрицу смежности**
- **End while**

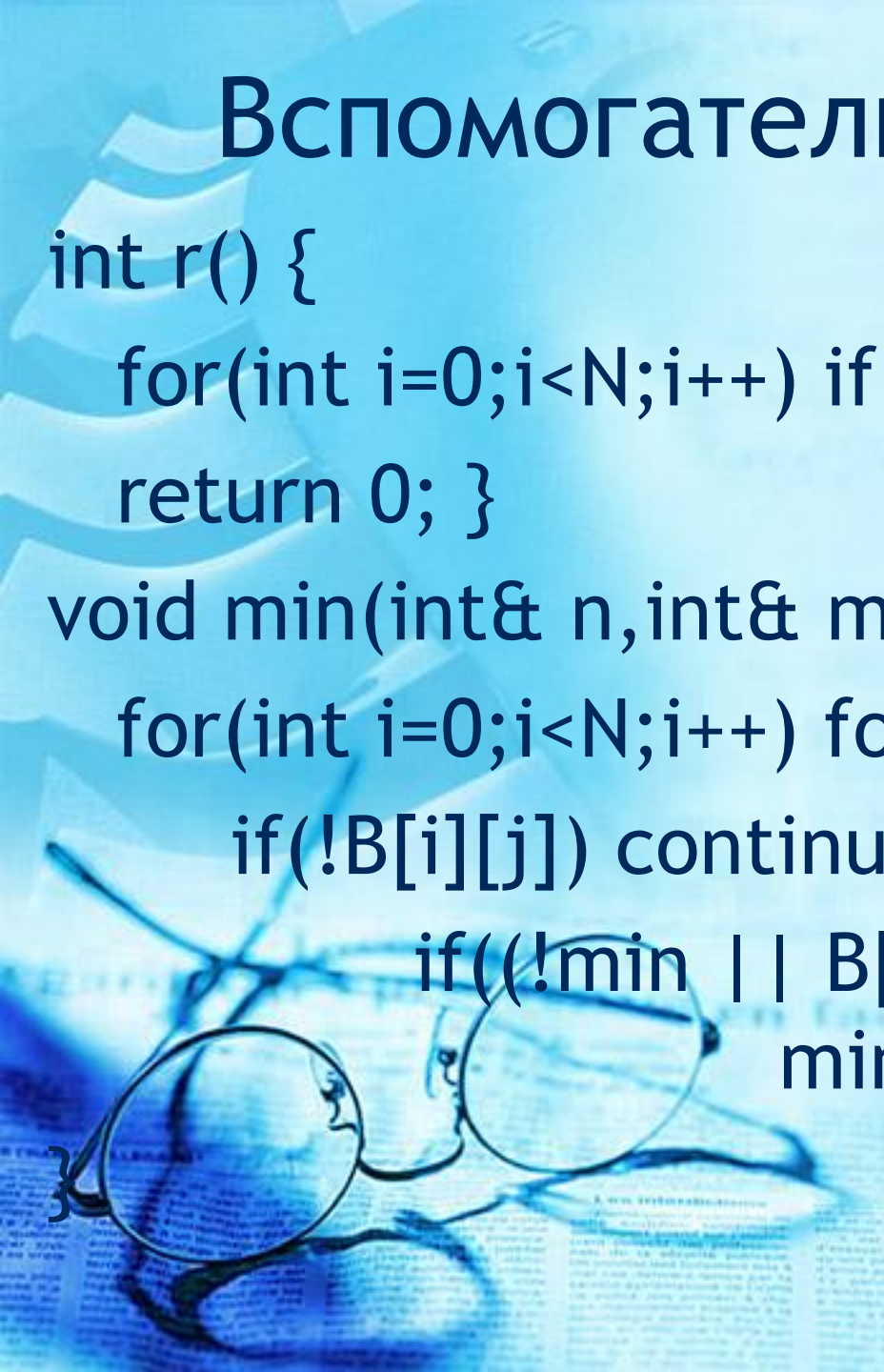
Задание матрицы смежности

```
#include <iostream>
int N=7; using namespace std;
int B[7][7],A[7][7]={   {0,2,4,7,0,5,0},
                        {0,0,0,6,3,0,8},
                        {0,0,0,0,0,6,0},
                        {0,0,0,0,0,1,6},
                        {0,0,0,0,0,0,7},
                        {0,0,0,0,0,0,6},
                        {0,0,0,0,0,0,0}};
```



Вспомогательные функции

```
int r() {
  for(int i=0;i<N;i++) if(!A[i][i]) return 1;
  return 0; }
void min(int& n,int& m) { int min=0;
  for(int i=0;i<N;i++) for(int j=0;j<N;j++)
     if(!B[i][j]) continue; else
           if((!min || B[i][j]<min)&&!A[j][j])
                  min=B[n=i][m=j];
}
```



Основная программа

```
int main() { int k=0,i,j,n,m;
for(n=0;n<N;n++) for(m=0;m<N;m++) {
        B[n][m]=0; if (n>m) A[n][m]=A[m][n]; }
for(m=0;m<N;m++) B[k][m]=A[k][m]; A[k][k]=1;
while( r() ) { min(n,k); A[k][k]=1;
for(m=0;m<N;m++) B[k][m]=!A[m][m]?A[k][m]:0;
   cout << char('A'+n) << char('A'+k) << endl;
   for(m=0;m<N;m++) for(k=n=0;n<N;n++)
        if(B[n][m]) if(!k) k=B[i=n][j=m]; else
                if(B[n][m]>k) B[n][m]=0; else {
                        B[i][j]=0; k=B[i=n][j=m]; }
   } /*end while*/ return 0; }
```

