Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики
Кафедра мультимедійних технологій

РОЗРОБКА АЛГОРИТМУ ПЕРЕРОЗПОДІЛУ ВСТУПНИКІВ ДО ВНЗ НА МІСЦЯ ДЕРЖАВНОГО ЗАМОВЛЕННЯ

Текстова частина до магістерської роботи (тези)
за спеціальністю «Інженерія програмного забезпечення»

Виконав: студент 2-го року навчання
Денисенко Андрій Васильович

Керівник: Франчук О. В., доцент, к.т.н.

Магістерська робота захищена з
оцінкою «_____»

Секретар ДЕК _____
«____» _____ 2021 р

Київ-2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики
Магістерська програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики, к.ф.-м.н.
_____С. С. Гороховський
(підпис)
«____» _____ 20__ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

(магістерську тезу)

студенту 2 року навчання МП «Інженерія програмного забезпечення»

Денисенку Андрію Васильовичу

на тему:

«Розробка алгоритму перерозподілу вступників до ВНЗ на місця державного замовлення»

Зміст ТЧ до магістерської роботи:

Дата видачі «___» _____ 20__ р.

Керівник: Франчук О. В, доцент, к.т.н.

Завдання отримав: Денисенко А. В.

Календарний план виконання роботи

Тема: Розробка алгоритму перерозподілу вступників до ВНЗ на місця державного замовлення

№ п/п	Назва етапу дипломного проекту	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	жовтень 2020 р.	
2.	Огляд літератури за темою роботи.	листопад -грудень 2020 р.	
3.	Розробка алгоритму.	січень -лютий 2020 р.	
4.	Написання роботи.	лютий - березень 2021 р.	
5.	Надання роботи керівнику для перевірки.	березень 2021 р.	
6.	Коригування роботи за результатами перевірки керівником .	березень -квітень 2021 р.	
7.	Створення слайдів для доповіді та написання доповіді.	травень 2021 р.	
8.	Остаточне оформлення пояснювальної записки та слайдів.	травень 2021 р.	
9.	Захист магістерської роботи .	червень 2021 р.	

Студент: Денисенко А. В.
Керівник: Франчук О. В, доцент, к.т.н.
“_____” _____

Зміст

Анотація

Для розподілу вступників на бюджетні місця в Україні використовується алгоритм адресного розміщення державного та регіонального замовлення, який побудований на основі алгоритму Гейла-Шеплі. Даний алгоритм пошуку стабільних пароутворень забезпечує правдивий механізм з точки зору груп, які роблять вибір, тобто університетів. Тому існують випадки коли студенти не завжди задоволені розподілом запропонованим державним алгоритмом.

Для того, щоб покращити місця розподілу студентів, був сформований алгоритм перерозподілу вступників до ВНЗ на місця державного замовлення і на його основі побудована програма, яка дістає дані вступної кампанії 2020 з відкритих джерел, і далі шукає можливі послідовності замін студентів.

Виконавши заміни, запропоновані програмою, студенти потраплять на місця навчання з вищим пріоритетом за поточне місце розподілу. При цьому, це не вплине негативно, ні на списки бюджетників, ні на роботу державного алгоритму адресного розміщення.

Вступ

Розподіл вступників на місця державного замовлення - важливе і відповідальне завдання. Автоматизація цього об’ємного процесу стала першочерговою задачею при введенні системи ЗНО в Україні.

На даний момент використовується алгоритм адресного розміщення державного та регіонального замовлення. Він побудований на основі алгоритму Гейла-Шеплі, який, як відомо, забезпечує правдивий механізм з точки зору груп які роблять вибір, тобто університетів.

Проаналізувавши списки зарахованих на бюджетні місця 2020 року було виявлено, що існують студенти яким можна покращити їх місце розподілу. Наприклад, припустимо існує студент, який з місця розподілу “A” хотів би потрапити на місце “B”. Мається на увазі, що “B” має більший пріоритет в списку даного студента. В свою чергу, є інший студент, який навпаки бажав би перейти з “B” до “A”. Замінивши даних двох студентів місцями, ми можемо дати їм можливість навчатися в закладі або напрямку, на якому вони хотіли б більше навчатися, але не потрапили на нього через роботу системи розподілу, яка розподілила студентів надаючи пріоритет вибору університетам.

Метою роботи є рішення даної проблеми. Потрібно означити алгоритм перерозподілу вступників до ВНЗ на місця державного замовлення і на його основі побудувати програму, яка дістає дані вступної кампанії 2020 з відкритих джерел, і далі шукає можливі послідовності замін студентів.

Робота складається з трьох розділів.

В першому розділі розглянуто процес формування, коригування та публікації рейтингових списків вступників до вищих навчальних закладів.

Другий розділ присвячено алгоритму адресного розміщення державного та регіонального замовлення 2020, який використовується на даний момент для формування рейтингових списків. Наведено опис його роботи і критерії верифікації справедливості. Також коротко оглянуто алгоритм пошуку стабільних пароутворень, на базі якого розроблений даний алгоритм.

В третьому розділі описано процес побудови алгоритму перерозподілу вступників до ВНЗ на місця державного замовлення. Розглянуто допоміжні алгоритми та теоретичні відомості, які використовуються при розробці. Сформована задача перерозподілу вступників і наведений алгоритм її розв’язання.

Четвертий розділ присвячено практичній частині роботи. Описано процес отримання даних, які використовуються при тестуванні розробленої програми і наведені результати виконання створеної програми.

1. Рейтинговий відбір вступників

Вступ у вищі навчальні заклади регламентується чинними умовами прийому на навчання для здобуття вищої освіти, що затверджені наказом Міністерства освіти і науки України № 1274 від 15 жовтня 2020 року.

Під час вступної кампанії 2020 року, вступ абітурієнтів до закладів вищої освіти України, для здобуття ступеня бакалавра, відбувався з урахуванням визначених вступником пріоритетностей заяв.

Вступники можуть подати не більше п’яти заяв на місця державного та регіонального замовлення і до тридцяти заяв на небюджетні конкурсні пропозиції (контракт).

Для всіх заяв вступник надає пріоритетність. Пріоритетність – це черговість заяв (від 1 до 5, де 1 - показник найбільшої пріоритетності заяви), за якою визначається черговість розгляду заяви при вступі на бюджетні місця. Після подання заяви встановлена вступником пріоритетність заяв не може бути змінена. Заклад вищої освіти у власних правилах прийому може передбачати встановлення локальних пріоритетностей.

