ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ОЛЕСЯ ГОНЧАРА

Факультет прикладної математики

Кафедра обчислювальної математики та математичної кібернетики

КУРСОВА РОБОТА

за спеціальністю

на тему «Алгоритми розрахунку характеристик в мережі PERT»

Другий (магістерський) рівень вищої освіти

Спеціальність 124 Системний аналіз

Освітня програма «Системний аналіз»

Виконавець

студентка групи ПС-21м-1

Маркова Анастасія Олександрівна

Керівник

кандидат ф.-м. наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. А. Турчина

Кількість балів

Оцінка за національною шкалою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії: \_\_\_\_\_\_\_\_\_\_\_\_В. А. Турчина

\_\_\_\_\_\_\_\_\_\_\_\_О. М. Притоманова

\_\_\_\_\_\_\_\_\_\_\_\_В. Л. Волошко

2022

РЕФЕРАТ

Курсова робота: 36 с., 20 рис., 15 табл., 9 джерел, 2 додатки.

Об’єкт дослідження: мережі PERT та їх характеристики.

Мета роботи: ознайомлення з задачами у мережевих постановках, застосування алгоритму переходу від довільного графу до мережі, розрахунок характеристик PERT мережі.

Одержані висновки та їх новизна: розглянуті алгоритми програмно реалізовані та протестовані на графах, які як не потребують перетворень, так і потребують перетворень.

Результати досліджень можуть бути застосовані при розрахунку характеристик реальних мереж, які класифікуються як мережі PERT.

Перелік ключових слів: МЕРЕЖІ PERT, КРИТИЧНИЙ ШЛЯХ, МОМЕНТИ НАСТАННЯ ПОДІЙ, ЗАТРИМКИ ВИКОНАННЯ.

ЗМІСТ

[ВСТУП 4](#_Toc106638708)

[ПОСТАНОВКА ЗАДАЧІ 5](#_Toc106638709)

[1 ПРИКЛАДНІ ЗАДАЧІ, ЩО ЗВОДЯТЬСЯ ДО ЗАДАЧ В МЕРЕЖЕВИХ ПОСТАНОВКАХ 6](#_Toc106638710)

[2 ПОНЯТТЯ МЕРЕЖІ. ПЕРЕТВОРЕННЯ ГРАФІВ НА МЕРЕЖІ 9](#_Toc106638711)

[2.1 Основні означення 9](#_Toc106638712)

[2.2 Алгоритм побудови мережі для незваженого графа 10](#_Toc106638713)

[2.3 Алгоритм побудови мережі для зваженого графа 17](#_Toc106638714)

[3 РОЗРАХУНОК ХАРАКТЕРИСТИК В МЕРЕЖІ PERT 25](#_Toc106638715)

[3.1 Алгоритм знаходження критичного шляху 25](#_Toc106638716)

[3.2 Алгоритми знаходження моментів настання подій та їх допустимих затримок 26](#_Toc106638717)

[4 ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОЗРАХУНКУ ХАРАКТЕРИСТИК МЕРЕЖІ 28](#_Toc106638718)

[4.1 Опис структури програмного продукту 28](#_Toc106638719)

[4.2 Опис основних модулів програми 28](#_Toc106638720)

[4.3 Опис інтерфейсу користувача 30](#_Toc106638721)

[4.4 Результати тестування 33](#_Toc106638722)

[4.5 Інструкція користувачеві 34](#_Toc106638723)

[ВИСНОВКИ 36](#_Toc106638724)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 37](#_Toc106638725)

[ДОДАТОК А 38](#_Toc106638726)

[ДОДАТОК Б 53](#_Toc106638727)

# ВСТУП

Серед оптимізаційних задач на графах особливе місце займають так звані задачі в мережевих постановках. Серед них є своя класифікація, яка відображає специфіку задач відповідного класу. Так, відома транспортна задача може моделюватися як в сітковій постановці, яка зводиться легкими перетвореннями до класичного поняття мережі. Добре вивченою задачею є задача про максимальний потік, яка формулюється в мережевій постановці. Ще однією із постановок задач мережевого типу є задачі календарного планування.

В даній роботі розглядається одна із специфічних мереж, що дістала назву PERT мережі, і відповідає техніці перегляду оцінки програми. Такі мережі використовуються перш за все при оптимізації реалізації алгоритмів, при розрахунку деяких важливих характеристик технологічних процесів на виробництві та для оцінки ефективності проектів.

Основними характеристиками таких мереж є мінімально можливий час завершення проекту, моменти настання подій, допустимі затримки при виконанні окремих складових проекту.

Розрахунок таких характеристик є актуальною задачею, оскільки попередній аналіз проектів дозволяє робити висновки про доцільність використання саме таких проектів, прогнозувати можливі діапазони змін в часові для окремих етапів проекту.

Об’єкт дослідження: мережі PERT та їх характеристики.

Мета роботи: ознайомлення з задачами у мережевих постановках, застосування алгоритму переходу від довільного графу до мережі, розрахунок характеристик PERT мережі.

# ПОСТАНОВКА ЗАДАЧІ

Задано довільний зважений по ребрах орієнтований граф G=(V, U).

Він може бути:

* мережею;
* довільним графом, що не містить циклів;
* довільним графом, що містить цикли.

Для останніх двох випадків необхідно побудувати мережу спеціального вигляду, що отримала назву PERT мережі.

Для кожного із випадків необхідно розрахувати основні характеристики мережі, такі як:

* мінімально можливий час завершення проекту;
* моменти настання подій;
* допустимі затримки при виконанні окремих складових проекту;

та програмно реалізувати перетворенні графів на мережу і розрахунку характеристик в мережі.

# ПРИКЛАДНІ ЗАДАЧІ, ЩО ЗВОДЯТЬСЯ ДО ЗАДАЧ В МЕРЕЖЕВИХ ПОСТАНОВКАХ

Оптимізаційні задачі на графах є одним із підкласів задач дискретної оптимізації. Вони представляють як теоретичний, так і практичний інтерес, оскільки існує багато прикладних сфер, моделювання процесів у яких зводиться саме до таких задач. В деяких випадках графи представляються як мережі спеціального вигляду.

Однією із класичних задач в мережевій постановці є задача про знаходження мінімального шляху в мережі [1]. В мережевій постановці формулюється також транспортна задача. Широке застосування має задача про максимальний потік в мережі, яка ще називається задачею про оптимальний потік.

До мережевих постановок може бути зведена задача про керування трафіком в мережі Інтернет [2]. Все більш актуальними стають проблеми зависання відеостримінгу, повільна робота додатків, сайтів, тощо. Вони можуть виникнути через проблеми з обладнанням або через непередбачені ситуації, такі як скачки напруги, порушення безпеки тощо. До проблем, які найчастіше виникають в мережах, відносять наступні:

* Низька пропускна спроможність (максимальний об’єм даних, що проходить через мережу).
* Високе навантаження на центральний процесор (виникає, коли через мережу проходить занадто багато трафіка).
* Проблеми з фізичним підключенням.
* Несправність пристроїв чи обладнання.
* Проблеми з системою доменних імен, коли втрачено доступ до мережі.
* Перешкоди в бездротовій мережі.

Перші дві задачі вирішуються за допомогою керування потоками даних, їх розпаралелення задля зменшення напруги на мережу та для того, щоб перешкодити виникнення інших перелічених проблем.

Також може виникнути проблема керування на виробництві незалежно від того, потрібно внести зміни в існуючу систему або замінити її на нову, якщо, наприклад, треба запровадити введення нової техніки або технологій. Підготовка плану по виконанню робіт, визначення їх об’єму та перевірка якості виконання цих дій в ході роботи – для вирішення цих задач використовується мережевий метод планування й управління. Він застосовний як до окремих задач, які виникають в процесі планування на виробництві, так і до всього процесу в цілому.

Розв’язки мережевих задач допомагають оптимізувати перевезення грузів між підприємствами, розміщення цих підприємств в межах певної області, забезпечити найбільш ефективне використання матеріальних ресурсів та часу, тощо. Аналіз моделей мережевого планування допомагає мінімізувати строки виконання виробничих задач за даними обмеженнями на ресурси та їх вартість, а також вирішити двоїсті до неї задачі (мінімізувати використання ресурсів при заданих строках роботи і вартості ресурсів або мінімізувати вартість виробництва за наявних строків завершення робіт та заданих обмеженнях на ресурси).

