**Билет 6**

1. Виртуальное наследование классов (вида class A: public virtual B). Назначение. Примеры.

Виртуальное наследование в C++ используется, чтобы **избежать проблемы ромбовидного наследования**.  
Это проблема возникает, когда класс **наследуется от двух классов**, которые **оба происходят от одного общего предка**.  
Пример проблемы без виртуального наследования:

#include <iostream>

class A **{**

public**:**

void show**()** **{**

std**::**cout **<<** "Class A\n"**;**

**}**

**};**

class B **:** public A **{};** // Наследуем A

class C **:** public A **{};** // Наследуем A

class D **:** public B**,** public C **{};** // Наследуем B и C (оба содержат A)

int main**()** **{**

D obj**;**

obj**.**show**();** // Ошибка! Компилятор не знает, какую версию show() вызвать

**}**

Здесь у класса D **две копии класса A**, поэтому при вызове obj.show() компилятор не знает, какую show() использовать.  
**Решение через виртуальное наследование:**

#include <iostream>

class A **{**

public**:**

void show**()** **{**

std**::**cout **<<** "Class A\n"**;**

**}**

**};**

class B **:** public virtual A **{};** // Виртуальное наследование

class C **:** public virtual A **{};** // Виртуальное наследование

class D **:** public B**,** public C **{};** // Теперь только одна копия A

int main**()** **{**

D obj**;**

obj**.**show**();** // Работает! Теперь только одна копия A

**}**

2. Используя объектно-ориентированный подход (допускается STL) написать программу, выполняющую следующие действия:   
а) Ввод дуг ориентированного графа G(X,U).   
б) Поиск кратчайшего пути – вершина-источник и вершина-назначение определяются пользователем.  
Код:  
#include <iostream>  
#include <vector>  
#include <queue>  
#include <limits>  
#include <unordered\_map>

// Используем стандартное максимальное значение для расстояний

const int INF **=** std**::**numeric\_limits**<**int**>::**max**();**

// Структура для представления ориентированного графа

class Graph **{**

private**:**

// Список смежности: key - вершина, value - список пар (сосед, вес)

std**::**unordered\_map**<**int**,** std**::**vector**<**std**::**pair**<**int**,** int**>>>** adj**;**

public**:**

// Добавление ребра в граф

void addEdge**(**int u**,** int v**,** int weight**)** **{**

adj**[**u**].**push\_back**({**v**,** weight**});** // Добавляем направленное ребро u -> v

**}**

// Алгоритм Дейкстры для поиска кратчайшего пути

void dijkstra**(**int start**,** int target**)** **{**

// Хранит минимальные расстояния от стартовой вершины

std**::**unordered\_map**<**int**,** int**>** dist**;**

// Очередь с приоритетом (min-heap)

std**::**priority\_queue**<**std**::**pair**<**int**,** int**>,** std**::**vector**<**std**::**pair**<**int**,** int**>>,** std**::**greater**<>>** pq**;**

// Инициализация всех расстояний бесконечностью

**for** **(**const auto **&**node **:** adj**)** **{**

dist**[**node**.**first**]** **=** INF**;**

**}**

// Расстояние от стартовой вершины до самой себя = 0

dist**[**start**]** **=** 0**;**

pq**.**push**({**0**,** start**});**

// Основной цикл алгоритма

**while** **(!**pq**.**empty**())** **{**

int currentDist **=** pq**.**top**().**first**;** // Текущая длина пути

int currentNode **=** pq**.**top**().**second**;** // Текущая вершина

pq**.**pop**();**

// Если достигли целевой вершины - выходим

**if** **(**currentNode **==** target**)** **break;**

// Обход всех соседей текущей вершины

**for** **(**auto **&[**neighbor**,** weight**]** **:** adj**[**currentNode**])** **{**

int newDist **=** currentDist **+** weight**;**

// Если нашли более короткий путь до соседа - обновляем

**if** **(**newDist **<** dist**[**neighbor**])** **{**

dist**[**neighbor**]** **=** newDist**;**

pq**.**push**({**newDist**,** neighbor**});**

**}**

**}**

**}**

// Вывод результата

**if** **(**dist**[**target**]** **==** INF**)** **{**

std**::**cout **<<** "Нет пути из " **<<** start **<<** " в " **<<** target **<<** "\n"**;**

**}** **else** **{**

std**::**cout **<<** "Кратчайший путь из " **<<** start **<<** " в " **<<** target **<<** " = " **<<** dist**[**target**]** **<<** "\n"**;**

**}**

**}**

**};**

int main**()** **{**

Graph g**;**

int edges**,** u**,** v**,** w**;**

std**::**cout **<<** "Введите количество рёбер: "**;**

std**::**cin **>>** edges**;**

std**::**cout **<<** "Введите рёбра в формате: начало конец вес\n"**;**

**for** **(**int i **=** 0**;** i **<** edges**;** **++**i**)** **{**

std**::**cin **>>** u **>>** v **>>** w**;**

g**.**addEdge**(**u**,** v**,** w**);**

**}**

int start**,** target**;**

std**::**cout **<<** "Введите начальную и конечную вершины: "**;**

std**::**cin **>>** start **>>** target**;**

g**.**dijkstra**(**start**,** target**);**

**return** 0**;**

**}**

Этот код:

1. **Создаёт граф** как список смежности.
2. **Добавляет рёбра** (дуги) с весами.
3. **Запрашивает у пользователя начальную и конечную вершины**.
4. **Использует алгоритм Дейкстры** для поиска кратчайшего пути.
5. **Выводит результат** или сообщает, если пути нет.

Если что-то непонятно — спрашивай! 😊