Доклад

По Алгоритми и структури от данни

Тема – Видове представяния на графите

Изготвил: Валентин Кьосев 12 ,,а‘‘ клас

1. Видове представяния на графите в компютърната памет

2. Обхождане в дълбочина на граф

3. Обхождане в ширина на граф

4. Най-кратък път в граф по алгоритъма на Дейкстра

5. Имплементация на дървета и графи

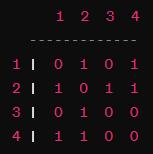
1. **Видове представяния на графите в компютърната памет**

Графите в компютърната памет могат да бъдат представени по различни начини в зависимост от нуждите на конкретната задача. Ето някои от основните видове представяния на графи:

**Матрица на съседство (Adjacency Matrix)**

Това е един от най-използваните начини за представяне на граф. В матрицата на съседство, редовете и колоните съответстват на върховете на графа, а клетките указват дали има връзка (ребро) между съответните върхове. За неориентирани графи, матрицата е симетрична относно главния диагонал, а за ориентирани графи, не е задължително да е симетрична.

**Пример:**

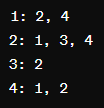
****

Тук 1 означава наличие на ребро между върховете, а 0 - липса на ребро.

**Списък на съседите (Adjacency List)**

В този вид представяне за всеки връх се съхранява списък от всички съседни върхове. Това е особено полезно за графи с много върхове и малко ребра, тъй като позволява бърз достъп до съседите на всеки връх.

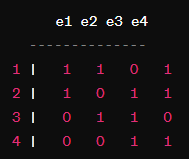
**Пример:**

****

**Матрица на инцидентност (Incidence Matrix)**

Този вид представяне се използва често за ориентирани графи или графи с тегла на ребрата. В тази матрица редовете съответстват на върховете, а колоните на ребрата. Елементът (i, j) показва дали връхът i е инцидентен с реброто j.

**Пример:**

****

Тук 1 означава, че върхът е инцидентен с реброто, а -1 (ако се използва) може да означава инцидентност на реброто със върха, но в различен начин (например за ориентирани графи).

**Списък на ребрата (Edge List)**

Тук всички ребра се изброяват в списък, като за всяко ребро се показват върховете, които го свързват.

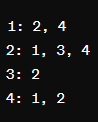
**Пример:**

****

**Свързаност на върховете (Connectivity of Nodes)**

Този подход не запазва пълната информация за графа, но само това кои върхове са свързани помежду си, без да се обръща внимание на конкретните ребра.

**Пример:**

****

Тук се вижда само списъците на съседите за всеки връх, без да се знае какви са ребрата между тях.