Пріоритетність застосовується під час зарахування на навчання для здобуття ступеня бакалавра, на основі повної загальної середньої освіти за державним або регіональним замовленням.

1.1 Правила формування рейтингових списків

Рейтинговий список вступників формується за категоріями в послідовності:

- вступники, які мають право на зарахування за результатами співбесіди (тільки на основі повної загальної середньої освіти);
- вступники, які мають право на зарахування за квотами (тільки на основі повної загальної середньої освіти);
- вступники, які мають право на зарахування на загальних умовах.

Вступники, які мають право на зарахування за результатами співбесіди, впорядковуються у рейтинговому списку за алфавітом.

Рейтинговий список вступників, які мають право на на загальних умовах та які мають право на зарахування за квотами, впорядковується наступним чином:

- спочатку за конкурсним балом — від більшого до меншого;
- потім за пріоритетністю заяви від 1 до останньої;
- в останню чергу за середнім балом атестата — від більшого до меншого.

Якщо зазначені правила не дозволяють визначити послідовність вступників у рейтинговому списку, приймальна комісія навчального закладу ухвалює відповідне рішення самостійно на підставі аналізу поданих вступниками документів.

1.2 Правила коригування рейтингових списків

Згідно умов прийому на навчання для здобуття вищої освіти, після завершення строків подання заяв абітурієнтами, у кожному навчальному закладі за кожною окремою спеціальністю формується рейтинговий список абітурієнтів , що буде автоматично скоригований у відповідності до зазначених абітурієнтами пріоритетів відповідно до алгоритму.

Абітурієнтами, які будуть рекомендовані для зарахування на навчання за кошти державного бюджету, відповідно до правил прийому, стануть вступники, які на підставі сформованого рейтингу потраплять на місця державного замовлення у будь-якому навчальному закладі за будь-якою спеціальністю.

При цьому рекомендація про зарахування абітурієнта на навчання буде надаватись за найвищим пріоритетом, а всі заяви абітурієнта з меншим пріоритетом у такому випадку будуть автоматично анулюватися, що дозволить звільнити списки рекомендованих і надасть можливість автоматично потрапити на бюджетні місця у рейтингових списках абітурієнтам з нижчими балами.

Такі автоматичні коригування проводитимуться, доки в них буде необхідність, при цьому у абітурієнтів, які при кожному наступному автоматичному коригуванні списків вперше потраплятимуть на місця державного замовлення, також будуть скасовуватися заяви з нижчими пріоритетами.

Варто зазначити, що якщо абітурієнт потрапляє на місце державного замовлення за спеціальністю з меншим пріоритетом, він не втрачатиме місце у рейтингу за спеціальністю з більш високим пріоритетом і може потрапити на бюджетну форму навчання на більш пріоритетну для себе спеціальність у випадку автоматичного звільнення місця. У такому випадку його заява з меншим пріоритетом також буде анульована, що автоматично надасть можливість потрапити у «бюджетний список» іншому за списком абітурієнту.

Такі коригування рейтингових списків проводяться автоматично до того моменту, поки всі бюджетні місця за всіма спеціальностями у всіх вишах України не будуть заповнені і кожному з абітурієнтів (які матимуть найвищі рейтингові бали) не буде запропоновано тільки одне бюджетне місце за одним із пріоритетів.

Наприклад, якщо абітурієнт рекомендований для зарахування на навчання за кошти державного бюджету на спеціальність за 3-м пріоритетом, то всі його заяви з меншими пріоритетами будуть анульовані. При цьому, при автоматичному коригуванні рейтингових списків абітурієнт може бути рекомендований на навчання за спеціальністю з вищим, 2-м пріоритетом, оскільки зі списків також будуть виключатися заяви інших абітурієнтів з нижчими для них пріоритетами. У такому випадку заява цього абітурієнта з 3-м пріоритетом також буде анульована, оскільки в процесі коригування списків він потрапить на місце державного замовлення у 2-му для себе пріоритеті. У такому випадку такий абітурієнт матиме вибір: вступити на бюджет за 2-м пріоритетом, або подавати документи на контракт за 1, 3, 4 або 5 пріоритетом.

1.3 Публікація рейтингових списків та зарахування на навчання

Після автоматично коригування сформується кінцевий рейтинговий список, за кожною спеціальністю у закладах вищої освіти, який буде оприлюднений на сайті Вступ.OCBITA.UA.

Таким чином абітурієнти з найвищими балами отримають право вступу на бюджет тільки на одну з п’яти конкурсних пропозицій з найвищим пріоритетом і матимуть час для подання оригіналів документів до вищого навчального закладу, де вони рекомендовані до зарахування.

Абітурієнти втрачатимуть право на зарахування (переведення) на навчання за кошти державного бюджету якщо з будь-яких причин не подадуть оригінали документів до вищого навчального закладу. При цьому вони зберігатимуть право вступу за контрактною формою навчання на будь-яку зі спеціальностей, на яку вони подавали заяву.

Якщо вступник не виконав вимог для зарахування за державним замовленням (не подав оригінали документів у визначені строки) рекомендація на бюджет анулюється, а його місце надається іншому вступнику.

2. Алгоритм адресного розміщення державного та регіонального замовлення 2020

2.1 Опис алгоритму

Згідно з умовами прийому на навчання до закладів вищої освіти України в 2020 році, алгоритм адресного розміщення призначений для розподілу місць державного та регіонального замовлення.

Перед застосуванням алгоритму проводиться коригування вхідної інформації:

- максимальні (загальні) обсяги державного або регіонального замовлення конкурсів, супер обсяги державного замовлення відповідних широких конкурсів зменшуються на використані обсяги квоти-2 та на кількість місць, на які зараховані вступники, які мають право на зарахування за співбесідою;
- кваліфікаційні мінімуми державного замовлення (далі - мінімальні обсяги) конкурсів зменшуються на використані обсяги квоти-2 та на кількість місць, на які зараховані вступники, які мають право на зарахування за співбесідою. Якщо в результаті отримується число менше ніж 1, конкурс виконав мінімальні обсяги і вони надалі для нього в алгоритмі не використовуються.

2.2 Визначення рекомендованих до зарахування за конкурсами

1. Етап А

Перший крок.

