Графы

Алгоритмы
и алгоритмические языки

goo.gl/c8pyqx

Лекция 12, 23 ноября, 2018

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru

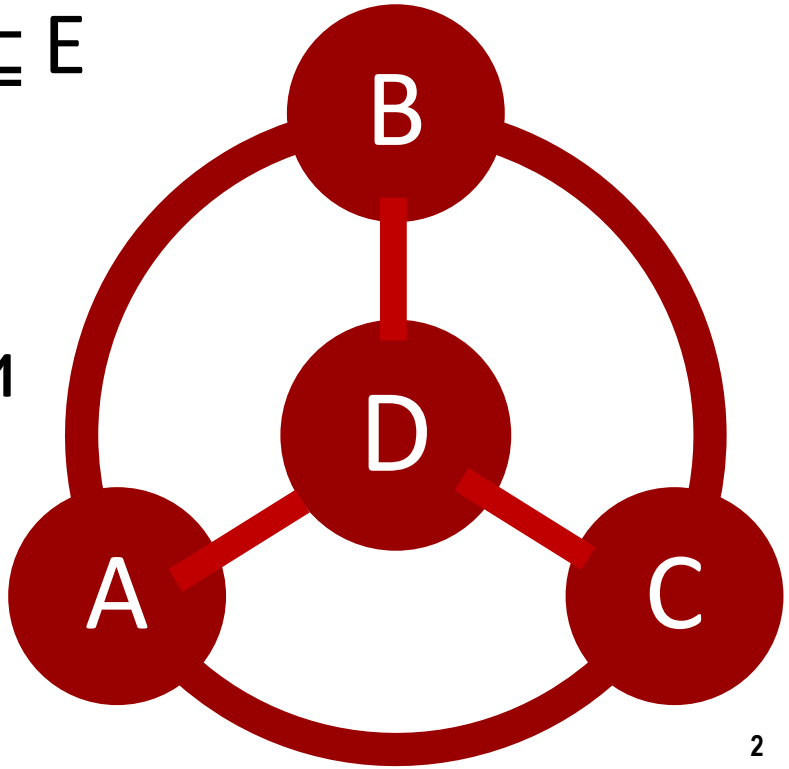
http://cs.mipt.ru/wp/?page_id=6077

Основные понятия

- граф G = (V,E)

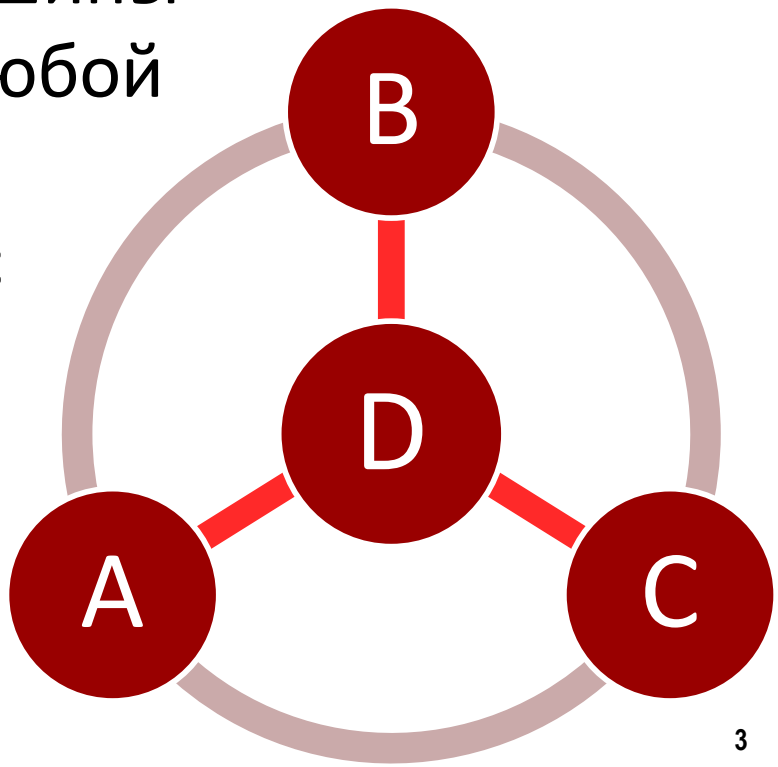
  V – множество вершин {A, B, C, D, …}

  E – множество ребер {AB, BC, CD, …}
- в *полном* графе любая пара вершин есть ребро
- подграф G g=(v,e): v $\subseteq$ V и e $\subseteq$ E
- (не)направленость рёбер
- число ненаправленных рёбер полного графа с N вершинами равно N(N-1)/2



Основные понятия

- *Взвешеный* граф имеет вес каждого ребра
- Путь P_{AC} из A в C – последовательность связанных рёбрами вершин графа
- В *простом* пути различны все вершины
- *Связный* граф имеет путь между любой парой вершин
- *Цикл* – путь, где конец совпадает с началом
- В *простом* цикле различны все вершины, кроме начала и конца

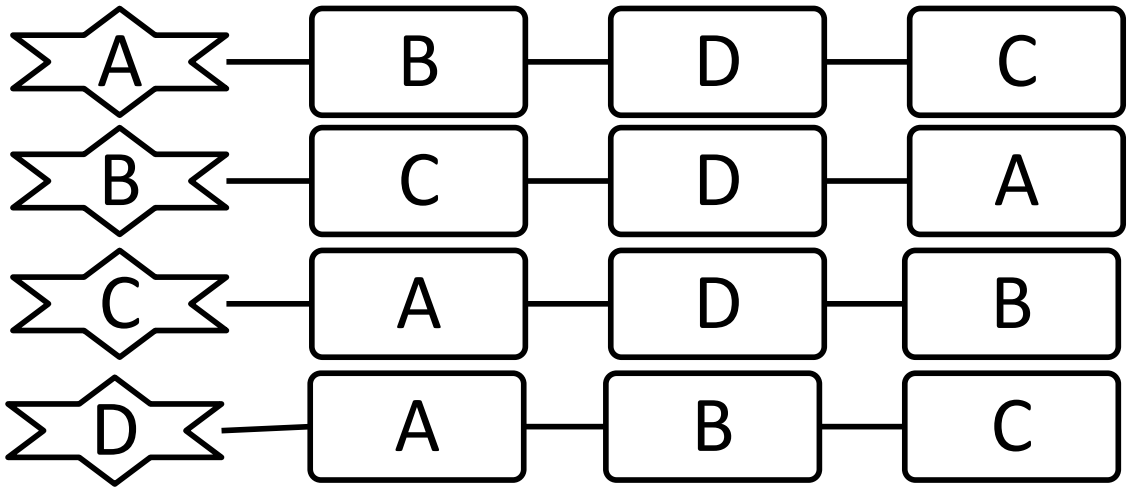


Структуры данных для представления графов

- Матрица смежности

	A	B	C	D
A	**0**	1	2	3
B		**0**	4	5
C			**0**	6
D				**0**

- Список примыканий



[*] graf.c [*] SQL.h

```c
6  typedef struct Item { T node; struct Item *next; } *Link_Item;
7  Link_Item Item_Create(T elem, Link_Item n) {
8      Link_Item rab = (Link_Item)calloc(1, sizeof(struct Item));
9      rab->node=elem; rab->next=n; return rab; }
10 T Item_get_node(Link_Item a) { return a->node; }
11 Link_Item Item_get_next(Link_Item a) { return a->next; }
12 void Item_Delete(Link_Item a) { free(a); }
13
14 typedef struct Stack { Link_Item Top; } *Link_Stack;
15 void Stack_ini(Link_Stack a) { a->Top = NULL; }
16 int Stack_Is_Empty(const Link_Stack a) { return a->Top==NULL; }
17 void Stack_push(Link_Stack a, T elem) { a->Top=Item_Create(elem,a->Top); }
18 T Stack_pop(Link_Stack a) {
19     if(Stack_Is_Empty(a)) ERR("Stack::pop: empty stack");
20     Link_Item p = a->Top; T rab = Item_get_node(p); a->Top = Item_get_next(p);
21     Item_Delete(p); return rab; }
22
23 typedef struct Queue { Link_Item Head; Link_Item Tail; } *Link_Queue;
24 void Queue_ini(Link_Queue a) { a->Head = a->Tail = NULL; }
25 int Queue_Is_Empty(const Link_Queue a) { return a->Head==NULL; }
26 void Queue_put(Link_Queue a, T elem) {
27     if(Queue_Is_Empty(a)) a->Tail = a->Head = Item_Create(elem,NULL);
28     else a->Tail = (a->Tail->next = Item_Create(elem,NULL)); }
29 T Queue_get(Link_Queue a) {
30     if(Queue_Is_Empty(a)) ERR("Queue::get: gueue is empty");
31     Link_Item p = a->Head; T rab = Item_get_node(p); a->Head=Item_get_next(p);
32     Item_Delete(p); if(Queue_Is_Empty(a)) a->Tail=NULL; return rab; }
33
34 typedef struct List { Link_Item Front; Link_Item Back; } *Link_List;
35 void List_ini(Link_List a) { a->Front = a->Back = NULL; }
```

