**ДНІпропетровський національний університет**

**імені Олеся Гончара**

**ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ**

**КАФЕДРА обчислювальної математики та математичної кібернетики**

**Дипломна робота**

**за рівнем магістра**

на тему: «Аналіз узагальнених задач паралельного упорядкування, що допускають переривання»

Виконала: студентка VІ курсу, групи ПС-14м-1

спеціальності 8.04030302 - системи і методи прийняття рішень

Зозуля Юлія Станіславівна

Керівник \_\_\_\_\_\_\_\_\_\_\_\_ к.ф.-м.н. доц .Турчина В.А.

підпис

Рецензент \_\_\_\_\_\_\_\_\_\_\_ д.т.н., доц. Земляна С.В.

підпис

м. Дніпропетровськ - 2016 року

**РЕФЕРАТ**

Дипломна робота: \_ с., рис. \_, табл. \_, джерел 8, 1 додаток.

***Об’єкт дослідження***: узагальнена задача паралельного впорядкування.

***Мета роботи***: розробка методів, основаних на рівневому принципі та декомпозиції при аналізі узагальнених задач паралельного упорядкування; виділення підкласів графів, оптимальні упорядкування яких належать до класу упорядкувань з перериваннями; розробка програмного продукту та аналіз отриманих результатів.

***Одержані висновки та їх новизна***: модифіковано алгоритм розв’язання узагальненої задачі паралельного впорядкування для дерева, сформульовано та доведено твердження, що дозволяє виділити підкласи графів, оптимальні впорядкування яких належать до класу упорядкувань з перериваннями, розроблено новий метод розв’язання узагальненої задачі паралельного впорядкування, який оснований на декомпозиції вхідного графу.

***Результати дослідження можуть бути застосовані*** для розв’язання практичних задач, моделі яких являють собою системи завдань, на порядок виконання яких накладено умови часткового порядку, а час виконання задач в загальному випадку різний, а також для оптимального вибору типу системи, на якій система завдань може бути виконана максимально ефективно.

***Перелік ключових слів***:задачі теорії розкладів, оптимізаційні задачі на графах, задача паралельного впорядкування, паралельні впорядкування з перериваннями, рівневий принцип, методи декомпозиції.

**ANNOTATION**

The graduation research of the 6–year student Y. Zozulia (DNU, Applied Mathematics Department, the Calculating Mathematics and Mathematical Cybernetics Chair) deals with the

investigation of precision polynomial complexity algorithms developed for the classical problem in solving the generalized problem of parallel ordering. Formulated and proved the theorem about the possible violation of the principle of choice of level vertices, as well as statements about the sufficient conditions for that ordering, built on the level principle, is optimal. Selected subclasses of graphs for which the algorithm based on level principle exactly solves the generalized problem of ordering. There was developed software which implements this algorithm. There was given the recommendations about the usefulness of the algorithm based on level principle to solve the generalized problem of ordering.

The work is interesting for those, who solve problems, models of which are the tasks with sequence of partial order imposed conditions.

Bibliography 15, pictures 34, tables 19, 1 supplement.

**Зміст**

[Вступ 6](#_Toc440917934)

[Розділ 1. Місце задач паралельного упорядкування в класі задач теорії розкладів 8](#_Toc440917935)

[1.1. Постановки задач теорії розкладів 8](#_Toc440917936)

[1.2. Розклади без переривань і з перериваннями 10](#_Toc440917937)

[1.3. Задачі паралельного упорядкування в класі задач теорії розкладів 11](#_Toc440917938)

[Розділ 2. Узагальнені задачі паралельного упорядкування 14](#_Toc440917939)

[2.1. Постановки узагальнених задач 14](#_Toc440917941)

[2.2. Алгоритми розв’язання узагальнених задач для дерева 16](#_Toc440917942)

[2.3. Підкласи оптимальних упорядкувань, що допускають переривання 24](#_Toc440917943)

[2.4. Методи декомпозиції при аналізі узагальнених задач паралельного упорядкування 28](#_Toc440917944)

[2.5. Підкласи графів, оптимальні упорядкування яких вимагають переривань 35](#_Toc440917945)

[Розділ 3. Програмна реалізація алгоритмів 42](#_Toc440917946)

[3.1. Опис програми 42](#_Toc440917947)

[3.2. Керівництво користувача 50](#_Toc440917948)

[3.3. Результати тестування програми 51](#_Toc440917949)

[3.4. Аналіз отриманих в програмі даних 52](#_Toc440917950)

[Висновки 53](#_Toc440917951)

[Список літератури 54](#_Toc440917952)

# Вступ

Задачі теорії розкладів почали активно вивчатися науковцями з середини минулого сторіччя. Інтерес до них був обумовлений перш за все практичними потребами, а науковців вони цікавили також і тому, що за постановкою вони здаються простими, але, як виявилося, для багатьох із них не існує точних методів поліноміальної складності. Якщо на порядок виконання робіт накладаються технологічні обмеження на порядок виконання завдань, то задачу можна також класифікувати як оптимізаційну задачу на графах. Зокрема, серед таких задач виділяють задачі, які отримали назву задач оптимального паралельного упорядкування вершин графу.

Перші теоретичні результати, отримані для розв’язання таких задач, базувалися на припущенні, що всі завдання, яким відповідають вершини графу, мають однаковий час виконання. На практиці ж, такі випадку зустрічаються рідко. Тому з практичної точки зору більш доцільно розглядати випадки, що стосуються різного часу виконання задач. Саме така задача і розглядається в даній дипломній роботі.

Оскільки першим алгоритмом розв’язання класичних задач був алгоритм, запропонований для дерев, то і для узагальненої задачі спочатку розглядалися графи, що являються деревами. Для цього випадку в роботі модифіковано алгоритм, який був розроблений для задачі у класичній постановці для випадку, коли граф обмежень – дерево, - а саме алгоритм, оснований на рівневому принципі.

Крім того, було запропоновано метод декомпозиції для розв’язання задач для випадку, коли граф обмежень довільний.

Останнім часом з’являються цікаві результати, отримані для задач теорії розкладів для випадку, коли у процесі виконання завдань дозволяються переривання. Тому однією з задач, поставлених переді мною, було виділення класів задач, оптимальні розв’язки яких належать до класу впорядкувань з перериваннями. Апріорне знання того, чи може дозвіл переривань зменшити довжину впорядкування є надзвичайно важливим для практичних задач, оскільки в залежності від того, дозволяються переривання чи ні, може обиратися різні системи, на яких будуть виконуватися завдання. Крім того, алгоритми розв’язання задач з перериваннями і без переривань є різними.

В першому розділі\_\_

# Розділ 1. Місце задач паралельного упорядкування в класі задач теорії розкладів

## Постановки задач теорії розкладів

В загальному формулюванні задача теорії розкладів полягає в наступному:

за допомогою деякої множини ресурсів (обслуговуючих приладів, робочих і т.д.) має бути виконана деяка фіксована система завдань. Метою є побудова розкладу, тобто деякої послідовності виконання завдань з інформацією, коли та на якому ресурсі має виконуватися завдання, - такого, що послідовність виконання завдань задовольняє технологічних вимогам, і при цьому є оптимальним.

Загальна модель задач теорії розкладів складається з сукупності моделей, що описують ресурси, систему завдань, обмеження при складанні розкладів, а також міри їх оцінки.