Кожному розрахунковому конкурсу пропонується перелік вступників, для яких цей розрахунковий конкурс має найвищу пріоритетність. Кожний розрахунковий конкурс включає до списку очікування кращих за власним

рейтинговим списком вступників із запропонованих вступників у кількості, що не перевищує обсягу розрахункового конкурсу, а решті відмовляє.

Кожна група субконкурсів кожного конкурсу (субконкурс А, та/або субконкурс Б, та/або субконкурс ББ і субконкурс В) перевіряється на перевищення максимального (загального) обсягу державного чи регіонального замовлення конкурсу. У разі перевищення визначається відповідна кількість вступників із суб конкурсу В з нижчими позиціями в рейтинговому списку вступників, які отримують відмову.

Кожний широкий конкурс перевіряється на перевищення суперобсягу державного замовлення. У разі перевищення за об'єднаним списком очікування визначається відповідна кількість вступників (не із субконкурсів А, і не із субконкурсів Б, і не із субконкурсів ББ) з найменшими значеннями конкурсного бала (за рівних конкурсних балів - з урахуванням пункту 2 розділу IX Умов прийому), які також отримують відмову.

На наступних кроках кожний вступник, який на цей момент не внесений до списку очікування жодного розрахункового конкурсу, пропонується тому розрахунковому конкурсу, який має для нього найвищу пріоритетність (крім тих, де він уже отримав відмову). Кожний розрахунковий конкурс об'єднує наявний у нього список очікування та отриману пропозицію, формує новий список очікування за власним рейтинговим списком вступників у кількості, що не перевищує обсягу розрахункового конкурсу, а решті відмовляє.

Кожна група субконкурсів кожного конкурсу (субконкурс А, та/або субконкурс Б, та/або субконкурс ББ і субконкурс В) перевіряється на перевищення максимального (загального) обсягу державного або регіонального замовлення конкурсу. У разі перевищення визначається відповідна кількість вступників із субконкурсу В з нижчими позиціями в його рейтинговому списку вступників, які отримують відмову.

Кожний широкий конкурс перевіряється на перевищення суперобсягу державного замовлення. У разі перевищення за об'єднаним списком очікування визначається відповідна кількість вступників (не із субконкурсів А, і не із

субконкурсів Б, і не із субконкурсів ББ) з найменшими значеннями конкурсного бала (за рівних конкурсних балів - з урахуванням пункту 2 розділу IX Умов прийому), які також отримують відмову.

Етап А вважається виконаним, коли вичерпується перелік пропозицій вступників до розрахункових конкурсів, які не перебувають у списках очікування та не отримали відмови за всіма розрахунковими конкурсами. Перехід до етапу Б.

2. Етап Б.

Якщо наявні конкурси, в яких кількість вступників у списку очікування (сума вступників у списках очікування субконкурсу А, та/або субконкурсу Б, та/або суб конкурсу ББ і суб конкурсу В) менше ніж мінімальний обсяг державного або регіонального замовлення, такі конкурси анулюються, а вступники з їх списків очікування виключаються, отримують відмову і помічаються як такі, що допущені до етапу В.

Фіналіст розрахункового конкурсу - вступник з найнижчим положенням у рейтинговому списку розрахункового конкурсу, включений до списку очікування, після завершення етапу А.

Фіналіст широкого конкурсу - вступник, крім вступників із субконкурсів А, Б та ББ, з найнижчим положенням у широкому рейтинговому списку широкого конкурсу, включений до списку очікування, після завершення етапу А.

Перехід до етапу В.

3. Етап В

Кожний допущений до етапу В вступник, який на цей момент не внесений до списку очікування жодного розрахункового конкурсу, пропонується тому розрахунковому конкурсу, який має для нього найвищу пріоритетність (крім тих, де він уже отримав відмову в межах етапів А та В). Кожний розрахунковий конкурс включає до свого списку очікування вступників з отриманих пропозицій.

Кожен розрахунковий конкурс перевіряється на перевищення обсягу розрахункового конкурсу. У разі перевищення визначається відповідна кількість вступників, допущених до етапу В, з нижчими позиціями в рейтинговому списку вступників, які отримують відмову, за винятком тих вступників, чия позиція в

рейтинговому списку розрахункового конкурсу вища за позицію фіналіста розрахункового конкурсу.

Кожна група субконкурсів кожного конкурсу (субконкурс А, та/або субконкурс Б, та/або субконкурс ББ і субконкурс В) перевіряється на перевищення максимального (загального) обсягу державного або регіонального замовлення конкурсу. У разі перевищення визначається відповідна кількість вступників, допущених до етапу В, із субконкурсу В з нижчими позиціями в його рейтинговому списку вступників, які отримують відмову, за винятком тих вступників, чия позиція вища за позицію фіналіста розрахункового конкурсу відповідного субконкурсу В.

Кожний широкий конкурс перевіряється на перевищення суперобсягу державного замовлення і в разі перевищення за об'єднаним списком очікування визначається відповідна кількість вступників (не із субконкурсів А, не із субконкурсів Б, не із субконкурсів ББ), допущених до етапу В, з найменшими значеннями конкурсного бала (за рівних конкурсних балів - з урахуванням пункту 2 розділу IX Умов прийому), які також отримують відмову, за винятком тих вступників, чия позиція в широкому рейтинговому списку вища за позицію фіналіста широкого конкурсу.

Етап В вважається виконаним, коли вичерпується перелік пропозицій вступників (допущених до етапу В) до розрахункових конкурсів, які не перебувають у списках очікування та не отримали відмови за всіма розрахунковими конкурсами.

Вступники, які на цей момент залишились у списках очікування, одержують рекомендацію до зарахування. Кількість вступників, що отримали рекомендацію, визначає кількість рекомендованих за кожним конкурсом вступників.

2.3 Критерії верифікації справедливості роботи алгоритму

1. Для вступника А, який був допущений до конкурсного відбору і не отримав рекомендації до зарахування до конкретного конкурсу Х, результат справедливий, якщо:

- для вступника, що не має права на зарахування за квотою-1, квотою-3 або квотою-4, виконується хоча б одне з тверджень 1 або 2;
- для вступника, що має право на зарахування за квотою-1, виконується твердження 1 або одночасно виконуються твердження 2 та 3;
- для вступника, що має право на зарахування за квотою-4, виконується твердження 1 або одночасно виконуються твердження 2 та 4;
- для вступника, що має право на зарахування за квотою-3, виконується твердження 1 або одночасно виконуються твердження 2 та 5.