Компилятор | Ресурсы | Журнал компиляции | Отладка | Результаты поиска

Данные типа Graf

```
#include <stdio.h>
#include <stdarg.h>

typedef int T;

#include "SQL.h"

void List_print(Link_List a) {
   Link_Item p = a->Front;
   while(p) { printf("%d ", Item_get_node(p)+1);
      p=Item_get_next(p); }
}

typedef struct Graf {
   Link_List S; int N; } *Link_Graf;
```

Методы типа Graf

```
Link_Graf Graf_create(int n) { int i;
   Link_Graf p = (Link_Graf)calloc(1,sizeof(struct Graf));
   p->S = (Link_List)calloc(p->N=n,sizeof(struct List));
   for(i=0;i<n;i++) List_ini(&p->S[i]); p->N=n; return p; }
void Graf_delete(Link_Graf *a) { free((*a)->S); free(*a);
  *a=NULL; }

Link_List Graf_Get_node(Link_Graf a, T name) {
   return &a->S[name]; }
void Graf_Set_node(Link_Graf a,T name,...) { T v;
  va_list p; va_start(p,name);
  while(v=va_arg(p,T)) List_Insert_back(&a->S[name-1],v-1);
  va_end(p); }

void Graf_print(Link_Graf a) {int i;
   for(i=0;i<a->N;i++) { printf("%d: ",i+1);
   List_print(&a->S[i]); putchar('\n'); } }
```

D:\Disk_S\Lect2016\Exmp\12\C\graf.c - Dev-C++ 5.11

Файл Правка Поиск Вид Проект Выполнить Сервис AStyle Окно Справка

TDM-GCC 4.9.2 64-bit Release

(globals)

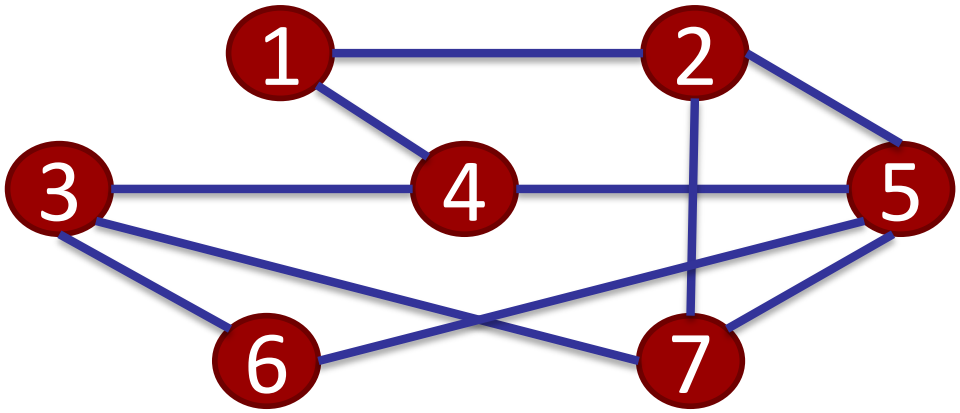
[*] graf.c SQL.h

```c
#include <stdio.h>
#include <stdarg.h>
typedef int T;
#include "SQL.h"

void List_print(Link_List a) { Link_Item p = a->Front;
    while(p) {
        printf("%d ",Item_get_node(p)+1);
        p=Item_get_next(p); }
}

typedef struct Graf { Link_List S; int N; } *Link_Graf;

Link_Graf Graf_create(int n) { int i;
    Link_Graf p = (Link_Graf)calloc(1, sizeof(struct Graf));
    p->S = (Link_List)calloc(p->N = n, sizeof(struct List));
    for(i = 0; i < n ; i++) List_ini(&p->S[i]);
    return p; }
Link_List Graf_Get_node(Link_Graf a,T name) {
    return &a->S[name]; }
void Graf_Set_node(Link_Graf a,T name,...) {
T v; va_list p; va_start(p,name);
    while(v = va_arg(p,T)) List_Insert_back(&a->S[name - 1], v - 1); va_end(p); }
void Graf_print(Link_Graf a) { int i;
    for(i=0;i<a->N;i++) { printf("%d: ",i+1); List_print(&a->S[i]); putchar('\n'); }
}
void Graf_delete(Link_Graf *a) { free((*a)->S); free(*a); *a=NULL; }
```

Компилятор Ресурсы Журнал компиляции Отладка Результаты поиска

Основная программа

```
int main() {
    Link_Graf G = Graf_create(7), g = Graf_create(7);
    Graf_Set_node(G,1,2,4,0);        Graf_Set_node(g,1,2,4,0);
    Graf_Set_node(G,2,1,5,7,0);      Graf_Set_node(g,2,1,5,7,0);
    Graf_Set_node(G,3,4,6,7,0);      Graf_Set_node(g,3,4,6,7,0);
    Graf_Set_node(G,4,1,3,5,0);      Graf_Set_node(g,4,1,3,5,0);
    Graf_Set_node(G,5,2,4,6,7,0);    Graf_Set_node(g,5,2,4,6,7,0);
    Graf_Set_node(G,6,3,5,0);        Graf_Set_node(g,6,3,5,0);
    Graf_Set_node(G,7,2,3,5,0);      Graf_Set_node(g,7,2,3,5,0);
    Graf_print(G);
    print_V(G,0); Graf_delete(&G); putchar('\n');
    print_G(g,0); Graf_delete(&g);
    return 0;
}
```



D:\Disk_S\Lect2016\Exmp\12\C\graf.c - Dev-C++ 5.11

Файл Правка Поиск Вид Проект Выполнить Сервис AStyle Окно Справка

TDM-GCC 4.9.2 64-bit Release

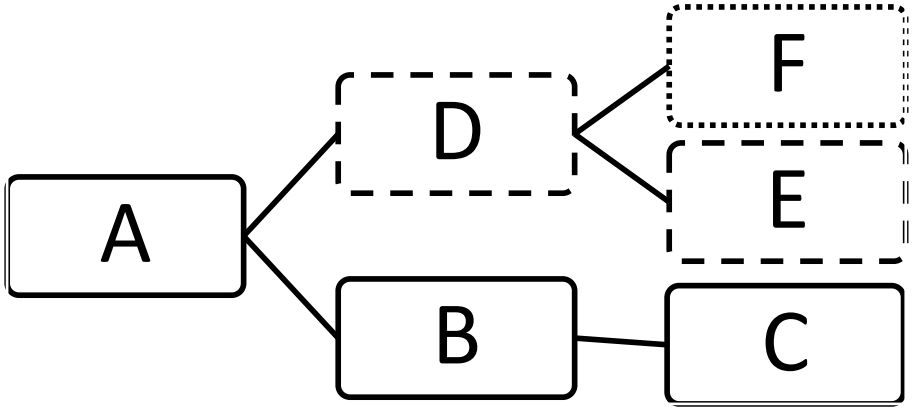
(globals)

[*] graf.c SQL.h

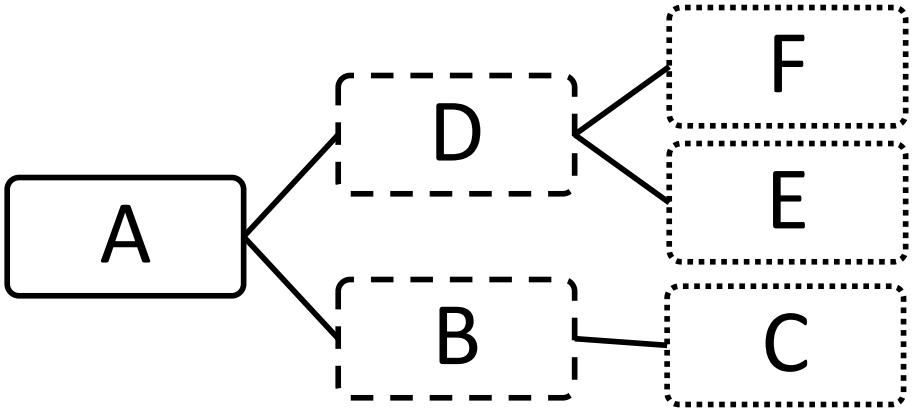
```c
int main() {
Link_Graf G = Graf_create(7), g = Graf_create(7);
    Graf_Set_node(G,1,2,4,0);       Graf_Set_node(g,1,2,4,0);
    Graf_Set_node(G,2,1,5,7,0);     Graf_Set_node(g,2,1,5,7,0);
    Graf_Set_node(G,3,4,6,7,0);     Graf_Set_node(g,3,4,6,7,0);
    Graf_Set_node(G,4,1,3,5,0);     Graf_Set_node(g,4,1,3,5,0);
    Graf_Set_node(G,5,2,4,6,7,0);   Graf_Set_node(g,5,2,4,6,7,0);
    Graf_Set_node(G,6,3,5,0);       Graf_Set_node(g,6,3,5,0);
    Graf_Set_node(G,7,2,3,5,0);     Graf_Set_node(g,7,2,3,5,0);
    Graf_print(G);
    print_V(G,0);   Graf_delete(&G); putchar('\n');
    print_G(g,0);   Graf_delete(&g);
    return 0;
}
```