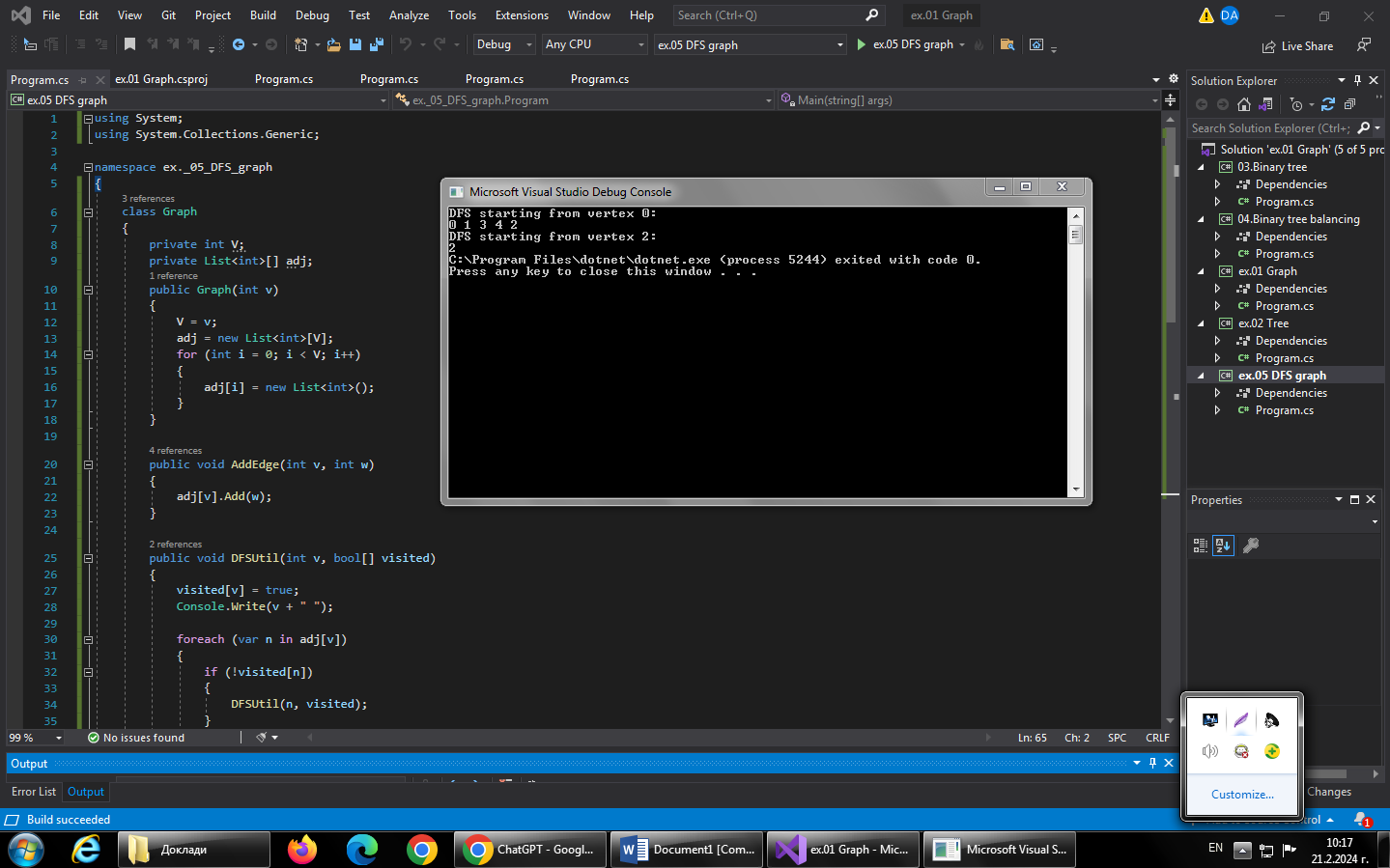
1. **Обхождане в дълбочина на граф**

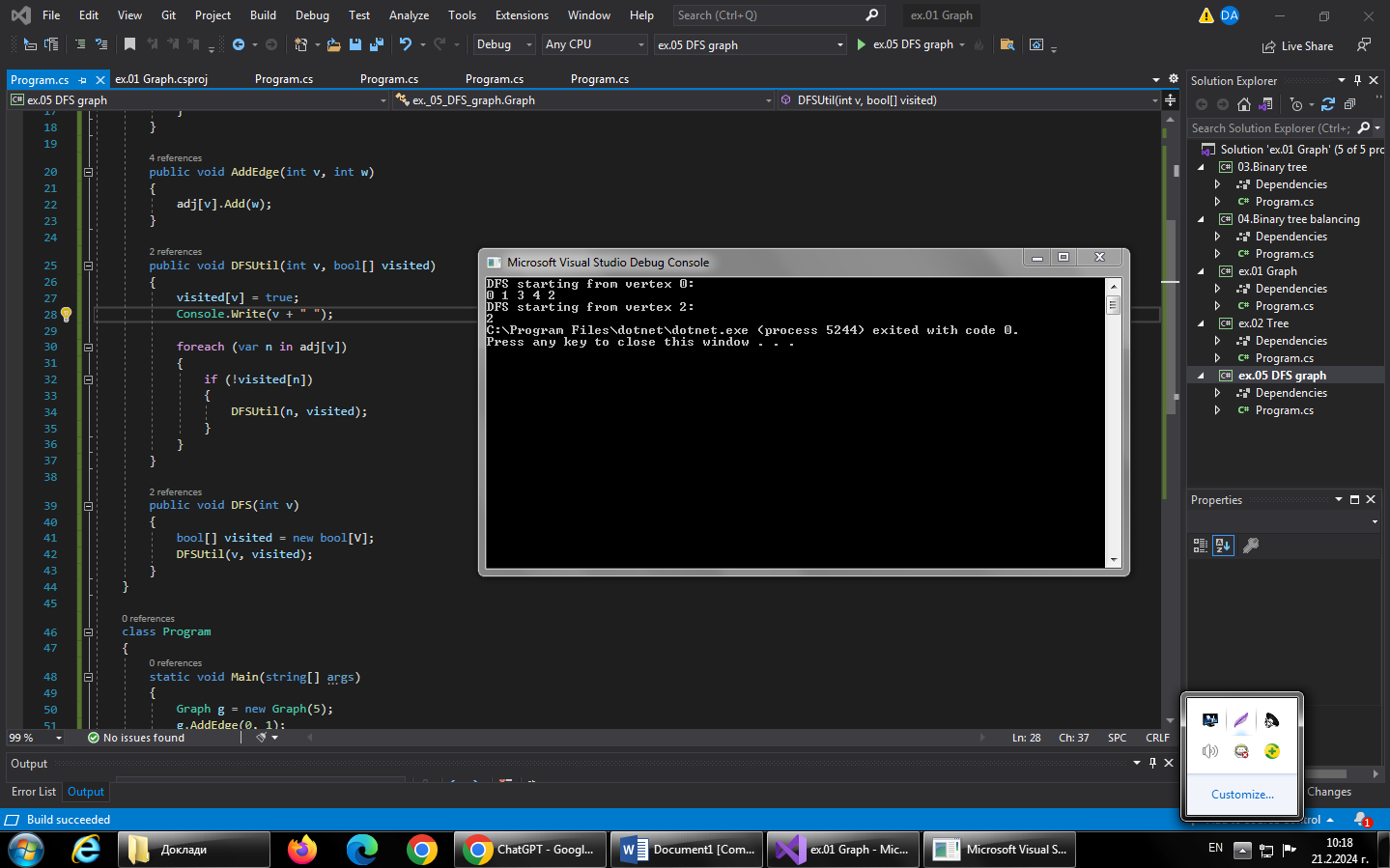
Обхождането в дълбочина (DFS - Depth-First Search) е един от основните алгоритми за обхождане на граф. Той използва стек или рекурсия, за да посети всички върхове на графа "в дълбочина", т.е. да се движи по всяка възможна връзка до край преди да се върне и да продължи със следващия връх.

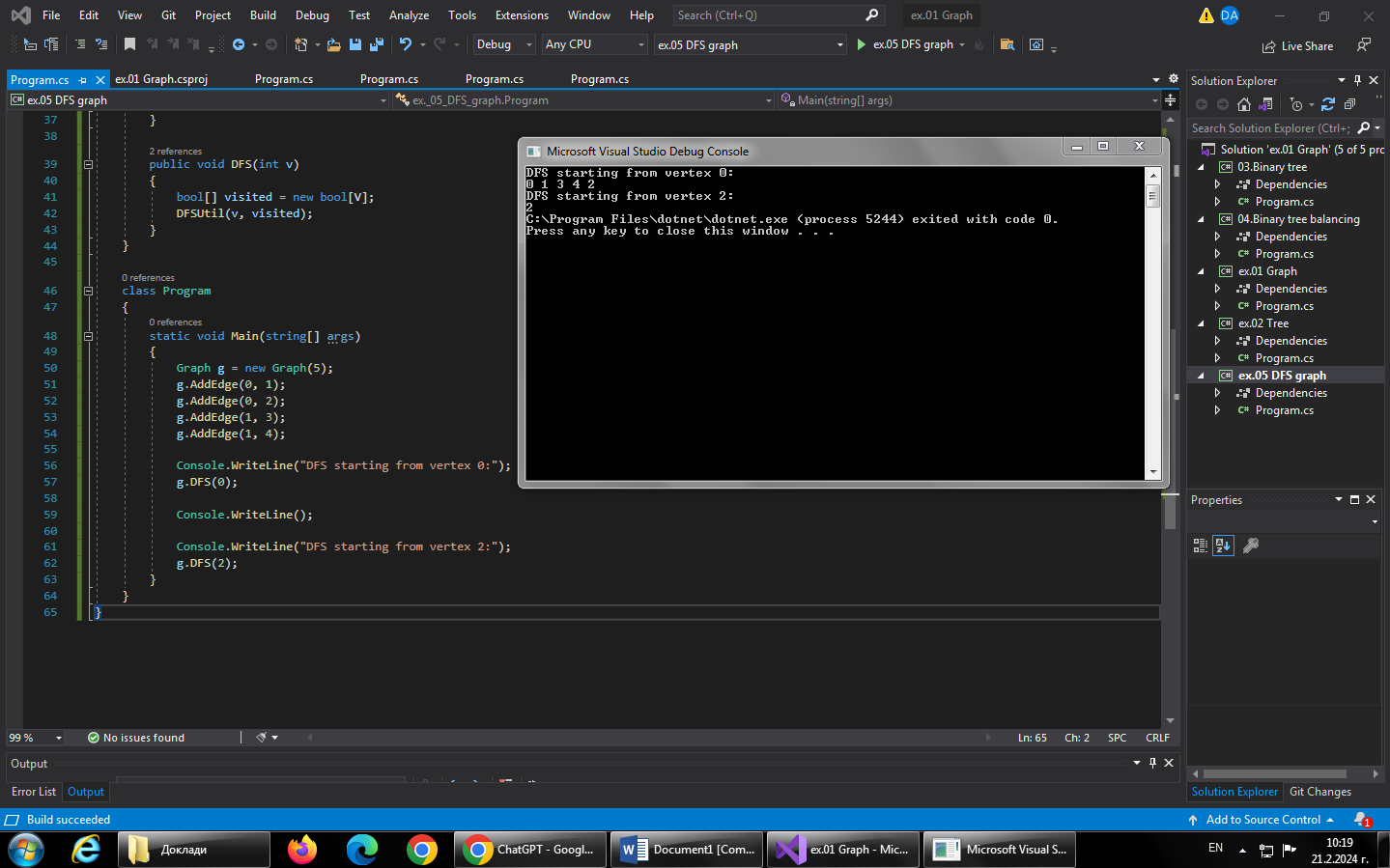
**Работа на алгоритъма:**

* **Избиране на начален връх:** За начало се избира произволен връх от графа като начален. Този връх се посещава и се маркира като посетен.
* **Посещение на съседите:** След това се избира един от непосетените съседи на текущия връх и се посещава. Този съсед става нов текущ връх.
* **Повтаряне на процеса:** Процесът се повтаря за новия текущ връх - избира се непосетен съсед, той се посещава и става нов текущ връх.
* **Рекурсивно или със стек:** Обикновено DFS се реализира рекурсивно, като се извиква функция за обхождане на всеки непосетен съсед. Другият начин е използване на стек за съхранение на върховете, които трябва да бъдат посетени.
* **Обратно след изчерпване на възможностите:** Процесът продължава докато не се стигне до връх, който няма непосетени съседи. Тогава се връща по стека или от рекурсията към предишния връх и продължава с останалите непосетени върхове.
* **Посетени върхове:** Всеки връх, който бъде посетен, може да бъде маркиран или използван за извеждане на някаква информация.

**Обхождане в дълбочина (DFS) на граф в C# - Пример:**

****

****

****

Този код дефинира клас Graph, който представя граф чрез списък на съседство. Методът DFSUtil е вътрешен метод, който се използва за рекурсивно обхождане на графа. Методът DFS е публичен метод, който стартира обхождането на графа от даден връх.

В главния метод Main се създава граф с 5 върха и се добавят ребра между тях. След това се извиква DFS методът, като се указва стартовият връх.