2. Твердження:

1) вступник А отримав рекомендацію до іншого конкурсу за вищою для нього пріоритетністю;

2) не існує жодного вступника Б, рекомендованого до зарахування в конкурсі Х (за винятком осіб, які отримали рекомендацію в межах квоти-1, квоти-3 або квоти-4), який має конкурсний бал не вище ніж у вступника А, у разі рівності конкурсних балів - має місце в рейтинговому списку нижче ніж вступник А, та (якщо конкурс Х входить до широкого конкурсу):

не існує жодного вступника Б, рекомендованого до зарахування в іншому конкурсі Y широкого конкурсу, до якого входить конкурс Х (за винятком осіб, які отримали рекомендацію в межах квоти-1, квоти-3 або квоти-4), який має конкурсний бал у конкурсі Y не вище ніж вступник А у конкурсі Х, у разі рівності конкурсних балів - розміщений нижче в широкому рейтинговому списку (з урахуванням пункту 2 розділу ІХ Умов прийому) ніж вступник А, крім такого випадку:

у конкурсі Х повністю вичерпано максимальний обсяг державного замовлення без врахування рекомендацій вступників, заяви яких отримували відмову під час ануляції конкурсів;

3) не існує жодного вступника Б, рекомендованого до зарахування в конкурсі Х за квотою-1, який має конкурсний бал не вище ніж у вступника А, у разі рівності конкурсних балів - має місце в рейтинговому списку нижче ніж вступник А;

4) не існує жодного вступника Б, рекомендованого до зарахування в конкурсі Х за квотою-4, який має конкурсний бал не вище ніж у вступника А, у разі рівності конкурсних балів - має місце в рейтинговому списку нижче ніж вступник А;

5) не існує жодного вступника Б, рекомендованого до зарахування в конкурсі Х за квотою-3, який має конкурсний бал не вище ніж у вступника А, у разі рівності конкурсних балів - має місце в рейтинговому списку нижче ніж вступник А.

3. Для закладу вищої освіти, запропонований яким конкурс Х (який входить до широкого конкурсу) не вичерпав максимального обсягу державного або регіонального замовлення:

не існує жодного вступника Б, допущеного до участі в конкурсі Х і не рекомендованого до зарахування в конкурсі Х, що не отримав рекомендації до зарахування в іншому конкурсі за вищою для нього пріоритетністю та має конкурсний бал у конкурсі Х не нижче ніж будь-який вступник В (за винятком осіб, які отримали рекомендацію в межах квоти-1, квоти-3 або квоти-4), який рекомендований до зарахування до одного з конкурсів цього широкого конкурсу в межах суперобсягу державного замовлення, у разі рівності конкурсних балів - має місце в широкому рейтинговому списку вище (з урахуванням пункту 2 розділу IX Умов прийому) ніж вступник В.

4. Для закладу вищої освіти, запропонований яким конкурс Х (що не входить до широкого конкурсу) не вичерпав загального обсягу державного або регіонального замовлення:

не існує жодного вступника, допущеного до участі в конкурсі Х і не рекомендованого до зарахування в конкурсі Х, що не отримав рекомендації до зарахування в іншому конкурсі за вищою для нього пріоритетністю.

5. Для закладу вищої освіти, запропонований яким конкурс Х не вичерпав квоту-1, квоту-3 або квоту-4:

не існує жодного вступника, що має право на зарахування за квотою-1 (квотою-3, квотою-4), допущеного до участі в конкурсі Х і не рекомендованого до зарахування в конкурсі Х, що не отримав рекомендації до зарахування в іншому конкурсі за вищою для нього пріоритетністю.

6. Для закладу вищої освіти, запропонований яким конкурс Х було анульовано:

рейтинговий список не містить вступників у кількості, не меншій від мінімального обсягу державного або регіонального замовлення, для яких виконуються такі умови:

вступник отримав рекомендацію за меншою для нього пріоритетністю;

вступник має конкурсний бал у конкурсі Х не нижче ніж будь-який вступник В (за винятком осіб, які отримали рекомендацію в межах квоти-1, квоти-3 або квоти-4, та осіб, які отримали рекомендацію і мали заяву, що отримала відмову під час ануляції конкурсів), який рекомендований до зарахування до одного з конкурсів цього широкого конкурсу в межах суперобсягу державного замовлення, у разі рівності конкурсних балів - має місце в широкому рейтинговому списку вище (з урахуванням пункту 2 розділу ІХ Умов прийому) порівняно із вступником В.

2.4 Алгоритм Гейла-Шеплі

Алгоритм наведений в попередньому розділі розроблений на основі алгоритму Гейла-Шеплі. Він дозволяє знайти оптимальні поєднання в ситуаціях,

коли для кожного члена однієї групи (група 1) необхідно знайти відповідну пару в іншій групі (група 2).

Кроки алгоритму:

1. Кожен член групи 1 робить пропозицію першому об’єкту із свого списку бажаних для пароутворення.
2. Кожен член групи 2 відповідає “можливо” на пропозицію, яку він вважає найкращою для себе і “ні” іншим.
3. В результаті утворилось тимчасове пароутворення з 2 субєктів.
4. В кожному наступному раунді незайнятий член групи 1 робить пропозицію найбажанішому об'єкту групи 2, якому він ще не робив пропозиції (незалежно від того, об'єднаний він в пару чи ні).
5. Потім кожен член групи 2 відповідає “можливо”, якщо до цього часу в нього немає пари або новий партнер більш бажаний за наявного (в даному випадку він відкидає пропозицію нинішнього тимчасового партнера, який стає незайнятим).
6. Наведений процес повторюється до тих пір, поки всі не будуть зайняті

Складність виконання цього алгоритму складає $O(n^2)$, де n - це кількість членів групи 1 або групи 2.

2.4.1 Зауваження щодо алгоритму пошуку стабільних пароутворень

Алгоритму Гейла-Шеплі гарантує, що всі отримають пару і утворені пари стабільні. Алгоритм забезпечує правдивий механізм з точки зору груп які роблять вибір. Тобто жоден член групи не може домогтися кращої відповідності для себе, спотворюючи свої переваги. Але цього не можна сказати про другу групу, її члени не завжди отримують найбажанішу пару.