Алгоритмы обхода

- В глубину



- В ширину



Обход в глубину (Си)

```
void print_V(Link_Graf A, int k) {
struct Stack Q; Stack_ini(&Q);
int *a = (int*)calloc(A->N,sizeof(int));
  Stack_push(&Q,k); a[k]=1;
  while(!Stack_Is_Empty(&Q)) {
    while(!List_Is_Empty(Graf_Get_node(A,k))){
      T i=List_Remove_front(Graf_Get_node(A,k));
      if(!a[i]) { a[i]=1; Stack_push(&Q,i);
        printf("(%d,%d)\n",k+1,i+1); k=i;}
    }
    k=Stack_pop(&Q);
  }
}
```

Обход в ширину (Си)

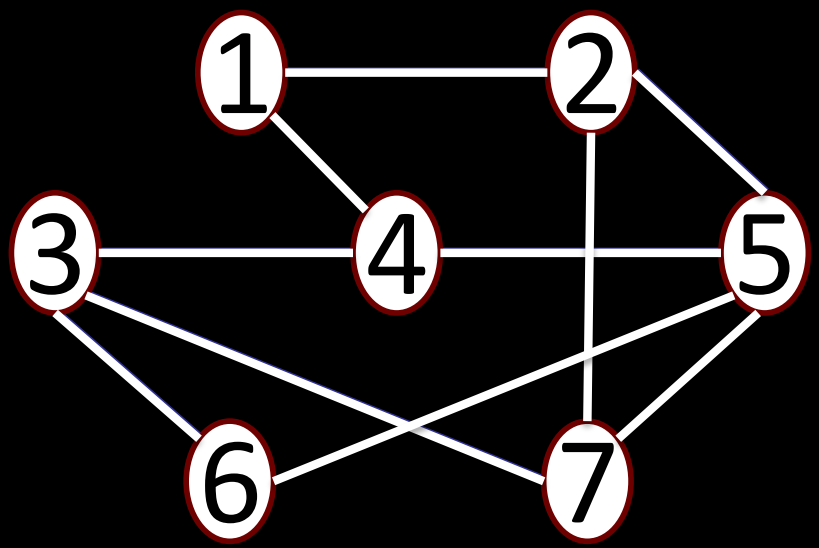
```
void print_G(Link_Graf A, int k) {
struct Queue Q; Queue_ini(&Q);
int *a = (int*)calloc(A->N,sizeof(int));
  Queue_put(&Q,k); a[k]=1;
  while(!Queue_Is_Empty(&Q)) {
    k=Queue_get(&Q);
    while(!List_Is_Empty(Graf_Get_node(A,k))) {
      T i=List_Remove_front(Graf_Get_node(A,k));
      if(!a[i]) { Queue_put(&Q,i); a[i]=1;
         printf("(%d,%d)\n",k+1,i+1); }
    }
  }
}
```

SQL.h graf.c

```c

void print_V(Link_Graf A, int k) { //вывести начиная с
struct Stack Q; Stack_ini(&Q); // стек плана обхода
int *a = (int*)calloc(A->N, sizeof(int)); // отметки о прохождении
   Stack_push(&Q, k); a[k]=1; // данную вершину в план обхода, были
   while(!Stack_Is_Empty(&Q)) { //пока не выполнили план обхода
       while(!List_Is_Empty(Graf_Get_node(A,k))) { // пока есть смежные
           T i=List_Remove_front(Graf_Get_node(A,k)); // следующая смежная
           // если не были, то в план, были, вывести ребро
          if(!a[i]) { Stack_push(&Q,i); a[i]=1; printf("(%d,%d)\n",k+1,i+1); k=i;}
       } // учли смежные
      k=Stack_pop(&Q); // следующая по плану
   } // обошли всё
   free(a);
}

void print_G(Link_Graf A, int k) { //вывести начиная с
struct Queue Q; Queue_ini(&Q); // очередь плана обхода
int *a = (int*)calloc(A->N,sizeof(int)); // отметки о прохождении
   Queue_put(&Q,k); a[k]=1; // данную вершину в план обхода, были
   while(!Queue_Is_Empty(&Q)) { //пока не выполнили план обхода
       k=Queue_get(&Q); // следующая по плану
       while(!List_Is_Empty(Graf_Get_node(A,k))) { // пока есть смежные
           T i=List_Remove_front(Graf_Get_node(A,k)); // сдедующая смежная
           // если не были, то в план, были, вывести ребро
           if(!a[i]) { Queue_put(&Q,i); a[i]=1; printf("(%d,%d)\n",k+1,i+1); }
       } // учли смежные
   }  // обошли всё
   free(a);
}
```

D:\rab\S1\2014\graf.exe

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

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

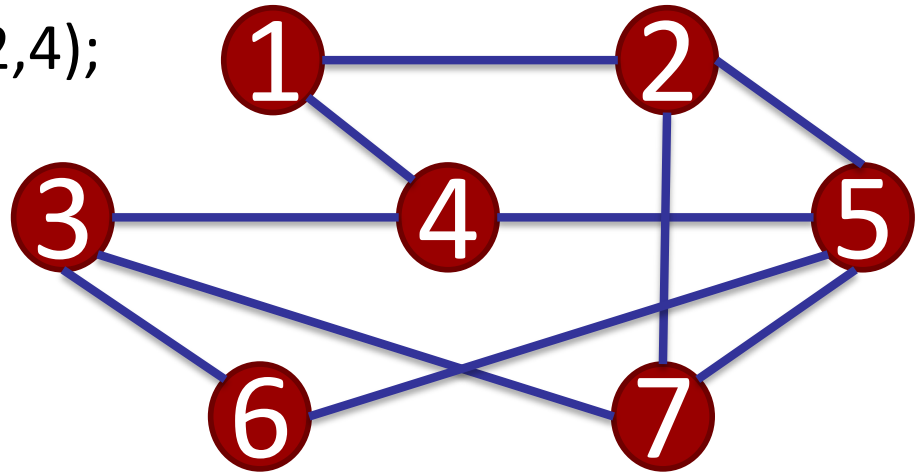


Описание шаблона типа Graph

```
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; }
```

Обход в глубину

```
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
}
```

Обход в ширину

```
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
}
```

D:\A1\Lect2015\Exmp\12\g1.cpp - Dev-C++ 5.10

Файл Правка Поиск Вид Проект Выполнить Сервис AStyle Окно Справка

TDM-GCC 4.8.1 32-bit Profiling

(globals)

g1.cpp SQL.h

```cpp
#include <iostream>
#include <cstdarg>
using namespace std;
#include "SQL.h"

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;}
};

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
}

```

Компилятор Ресурсы Журнал компиляции Отладка Результаты поиска

Line: 49 Col: 5 Sel: 0 Lines: 52 Length: 1533 Вставка Done parsing in 0,031 seconds

D:\A1\Lect2015\Exmp\12\g1.cpp - [Executing] - Dev-C++ 5.10

Файл Правка Поиск Вид Проект Выполнить Сервис AStyle Окно Справка

TDM-GCC 4.8.1 32-bit Profiling

(globals)

g1.cpp | SQL.h | [*] graf.c

```
27
28 void print_G(Graph<int,7> A, int k) {
29 Queue<int> Q; Q.put(k); A[k]++;
30     while(!Q.empty()) {
31         k=Q.get();
32         while(!A(k).empty()) {
33             int i=A(k).pop_front();
34             if(!A[i]) { Q.put(i); A[i]++; cout << '(' << k+1 << ',' << i+1 << ")\n"; }
35         }
36     }   // end while
37 }
38
39 int main() {
40 Graph<int,7> A;
41     A(0,2,1,3); A(1,3,0,4,6); A(2,3,3,5,6); A(3,3,0,2,4);
42     A(4,4,1,3,5,6); A(5,2,2,4); A(6,3,1,2,4);
43     cout << A << endl;
44     print_V(A,0); cout << endl;
45
46 Graph<int,7> B;
47     B(0,2,1,3); B(1,3,0,4,6); B(2,3,3,5,6); B(3,3,0,2,4);
48     B(4,4,1,3,5,6); B(5,2,2,4); B(6,3,1,2,4);
49     print_G(B,0);
50     return 0;
51 }
52
```