* DFS е основен алгоритъм за много задачи в графите, като намиране на пътища, цикли, проверка за свързаност и други. Той е мощен и ефективен, но трябва да се обърне внимание на възможностите за безкрайно рекурсивни цикли в некоректно дефинирани графи или в случай на големи графи, където може да се използва много памет.

1. **Обхождане в ширина на граф**

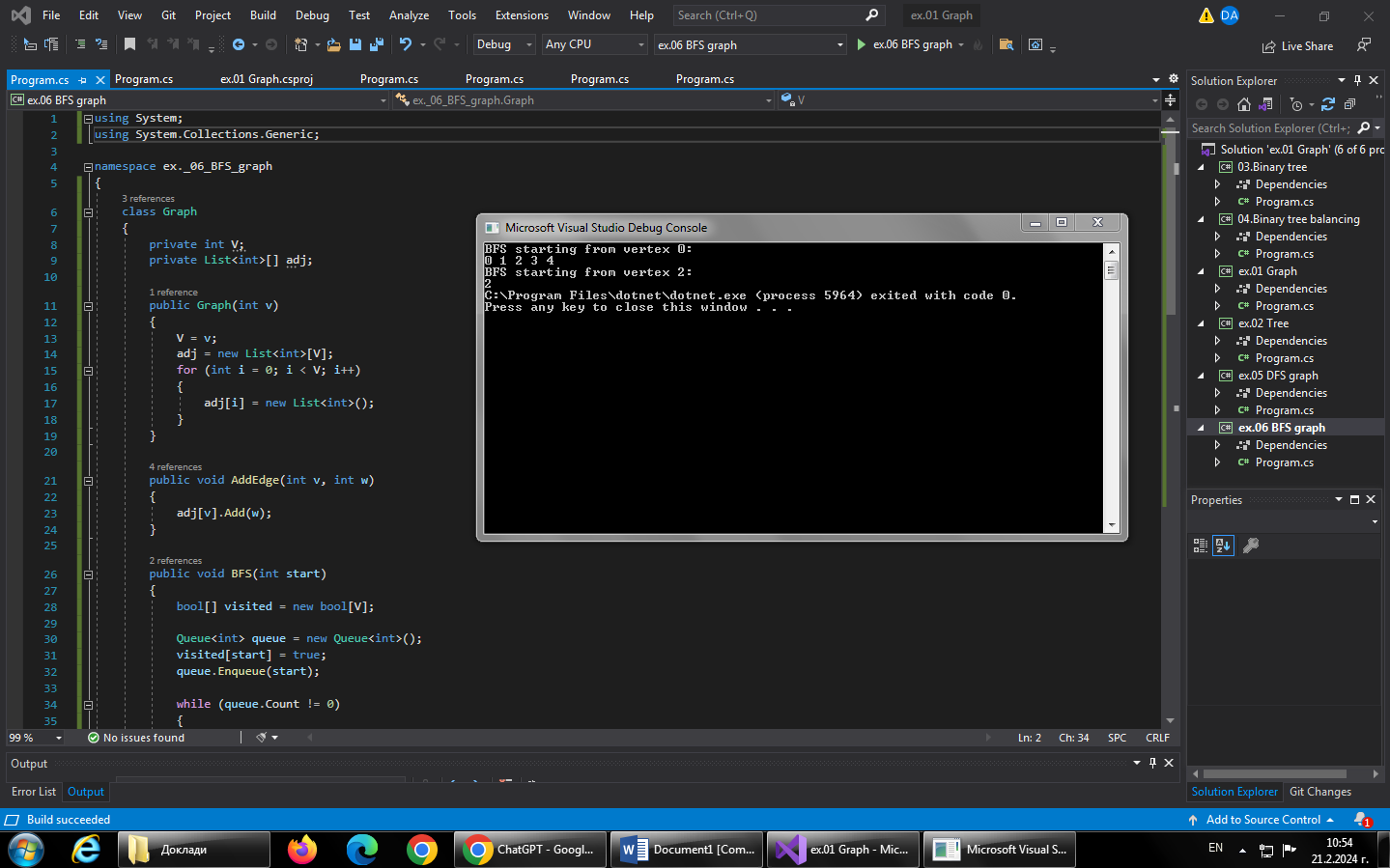
Обхождането в ширина (BFS - Breadth-First Search) на граф е алгоритъм за обхождане на всички върхове на графа, при което се стартира от даден начален връх и се посещават всички негови съседи преди да се продължи със следващите нива на съседство.

