Зміст

[Вступ 5](#_Toc390674269)

[Розділ 1. Задачі паралельного впорядкування та їх узагальнення 7](#_Toc390674270)

[1.1. Класичні постановки задач паралельного впорядкування 7](#_Toc390674271)

[1.2. Постановка узагальненої задачі паралельного впорядкування 9](#_Toc390674272)

[1.3. Складність алгоритмів розв’язання задач 12](#_Toc390674273)

[Розділ 2. Методи розв’язання узагальнених задач паралельного впорядкування 15](#_Toc390674274)

[2.1. Алгоритми, засновані на рівневому принципі 15](#_Toc390674275)

[2.2. Узагальнені задачі для часткового випадку 24](#_Toc390674276)

[2.3. Сітковий підхід до розв’язання задачі паралельного впорядкування 27](#_Toc390674277)

[Розділ 3. Програмна реалізація алгоритмів 34](#_Toc390674278)

[3.1. Опис програми 34](#_Toc390674279)

[3.2. Керівництво користувача 37](#_Toc390674280)

[3.3. Результати тестування програми 42](#_Toc390674281)

[3.4. Аналіз отриманих в програмі даних 48](#_Toc390674282)

[Висновки 59](#_Toc390674283)

[Розділ 4. Охорона праці 60](#_Toc390674284)

[4.1. Вимоги до організації робочого місця 61](#_Toc390674285)

[4.2. Опис робочого місця 61](#_Toc390674286)

[4.3. Освітленість 62](#_Toc390674287)

[4.4. Параметри мікроклімату 63](#_Toc390674288)

[4.5. Електробезпека на робочому місці 64](#_Toc390674289)

[4.6. Шум 65](#_Toc390674290)

[Висновок 66](#_Toc390674291)

[Список літератури 67](#_Toc390674292)

[Додаток А 69](#_Toc390674293)

# Вступ

Серед задач дискретної оптимізації особливе місце займають задачі теорії впорядкувань.  Оптимальні розв’язки спеціальних дискретних задач на графах дозволяють економити ресурси у системах, моделі яких являють собою завдання, на послідовність виконання яких накладено умови часткового порядку. Прикладами таких моделей можуть бути множини робіт, які треба завершити за мінімальний термін при заданому обсязі ресурсів, послідовності операцій, які виконуються багатопроцесорними ЕОМ [1][2]. Актуальність дослідження проблеми оптимального вирішення узагальнених задач упорядкування зумовлена активним розвитком багатопроцесорних ЕМО, багатоядерних процесорів, а також розвитком хмарних обчислень, які також є розпаралеленими.

Задача паралельного впорядкування у класичній постановці передбачає, що на кожному кроці виконання робіт кількість виконавців є сталою. Але у більшості реальних ситуацій вона може змінюватися з часом. Так, якщо ресурсами є, наприклад, робітники, то їх кількість буде змінюватися у зв’язку з тим, що дехто йде у планову відпустку. Якщо виконуються довготривалі обчислення, то зайнятість процесорів ЕОМ може змінюватися протягом часу. Тому особливої уваги заслуговує узагальнена задача паралельного впорядкування. Вона дозволяє вирішувати оптимізаційні задачі у випадку, коли кількість виконавців змінюється з часом.

В цілому, задачі паралельного впорядкування як у класичні постановці, так і узагальнені, відносяться до класу -повних задач. Тобто вважається вірною гіпотеза, що не існує алгоритмів поліноміальної складності, що розв’язують іі точно.

Для узагальненої задачі паралельного впорядкування досліджувалася точність відомих алгоритмів поліноміальної складності, розроблених для класичної задачі.

В моделях, що вивчаються, обмеження на порядок виконання робіт представляються у вигляді графів. Для класичної задачі були виділені підкласи графів, для яких існують точні алгоритмі поліноміальної складності. Важливим є виділення задач, для яких ці алгоритми будуть точними і для узагальненої задачі.

Роботи в напрямку розв’язання задач **побудови оптимального паралельного упорядкування вершин ациклічного орграфа** та відповідних задач теорії розкладів проводяться багатьма науковцями. Серед них такі вчені, як Ху Т., Фуджі М., Гері М., Джонсон Д., Ленстра Д., Танаєв В.С., Шкурба В.В., Кукса А.І., Бурдюк В.Я. та інші.

Робота складається з чотирьох розділів. В першому розглядається постановка задачі паралельного впорядкування у класичній постановці, а також постановка узагальненої задачі, досліджується складність алгоритмів, що розв’язують відповідні задачі. В другому розглядаються алгоритми розв’язання узагальненої задачі паралельного впорядкування, а також формулюються умови, за яких відомі методи для розв’язання задачі у класичній постановці будуть точними для узагальненої задачі. Третій розділ присвячений програмній реалізації алгоритму, основаному на рівневому принципі для узагальненої задачі, а також аналізу отриманих результатів. У четвертому розділі розглядаються питання, пов’язані з охороно праці, що стосуються нормування параметрів робочого приміщення.

# Розділ 1. Задачі паралельного впорядкування та їх узагальнення

* 1. **Класичні постановки задач паралельного впорядкування**

Впорядкуванням скінченої множини , що містить елементів, будемо називати розміщення цих елементів по місцям, розташованих в лінію так, що кожен елемент з розміщується лише на одному місці.

Кількість непустих місць в впорядкуванні називається довжиною упорядкування та позначається

Нехай – множина тих елементів з , які розташовані в на -му місці.

Шириною впорядкування називається кількість елементів найбільшої по потужності множини , і позначається

Введемо до розгляду граф , де – множина вершин, а – множина дуг.

Впорядкування множини вершин графу назвемо паралельним впорядкуванням вершин орграфу , якщо з того, що слідує, що розташовано в лівіше за .

Для існування паралельного впорядкування необхідно і достатньо, або граф був ациклічним [3].

Для прикладу, розглянемо граф, зображений на рис. 1.1.

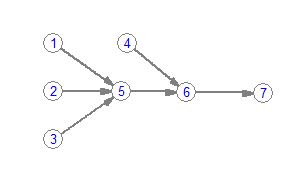


Рис.1.1. Вигляд графу

Побудуємо деякі паралельні впорядкування вершин цього графу. Їх зображено на рис. 2 та рис. 3.

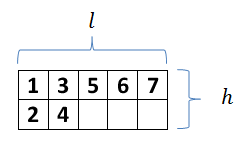


Рис. 1.2. Паралельне впорядкування вершин графу *G*

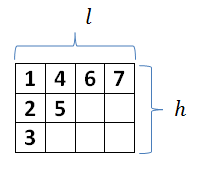


Рис. 1.3. Паралельне впорядкування вершин графу *G*

Як бачимо, при довільному допустимому розміщенні вершин графу *G* ми можемо отримувати допустимі впорядкування різної довжини та ширини.

Будемо називати впорядкування оптимальним, якщо воно або при заданій довжині має мінімальну ширину, або при заданій ширині має мінімальну довжину. Позначимо описані задачі відповідно та [4, 5].

Розглянемо питання про існування точного розв’язку для загальної задачі паралельного впорядкування, а саме задачу побудови паралельного впорядкування для орграфу , де .

Розглянемо алгоритм побудови деякого впорядкування

Алгоритм

1. Вважаємо в всі місця пустими.
2. В орграфі шукаємо вершини, які не мають вхідних дуг і записуємо їх на к-те місце в . Викреслюємо ці вершини та дуги, що з них виходять. Граф, що залишився, позначимо .
3. Якщо множина вершин графу пуста, то кінець.
4. , переходимо на крок 3, обираючи в якості орграфу орграф .

Стверджуємо, що впорядкування, побудоване для даної задачі, є оптимальним.

Довжину отриманого впорядкування позначимо , а саме впорядкування - . Впорядкування буде також оптимальним, якщо шукане впорядкування будувати справа наліво, тобто на кожному кроці шукати в орграфі вершини, що не мають вхідних дуг, і розміщувати їх починаючи з місця . Таке впорядкування позначатимемо [6].

Таким чином, ми отримали деякі точні рішення для задачі паралельного впорядкування, але лише для випадку, коли по суті не обмежене.

Впорядкування та знадобляться нам при розв’язанні і узагальнених задач паралельного впорядкування.

* 1. **Постановка узагальненої задачі паралельного впорядкування**

Введемо деякі обмеження на вигляд впорядкування . Розглянемо вектор . Тоді будемо будувати впорядкування так, щоб потужність множини не перевищувала , .

Для прикладу розглянемо деякий граф , що зображений на рис. 1.4.

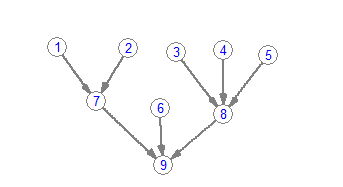


Рис. 1.4. Вигляд графу

Задамо цілочисельний вектор .

Побудуємо деякі впорядкування так, щоб потужність множини не перевищувала , . Наприклад так, як записано в таблицях 1.1 і 1.2:

*Таблиця 1.1.*

Допустиме впорядкування *S1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *i=1* | 1 | 2 |  |  |  |
| *i=2* | 3 | 4 | 5 | 6 | 7 |
| *i=3* | 8 |  |  |  |  |
| *i=4* | 9 |  |  |  |  |

*Таблиця 1.2.*

Допустиме впорядкування *S2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 3 | 4 |  |  |
| *i=2* | 1 | 2 | 5 | 6 |
| *i=3* | 7 |  |  |  |
| *i=4* | 8 |  |  |  |
| *i=5* | 9 |  |  |  |

Розглянемо також деякий граф , що зображений на рис. 1.5.

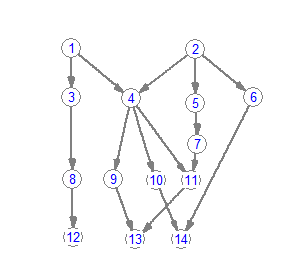


Рис.1.5. Вигляд графу

Задамо вектор .

Побудуємо деякі впорядкування так, щоб потужність множини не перевищувала , . Наприклад так, як записано в таблицях 1.3 і 1.4:

*Таблиця 1.3.*

Допустиме впорядкування *S1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 1 |  |  |  |
| *i=2* | 2 | 3 |  |  |
| *i=3* | 4 | 5 | 6 |  |
| *i=4* | 8 | 9 | 10 | 7 |
| *i=5* | 11 | 12 |  |  |
| *i=6* | 13 | 14 |  |  |

*Таблиця 1.4.*

Допустиме впорядкування *S2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 2 |  |  |  |
| *i=2* | 6 | 5 |  |  |
| *i=3* | 1 | 7 |  |  |
| *i=4* | 3 | 4 |  |  |
| *i=5* | 8 | 9 |  |  |
| *i=6* | 10 | 11 |  |  |
| *i=7* | 12 |  |  |  |
| *i=8* | 13 |  |  |  |
| *i=9* | 14 |  |  |  |

Бачимо, що відповідні впорядкування можуть мати різні довжини.

Будемо називати впорядкування оптимальним, якщо воно має мінімальну довжину.

Позначимо відповідну задачу .

* 1. **Складність алгоритмів розв’язання задач**

Для початку розглянемо клас переборних задач.

Перебірна задача визначається як множина об’єктів, що називаються індивідуальними задачами, причому для кожної індивідуальної задачі відома множина рішень задачі.

Будемо говорити, що алгоритм вирішує переборну задачу, якщо, отримавши на вхід довільну індивідуальну задачу, він видає відповідь «ні», якщо множина рішень задачі пуста, і видає деяке рішення з множини рішень в супротивному випадку.

Розглянемо клас NP всіх перебірних задач, і клас Р перебірних задач, що вирішуються за поліноміальний час на машині Тюрінга. Очевидно, що . Центральним є питання, чи співпадають класи .

В класі виявлені так звані універсальні (-повні) задачі, до яких поліноміально зводиться будь-яка задача з . Якщо б вдалося довести, що хоча б одна з -повних задач належить до класу , то було б доведено, що . Тоді б можна було сподіватися на побудову ефективних алгоритмів для різних класів дискретних задач. На сьогоднішній день вважається вірною гіпотеза, що класи різні, тому доводиться розробляти ефективні алгоритми для кожного з підкласу задач.

З практичного досвіду відомо, що -повні задачі та задачі з класу сильно відрізняються по складності вирішення, однак ця різниця формально не доведена.

Часова складність алгоритму відображує затрати часу, що необхідні для його роботи. Це функція, котра кожній вхідній довжині n ставить у відповідність максимальний з всіх індивідуальних задач довжини n, час, що затрачується алгоритмом на рішення індивідуальних задач цієї довжини.

Для порівняння ефективності алгоритмів розрізняють поліноміальні та експоненційні алгоритми.

Будемо говорити, що функція є , якщо існує константа с така, що для всіх значень . Поліноміальним алгоритмом називається алгоритм, у якого часова складність дорівнює , де – деяка поліноміальна функція, а – вхідна довжина. Алгоритми, часова складність яких не піддається подібній оцінці, називаються експоненційними [7].

Різниця між вказаними типами алгоритмів особливо помітна при вирішення задач великої розмірності. Таблиця 1.5 дозволяє порівняти швидкості росту деяких типових поліноміальних і експоненційних функцій [4].

*Таблиця 1.5.*

Порівняння декількох поліноміальних і експоненційних функцій часової складності

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Розмір | | | | | | |
| Функція часової складності |  | 10 | 20 | 30 | 40 | 50 | 60 |
|  | 0.00001 с | 0.00001 с | 0.00003 с | 0.00004 с | 0.00005 с | 0.00006 с |
|  | 0.0001 с | 0.0004 с | 0.0009 с | 0.0016 с | 0.0025 с | 0.0036 с |
|  | 0.001 с | 0.008 с | 0.027 с | 0.064 с | 0.125 с | 0.216 с |
|  | 0.1 с | 3.2 с | 24.3 с | 1.7 хв. | 5.2 хв. | 13.0 хв. |
|  | 0.001 с | 1.0 с | 17.9 хв. | 12.7 днів | 35.7 років | 366 ст. |
|  | 0.059 с | 58 хв. | 6.5 років | 3855 ст. | ст. | століть |

Ця таблиця показує, чому віддають перевагу поліноміальним алгоритмам перед експоненційними. Більшість експоненційних алгоритмів – це просто варіанти направленого перебору, тоді як поліноміальні алгоритми вдається побудувати, глибоко проаналізувавши суть задачі, що вирішується.

Вважається, що задача вважається «добре розв'язуваною» лише тоді, коли для неї отримано поліноміальний алгоритм. Тому будемо називати задачу важко вирішуваною, якщо для її рішення не існує поліноміального алгоритму.

Варто зазначити, що існують деякі експоненційні алгоритми, що досить добре працюють на практиці, адже часова складність визначена як міра поведінки алгоритму в найгіршому випадку, а може виявитися, що більшість індивідуальних задач потребують для свого вирішення значно менших затрат часу.

Важливо вміти визначати, чи являється конкретна задача NP-повною, тому що якщо являється, то нам, скоріш за все, знадобиться евристичний метод, що дозволяє знайти наближення до оптимального рішення. Якщо ж задача вирішується за поліноміальний час, то варто намагатися знайти точне рішення.

Загальна задача паралельного упорядкування відноситься до класу NP-складних задач, тому виникає необхідність побудови та дослідження не лише точних, а і наближених алгоритмів побудови впорядкувань.

Відомі точні алгоритмі поліноміальної складності розв’язання задачі в класичній постановці, якщо кількість виконавців та граф не має транзитивних дуг, а також якщо довільне, але граф обмежень представляється у вигляді лісу, або має доповненням триангулярний граф. Для випадку, коли граф обмежень довільного вигляду, а задача належить класу NP. Можливо, існує деяке фіксоване значення , для якого задача буде поліноміально розв’язною, але поки що такі факти не встановлені.

При задача стає NP-повною, якщо час виконання завдань приймає значення 1 або 2.

**Розділ 2. Методи розв’язання узагальнених задач паралельного впорядкування**

* 1. **Алгоритми, засновані на рівневому принципі**

Нехай задано граф . Побудуємо для нього впорядкування . Характеризувати рівень вершини буде величина, що обчислюється за формулою:

де довжина впорядкування ;

- номер місця вершини в упорядкуванні.

Вільною будемо називати таку вершину, попередники якої вже є в упорядкуванні. При побудові впорядкування будемо брати ту вільну вершину, яка має максимальний рівень. Якщо таких вершин декілька – беремо будь-яку. Цей точний алгоритм поліноміальної складності був розроблений для дерева (лісу) для випадку, коли .