У ряді моделей ресурси – це набір процесорів. Позначимо їх . В залежності від задачі вони можуть мати як однакові характеристики, так і бути різними за функціональністю та/або швидкодією. В деяких випадках описують також додаткові ресурси. Це може бути оперативна пам'ять, зовнішня пам'ять, з’єднання з мережею Інтернет, бібліотеки, тощо.

Система завдань в найбільш загальному випадку може бути представлена як система

що побудована наступним чином:

1. – набір завдань, які потрібно виконати.
2. позначає задане на відношення часткового порядку (не рефлексивне), котре визначає обмеження на послідовність виконання завдань. Запис означає, що має бути виконане раніше, ніж почнеться виконання .
3. – матриця розміром , елементи якої представляють собою час виконання завдання ( ) на процесорі ( ). , якщо не може бути виконане на процесорі . Для кожного існує , для якого . Якщо всі процесори ідентичні, то матриця вироджується в вектор .
4. – набір, –та координата якого являє собою кількість ресурсу типу , необхідного для виконання завдання .
5. – вага, що інтерпретується як вартість перебування завдання в системі.

При дослідженні задач розглядають два підходи, згідно з якими завдання виконуються з перериваннями або без переривань.

Зазвичай у якості міри ефективності розкладу розглядають максимальний час завершення задач (тобто довжину розкладу) чи середньозважений час завершення завдань.

Наведена модель є загальною. Зазвичай всі реальні задачі – це її частковий випадок. В подальшому будемо вважати, що тривалість виконання роботи не залежить від виконавця.

Величину, що відповідає часу завершення всіх робіт називають довжиною розкладу. Також надалі будемо називати розклад оптимальним, якщо він або має задану довжину () і вимагає мінімальної кількості виконавців (), або при заданій кількості виконавців має мінімальну довжину.

## Розклади без переривань і з перериваннями

Якщо в системі всі завдання мають однакові часи виконання, то можна вважати, що всі ці часи рівні одиниці. У цьому випадку ми маємо так звану UET-систему (unit-execution-time). Це класична постановка задачі. В такому випадку задача переривань не виникає.

Якщо ж час виконання робіт у загальному випадку різний, то виділяють два підходи – розв’язання задач з дозволом чи забороною переривань.

Розклад без переривань передбачає, що завдання після початку виконання вже не може бути зупинене до повного завершення. Розклад з перериваннями дозволяє знімати задачу з ресурсу (наприклад, з процесора) до її завершення, з подальшим відновленням виконання завдання.

При цьому вважаємо, що сумарний час виконання завдання залишається незмінним, тобто завдання відновлюється саме з того моменту, на якому було зупинене.

При перериванні з’являється необхідність знімати задачу з виконання до її завершення і зберігати інформацію про неї протягом часу очікування. Але такі затрати зазвичай є незначними. Також вважаємо, що не виникає необхідності в додатковому часі виконання, пов’язаному зі збереженням та подальшим зчитуванням інформації про стан задачі.

## Задачі паралельного упорядкування в класі задач теорії розкладів

Задачі теорії розкладу природнім чином пов’язуються з задачами впорядкувань. Покажемо це. Для початку введемо деякі визначення.

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, то слідує, що розташовано в лівіше за .

Для існування паралельного впорядкування необхідно і достатньо, або граф був ациклічним [1].

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

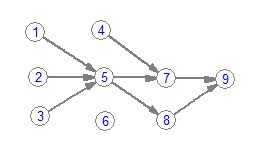


Рис.1.1. Вигляд графу

Побудуємо деякі паралельні впорядкування вершин цього графу. Їх зображено на рис. 1.2 та рис. 1.3.



Рис. 1.2. Паралельне впорядкування вершин графу *G*



Рис. 1.3. Паралельне впорядкування вершин графу *G*



Рис. 1.4. Паралельне впорядкування вершин графу *G*

В даних таблицях стовпчики відповідають місцям в упорядкуванні. Так, -й стовпчик відповідає множині . Відповідно, кількість елементів в -му стовпчику – це потужність множини . Таким чином, кількість строк в таблиці – це ширина впорядкування, а кількість стовпчиків відповідає довжині.

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 [2, 3].

Наприклад, впорядкування, зображене на рис. 1.2. є оптимальним для задачі , а на рис. 1.4. – для задачі .

Оскільки кожному завданню можна поставити у відповідність вершину графу, а порядку слідування робіт – дуги графу, то можна задачі теорії розкладів інтерпретувати як оптимізаційні задачі на графах.

Ациклічний граф може слугувати математичною моделлю обмежень на порядок виконання робіт. Множина вершин графу буде відповідати множині завдань , множина ребер – технологічним обмеженням, заданим відношенням при цьому з того, що слідує, що робота має бути завершена не пізніше, ніж почнеться виконання роботи .

Таким чином, паралельні впорядкування вершин графу будуть однозначним чином задавати допустимі розклади. При цьому довжина упорядкування , а ширина впорядкування відповідатиме мінімальній кількості виконавців, необхідних для виконання робіт ().

Таким чином, оптимізаційним задачам на графах можна однозначно поставити у відповідність оптимізаційні задачі теорії впорядкувань. Надалі будемо будувати оптимальні розклади за допомогою вирішення відповідних оптимізаційних задач на графах.

# Розділ 2. Узагальнені задачі паралельного упорядкування



## Постановки узагальнених задач

У більшості реальних задач виникає необхідність розглядати системи завдань, у яких час виконання задач різний. Такому випадку відповідає узагальнена задача паралельного впорядкування, яка формулюється наступним чином.

Задано граф , де – множина вершин, а – множина дуг. Цей граф задає відношення часткового порядку виконання робіт. Кожній вершині поставлено у відповідність число - час виконання роботи. У загальному випадку часи виконання різні.

Необхідно побудувати таке допустиме впорядкування множини вершин графу , яке або при заданій довжині має мінімальну ширину, або при заданій ширині має мінімальну довжину. Позначимо описані задачі відповідно та .

Надалі у роботі розглядаються лише задачі , тобто задачі, у яких при заданій ширині необхідно мінімізувати довжину.

Розглянемо граф, зображений на рис. 2.1. Позначки біля вершини відповідають часу виконання роботи .

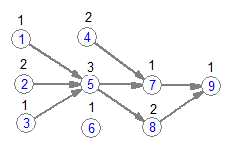


Рис. 2.1. Вигляд графу

Оптимальне впорядкування, побудоване для задачі , можемо бачити в таблиці 2.1.

*Таблиця 2.1.*

Оптимальне впорядкування вершин графу

для узагальненої задачі



В таблиці стовпчики відповідають місцям в упорядкуванні. Таким чином, кількість строк в таблиці – це ширина впорядкування, а кількість стовпчиків відповідає довжині.

Виконання задач, що мають неодиничний час виконання, позначене з’єднанням декількох клітинок таблиці, де кількість таких об’єднаних клітинок відповідає часу виконання конкретної задачі.

Так, з таблиці 2.1. видно, що час виконання задачі 2 дорівнює двом, задачі 5 - трьом, задачі 8 – двом. Довжина даного впорядкування – 8.

## **Алгоритми розв’язання узагальнених задач для дерева**

Для розв’язання задачі паралельного упорядкування у класичній постановці існують точні алгоритми поліноміальної складності, які є точними для випадку, коли – тобто, граф обмежень є лісом або деревом, а також коли ширина бажаного впорядкування

Для того, аби перейти до розгляду алгоритмів побудови оптимальних впорядкувань для узагальненої задачі, розглянемо підхід, заснований на рівневому принципі на прикладі задач у класичній постановці.