Компилятор (5) | Ресурсы | Журнал компиляции | Отладка | Результаты поиска

Line: 49 Col: 5 Sel: 0 Lines: 52 Length: 1533 Вставка Done parsing in 0,125 seconds

Построение Минимального Остового Дерева

- Остовое дерево графа
  - подграф из всех вершин и части рёбер
  - связный
  - ациклический
- МОД взвешенного графа
  - остовое дерево
  - минимального суммарного веса
- Трудоемкость полного перебора O(exp(N))
- Жадный алгоритм
  - на каждом шаге по части данных выберем лучший следующий

Алгоритм Прима роста МОД

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

M – уже найденные вершины МОД (дерево)

M’ – вершины, смежные с вершинами M (кайма)

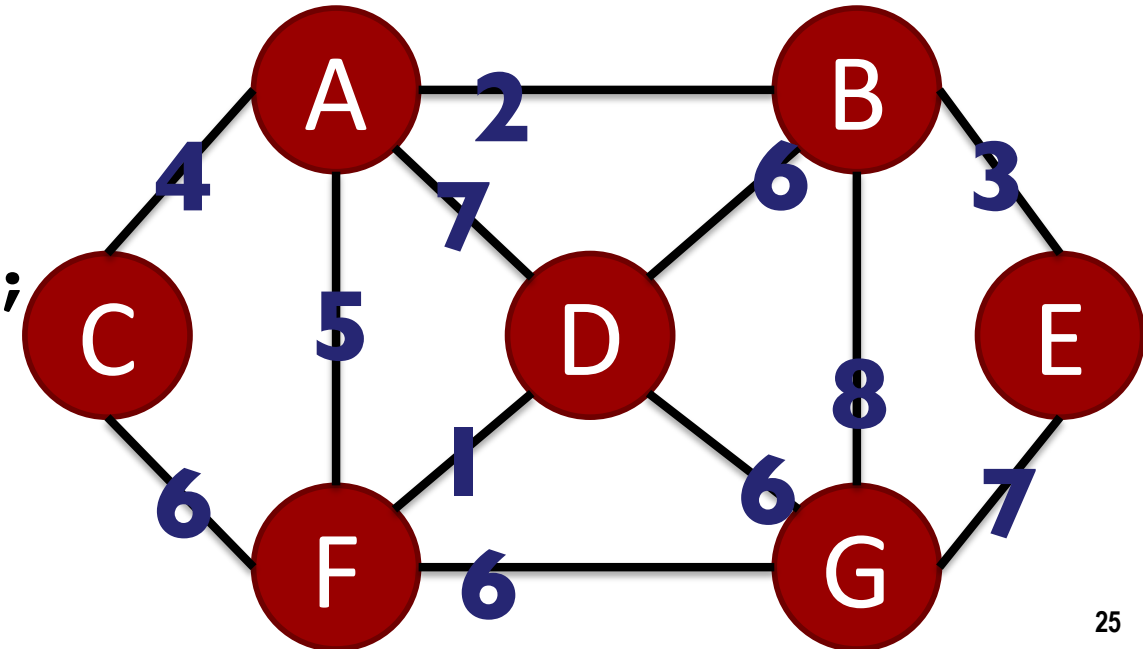
$R = V \backslash M \backslash M'$ – остальные вершины графа

- **Укоренить дерево (выбрать корень)**
- **Окаймить корень**
- **Пока кайма не опустеет**
  - Выбирать между деревом и каймой легчайшее ребро
  - Дополнять дерево найденным ребром
  - Уточнять кайму: вершины и рёбра

отнесение к дереву – на диагонали матрицы смежности

Задание матрицей смежности

```
#include <iostream>
using namespace std;
int N=7, 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() { // дерево растёт ?
   for(int i=0;i<N;i++)
     if(!A[i][i]) return 1;
   return 0; }

void min(int& n,int& m) { // Легчайшее в кайме
  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];
}
```

Растим дерево

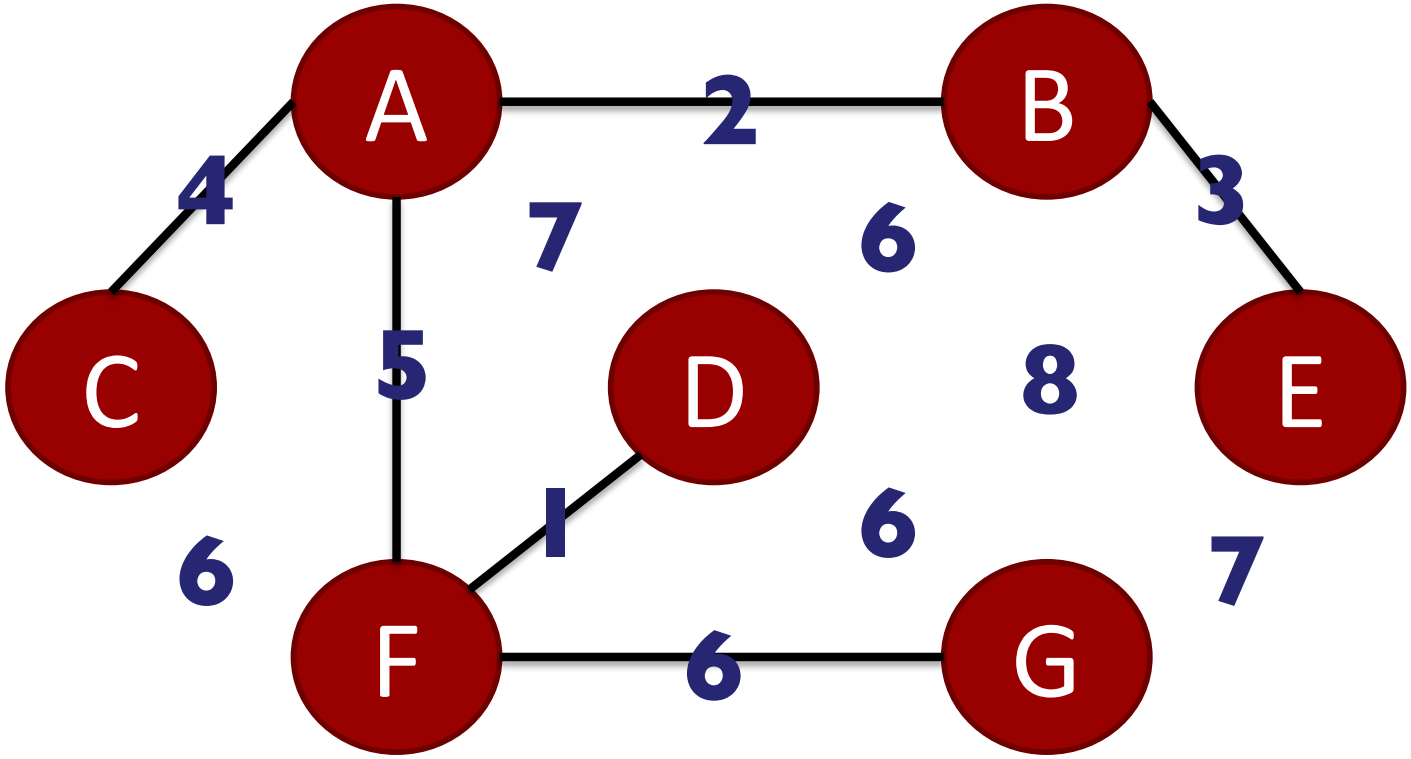
```
void mst(int k) { // Растить МОД от корня
int i,j,m,n;
  for(n=0;n<N;n++) for(m=0;m<N;m++) { B[n][m]=0;
    if(n==m) A[n][n]=0;
    else if (n>m) A[n][m]=A[m][n]; }
  A[k][k]=1; // укоренить дерево
  for(m=0;m<N;m++)
    B[k][m]=A[k][m]; // окаймить корень
  while( r() ) { // пока дерево растёт
    min(n,k); // найти легчайшее
    A[k][k]=1; // внести конец в дерево
    cout << char('A'+n) << char('A'+k) << endl;
```