Розглянемо деякий довільний граф. Наприклад, зображений на рис. 2.1 граф :

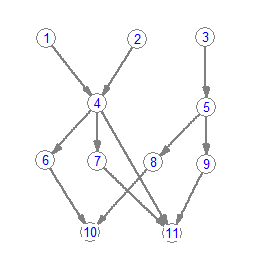


Рис. 2.1. Вигляд графу

Задамо деяку послідовність цілих чисел - вектор .

Побудуємо впорядкування згідно з рівневим принципом. Для цього побудуємо для впорядкування , що зображене в таблиці 2.1. Номер строки *k* характеризує номер місця вершини в упорядкуванні, а відповідно – і її рівень.

*Таблиця 2.1.*

Впорядкування вершин графу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *k=1* | 1 | 2 | 3 |  |
| *k=2* | 4 | 5 |  |  |
| *k=3* | 6 | 7 | 8 | 9 |
| *k=4* | 10 | 11 |  |  |

Тепер побудуємо деякі впорядкування згідно з рівневим принципом. Їх зображено в таблиці 2.2 та 2.3.

*Таблиця 2.2.*

Впорядкування вершин графу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 3 |  |  |  |
| *i=2* | 1 | 2 | 5 |  |
| *i=3* | 4 | 8 | 9 |  |
| *i=4* | 6 | 7 |  |  |
| *i=5* | 10 |  |  |  |
| *i=6* | 11 |  |  |  |

*Таблиця 2.3.*

Впорядкування вершин графу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 1 |  |  |  |
| *i=2* | 2 | 3 |  |  |
| *i=3* | 4 | 5 |  |  |
| *i=4* | 8 | 9 |  |  |
| *i=5* | 6 |  |  |  |
| *i=6* | 7 |  |  |  |
| *i=7* | 10 |  |  |  |
| *i=8* | 11 |  |  |  |

В обох впорядкуваннях вершини заносилися в упорядкування згідно з рівневий принципом. Але в першому випадку довжина впорядкування , а в другому - .

З’ясуємо, чи можливо, керуючись рівневим принципом, побудувати неоптимальне впорядкування для випадку, коли заданий граф являється деревом.

Розглянемо зображений на рис. 2.2 граф . Задамо деякий вектор .

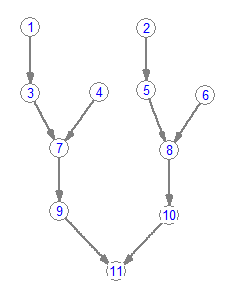


Рис. 2.2. Вигляд графу

Побудуємо впорядкування згідно з рівневим принципом. Для цього побудуємо для впорядкування , що зображене в таблиці 2.4.

*Таблиця 2.4.*

Впорядкування вершин графу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *k=1* | 1 | 2 |  |  |
| *k=2* | 3 | 4 | 5 | 6 |
| *k=3* | 7 | 8 |  |  |
| *k=4* | 9 | 10 |  |  |
| *k=5* | 11 |  |  |  |

Побудуємо деякі впорядкування згідно з рівневим принципом. Їх зображено в таблиці 2.5 та 2.6.

*Таблиця 2.5.*

Впорядкування вершин графу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 1 | 2 |  |  |
| *i=2* | 4 | 6 |  |  |
| *i=3* | 3 | 5 |  |  |
| *i=4* | 7 |  |  |  |
| *i=5* | 8 | 9 |  |  |
| *i=6* | 10 |  |  |  |
| *i=7* | 11 |  |  |  |

*Таблиця 2.6.*

Впорядкування вершин графу

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 1 | 2 |  |  |
| *i=2* | 3 | 4 |  |  |
| *i=3* | 5 | 6 | 7 |  |
| *i=4* | 8 |  |  |  |
| *i=5* | 9 | 10 |  |  |
| *i=6* | 11 |  |  |  |

В першому випадку довжина впорядкування , а в другому - . Як бачимо, різниця в довжині викликана різним вибором вершин одного рівня.

Розглянемо ще одне дерево , зображене на рис. 2.3. Задамо вектор .

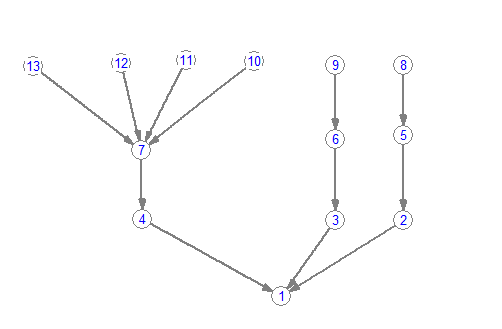


Рис. 2.3. Вигляд графу

Побудуємо для впорядкування , що зображене в таблиці 2.7.

*Таблиця 2.7.*

Впорядкування вершин графу

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *k=1* | 8 | 9 | 10 | 11 | 12 | 13 |
| *k=2* | 5 | 6 | 7 |  |  |  |
| *k=3* | 2 | 3 | 4 |  |  |  |
| *k=4* | 1 |  |  |  |  |  |

Побудуємо деякі впорядкування: перше згідно з рівневим принципом, друге – довільно, з порушенням принципу. Отримані впорядкування зображено відповідно в таблицях 2.8 та 2.9.

*Таблиця 2.8.*

Впорядкування вершин графу , побудоване згідно з рівневим принципом

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 8 | 9 | 10 | 11 |
| *i=2* | 12 | 13 |  |  |
| *i=3* | 5 | 6 | 7 |  |
| *i=4* | 3 |  |  |  |
| *i=5* | 2 |  |  |  |
| *i=6* | 4 |  |  |  |
| *i=7* | 1 |  |  |  |

*Таблиця 2.9.*

Впорядкування вершин графу , побудоване довільно

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *i=1* | 8 | 9 | 10 | 11 |
| *i=2* | 5 | 6 |  |  |
| *i=3* | 13 | 12 | 2 | 3 |
| *i=4* | 7 |  |  |  |
| *i=5* | 4 |  |  |  |
| *i=6* | 1 |  |  |  |

Вершини 5 і 6 були внесені в упорядкування раніше за вершини 12 і 13, хоча вони були вільними. В першому випадку довжина впорядкування , а в другому - .

Як бачимо, можливі порушення рівневого принципу, тобто впорядкування, побудовані згідно з рівневим принципом можуть бути не оптимальними. Як бачимо з прикладів, таке порушення можливе в двох випадках:

1. Різний вибір вершин одного рівня приводить до різної довжини впорядкування.
2. Якщо на певному кроці ставити в упорядкування вершину, що має не максимальний з можливих рівень, довжина впорядкування стає меншою, ніж впорядкування, побудоване згідно з рівневим принципом.

Тобто, існують такі графи і обмеження , що рівневий принцип не завжди будує оптимальні впорядкування.

Спробуємо з’ясувати, коли можливе порушення рівневого принципу.

Позначимо множину дуг, що не мають вхідних дуг на -му кроці побудови впорядкування і при цьому мають максимальну помітку.

Сформулюємо необхідну умову порушення рівневого принципу вибору вершин:

**Теорема 2.1:**

Порушення рівневого принципу вибору вершин графу можливе лише у випадку, коли існує таке , що виконується наступна нерівність:

.

Доведення:

Доведемо від супротивного. Розглянемо ліву частину нерівності.

Нехай не існує такого, що . Тобто для всіх . Це означає, що на кожному кроці побудови ми можемо ставити в упорядкування всі вершини з . Вибір вершин не має значення, оскільки ми поставимо їх всі на поточне місце в упорядкуванні.

Розгладнемо праву частину нерівності.

Нехай не існує такого, що . Тобто для всіх

. Це значить, що на j-те місце ми можемо поставити вершин з . Вершини, що залишаться після видалення вказаних, утворюють множину . Отже, виконується нерівність

А для цього випадку ми вже довели, що рівневий принцип не порушується.

Отже, отримали протиріччя.

Але з рішення конкретних задач відомо, що рівневий принцип може порушуватися. А отже, це можливо лише тоді, коли існує таке , що виконується наступна нерівність:

, що і треба було довести.

**Твердження:**

Для того, щоб для графу і обмежень впорядкування, побудовані за рівневим принципом, були оптимальними, достатньо, аби виконувалася наступна нерівність:

де - мінімальна кількість , сума яких, рахуючи з початку підряд, не менша .

Довжина отриманого впорядкування

,

при цьому ця оцінка досяжна.

Розглянемо приклади графів, для яких виконується ця умова.

Побудуємо впорядкування для графу , зображеному на рис. 2.4. Задамо вектор .

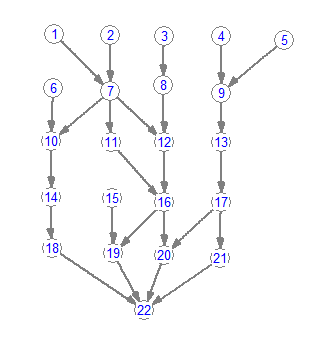


Рис. 2.4. Вигляд графу

Побудуємо для впорядкування , що зображене в табл. 2.10.

*Таблиця 2.10.*

Впорядкування вершин графу

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *k=1* | 1 | 2 | 3 | 4 | 5 |
| *k=2* | 6 | 7 | 8 | 9 |  |
| *k=3* | 10 | 11 | 12 | 13 |  |
| *k=4* | 14 | 15 | 16 | 17 |  |
| *k=5* | 18 | 19 | 20 | 21 |  |
| *k=6* | 22 |  |  |  |  |

Перевіримо виконання умови (1).

.

, отже, умова виконується.

Побудуємо впорядкування, керуючись алгоритмом, основаним на рівневому принципі (табл. 2.11).

*Таблиця 2.11.*

Впорядкування вершин графу , побудоване згідно з рівневим принципом

|  |  |  |  |
| --- | --- | --- | --- |
| *i=1* | 4 | 5 | 1 |
| *i=2* | 2 | 3 |  |
| *i=3* | 8 |  |  |
| *i=4* | 6 | 7 | 9 |
| *i=5* | 12 | 13 | 11 |
| *i=6* | 10 | 17 |  |
| *i=7* | 15 |  |  |
| *i=8* | 16 | 14 |  |
| *i=9* | 18 | 19 | 21 |
| *i=10* | 20 |  |  |
| *i=11* | 22 |  |  |

Впорядкування є оптимальним.

Покажемо справедливість оцінки (2).

.

Отже, оцінка справджується і при цьому є досяжною.

Аналізуючи структуру таких графів, можна відмітити, що вона може виконуватися для досить вузького класу графів і обмежень на ширину впорядкування. Але якщо вдалося встановити, що для даної задачі справедлива нерівність (1), то алгоритм, оснований на рівневому принципі, є точним для цієї задачі.

* 1. **Узагальнені задачі для часткового випадку**

Розглянемо розв’язок узагальненої задачі паралельного упорядкування у випадку .

У випадку довільного дерева задача не може бути розв’язана точно алгоритмом, що базується на рівневому принципі.

Дослідимо цей принцип, коли дорівнює 1 чи 2.

Згідно з теоремою 2.1, у цьому разі рівневий принцип може бути порушений.

Розглянемо відомий алгоритм розв’язку задачі , що базується на лексикографічному порядку нумерації вершин. Покажемо, що він буде оптимальним для довільного орграфу без транзитивних дуг та випадку коли дорівнює 1 чи 2.

Нехай в графі вершини занумеровані згідно з лексикографічним порядком, та .

**Лема 2**.**1**.

У будь-якому підграфі графа , що отриманий внаслідок вилучення з початкового графа вершин з максимальними номерами, зберігається початкова лексикографічна нумерація.

Доведення:

Зафіксуємо в графі довільну множину . З множини вилучимо вершин з максимальними номерами (<). Лексикографічна нумерація є окремим випадком рівне його принципу, отже, вершини рівнів, номери яких менші за та вершини з , утворюють множину з вершин, які мають максимальні номери у лексикографічній нумерації вершин графа . Оскільки номера вилучених вершин не входять в лексикографічні списки решти вершин множини , то їх нумерація буде формуватися згідно з нумерацією вершин множин , а отже, залишиться незмінною.

Лема 2.1 доведена.

На базі цієї леми можливо побудувати та обґрунтувати алгоритм точного розв’язку задачі у випадку, коли може набувати значення 1 або 2.

**Теорема 2.2.**

Алгоритм розв’язку задачі , що базується на лексикографічному порядку нумерації вершин, точно розв’язує задачу задача у випадку коли може набувати значення 1 або 2.

Доведення:

Покажемо, що на ше місце в упорядкуванні, яке будується за лексикографічною нумерацією, можна поставити максимально можливе число вершин з . У випадку, коли , на -ше місце завжди можливо вибрати одну вершину з графа в незалежно від попереднього значення . Тоді при доведенні розглянемо два випадки: на -ше місце потрібно поставити дві вершини після того, як на -те місце вибрана одна вершина, незалежно від значення , та на -те місце треба поставити дві вершини після того, як на - те місце також вибрані дві вершини.

Розглянемо набір обмежень . Позначимо в ньому мінімальні номери та , для них , або або .

Нехай місця з першого по -те включно заповнені згідно лексикографічної нумерації, тобто заповнені оптимально. Видалимо з графа вершини, які занесені в упорядкування. Отримаємо граф . Для цього графа зберігається початкова лексикографічна нумерація (згідно з лемою 2.1). Тобто на перше місце в упорядкуванні вершин графа можна поставити вершини з максимальними номерами і такий вибір буде оптимальним. Це місце відповідає -му в упорядкуванні, що будується для графа .Видалимо з графа вершини, які стоять а -му місці в упорядкуванні, отримаємо новий граф, який приймемо за початковий. Змінимо набір , де - число місць в побудованому упорядкуванні вершин графа . Повторимо описані дії для графа до тих пір, поки . За кінцеве число ітерацій визначимо місце кожної вершини в оптимальному упорядкуванні вершин вихідного графа.

Теорема 2.2 доведена.

* 1. **Сітковий підхід до розв’язання задачі паралельного впорядкування**

Метод, що пропонується, оснований на алгоритмі відшукання максимального умовного потоку у мережі спеціального виду.

Побудуємо – мережу спеціального вигляду, використовуючи довільний дводольний граф D={}, де . Для кожної вершини побудуємо дуги , а для кожної вершини побудуємо дуги . Вершини i відповідно джерело і стік, причому . Отриманий гарф і назвемо – мережею спеціального вигляду.

Нехай на множині дуг – мережі задані цілі додатні числа , котрі будемо називати пропускними спроможностями дуг. На множині вершин цієї мережі введено деяке ациклічне бінарне відношення , тобто відношення, орграф котрого не містить контурів і петель. Фактично цим орграфом буде граф задачі, що розглядається, так що , .

Визначення. Невід’ємну цілочисельну функцію , визначену на дугах побудованої – мережі назвемо умовним потоком, якщо виконуються умови:

де – множина вершин, що виходять з вершини ,

– множина вершин, що входять у вершину

*.* (3)

Тоді задача про максимальний потік – це задача знаходження такого умовного потоку , при якому буде максимальною величина

Очевидно, що не будь-який потік в – мережі буде умовним потоком, саме тому не можна для знаходження максимального умовного потоку використовувати алгоритм Форда-Фалкерсона [6].

Для того, щоби побудувати потрібно виконати наступні дії:

1. Побудувати спеціальну – мережу.
2. Знайти максимальний умовний потік.
3. Якщо максимальний потік такий, що , то перейти на пункт 5.
4. Змінити певним чином – мережу. Перейти на пункт 2.
5. Задача вирішена. Записати отримане впорядкування.

Розглянемо реалізацію кожного з кроків:

1. Побудова спеціальної – мережі.

Запишемо паралельні впорядкування довжини . Оцінемо знизу пераметр , отриману оцінку позначимо . Тоді допустиме місце кожної вершини в шуканому впорядкуванні буде задовольняти нерівності

В якості візьмемо множину вершин орграфу , а в якості – множину номерів . є лівою долею дводольного графу, - правою.

Нехай в вершини розташовані згори донизу в порядку, в якому вони розташовані в упорядкуванні , а в - згори донизу в порядку .

Введемо джерело і стік . Кожну вершину з’єднаємо з вершиною тільки в тому випадку, якщо . Це значить, що вершина може стояти в шуканому впорядкуванні на -му місці.

Пропускні можливості дуг та задаємо рівними одиниці, а дуг – рівними .

1. Знаходження максимального умовного потоку

Алгоритм знаходження максимального умовного потоку працює так, що ми щоразу намагаємося пропустити потік з в через вершину , що має найменший номер . Якщо це не вдається, то пропускаємо через вершину з номером .

В алгоритмі будемо використовувати такі позначення:

– кількість вершин таких, що

– кількість вершин таких, що ;

ПМЗ – постійний масив заборон;

ТМЗ – поточний масив заборон.