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 -те місце в . Видаляємо 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 [4].

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

Але нас цікавлять саме впорядкування та , які знадобляться нам надалі.

Нехай задано граф . Побудуємо для нього впорядкування . Характеризувати рівень вершини буде величина, що обчислюється згідно з наступним виразом:

де довжина впорядкування ;

- номер місця вершини в упорядкуванні.

Вільною будемо називати таку вершину, всі попередники якої вже є в упорядкуванні. При побудові впорядкування будемо брати перш за все ту вільну вершини, яка має максимальний рівень. Якщо таких вершин декілька – беремо будь-яку.

Розглянемо граф , зображений на рис. 2.2.

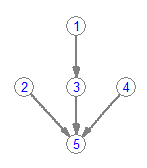


Рис. 2.2. Вигляд графу

Побудуємо впорядкування згідно з рівневим принципом для задачі . Для цього побудуємо для впорядкування , що зображене в таблиці 2.2. Номер строки *k* характеризує номер місця вершини в упорядкуванні, а відповідно – і її рівень.

*Таблиця 2.2.*

Впорядкування вершин графу

|  |  |  |  |
| --- | --- | --- | --- |
| *k = 1* | 1 |  |  |
| *k = 2* | 2 | 3 | 4 |
| *k = 3* | 5 |  |  |

В даній таблиці строки відповідають місцям в упорядкуванні. Кількість строк – довжина впорядкування, стовпчиків – ширина впорядкування.

Тепер побудуємо впорядкування згідно з рівневим принципом (таблиця 2.3).

*Таблиця 2.3.*

Впорядкування вершин графу , побудоване згідно

з рівневим принципом

|  |  |
| --- | --- |
| **1** | **2** |
| **3** | **4** |
| **5** |  |

У випадку узагальненої задачі розрізняють підходи з дозволом та без дозволу переривань.

Розглянемо граф , зображений на рис. 2.3. Необхідно розв’язати задачу .

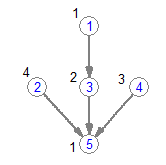


Рис. 2.3. Вигляд графу

Побудуємо оптимальне впорядкування без переривань для задачі (таблиця 2.4.)

*Таблиця 2.4.*

Оптимальне впорядкування без переривань

|  |  |
| --- | --- |
| **1** | **2** |
| **4** |
|
|
| **3** |  |
|  |
| **5** |  |

Довжина отриманого впорядкування – 7. Повторення однієї і тієї ж задачі на суміжних строках означає продовження її виконання. Час виконання задачі має відповідати кількості таких повторень. У випадку задачі без переривань не допускається розірвання цієї послідовності.

Виявляється, що дозвіл на переривання у виконанні робіт може зменшити довжину.

Побудуємо впорядкування з перериваннями. Для цього розглянемо допоміжну задачу.

Множину вершин будемо називати ланцюгом, якщо

.

Знайдемо деякий час , який є спільним дільником для часу виконання кожного з завдань, і в графі для кожного завдання замінимо вершину на ланцюг вершин , де кількість елементів цього ланцюга буде дорівнювати кратності часу виконання даного завдання до часу . Таким чином, отримали допоміжну задачу паралельного впорядкування у класичній постановці (час виконання всіх завдань однаковий).

Позначимо як граф, що отримуємо з заміною вершини ланцюгом , де . Попередники стають попередниками , а її наступники – наступниками . Після цього кожна вершина має час виконання , що дозволяє застосовувати до алгоритми побудови впорядкувань без переривань, розроблені для задач у класичній постановці.

Вершини можуть бути ще раз подані у вигляді ланцюгів вершин з однаковим часом виконання. Позначимо як граф, що отримуємо з заміною кожної вершини ланцюгом з вершин, що мають однаковий час виконання. При збільшенні можна очікувати, що оптимальне впорядкування з перериваннями для системи, що представляється графом стає все ближчим до розкладу з перериваннями для системи, що представляється графом .

Нехай – довжина оптимального розкладу без переривань для системи, заданої графом , що виконується виконавцями, а – з перериваннями.

Тоді справедлива наступна теорема:

**Теорема:**

Нехай маємо граф , , для кожної вершини задано час виконання (у загальному випадку різний), – додатнє дійсне число, якому пропорційні всі , .

Тоді , де – константа, що залежить тільки від [2].

З теореми випливає, що кожного разу, коли ми можемо побудувати оптимальне впорядкування без переривань для задачі паралельного упорядкування у класичній постановці, ми можемо побудувати впорядкування з перериваннями і для узагальненої задачі, котре буде наскільки завгодно близьким до оптимального.

Проілюструємо описаний підхід прикладами.

Допоміжну задачу можна зобразити наступним чином (рис 2.4).

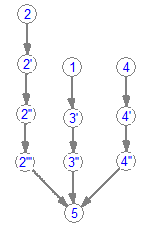


Рис. 2.4. Вигляд графу для допоміжної задачі

Тепер побудуємо впорядкування згідно з рівневим принципом для допоміжної задачі. Результат видно у таблиці 2.5.

*Таблиця 2.5.*

Оптимальне впорядкування з перериваннями



Бачимо переривання у процесі виконання задачі 2 та задачі 4.

Довжина впорядкування – 6. Як бачимо, отримане впорядкування має меншу довжину. Воно і є впорядкуванням з перериваннями для початкової задачі.

Отже, впорядкування з перериваннями можуть мати меншу довжину, ніж впорядкування, побудовані без дозволу переривань. Окрім того, впорядкування з перериваннями можна будувати базуючись на рівневому принципі, який є алгоритмом поліноміальної складності, тоді як при побудові впорядкувань без переривань рівневий принцип не дає оптимальних результатів.

Також постає питання, чи можливо зменшити кількість переривань в упорядкуванні, при цьому не збільшуючи довжину.

При побудові впорядкувань згідно з рівневий принципом, на кожному кроці беремо перш за все ту вільну вершину, яка має максимальний рівень. Якщо таких вершин декілька – беремо будь-яку.

Дещо модифікуємо рівневий принцип. А саме, при побудові впорядкування будемо брати перш за все ту вільну вершину, яка має максимальний рівень. Якщо таких вершин декілька – беремо перш за все ту, яка вже стоїть на попередньому місці в упорядкуванні.

Даний алгоритм не суперечить рівневому принципу, адже по суті ми лише визначаємо спосіб обрання вершини у випадку, коли декілька вершин мають максимальний рівень, тоді як в класичному рівневому принципі цей вибір довільний. Таким чином, для модифікованого алгоритму зберігаються всі властивості, справедливі для рівневого принципу, але при цьому кількість переривань є меншою.

Продемонструємо роботу даного алгоритму на прикладі.

Як було показано раніше, впорядкування для графу з рис. 2.3, побудоване класичним рівневим принципом з дозволом переривань, зображене в таблиці 2.5. Його довжина – 6, а кількість переривань – 2.

В таблиці 2.6. різними кольорами показані роботи, що перериваються.

З таблиці видно, що з перериваннями виконуються роботи 2 та 4. Довжина отриманого впорядкування – 6.

*Таблиця 2.6.*

Оптимальне впорядкування з перериваннями



Тепер побудуємо впорядкування модифікованим способом. Результат наведено в таблиці 2.7.