Результат

AB

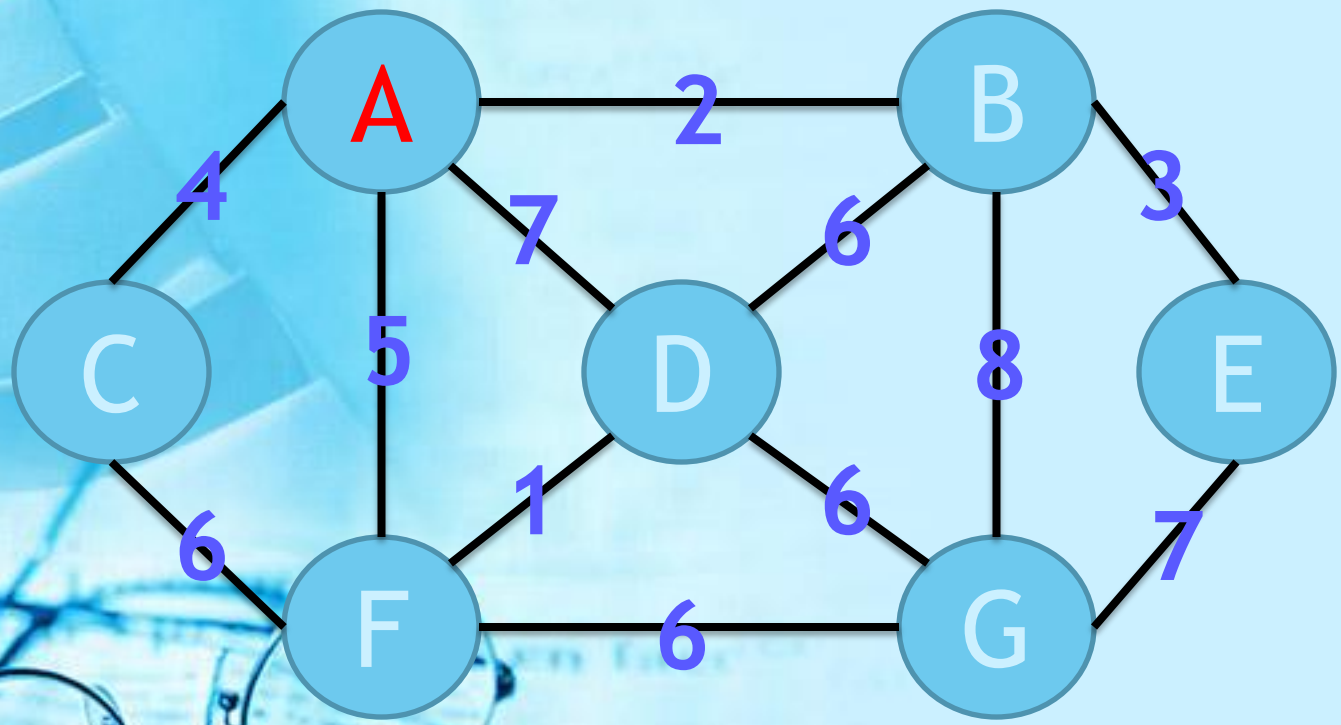
BE

AC

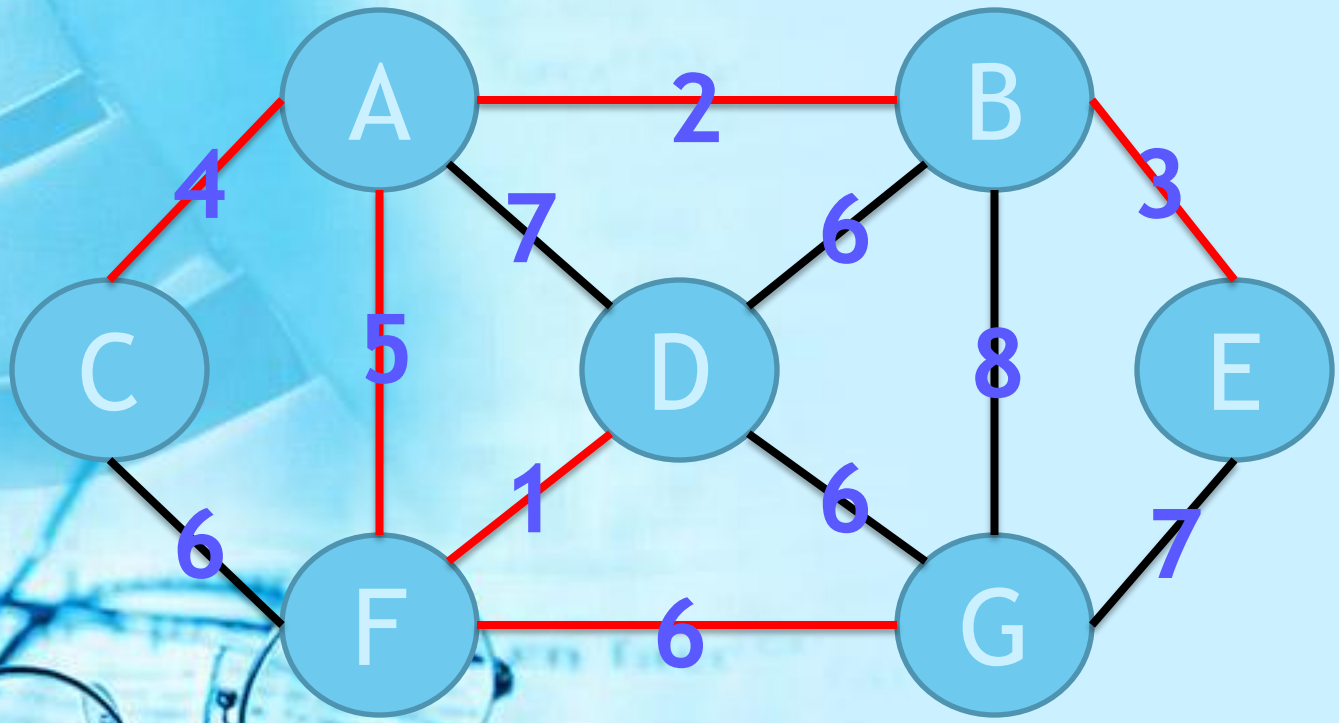
AF

FD

FG



Поиск кратчайшего пути



Алгоритм Дейкстры

- Выбрать начальную вершину
- Создать кайму
- While целевая вершина не достигнута do
  - Выбрать из каймы вершину ближайшую к начальной
  - Добавить эту вершину к дереву пути
  - Добавить в кайму вершины, соседние к добавленной
  - For для каждой вершины каймы do
    - Выбрать min ребро, соединяющее её с деревом
  - End for
- End while

Вспомогательные функции

```
int r() { for(int i=0;i<N;i++) if(!A[i][i]) return 1; return 0;}

int min() { int r,m=0;
   for(int i=0;i<N;i++) if(!B[i][i]) continue; else
        if((!m || B[i][i]<m) && !A[i][i]) m=B[r=i][i];
   return r; }

void print_B() { cout << endl;
   for(int n=0;n<N;n++) { cout << char('A'+n) <<": ";
   for(int m=0;m<N;m++) cout <<(n==m?0:B[n][m]) << ' ';
   cout << '\t' << B[n][n] << endl; } }
```