Уточняем кайму

```
  for(m=0;m<N;m++)
    // новые ребра в кайму
    B[k][m]=!A[m][m]?A[k][m]:0;
 // каждое ребро в кайме если...
  for(m=0;m<N;m++) for(k=n=0;n<N;n++) if(B[n][m])
    if(!k) // первое?
      k=B[i=n][j=m]; // взять легчайшим
    else if(B[n][m]>k) // тяжелее?
      B[n][m]=0; // убрать из каймы
    else {
      B[i][j]=0;
      k=B[i=n][j=m];} // взять легчайшим
    // рёбра в кайме пересчитаны
  }// дерево выросло в МОД
}
int main() { mst(0); return 0; }
```

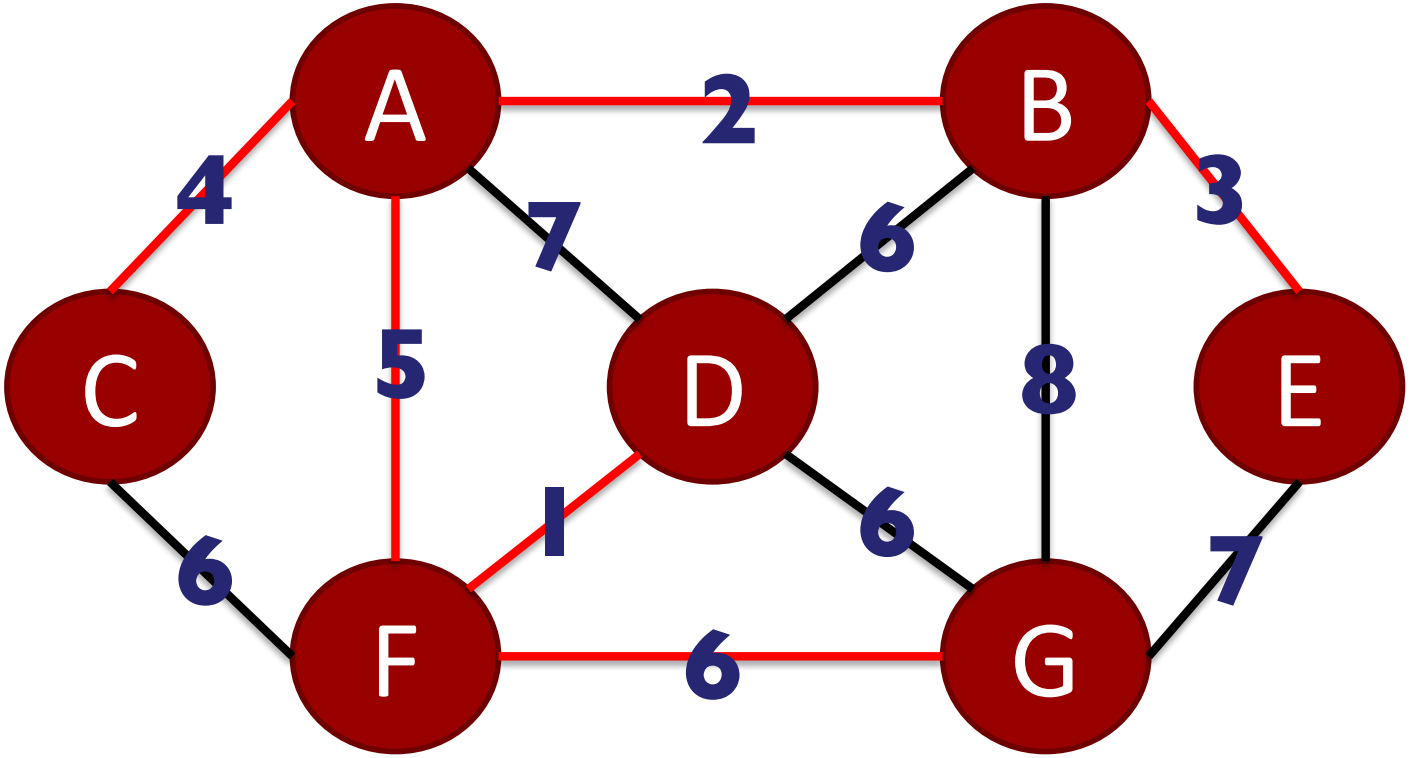
Результат

AB
BE
AC
AF
FD
FG



Поиск кратчайших путей

- ищем не легчайшее, а элемент кратчайшего пути



Алгоритм Дейкстры

$V = M \cup M' \cup R$
M – уже найденные вершины карты (дерево)
M’ – вершины смежные с вершинами M (кайма)
$R = V \backslash M \backslash M'$ – остальные вершины графа

- Начать карту
  - выбрать начальную вершину дерева
- Окаймить начало
- Пока нет конца
  - Найти в кайме ближайшую к начальной
  - Дополнить карту
  - Уточнить кайму: вершины и рёбра

расстояние до начала – на диагонали матрицы смежности

Вспомогательные функции

```
int r() { // дерево растёт ?
  for(int i=0;i<N;i++)
    if(!A[i][i]) return 1;
  return 0; }
int min() { int r,min=0; // ближайшая
  for(int i=0;i<N;i++) if(!B[i][i]) continue;
     else if((!min || B[i][i]<min) && !A[i][i])
       min=B[r=i][i];
  return r;
}
void print_B() { cout << endl; // вывод каймы
  for(int n=0;n<N;n++) { cout << char('A'+n) <<":\t";
    for(int m=0;m<N;m++) cout << (n==m?0:B[n][m]) << ' ';
    cout << '\t' << B[n][n] << endl; }
}
```

Основная программа – 1

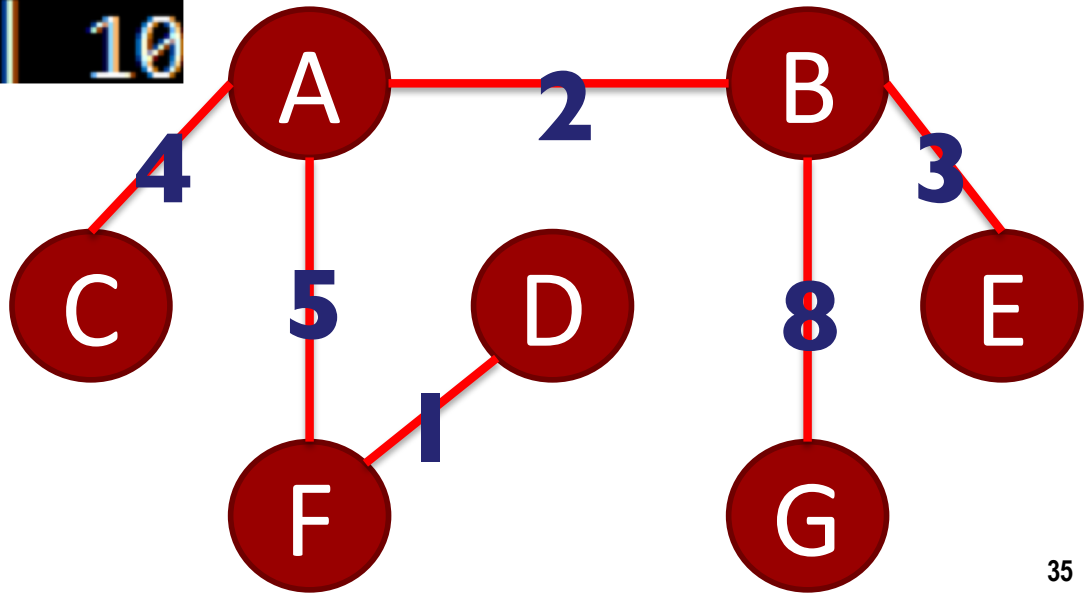
```
int main() {
int d,n,m,k=0;
  for(n=0;n<N;n++) for(m=0;m<N;m++) { B[n][m]=0;
    if(n>m) A[n][m]=A[m][n]; }
  for(m=0;m<N;m++) B[m][m]=B[k][m]=A[k][m];
    A[k][k]=1; // начать карту
  while( r() ) { // пока дерево растёт
    k=min(); // найти ближайшую
    A[k][k]=1; // внести в дерево
    for(m=0;m<N;m++) //найти связи
      // себя, несвязанные, пройденные – пропустить
      if(k==m || !A[k][m] || A[m][m]) continue;
```

Основная программа – 2

```
    else { d=B[k][k]+A[k][m]; // расстояние до
начала
        // для новой – запомнить расстояние и ребро
        if(!B[m][m]) { B[m][m]=d; B[k][m]=A[k][m]; }
        // для расстояния длиннее – ребро не добавлять
        else if(B[m][m]<=d) B[k][m]=0;
        // иначе – запомнить расстояние и ребро
        else { B[m][m]=d; B[k][m]=A[k][m];
          for(n=0;n<N;n++) // удалить старую связь
            if(n!=m && n!=k && B[n][m])
              B[n][m]=0; }
    } // расстояние учтено
  } // дерево выросло в карту кратчайших путей
  print_B(); return 0;
}
```