Також застосування даного алгоритму для розподілу студентів накладає певні обмеження і модифікації. Так студенти мають обмеження на вибір бажаних вищих навчальних закладів, у розмірі 5-ти спеціальностей. З іншої сторони, навчальні заклади вибирають не одного студента, а деяку кількість по квоті.

Дані доповнення призводять до певного спотворення уподобань студентів. Вищі навчальні заклади надають перевагу студентам, які мають кращі бали, в той час як вступники малоймовірно будуть допущені до вузу, якому вони віддали перевагу згідно з власними інтересами, тому що вони виберуть більш реалістичні напрямки, в яких вони будуть займати вищі місця.

3. Розробка алгоритму перерозподілу вступників до ВНЗ на місця державного замовлення

3.1 Означення теорії графів

Орієнтованим графом називається сукупність $G = (V, E, L, \partial E, \partial L)$, яка складаэться з трьох множин: V – множина вершин (vertices), E – множина ребер (edges), L – множина петель (loops) та відображень:

$$\partial E : E \to V \times V,$$

$$\partial L : L \to V.$$

При цьому якщо $\partial E(e) = (v_1, v_2)$, то вершину v_1 будемо називати початком ребра e, а вершину v_2її кінцем.

Мультиорграф — це орієнтований граф, в якому допустимі кратні дуги, тобто є дуги, які мають однакові початкові і кінцеві вершини. Всякий граф є мультиграфом, але не всякий мультиграф буде графом. Якщо $G = (V,\ E)$ – мультиграф, то E може мати кілька ребер, які з'єднують одні і ті ж вершини. Такі ребра називаються кратними ребрами.

Мультиграф називається скінченним, якщо множини V і E скінчені. Для скінченного графа, достатньо лише скінченності множин його вершин V, оскільки скінченність V дає скінченність $V^{(2)}$, тобто граф називається скінченним, якщо скінченна множина його вершин. Скінченний граф з n вершинами називається графом n-го порядку.

Дві вершини u і v графа $G = (V,\ E)$ суміжні, якщо $(u,\ v) \in E$, і несуміжні – в протилежному випадку. Якщо $(u,\ v) \in E$, то вершини u, і v називаються кінцями ребра $(u,\ v)$.

Два ребра називаються суміжними, якщо вони мають спільний кінець. Відношення суміжності як для вершин, так і для ребер є симетричним відношенням. Вершина u і ребро e називаються інцидентними, якщо u є кінцем ребра e, і неінцидентним – в протилежному випадку.

Маршрутом у заданому графі $G = (V,\ E)$ називається скінченна послідовність його ребер, яка має вигляд $(v_0,\ v_1), (v_1,\ v_2),\ \ldots\ (v_{k-1},\ v_k)$. Число k ребер маршруту називається довжиною цього маршруту.

Маршрут називається ланцюгом, якщо всі його ребра різні, і простим ланцюгом, якщо всі його вершини, крім можливо першої і останньої – різні.

Циклом, називається циклічний ланцюг, а простим циклом – простий циклічний ланцюг.

Дві вершини v і w називаються зв'язними, якщо існує маршрут вигляду $(\ldots,\ v_1,\ e_1,\ v_2,\ e_2, \ldots,\ v_{n-1},\ e_{n-1},\ v_n,\ \ldots)$ із кінцями v та w. В цьому випадку також говорять, що вершина v досяжна з вершини w.

Граф називається зв'язним, якщо будь-яка пара його вершин є зв'язаною. Зв'язністю графу називається мінімальна кількість вершин, вилучення яких приводить до утворення незв'язного графу.

3.2 Компоненти сильної зв'язності в орієнтованому графі

Зв'язність – це бінарне відношення на множині вершин. Воно рефлексивне (кожна вершина зв'язана сама із собою за означенням), симетричне (для кожного маршруту є обернений маршрут) та транзитивне. Транзитивність означає, що якщо є маршрут з v до w та маршрут з w до u, то є маршрут з v до u. Це очевидно: щоб отримати такий маршрут, достатньо до послідовності ребер, які ведуть з v до w, дописати справа послідовність ребер, яка веде з w до u.

Таким чином, відношення зв'язності є відношенням еквівалентності на множині вершин графу G і розбиває цю множину на підмножини, що не перетинаються – класи еквівалентності. Всі вершини одного класу зв'язані між собою, вершини різних класів між собою не зв'язані. Підграф, утворений всіма вершинами одного класу, називається компонентою зв'язності графу G.

Числом вершинної зв'язності $k(G)$ (читається «каппа від G») простого графу G називають найменшу кількість вершин, вилучення яких утворює незв'язаний або одновершинний граф.

Нехай G – простий граф з $n > 1$ вершинами. Числом реберної зв'язності $\lambda(G)$ (читається «лямбда від G») графу G називають найменшу кількість ребер, вилучення яких дає незв'язаний граф.

Вершину u простого графу G називають точкою з'єднання, якщо граф G в разі її вилучення матиме більше компонент, ніж даний граф G. Тобто кількість компонент зв'язності при вилученні цієї вершини у графу G збільшується. Множина ребер графу називається розрізом, якщо вилучення цих ребер з графу G приводить до збільшення кількості компонент зв'язності. Якщо розріз містить одне ребро, то його називають мостом.

Граф називається роздільним, якщо він містить хоча б одну точку з'єднання, та нероздільним в іншому випадку. Максимальні нероздільні підграфи графу називаються блоками.

Отже, точки з'єднання й мости – це своєрідні «вузькі місця» простого графу.

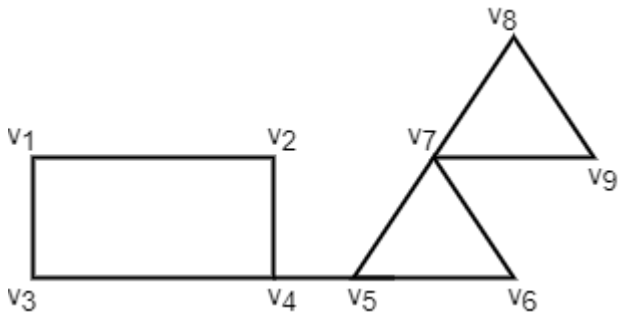

Рисунок 3.1 - Приклад графу

Граф на рисунку 3.1 має три точки з'єднання v_4, v_5 та v_7 та один міст (v_4, v_5).