Метод мережевого планування може складатися з трьох рівнів планування проекту [3]. На першому рівні йде визначення логічної послідовності виконання робіт та їх об’єму, на другому – тривалості робіт, а на третьому йде з’ясування місця знаходження роботи в мережі, тобто зв’язків між даною та іншими роботами. Отриманий результат аналізу планування зображають у графічному вигляді – загальній схемі виробництва. В ході проекту може виконуватись корегування плану в залежності від змін, що вносяться в робочий процес, або зовнішніх факторів випадкової природи. Саме тому метод мережевого планування може (та для наближеного для реальності планування робіт повинен) бути ітеративним.

Методи мережевого планування та керування сьогодні застосовуються в широкому колі практичних задач [4]:

* календарне планування;
* прогнозування та попередження можливих проблем в ході виконання робіт;
* поліпшення роботи всього процесу завдяки перерозподіленню задач та сфер відповідальності між різними рівнями в ієрархії керівників та виконавців;
* керування ресурсами (часом, виконавцями, обладнанням, фінансовими) з метою оптимізації їх використання.

Наприклад, за допомогою мережі можна розв’язати наступну задачу:

Потрібно знайти очікуваний час виконання проекту, інтервал, в межах якого проект буде виконано з деякою долею ймовірності (), ймовірність, при якій проект буде завершено не пізніше заданого строку та максимально можливий час виконання проекту, при заданій надійності (). Розрахунки провести для конкретного графу зі зваженими дугами.

# ПОНЯТТЯ МЕРЕЖІ. ПЕРЕТВОРЕННЯ ГРАФІВ НА МЕРЕЖІ

## 2.1 Основні означення

Означення 1. Мережа – це ациклічний орграф, в якого є:

* не більше ніж одна початкова вершина (вершина, що не має вхідних дуг), з якої досяжна будь-яка інша вершина;
* не більше ніж одна кінцева вершина (вершина, що не має вихідних дуг), що є досяжною для будь-якої іншої вершини [5].

На рис. 1 зображено приклад мережі. Граф на рис. 2 не є мережею, бо в ньому є три початкової вершини (1 та 2), дві кінцеві (5 та 12) та цикл (вершини 7, 9, 10, 11).

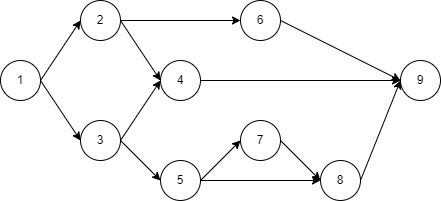


Рисунок 1 – Приклад мережі.

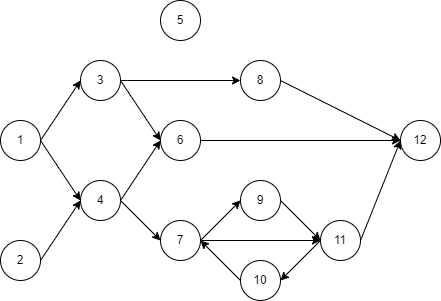


Рисунок 2 – Приклад графу, що не є мережею.

В загальному випадку при переході від графів до мереж виникає необхідність використовувати сильно зв’язані компоненти орграфа. Тому попередньо наведемо відповідне означення:

Означення 2. Сильно зв’язною компонентою орграфа називають підграф, верши якого утворюють цикл.

Розрізняючи випадки, коли граф є мережею, а коли ні, наведемо алгоритм переходу від незваженого довільного графа до мережі (алгоритм 1) та алгоритм переходу від зваженого довільного графа до мережі (алгоритм 2). Ці алгоритми складаються з двох етапів. На першому етапі виділяються сильно зв’язні компоненти орграфа. На другому етапі будується мережа.

## 2.2 Алгоритм побудови мережі для незваженого графа

Для заданого незваженого графа на першому етапі визначимо сильно зв’язні компоненти та видалимо їх. На другому етапі визначимо або створимо джерело та стік, щоб отримати мережу.

Етап 1.

1. Знайти сильно зв’язну компоненту в даному графі одним із відомих алгоритмів (наприклад, за алгоритмом Косарайю [6], алгоритмом пошуку компонент сильної зв’язності з двома стеками або алгоритмом Тар’яна). Якщо таких немає, то перейти до 2 етапу алгоритму.
2. Стягнути вершини знайденої на 1 кроці компоненти в одну вершину. Для цього треба виконати наступні кроки.
   1. Додати до графу нову вершину.
   2. Провести в неї дуги з тих вершин, з яких ідуть дуги до вершин компоненти, що вилучається.
   3. Провести з цієї вершини дуги до тих вершин, в які входять дуги з компоненти, що вилучається.
3. Видалити з графа вершини, що входять у визначену на 1 кроці компоненту, та всі інцидентні до них дуги. Перейти до кроку 1.

Етап 2.

1. Визначити в графі вершини, в яких немає вхідних дуг. Якщо така вершина одна, то крок 6. Якщо таких вершин декілька, то крок 5.
2. Додати до даного графу вершину та провести з неї дуги до вершин, визначених на кроці 4.
3. Визначити в графі вершини, в яких немає вихідних дуг. Якщо така вершина одна, то кінець, мережа побудована. Якщо таких вершин декілька, то крок 7.
4. Додати до даного графу вершину та провести до неї дуги з вершин, визначених на кроці 6. Кінець алгоритму. Мережа побудована.

Псевдокод алгоритму 1:

START:

V - множина вершин графу G

U - множина дуг графу G

n = кількість вершин в графі G

WHILE Function: Знайти кількість вершин в сильно зв’язній компоненті >1

n:=n+1

Function: Додати вершину з номером n до множини V

V:=V

V\_before - множина вершин, з яких дуги прямують до вершин компоненти

Function: Провести дуги з кожної вершини множини V\_before до вершини з номером n та додати їх до множини U

U:=U

V\_after - множина вершин, в які дуги входять з вершин компоненти

Function: Провести дуги з вершини з номером n до кожної вершини множини V\_after та додати їх до множини U

U:=U

Function: Видалити зв’язну компоненту з G

ENDWHILE

V\_first - множина вершин, що не мають вхідних дуг

IF THEN

n:=n+1

Function: Додати вершину з номером n до множини V

V:=V

Function: Провести дуги з вершини з номером n до кожної вершини множини V\_first та додати їх до множини U

U:=U

ENDIF

V\_last - множина вершин, що не мають вихідних дуг

IF THEN

n:=n+1

Function: Додати вершину з номером n до множини V

V:=V

Function: Провести дуги з кожної вершини множини V\_last до вершини з номером n та додати їх до множини U

U:=U

ENDIF

END

Проілюструємо роботу першого алгоритму на наступному прикладі. Нехай задано незважений граф (рис. 3).

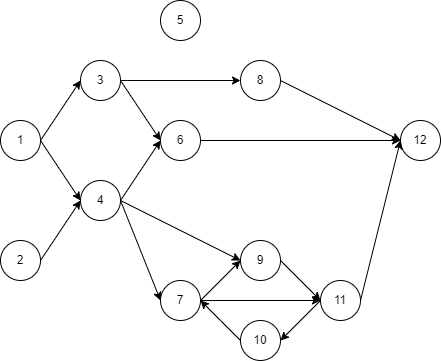


Рисунок 3 – Приклад роботи алгоритму 1. Крок 1.

1. Знаходимо сильну зв’язну компоненту в даному графі. Її утворюють вершини 7, 9, 11, 10.
2. Стягуємо знайдену на першому кроці компоненту в одну вершину. Для цього додаємо до даного графу нову вершину (вона матиме номер 13). Проводимо до вершини 13 дуги з вершин, що прямували до компоненти. Це дуги (4, 7) та (4, 9). Так як вони виходять з однієї вершини, то нова дуга буде одна – (4, 13). Далі проводимо дуги з нової вершини до вершин, в які дуги входили, прямуючи від компоненти. Це одна дуга (11, 12). Нова дуга буде (13, 12). Результат показано на рис. 4.

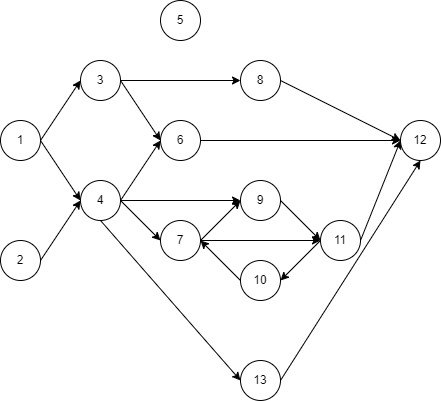


Рисунок 4 – Приклад роботи алгоритму 1. Крок 2.

1. Видаляємо з графу компоненту, отриману на кроці 1 та всі інцидентні до неї дуги. Отримаємо наступний граф (рис. 5).