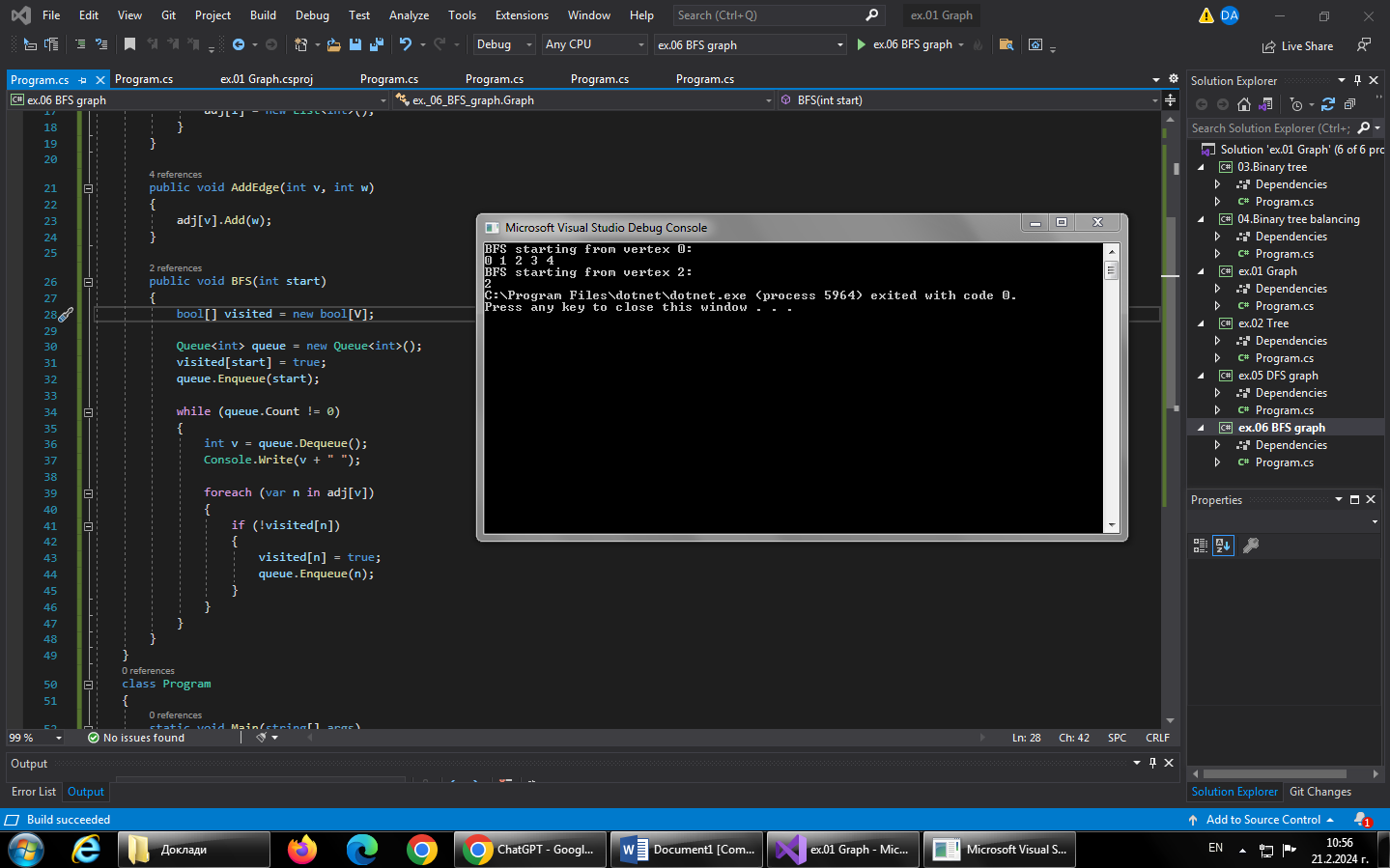
**Работа на алгоритъма:**

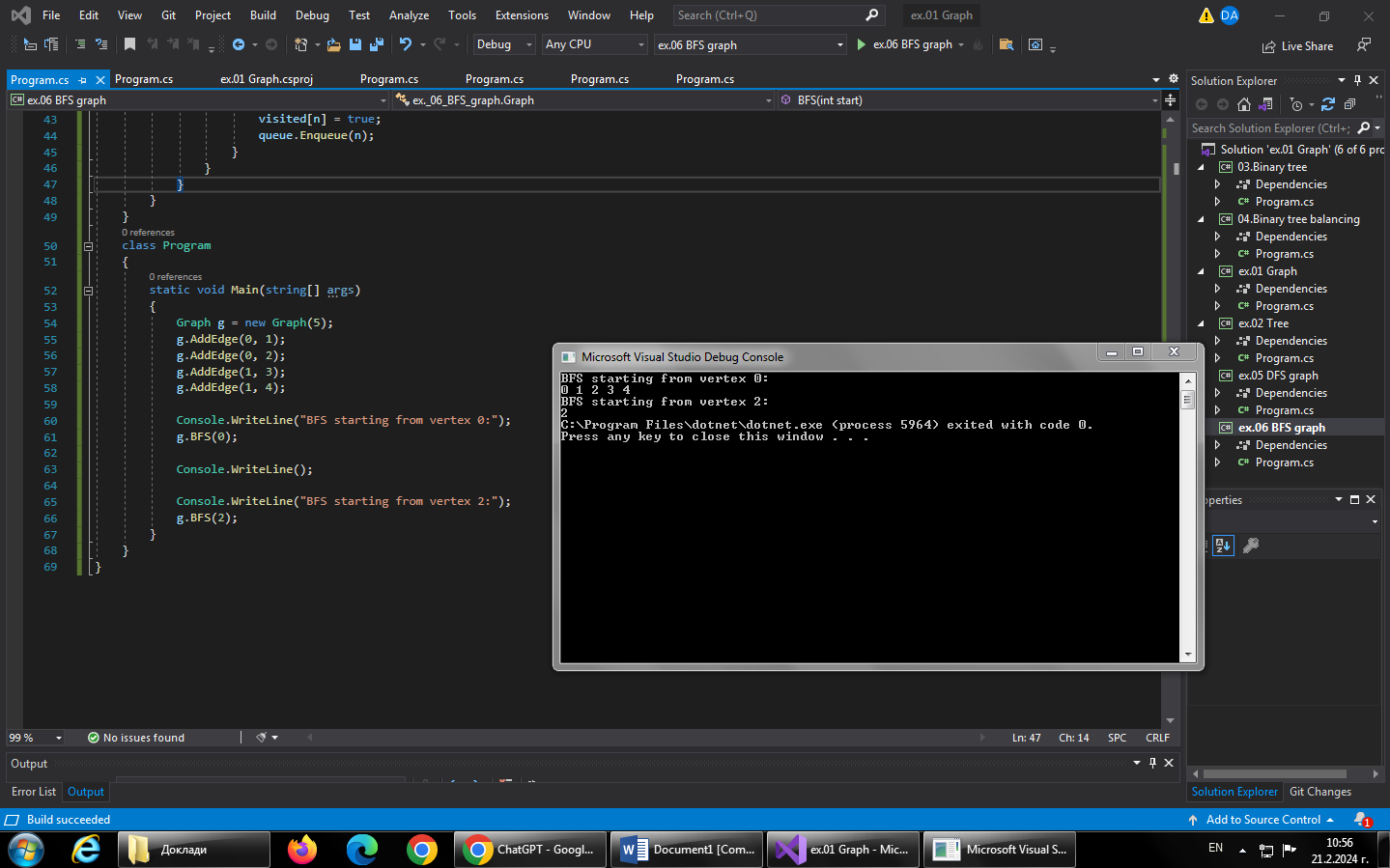
* **Избиране на начален връх:** За начало се избира произволен връх от графа като начален. Този връх се посещава и се маркира като посетен.
* **Посещение на съседите:** След това се посещават всички съседи на началния връх (всички върхове, до които има пряка връзка от началния връх). Тези съседи се маркират като посетени и се добавят в опашка.
* **Посещение на съседите на посетените върхове:** След като всички съседи на началния връх са посетени, се продължава с посещението на съседите на всички вече посетени върхове. Тези върхове също се маркират като посетени и се добавят в опашката.
* **Повтаряне на процеса:** Процесът се повтаря, докато не се посетят всички върхове на графа или докато опашката не бъде изпразнена.
* **Извеждане на върховете:** Посетените върхове се извеждат в реда, в който са посетени.

Този алгоритъм се нарича "в ширина", защото посещава всички върхове от едно ниво на графа преди да премине към следващото ниво.

**Обхождането в ширина (BFS) – Пример**

****

****

****

Този код дефинира клас Graph, който представя граф чрез списък на съседство. Методът AddEdge се използва за добавяне на връзка между два върха. Методът BFS изпълнява обхождане в ширина на графа от даден начален връх.

В главния метод Main се създава граф с 5 върха и се добавят няколко ребра между тях. След това се извиква методът BFS два пъти - първият път с начален връх 0, а вторият път с начален връх 2.

1. **Най-кратък път в граф по алгоритъма на Дейкстра**

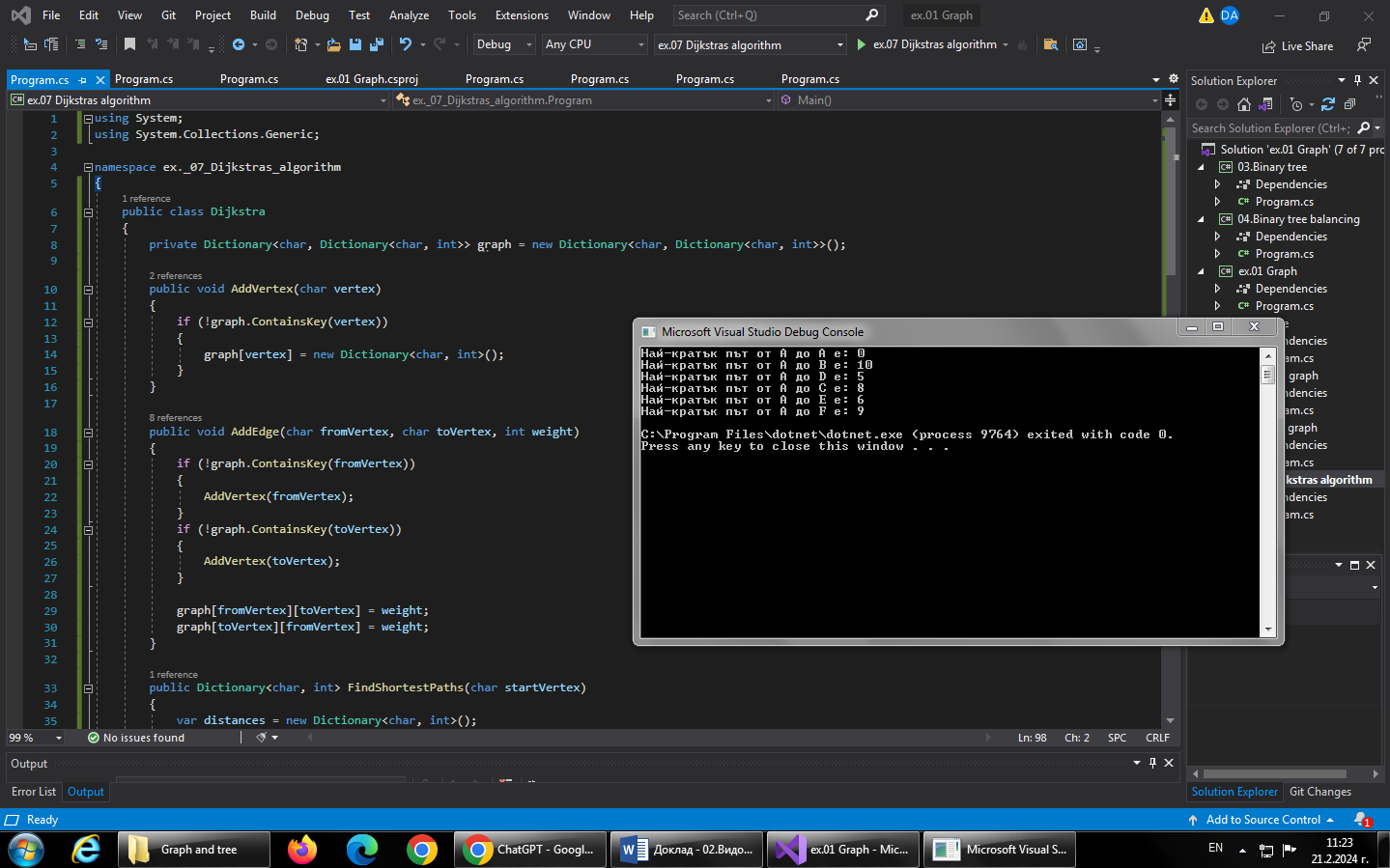
Алгоритъмът на Дийкстра се използва за намиране на най-кратък път между два върха в граф с неотрицателни тегла на ребрата.