Орієнтований граф називається сильно зв'язним, якщо для будь-яких двох його вершин v та w існує шлях в обох напрямках. Орієнтований граф називається

однобічно-зв’язним, якщо для будь-яких двох його вершин v та w існує шлях хоча б в одному напрямку.

Псевдографом, асоційованим з орієнтованим псевдографом $D_1(V)$, називається псевдограф $D_2(V)$ такий, коли кожну дугу графу D_1 замінено на ребро графу D_2. Орієнтований граф називається слабкозв’язним, якщо зв’язним є асоційований з ним псевдограф. Якщо граф (орієнтований граф) не є зв’язним (слабкозв’язним), то він називається незв’язним.

У сильно зв’язаному орграфі довільна вершин v входить принаймні в один цикл, утворений шляхами з v до деякої іншої u та назад з u до v. Цикли, які проходять через v та інші вершини графу, не обов’язково всі різні. Так, сильно зв’язний граф, який містить n вершин, може представляти собою один простий цикл, який проходить скрізь всі вершини.

3.3 Розбиття графа на компоненти сильної зв’язності. Алгоритм Косараджу

Алгоритм Косараджу - це алгоритм з лінійним часом для пошуку сильно зв’язних компонентів в орієнтованому графі. Косараджу запропонував його в 1978 році, але не опублікував, в той час як Шаріра незалежно виявив його і опублікував в 1981 году.

Пояснення і реалізація алгоритму Косараджу відносно прості. Щоб знайти компоненти сильної зв'язності потрібно:

1. Запустити алгоритм пошуку в глибину на графі G, щоб отримати порядок обходу кожного вузла.
2. Створити зворотній граф GT поданого графа.
3. Виконати пошук алгоритмом пошуку в глибину на зворотному графі, але тут порядок доступу вершин під час пошуку не відповідає розміру мітки вершини, а відповідає порядку значення f кожної вершини від більшої до меншої.

Ліс отриманий зворотнім графом під час виконання алгоритму пошуку в глибину - відповідна зв'язана область. Даний процес показаний на групі рисунків 3.2 - 3.5

Вище ми згадували зворотний граф GT вихідного графа G, який визначається як: $GT = (V, ET)$, $ET = \{(u,\ v): (v, u) \in E\}$. Іншими словами, GT складається з ребер, зворотних реберам з G, що зазвичай називається транспонуванням графа G. Тут варто згадати, що зворотний граф GT і вихідний граф G мають в точності однакові компоненти зв'язності, тобто вершини s і t взаємно досяжні в G , тоді і тільки тоді, коли s і t також знаходяться в GT.