Основная программа

```
int main() { int k=0,d,n,m;
for(n=0;n<N;n++) for(m=0;m<N;m++) { B[n][m]=0;
   if (n>m) A[n][m]=A[m][n]; }
for(m=0;m<N;m++) B[m][m]=B[k][m]=A[k][m]; A[k][k]=1;
while( r() ) { k=min(); A[k][k]=1;
for(m=0;m<N;m++)if(k==m||!A[k][m]||A[m][m])continue;
   else { d=B[k][k]+A[k][m];
      if(!B[m][m]) { B[m][m]=d; B[k][m]=A[k][m]; } else
         if(B[m][m]<=d) B[k][m]=0; else {
            B[m][m]=d; B[k][m]=A[k][m];
   for(n=0;n<N;n++)
            if(n!=m && n!=k && B[n][m]) B[n][m]=0; } }
   } //end while
print_B(); return 0; }
```

Результат

A: 0 2 4 0 0 5 0 0

B: 0 0 0 0 3 0 8 2

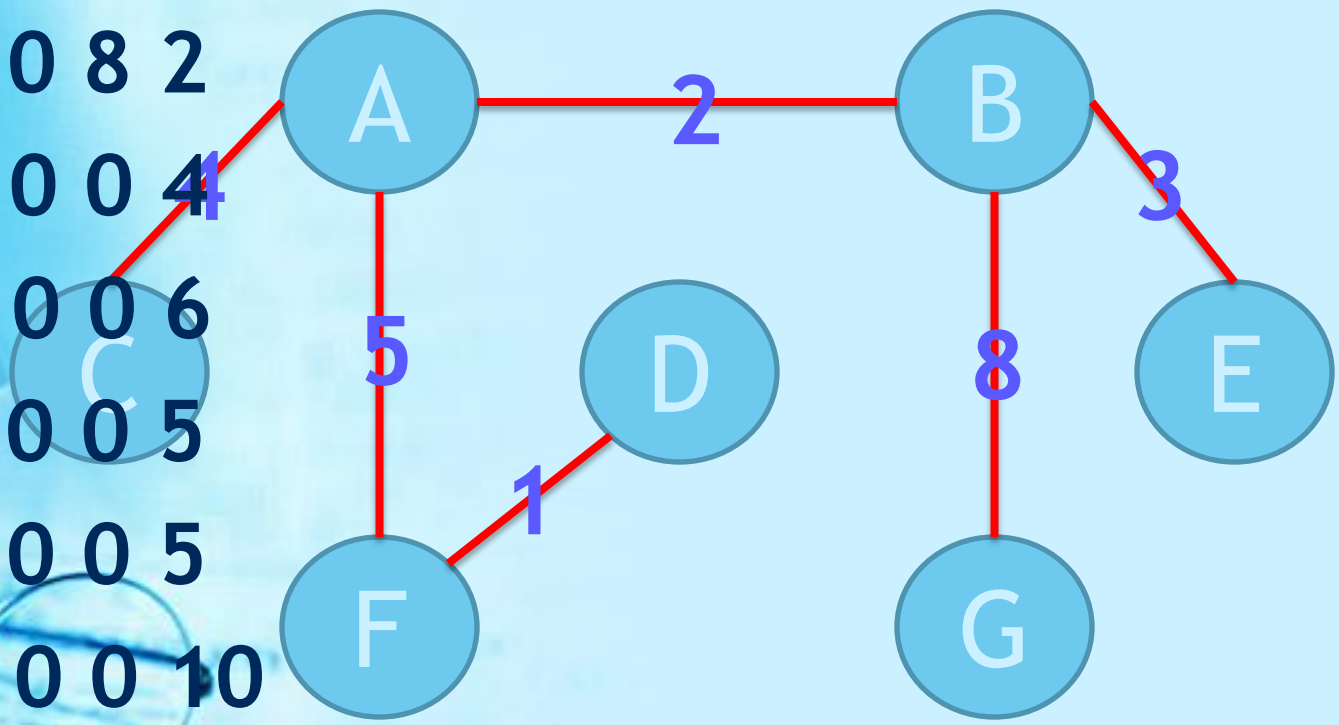
C: 0 0 0 0 0 0 0 4

D: 0 0 0 0 0 0 0 6

E: 0 0 0 0 0 0 0 5

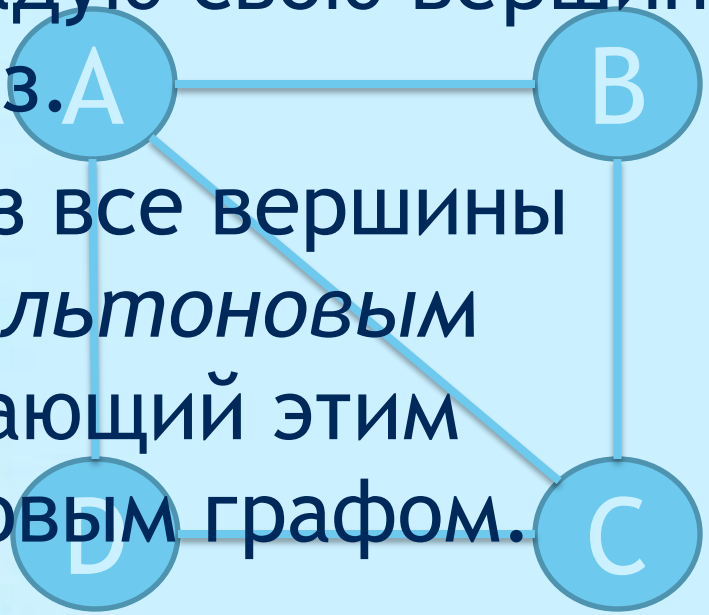
F: 0 0 0 1 0 0 0 5

G: 0 0 0 0 0 0 0 10