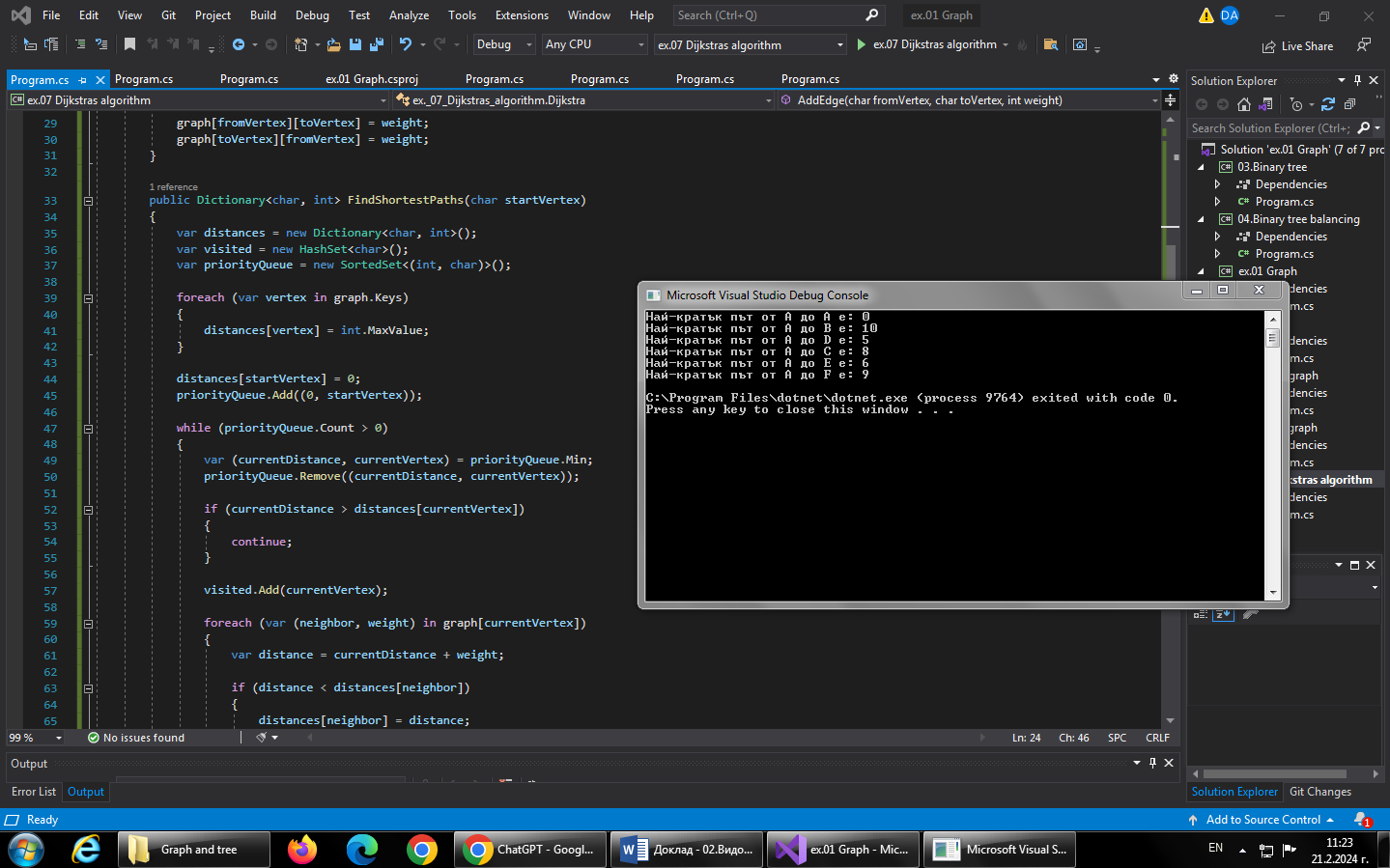
**Определение:**

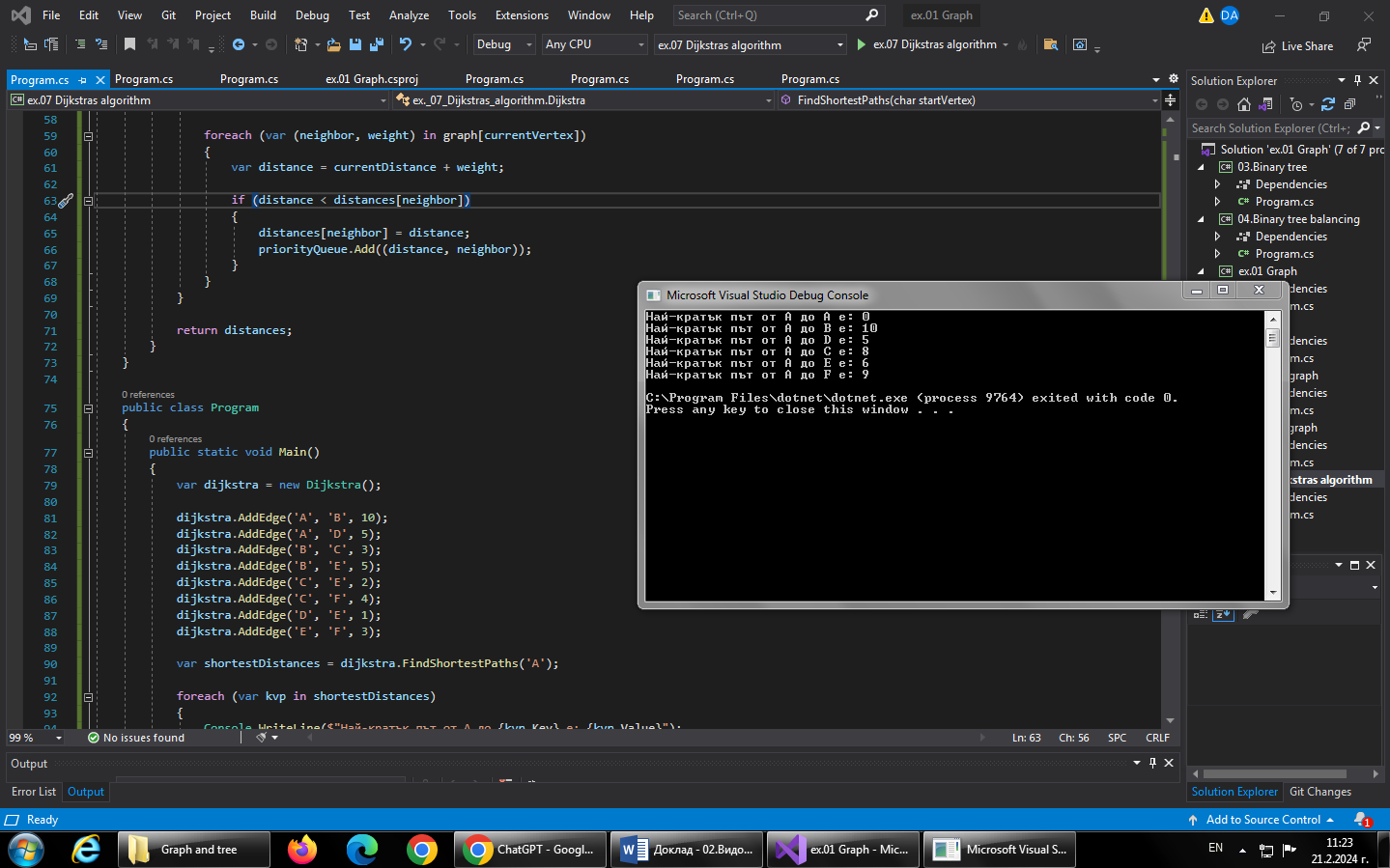
Даден е граф G с върхове V и ребра E. Нека имаме начален връх start и искаме да намерим най-краткия път от start до всеки друг връх в графа. Алгоритъмът на Дийкстра работи по следния начин:

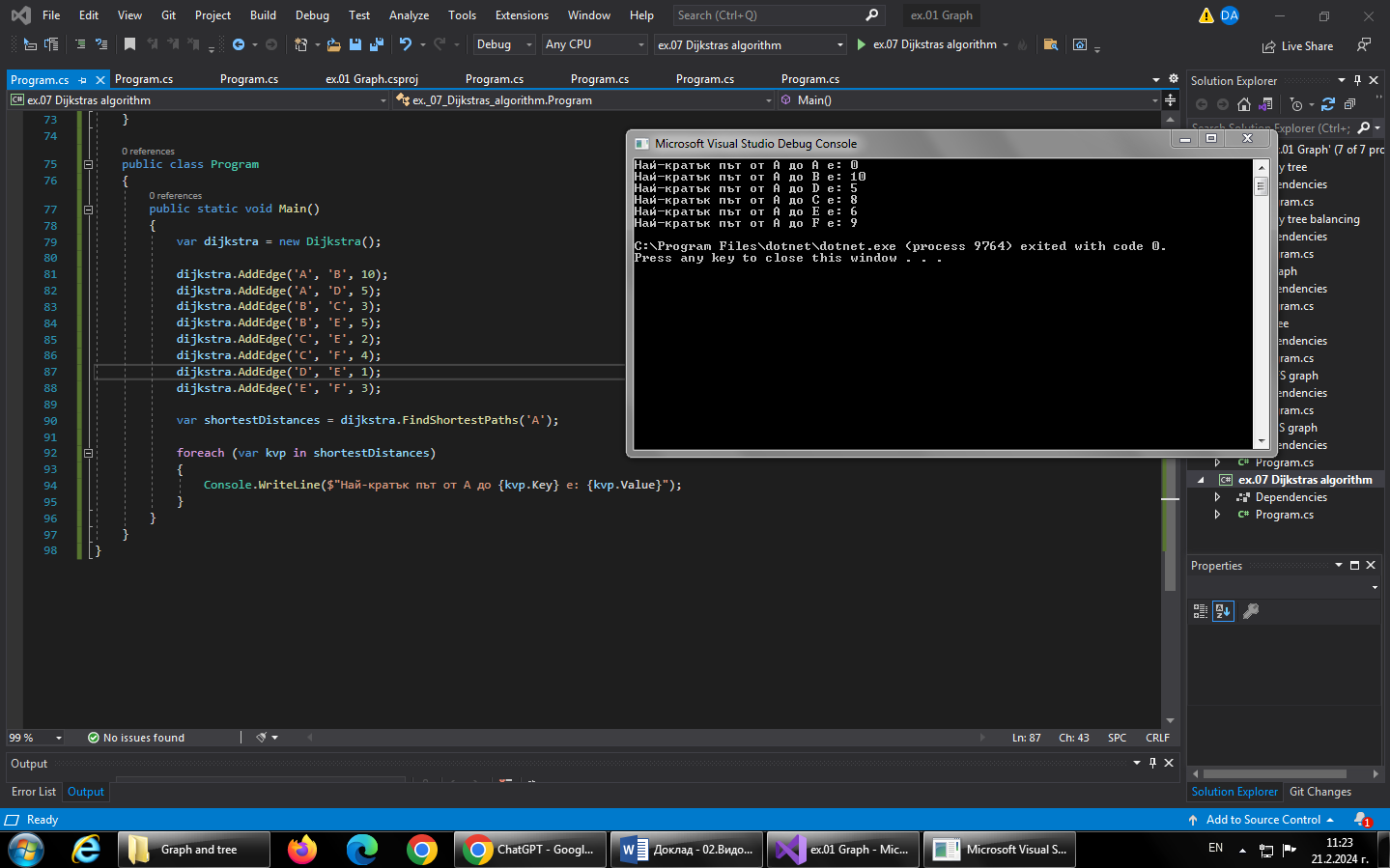
* Инициализира се масив dist с дължини на всички пътища до върховете в графа като безкрайност, освен за началния връх start, чиято дължина се инициализира на 0.
* Създава се празен множество visited за вече посетените върхове.
* Създава се приоритетна опашка (приоритетна опашка) pq с върховете на графа, сортирани по дължина на пътя до тях.
* Поставя се началният връх start в опашката с дължина 0.
* Повтаря се докато опашката не е празна:
* Изважда се връх current с най-малка дължина от опашката.
* За всеки съсед neighbor на current, който не е в visited:
* Пресмята се нова дължина new\_dist за пътя до neighbor като се добави дължината на реброто между current и neighbor към дължината на пътя до current.
* Ако new\_dist е по-малка от текущата дължина на пътя до neighbor в dist, тогава се актуализира dist за neighbor на new\_dist.
* neighbor се добавя в pq с новата дължина new\_dist.
* current се добавя в visited.
* Когато цикълът приключи, dist съдържа най-кратките пътища от start до всеки връх в графа.