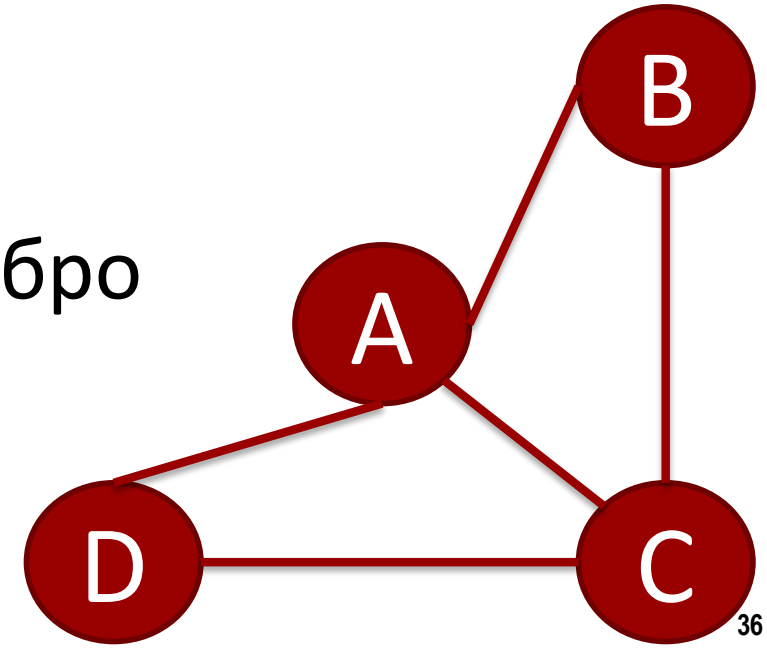
Результат

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

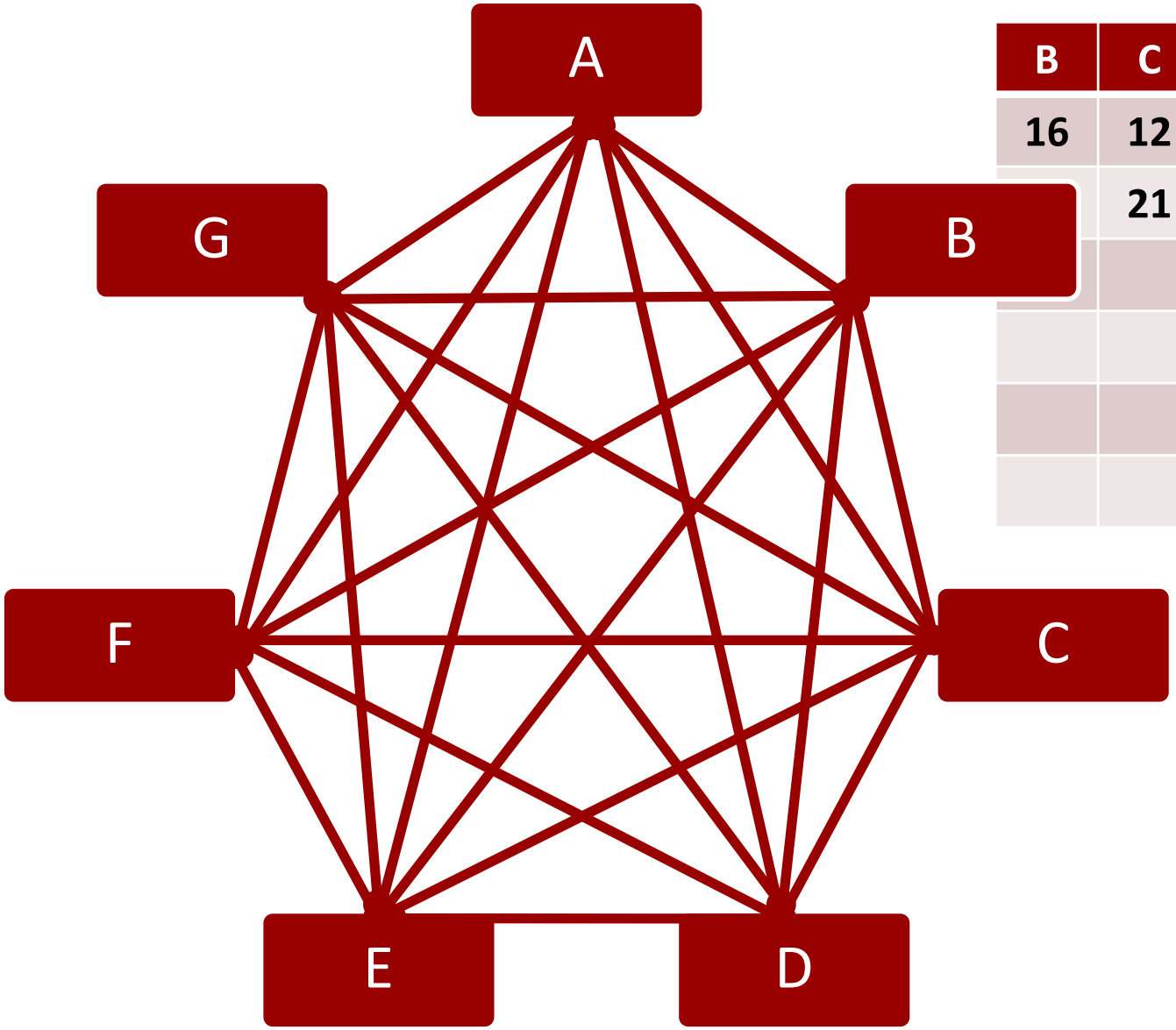


Циклы

- *Цикл* – путь, где конец совпадает с началом
- В *простом* цикле различны все вершины, кроме начала и конца
- *Гамильтонов* цикл прост и содержит все вершины графа
- ***Эйлеров*** граф содержит цикл, проходящий через каждое ребро графа ровно один раз



Задача коммивояжера – найти гамильтонов цикл



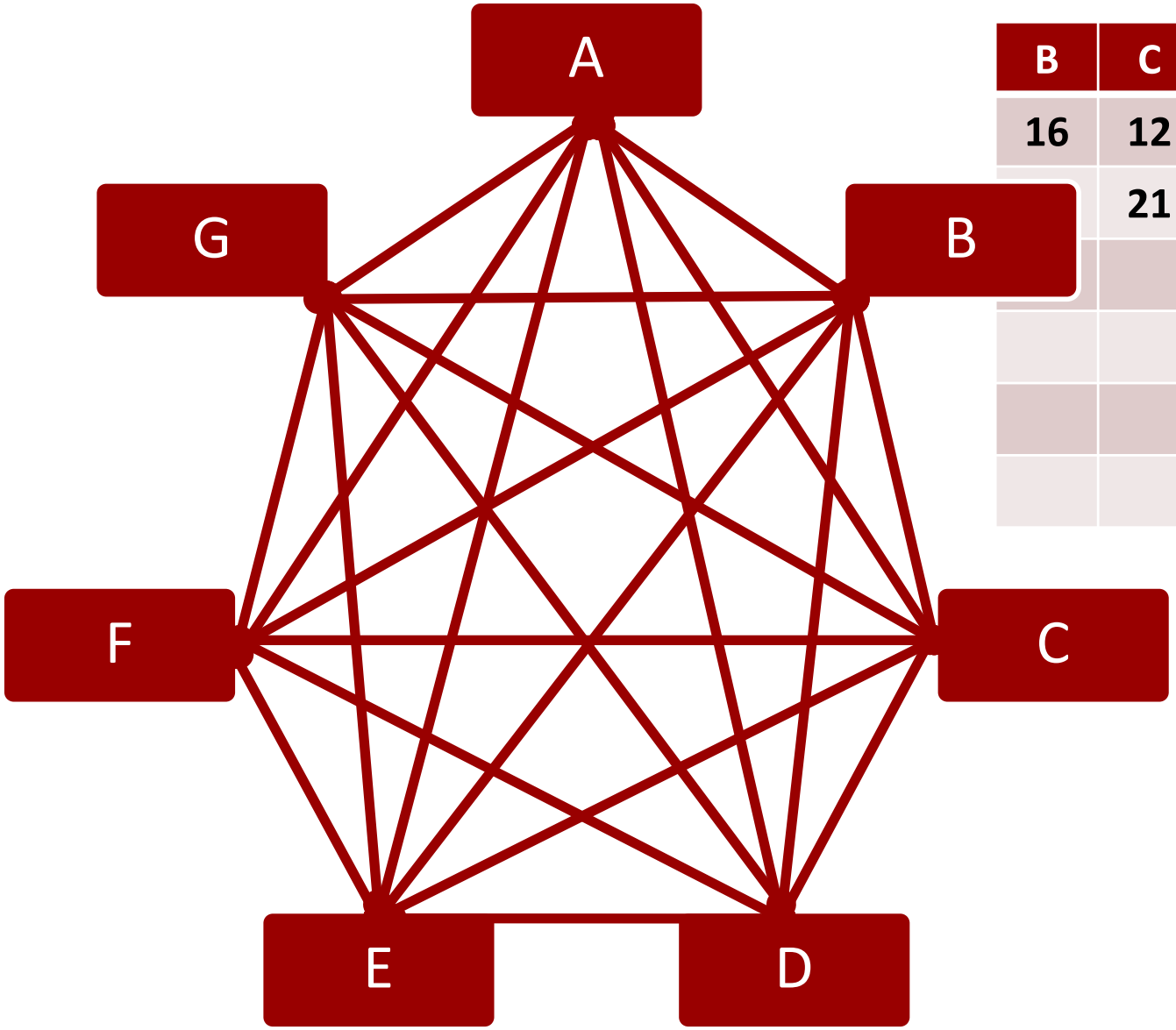
B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