Масиви ПМЗ і ТМЗ вводяться для вершин . До них заносяться вершини з множини . Якщо в ПМЗ або ТМЗ вершини розміщена вершина з наступним списком вершин множини , то це значить, що умовний потік, що проходить через вершини, що знаходяться у списку, неможна пропустити через вершину з номером .

Сформулюємо правило, згідно з яким формується ПМЗ: якщо вершину ми поставили на –те місце, то в ПМЗ вершини заносимо вершину , а за нею розміщуємо такі вершини , котрі є досяжними з вершини в графі і для яких .

Алгоритм знаходження максимального умовного потоку

1. Потік на всіх дугах дорівнює нулю.
2. Працюємо з вершиною , .
3. Якщо , то переходимо на крок 6.
4. .
5. Якщо , то переходимо на крок 6.
6. Якщо , то .
7. Якщо в множині немає вершин, що містять від’ємну мітку, то переходимо на крок 11.
8. Нехай - вершина, що має від’ємну мітку , а також .
9. Якщо , то видаляємо вершину з ПМЗ вершини .
10. . Прибираємо мітку вершини.
11. Якщо , то заносимо вершину в ПМЗ вершини .
12. Якщо або , то переходимо на крок 13.
13. Заносимо вершину в ПМЗ вершини .
14. .
15. .
16. Якщо ,то переходимо на крок 10.
17. Перехід на крок 1.
18. Якщо , то переходимо на крок 36.
19. Якщо в ПМЗ вершини немає заборони на вершину , то переходимо на крок 3.
20. .
21. , то переходимо на крок 36.
22. Якщо є вершина , що забороняє вершину в ПМЗ вершини , то переходимо на крок 23.
23. Перехід на крок 36.
24. Обнуляємо попередній ТМЗ вершини .
25. Будуємо новий ТМЗ вершини . Для цього заносимо вершину разом з вершинами, котрі пов’язані з нею шляхом в орграфі , і для котрих в – мережі є дуги з кінцем у вершині .
26. Помічаємо вершину міткою .
27. .
28. Якщо немає заборони на вершину в ПМЗ і ТМЗ вершини , то йдемо на крок 30.
29. Видаляємо мітку вершини .
30. Якщо існує таке, що виконуються умови і вершина не міститься у ТМЗ вершини , то переходимо на крок 21.
31. Якщо немає вершини , такої, що і не міститься в ТМЗ і ПМЗ вершини , то видаємо помітки з всіх вершин. Переходимо на крок 36.
32. Якщо таких вершин декілька, то в якості обираємо вершину з найбільшим значенням .
33. .
34. Якщо , то видаляємо вершину з ПМЗ вершини .
35. . Якщо , то заносимо вершину в ПМЗ вершини

.

1. Переходимо на крок 3.
2. .
3. Якщо , то вершині не знайшлося місця в упорядкуванні. Переходимо на крок 14.
4. Якщо , то .
5. Переходимо на крок 17.
6. Кінець. Записуємо шукане впорядкування.

Очевидно, що в кожну вершину може втікати поток, рівний одиниці. Такий же потік може витікати з цієї вершини. А так як ми розглядаємо лише цілочисельні потоки, то цей потік з вершини може потрапити лише в одну вершину , тобто в шуканому впорядкуванні вершина може бути розташована лише на одному місці.

Так як пропускна спроможність дуги дорівнює то з вершини не може витікати потік величини більше, ніж а це рівнозначно тому, що в шуканому впорядкуванні на місці буде стояти не більше, ніж вершин з .

В процесі роботи алгоритму нам або вдається на кожному кроці збільшити потік на одиницю, або прийдемо до випадку, коли стік помітити не вдається. Тоді отриманий потік максимальний.

Так як вершини в розташовуються згідно з їх порядком в , то в першу чергу будуть розподілятися ті вершини, що стоять на критичному шляху в графі .

Варто зазначити, що масиви ПМЗ і ТМЗ дозволяють слідкувати за тим, аби не порушувалась умова (3).

4. Порядок зміни – мережі

Так як побудований максимальний потік і при цьому існує вершина така, що , то не можна побудувати по заданим паралельне впорядкування довжини . Спробуємо побудувати впорядкування довжини .

Для цього в множину додамо вершину з номером . З’єднаємо цю вершину з вершиною дугою з пропускною спроможністю, рівною . Для кожної вершини введемо по одній дузі наступним чином: якщо вершина з’єднана з вершинами , то додаємо дугу , пропускна спроможність котрої дорівнює одиниці.

5. Процедура запису отриманого впорядкування

Побудований максимальний умовний потік такий, що для всіх . Потік через кожну вершину дорівнює одиниці. Вершину, що стоїть на -му місці в , ставимо в шукане впорядкування на місце , тільки якщо .

Розглянемо граф , зображений на рис. 2.5. Задамо вектор .

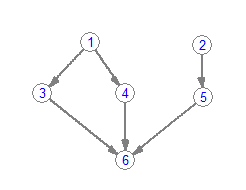


Рис. 2.5. Вигляд графу

Побудуємо для нього спеціальну – мережу. Її вигляд зображено на рис. 2.6.

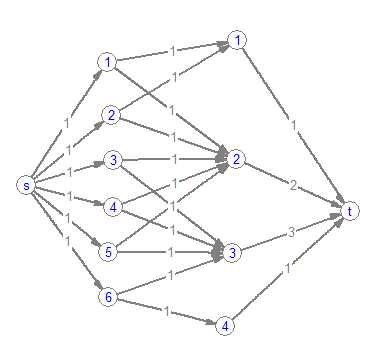


Рис. 2.6. Вигляд мережі для графу

Застосувавши описаний алгоритм, отримаємо наступне впорядкування (табл.2.12). Бачимо, що воно є оптимальним.

*Таблиця 2.12.*

Впорядкування вершин графу

|  |  |  |
| --- | --- | --- |
| *i=1* | 1 |  |
| *i=2* | 2 | 4 |
| *i=3* | 5 | 3 |
| *i=4* | 6 |  |

**Розділ 3. Програмна реалізація алгоритмів**

* 1. **Опис програми**

Для реалізації алгоритму рівневого принципу була написана програма. В якості мови програмування було обрано мову C# завдяки зручності реалізацій в ній операцій зі списками.

Для реалізації алгоритму був створений клас class Graph, який зберігає інформацію про граф обмежень, а також реалізує всі функції, необхідні для роботи з ним та реалізації потрібного алгоритму.

Було реалізовано наступні функції – члени класу:

* public Graph() – конструктор класу;
* List<int> ConnectedNode(int el, List<List<int>> A) – функція, яка приймає в якості параметрів список списків (який, у нашому випадку, представляє собою матрицю суміжності), та цілочисельне значення – номер вершини. Повертає список пов’язаних вершин, які знаходяться за поточною.
* int max(List<int> l) - функція, яка приймає в якості параметрів цілочисельний список та знаходить максимальне за значенням число в ньому.
* public void Draw(Graphics g) - функція, яка приймає в якості параметрів поверхню для малювання та реалізує графічне зображення графу.
* List<int> AND(List<int> small, List<int> big) - функція, яка приймає в якості параметрів два списки та повертає список чисел, які містяться і в першому, і в другому списках одночасно.
* static object DeepClone(object obj) - функція, що дозволяє передавати об’єкти так, наче вони є значимими типами данних.
* List<List<int>> Remove(List<int> index) - функція, яка приймає в якості параметру список індексів, та видаляє з матриці суміжності строки і стовпці, індекси яких містяться в переданому списку.
* public void Build()- функція, яка будує впорядкування згідно з рівневим принципом.
* public void FindMaxMin(int n) - функція, яка приймає в якості параметру ціле число, яке характеризує, скільки разів в циклі будуються впорядкування. В процесі виконання знаходить впорядкування з максимальною та мінімальною довжиною.
* List<int> LookForNotIn(List<List<int>> A, int flag) - функція, що приймає в якості параметрів список списків, та цілочисельне значення flag, та повертає вершини, які, в залежності від значення змінної flag, або не мають вхідних, або вихідних параметрів. Використовується для побудови впорядкування S.
* List<List<int>> FindS(List<List<int>> a, int flag) - функція, яка приймає в якості параметрів список списків (який, у нашому випадку, представляє собою матрицю суміжності), та цілочисельне значення flag. Повертає впорядкування ▁S або ¯S, в залежності від значення flag.
* public bool ReadFromFile() – функція, що реалізує зчитування матриці суміжності з файлу.
* public void ReadFromTable(DataGridView dg\_enter) – функція, яка приймає в якості параметру таблицю DataGridView та зчитує матрицю суміжності з цієї таблиці.
* public void S\_DG\_Max(DataGridView dg\_enter) - функція, яка приймає в якості параметру таблицю DataGridView та виводить на неї впорядкування з максимальною довжиною.
* public void S\_DG\_Min(DataGridView dg\_enter) - функція, яка приймає в якості параметру таблицю DataGridView та виводить на неї впорядкування з мінімальною довжиною.
* public void GetH(DataGridView dg\_enter, bool flag) - функція, яка приймає в якості параметрів таблицю DataGridView та булеве значення flag, в залежності від якого по-різному заповняється список ширин впорядкування.
* public void UpdateDG(DataGridView dg\_enter) - функція, яка приймає в якості параметру таблицю DataGridView та виводить на неї матрицю суміжності графу.

Логіку роботи користувача з програмою реалізовано у класі Form1.

У класі містяться наступні функції:

* private void Form1\_Load(object sender, EventArgs e) - функція, що викликається при завантаженні форми. Встановлює юажані початкові значення змінним.
* private void LoadFromFile\_Click(object sender, EventArgs e) – функція, яка викликається при натисненні кнопки з написом «Зчитати граф з файлу». Вона реалізує відкривання файлу, зчитує з обраного користувачем діапазону матрицю суміжності та передає ці данні класу Graph.
* private void Enter\_Click(object sender, EventArgs e) – функція, яка викликається при натисненні кнопки з написом «Знайти впорядкування». Реалізує виклик відповідних функції з класу Graph. Знаходить впорядкування з мінімальною та максимальною довжиною та виводить їх на екран.
* private void radioButton2\_CheckedChanged(object sender, EventArgs e) – функція, що викликається при зміні виділення елементу radioButton з написом «h = const». Відповідає ситуації, коли будуються впорядкування зі сталою шириною.
* private void radioButton1\_CheckedChanged(object sender, EventArgs e) – функція, що викликається при зміні виділення елементу radioButton з написом «h різні». Відповідає ситуації, коли будуються впорядкування з різною шириною.
* private void ReadFromTable\_Click\_1(object sender, EventArgs e) - функція, яка викликається при натисненні кнопки з написом «Зчитати граф з таблиці». Вона викликає функцію з класу Graph, яка зчитує з таблиці матрицю.
* private void dg\_enter\_RowsAdded(object sender, DataGridViewRowsAddedEventArgs e) – функція, яка викликається при додавання користувачем нової строки в DataGridView, що відповідає матриці суміжності.

Метою написання програми, що реалізує побудову впорядкувань згідно з рівневим принципом було дослідження точності методу для узагальненої задачі впорядкувань. Оскільки рівневий принцип передбачує довільний вибір вершин, то функція побудови конкретного впорядкування Build() запускається n разів в функції FindMaxMin(int n). При достатньо великому значенні n можна розраховувати на те, що протягом обчислень все таки буде знайдене максимальне і мінімальне за довжиною впорядкування, яке можна отримати згідно з рівневим принципом. Оскільки алгоритм, заснований на рівневому принципі відноситься до поліноміально складних алгоритмів, то навіть при великих n розрахунки проводяться дуже швидко.

Для того, аби точно знати оптимальне впорядкування, була модифікована програма, що будує множину всіх допустимих впорядкувань заданої довжини. Програма запускає процедуру побудови множини впорядкувань в циклі з різними , починаючи з , та на кожній ітерації збільшуючи на одиницю, до тих пір, поки отримана множина не міститиме впорядкування, що задовольняє умови обмежень на Варто зазначити, що залежно від графу та побудова бажаного впорядкування займає дуже багато часу. Тож не для кожного з бажаних графів вдавалося знайти таке впорядкування.

## Керівництво користувача

Програма отримує на вхід матрицю суміжності, що відповідає графу обмежень, а також інформацію про обмеження на ширину В результаті користувач отримує впорядкування з мінімальною та максимальною довжинами, а також величину довжин.

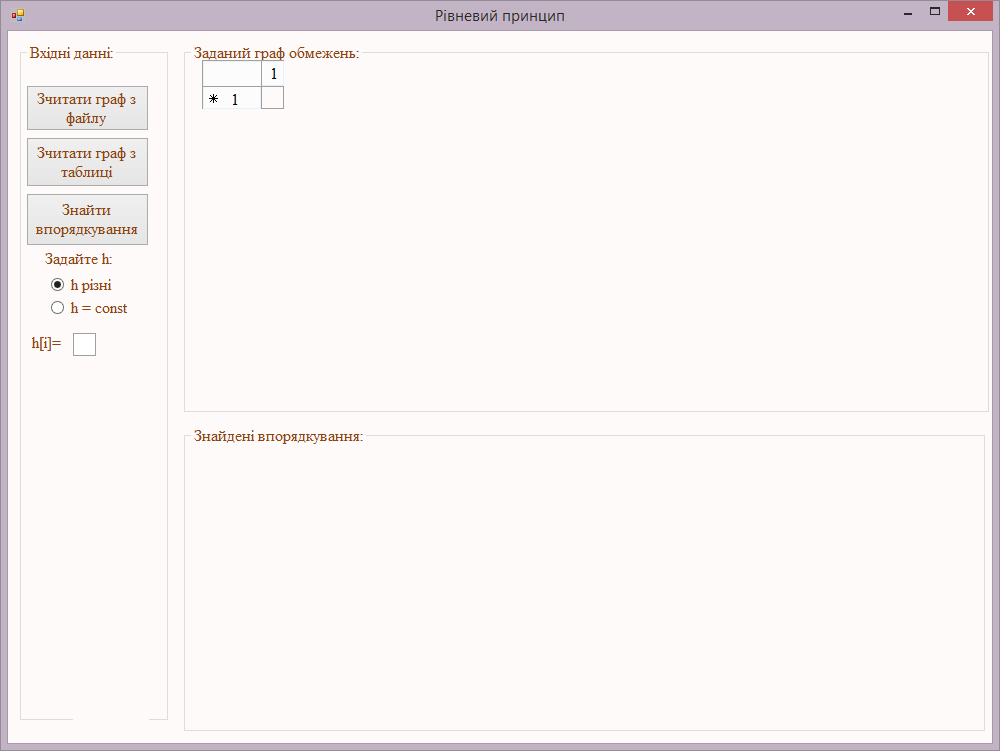
Запустивши програму, користувач побачить наступне зображення (рис. 3.1.):

Рис. 3.1. Вигляд вікна при запуску

Для початку роботи можна або ввести бажаний граф з клавіатури безпосередньо на екрані, або зчитати з файлу Excel.

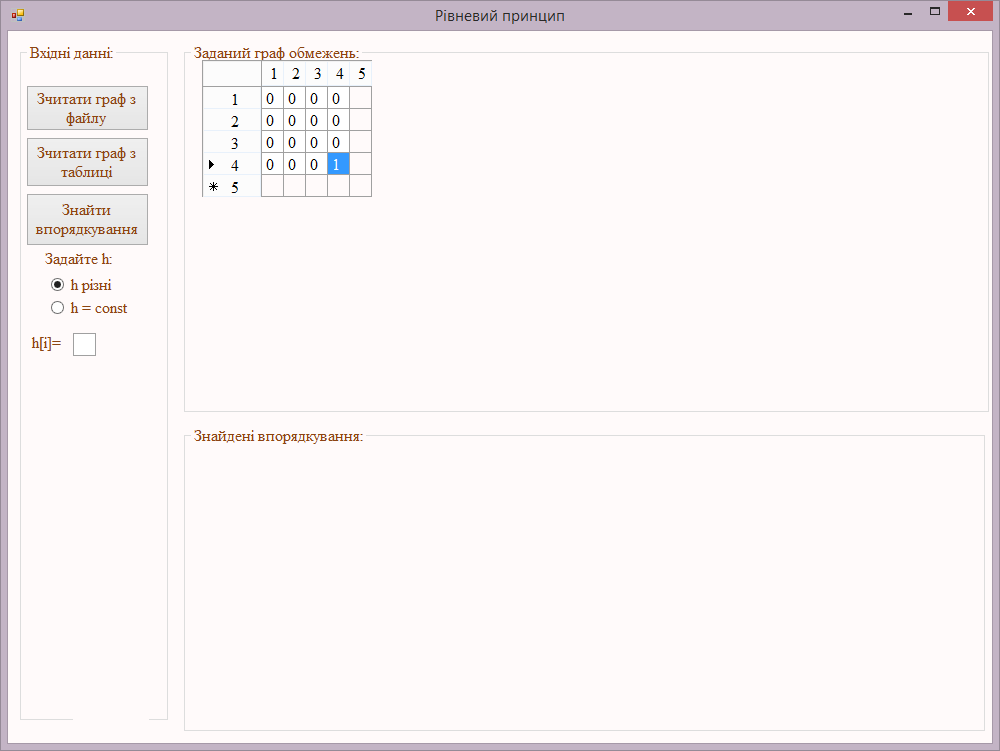
Для реалізації першого способу необхідно почати вводити матрицю суміжності в таблиці під написом «Заданий граф обмежень». По мірі введення даних матриця буде автоматично розширюватися, заповнюючи вільні місця нулями. Процес заповнення матриці бачимо на рис. 3.2. Після цього необхідно натиснути кнопку «Зчитати граф з таблиці».