*Таблиця 2.7.*

Оптимальне впорядкування з перериваннями, побудоване згідно з модифікованим рівневим принципом

|  |  |
| --- | --- |
| **2** | **1** |
| **2** | **4** |
| **2** | **4** |
| **2** | **3** |
| **4** | **3** |
| **5** |  |

Як бачимо, воно містить лише одне переривання, а саме – переривання у процесі виконання роботи 4. При цьому довжина, як і у попередньому випадку, дорівнює шести, тобто, залишилася незмінною.

Отже, для побудови впорядкувань з перериваннями для випадку, коли граф обмежень – дерево, можна рекомендувати використовувати модифікований рівневий принцип.

## Підкласи оптимальних упорядкувань, що допускають переривання

Ми показали на прикладі дерев як дозвіл на переривання у процесі виконання робіт може зменшити довжину впорядкування. Для відповіді на питання, чи має сенс дозволяти переривання у випадку, коли граф обмежень – довільне дерево, розглянемо декілька прикладів і покажемо, як переривання впливають на оптимальність розв’язку.

Розглянемо граф, зображений на рис. 2.5. Позначки біля вершини відповідають часу виконання роботи . Для спрощення сприйняття, у випадку, коли позначку на рисунку будемо пропускати.

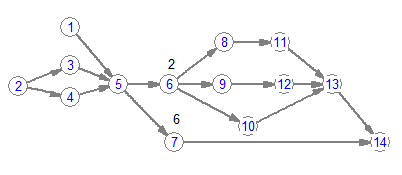


Рис. 2.5. Вигляд графу

Нехай необхідно розв’язати задачу . Оптимальне впорядкування без переривань для наведеного графу зображене в таблиці 2.8.

*Таблиця 2.8.*

Оптимальне впорядкування без переривань

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

Довжина отриманого впорядкування – 12.

Для цього ж графу оптимальне впорядкування, побудоване з дозволом переривань наведене в таблиці 2.9.

*Таблиця 2.9.*

Оптимальне впорядкування з дозволом переривань

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

Як видно з таблиці, задача 7 має переривання у процесі виконання. Довжина отриманого впорядкування – 11. Як бачимо, для даного прикладу дозвіл на переривання у процесі виконання робіт дозволяє зменшити довжину впорядкування.

Розглянемо граф, зображений на рис. 2.6.

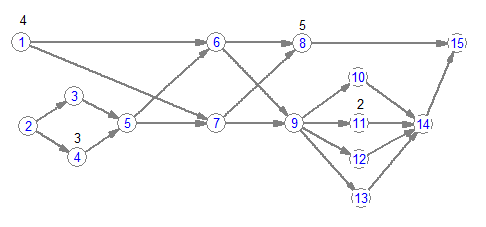


Рис. 2.6. Вигляд графу

Необхідно розв’язати задачу . Оптимальне впорядкування без переривань для наведеного графу зображене в таблиці 2.10, а з перериваннями – в таблиці 2.11.

*Таблиця 2.10.*

Оптимальне впорядкування без переривань

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **1** | | | |  |  | **6** | **8** | | | | |  |  | **15** |  |
|  | **2** | **4** | | | **3** | **5** | **7** | **9** | **10** | **11** | | **12** | **13** | **14** |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Довжина – 15.

*Таблиця 2.11.*

Оптимальне впорядкування з дозволом переривань

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **1** | **1** | **3** | **1** | **1** | **6** | **8** | **8** | **8** | **12** | **8** | **8** | **15** |  |
|  | **2** | **4** | **4** | **4** | **5** | **7** | **9** | **10** | **11** | **11** | **13** | **14** |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Впорядкування має переривання робіт 1 та 8. За рахунок дозволу переривань вдалося зменшити довжину до 13.

З наведених прикладів видно, що дозвіл переривань може призвести до зменшення довжини впорядкування. Очевидно, що це справедливо не для всіх графів та задач. Для багатьох з них дозвіл на переривання не змінює довжини впорядкування.

Так, для того ж графу, але для задачу оптимальне впорядкування що з перериваннями, що без, матиме однакову довжину. Крім того, довжина отриманих впорядкувань дорівняю довжині критичного шляху графу, тобто впорядкування вже не можна оптимізувати. Саме впорядкування наведене в таблиці 2.12.

*Таблиця 2.11.*

Оптимальне впорядкування

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | **1** | **1** | **1** | **1** |  | **6** | **8** | **8** | **8** | **8** | **8** | **15** |  |
|  | **2** | **4** | **4** | **4** | **5** | **7** | **9** | **10** | **12** | **13** | **14** |  |  |
|  |  | **3** |  |  |  |  |  | **11** | **11** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Варто зазначити, що побудова впорядкувань з перериваннями та без – це зовсім різні задачі, і до їх розв’язання можуть застосовуватися зовсім різні методи. Також безпосередньо процес виконання робіт може значно відрізнятися в залежності від дозволу чи заборони переривань. Так, може бути потрібною зовсім інша система виконавців чи організація робіт.

Тому однією з задач, яка потребує подальшого дослідження, є виділення підкласів графів, для яких можна заздалегідь сказати, чи можуть переривання зменшити довжину, чи ні.

Такі задачі є актуальними для широкого класу задач теорії розкладів. Наприклад, в роботах [5,6], в яких розглядається задача Джонсона, було отримано умови, при виконанні яких дозвіл на переривання у виконанні робіт зменшує довжину відповідного розкладу порівно з випадком, коли розклад будується без переривань.

## Методи декомпозиції при аналізі узагальнених задач паралельного упорядкування

Як відомо, для задач паралельного впорядкування у класичній постановці існують точні алгоритми поліноміальної складності для випадку, коли заданий граф обмежень – дерево. Так як точні алгоритми для задачі в загальному випадку мають експоненційну складність, інтерес представляють наближені методи. Вони дозволяють будувати близькі до оптимальних упорядкування за прийнятний час.

Запропоновано наближений метод, який базується на рівневому принципі. Загальною ідеєю є виділення остовного дерева для заданого графу обмежень, побудова оптимального впорядкування для нього, і модифікація отриманого впорядкування таким чином, аби воно було допустимим для початкового графу.

Необхідною умовою застосування методу є зв’язність графу.

Нехай задано граф , що зображений на рис. 2.7.

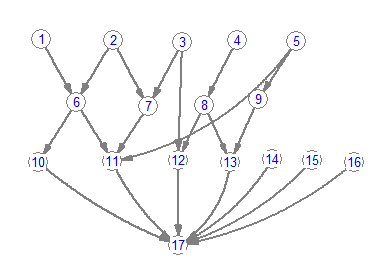


Рис. 2.7. Вигляд графу

За класичним визначенням, остовне дерево — (англ. Spanning tree) — ациклічний зв'язний підграф зв'язаного неорієнтованого графа, який містить всі його вершини [7].

Подалі будемо розуміти під остовним деревом зв'язаного орієнтованого графа ациклічний зв'язний підграф цього графа, який містить всі його вершини.

Для графу з рис. 2.7. знаходимо довільне остовне дерево, наприклад таке, як на рис. 2.8.

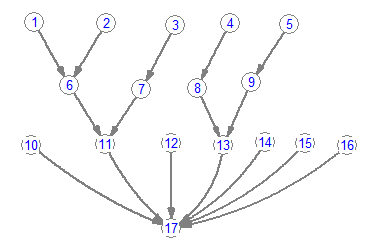


Рис. 2.8. Вигляд графу для допоміжної задачі

Необхідно розв’язати задачу . Знаходимо для графу оптимальне впорядкування за допомогою алгоритму, що базується на рівневому принципі. Результат наведено в таблиці 2.12.