Алгоритм поиска пути

- Пока не найден полный цикл
  - Добавлять в путь ребро..
    1. легчайшее
    2. не третье при любой вершине пути
    3. не образующее неполного цикла

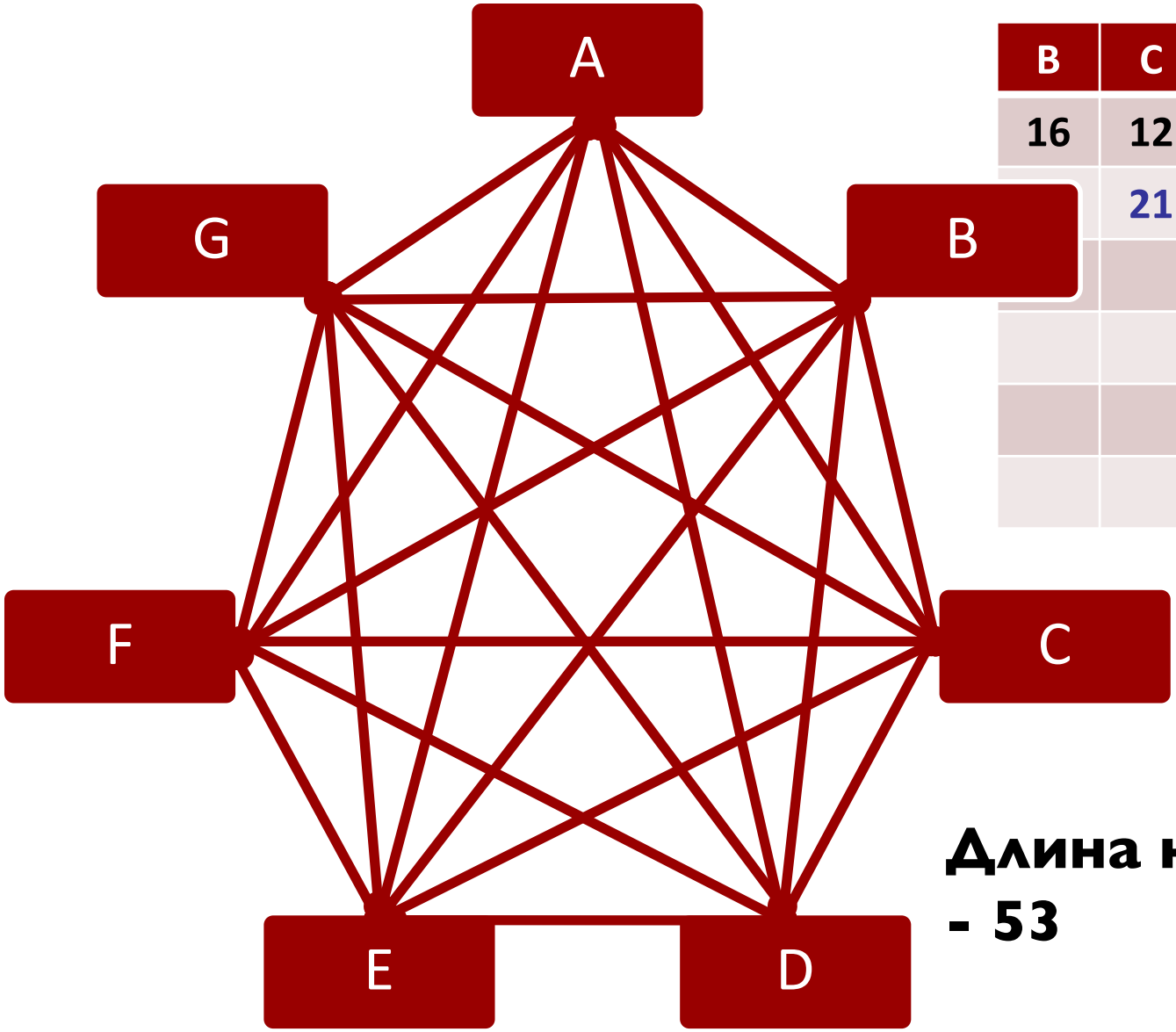
количество рёбер пути при вершине – на диагонали матрицы смежности

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

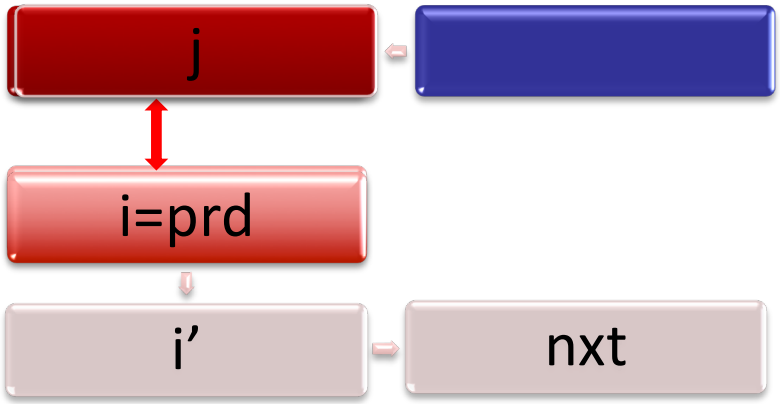
Задание матрицы смежности

```
#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}};
```