Рис. 3.2. Введення матриці обмежень

Для реалізації другого способу необхідно натиснути на кнопку «Зчитати граф з файлу». Після цього користувач побачить діалогове вікно з пропозицією обрати файл. Обравши файл, він має обрати діапазон, на якому задана матриця суміжності. Перший рядок та відповідно перший стовпець мають містити номера вершин. Процес обрання діапазону зображений на рис.3.3.

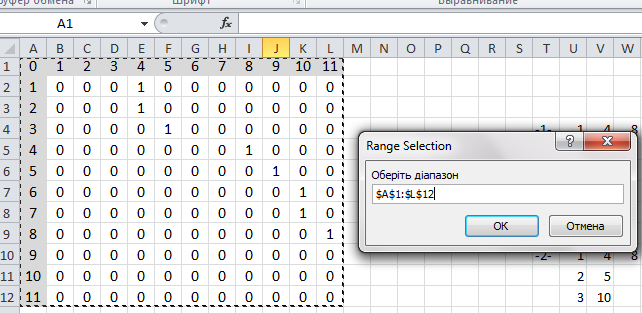


Рис.3.3. Програма запитує діапазон

Для прикладу зчитаємо з файлу деяку матрицю. Реакцію програми бачимо на рис. 3.4.

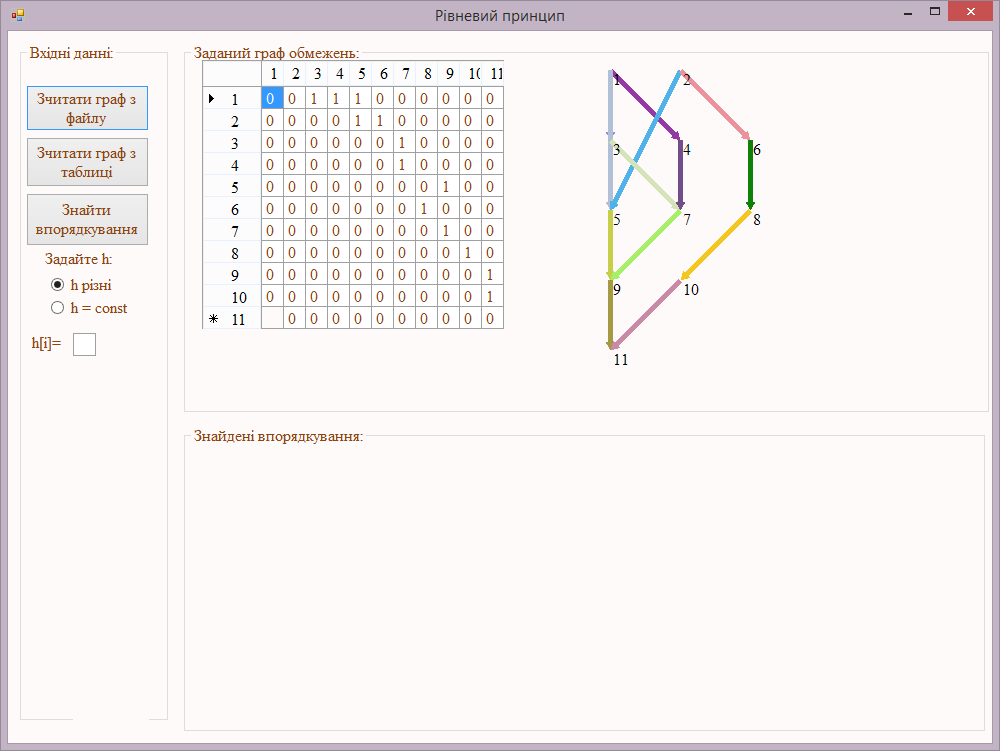


Рис. 3.4. Реакція на зчитування з файлу

Користувач бачить візуальне зображення графу обмежень, що спрощує подальший аналіз результатів.

Користувач може або задати вектор обмежень , або задати . Щоб задати обмеження у вигляду , необхідно обрати пункт « різні» та задати у таблиці бажаний цілочисельний вектор. При бажанні побудувати впорядкування з однаковою шириною, необхідно обрати пункт та ввести єдине число.

Для того, або виконати обчислення і побачити впорядкування, користувач має натиснути на кнопку «Знайти впорядкування». Реакцію програми бачимо на рис. 3.5.

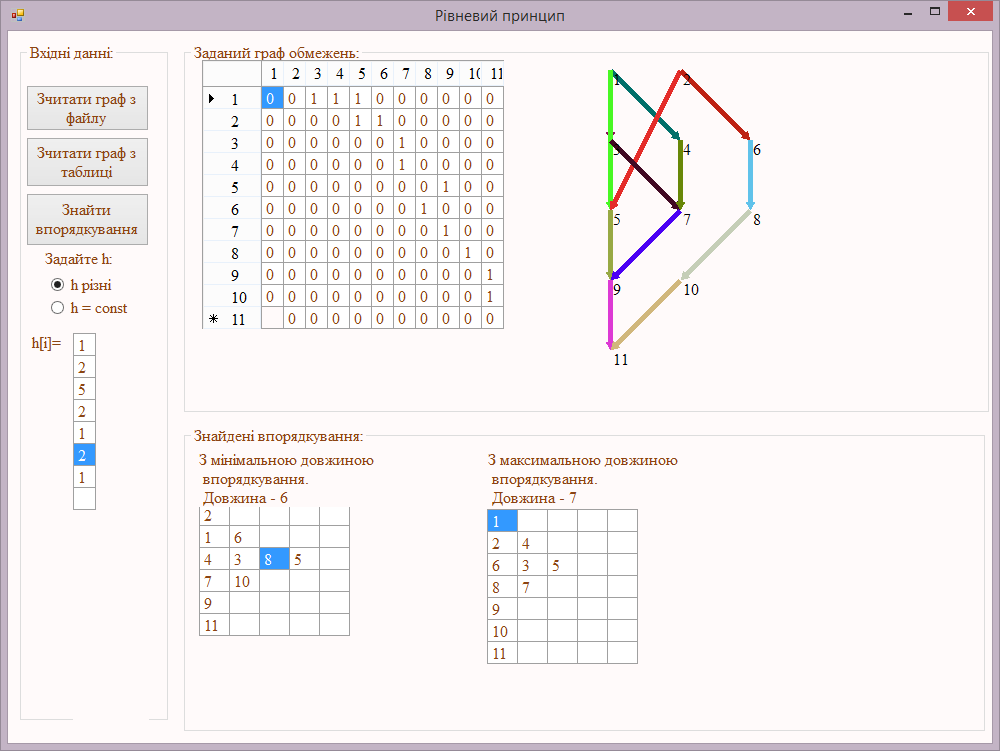


Рис. 3.5. Результат роботи програми

На екран видано впорядкування з мінімальною і максимальною довжиною. Таким чином можемо зробити висновок, що для заданого графу обмежень та даному векторі рівневий принцип порушується.

## Результати тестування програми

Для того аби переконатися в тому, що програма працює коректно, спробуємо виконати деякі приклади.

Приклад 1.

Протестуємо програму на стійкість до помилок. Задамо на вхід граф з петлею. Реакцію програми бачимо на рис. 3.6.

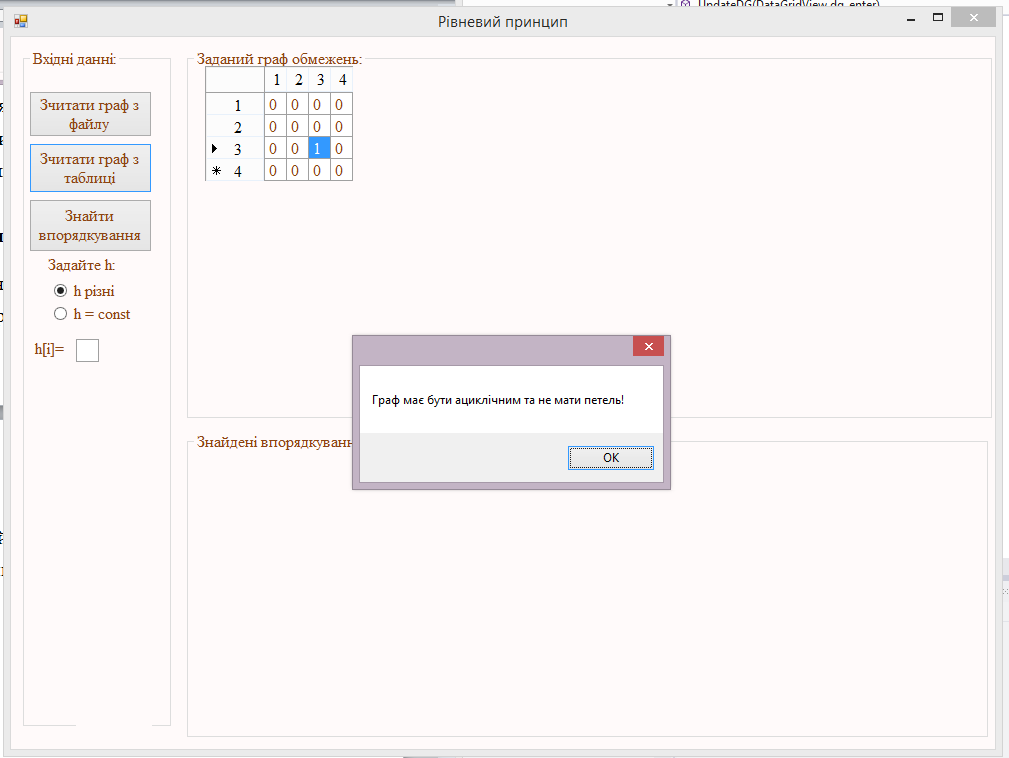


Рис. 3.6. Результат роботи програми

Приклад 2.

Протестуємо програму на стійкість до помилок. Задамо на вхід циклічний граф. Реакцію програми бачимо на рис. 3.7.

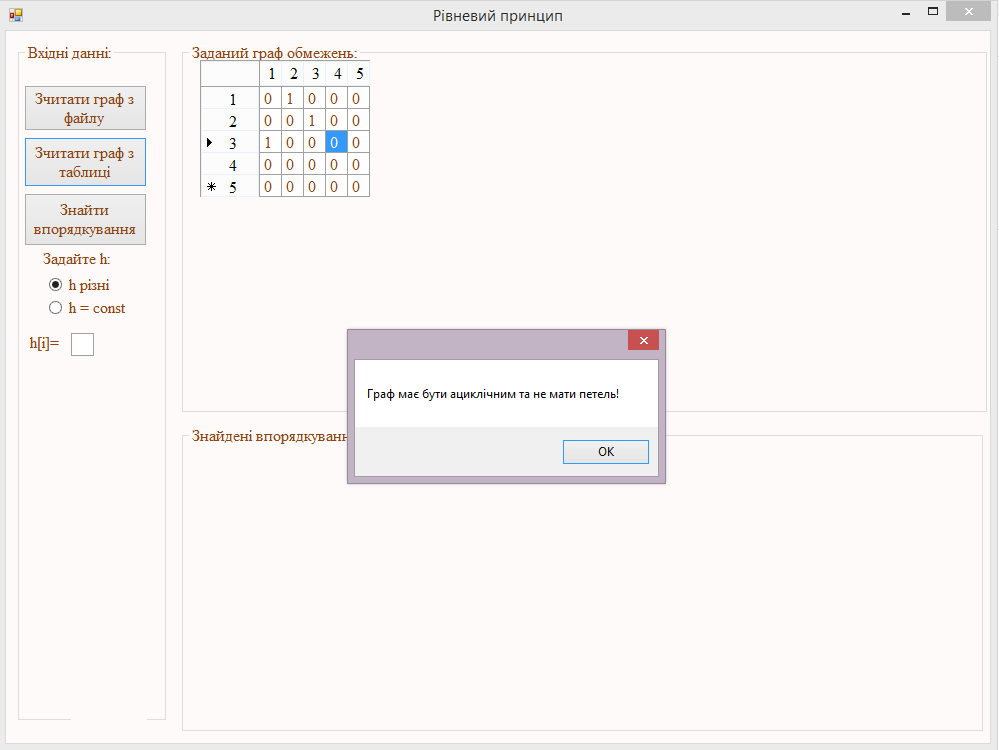


Рис. 3.7. Результат роботи програми

Приклад 3.

Задамо вектор обмежень замалим за довжиною. Отримали результат, що зображений на рис.3.8.

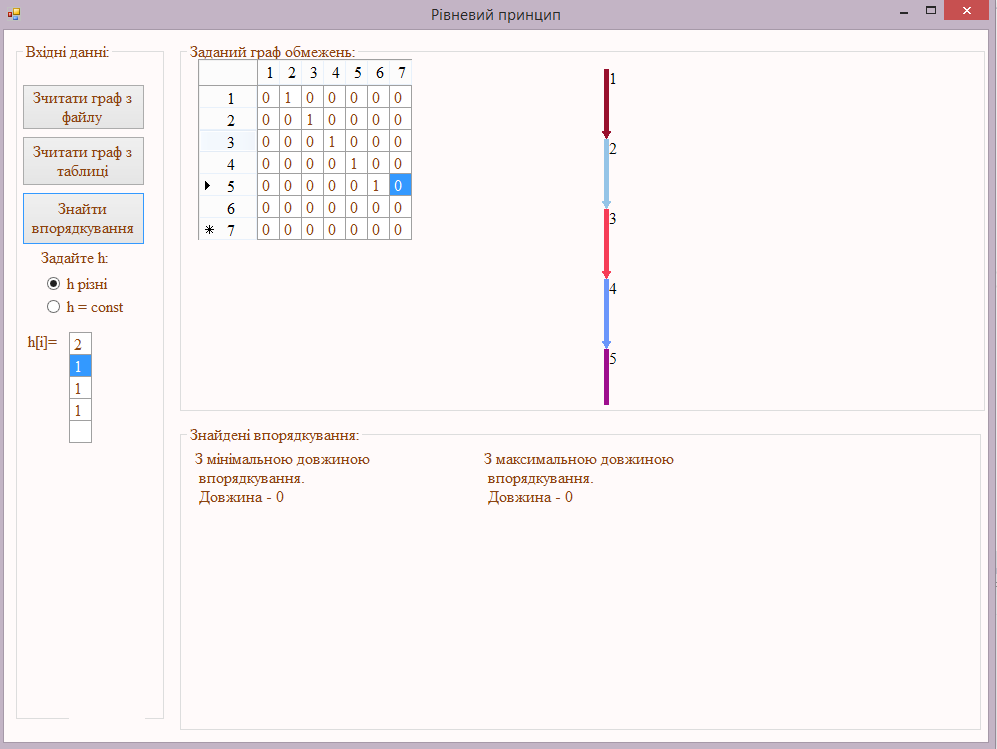


Рис. 3.8. Результат роботи програми

Упорядкування не були побудовані, як і має бути.

Приклад 4.

Зчитаємо з файлу деякий граф, задамо Реакцію програми бачимо на рис. 3.9.