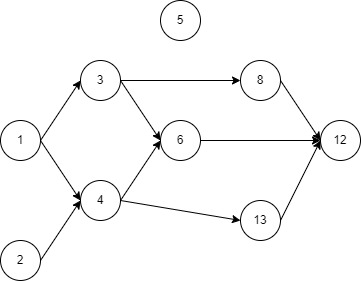


Рисунок 5 – Приклад роботи алгоритму 1. Крок 3.

Переходимо на крок 1.

Крок 1. Ітерація 2. Перевіряємо наявність сильно зв’язної компоненти в даному графі (рис. 5). Її немає, отже переходимо до 2 етапу алгоритму.

Етап 2.

1. Визначаємо в графі вершини, в яких немає вхідних дуг. Таких вершин три: 1, 2 та 5. Отже переходимо на крок 5.
2. Додаємо до даного графу нову вершину (її номер буде 14). Проводимо з вершини 14 дуги до вершин, визначених на кроці 4 (це вершини 1, 2 та 5). Результат зображено на рис. 6.

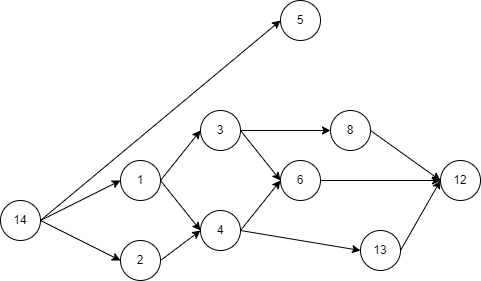


Рисунок 6 – Приклад роботи алгоритму 1. Крок 5.

1. Визначаємо в графі вершини, в яких немає вихідних дуг. Це вершини 5 та 12. Їх дві, отже переходимо до кроку 7.
2. Додаємо до даного графу нову вершину (її номер буде 15). Проводимо до вершини 15 дуги з вершин, визначених на кроці 6 (це вершини 5 та 12). Кінець алгоритму. Побудована мережа показана на рис. 7.

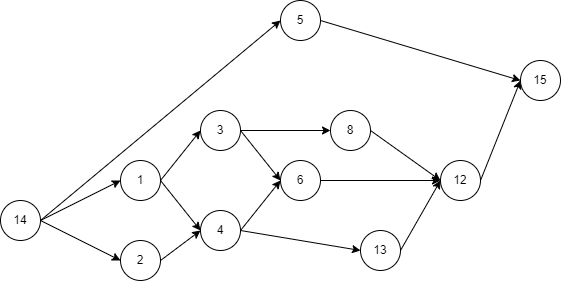


Рисунок 7 – Приклад роботи алгоритму 1. Крок 7.

## 2.3 Алгоритм побудови мережі для зваженого графа

Для заданого зваженого графа на першому етапі визначимо сильно зв’язні компоненти та видалимо їх. На другому етапі визначимо або створимо джерело та стік, щоб отримати мережу.

Етап 1.

1. Знайти сильно зв’язну компоненту в даному графі. Якщо таких немає, то перейти до 2 етапу алгоритму.
2. Стягнути вершини знайденої на 1 кроці компоненти в одну вершину. Для цього треба виконати наступні кроки.
   1. Додати до графу нову вершину та присвоїти їй вагу, що є сумою ваг вершин компоненти (або найбільше значення з ваг вершин в залежності від конкретної задачі).
   2. Провести до нової вершини дуги з вершин, з яких дуги виходили і прямували до компоненти, та присвоїти їм відповідні нові ваги.
   3. Провести з нової вершини дуги до вершин, в які дуги входили, прямуючи з компоненти та присвоїти їм нові відповідні ваги.
3. Видалити з графа вершини, що входять у визначену на 1 кроці компоненту, та всі інцидентні до них дуги. Перейти до кроку 1.

Етап 2.

1. Визначити в графі вершини, в яких немає вхідних дуг. Якщо така вершина одна, то крок 6. Якщо таких вершин декілька, то крок 5.
2. Додати до даного графу вершину та присвоїти їй вагу 0 (або в залежності від конкретної задачі). Провести з нової вершини дуги до вершин, визначених на кроці 4 та присвоїти їм відповідну вагу.
3. Визначити в графі вершини, в яких немає вихідних дуг. Якщо така вершина одна, то кінець, мережа побудована. Якщо таких вершин декілька, то крок 7.
4. Додати до даного графу вершину та присвоїти їй відповідну вагу. Провести до нової вершини дуги з вершин, визначених на кроці 6 та присвоїти їм відповідну вагу. Кінець алгоритму. Мережа побудована.

Псевдокод алгоритму 2:

START:

V = множина вершин графу G

U = множина дуг графу G

n = кількість вершин в графі G

WHILE Function: Знайти кількість вершин в сильно зв’язній компоненті >1

n:=n+1

Function: Додати вершину з номером n до множини V

Function: Присвоїти вершині з номером n вагу 0

V:=V

V\_before - множина вершин, з яких дуги прямують до вершин компоненти

Function: Провести дуги з кожної вершини множини V\_before до вершини з номером n та додати їх до множини U

Function: Присвоїти кожній з проведених до вершини з номером n дуг вагу, що є сумою ваг відповідних дуг, що прямували з відповідної вершини множини V\_before до вершин компоненти

U:=U

V\_after - множина вершин, в які дуги входять з вершин компоненти

Function: Провести дуги з вершини з номером n до кожної з вершин множини V\_after та додати їх до множини U

Function: Присвоїти кожній з дуг, що проведені з вершини з номером n, вагу, що є сумою ваг відповідних дуг, що прямували з вершин компоненти до відповідної вершини множини V\_after

U:=U

Function: Видалити зв’язну компоненту з G

ENDWHILE

V\_first - множина вершин, що не мають вхідних дуг

IF THEN

n:=n+1

Function: Додати вершину з номером n до множини V

Function: Присвоїти вершині з номером n вагу 0

V:=V

Function: Провести дуги з вершини з номером n до кожної вершини множини V\_first та додати їх до множини U

U:=U

Function: Присвоїти кожній з дуг, що виходять з вершини з номером n, вагу 0

ENDIF

V\_last - множина вершин, що не мають вихідних дуг

IF THEN

n:=n+1

Function: Додати вершину з номером n до множини V

Function: Присвоїти вершині з номером n вагу 0

V:=V

Function: Провести дуги з кожної вершини множини V\_last до вершини з номером n та додати їх до множини U

U:=U

Function: Присвоїти кожній з дуг, що входять в вершину з номером n, вагу 0

ENDIF

END

Проілюструємо роботу другого алгоритму на наступному прикладі. Нехай задано зважений граф (рис. 8).

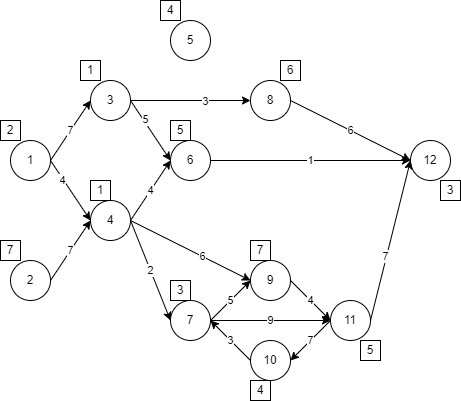


Рисунок 8 – Приклад роботи алгоритму 2. Крок 1.

1. Знаходимо сильну зв’язну компоненту в даному графі. Її утворюють вершини 7, 9, 11, 10.
2. Стягуємо знайдену на першому кроці компоненту в одну вершину. Для цього додаємо до даного графу нову вершину (вона матиме номер 13) та присвоюємо їй вагу, що є сумою ваг вершин компоненти (відповідно, вага буде дорівнювати 19). Проводимо до вершини 13 дуги з вершин, що прямували до компоненти. Це дуги (4, 7) та (4, 9). Так як вони виходять з однієї вершини, то нова дуга буде одна – (4, 13), її вага буде сумою ваг відповідних дуг, тобто 8. Далі проводимо дуги з нової вершини до вершин, в які дуги входили, прямуючи від компоненти. Це одна дуга (11, 12). Нова дуга буде (13, 12) та матиме вагу 7. Результат показано на рис. 9.

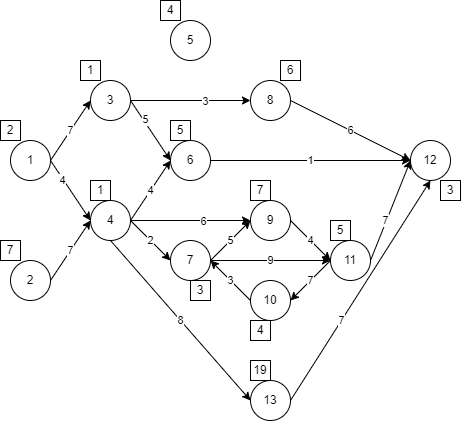


Рисунок 9 – Приклад роботи алгоритму 2. Крок 2.