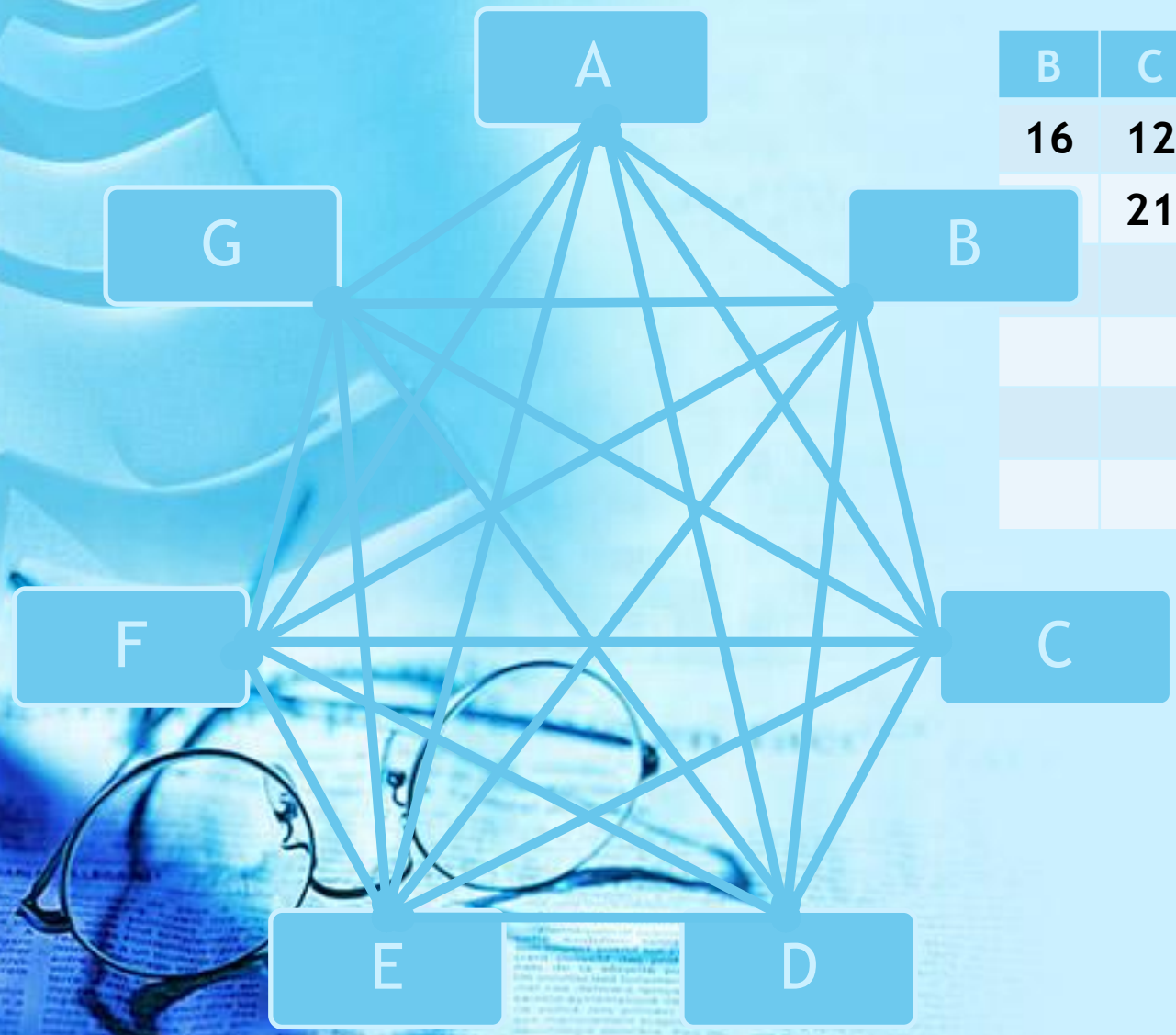


Циклы

- *Цикл* – это замкнутый маршрут проходящий, через каждую свою вершину и ребро только один раз.
- Цикл проходящий через все вершины графа называется *гамильтоновым* циклом. А граф, обладающий этим свойством – гамильтоновым графом.
- Граф, обладающий замкнутым маршрутом, проходящим через все ребра графа ровно один раз, - называется эйлеровым.



Задача коммивояжера



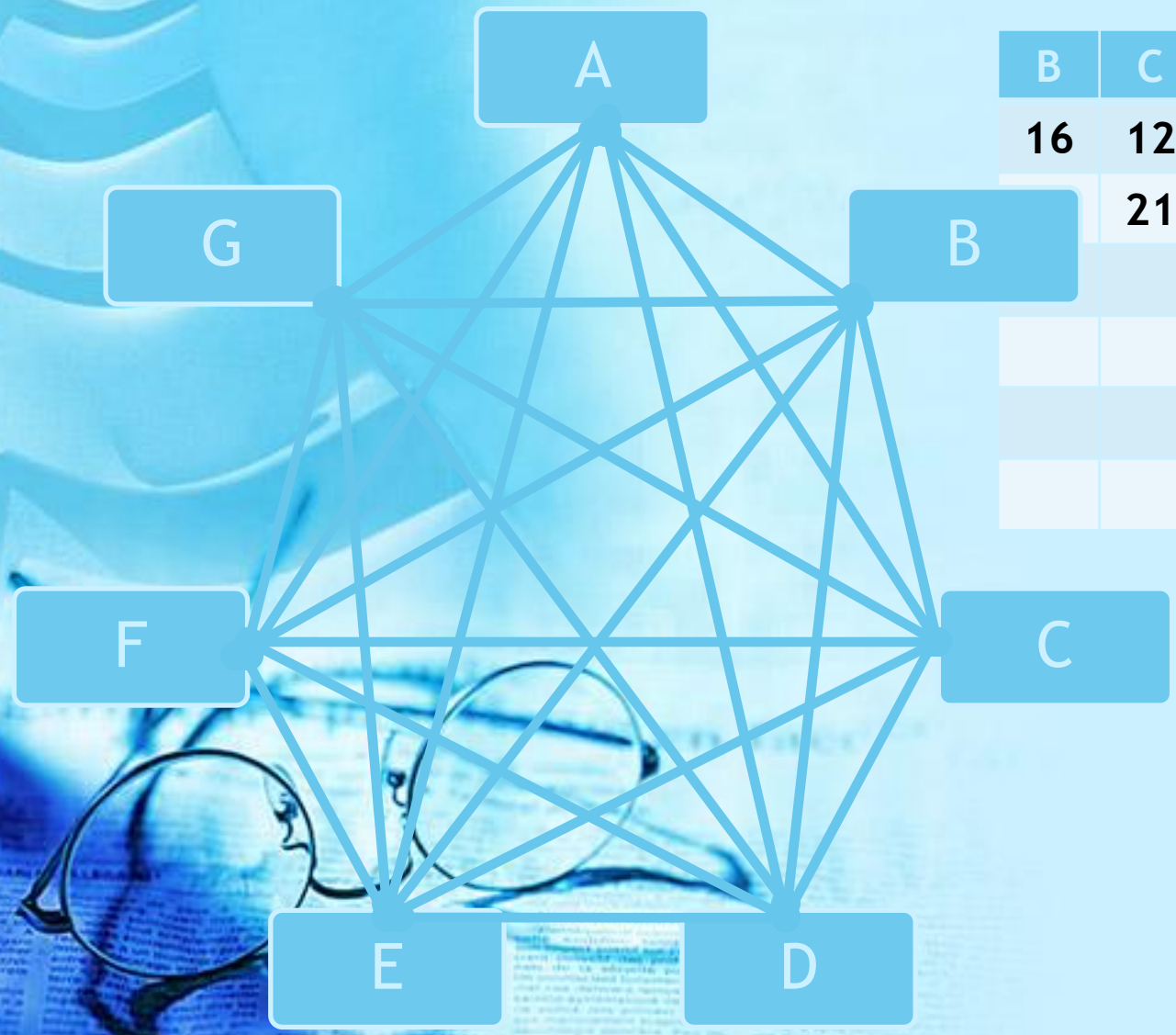
B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

Жадный алгоритм поиска пути

Пока не найден полный цикл

- Выбрать ребро с минимальным весом и добавить его в путь, при условии что:
  - ✓ Рассматриваемое ребро не является третьим, инцидентным какой-либо вершине уже включенной в путь;
  - ✓ Добавление указанного ребра не приводит к образованию неполного цикла.