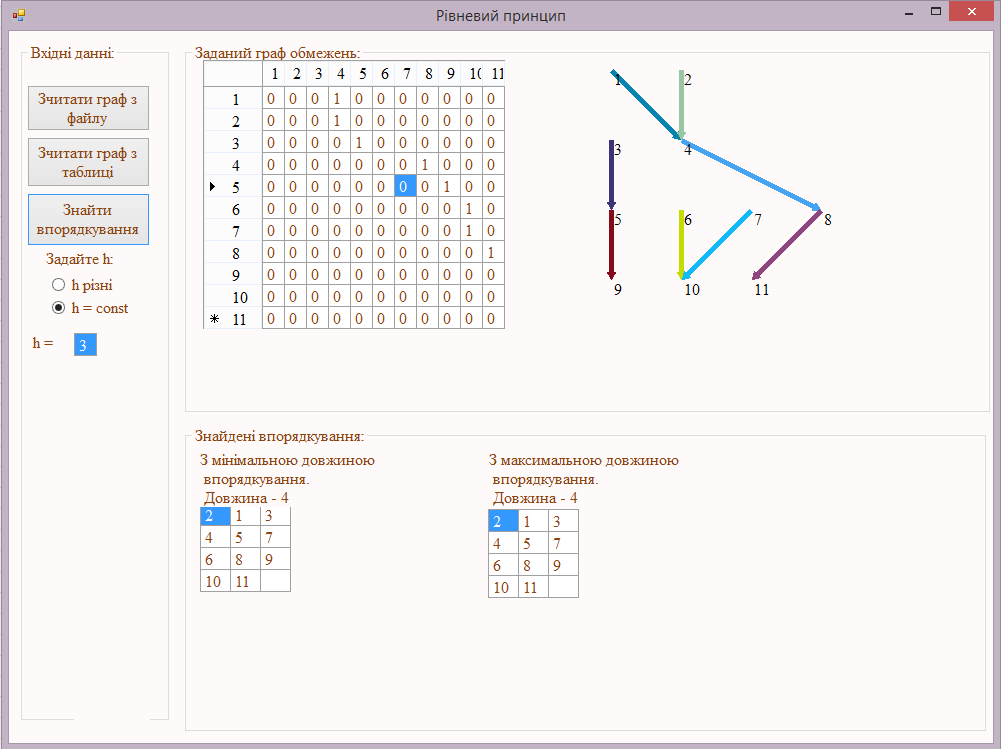


Рис. 3.9. Результат роботи програми

Результати, отримані в програмі, відповідають очікуваним.

Приклад 5.

Зчитаємо з файлу деякий граф, задамо довільний вектор . Результат роботи програми бачимо на рис. 3.10.

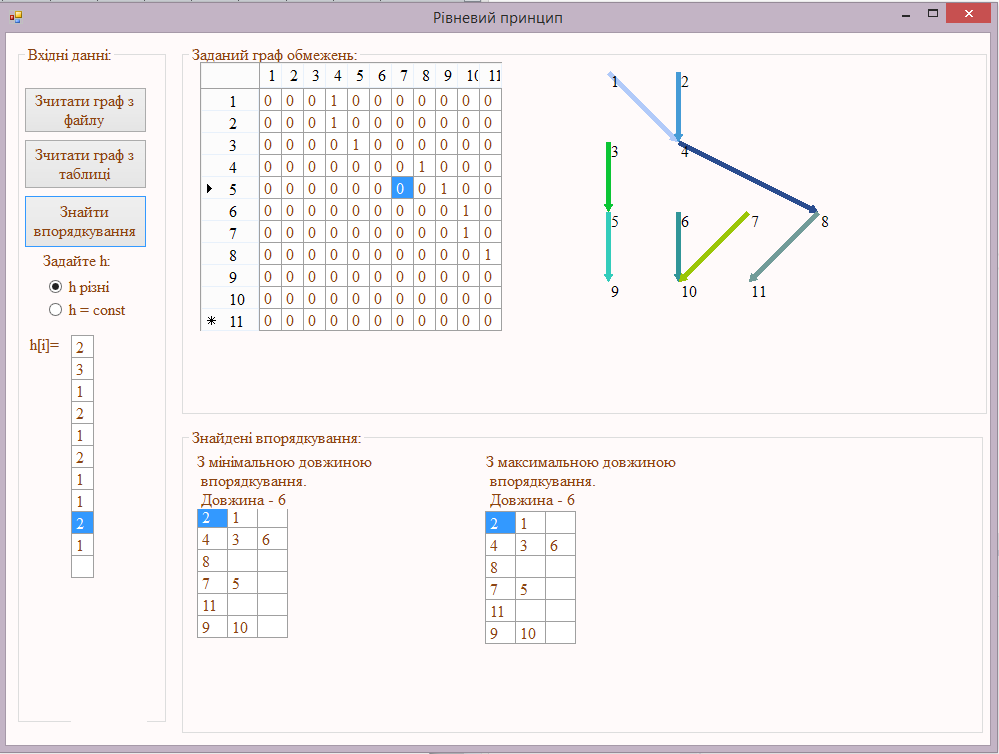


Рис. 3.10. Результат роботи програми

Результати, що ми бачимо, відповідають очікуванням.

Приклад 6.

Зчитаємо з файлу деякий граф, задамо довільний вектор . Результат роботи програми бачимо на рис. 3.11.

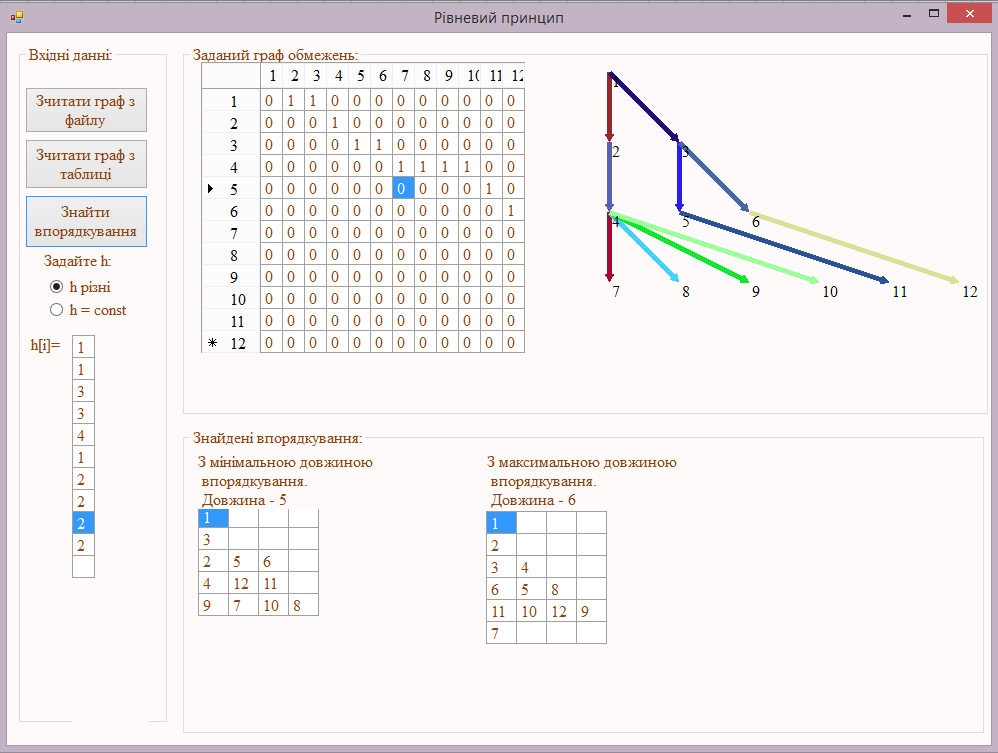


Рис. 3.11. Результат роботи програми

Результати відповідають очікуванням. Як бачимо, вдалося отримати впорядкування різної довжини.

Приклад 7. Зчитаємо з файлу деякий граф, задамо довільний вектор . Результат роботи програми бачимо на рис. 3.12.

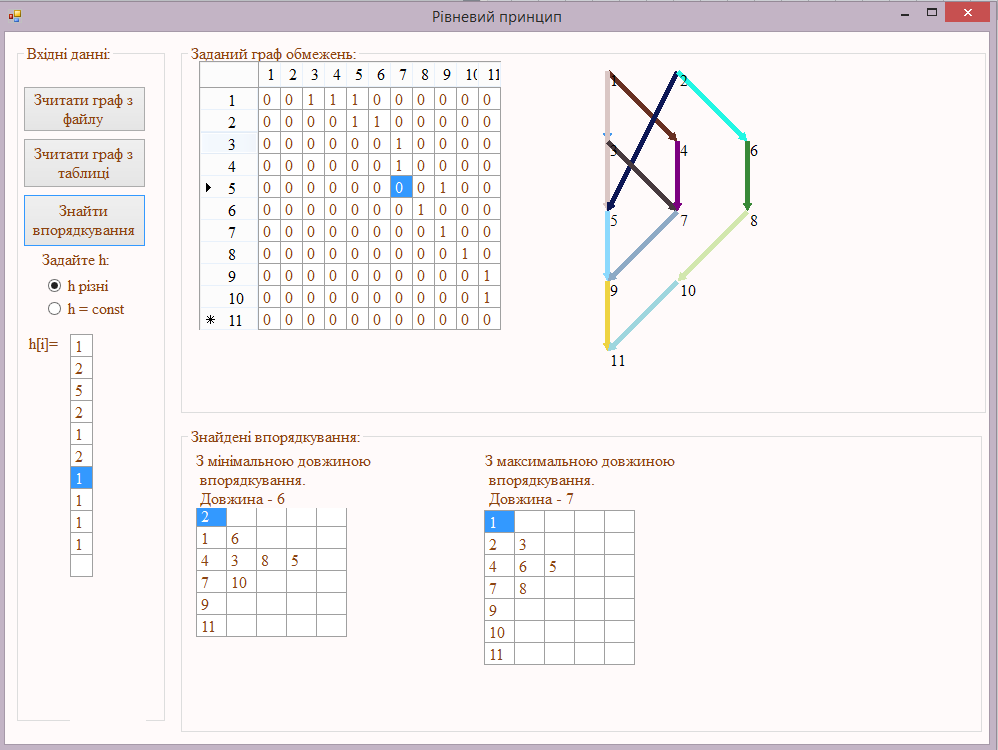


Рис. 3.12. Результат роботи програми

Результати відповідають очікуванням. Як бачимо, вдалося отримати впорядкування різної довжини.

* 1. **Аналіз отриманих в програмі даних**

За мету написання програми ставилося з’ясування точності алгоритму побудови впорядкувань, заснованому на рівневому принципі, для вирішення узагальнених задач теорії впорядкувань.

Для зручності аналізу, всі задачі розділи на 4 класи:

1. – дерево (ліс), .
2. – дерево (ліс), різні.
3. – довільний граф, .
4. – довільний граф, різні.

Розглянемо кожний з класів детальніше.

1. – дерево (ліс), .

Саме для цього випадку і розроблявся описаний Ху точний алгоритм побудови паралельних впорядкувань. Рівневий принцип не порушується. Програмі задавалось достатньо багато різних дерев, але прикладу порушення рівневого принципу не було знайдено, як і має бути.

Розглянемо один з прикладів. Подамо на вхід деяке дерево та задамо

. Результат роботи програми бачимо на рис. 3.13.

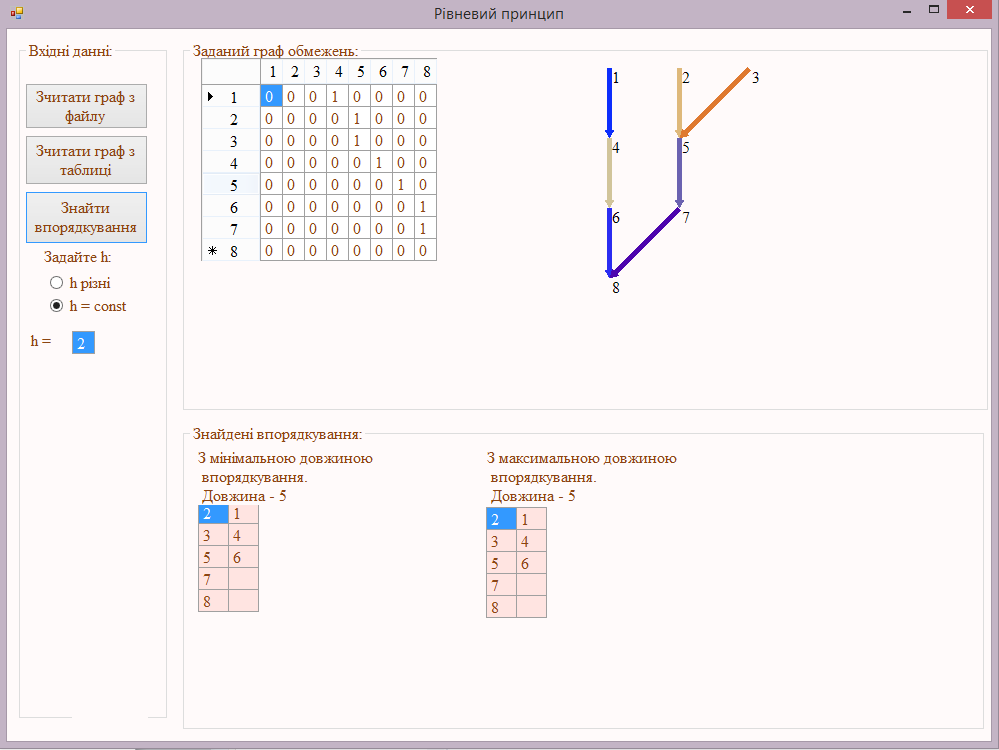


Рис. 3.13. Результат роботи програми

Для того, аби переконатися, що знайдене впорядкування є оптимальним, задамо той самий граф обмежень на вхід модифікованої програми побудови множини всіх допустимих паралельних впорядкувань заданої довжини. Результат роботи програми бачимо на рис. 3.14.

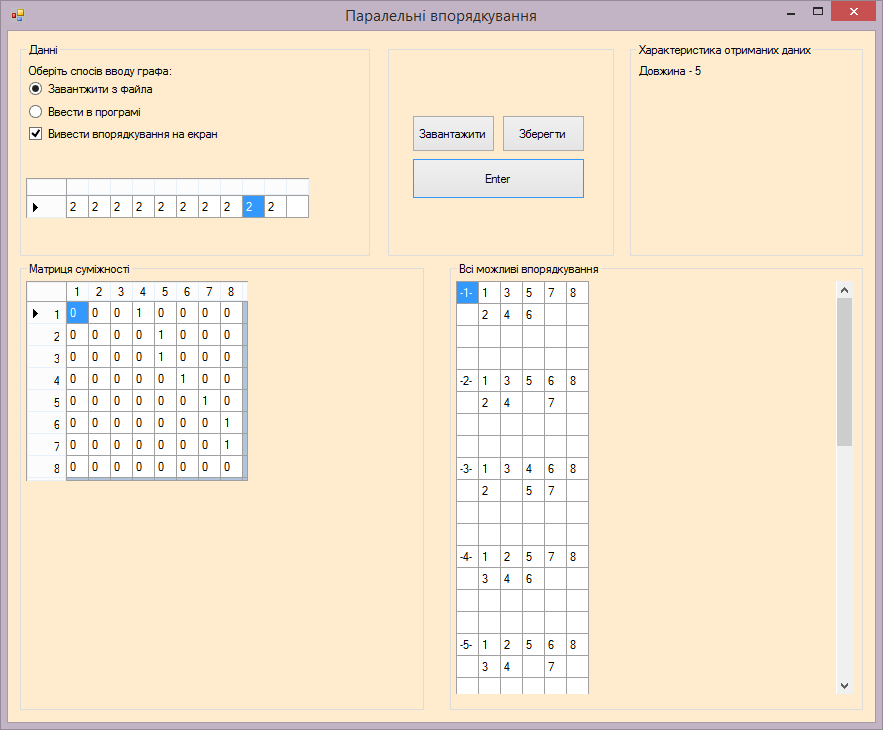


Рис. 3.14. Результат роботи програми

Як видно з рис. 3.13 та рис. 3.14, довжина впорядкувань, отриманих в результаті роботи обох програм однакова. Одним з впорядкувань, отриманих в результаті роботи другої програми, і є отримане першою програмою впорядкування.

1. – дерево (ліс), різні.

З прикладів ми бачили, що для цього випадку алгоритм, оснований на рівневому принципі, може порушуватися.

За допомогою програмного продукту було оброблено багато різних дерев, та знайдено досить багато випадків порушення рівневого принципу. Для всіх випадків порушення рівневого принципу виконувалися умови теореми 2.1. Розглянемо деякі з прикладів.

Задамо деякий граф і деякий вектор. Результати роботи програми бачимо на рис. 3.15.