1. Видаляємо з графу компоненту, отриману на кроці 1 та всі інцидентні до неї дуги. Отримаємо наступний граф (рис. 10).

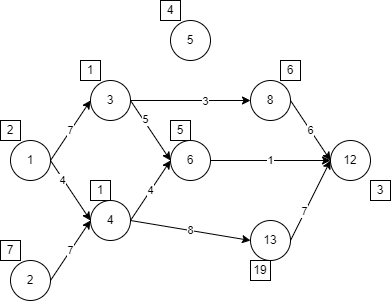


Рисунок 10 – Приклад роботи алгоритму 2. Крок 3.

Переходимо на крок 1.

Крок 1. Ітерація 2. Перевіряємо наявність сильно зв’язної компоненти в даному графі (рис. 10). Її немає, отже переходимо до 2 етапу алгоритму.

Етап 2.

1. Визначаємо в графі вершини, в яких немає вхідних дуг. Таких вершин три: 1, 2 та 5. Отже переходимо на крок 5.
2. Додаємо до даного графу нову вершину (її номер буде 14) та присвоюємо їй вагу 0. Проводимо з вершини 14 дуги до вершин, визначених на кроці 4 (це вершини 1, 2 та 5) та присвоюємо кожній з них вагу 0. Результат зображено на рисунку 11.

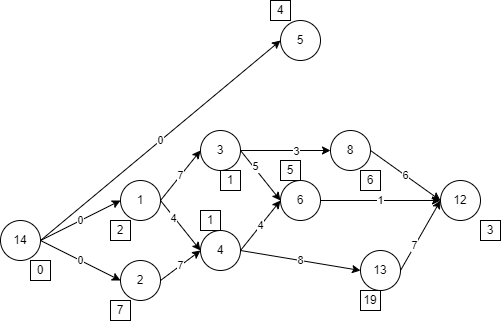


Рисунок 11 – Приклад роботи алгоритму 2. Крок 5.

1. Визначаємо в графі вершини, в яких немає вихідних дуг. Це вершини 5 та 12. Їх дві, отже переходимо до кроку 7.
2. Додаємо до даного графу нову вершину (її номер буде 15) та присвоюємо їй вагу 0. Проводимо до вершини 15 дуги з вершин, визначених на кроці 6 (це вершини 5 та 12) та присвоюємо кожній з них вагу 0. Кінець алгоритму. Побудована мережа показана на рис. 12.

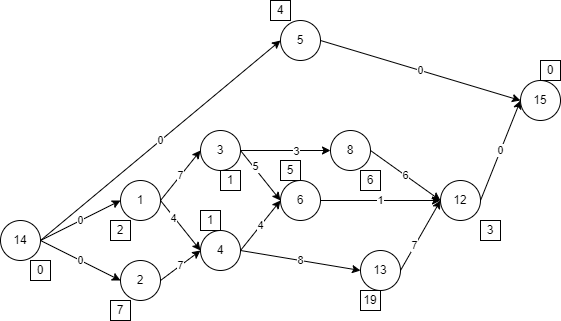


Рисунок 12 – Приклад роботи алгоритму 2. Крок 7.

# РОЗРАХУНОК ХАРАКТЕРИСТИК В МЕРЕЖІ PERT

PERT (Program Evaluation Review Technique) – мережа, яка графічно представляє хронологію проекту та дозволяє керувати проектом, розбиваючи його на окремі завдання для проведення оцінки і аналізу строків виконання роботи [8].

Існує три задачі, кожна з яких пов’язана з розрахунком окремої характеристики в мережі PERT.

Перша задача полягає у відшуканні довжини критичного шляху. Розв’язок цієї задачі допомагає визначити максимально можливий час, потрібний на виконання всього проекту, щоб уникнути затримки під час його реалізації. Для її розв’язання можна використовувати відомі алгоритми пошуку критичних шляхів в графах. Найбільш універсальним серед них є алгоритм Флойда-Воршола.

Другою задачею є пошук моменту настання кожної можливої події. На практиці це означає, що потрібно знайти час, до якого буде виконана конкретна задача.

І третя задача являє собою розрахунок допустимих затримок виконання кожної задачі, які не вплинуть на загальний час завершення проекту.

При використанні мережі PERT потрібно розрахувати найменший час роботи над проектом, найбільший час та найбільш вірогідний [7]. Очікуваний час завершення проекту може бути представлений, наприклад, середнім арифметичним цих трьох величин.

## 3.1 Алгоритм знаходження критичного шляху

Одна із основних характеристик PERT мережі, на якій базуються і інші це критичний шлях в графі та його довжина. Розглянемо один із найбільш універсальних алгоритмів знаходження критичних шляхів – алгоритм Флойда-Воршола для знаходження критичного шляху.

1. Для , ( – кількість вершин у графі ) створити масив distance[][] та ініціалізувати його як (за в даному алгоритмі можна взяти будь-яке додатне число).
2. Для ( – кількість дуг у графі ) записати у масив distance[][] значення ваги дуги з протилежним знаком.
3. Для записати у масив distance[][] значення 0.
4. Для , , якщо distance[][] > distance[][] + distance[][], distance[][] < та distance[][] < , то distance[][] = distance[][] + distance[][].
5. Для , distance[][] = (-1) \* distance[][].
6. Маємо матрицю всіх можливих шляхів із вершини () до вершини () через будь-яку вершину . Це матриця distance[][].
7. Запишемо до критичного шляху V\_crit .
8. Якщо V\_crit[1] = , то кінець. Інакше отримаємо з таблиці дуг значення та запишемо його в початок критичного шляху. Крок 8.

Таким чином, застосувавши алгоритм Флойда-Воршола, ми знаходимо критичний шлях V\_crit, а його довжина – distance[].

## 3.2 Алгоритми знаходження моментів настання подій та їх допустимих затримок

Для знаходження моменту настання кожної події, реалізуємо наступні пункти:

1. Застосувати алгоритм Флойда-Воршола. Зберегти матрицю відстаней, отриманий за цим алгоритмом.
2. Для вага distance[][] це шуканий момент настання кожної можливої події. Кінець.

Знаходження допустимих затримок подій здійснюється наступним чином:

1. Розв’язати задачу знаходження моментів настання подій. Зберегти дані зваженої мережі.
2. Для створити масив delay[] та записати в нього значення delay[] = distance[][] - distance[][] - weight(()).

# ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОЗРАХУНКУ ХАРАКТЕРИСТИК МЕРЕЖІ

## 4.1 Опис структури програмного продукту

Програма написана мовою програмування C++ [9]. Вона містить вихідний код файлу «courseWork.cpp» та виконуваний файл «courseWork.exe»

Вхідні дані задані у текстових файлах:

* вершини задані у файлі «courseWork\_V.txt» у вигляді стовпчику номерів вершин;
* дуги задані у файлі «courseWork\_U.txt». Вони мають вигляд стовпчику пар вершин, що з’єднані відповідними дугами. Пари задані двома числами, що розділені пробілом. Перше число відповідає номеру вершини, з якої виходить дуга, друге – номеру вершини, в яку дуга виходить;
* ваги дуг задані у файлі «courseWork\_UWeight.txt» у вигляді стовпчику значень ваг відповідних дуг з файлу, в якому задані дуги.

Вихідні дані генеруються після запуску додатку у текстові файли:

* мережа PERT міститься у файлі «network.txt». Вона представляється набором вершин та дуг, що утворюють мережу;
* характеристики мережі PERT містяться у файлі «PERT.txt». В ньому міститься критичний шлях у вигляді послідовності номерів вершин, довжина критичного шляху, моменти настання подій, де через дефіс записана конкретна подія (номер вершини) та значення моменту. Також показані допустимі затримки у вигляді пар дуг та відповідних затримок на цих дугах.

## 4.2 Опис основних модулів програми

В основній функції «main()» послідовно реалізовані два основні модуля програми.

Спочатку перевіряються умови, що заданий граф є мережею. Це відбувається у циклі, де у графі шукається сильно зв’язна компонента, та за допомогою двох перевірок на єдиність вершини, що не має вхідних дуг, та вершини, що не має вихідних дуг.

Після цього виконуються розрахунки на мережі PERT. Знаходиться матриця відстаней «distanceMatrix[][]», в якій ітераційно розраховуються відстані між кожною парою вершин у мережі. Критичний шлях записується до вектору «criticalPath», допустимі затримки – до масиву «delay[]», а моменти настання подій містяться всередені матриці відстаней. Потрібні вихідні дані записуються до відповідних файлів.