**Най-кратък път в граф, изчислен по алгоритъма на Дейкстра в C#:- Пример**

****

****

****

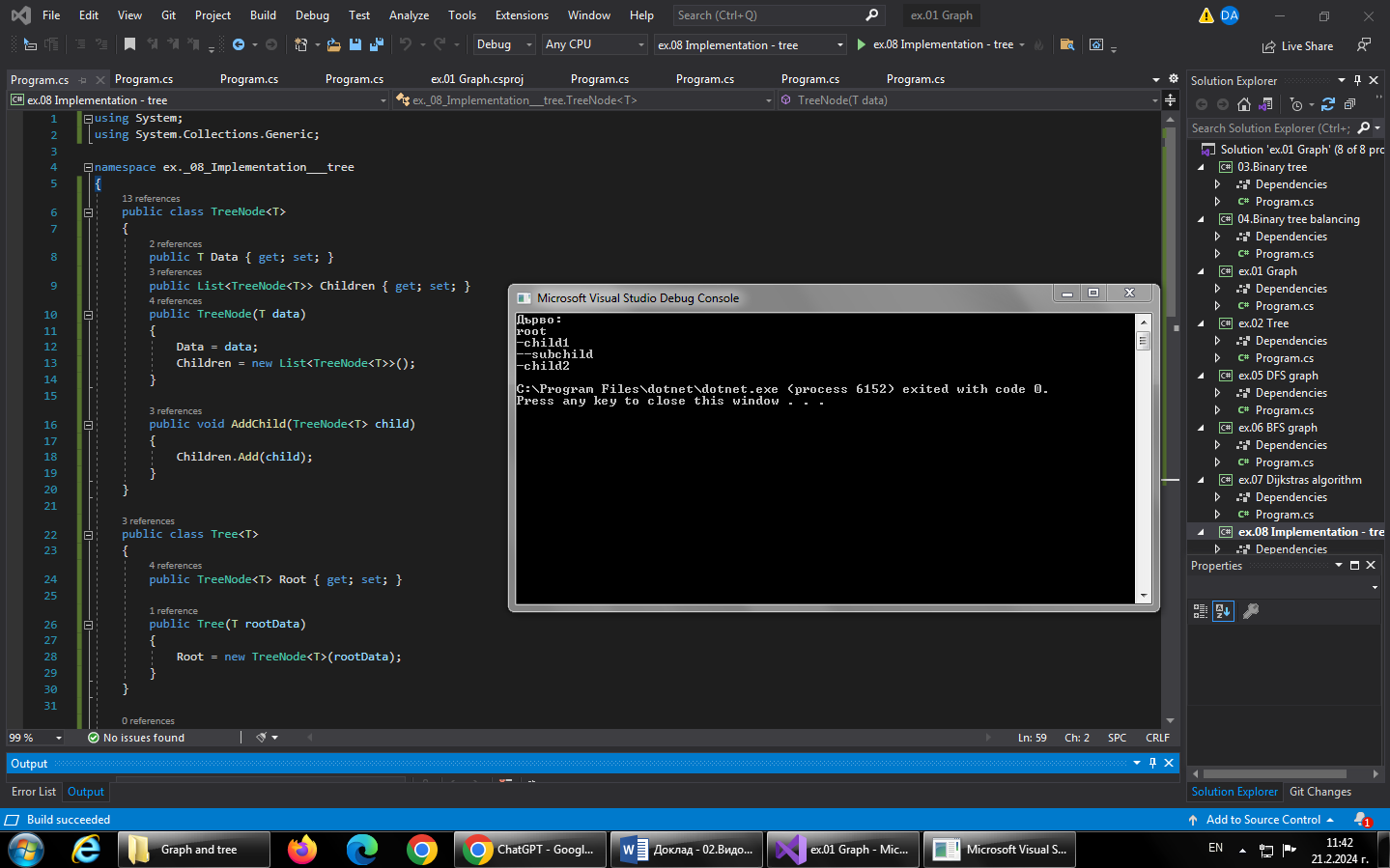
****

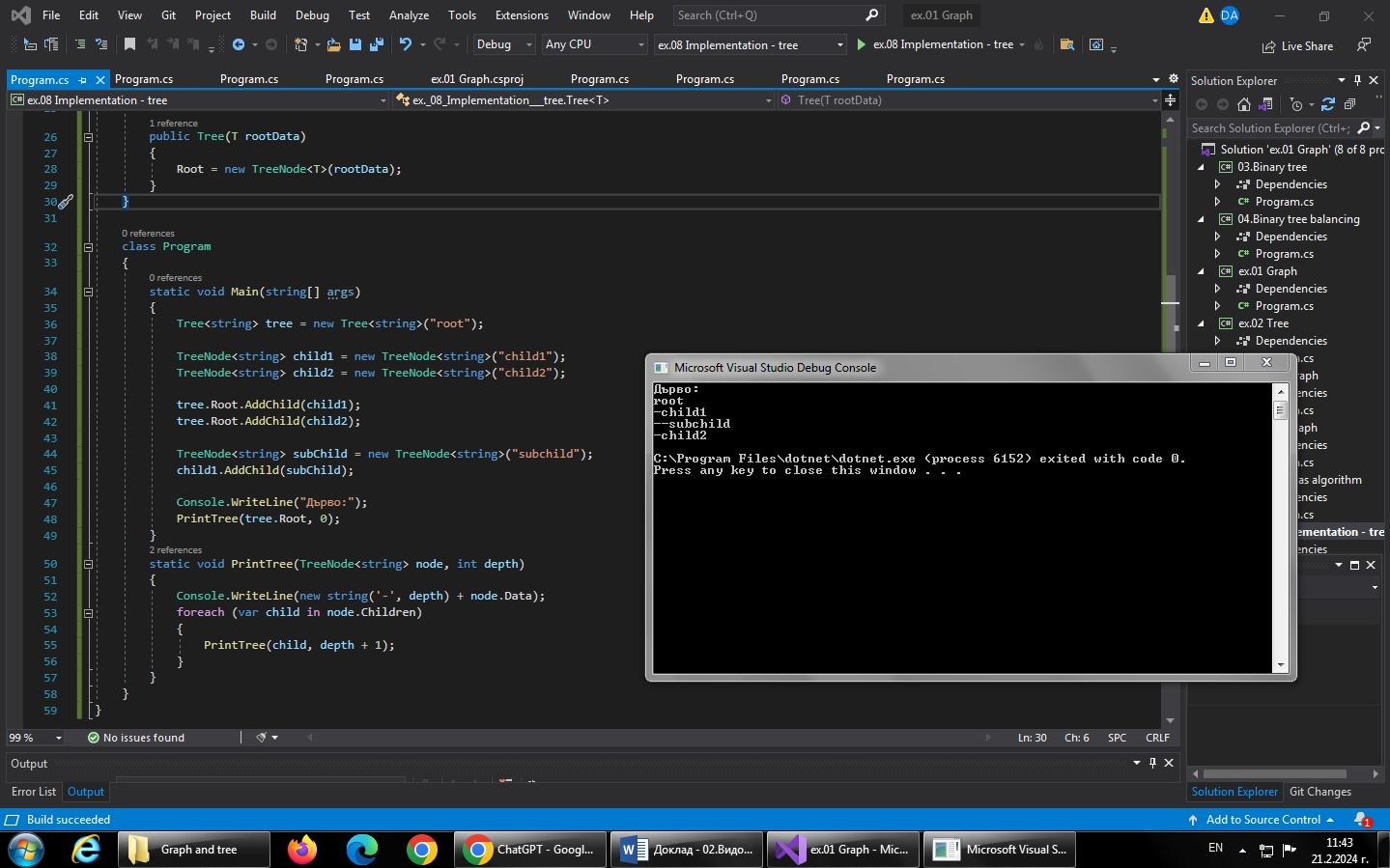
Този код ще намери най-кратките пътища от връх 'A' до всички останали върхове в графа и ще ги изведе на конзолата.