Задача коммивояжера



B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

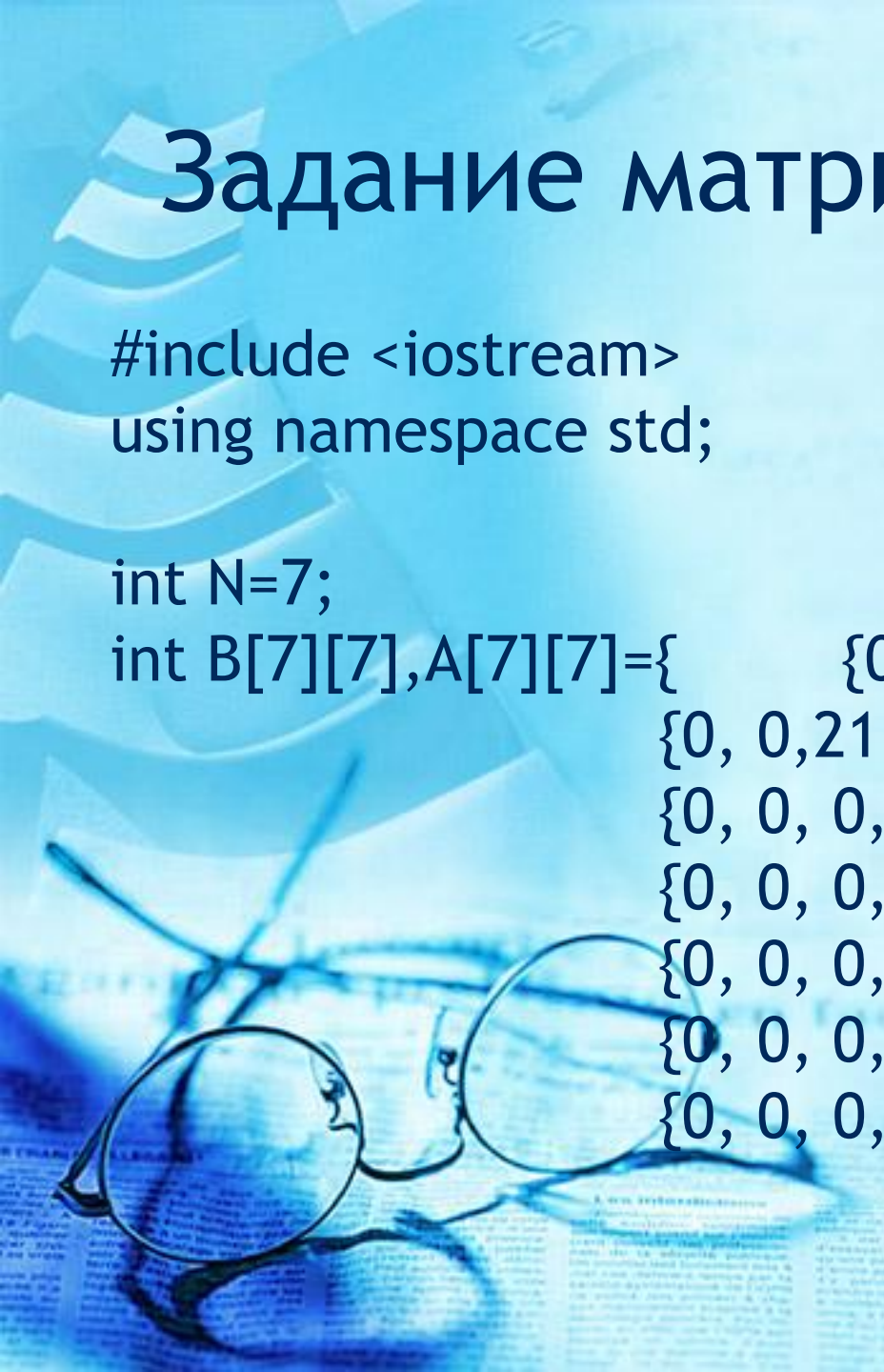
Задача коммивояжера

B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

Длина найденного пути - 53

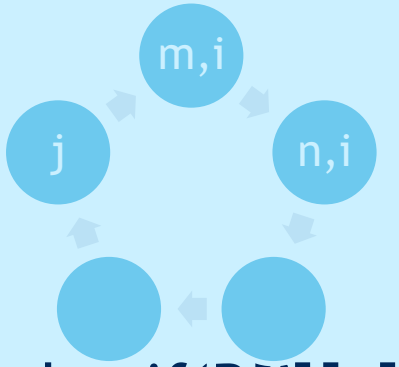
Задание матрицы смежности

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

int N=7;
int B[7][7],A[7][7]={       {0,16,12,13,6,7,11},
                            {0, 0,21,18,8,19,5},
                            {0, 0, 0,20,1,3,15},
                            {0, 0, 0,0,14,10,4},
                            {0, 0, 0,0, 0, 2,7},
                            {0, 0, 0,0, 0, 0,9},
                            {0, 0, 0,0, 0, 0,0}};
```



Проверка на возникновение цикла