Функції «readingV(int &, int \*)», «readingU(int &, int \*\*)», «readingUWeights(int &, int \*)» зчитують вхідні дані з відповідних файлів «courseWork\_V.txt», «courseWork\_U.txt» та «courseWork\_UWeight.txt». Вершини та значення ваг дуг записуються до одномірних масивів, а дуги записуються до двомірного масиву.

Функції «countNumOfV()» та «countNumOfU()» підраховують відповідну кількість вершин та дуг з початкових даних та заносять значення до відповідних змінних.

Функція «stronglyConnecredComponent(int &, std::vector<int> &, int \*\*)» знаходить сильно зв’язну компоненту в даному графі та повертає в результаті роботи вектор, в якому містяться номера вершин, що її утворюють.

Функція «VFirst(int &, int \*\*)» знаходить множину вершин, що не мають вхідних дуг, а функція «VLast(int &, int \*\*)» знаходить множину вершин, що не мають вихідних дуг. Ці функції повертають вектори значень відповідних вершин.

Функція «reallocUp(int \*\*, int, int)» дозволяє збільшити розмір матриці, що подається на вхід. Ця функція використовується при перетворенні довільного орграфа на мережу, а саме при додаванні нової вершини (джерела або стоку). Функція «reallocDown(int \*\*, int, int)» зменшує розмір матриці та використовується при видаленні сильно зв’язної компоненти з початкового графу, якщо така є.

## 4.3 Опис інтерфейсу користувача

При роботі з програмою користувач повинен записати до відповідних файлів (рис. 13-15) за зразком свій довільний орієнтований граф.

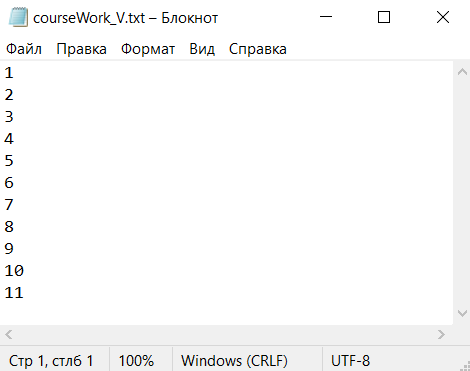


Рисунок 13 – Файл з вхідними даними «courseWork\_V.txt»

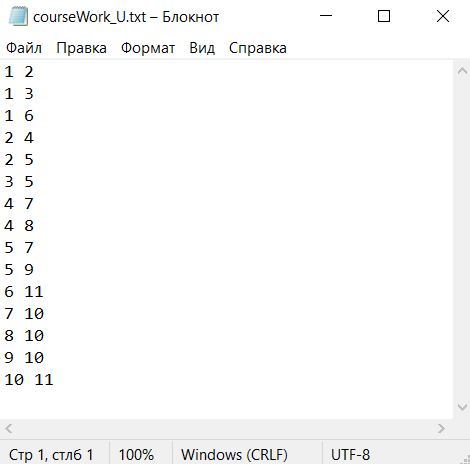


Рисунок 14 – Файл з вхідними даними «courseWork\_U.txt»

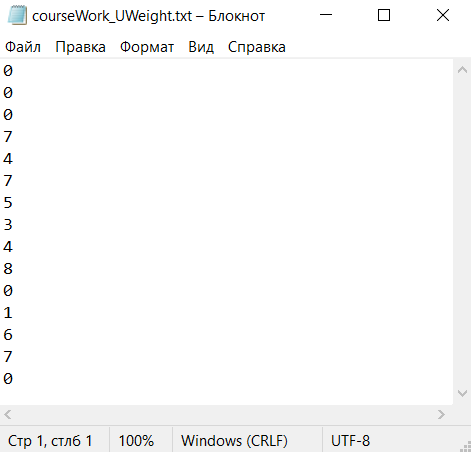


Рисунок 15 – Файл з вхідними даними «courseWork\_UWeight.txt»

Після занесення вхідних даних до файлів, користувач повинен зберегти їх та запустити додаток з розширенням «.exe». Програма перевірить чи є граф мережею та в разі необхідності проведе необхідні перетворення. В результаті ми отримаємо вихідний файл (рис. 16), який містить дані щодо мережі PERT.

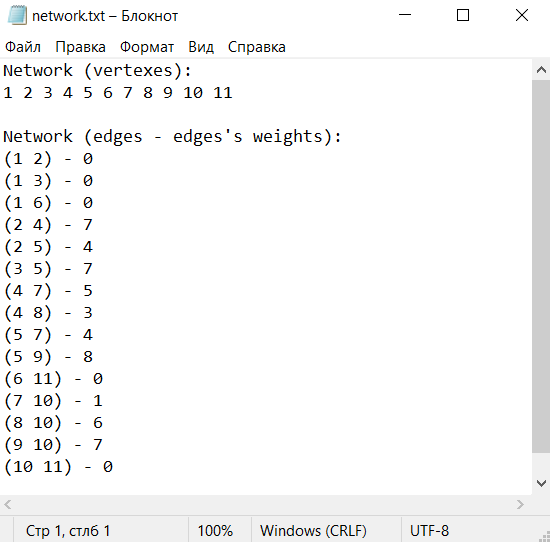


Рисунок 16 – Файл з вихідними даними «network.txt»

Також програма проведе розрахунки на вихідній мережі та представить їх у вигляді наступного файлу (рис. 17). Тут зображений критичний шлях у вигляді послідовності номерів вершин, через які він проходить, довжину цього шляху. Також тут представлено моменти настання подій у вигляді номера вершини та відповідного моменту, та затримки дуг у вигляді пар дуг та відповідного значення затримки.

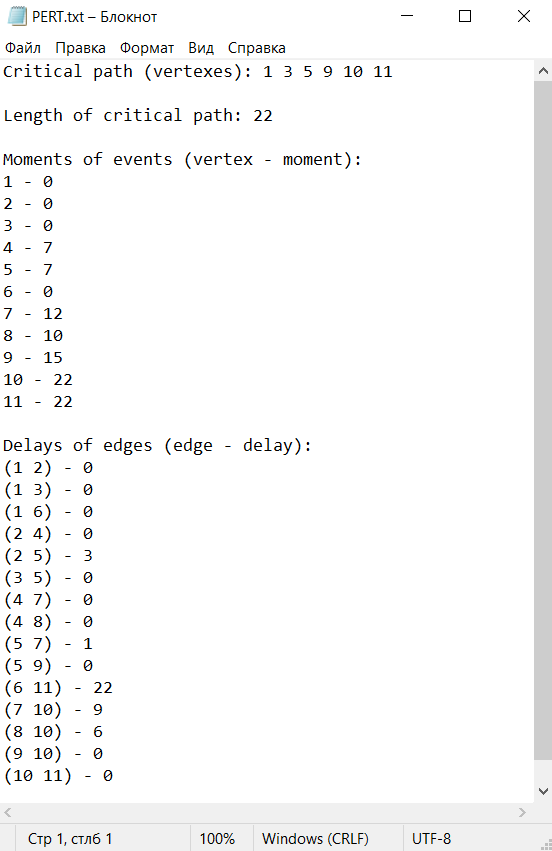


Рисунок 17 – Файл з вихідними даними «PERT.txt»

## 4.4 Результати тестування

Програма була протестована на прикладі, що зображено на рис. 18. Даний граф був заданий до програми у вигляді вершин, дуг та ваг відповідних дуг як на рис. 13-15.

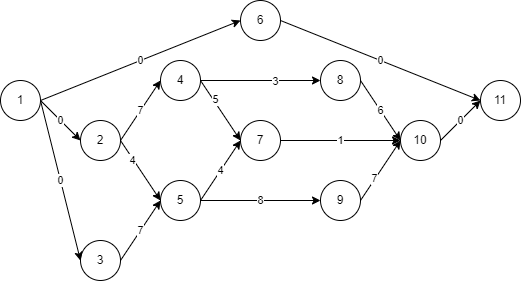


Рисунок 18 – Приклад для тестування програми

Так як граф з рис. 18 є мережею, то програма пропустить модуль перетворення графу на мережу та просто виведе її у вигляді рис. 19.