1. **Имплементация на дървета и графи**

**Примерни имплементации на дървета и графи в C#:**

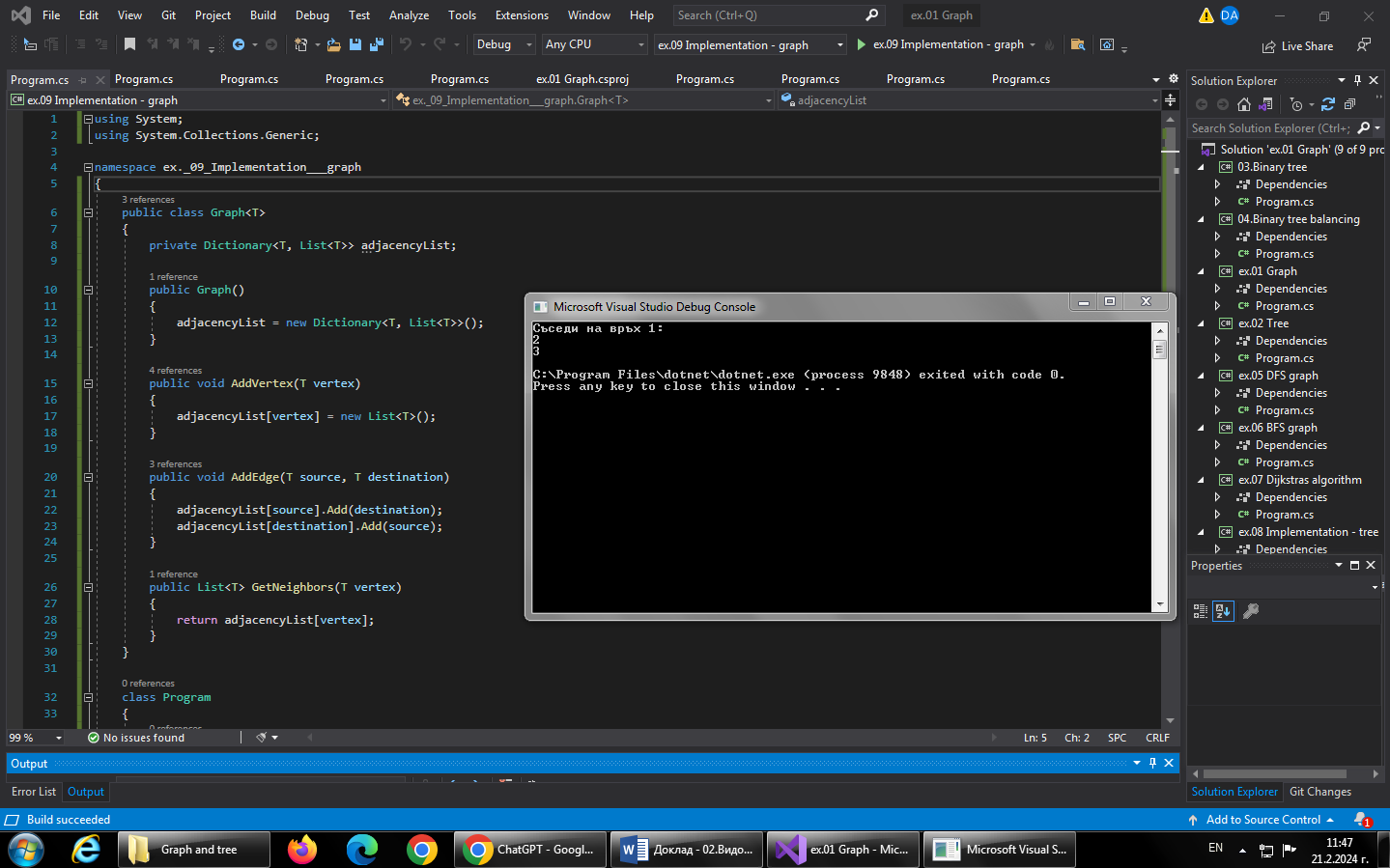
* Дърво:

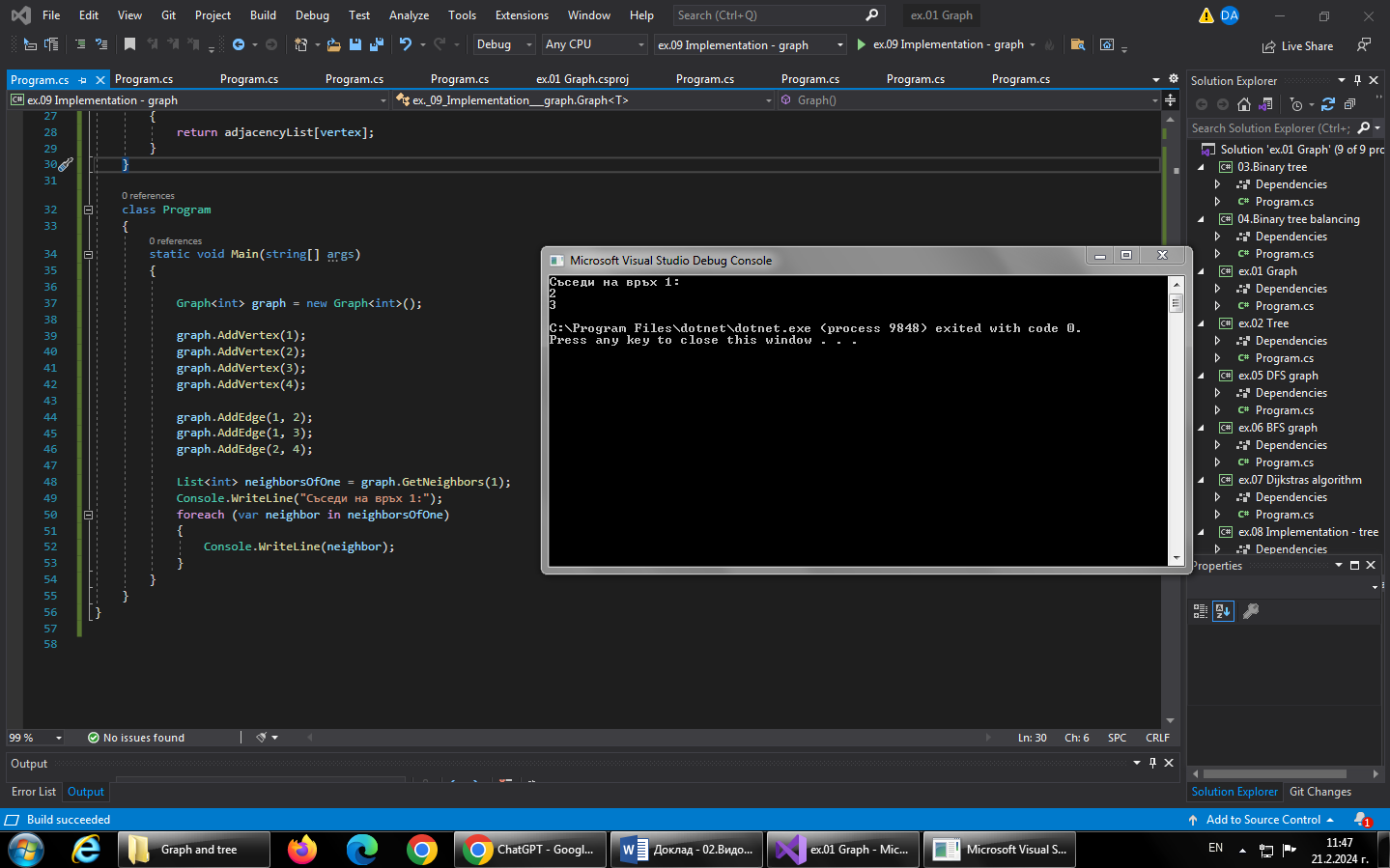
****

****

Този код ще създава дърво с корен "root", което има две деца - "child1" и "child2", и "child1" има дете "subchild".

* Граф:





**Използвана литература/Източници:**

<https://trekto.info/algoritmi-strukturi-danni/8-grafi/>

[https://bg.wikipedia.org/wiki/%D0%9E%D0%B1%D1%85%D0%BE%D0%B6%D0%B4%D0%B0%D0%BD%D0%B5\_%D0%B2\_%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D0%B](https://bg.wikipedia.org/wiki/%D0%9E%D0%B1%D1%85%D0%BE%D0%B6%D0%B4%D0%B0%D0%BD%D0%B5_%D0%B2_%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D0%25B)0

<https://medium.com/dvt32/algorithmo-5-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D1%8A%D0%BC-%D0%BD%D0%B0-%D0%B4%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0-%D0%BD%D0%B0%D0%B9-%D0%BA%D1%80%D0%B0%D1%82%D1%8A%D0%BA-%D0%BF%D1%8A%D1%82-%D0%BE%D1%82-%D0%B4%D0%B0%D0%B4%D0%B5%D0%BD-%D0%B2%D1%80%D1%8A%D1%85-%D0%B4%D0%BE-%D0%B2%D1%81%D0%B8%D1%87%D0%BA%D0%B8-%D0%BE%D1%81%D1%82%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8-%D0%B7%D0%B0-%D0%B3%D1%80%D0%B0%D1%84-%D1%81-e5056e745b96>

<https://bg.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D1%8A%D0%BC_%D0%BD%D0%B0_%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0>

<https://bg.wikipedia.org/wiki/%D0%9E%D0%B1%D1%85%D0%BE%D0%B6%D0%B4%D0%B0%D0%BD%D0%B5_%D0%B2_%D0%B4%D1%8A%D0%BB%D0%B1%D0%BE%D1%87%D0%B8%D0%BD%D0%B0>