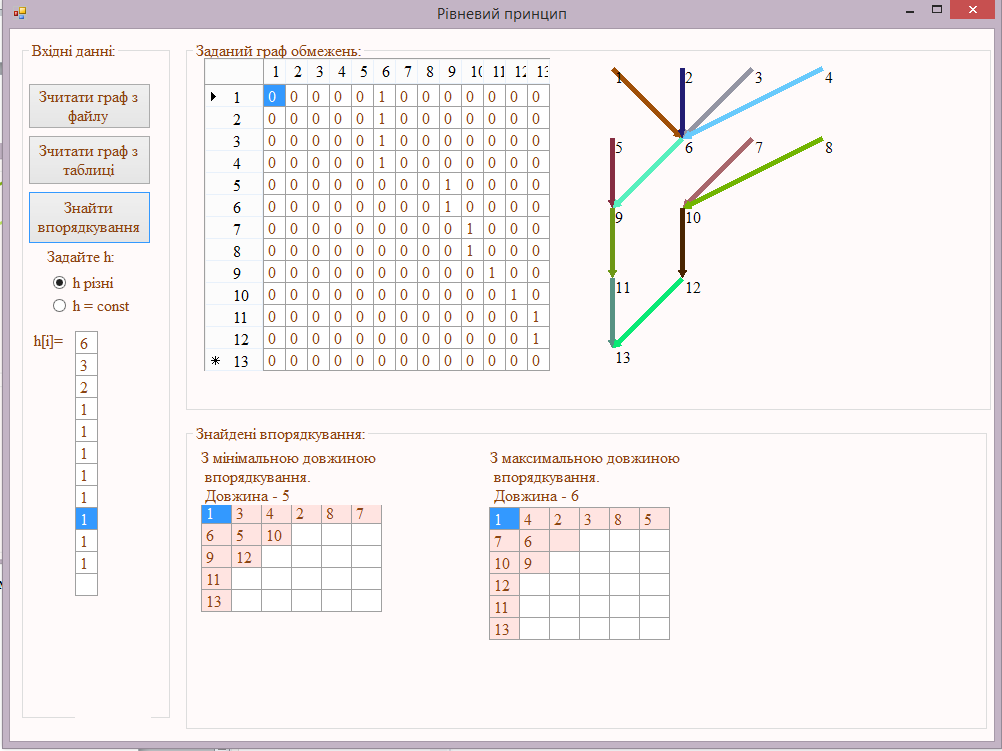


Рис. 3.15. Результат роботи програми

Ті ж самі вхідні дані задамо другій програмі. Отримаємо наступний результат, зображений на рис. 3.16:

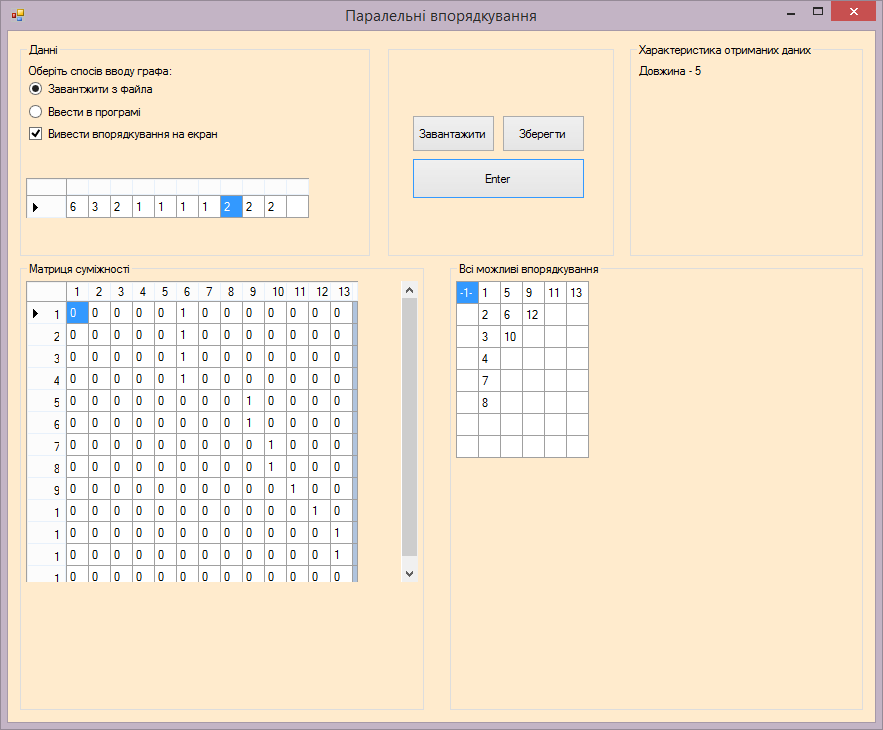


Рис. 3.16. Результат роботи програми

Як бачимо, оптимальним є лише одне впорядкування, яке і знайшла перша програма.

Задамо деякий інший граф і деякий вектор. Результати роботи програми бачимо на рис. 3.17.

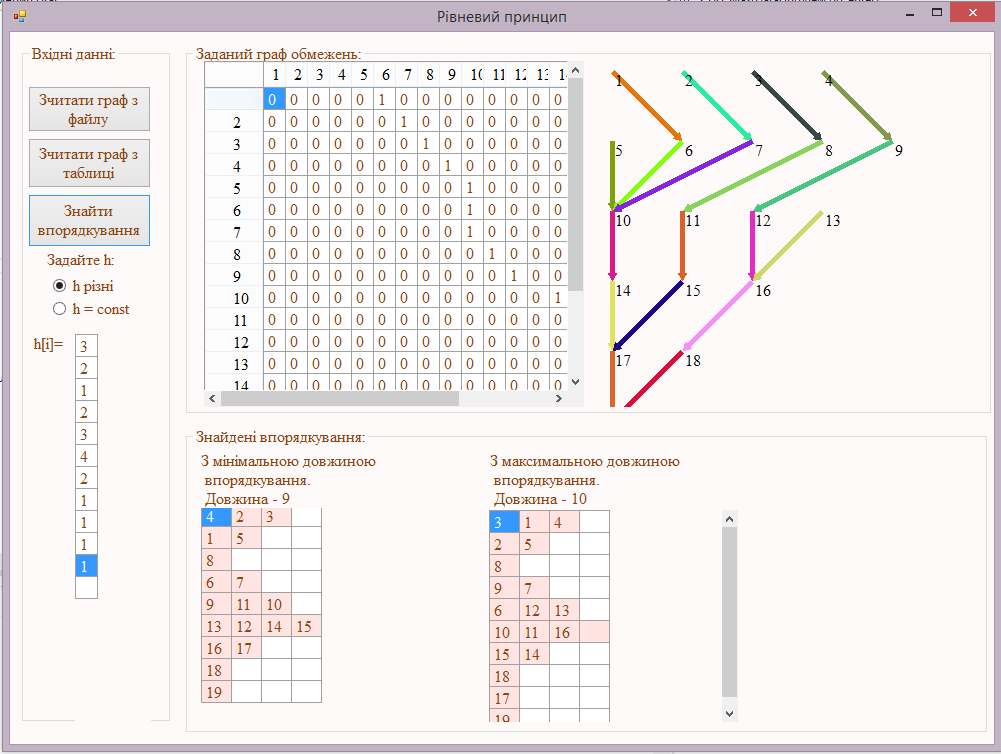


Рис. 3.17. Результат роботи програми

Як бачимо, знайшли впорядкування з рівною довжиною.

Отримати результат за допомогою програми, основаної на направленому переборі вершин, не вдалося через завеликий час виконання.

Задамо ще один граф. Результат роботи бачимо на рис. 3.18.

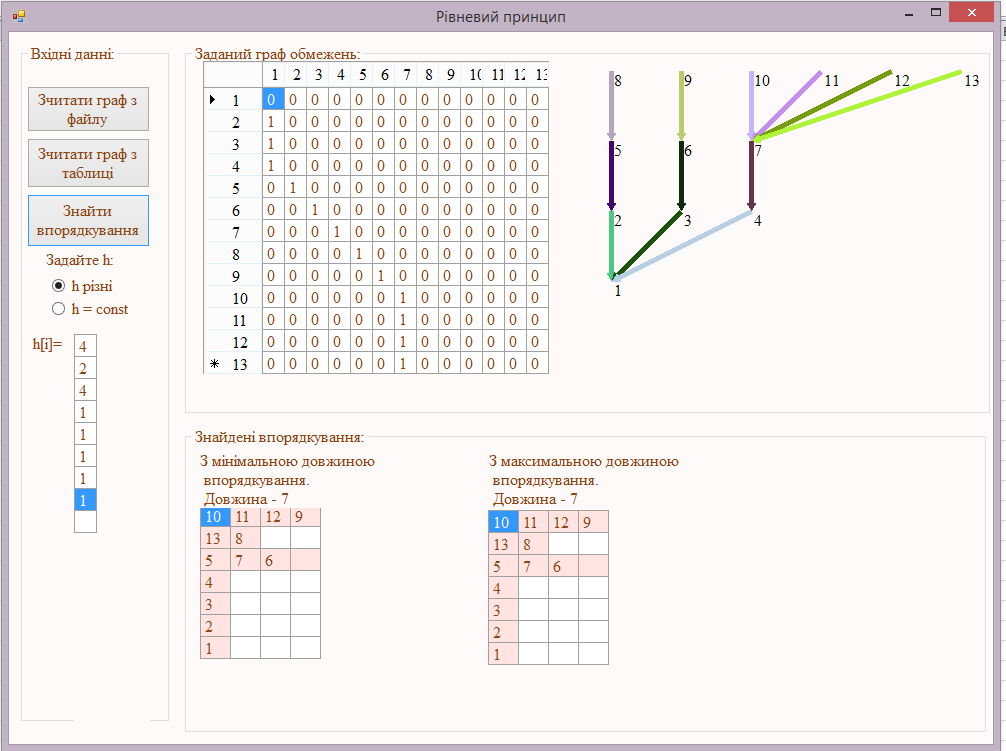
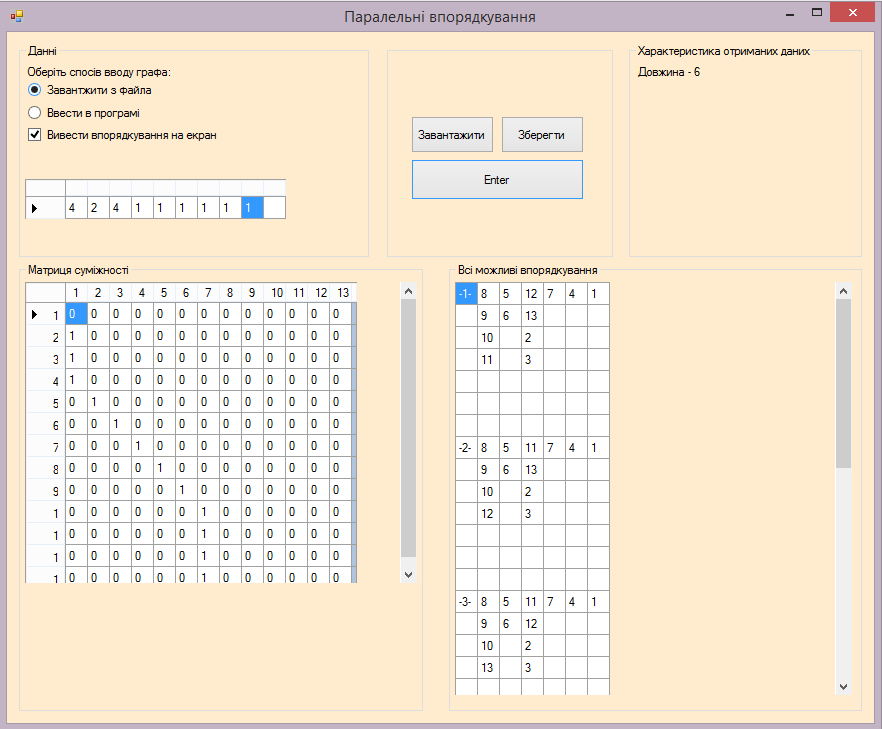


Рис. 3.18. Результат роботи програми

Ті ж самі вхідні дані задамо другій програмі. Отримаємо наступний результат, зображений на рис. 3.19:

 Рис. 3.19. Результат роботи програми

Бачимо, що побудоване впорядкування має довжину 6, тоді як всі впорядкування, побудовані за рівневим принципом, мали довжину 7. Спостерігаємо порушення рівневого принципу, що зумовлене другою причиною, а саме тим, що якщо на певному кроці ставити в упорядкування вершину, що має не максимальний з можливих рівень, довжина впорядкування стає меншою, ніж впорядкування, побудоване згідно з рівневим принципом.

То ж алгоритм для другого класу задач є наближеним.

1. – довільний граф, .

Для цього випадку алгоритм, заснований на рівневому принципі, дає наближені результати. Покажемо це на прикладі (рис.3.20).

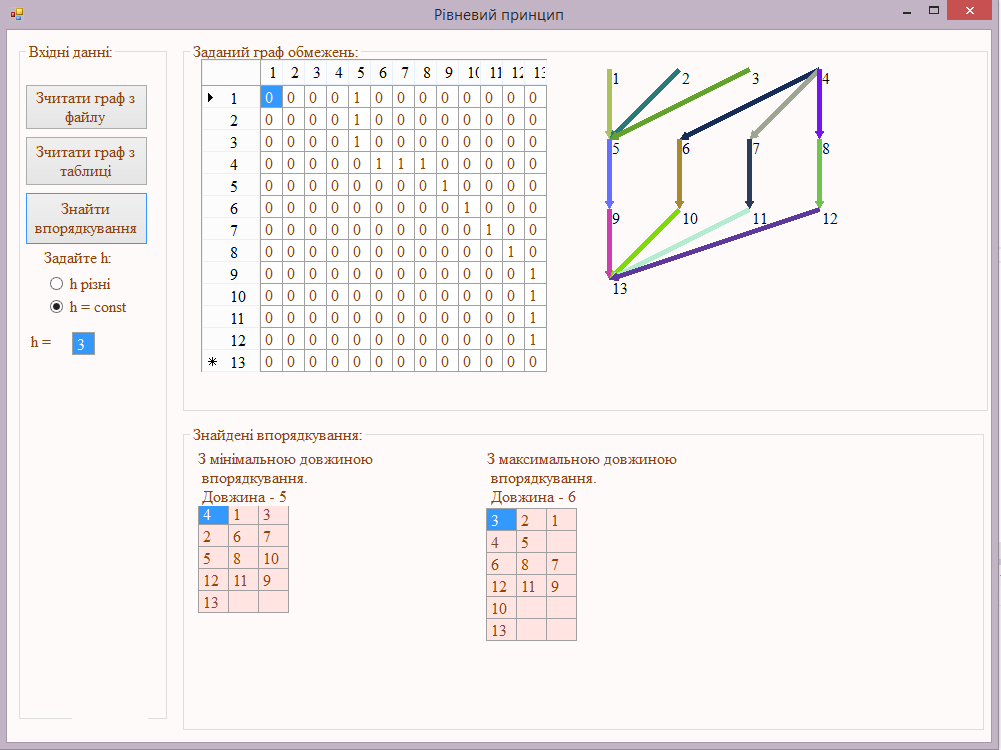


Рис. 3.20. Результат роботи програми

Крім того, відомо, що для цього класу виконується наступна нерівність:

Перевіримо її для отриманих даних:

Тобто, нерівність справджується.

1. – довільний граф, різні.

Алгоритм, заснований на рівневому принципі, для цього класу задач є наближеним. Покажемо це на одному з прикладів, зображеному на рис. 3.21.