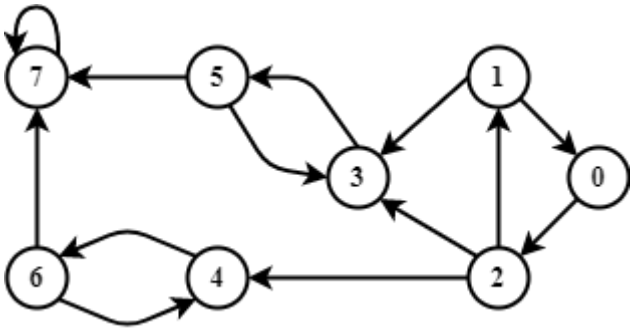

Рисунок 3.2 - Приклад роботи алгоритму Косараджу. Крок 1

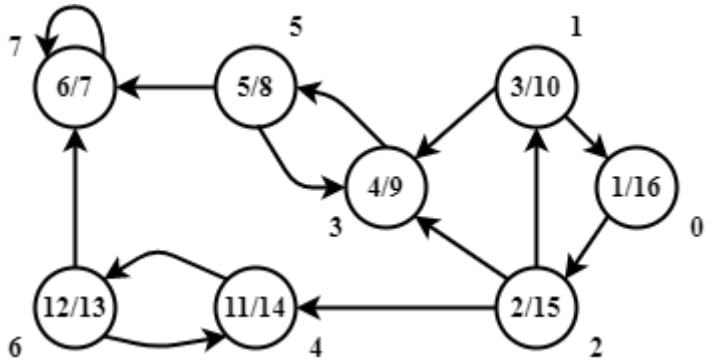

Рисунок 3.3 - Приклад роботи алгоритму Косараджу. Крок 2

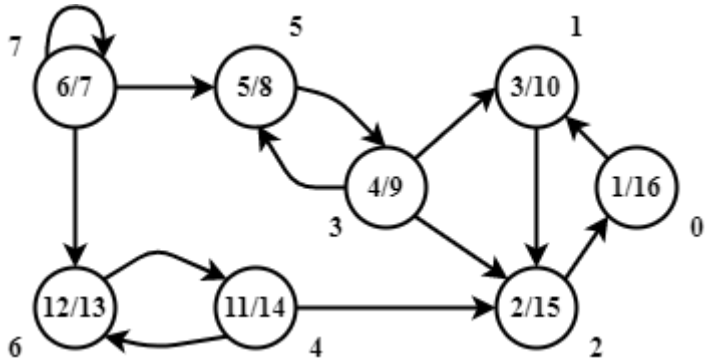

Рисунок 3.4 - Приклад роботи алгоритму Косараджу. Крок 3

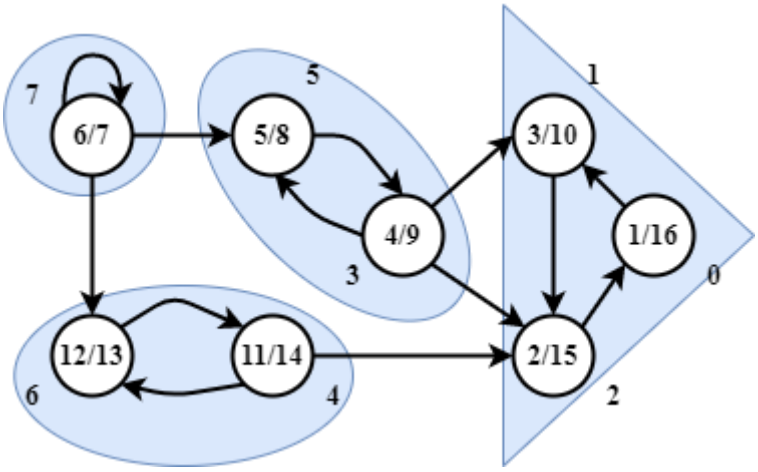

Рисунок 3.5 - Приклад роботи алгоритму Косараджу. Крок 4

Згідно обговоренню алгоритму Косараджу вище, його реалізація виглядає наступним чином:

```
/*
Алгоритм 1: Косараджу, кроки алгоритму:
1. Виконати DFS на вихідному графі G, щоб отримати порядок обходу кожного вузла
orderedArr [];
2. Створити зворотний граф GT вихідного графа;
3. Відвідайте кожен вузол в GT в порядку, зворотному orderedArr [];
  Функція повертає двомірний однозв'язний список singleLinkedList
  Кожен вузол являє собою однозв'язний список
 */
    public static Link Kosaraju(GraphLink Graph){
        Link singleLinkedList = new Link();
        int orderedArr[] = new int[Graph.getNewVert()];
```

```
        // Пошук в глибину
        SearchGraph.DFS(Graph);
        // Зберігаємо час початку пошуку
        for(int i = 0; i < SearchGraph.f.length; i++){
            orderedArr[i] = SearchGraph.f[i];
            System.out.print(SearchGraph.parent[i] + " || ");
        }
        System.out.println();
        // GT зворотний граф графу G
        GraphLink GT = Utilities.reverseTheGraph(Graph);
        /* Функція відвідує кожен вузол в порядку, зворотному orderedArr,
          і додає нові елементи зв'язного списку в singleLinkedList; */
        SearchGraph.DfsAlgOfKosaraju(GT, orderedArr, singleLinkedList);
        // Роздрукувати всі зв'язні області
        for(singleLinkedList.goFirst(); singleLinkedList.getCurrentValue()!= null; singleLinkedList.next(
)){
            // Отримати зв'язну область
            GraphNodeSingleLink  graphComponent = (GraphNodeSingleLink
)(singleLinkedList.getCurrentValue().elem);
            // Вивести кожен вузол графа в цій зв'язній області
            for(graphComponent.goFirst();
                graphComponent.getCurrentValue() != null; graphComponent.next()){
                System.out.print(graphComponent.getCurrentValue().elem + "\t");
            }
        }
        return singleLinkedList;
    }
```

Спочатку виконуємо алгоритм пошуку в глибину на вихідному графу і зберігаємо час початку пошуку, orderedArr, для кожної вершини. Потім викликаємо функцію зворотного графа reverseTheGraph в класі Utilities для отримання зворотного графа GT вихідного графа. Нарешті викликає рекурсивну функцію DfsAlgOfKosaraju класу SearchGraph.

Ця функція виконує пошук в глибину на графі, відповідно до часу кожної вершини. Функція повертає двовимірний однозв'язний список singleLinkedList. Елемент елемента кожного вузла однозв'язного списку також є однозв'язним списком. Однозв'язний список в кожному вузлі є зв'язна область, довжина singleLinkedList являє собою кількість об'єднаних областей на малюнку. Причина, по якій така структура даних використовується в якості результату функції, полягає в тому, що неможливо дізнатися, скільки сильно зв'язних гілок в графі до того, як функція повернеться, а також кількість вершин в кожній гілці. Зв'язний список природно стає оптимальним вибором. Результати відповіді двох інших алгоритмів також використовують цю форму вихідних результатів зв'язного списку.

Зворотний граф, що повертається функцією reverseTheGraph, являє собою новостворений граф з кожним ребром в напрямку, протилежному напрямку вихідного ребра. Суміжні вершини в зв'язному списку, відповідні вершині в оригінальному графі, розташовані в порядку зростання міток, що може підвищити ефективність зв'язних алгоритмів (наприклад, функція isEdge (int, int)). Цієї умови необхідно також дотримуватися при розрахунку зворотного графа. Реалізація статичної функції виглядає наступним чином:

```
// Створюємо транспонований граф
    public static GraphLink reverseTheGraph(GraphLink G){
       GraphLink GT = new GraphLink(G.getNewVert());
       for(int i = 0; i < G.getNewVert(); i++){
          for(Edge w = G.firstEdge(i); G.isEdge(w);
             w = G.nextEdge(w)) {
              GT.setEdgeW(w.getV2(), w.getV1(),
                      G.getEdgeW(w));
          }
       }
       return GT;
    }
```

У функції вершини переходять від меншої до більшої, відповідно до їх міток. Для вершини i обходимо вершини $i_0, \ldots, i_n$, у відповідному зв'язному списку і додаємо ребра (i_0, i_1), ... (i_n, i) до зворотного графу. Отримуємо зворотний граф GT вихідного графу. Можна виявити, що функція setEdgeW викликається для додавання ребер в GT. Якщо ребро, вага якого потрібно змінити, не існує, функція setEdgeW діє як функція додавання ребер і може гарантувати, що мітки всіх суміжних вершин розташовані в порядку зростання в зв'язному списку.

Ключовим кроком в реалізації Косараджу є виклик функції DfsAlgOfKosaraju для виконання рекурсивного пошуку в глибину на зворотному графі GT. Два інших формальних параметра функції - це час закінчення кожного пошуку вершини, отриманого з вихідного графа при виконанні алгоритму пошуку в глибину, і зв'язний список singleLinkedList, який містить результат зв'язних компонентів.

Реалізація DfsAlgOfKosaraju виглядає наступним чином:

```
/*
    @param  Зворотний граф GT
    @param  Масив обходу порядків і часу кожної вершини вихідної DFS
    @param SingleLinkedList під'єднання гілки зберігаються у зв'язаному списку
 */
   public static void DfsAlgOfKosaraju(GraphLink GT, int orderedArr[],
                                Link singleLinkedList){
      /* Обчислюємо масив newList на основі масиву orderedArr, і новий порядок доступу:
      i-й відвіданий вузол - це вузол newList [i] на вихідному графі. * /
      int newList[] = new int[orderedArr.length];
      //  Викликаємо функцію newOrderedArrayMap, щоб змінити порядок масиву
      newOrderedArrayMap(orderedArr, newList);
      int n = GT.getNewVert();
          // Тут потрібно тільки записати колір, інша інформація не важлива
      color  = new COLOR[n];
         //  Колір ініціалізується білим
```

```
        for(int i = 0; i < n; i++)
            color[i] = COLOR.WHITE;
             // Щоб знайти всі зв'язані області на малюнку, повторюємо цикл
        for(int i = 0; i < newList.length; i++){
            // I-й відвіданий вузол - це вузол newList [i] на вихідному графі
            int  newNodeNum = newList[i];
            if(color[newNodeNum] != COLOR.WHITE)
                continue;
            // Створюємо зв'язний список вузлів графа
            GraphNodeSingleLink  grsphNodeSingleLink = new GraphNodeSingleLink ();
            // Викликаємо рекурсивно функцію і глибокий пошук по графу з newNodNum в якості
початкової    точки
            DfsAlgOfKosarajuVISIT(GT, orderedArr, newNodeNum, grsphNodeSingleLink);
            // Додаємо зв'язний список
            singleLinkedList.append(new ElemItem<GraphNodeSingleLink >(grsphNodeSingleLink));
        }
    }
```