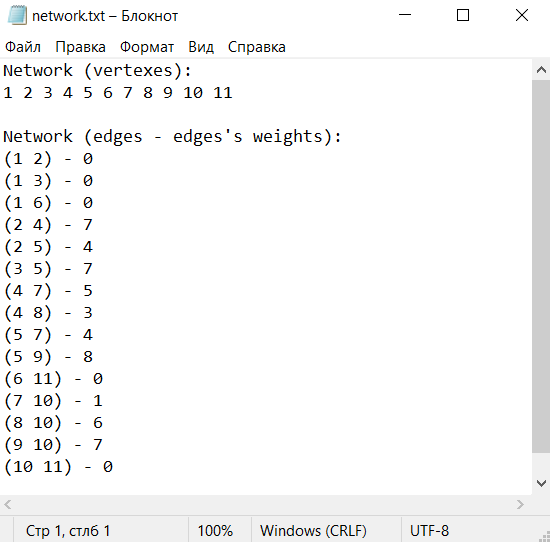


Рисунок 19 – Результат роботи програми – мережа PERT

В результаті також було отримано розрахунки на мережі PERT, що мають наступний вигляд (рис. 20):

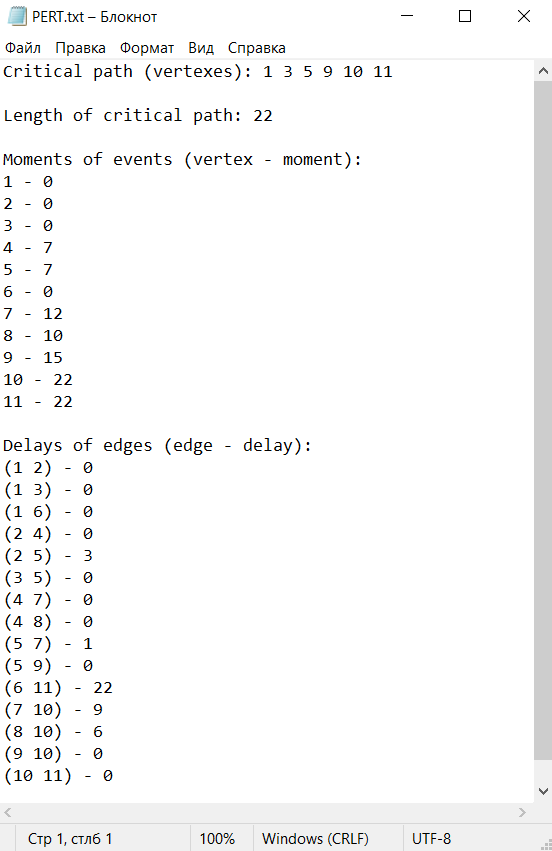


Рисунок 20 – Результат роботи програми – розрахунки на мережі PERT

## 4.5 Інструкція користувачеві

Користувач повинен дотримуватися наступних правил при роботі з програмою:

* всі номера вершин, що використовуються для запису вершин та дуг графу, повинні бути цілими додатними;
* ваги дуг повинні містити лише цілі невід’ємні значення в своєму запису;
* дуги та ваги дуг, що задаються в окремих файлах повинні по строках відповідати один одному;
* номери вершин графу задаються стовпчиком;
* дуги задаються стовпчиком, де кожне місце посідають два числа через знак пробілу. Це відповідні номера вершин.

# ВИСНОВКИ

В ході виконання курсової роботи було зроблено наступне:

* вивчення прикладних задачам, що зводяться до задач в мережевих постановках;
* розглянуто алгоритм перетворення зваженого графу на мережу та протестовано його на конкретному прикладі;
* вивчено основні характеристики мереж PERT та алгоритми їх розрахунку;
* розроблено програмний продукт, що реалізує алгоритм переходу від довільного графу до мережі та алгоритми, що знаходять характеристики мереж PERT.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaidehi Joshi. Finding the shortest path, with a little help from Dijkstra. Дата оновлення: 17.10.2017. URL: <https://medium.com/basecs/finding-the-shortest-path-with-a-little-help-from-dijkstra-613149fbdc8e> (дата звернення: 17.05.2022)
2. Alyssa Lamberti. 6 most common network problems. Дата оновлення: 2.08.2021. URL: <https://obkio.com/blog/common-network-problems/> (дата звернення: 17.05.2022)
3. Jeyad M. Baig. Planning methods – 3 levels of project planning. Дата оновлення: 28.02.2021. URL: <http://apppm.man.dtu.dk/index.php/Planning_Methods_-_3_Levels_of_Project_Planning> (дата звернення: 17.05.2022)
4. Плескунов М. А. Задачи сетевого планирования. Екатеринбург: Изд-во Урал. Ун-та. 2014. 92с.
5. Іглін С.П. Теорія графів. Харків: НТУ «ХПІ». 2017. 146с.
6. Тарануха В.Ю. Задачі на графах для курсу Алгоритміка. Київ: електронна публікація на сайті факультету. 2021. 72с.
7. Филин С. А. Сетевое планирование. 2021.URL: <http://upr-proektom.ru/setevoe-planirovanie> (дата звернення: 17.05.2022)
8. Carol M. Kopp. Program Evaluation Rewiev Technique (PERT) Chart. Дата оновлення: 17.03.2022. URL: <https://www.investopedia.com/terms/p/pert-chart.asp> (дата звернення: 17.05.2022)
9. C++ language documentation. Дата оновлення: 2022 р. URL: <https://docs.microsoft.com/en-us/cpp/cpp/?view=msvc-170> (дата звернення: 17.05.2022)

# ДОДАТОК А

Приклад розв’язку задач за алгоритмом Флойда-Воршола

Нехай задано мережу зі зваженими дугами (рис. 21). Для розв’язання задач 1-3 потрібно знати критичний шлях в графі та його довжину. Для його знаходження застосуємо алгоритм Флойда-Воршола. Проілюструємо покрокову роботу цього алгоритму для мережі (рис. 21).

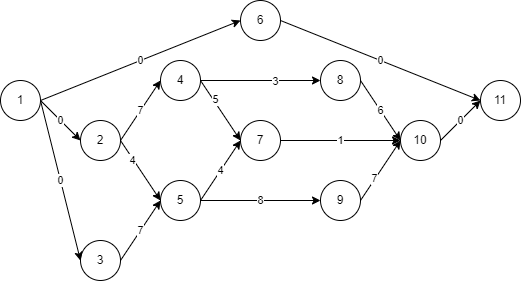


Рисунок 21 – Мережа для тестування алгоритму Флойда-Воршола

Крок 0. Для k=0 запишемо до табл. 1 значення ваг дуг для та значення 0 для . Позначимо їх синім кольором. До всіх інших клітин таблиці занесемо значення чорним кольором. Воно означає, що між двома різними вершинами не існує дуги.

Таблиця 1 – Крок 0

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=0 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** |  |  | **0** |  |  |  |  |  |
| 2 |  | **0** |  | **-7** | **-4** |  |  |  |  |  |  |
| 3 |  |  | **0** |  | **-7** |  |  |  |  |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Далі на кожному кроці для знаходимо відстань між вершинами та , для яких існує шлях через вершину . Якщо значення відстані менше, ніж в попередньому кроці, то переписуємо його до нової таблиці та виділяємо синім кольором.

Крок 1. Так як вершина не має вхідних дуг, то через неї не існує шляху між двома вершинами. Переписуємо значення з табл. 1 до табл. 2 без змін.

Таблиця 2 – Крок 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=1 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** |  |  | **0** |  |  |  |  |  |
| 2 |  | **0** |  | **-7** | **-4** |  |  |  |  |  |  |
| 3 |  |  | **0** |  | **-7** |  |  |  |  |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 2. Через вершину існує шлях з вершини до вершин та . Щоб розрахувати довжину цього шляху, потрібно з останньої таблиці (табл. 2) взяти значення відстані з вершини до та відповідні значення з до та з до та просумувати їх. Відстань між вершинами та дорівнює , а між та дорівнює . Так як та , то отримані значення заносимо до табл. 3 (виділені синім кольором).

Таблиця 3 – Крок 2

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=2 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-4** | **0** |  |  |  |  |  |
| 2 |  | **0** |  | **-7** | **-4** |  |  |  |  |  |  |
| 3 |  |  | **0** |  | **-7** |  |  |  |  |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 3. Через вершину існує шлях з до . Його довжина дорівнює . Так як ( – це значення довжини шляху між вершинами та на кроці 2), то переписуємо відстань між вершинами та (табл. 4).

Таблиця 4 – Крок 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=3 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** |  |  |  |  |  |
| 2 |  | **0** |  | **-7** | **-4** |  |  |  |  |  |  |
| 3 |  |  | **0** |  | **-7** |  |  |  |  |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 4. Через вершину існує шлях з вершини до вершини та з до . Також через вершину існують такі ж шляхи, тільки з вершини . Значення відстаней будемо розраховувати як суму значень відстаней:

* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто .

Порівнюємо отримані значення відстаней зі значеннями табл. 4. Всі вони менші, отже записуємо їх синім кольором замість попередніх (табл. 5).

Таблиця 5 – Крок 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=4 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** |  |  |  |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** |  |  |  |
| 3 |  |  | **0** |  | **-7** |  |  |  |  |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 5. Розрахуємо значення відстаней між вершинами, де існує шлях через вершину :

* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто .