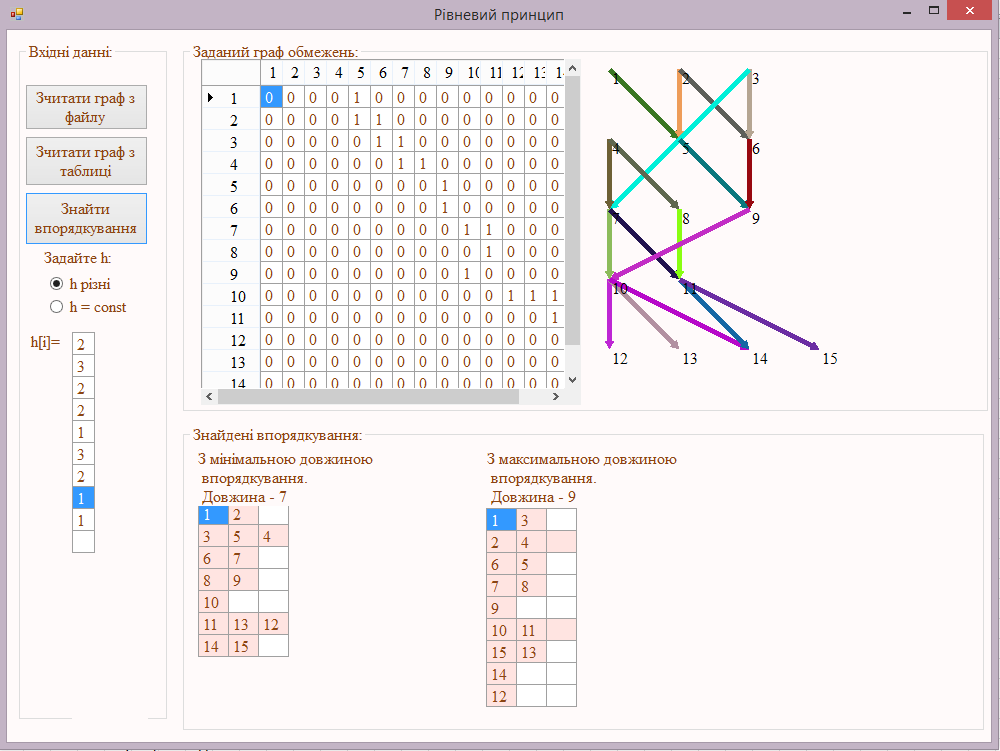


Рис. 3.21. Результат роботи програми

Ті ж самі вхідні дані задамо другій програмі. Отримаємо наступний результат, зображений на рис. 3.22:

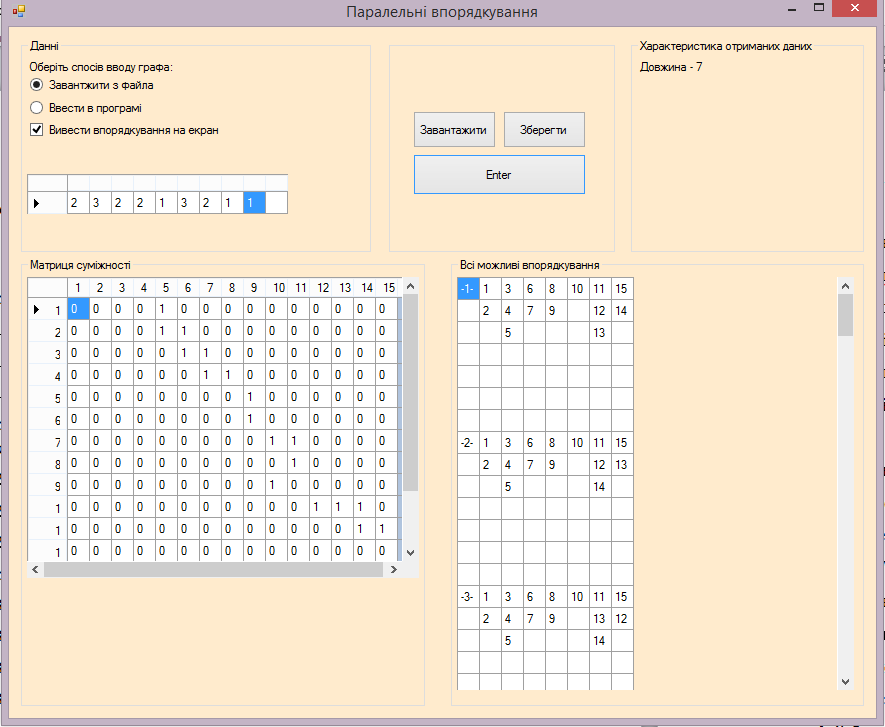


Рис. 3.22. Результат роботи програми

Результати співпадають.

Можна зробити висновки, що для тих класів, для яких алгоритм є наближеним, якщо порушення рівневого принципу зумовлене різним порядком вибору вершин, то побудувати оптимальне впорядкування можна багаторазовим запуском алгоритму рівневого принципу, що не є затратним, зважаючи на швидкість роботи алгоритму. А якщо ж порушення викликано другою причиною, - тим, що якщо на певному кроці ставити в упорядкування вершину, що має не максимальний з можливих рівень, довжина впорядкування стає меншою, ніж впорядкування, побудоване згідно з рівневим принципом, то побудувати оптимальне впорядкування цим алгоритмом не вийде взагалі.

# Висновки

В процесі виконання дипломної роботи було:

* застосовано алгоритм, оснований на рівневому принципі для узагальненої задачі;
* виділені підкласи графів, для яких алгоритм точно розв’язує узагальнену задачу;
* сформульовано та доведено теорему про можливе порушення рівневого принципу вибору вершин, а також твердження про достатні умови того, що впорядкування, побудовані за рівневим принципом, будуть оптимальними.
* розроблено програмний продукт, що реалізує вказаний алгоритм;
* дано рекомендації щодо доцільності застосування алгоритму, основаного на рівневому принципі для вирішення узагальненої задачі впорядкування.

**Розділ 4. Охорона праці**

Робота над дипломною роботою пов’язана з постійною працею за персональним комп’ютером.

На робочому місці користувача ПК можуть виникати наступні небезпечні та шкідливі фактори: підвищений рівень шуму, несприятливі мікрокліматичні умови, недостатній рівень освітленості, шкідливі речовини, підвищений рівень електромагнітних випромінювань радіочастот, висока напруга електричної мережі, статична електрика та інші. Робота з ПК супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи.

Робота оператора ПК згідно з діючим класифікатором професій відноситься до другої професійної групи.

Для збереження здоров’я користувачів ПК, запобігання професійним захворюванням і підтримки працездатності слід дотримуватися вимог   
ДСан ПіН 3.3.2.007-98 щодо режиму праці та відпочинку [8]. Для цього призначаються регламентовані перерви для відпочинку.

Протягом робочого дня мають передбачатися:

* перерви для відпочинку і вживання їжі (обідні перерви);
* перерви для відпочинку і особистих потреб (згідно з трудовими нормами);
* додаткові перерви.

Для зниження нервово-емоційного напруження, стомлення зорового аналізатора, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно деякі перерви використовувати для виконання комплексу вправ, які наведені у Державних санітарних правилах і нормах роботи з візуальними дисплейними терміналами електронно-обчислювальних машин.

## Вимоги до організації робочого місця

Головними елементами робочого місця людини, що працює за комп’ютером, є стіл і крісло. Основним робочим положенням є положення сидячи.

Робоча поза сидячи викликає мінімальне стомлення програміста. Раціональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Істотне значення для продуктивної і якісної роботи на комп'ютері мають розміри знаків, густину їх розміщення, контраст і співвідношення яскравості символів і фону екрану. Якщо відстань від очей оператора до монітору складає 60 - 80 см, то висота знака повинна бути не менше 3 мм, оптимальне співвідношення ширини і висоти знака складає 3:4, а відстань між знаками – 15 - 20% їх висоти. Співвідношення яскравості фону екрану і символів - від 1:2 до 1:15 .

Під час користування комп'ютером медики радять встановлювати монітор на відстані 50-60 см від очей. Фахівці також вважають, що верхня частина монітору повинна бути на рівні очей або трохи нижче.

## Опис робочого місця

Кімната являє собою приміщення загальною площею 11 м2, і висотою стелі 3м. У приміщенні знаходиться 1 робоче місце з персональним комп’ютером. Робоче місце обладнане робочим столом площею 1,5 м2, стільцем та персональним комп'ютером, що складається з монітора, системного блоку, клавіатури та миші. План кімнати, де розташоване робоче місце, зображено на рис. 4.1.

Слід відзначити, що площа одного робочого місця оператора ПК не повинна бути меншою за 6м2, а об'єм не менший за 20м3, тобто площі та об'єму даного приміщення вистачає для розташування цього робочого місця користувача ПК.

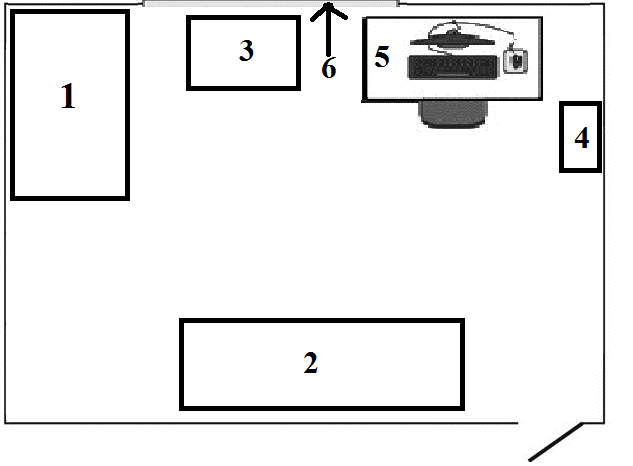


Рис. 4.1. План кімнати

Цифрами на рисунку 4.1. позначені наступні предмети:

1,2 - шафа; 3 - тумба; 4 - вогнегасник; 5 - комп’ютерний стіл; 6 - вікно.

## Освітленість

Переважна частина інформації, яку сприймає людина, надходить через органи зору. Тому важливо слідкувати, аби освітленість на робочому місці була достатньою та правильною. Окрім рівня освітленості, важливо слідкувати за рівномірністю світлового потоку. Для економії ресурсів при використанні джерел штучного світла рекомендується в кімнаті мати стіни світлого кольору [9].

Вимоги щодо освітлення робочого місця оператора ПК наведено у державних санітарних правилах і нормах при роботі з відео терміналами електронно - обчислювальних машин.

У досліджуваному приміщенні в якості джерел штучного світла застосовуються люмінесцентні лампи, які краще поєднуються з природним освітленням, ніж лампи розжарювання. Окрім того, вони створюють більш дифузні світлові потоки, через що знижується можливість засліплюючої дії світла, відбитого екраном [10].

Мінімальна освітленістьвстановлюється в залежності від розряду виконуваних зорових робіт. Для ІV розряду зорових робіт вона складає 300- 500 лк.

В кімнаті, яка розглядається, всі вимоги, що стосуються освітленості, дотримано. Освітленість достатня та рівномірна. Тому під час роботи не створюється додаткового навантаження на органи зору.

## Параметри мікроклімату

Відповідно до ДСН 3-3-6-042-99 «Санітарні норми мікроклімату виробничих приміщень» [робота](http://ua-referat.com/%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0) програміста відноситься до категорії I б, тобто робота, сидяча, стояча або пов'язана з ходьбою і супроводжується деякими фізичними напруженнями [11].

Оптимальні мікрокліматичні умови являють собою поєднання кількісних показників мікроклімату, які при тривалому і систематичному впливі на людину забезпечують збереження нормального теплового стану його організму без напруги механізмів терморегуляції. Вони забезпечують відчуття теплового комфорту та створюють передумови для високого рівня працездатності.

Оптимальні параметри мікроклімату для приміщень з персональними комп’ютерами наведено в табл. 4.1.

*Таблиця 4.1.*

Параметри мікроклімату для приміщень з ПК

|  |  |  |
| --- | --- | --- |
| Період року | Параметр мікроклімату | Величина |
| Холодний | Температура повітря в приміщенні | 22...24°С |
| Відносна вологість | 40... 60% |
| Швидкість руху повітря | до 0,1 м/с |
| Теплий | Температура повітря в приміщенні | 23...25 °С |
| Відносна вологість | 40... 60% |
| Швидкість руху повітря | 0,1...0,2 м/с |

Кімната має такі параметри: опалювання – централізоване; вентиляція природна; кондиціонування лише влітку, провітрювання за необхідності, температура повітря взимку 20-22°С, влітку − 25-28°С; вологість 45-65%.

* 1. **Електробезпека на робочому місці**

Розглянуте приміщення відноситься до класу приміщень без підвищеної небезпеки поразки електричним струмом, тому що в даному приміщенні відсутні ознаки підвищеної чи особливої небезпеки (вологості, провідникового пилу, струмоведучих основ (металевих, земляних), підвищеної температури і т.д [12].

Застосовувана техніка відноситься до електроустановок напругою до 1000 В, підключення яких здійснюється від мережі однофазного змінного струму напругою 220 В, частотою 50 Гц, струм спрацьовування 25 А.

Небезпека поразки струмом при замиканні на корпус усунута за допомогою установки в приміщенні нульового захисного провідника. Призначення нульового захисного провідника - створення для струму короткого замикання ланцюга з малим опором, щоб цей струм був достатнім для швидкого відключення ушкодженої установки від мережі.

Для забезпечення безпеки ЕОМ периферійні пристосування та устаткування для обслуговування, ремонту і налагодження, електропроводи і кабелі за виконанням і ступенем захисту мають апаратуру захисту від струму короткого замикання та інших аварійних режимів.

Будинки і ті їхні частини, в яких розміщуються ЕОМ, мають не нижче II ступеня вогнестійкості. Приміщення для обслуговування, ремонту і налагодження ЕОМ відносяться за пожежо-вибухонебезпекою до категорії В.

У досліджуваному робочому приміщенні всі вимоги щодо пожежної безпеки виконано [13,14].

## Шум

У кімнаті знаходиться одне робоче місце користувача ПК, яке устатковане монітором, клавіатуру, мишку та системний блок вінчестером, що містить, зокрема, вінчестер, відео карту з системою охолодження, та три вентилятора системи охолоджування ПК. Таким чином, у приміщенні мають місце шуми механічного і аеродинамічного походження.

Орієнтовні еквівалентні рівні звукового тиску джерел шуму, що діють на оператора на його робочому місці, представлені в табл. 4.2 [15].

*Таблиця 4.2*

Рівні шуму

| Вид трудової  діяльності,  робоче місце | Рівні звукового тиску в дБ в октавних смугах з середньогеометричними частотами, Гц | | | | | | | | | Рівні шуму та еквівалентні рівні шуму, дБА, дБАекв. |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31,5 | 63 | 125 | 250 | 500 | 1000 | 2000 | 4000 | 8000 |
| Наукова діяльність, програмування, викладання та навчання. | 86 | 71 | 61 | 54 | 49 | 45 | 42 | 40 | 38 | 50 |

Допустимий еквівалентний рівень шуму для робочого місця програміста складає 50 дБА.

Рівень шуму на робочому місці не перевищує допустимий.

## Висновок

Показники приміщення знаходяться в межах норми. Отже, робота у кімнаті не буде завдавати шкоди здоров’ю людини. Дотримання зазначених вимог до організації робочого місця не лише збереже гарне самопочуття програміста, а і сприятиме високій працездатності.

# Список літератури

1. Воеводин В.В. Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. – СПб., БХВ-Петербург, 2002. – 608 с.
2. Богачёв К.Ю. Основы параллельного программирования / К.Ю. Богачёв. – М., БИНОМ. Лаборатория знаний, 2003. – 342 с.
3. Турчина В.А. Алгоритми перерахування всіх паралельних упорядкувань фіксованої довжини: зб. наук. пр. / В.А. Турчина, Ю.С. Зозуля, А.К. Підаш // Питання прикладної математики і математичного моделювання. – Д., 2013. – С. 256 – 261.
4. Коффман Э.Г. Введение в теорию расписаний / Э.Г. Коффман. – М., «НАУКА», 1984. – 333 с.
5. Лазарев А.А. Теория расписаний. Задачи и алгоритмы / А.А. Лазарев, Е.Р. Гафаров. – М., Московский государственный университет им. М.В. Ломоносова (МГУ), 2011. – 222 с.
6. Бурдюк В.Я. Алгоритмы параллельного упорядочения / В.Я. Бурдюк, В.А. Турчина. – Днепропетровск,- 1985. – 84 с.
7. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М., Мир, 1982. – 416 с.
8. ДСанПіН 3.3.2.007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ЕОМ [Електронний ресурс]. – Режим доступу: <http://document.ua/derzhavni-sanitarni-pravila-i-normi-roboti-z-vizualnimi-disp-nor4881.html>.
9. Самгін Е.Б. Освітлення робочих місць. / Е.Б. Самгін - М., МІРЕА, 1989. - 186 с.
10. ДБН В.25-28-2006. Природне і штучне освітлення [Електронний ресурс]. – <http://www.info-build.com.ua/normativ/detail.php?ID=45079>.
11. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Електронний ресурс]. – Режим доступу: http://www.document.ua/sanitarni-normi-mikroklimatu-virobnichih-primishen-nor4880.html.
12. ДНАОП 0.00-1.21-98 Правила безпечної експлуатації електроустановок споживачів [Електронний ресурс]; офіц. текст станом на 10 травня 2014 р. / Кабмін України. – Режим доступу: http://www.document.ua/sanitarni-normi-mikroklimatu-virobnichih-primishen-nor4880.html.
13. ГОСТ 12.1.004-91. ССБТ. Пожарная безопасность. Общие требования [Електронний ресурс]. – Режим доступу: http://document.ua/ssbt.-pozharnaja-bezopasnost.-obshie-trebovanija-nor3057.html.
14. Типові норми належності вогнегасників споживачів [Електронний ресурс]; офіц. текст станом на 10 травня 2014 р. / Кабмін України. – Режим доступу: http://zakon4.rada.gov.ua/laws/show/z0554-04.
15. ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку [Електронний ресурс]. – Режим доступу: http://mozdocs.kiev.ua/view.php?id=1789.