```
int r() { for(int i=0;i<N;i++) if(A[i][i]!=2) return 1; return 0;
    }

int loop(int i, int j) { int n,m=i;
    if(A[i][i]<2 || A[j][j]<2) return 0;
    do {
    for(n=0;n<N;n++) if(n==m) continue; else if(B[i][n])
    break; m=i; i=n; }
    while(A[n][n]==2 && i!=j);
    if(i==j) if( !r() ) return 0; else return 1; else return 0;
}
```

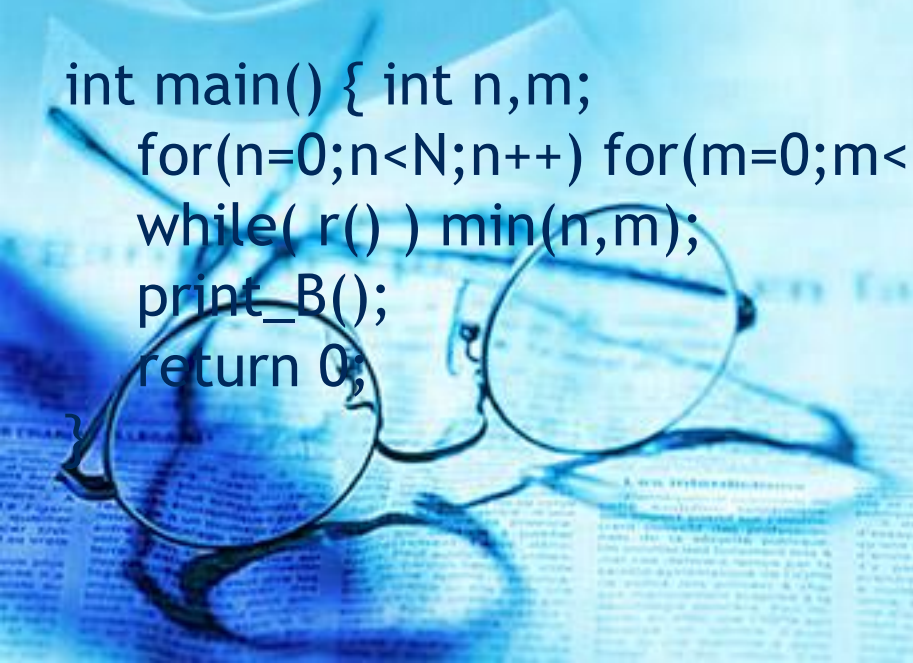


Выбор следующего ребра

```
void min(int& i,int& j) {
int n,m,k=0;
   while(!k) {
   for(m=1;m<N;m++) for(n=0;n<m;n++)
        if(A[n][m]>0 && (!k || A[n][m]<k) &&
                A[n][n]<2 && A[m][m]<2) k=A[i=n][j=m];
   A[i][i]++; A[j][j]++;
   if(loop(i,j)) { A[i][i]--; A[j][j]--; A[i][j]=k=0; }
        else { B[i][j]=B[j][i]=A[i][j]; A[i][j]=0; }
   }
}
```

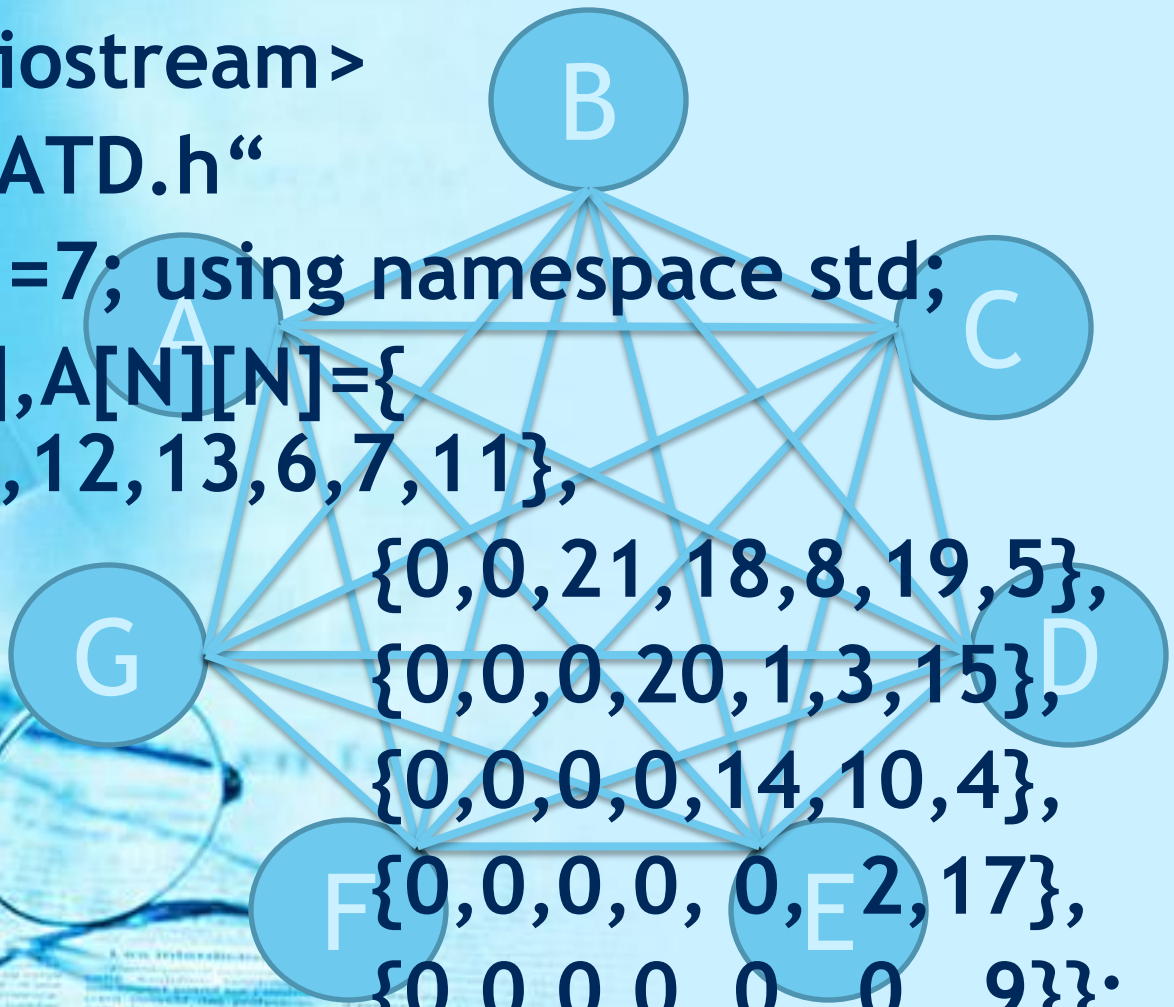
Основная программа