*Таблиця 2.12.*

Оптимальне впорядкування, побудоване згідно з рівневим принципом для остовного дерева

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **3** | **4** | **5** |
| **2** | **7** | **8** | **9** |
| **6** | **10** | **12** | **13** |
| **11** | **14** | **15** | **16** |
| **17** |  |  |  |

Як бачимо, воно не є допустимим (наприклад, вершини 2 та 7, а також 6 та 10 не можуть стояти на одному місці в упорядкуванні). Довжина – 5.

Тож необхідно деяким чином модифікувати впорядкування.

Довільним чином це можна зробити, так, як показано в таблиці 2.13.

*Таблиця 2.13.*

Модифіковане впорядкування

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **3** | **4** | **5** |
| **2** | **8** | **9** |  |
| **7** | **6** | **12** | **13** |
| **10** | **11** | **14** | **15** |
| **16** |  |  |  |
| **17** |  |  |  |

Отримане впорядкування має більшу довжину (6), але є допустимим. Хотілося б отримувати як можна більш близькі до оптимального впорядкування.

Запропоновано наступний алгоритм, який реалізує зазначену ідею.

Нехай задано зв’язний граф . Побудуємо для нього остовне дерево , керуючись наступним правилом:

Для кожної вершини знаходимо список всіх вершин, що за нею слідують. Залишаємо лише одну (довільну дугу) зі знайдених. Решту видаляємо.

Побудований вказаним способом граф буде обов’язково деревом, так як кожна вершина матиме лише одну вихідну дугу, а також міститиме всі вершини початкового графу, так як початковий граф є зв’язним. Таким чином отриманий граф є остовним деревом для початкового графу.

Для отриманого графу побудуємо впорядкування згідно з рівневим принципом. Як відомо, рівневий принцип для дерева є точним алгоритмом, що має поліноміальну складність.

Як було зазначено вище, отримане впорядкування може не бути допустимим для початкового графу . Конфліктною парою будемо називати таку пару вершин із множини , яка не задовольняє відношення часткового порядку, що задається графом . Вершину, яка має бути поставлена в упорядкування раніше з двох, називатимемо початковою, а яка пізніше – кінцевою.

Для нашого прикладу конфліктними будуть пари вершин (2,7) та (6,10).

Для того, аби знайти всі конфлікти, достатньо продивитися всі дуги, що містяться в графі , але не увійшли в остовне дерево , тобто елементи множини .

Для того, аби зробити впорядкування допустимим, і при необхідності збільшення довжини, мінімізувати її відхилення від оптимальної, запропоновано наступний алгоритм:

1. Знаходимо всі конфлікти. Якщо конфліктних пар не знайдено, - поточне впорядкування допустиме, кінець алгоритму. Інакше - обходимо впорядкування з початку.
2. Якщо знаходимо вершину, яка входить до однієї з конфліктних пар, то вибираємо відповідну дугу і йдемо на пункт 3. Якщо конфліктних пар не знайдено, - поточне впорядкування допустиме, йдемо на пункт 8.
3. Якщо у початкової вершини зі знайденого конфлікту немає попередників на попередньому місці в упорядкуванні, то:
   1. Якщо на попередньому місці в упорядкуванні є вільне місце, то ставимо туди початкову вершину. Таким чином, ми позбавилися від даного конфлікту, додатково не порушивши умов часткового порядку, та не збільшивши довжину впорядкування. Йдемо на пункт 2.
   2. Якщо на попередньому місці в упорядкуванні є вершина , яка не має наступників на поточному місці, то міняємо місцями початкову вершину зі знайденого конфлікту, і вершину . Таким чином, ми позбавилися від даного конфлікту, додатково не порушивши умов часткового порядку, та не збільшивши довжину впорядкування. Йдемо на пункт 2.
4. Якщо у кінцевої вершини зі знайденого конфлікту немає наступників на наступному місці в упорядкуванні, то:
   1. Якщо на наступному місці в упорядкування є вільне місце, то ставимо туди кінцеву вершину. Таким чином, ми позбавилися від даного конфлікту, додатково не порушивши умов часткового порядку, та не збільшивши довжину впорядкування. Йдемо на пункт 2.
   2. Якщо на наступному місці в упорядкуванні є вершина , яка не має попередників на поточному місці, то міняємо місцями кінцеву вершину зі знайденого конфлікту, і вершину . Таким чином, ми позбавилися від даного конфлікту, додатково не порушивши умов часткового порядку, та не збільшивши довжину впорядкування. Йдемо на пункт 2.
5. Збільшуємо довжину впорядкування, додавши місце в упорядкування відразу за поточним. Ставимо на нього кінцеву вершину з конфліктної пари.
6. Допоки є можливість додавати вершини на поточне місце або на щойно додане (тобто поки на нього вже не поставили вершин), якщо в графі є вільні вершини, то сортуємо їх, віддаючи пріоритет початковим вершинам з конфліктних пар, і переставляємо їх по одній на відповідне місце.
7. Йдемо на пункт 2.
8. На цей пункт потрапляємо, коли всі вершини розподілені допустимим чином, тобто конфліктів немає. Для того, щоб, якщо можливо, зменшити довжину впорядкування, проходимо по рівням впорядкування починаючи з першого і до останнього. Покладемо . Переходимо на пункт 9.
9. Якщо є вільні місця на рівні з номером , то йдемо на пункт 10. Інакше – на пункт 11.
10. Перевіряємо, чи є вільні вершини (тобто такі, попередники яких містяться в упорядкування на попередніх рівнях) на рівні з номером . Якщо є – ставимо знайдені вершини на вільні місця поточного рівня. Незалежно від того, знайшлись вільні вершини чи ні, йдемо на пункт 11.
11. Якщо рівень - останній в упорядкуванні, тобто, , то кінець алгоритму – шукане допустиме впорядкування знайдене. Інакше йдемо на пункт 9 для .

Проілюструємо даний алгоритм, беручи за основу вищенаведений приклад графу, остовного дерева та впорядкування для нього.

Нагадаємо вигляд упорядкування, отриманого для остовного дерева (таблиця 2.12).

*Таблиця 2.12.*

Оптимальне впорядкування, побудоване згідно з рівневим принципом для остовного дерева

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **3** | **4** | **5** |
| **2** | **7** | **8** | **9** |
| **6** | **10** | **12** | **13** |
| **11** | **14** | **15** | **16** |
| **17** |  |  |  |

1. Конфліктні пари - (2,7) та (6,10).
2. Знаходимо вершину 2 та відповідну їй вершину 7.
3. У вершини немає 2 попередників на попередньому місці. Тож переходимо на пункт 3.1.
   1. На попередньому місці в упорядкуванні вільних місць немає.
   2. На попередньому місці в упорядкуванні є вершина 1, яка не має наступників на поточному місці. Тож міняємо місцями початкову вершину зі знайденого конфлікту 2, і вершину 1.

Впорядкування набуває вигляду, зображеного в табл. 2.14.

*Таблиця 2.14.*

Поточне впорядкування

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **3** | **4** | **5** |
| **1** | **7** | **8** | **9** |
| **6** | **10** | **12** | **13** |
| **11** | **14** | **15** | **16** |
| **17** |  |  |  |

Йдемо на пункт 2.

1. Знаходимо вершину 6 та відповідну їй вершину 10.
2. У вершини 6 є попередники на попередньому місці. Йдемо на пункт 4.
3. У вершини 10 немає наступників на наступному місці в упорядкуванні, тож йдемо на пункт 4.1.
   1. На наступному місці в упорядкуванні вільних місць немає.
   2. На наступному місці в упорядкуванні є вершина 14, яка не має попередників на поточному місці, тож міняємо місцями вершину 10 і вершину 14.

