**Най-кратък път в мрежа**

*Денис Зангъров*

*F95108*

***Кратко резюме***

Ще имплементираме „универсален“ алгоритъм, който е частично базиран на алгоритъма на Дийкстра за намиране на най-кратък път в смесен граф – бил той претеглен или не. Ще преминем през всички основни понятия, от които се нуждаем, представянето - нашите абстракции, структурите от данни, които ще използваме(и напишем) и алгоритмите за обхождане.

***Основни понятия***

***Математическо определение:***

Графът може да бъде представен като двойката *G=(V,E)*, където *V* е множеството на възлите, а *E* множеството на свързаните възли, чиито елементи наричаме ребра. Самото ребро се представя като *{c,d},* което показва че има връзка между възлите c и d.

***Видове графи:***

Преди да продължим с нашият начин на представяне, надграждането на абстракциите и методите, нека се спрем за момент върху няколко основни видове графи спрямо:

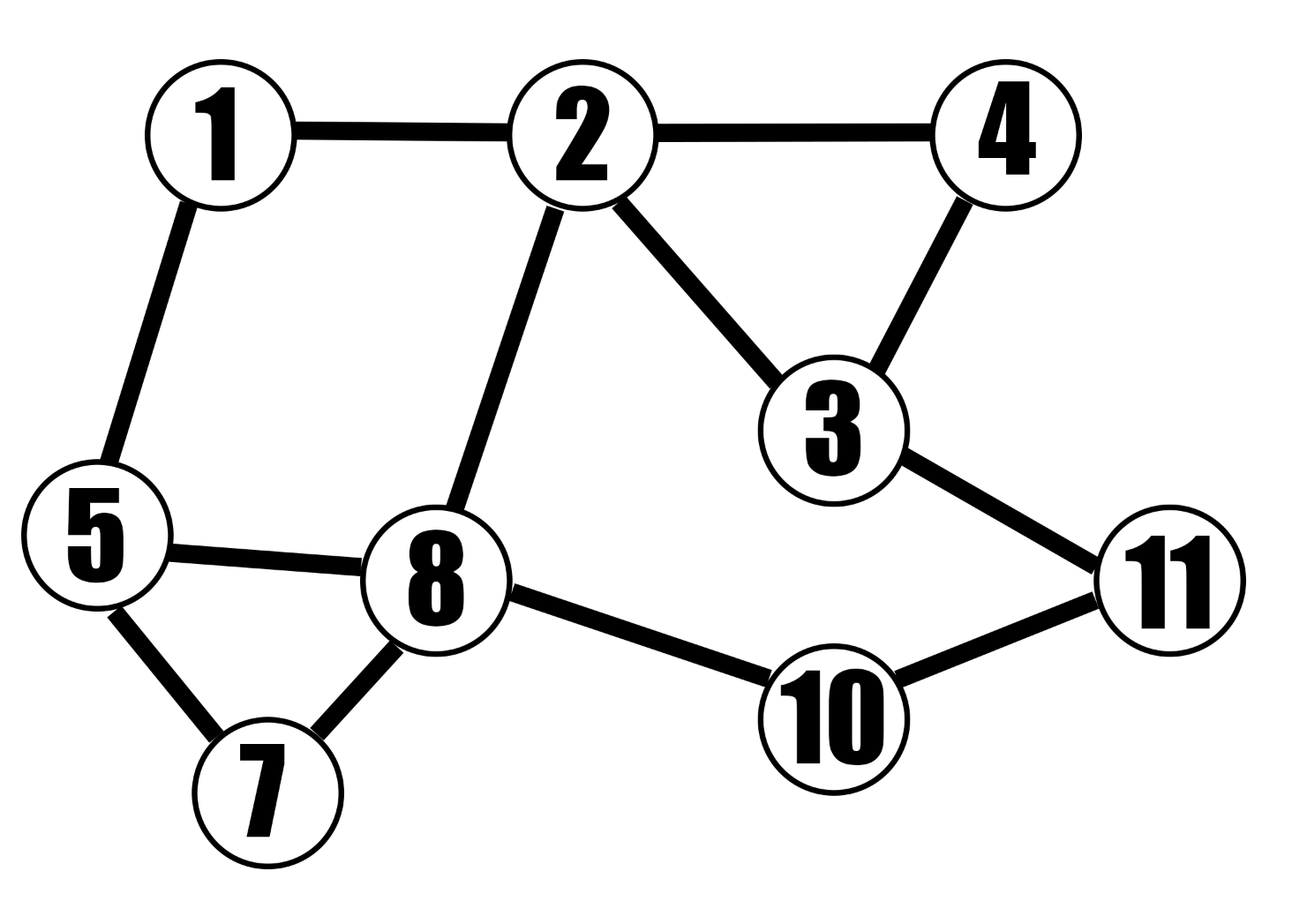
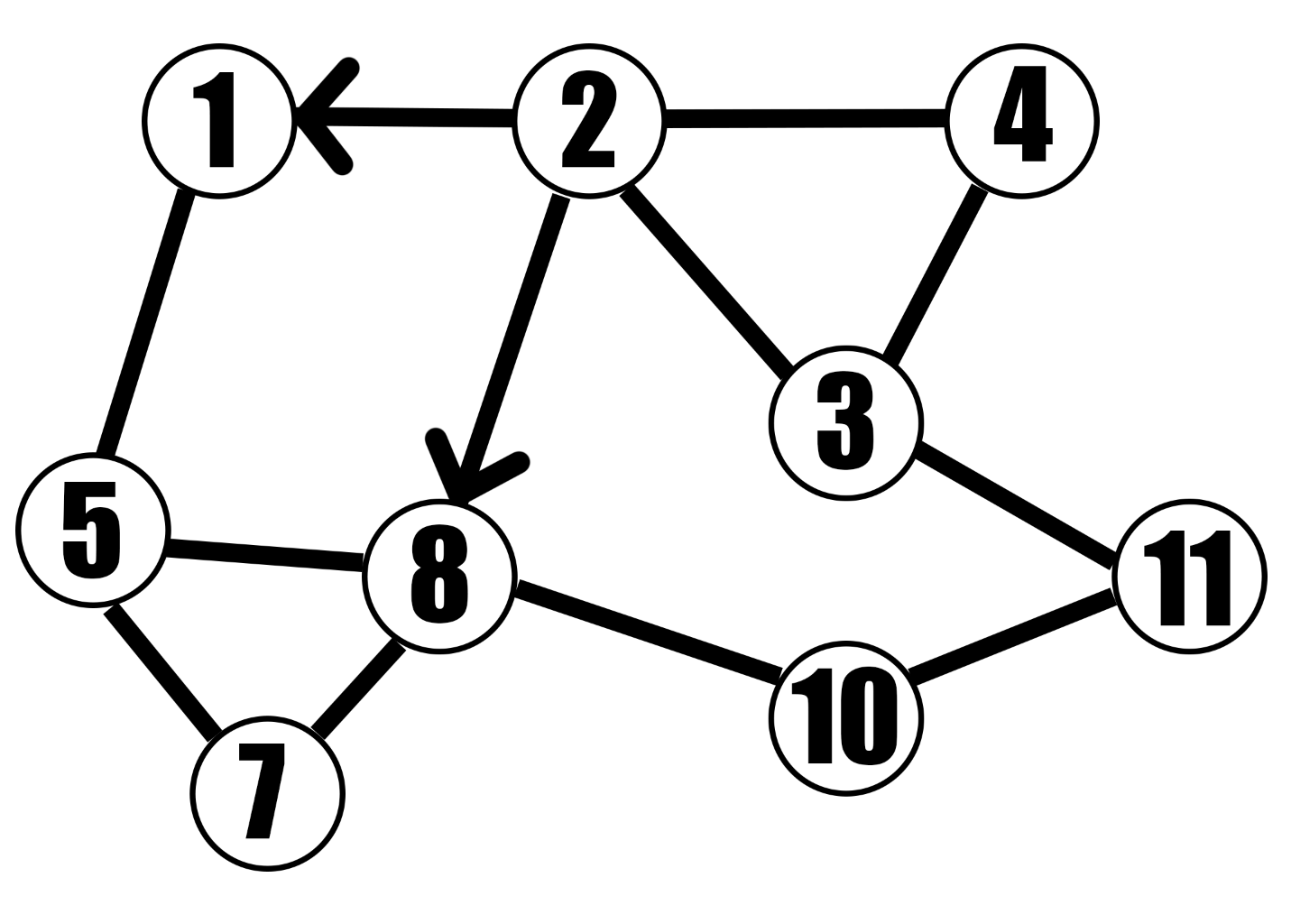
- посоката на придвижване:

* *ориентиран граф* – от всеки възел може да се пътува само еднопосочно – т.е. не може да се върнем обратно след напущане на възела.
* *неориентиран граф* – от всеки възел може да се пътува двустранно – след като напуснем възела, може да се върнем обратно.
* *смесен граф* – дадено ребро може да бъде еднопосочно или двупосочно.

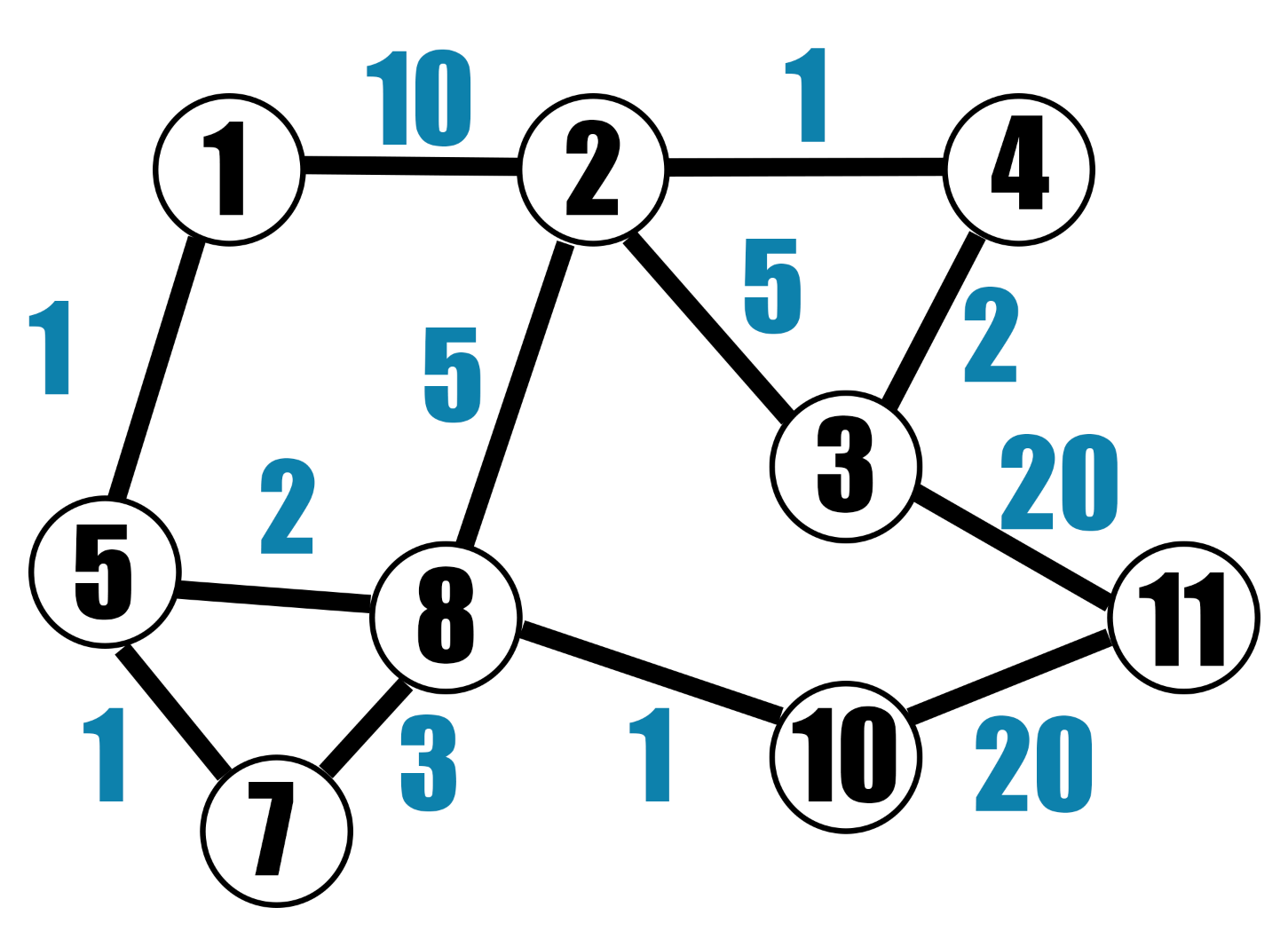
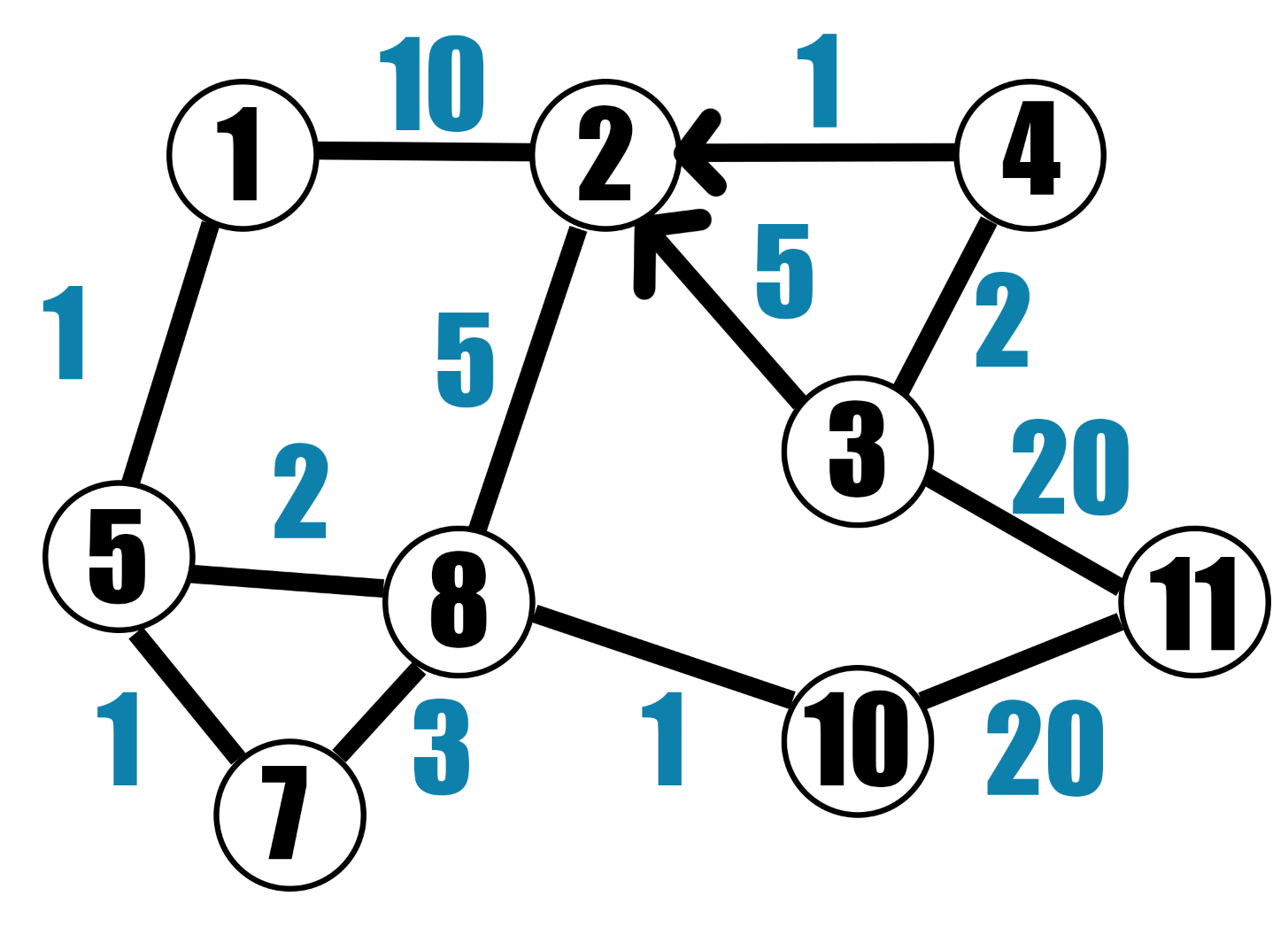
- ребрата:

* *претеглен* – ребрата между всеки два възела имат тежест на придвижването – например ако град Стара Загора е възел A, град Пловдив е възел B и град София е възел C. То от А до B имаме тегло 104 километра, но от A до C имаме тегло 230 километра.
* *непретеглен* – ребрата нямат тегло, взима се предвид броя придвижвания

***Най-кратък път в непретеглен граф***

В непретегления граф, най-кратък е пътя, който преминава през най-малко възли до достигане на дестинацията. Ще го разгледаме в няколко примери.  
 ***Пример 1:***  
  
Ако започнем от възел ***1*** и искаме да достигнем възел ***3*** има много начини на преминаване, но очевидния кратък път е поредицата ***1->2->3***.  
  
***Пример 2:***  
  
Когато променим графът дори малко, оптималният резултат може да бъде тотално различен. В случая, тъй като пътищата от ***2*** към ***8*** и ***1*** са еднопосочни, оптималното решение ще бъде ***1->5->8->10->11->3***.

***Най-кратък път в претеглен граф***

В претегления граф, най-кратък е пътя, който отнема най-малко(или най-много) точки до достигането на търсения възел.  
  
***Пример 1:***  
  
Ако отново желаем да достигнем възел ***3***, този път оптималният път ще бъде в последователността ***1->5->8->2->4->3***.  
  
***Пример 2:***  
  
Тъй като вече ***2*** няма проходимост до ***4*** и ***3***, сега най-краткият път ще бъде  
***1->5->8->10->11->3***.

***Представяне***

При създаването на класове за графи, лесно може да попаднем в ситуация, в която да създадем множество класове за различните видове – ориентиран, неориентиран, смесен, претеглен, непретеглен и т.н. Може би дори обектна абстракция за представяне на възлите? Абсолютно ненужно.  
Ще имплементираме само един единствен обект , който ще бъде само един вид – смесен. Също и достатъчно динамичен, за да поддържа претеглени и непретеглени графи – дори преобразуването на непретеглен в претеглен. Разбира се – темплейтен:

***Основна структура на класа:***

template <typename T = int>

class Graph {

private:

    std::unordered\_map<T, std::unordered\_map<T, double>> edges;

    bool weighted=false;

*//...*

}

Целият граф ще бъде представен чрез ***Hash Map***. Всеки ключ в map-a ще има отделен ***Hash Map***, което ще представляват възлите, с които е свързан и тежестта, за да ги достигне.   
Какво печелим, когато използваме тази структура? O(1) време за „издърпване“ на възел и негов съсед/тежест.  
Променливата ***weighted*** ще индикира дали сме преминали в състояние да третираме графа като претеглен или не.

***Конструктор:***

Graph(std::vector<std::pair<std::vector<T>, bool>> listOfEdges) {

        this->add\_edges(listOfEdges);

    }

Конструкторът ще очаква вход с формат във вида:

        { {10, 11}, true },

        { {5,   7}, false },

        // или

        { {1, 3, 1}, true },

        { {1, 2, 1}, true },

С наредената двойка показваме кой възел с кого е свързан, а ако е наредена тройка, то третото число показва тежестта, която има реброто, чрез true и false показваме респективно дали реброто е двупосочно.  
Чрез функцията add\_edges(), ние детерминираме вътрешно като какъв вид да третираме реброто и дали е нужно да преминем към претеглен вид.