```
void print_B() { int s=0;          cout << endl;
    for(int n=0;n<N;n++) { cout << char('A'+n) <<":\t";
          for(int m=0;m<N;m++) { s+=B[n][m]; cout << B[n][m] << ' '; }
          cout << endl; }
    cout << "Length= " << s/2 << endl;
}

int main() { int n,m;
    for(n=0;n<N;n++) for(m=0;m<N;m++) B[n][m]=0;
    while( r() ) min(n,m);
    print_B();
    return 0;
}
```

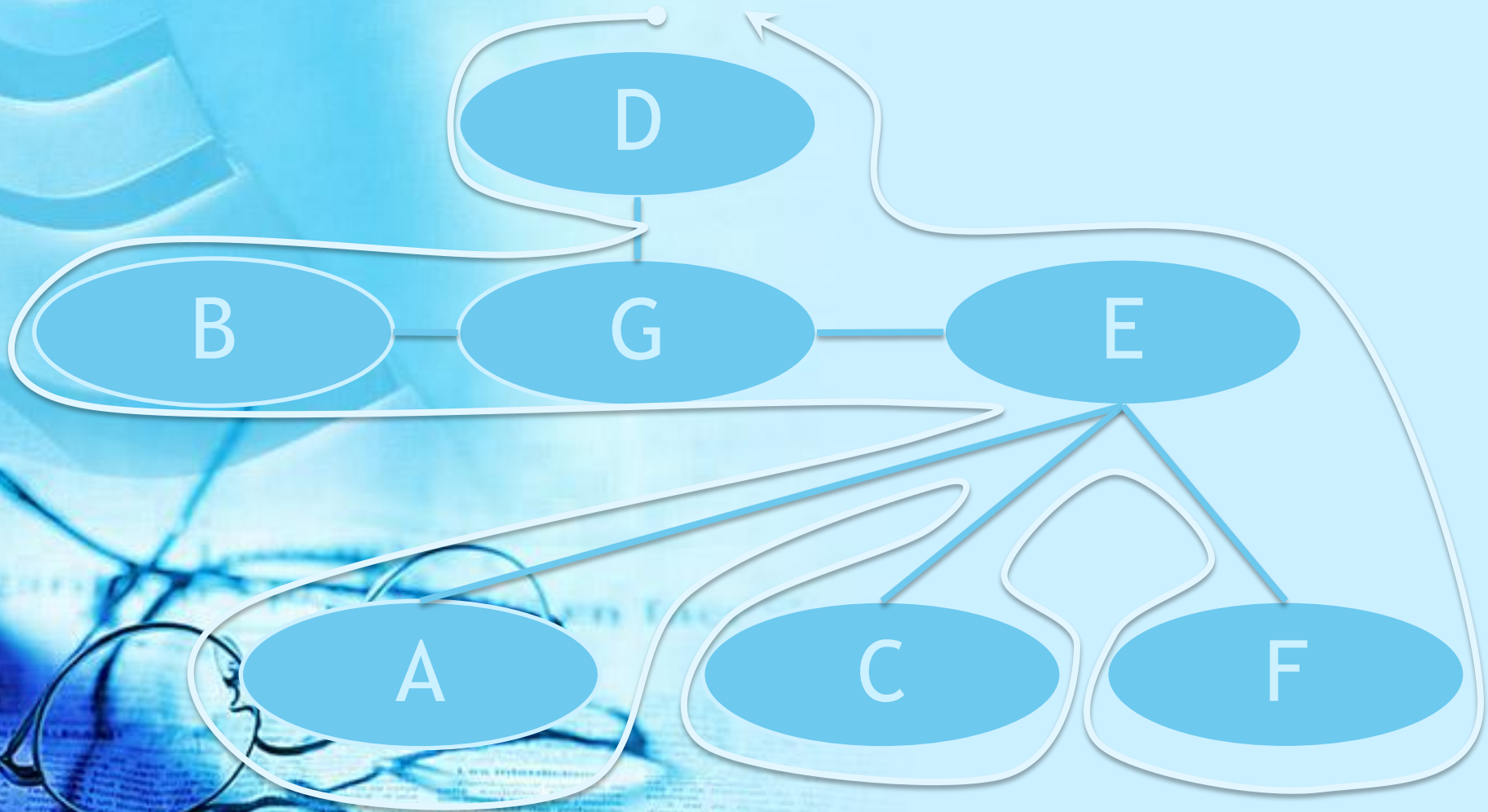


Матрица примыкания:
C(u,w) ≤ C(u,v) + C(v,w)

```
#include <iostream>
#include “ATD.h“
const int N=7; using namespace std;
int B[N][N],A[N][N]={
        {0,16,12,13,6,7,11},
        {0,0,21,18,8,19,5},
        {0,0,0,20,1,3,15},
        {0,0,0,0,14,10,4},
        {0,0,0,0, 0, 2,17},
        {0,0,0,0, 0, 0, 9}};
```



Обход МОД



Построение МОД и гамильтонова цикла на его основе