Впорядкування набуває вигляду, зображеного в табл. 2.15.

*Таблиця 2.15.*

Поточне впорядкування

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **3** | **4** | **5** |
| **1** | **7** | **8** | **9** |
| **6** | **14** | **12** | **13** |
| **11** | **10** | **15** | **16** |
| **17** |  |  |  |

Йдемо на пункт 2.

1. Конфліктів немає, йдемо на пункт 8.

8. Вільних місць на рівнях, окрім останнього, немає.

9. Кінець алгоритму.

Вдалося отримати оптимальне впорядкування. Його довжина дорівняє 5.

Даний алгоритм є наближеним, тож в деяких випадках будемо отримувати лише наближені розв’язки. Він має лише незначні ускладнення порівняно з рівневим принципом, тож можемо говорити про невелику складність алгоритму.

За допомогою даного алгоритму можна розв’язувати і узагальнену задачу. Для цього необхідно перейти до допоміжної задачі так, як було показано в даній роботі вище, а потім застосувати описаний алгоритм до допоміжної задачі. Таким чином, отримаємо впорядкування і для початкової узагальненої задачі.

## Підкласи графів, оптимальні упорядкування яких вимагають переривань

Цікавим є питання, чи доцільно дозволяти переривання у системі, тобто, чи належать оптимальні впорядкування до класу впорядкувань з перериваннями. Відповідь на це питання важлива перед усім через те, що в залежності від того, дозволяються переривання чи ні, може обиратися різна система, на якій будуть виконуватися завдання. Також алгоритми розв’язання задач з перериваннями та без переривань належать до різних класів.

Тож метою було створення деякого критерію, який дозволяє з’ясувати, до якого класу належить впорядкування, що має меншу довжину.

Перш за все введемо деякі позначення.

Для задачі паралельного впорядкування у класичній постановці важливим поняттям є множина її допустимих місць в упорядкування. Будуємо допоміжні упорядкування , , а потім для кожної вершини орграфу визначаємо величини – місце вершини в упорядкуванні , та – місце вершини в упорядкуванні .

Узагальнимо ці величини для задачі з різним часом виконання.

Позначимо час виконання роботи через .

Дещо змінимо алгоритм побудови впорядкувань та .

Алгоритм побудови впорядкування :

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

З аналогічними змінами будуємо .

Для прикладу розглянемо граф, зображений на рис. 2.9.

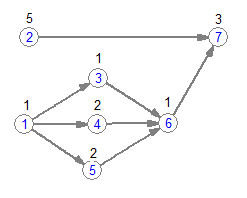


Рис. 2.9. Вигляд графу

Впорядкування наведене в таблиці 2.16.

*Таблиця 2.16.*

Впорядкування

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  | **2** | **2** | **2** | **2** | **2** | **7** | **7** | **7** |  |
|  | **1** | **4** | **4** | **6** |  |  |  |  |  |
|  |  | **5** | **5** |  |  |  |  |  |  |
|  |  | **3** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Впорядкування можемо бачити в таблиці 2.17.

*Таблиця 2.17.*

Впорядкування

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  | **2** | **2** | **2** | **2** | **2** | **7** | **7** | **7** |  |
|  |  | **1** | **4** | **4** | **6** |  |  |  |  |
|  |  |  | **5** | **5** |  |  |  |  |  |
|  |  |  |  | **3** |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Позначимо - крайнє ліве положення вершини в упорядкуванні ,

- крайнє праве положення вершини в упорядкуванні ,

- крайнє ліве положення вершини в упорядкуванні ,

- крайнє праве положення вершини в упорядкуванні .

Так, наприклад, для вершини 4 зазначені величини мають наступні значення:

;

;

;

.

Для відповіді на питання про доцільність дозволу переривань було сформульовано наступне твердження.

***Твердження:***

Нехай маємо граф . Маємо розв’язати узагальнену задачу (час виконання робіт в загальному випадку різний). Якщо в є підграф , такий, що:

* 1. при цьому ця робота може виконуватися одночасно з деякою іншою роботою з ; (1)
  2. , – довільне ціле число, ; (2)
  3. ; (3)

1. ; (4)
2. безпосередньо слідує за вершинами, які мають максимальний допустимий час завершення серед всіх вершин , тобто , при цьому не слідує після ; (5)

то оптимальне впорядкування належить до класу впорядкувань з перериваннями.

*Доведення:*

Нехай переривання у виконанні операцій заборонені.

З (4) слідує, що виконання жодної з робіт не може початися раніше чи одночасно з роботою . Нехай почалося обслуговування роботи виконавцем . Поки не завершиться її виконання, решту робіт може виконувати лише інший виконавець b.

З умов (2) та (3) слідує, що час виконання роботи та cумарний час виконання всіх робіт з однаковий і дорівнює . Так як обслуговування робіт з почалось пізніше за на один такт, то і завершиться виконання на один такт пізніше. В цей час виконавець буде простоювати, так як згідно з умовами в немає вільних вершин.

Після обслуговування всіх робіт з може початися виконання роботи яка задовольняє умові (5). При цьому виконання інших робіт в цей час не відбувається. То ж один з виконавців простоює.

Загальний час простою дорівнює два такти.

Нехай переривання у виконання операцій дозволені.

Так як згідно з умовою (1) в існує вершина з часом виконання 1, яка може виконуватися одночасно з деякою іншою роботою з , то ми можемо перервати обслуговування роботи на користь вказаної роботи. Кількість тактів для виконання робіт з буде дорівнювати , після чого одразу може початися виконання роботи , яка задовольняє умові (5). Час від початку до завершення виконання роботи буде дорівнювати . Тобто таким чином виконання робіт та завершиться одночасно, простоїв немає, час завершення обслуговування всіх згаданих робіт зменшиться на 1.

Отже, оптимальне впорядкування належить до класу впорядкувань з перериваннями, що і треба було довести. [8].

Для прикладу перевіримо виконання умов твердження для вищенаведеного графу

Нехай вершина – це вершина 2, а підграф складається з вершин 3, 4 та 5.

* 1. Умова виконується для вершини 3;
  2. Виконується для , ;
  3. Виконується для вершини 2, ;

1. Спочатку необхідно знайти .

, , . Мінімум серед них – 2.

, 1+1 = 2.

Має виконуватися умова . , , отже, умова виконується.

1. Спочатку необхідно знайти .

, , . Максимум серед них – 4.

Вершиною може бути вершина 6. Вона безпосередньо слідує за всіма вершинами 3, 4, 5 та не слідує після 2.

Отже, умови твердження виконуються, можемо стверджувати, що оптимальне впорядкування для задачі належить до класу впорядкувань з перериваннями.

Наведемо оптимальні впорядкування без переривань (таблиця 2.18) та з перериваннями (таблиця 2.19).

*Таблиця 2.18.*

Оптимальне впорядкування без переривань

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | **2** | | | | |  |  | **7** | | |  |
|  | **1** | **3** | **4** | | **5** | | **6** |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

Довжина впорядкування – 10.