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

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



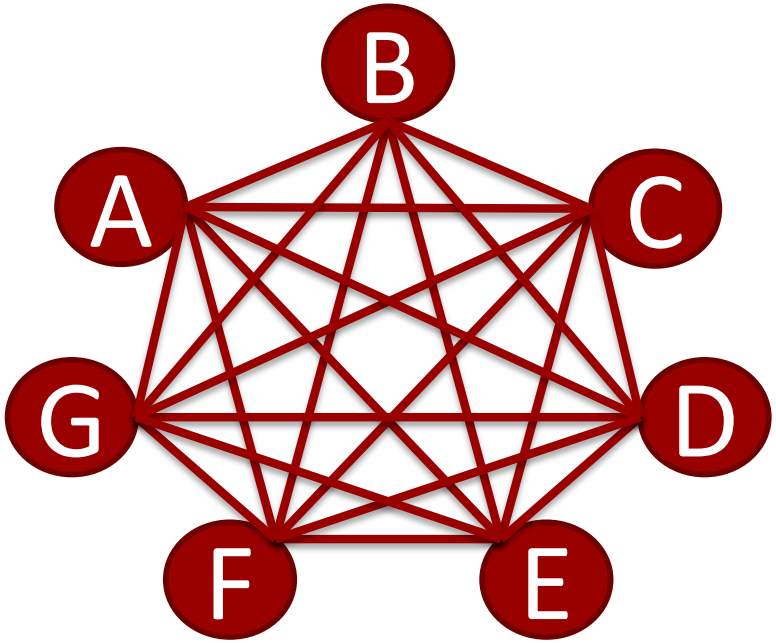
Выбор следующего ребра

```
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

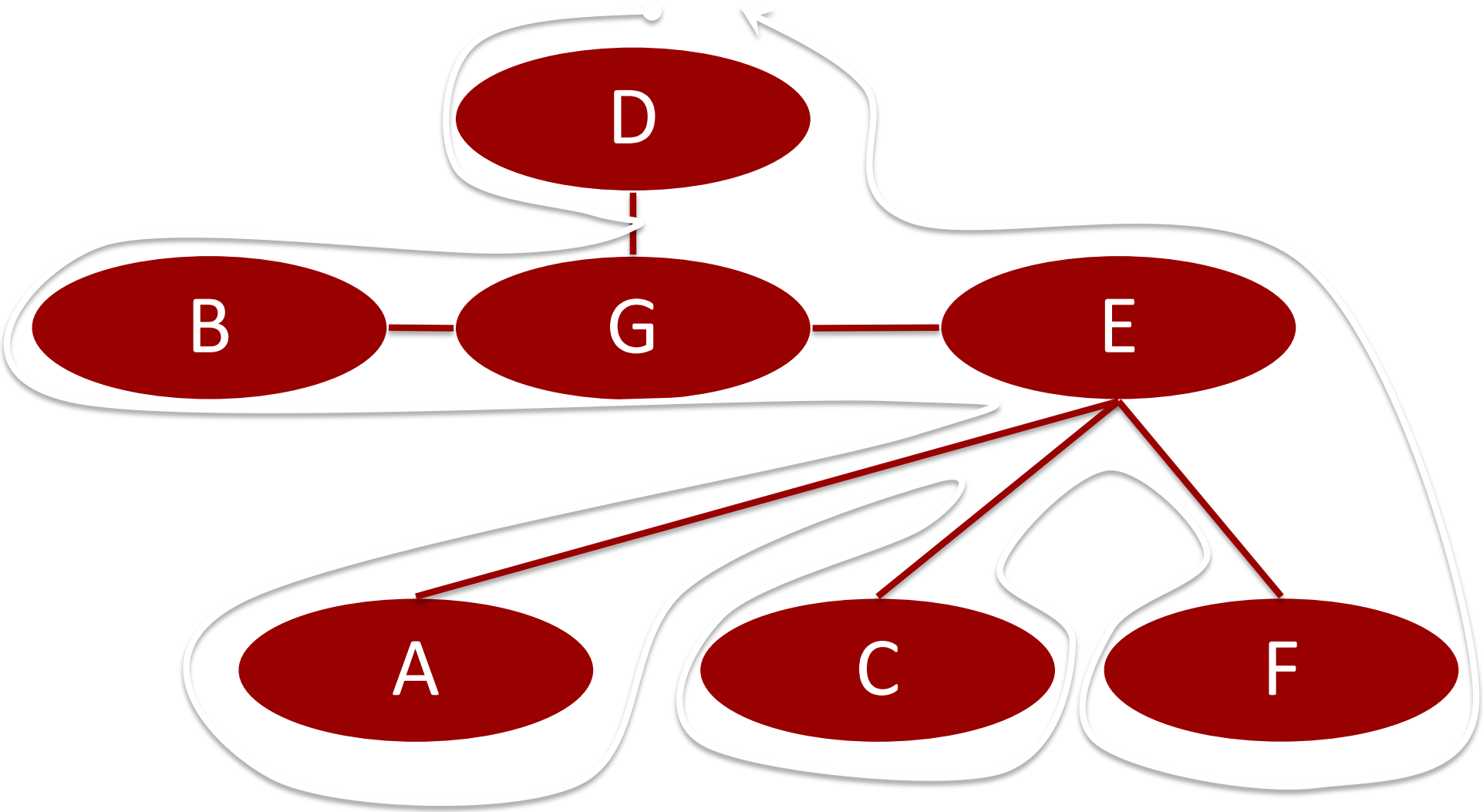
```
void print_B() { int s=0;  cout << endl;
   for(int n=0;n<N;n++) { cout << char('A'+n) <<":\t";
            for(int m=0;m<N;m++) { s+=B[n][m]; cout << B[n][m] << ' '; }
            cout << endl; }
   cout << "Length= " << s/2 << endl;
}

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 “SQL.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; }
```

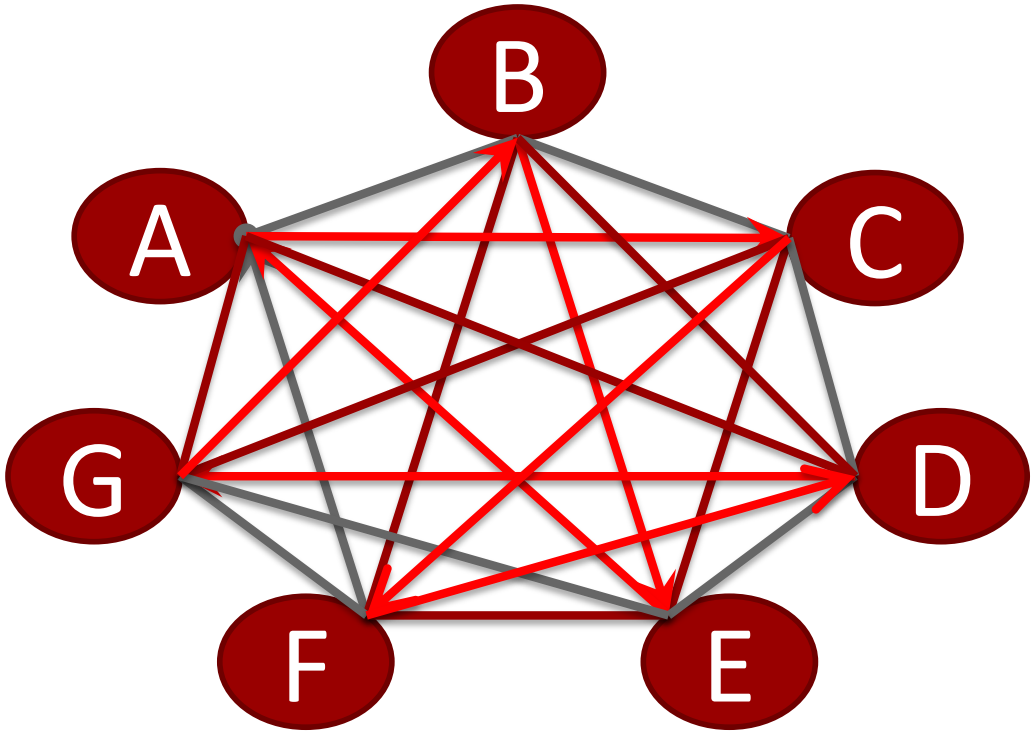
Гамильтонов цикл

D→G→B→E→A→C→F→D

Длина найденного пути – 48

Длина оптимального пути – 41

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



Гамильтонов цикл

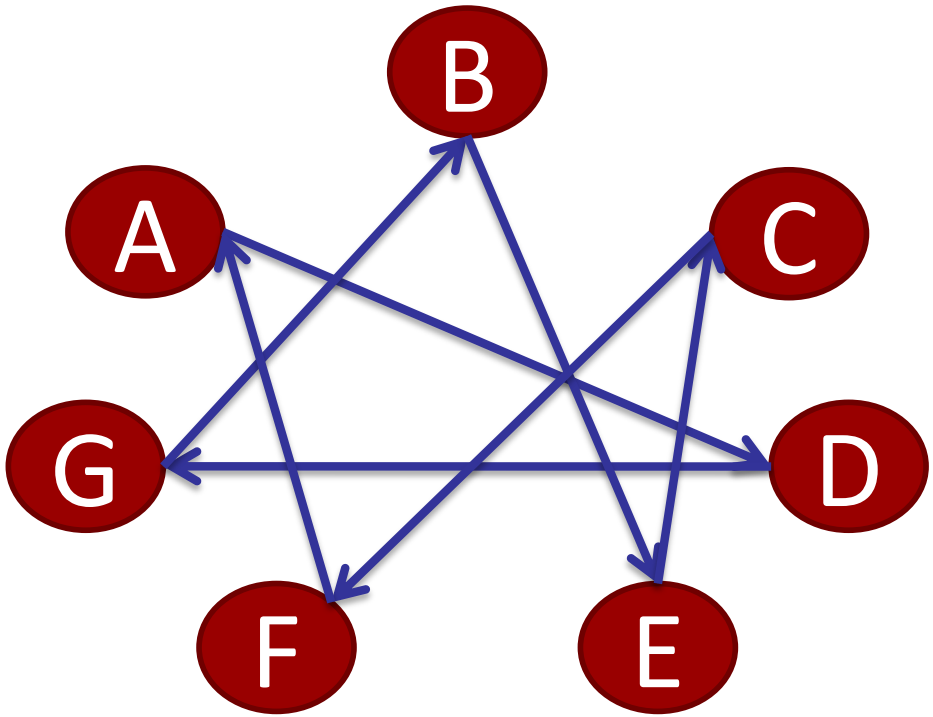
B	C	D	E	F	G	
16	12	13	6	7	11	A
	21	18	8	19	5	B
		20	1	3	15	C
			14	10	4	D
				2	7	E
					9	F

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 С++» -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