# Додаток А

Лістинг програми

class Graph:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

using System.Runtime.InteropServices;

using System.Data.Odbc;

using System.Threading;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using System.Drawing.Drawing2D;

using System.Collections;

namespace Уровневый\_принцип

{

public class Graph

{

List<List<int>> A;

List<List<int>> mainA;

List<List<int>> S;

List<List<int>> s;

public List<List<int>> Rez;

public List<List<int>> Smin = new List<List<int>>();

public List<List<int>> Smax = new List<List<int>>();

List<int> h;

Random r;

public Graph()

{

h = new List<int>();

A = new List<List<int>>();

mainA = new List<List<int>>();

Rez = new List<List<int>>();

r = new Random();

}

List<int> ConnectedNode(int el, List<List<int>> A)

{

List<int> returning = new List<int>();

for (int i = 1; i < A.Count; i++)

{

if (A[A[0].IndexOf(el)][i] == 1)

returning.Add(A[0][i]);

}

return returning;

}

int max(List<int> l)

{

int m =0;

for (int i=0; i<l.Count; i++)

{

if (m < l[i])

{

m = l[i];

}

}

return m;

}

public void Draw(Graphics g)

{

Dictionary<int, Point> Node = new Dictionary<int, Point>();

g.Clear(Color.Snow);

int w = 5;

int x1 = 10, dx = 70;

int y1 = 10, dy = 70;

S = FindS(mainA, 0);

if (S.Count > 0)

{

for (int i = 0; i < S.Count; i++)

{

for (int j = 0; j < S[i].Count; j++)

{

Node.Add(S[i][j], new Point(x1 + dx \* j, y1 + dy \* i));

}

}

for (int i = 0; i < S.Count; i++)

{

for (int j = 0; j < S[i].Count; j++)

{

Point p1 = new Point(), p2;

List<int> connected = ConnectedNode(S[i][j], mainA);

for (int k = 0; k < connected.Count; k++)

{

Node.TryGetValue(S[i][j], out p1);

Node.TryGetValue(connected[k], out p2);

Color c = Color.FromArgb(r.Next(0, 255), r.Next(0, 255), r.Next(0, 255));

Pen pen = new Pen(c, w);

pen.EndCap = LineCap.ArrowAnchor;

g.DrawLine(pen, p1, p2);

}

Node.TryGetValue(S[i][j], out p1);

g.DrawString((S[i][j]).ToString(), new Font("Times New Roman", 12.0f), Brushes.Black, p1);

}

}

}

}

List<int> AND(List<int> small, List<int> big)

{

List<int> temp = new List<int>();

for (int i=0; i<small.Count; i++)

{

if ((big.Contains(small[i])))

{

temp.Add(small[i]);

}

}

return temp;

}

static object DeepClone(object obj)

{

object objResult = null;

using (MemoryStream ms = new MemoryStream())

{

BinaryFormatter bf = new BinaryFormatter();

bf.Serialize(ms, obj);

ms.Position = 0;

objResult = bf.Deserialize(ms);

}

return objResult;

}

List<List<int>> Remove(List<int> index)

{

List<List<int>> NEW = new List<List<int>>();

for (int i = 0; i < A.Count; i++)

{

if (!(index.Contains(A[i][0])))

{

List<int> temp = new List<int>();

for (int j = 0; j < A[i].Count; j++)

{

if(!(index.Contains(A[0][j])))

{

temp.Add(A[i][j]);

}

}

NEW.Add(temp);

}

}

return NEW;

}

public void Build()

{

A = DeepClone(mainA) as List<List<int>>;

Rez.Clear();

List<int> index = new List<int>(); // номера вершин, которые мы занесли в упорядочение на это место hi

List<int> S0 = new List<int>();

for (int i = 0; i < h.Count; i++) //для всех hi

{

if (index.Count > 0)

{

Rez.Add(DeepClone(index) as List<int>);

}

A = Remove(index);

index.Clear();

if (A.Count == 0)

{

return;

}

S = FindS(A, 0);

s = FindS(A, 1);

int k = 0;

for (int j = 0; j < h[i]; j++) // не более, чем hi раз

{

if ((S0.Count == 0) ||(j==0)) //брать из S[0] уже нечего

{

if (k < S.Count)

{

S0 = AND(S[k], s[0]);

k++;

}

else

{

break;

}

}

if (S0.Count > 0)

{

int t = r.Next(0, S0.Count );

index.Add(S0[t]);

S0.RemoveAt(t);

}

}

}

if (index.Count > 0)

{

Rez.Add(DeepClone(index) as List<int>);

A = Remove(index);

}

if (A.Count > 1)

{

Rez.Clear();

}

}

public void FindMaxMin(int n)

{

int max = 0;

int min = System.Int32.MaxValue;

for (int i = 0; i < n; i++)

{

Build();

if (Rez.Count >= max)

{

max = Rez.Count;

Smax = DeepClone(Rez) as List<List<int>>;

}

if (Rez.Count <= min)

{

min = Rez.Count;

Smin = DeepClone(Rez) as List<List<int>>;

}

}

}

List<int> LookForNotIn(List<List<int>> A, int flag)

{

List<int> a = new List<int>();

for (int i = 1; i < A.Count(); i++)

{

bool IsIn = false;

if (flag == 1)

{

for (int j = 1; j < A.Count; j++)

{

if (A[j][i] != 0) { IsIn = true; }

}

}

if (flag == 0)

{

for (int j = 1; j < A.Count; j++)

{

if (A[i][j] != 0) { IsIn = true; }

}

}

if (!IsIn) { a.Add(A[i][0]); }

}

return a;

}

public bool Check()

{

List<List<int>> ss = FindS(mainA, 0);

if (ss== null)

{

return false;

}

return true;

}

List<List<int>> FindS(List<List<int>> a, int flag)

{

int acount = a.Count;

List<List<int>> s = new List<List<int>>();

while (a.Count() > 1)

{

if (s.Count > acount)

{

MessageBox.Show("Граф має бути ациклічним та не мати петель!");

return null;

}

List<int> temp = new List<int>();

if (flag == 1) temp = LookForNotIn(a, 1);

if (flag == 0) temp = LookForNotIn(a, 0);

s.Add(temp);

List<List<int>> newA = new List<List<int>>();

List<int> Aa = new List<int>();

for (int i = 0; i < a.Count(); i++)

{

if (temp.Contains(a[i][0]))

continue;

else

{

Aa.Add(a[i][0]);

}

}

newA.Add(Aa);

for (int i = 1; i < a.Count(); i++)

{

List<int> temp1 = new List<int>();

temp1.Add(a[i][0]);

for (int j = 1; j < a.Count(); j++)

{

//if ((temp.Contains(i)) || (temp.Contains(j)))

if ((temp.Contains(a[i][0])) || (temp.Contains(a[j][0])))

continue;

else

{

temp1.Add(a[i][j]);

}

}

if (temp1.Count() > 1)

{

newA.Add(temp1);

}

}

a = newA;

}

if (flag == 0)

{

List<List<int>> sInvers = new List<List<int>>();

for (int i = s.Count() - 1; i >= 0; i--)

{

sInvers.Add(s[i]);

}

return sInvers;

}

return s;

}

public bool ReadFromFile()

{

Excel.Application App = new Excel.Application();

Excel.Range rng = null;

OpenFileDialog OFD = new OpenFileDialog();

OFD.Filter = "Excel Worksheets|\*.xls;\*.xlsx;\*.xlsm;\*.csv";

if (OFD.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

App.Visible = true;

App.Workbooks.Open(OFD.FileName);

}

else { return false; }

try { rng = (Excel.Range)App.InputBox("Оберіть діапазон", "Range Selection", Type: 8); }

catch { } // user pressed cancel on input box

if (rng != null)

{

mainA.Clear();

A.Clear();

for (int i = 0; i < rng.Columns.Count; i++)

{

List<int> temp = new List<int>();

for (int j = 0; j < rng.Columns.Count; j++)

{

temp.Add(Convert.ToInt32(rng.Cells[i + 1, j + 1].Value2));

}

A.Add(temp);

mainA.Add(temp);

}

}

try { Marshal.ReleaseComObject(rng); }

catch { }

finally { rng = null; }

try { App.Quit(); Marshal.ReleaseComObject(App); }

catch { }

finally { App = null; }

if (A.Count > 0)

{

return true;

}

return false;

}

public void ReadFromTable(DataGridView dg\_enter)

{

int n = dg\_enter.Rows.Count;

if (dg\_enter.Rows[n - 1].Cells[0].Style.BackColor == Color.Snow)

{

n = dg\_enter.Columns.Count - 1;

}

List<int> Aa = new List<int>();

for (int i = 0; i < n + 1; i++)

{

Aa.Add(i);

}

mainA.Add(Aa);

A.Add(Aa);

for (int i = 0; i < n ; i++)

{

List<int> temp = new List<int>();

temp.Add(i + 1);

for (int j = 0; j < n; j++)

{

temp.Add(Convert.ToInt32(dg\_enter.Rows[i].Cells[j].Value));

}

mainA.Add(temp);

A.Add(temp);

}

}

public void S\_DG\_Max(DataGridView dg\_enter)

{

dg\_enter.ColumnHeadersVisible = false;

dg\_enter.RowHeadersVisible = false;

dg\_enter.Rows.Clear();

if (Smax.Count > 0)

{

dg\_enter.RowCount = Smax.Count;

dg\_enter.ColumnCount = max(h);

for (int i = 0; i < Smax.Count; i++)

{

for (int j = 0; j < Smax[i].Count; j++)

{

dg\_enter.Rows[i].Cells[j].Value = Smax[i][j];

}

for (int j = 0; j < h[i]; j++)

{

dg\_enter.Columns[j].Width = 30;

dg\_enter.Rows[i].Cells[j].Style.BackColor = Color.MistyRose;

}

}

}

}

public void S\_DG\_Min(DataGridView dg\_enter)

{

dg\_enter.ColumnHeadersVisible = false;

dg\_enter.RowHeadersVisible = false;

dg\_enter.Rows.Clear();

if (Smin.Count > 0)

{

dg\_enter.RowCount = Smin.Count;

dg\_enter.ColumnCount = max(h);

for (int i = 0; i < Smin.Count; i++)

{

for (int j = 0; j < Smin[i].Count; j++)

{

dg\_enter.Rows[i].Cells[j].Value = Smin[i][j];

}

for (int j = 0; j < h[i]; j++)

{

dg\_enter.Columns[j].Width = 30;

dg\_enter.Rows[i].Cells[j].Style.BackColor = Color.MistyRose;

}

}

}

}

public void GetH(DataGridView dg\_enter, bool flag)

{

h.Clear();

if (flag)

{

int H = 0;

try

{

H = Convert.ToInt32(dg\_enter.Rows[0].Cells[0].Value.ToString());

}

catch

{ ;}

for (int i = 0; i < A.Count; i++)

{

h.Add(H);

}

}

else

{

for (int i = 0; i < dg\_enter.RowCount - 1; i++)

{

try

{

h.Add(Convert.ToInt32(dg\_enter.Rows[i].Cells[0].Value.ToString()));

}

catch { ;}

}

}

}

public void UpdateDG(DataGridView dg\_enter)

{

if (A.Count > 0)

{

dg\_enter.RowHeadersWidth = 60;

dg\_enter.RowCount = A.Count - 1;

dg\_enter.ColumnCount = A.Count - 1;

for (int i = 0; i < A.Count - 1; i++)

{

dg\_enter.Columns[i].Width = 22;

if (i == A.Count - 1)

{ }

else

{

dg\_enter.Columns[i].HeaderText = (A[i + 1][0]).ToString();

dg\_enter.Rows[i].HeaderCell.Value = (A[i + 1][0]).ToString();

}

}

for (int i = 0; i < A.Count - 1; i++)

{

for (int j = 0; j < A.Count - 1; j++)

{

if ((i == A.Count - 1) || (j == A.Count - 1))

{ }

else

{

dg\_enter.Rows[i].Cells[j].Value = A[i + 1][j + 1].ToString();

dg\_enter.Rows[i].Cells[j].Style.BackColor = Color.White;

dg\_enter.Rows[i].Cells[j].Style.ForeColor = Color.SaddleBrown;

}

}

}

}

}

}

}

class Form1:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Excel = Microsoft.Office.Interop.Excel;

using System.Runtime.InteropServices;

using System.Data.Odbc;

using System.Threading;

using System.IO;

namespace Уровневый\_принцип

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

Graph G;

bool IsConst = false;

bool FromFile = false;

private void Form1\_Load(object sender, EventArgs e)

{

radioButton1.Checked = true;

dg\_h.ColumnCount = 1;

dg\_h.Columns[0].Width = 22;

dg\_h.ColumnHeadersVisible = false;

dg\_h.RowHeadersVisible = false;

dg\_enter.RowCount = 1;

dg\_enter.Columns[0].Width = 22;

dg\_enter.RowHeadersWidth = 60;

dg\_h.RowCount = 1;

dg\_enter.AllowUserToAddRows = true;

}

private void LoadFromFile\_Click(object sender, EventArgs e)

{

FromFile = true;

G = new Graph();

G.ReadFromFile();

G.UpdateDG(dg\_enter);

FromFile = false;

if (!G.Check())

{

G = null;

return;

}

Graphics g = panel1.CreateGraphics();

G.Draw(g);

}

private void button1\_Click(object sender, EventArgs e)

{

if (G != null)

{

Graphics g = panel1.CreateGraphics();

G.Draw(g);

G.GetH(dg\_h,IsConst);

G.FindMaxMin(100);

G.S\_DG\_Max(dg\_Smax);

G.S\_DG\_Min(dg\_Smin);

label3.Text = "З мінімальною довжиною \n впорядкування. \n Довжина - " + dg\_Smin.RowCount.ToString();

label4.Text = "З максимальною довжиною \n впорядкування. \n Довжина - " + dg\_Smax.RowCount.ToString();

}

}

private void radioButton2\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton2.Checked == true)

{

dg\_h.AllowUserToAddRows = false;

dg\_h.RowCount = 1;

label2.Text = "h =";

IsConst = true;

}

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

if (radioButton1.Checked == true)

{

dg\_h.AllowUserToAddRows = true;

label2.Text = "h[i]=";

IsConst = false;

}

}

private void dg\_h\_RowsAdded(object sender, DataGridViewRowsAddedEventArgs e)

{

}

private void button1\_Click\_1(object sender, EventArgs e)

{

// FromFile = true;

G = new Graph();

G.ReadFromTable(dg\_enter);

G.UpdateDG(dg\_enter);

if (!G.Check())

{

G = null;

return;

}

Graphics g = panel1.CreateGraphics();

G.Draw(g);

}

private void dg\_enter\_RowsAdded(object sender, DataGridViewRowsAddedEventArgs e)

{

if (!FromFile)

{

dg\_enter.ColumnCount = dg\_enter.Rows.Count;

for (int i = 0; i < dg\_enter.ColumnCount - 1; i++)

{

dg\_enter.Rows[i].Visible = true;

dg\_enter.Columns[i].Visible = true;

for (int j = 0; j < dg\_enter.ColumnCount - 1; j++)

{

dg\_enter.Rows[i].Cells[j].Style.BackColor = Color.White;

dg\_enter.Rows[i].Cells[j].Style.ForeColor = Color.Black;

}

}

dg\_enter.Columns[dg\_enter.Rows.Count - 1].Width = 22;

dg\_enter.Rows[dg\_enter.Rows.Count - 1].Cells[dg\_enter.Rows.Count - 1].Value = 0;

dg\_enter.Rows[dg\_enter.Rows.Count - 1].Cells[dg\_enter.Rows.Count - 1].Style.BackColor = Color.Snow;

dg\_enter.Rows[dg\_enter.Rows.Count - 1].Cells[dg\_enter.Rows.Count - 1].Style.ForeColor = Color.Snow;

dg\_enter.Columns[dg\_enter.Rows.Count - 1].HeaderText = Convert.ToString(dg\_enter.Rows.Count);

dg\_enter.Rows[dg\_enter.Rows.Count - 1].HeaderCell.Value = Convert.ToString(dg\_enter.Rows.Count);

for (int i = 0; i < dg\_enter.ColumnCount - 1; i++)

{

dg\_enter.Rows[dg\_enter.Rows.Count - 1].Cells[i].Value = 0;

dg\_enter.Rows[i].Cells[dg\_enter.Rows.Count - 1].Value = 0;

dg\_enter.Rows[i].Cells[dg\_enter.Rows.Count - 1].Style.BackColor = Color.Snow;

dg\_enter.Rows[dg\_enter.Rows.Count - 1].Cells[i].Style.BackColor = Color.Snow;

dg\_enter.Rows[i].Cells[dg\_enter.Rows.Count - 1].Style.ForeColor = Color.Snow;

dg\_enter.Rows[dg\_enter.Rows.Count - 1].Cells[i].Style.ForeColor = Color.Snow;

}

}

}

private void dg\_enter\_RowsRemoved(object sender, DataGridViewRowsRemovedEventArgs e)

{

dg\_enter.Columns.RemoveAt(e.RowIndex);

}

}

}