*Таблиця 2.19.*

Оптимальне впорядкування з дозволом переривань

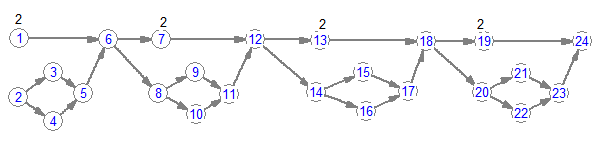
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | **2** | **2** | **3** | **2** | **2** | **7** | **7** | **7** |
| **1** | **4** | **4** | **5** | **5** | **6** |  |  |  |

Впорядкування має переривання роботи 2 на користь роботи 3. За рахунок дозволу переривань вдалося зменшити довжину до 9. Отже, як і стверджувалося в твердженні, оптимальне впорядкування для задачі належить до класу впорядкувань з перериваннями.

Для графів, які були наведені вище в роботі, а саме, графів з рис. 2.5 та 2.6 (для яких переривання також зменшували довжину), також виконуються умови твердження.

Якщо граф можна представити у вигляді послідовної комбінації з графів, для кожного з яких виконуються умови, то довжина впорядкування зменшиться щонайменше на .

Розглянемо граф , який наведено на рис. 2.10.

 Рис. 2.10. Вигляд графу

Його можна представити у вигляді послідовної комбінації під графів, для кожного з яких виконуються умови твердження.

;

;

;

;

.

Звичайно, умови будуть виконуватися і для графу в цілому.

Наведемо оптимальні впорядкування без переривань (таблиця 2.20) та з перериваннями (таблиця 2.21).

*Таблиця 2.20.*

Оптимальне впорядкування без переривань

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | |  |  | **6** | **7** | |  |  | **12** | **13** | |  |  | **18** | **19** | |  |  | **24** |
| **2** | **3** | **4** | **5** |  | **8** | **9** | **10** | **11** |  | **14** | **15** | **16** | **17** |  | **20** | **21** | **22** | **23** |  |

Довжина – 20.

*Таблиця 2.21.*

Оптимальне впорядкування без переривань

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **3** | **1** | **6** | **7** | **9** | **7** | **12** | **13** | **15** | **13** | **18** | **19** | **21** | **19** | **24** |
| **2** | **4** | **5** |  | **8** | **10** | **11** |  | **14** | **16** | **17** |  | **20** | **22** | **23** |  |

Довжина – 16. Переривання допущені у процесі виконання робіт 1, 7, 13 та 19. Отже, за рахунок дозволу переривань вдалося зменшити довжину на 4, тобто на 5% від початкової. Для реальних задач збільшення швидкості виконання системи завдань на 5% може бути суттєвим прискоренням.

Таким чином, вдалося виділити підкласи графів та задач, для яких доцільно дозволяти переривання. Доведення твердження конструктивне, і воно дозволяє відразу в процесі перевірки умов з’ясувати, яку роботу на користь якої доцільно переривати.

Важливою задачею для подальших досліджень є узагальнення отриманих умов на випадок будь-якої ширини впорядкування, а також виділення інших класів графів та задач, для яких оптимальні впорядкування належать до класу впорядкувань з перериваннями.

# Розділ 3. Програмна реалізація алгоритмів

## Опис програми

Для реалізації зазначених алгоритмів була написана програма. В якості мови програмування було обрано мову C#.

Для реалізації алгоритму був створений клас class Graph, який зберігає інформацію про граф обмежень, а також реалізує необхідні функції, спільні для всіх реалізованих методів.

Діаграму класів бачимо на рис. 3.1.

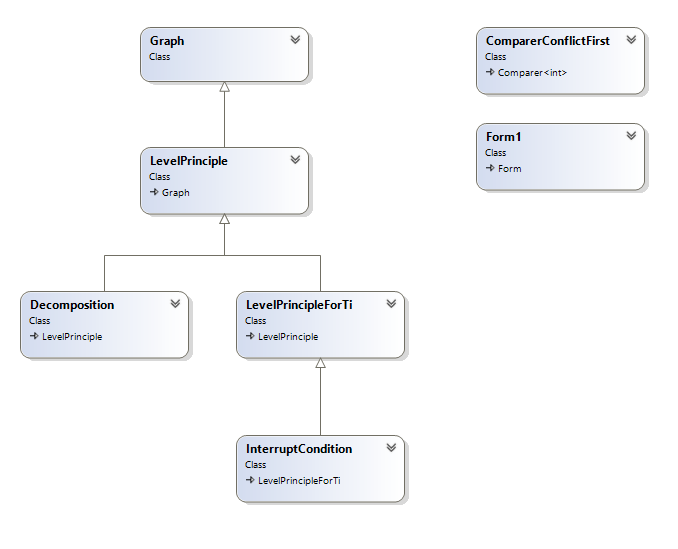


Рис. 3.1. Діаграма класів

Опишемо кожен з класів детальніше.

1. Для класу class Graph було наступні функції – члени класу:

- protected 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.

protected List<int> ConnectedPreviousNode(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 перед поточною.

* public virtual void Draw(Graphics g) - функція, яка приймає в якості параметрів поверхню для малювання та реалізує графічне зображення графу.
* protected 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.
* public static object DeepClone(object obj) – функція, що повертає глибоку копію об’єкта.
* protected virtual 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.
* protected virtual List<List<int>> Remove(List<List<int>> A, List<int> index) - функція, яка приймає в якості параметру список індексів та матрицю суміжності та видаляє з неї строки і стовпці, індекси яких містяться в переданому списку.
* protected 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.
* public bool Check() - функція, що перевіряє, чи є заданий граф підходящим для подальшої роботи.
* protected virtual 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 або , 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 UpdateDG(DataGridView dg\_enter) - функція, яка приймає в якості параметру таблицю DataGridView та виводить на неї матрицю суміжності графу.
* public virtual void SetH(DataGridView dg\_enter, bool flag) - функція, яка приймає в якості параметрів таблицю DataGridView та булеве значення flag. Задає ширину впорядкування.
* public virtual 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.
* public virtual void S\_DG\_Max(DataGridView dg\_enter) - функція, яка приймає в якості параметру таблицю DataGridView та виводить на неї впорядкування з максимальною довжиною.
* public virtual void S\_DG\_Min(DataGridView dg\_enter) - функція, яка приймає в якості параметру таблицю DataGridView та виводить на неї впорядкування з мінімальною довжиною.

1. Клас class LevelPrinciple : Graph є наслідником класу Graph. Реалізує логіку роботи рівневого принципу для класичної задач паралельного впорядкування.

Містить наступні функції - члени класу:

* public LevelPrinciple() : base() – конструктор класу.
* virtual public void Build() - функція, яка будує впорядкування згідно з рівневим принципом.
* override public void FindMaxMin(int n) – перевизначає відповідну функцію батьківського класу.
* override public void S\_DG\_Max(DataGridView dg\_enter) – перевизначає відповідну функцію батьківського класу.
* override public void S\_DG\_Min(DataGridView dg\_enter) – перевизначає відповідну функцію батьківського класу.
* override public void SetH(DataGridView dg\_enter, bool flag) – перевизначає відповідну функцію батьківського класу.

1. Клас class LevelPrincipleForTi : LevelPrinciple є наслідником класу LevelPrinciple . Реалізує алгоритм, оснований на модифікованому рівневому принципі.