Порівнюємо отримані значення відстаней зі значеннями табл. 5. Менші значення записуємо синім кольором замість попередніх значень (табл. 6).

Таблиця 6 – Крок 5

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=5 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** |  |  |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** |  |  |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 6. Через вершину існує єдиний шлях з вершини до . Він дорівнює 0, запишемо це значення до табл. 7 синім кольором.

Таблиця 7 – Крок 6

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=6 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** |  | **0** |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** |  |  |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** |  |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  |  |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** |  |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 7. Розрахуємо значення відстаней між вершинами, де існує шлях через вершину :

* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто .

Порівнюємо отримані значення відстаней зі значеннями табл. 7. Менші значення записуємо синім кольором замість попередніх значень (табл. 8).

Таблиця 8 – Крок 7

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=7 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** | **-13** | **0** |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** | **-13** |  |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** | **-12** |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  | **-6** |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** | **-5** |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 8. Розрахуємо значення відстаней між вершинами, де існує шлях через вершину :

* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто .

Порівнюємо отримані значення відстаней зі значеннями табл. 8. Менші значення записуємо синім кольором замість попередніх значень (табл. 9).

Таблиця 9 – Крок 8

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=8 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** | **-16** | **0** |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** | **-16** |  |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** | **-12** |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  | **-9** |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** | **-5** |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 9. Розрахуємо значення відстаней між вершинами, де існує шлях через вершину :

* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто .

Порівнюємо отримані значення відстаней зі значеннями табл. 9. Менші значення записуємо синім кольором замість попередніх значень (табл. 10).

Таблиця 10 – Крок 9

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=9 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** | **-22** | **0** |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** | **-19** |  |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** | **-22** |  |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  | **-9** |  |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** | **-15** |  |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** |  |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** |  |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** |  |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 10. Розрахуємо значення відстаней між вершинами, де існує шлях через вершину :

* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто ;
* з до та з до , тобто .

Порівнюємо отримані значення відстаней зі значеннями табл. 10. Менші значення записуємо синім кольором замість попередніх значень (табл. 11).

Таблиця 11 – Крок 10

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=10 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** | **-22** | **-22** |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** | **-19** | **-19** |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** | **-22** | **-22** |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  | **-9** | **-9** |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** | **-15** | **-15** |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** | **-1** |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** | **-6** |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** | **-7** |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 11. Так як через вершину не існує шляху між двома вершинами, то залишаємо значення відстаней без змін.

Таблиця 12 – Крок 11

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k=11 | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **-7** | **-7** | **0** | **-12** | **-10** | **-15** | **-22** | **-22** |
| 2 |  | **0** |  | **-7** | **-4** |  | **-12** | **-10** | **-12** | **-19** | **-19** |
| 3 |  |  | **0** |  | **-7** |  | **-11** |  | **-15** | **-22** | **-22** |
| 4 |  |  |  | **0** |  |  | **-5** | **-3** |  | **-9** | **-9** |
| 5 |  |  |  |  | **0** |  | **-4** |  | **-8** | **-15** | **-15** |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **-1** | **-1** |
| 8 |  |  |  |  |  |  |  | **0** |  | **-6** | **-6** |
| 9 |  |  |  |  |  |  |  |  | **0** | **-7** | **-7** |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

Крок 12. Після проведення розрахунків відстаней помножимо від’ємні значення на , а замість запишемо (табл. 13). Отримали матрицю відстаней між будь-якими двома вершинами.

Таблиця 13 – Крок 12

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 | **0** | **0** | **0** | **7** | **7** | **0** | **12** | **10** | **15** | **22** | **22** |
| 2 |  | **0** |  | **7** | **4** |  | **12** | **10** | **12** | **19** | **19** |
| 3 |  |  | **0** |  | **7** |  | **11** |  | **15** | **22** | **22** |
| 4 |  |  |  | **0** |  |  | **5** | **3** |  | **9** | **9** |
| 5 |  |  |  |  | **0** |  | **4** |  | **8** | **15** | **15** |
| 6 |  |  |  |  |  | **0** |  |  |  |  | **0** |
| 7 |  |  |  |  |  |  | **0** |  |  | **1** | **1** |
| 8 |  |  |  |  |  |  |  | **0** |  | **6** | **6** |
| 9 |  |  |  |  |  |  |  |  | **0** | **7** | **7** |
| 10 |  |  |  |  |  |  |  |  |  | **0** | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  | **0** |

За таблицями відстаней, що відображають покрокову роботу алгоритму, отримаємо критичний шлях таким чином. В таблиці, отриманій на останньому кроці (табл. 13), в стовпчику, що відповідає вершині-стоку (), тобто для j=11, по рядках знайдемо максимальне значення (їх може бути декілька), що відповідає вершині-джерелу (), тобто i=1. Це значення 22, що є довжиною критичного шляху.

В табл. 12 на тому самому місці, де в табл. 13 було значення довжини критичного шляху (i=1, j=11), воно знаходиться з протилежним знаком. Відповідну вершину з номером j запишемо в кінець критичного шляху. Зараз він має вигляд: .

Щоб знайти весь критичний шлях, відшукаємо, в таблиці для якого k для i=1 воно вперше з кінця з’явилося в синьому кольорі. Це було для k=10. Отже запишемо вершину з номером 10 на початок критичного шляху: .

Далі знаходимо в якій таблиці значення для i=1, j=10 вперше з кінця з’явилося в синьому кольорі. Це було для k=9. Критичний шлях: .

Значення для i=1, j=9 вперше з кінця з’явилося в синьому кольорі в таблиці, де k=5. Критичний шлях: .

Значення для i=1, j=5 вперше з кінця з’явилося в синьому кольорі в таблиці, де k=3. Критичний шлях: .

Значення для i=1, j=3 вперше з кінця з’явилося в синьому кольорі в таблиці, де k=0. Критичний шлях: .

Так як ми дісталися вершини-джерела , то критичний шлях знайдено.

Моменти настання кожної події містяться в табл. 13 в рядку для i=1, де j – номер події (відповідної вершини).

Для задачі 2, в якій ми знаходимо моменти настання кожної події, отримаємо результат, наведений в табл. 13. На рис. 22 цей момент вказаний в квадраті біля відповідної вершини (події). Зобразимо їх у вигляді окремої таблиці (табл. 14) та на рис. 22 біля відповідної вершини.

Таблиця 14 – Моменти настання подій

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Подія | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Момент | 0 | 0 | 0 | 7 | 7 | 0 | 12 | 10 | 15 | 22 | 22 |

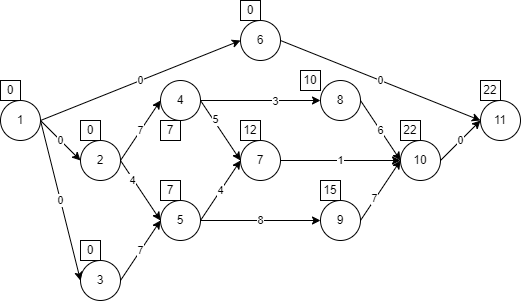


Рисунок 22 – Моменти настання подій

Для обчислення допустимих затримок настання подій на кожному кроці для множини вершин V\_end, що не мають вихідних дуг, обчислимо затримки delay для відповідних дуг, що входять в ці вершини. Ваги відповідних вершин та дуг будемо брати з табл. 13 та табл. 1 (з протилежним знаком).

Крок 1.

V\_end={}

delay[6][11] = weight() - weight() - weight(()) = 22-0-0=22.

delay[10][11] = weight() - weight() - weight(()) = 22-22-0=0.

Крок 2.

V\_end={, }

delay[1][6] = weight() - weight() - weight(()) = 0-0-0=0.

delay[8][10] = weight() - weight() - weight(()) = 22-10-6=6.

delay[7][10] = weight() - weight() - weight(()) = 22-12-1=9.

delay[9][10] = weight() - weight() - weight(()) = 22-15-7=0.

Крок 3.

V\_end={, , }

delay[4][7] = weight() - weight() - weight(()) = 12-7-5=0.

delay[5][7] = weight() - weight() - weight(()) = 12-7-4=1.

delay[4][8] = weight() - weight() - weight(()) = 10-7-3=0.

delay[5][9] = weight() - weight() - weight(()) = 15-7-8=0.

Крок 4.

V\_end={, }

delay[2][4] = weight() - weight() - weight(()) = 7-0-7=0.

delay[2][5] = weight() - weight() - weight(()) = 7-0-4=3.

delay[3][5] = weight() - weight() - weight(()) = 7-0-7=0.

Крок 5.