```
void gloop(int k,Queue<int>& a) {  a.put(k);
   for(int m=0;m<N;m++) if(B[k][m]) gloop(m,a);
   }
void main() { int k,m,i=0;
Queue<int> a;
   mst(3); gloop(3,a); k=a.get(); a.put(k);
   cout << char('A'+k);
   while(!a.empty()) { m=a.get(); i+=A[k][m];
   k=m;
        cout << "-->" << char('A'+k); }
   cout << "\nlen= " << i << endl; }
```

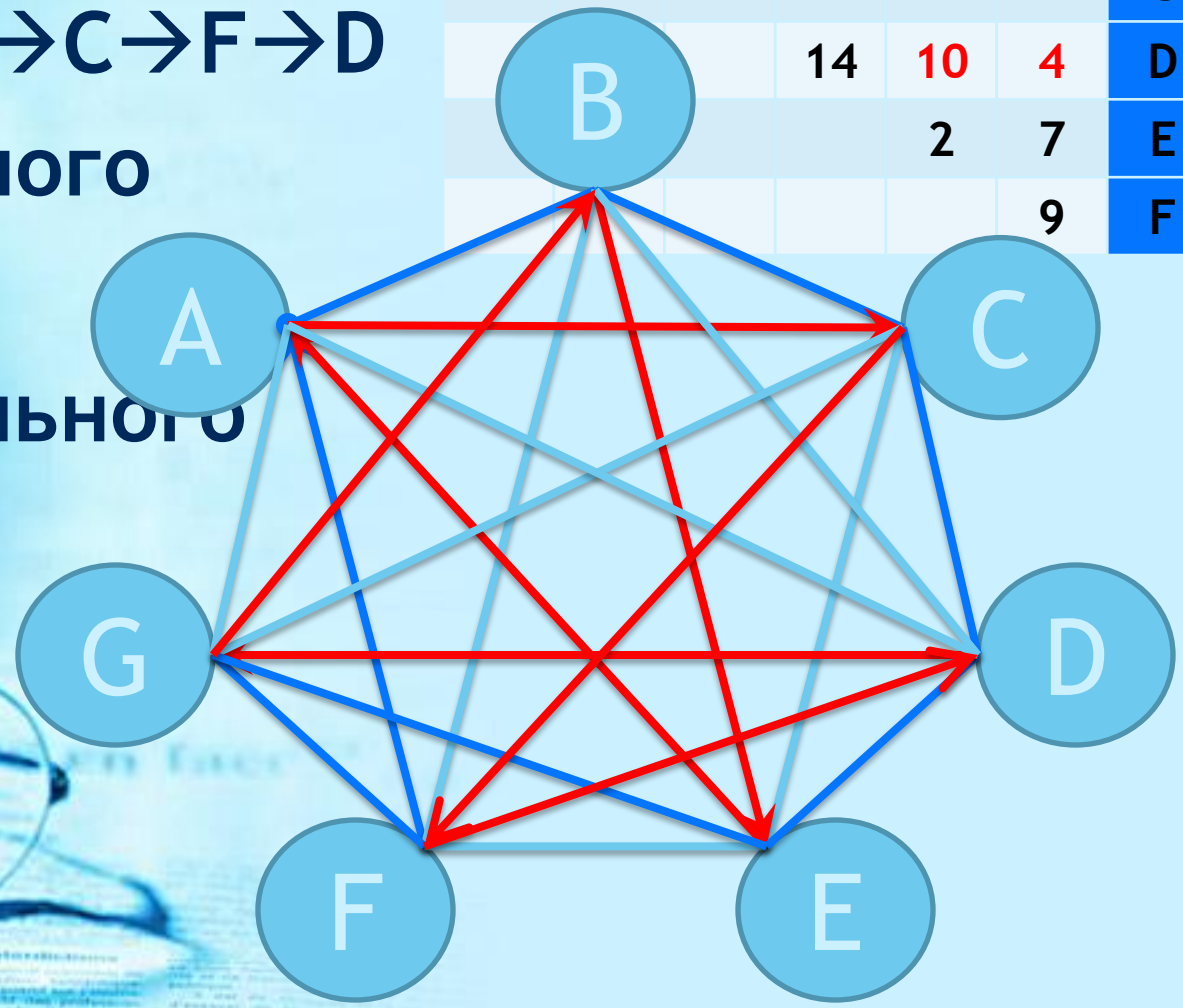
Гамильтонов цикл

B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

D→G→B→E→A→C→F→D

Длина найденного пути - 48

Длина оптимального пути - 41



Гамильтонов цикл

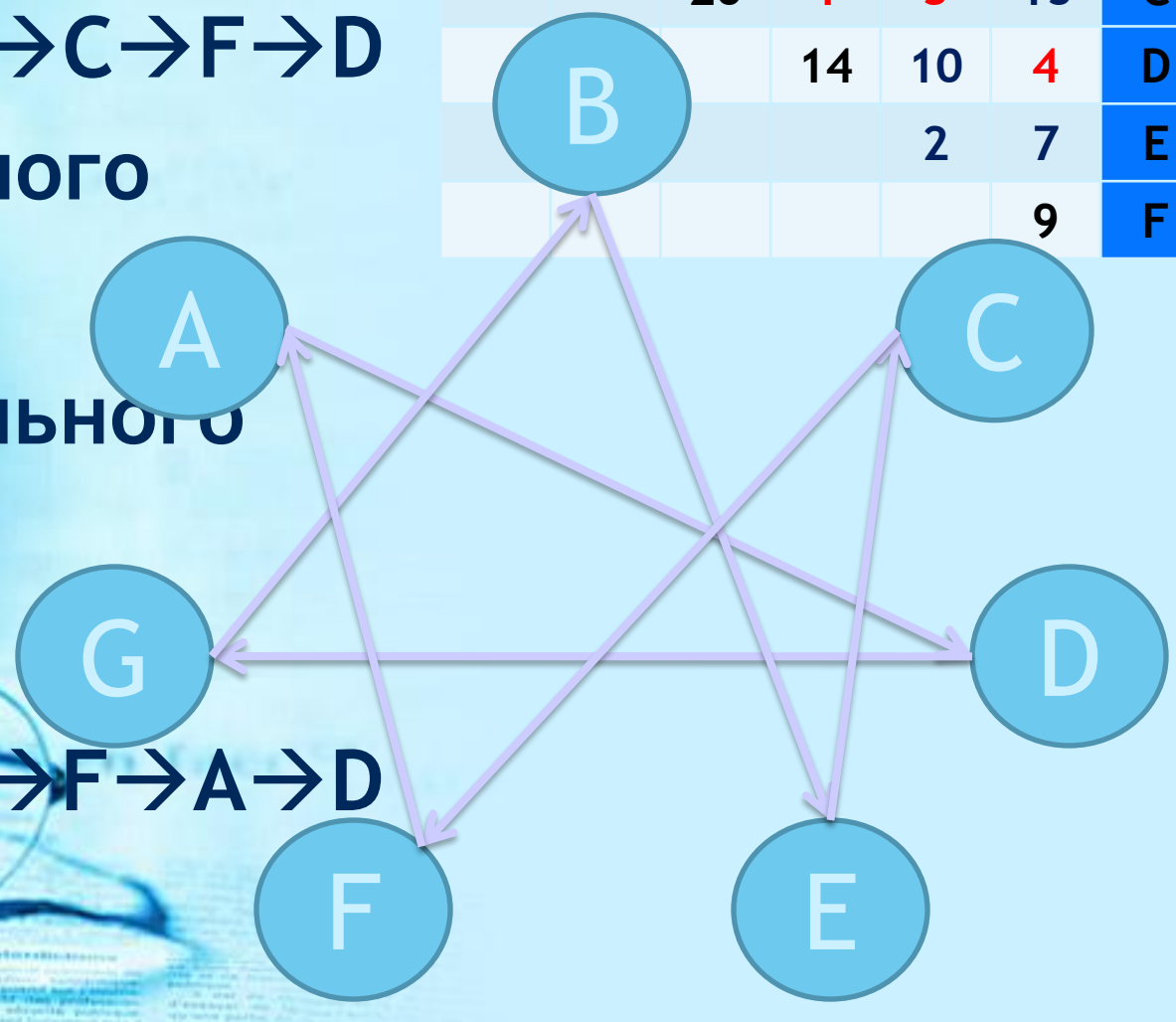
B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

D→G→B→E→A→C→F→D

Длина найденного пути - 48

Длина оптимального пути - 41

D→G→B→E→C→F→A→D



Литература:

- Р.Седжвик «Алгоритмы на C++» -М.: «И.Д. Вильямс», 2011. ISBN 978-5-8459-1650-1
- Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн «Алгоритмы: построение и анализ» -М.: «И.Д. Вильямс», 2015. ISBN 5-8459-0857-4
- Н.Вирт «Алгоритмы и структуры данных» -СПб.: Невский Диалект, 2001. ISBN 5-7940-0065-1