Містить наступні функції - члени класу:

* public LevelPrincipleForTi() : base() – конструктор класу.
* protected override List<List<int>> FindS(List<List<int>> originalA, int flag) – перевизначає відповідну функцію батьківського класу.
* override public void Build() – перевизначає відповідну функцію батьківського класу.
* int findBestFitEasy(List<int> S0, List<List<int>> Rez) – знаходить найкращі вершини на поточне місце.
* int findAny(List<int> V) – повертає будь-який елемент з переданого списку.
* List<int> fromPreviousLevel(List<List<int>> Rez) – повертає всі вершини з попереднього рівня, які все ще існують.
* protected override List<List<int>> Remove(List<int> index) – перевизначає відповідну функцію батьківського класу.
* List<List<int>> FindMu(List<List<int>> A, int l0) – знаходить допістимі місця вершини в упорядкуванні.
* override public void Draw(Graphics g) – перевизначає відповідну функцію батьківського класу.

1. Клас class Decomposition : LevelPrinciple є наслідником класу LevelPrinciple та реалізує метод, оснований на декомпозиції.

Містить наступні функції - члени класу:

* public Decomposition() : base() – конструктор класу.
* void getAnySpanningTree() – будує остовне дерево.
* int LevelOfV(List<List<int>> Rez, int v) – визначає рівень вершини.
* List<Point> getAllConflicts() – повертає список всіх конфліктних пар.
* int canSwitchPrev(int currentLevel) – визначає, чи може переставити вершину, яка передана в аргументі, на попередній рівень.
* bool hasPreOnLevel(int v, int currentLevel) – визначає, чи є у вершини попередники на попередньому рівні.
* int canSwitchNext(int currentLevel) – визначає, чи може переставити вершину, яка передана в аргументі, на наступний рівень.
* void SwitchVs(int v1, int v2) – переставляє вершини місцями в упорядкуванні.
* List<List<int>> getCurrentMatrix(int currentLevel) – повертає поточну матрицю суміжності.
* override public void Build() – перевизначає відповідну функцію батьківського класу.
* void AddVInroFreePlaces(int currentLevel) – додає вершини на вільні місця.

1. Клас class InterruptCondition : LevelPrincipleForTi є наслідником класу LevelPrincipleForTi та реалізує перевірку умов доцільності переривань.

* public InterruptCondition() : base() – конструктор класу.
* void generateCombinations(int summaryDuration, List<int> vs) – генерує комбінації заданої довжини з заданих вершин.
* int getSummaryDuration(List<int> list) – повертає сумарний час виконання завдань.
* public static List<List<int>> GetPowerSet(List<int> list) – генерує Power Set з заданого списку об’єктів.
* List<int> getPossibleVs(int mUpIPlusOne) – повертає спискок вершин, які потенційно можуть скласти шукану підмножину графу.
* int getmUpStart(int v) – повертає .
* int getmDownStart(int v) – повертає .
* int getmUpFinish(int v) – повертає .
* void checkFirstPlace(int mUpIPlusOne) – перевіряє умови на початку підмножини.
* List<int> getMaxFinishTime(List<int> vs) – знаходить вершини, які мають максимальний час завершення обслуговування серед заданих.
* int existR(int I, List<int> maxFinishTime, List<int> vs\_) – перевіряє наявність вершини з умов твердження.
* void checkLastPlace(int I) – перевіряє умови наприкінці підграфу.
* public bool canInterrupt() – перевіряє доцільність переривань.

1. Логіку роботи користувача реалізовано у класі Form1.

У класі містяться наступні функції:

* private void Form1\_Load(object sender, EventArgs e) - функція, що викликається при завантаженні форми. Встановлює бажані 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 button1\_Click(object sender, EventArgs e) – функція, яка викликається при натисненні кнопки з написом «Модифікований рівневий принцип». Реалізує виклик відповідних функції з класу LevelPrincipleForTi. Знаходить впорядкування з мінімальною та максимальною довжиною та виводить їх на екран.
* private void button2\_Click(object sender, EventArgs e) – функція, яка викликається при натисненні кнопки з написом «Метод декомпозиції». Реалізує виклик відповідних функції з класу Decomposition. Знаходить впорядкування з мінімальною та максимальною довжиною та виводить їх на екран.
* private void button3\_Click(object sender, EventArgs e) – функція, яка викликається при натисненні кнопки з написом «Перевірка доцільності переривань». Реалізує виклик відповідних функції з класу InterruptCondition. Знаходить впорядкування та виводить на екран, чи варто дозволяти переривання.
* 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 програм була реалізація методу, заснованого на модифікації рівневого принципу, методу декомпозиції для розв’язання узагальненої задачі паралельного впорядкування, а також перевірки умов доцільності переривань.

Оскільки рівневий принцип, який в тому чи іншому вигляді зустрічається в перших двох методах, передбачає довільний вибір вершин, то функція побудови конкретного впорядкування Build() запускається разів в функції FindMaxMin(int n). При достатньо великому значенні 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, і реалізовані методи працюють майже миттєво, то навіть при великих розрахунки проводяться дуже швидко. Це дозволяє проводити більш ґрутовний аналіз результатів, не спираючись на ймовірнісну складову рівневого принципу.

## Керівництво користувача

## Результати тестування програми

## Аналіз отриманих в програмі даних

# Висновки

В процесі виконання дипломної роботи було:

* модифіковано алгоритм, заснований на рівневому принципі, який був розроблений для задачі у класичній постановці для випадку, коли граф обмежень – дерево. Таким чином вдалося не лише розв’язувати узагальнені задачі за поліноміальний час, а і мінімізувати кількість переривань в процесі виконанні завдань;
* запропоновано наближений метод декомпозиції для розв’язання задач для випадку, коли граф обмежень довільний. Метод оснований на виділенні остовного дерева, отриманні оптимального впорядкування для нього, а потім модифікації впорядкування таким чином, аби упорядкування стало допустимим і при цьому як можна менше відхилялося від оптимального;
* сформульовано да доведено твердження, яке дозволяє виділити класи задач, оптимальні розв’язки яких належать до класу впорядкувань з перериваннями. Доведення твердження конструктивне, воно дає можливість в процесі перевірки умов відразу визначити, яку роботу на користь якої варто переривати;
* розроблено програмний продукт, який реалізує зазначені підходи та методи;
* результати, що стосуються задач теорії розкладів та упорядкувань з перериваннями, опубліковано в [5,6,8].

# Список літератури

1. 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.
2. Коффман Э.Г. Введение в теорию расписаний / Э.Г. Коффман. – М., «НАУКА», 1984. – 333 с.
3. 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.
4. Бурдюк В.Я. Алгоритмы параллельного упорядочения / В.Я. Бурдюк, В.А. Турчина. – Днепропетровск,- 1985. – 84 с.
5. Турчина В.А. Умови оптимальності для задачі Джонсона з перериваннями / В.А. Турчина, Ю.С. Зозуля // Сімнадцятий Міжнародний науково-практичний семінар Комбінаторні конфігурації та їх застосування. Кіровоград, 17–18 квітня 2015 р.; Кіровоград, 2015. – С. 99 – 105.
6. Зозуля Ю.С. Один випадок узагальнення задачі Джонсона // Тези доповідей ІІІ Міжнародного форуму студентів, аспірантів і молодих учених. Дніпропетровськ, 23-24 квітня 2015 р.; Дніпропетровськ, 2015. – С. 492 – 494.
7. Кістякове дерево [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D1%96%D1%81%D1%82%D1%8F%D0%BA%D0%BE%D0%B2%D0%B5\_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
8. Турчина В.А. Паралельні впорядкування з перериваннями / В.А. Турчина, Ю.С. Зозуля // ХІІІ міжнародна науково-практична конференція Математичне та програмне забезпечення інтелектуальних систем (MPZIS-2015). Дніпропетровськ, 18 – 20 листопада 2015 р.; Дніпропетровськ, 2015. – С. 237 – 238.