V\_end={, }

delay[1][2] = weight() - weight() - weight(()) = 0-0-0=0.

delay[1][3] = weight() - weight() - weight(()) = 0-0-0=0.

Подамо отримані результати, тобто допустимі затримки подій, у вигляді табл. 15 та рис. 23.

Таблиця 15 – Затримки подій

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Затримка | | j | | | | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| i | 1 |  | **0** | **0** |  |  | **0** |  |  |  |  |  |
| 2 |  |  |  | **0** | **3** |  |  |  |  |  |  |
| 3 |  |  |  |  | **0** |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  | **0** | **0** |  |  |  |
| 5 |  |  |  |  |  |  | **1** |  | **0** |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  | **22** |
| 7 |  |  |  |  |  |  |  |  |  | **9** |  |
| 8 |  |  |  |  |  |  |  |  |  | **6** |  |
| 9 |  |  |  |  |  |  |  |  |  | **0** |  |
| 10 |  |  |  |  |  |  |  |  |  |  | **0** |
| 11 |  |  |  |  |  |  |  |  |  |  |  |

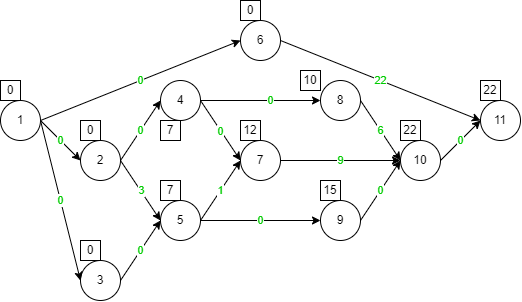


Рисунок 23 – Ілюстрація затримок подій

# ДОДАТОК Б

Код програми (розрахунки на мережі PERT)

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

int \*readingV(int &, int \*);

int \*\*readingU(int &, int \*\*);

int \*readingUWeights(int &, int \*);

size\_t countNumOfV();

size\_t countNumOfU();

int main()

{

setlocale(LC\_ALL, "ukr");

int numOfV = countNumOfV();

int numOfU = countNumOfU();

int \*V = new int[numOfV];

int \*\*U = new int \*[numOfU];

int \*UWeights = new int[numOfU];

for (size\_t i = 0; i < numOfU; i++)

{

U[i] = new int[numOfU];

}

auto inputV = readingV(numOfV, V);

auto inputU = readingU(numOfU, U);

auto inputUWeights = readingUWeights(numOfU, UWeights);

// network

std::ofstream network("network.txt", std::ios::out);

if (network.is\_open())

{

network << "Network (vertexes):" << std::endl;

for (size\_t i = 0; i < numOfV; i++)

{

network << V[i] << " ";

}

network << std::endl << std::endl;

network << "Network (edges - edges's weights):" << std::endl;

for (size\_t i = 0; i < numOfU; i++)

{

network << "(" << U[i][0] << " " << U[i][1] << ") - " << UWeights[i] << std::endl;

}

}

network.close();

// PERT

int inf = 1;

int \*\*distanceMatrix = new int \*[numOfV];

for (size\_t i = 0; i < numOfV; i++)

{

distanceMatrix[i] = new int[numOfV];

}

for (size\_t i = 0; i < numOfV; i++)

{

for (size\_t j = 0; j < numOfV; j++)

{

distanceMatrix[i][j] = inf;

}

}

for (size\_t i = 0; i < numOfU; i++)

{

distanceMatrix[inputU[i][0] - 1][inputU[i][1] - 1] = (-1) \* inputUWeights[i];

}

for (size\_t i = 0; i < numOfV; i++)

{

distanceMatrix[i][i] = 0;

}

for (size\_t k = 0; k < numOfV; k++)

{

for (size\_t i = 0; i < numOfV; i++)

{

for (size\_t j = 0; j < numOfV; j++)

{

if (distanceMatrix[i][k] < inf && distanceMatrix[k][j] < inf)

if (distanceMatrix[i][k] + distanceMatrix[k][j] < distanceMatrix[i][j])

distanceMatrix[i][j] = distanceMatrix[i][k] + distanceMatrix[k][j];

}

}

}

for (size\_t i = 0; i < numOfV; i++)

{

for (size\_t j = 0; j < numOfV; j++)

{

distanceMatrix[i][j] = (-1) \* distanceMatrix[i][j];

}

}

std::vector<int> criticalPath;

criticalPath.insert(criticalPath.begin(), inputU[numOfU - 1][1]);

int pathValue, VPrevious;

while (criticalPath[0] != 1)

{

pathValue = -inf, VPrevious = -inf;

for (size\_t i = numOfU; i > 0; i--)

{

if (inputU[i - 1][1] == criticalPath[0])

{

if (pathValue < distanceMatrix[inputU[i - 1][0] - 1][inputU[i - 1][1] - 1])

{

pathValue = distanceMatrix[inputU[i - 1][0] - 1][inputU[i - 1][1] - 1];

VPrevious = inputU[i - 1][0];

}

}

}

criticalPath.insert(criticalPath.begin(), VPrevious);

}

int \*delay = new int[numOfU];

for (size\_t i = 0; i < numOfV; i++)

{

for (size\_t j = 0; j < numOfU; j++)

{

if (U[j][1] == V[i])

{

delay[j] = distanceMatrix[0][U[j][1] - 1] - distanceMatrix[0][U[j][0] - 1] - inputUWeights[j];

}

}

}

// critical path

std::ofstream PERTPath("PERT.txt", std::ios::out);

if (PERTPath.is\_open())

{

PERTPath << "Critical path (vertexes): ";

for (size\_t i = 0; i < criticalPath.size(); i++)

{

PERTPath << criticalPath[i] << " ";

}

PERTPath << std::endl;

}

PERTPath.close();

// task 1

std::ofstream PERTLengthOfPath("PERT.txt", std::ios::app);

if (PERTLengthOfPath.is\_open())

{

PERTLengthOfPath << std::endl;

PERTLengthOfPath << "Length of critical path: " << distanceMatrix[0][numOfV - 1] << std::endl;

}

PERTLengthOfPath.close();

// task 2

std::ofstream PERTMomentsOfEvents("PERT.txt", std::ios::app);

if (PERTMomentsOfEvents.is\_open())

{

PERTMomentsOfEvents << std::endl;

PERTMomentsOfEvents << "Moments of events (vertex - moment): " << std::endl;

for (size\_t i = 0; i < numOfV; i++)

{

PERTMomentsOfEvents << inputV[i] << " - "

<< distanceMatrix[0][i] << std::endl;

}

}

PERTMomentsOfEvents.close();

// task 3

std::ofstream PERTEdgeDelays("PERT.txt", std::ios::app);

if (PERTEdgeDelays.is\_open())

{

PERTEdgeDelays << std::endl;

PERTEdgeDelays << "Delays of edges (edge - delay): " << std::endl;

for (size\_t i = 0; i < numOfU; i++)

{

PERTEdgeDelays << "(" << inputU[i][0] << " " << inputU[i][1] << ")"

<< " - "

<< delay[i] << std::endl;

}

}

PERTEdgeDelays.close();

delete[] V;

delete[] UWeights;

delete[] VToCheck;

for (size\_t i = 0; i < numOfV; i++)

{

delete[] U[i];

delete[] distanceMatrix[i];

}

delete[] U;

delete[] distanceMatrix;

delete[] delay;

return 0;

}

int \*readingV(int &numOfV, int \*V)

{

std::ifstream input("courseWork\_V.txt");

if (!input.eof())

{

for (size\_t i = 0; i < numOfV; i++)

{

input >> V[i];

}

input.close();

}

return V;

}

int \*\*readingU(int &numOfU, int \*\*U)

{

std::ifstream input("courseWork\_U.txt");

if (!input.eof())

{

for (size\_t i = 0; i < numOfU; i++)

{

for (size\_t j = 0; j < 2; j++)

{

input >> U[i][j];

}

}

input.close();

}

return U;

}

int \*readingUWeights(int &numOfU, int \*UWeights)

{

std::ifstream inputU("courseWork\_UWeight.txt");

if (!inputU.eof())

{

for (size\_t i = 0; i < numOfU; i++)

{

inputU >> UWeights[i];

}

inputU.close();

}

return UWeights;

}

size\_t countNumOfV()

{

size\_t numOfV = 0;

std::ifstream input("courseWork\_V.txt");

if (!input.bad())

{

std::string line;

while (getline(input, line))

{

numOfV++;

}

}

input.close();

return numOfV;

}

size\_t countNumOfU()

{

size\_t numOfU = 0;

std::ifstream input("courseWork\_U.txt");

if (!input.bad())

{

std::string line;

while (getline(input, line))

{

numOfU++;

}

}

input.close();

return numOfU;

}