Функція спочатку визначає порядок доступу вершин в GT, відповідно до масиву orderedArr. Метод полягає в тому, щоб створити масив newOrderedArr, рівний за розміром до orderedArr, і викликати функцію newOrderedArrayMap для присвоєння значень елементів newOrderedArr. Значення newOrderedArr[i] таке: порядок доступу i-го вузла на вихідному графі в зворотному графі - newOrderedArr [i].

Наприклад, час обходу алгоритму пошуку в глибину для вершини вихідного графа складає:

0)16 1)10 2)15 3)9 4)14 5)8 6)13 7)7 8)20 9)19

Тоді порядок відвідування вершин в зворотному графі:

8, 9, 0, 2, 4, 6, 1, 3, 5, 7,

Функція newOrderedArrayMap реалізована наступним чином:

```
/* Відповідно до часу закінчення обходу, отриманим DFS кожної вершини вихідного графа,
отримуємо новий вузол.
```

```
     @param Час закінчення обходу OrderedArr DFS кожної вершини вихідного зображення
     @param NewOrderedArr Порядок відвідування вузлів в зворотному графі
    */
   public static void newOrderedArrayMap(int[] orderedArr, int[] newOrderedArr){
      // Щоб запобігти руйнуванню вихідного масиву orderedArr, скопіюємо його.
      int orderedArrTemp[] = new int[orderedArr.length];
      for(int i = 0; i < orderedArr.length; i++)
         orderedArrTemp[i] = orderedArr[i];
      int max, maxIndex;
      // Знаходимо найбільше значення n раз в orderedArrTemp і зберігаємо його індекс масиву
      // Помістимо його в newOrderedArr; потім присвоїмо максимальне значення -1
      for(int i = 0; i < orderedArr.length; i++){
         maxIndex = 0;
         max = orderedArrTemp[maxIndex];
         for(int j = 0; j < orderedArr.length; j++){
            if(orderedArrTemp[j] == -1)
               continue;
            if(orderedArrTemp[j] > max){
               max = orderedArrTemp[j];
               maxIndex = j;
            }
         }
         newOrderedArr[i] = maxIndex;
         orderedArrTemp[maxIndex] = -1;
      }
   }
```

Після визначення порядку доступу кожної вершини викликаємо рекурсивну функцію DfsAlgOfKosarajuVISIT відповідно до порядку вершин в newOrderedArr, щоб виконати пошук в глибину на зворотному графі. Перед кожним викликом створюється новий зв'язний список, як вузол зв'язаного списку singleLinkedList, відповідний новій сильно зв'язаній гілці.

```
/*
Рекурсивна функція, відправною точкою є граф глибокого пошуку G,
коли вузол викликає цю функцію рекурсивно, послідовність доступу вузла
визначається за допомогою orderedArr для вузлів v1, v2, ..., які підключені до вузла node
і позначені білим кольором.
Спочатку відвідуємо вузол vi з найбільшим orderedArr [vi]. Коли немає вузла, підключеного до
вузла node, або всіх вузлів, підключених до node.
Вузли більше не білі. У цей час виходить зв'язна область, і функція повертає результат
```

```
*/
    public static void DfsAlgOfKosarajuVISIT(GraphLink G, int orderedArr[],
                        int node, GraphNodeSingleLink  singleLinkedList){
        // Відвідуємо node і позгачаємо його колір як сірий
        color[node] = COLOR.GRAY;
        // Додаємо вузол node в поточну область
        singleLinkedList.append(new ElemItem<Integer>( node));
        // Спочатку підрахуємо кількість вузлів, підключених до вузла білого кольору.
        int counter = 0;
        for(Edge i = G.firstEdge(node);

G.isEdge(i ); i  = G.nextEdge(i )){
            if(color[w.getV2()] == COLOR.WHITE)
                counter++;
        }
        // Якщо до вузла не підключений ні один вузол і в цей час колір білий, виходимо з функції
        if(counter == 0)
            return;
        // В іншому випадку тимчасово збережіть білий вузол, підключений до вузла в масиві e
        Edge edges[] = new Edge[counter];
        counter = 0;
        for(Edge i = G.firstEdge(node); G.isEdge(i); w = G.nextEdge(i)){
            if(color[i.getV2()] == COLOR.WHITE)
                edges [counter++] = i;
        }
        /* Відсортуємо масив e відповідно до порядку доступу кінцевих точок ребер orderedArr [..]
            Тут ми використовуємо вибіркове сортування для завершення цього процесу */
        int maxIndex;
        for(int i = 0; i < edges.length - 1; i++){
            // Знаходимо i-й за величиною елемент в i-му раунді
            maxIndex = i;
            for(int j = i + 1; j < edges.length; j++){
                if(orderedArr[e[j].getV2()] > orderedArr[e[maxIndex].getV2()]){
                    maxIndex = j;
                }
```

```
        }
        // Якщо вихідна i-та позиція не найбільша, виконуємо операцію обміну
        if(maxIndex != i){
            Edge t = edges[i];
            edges[i] = edges[maxIndex];
            edges[maxIndex] = t;
        }
    }
    // Рекурсивно викликати відсортовані ребра один за іншим
    for(int i = 0; i < edges.length; i++)
        DfsAlgOfKosarajuVISIT(G, orderedArr, edges [i].getV2(), singleLinkedList);
}
```

При рекурсивному пошуку сусідніх вершин вершині node, в функції зберігаємо всі сусідні вершини (ребра) вершині node, що не були відвідані, в масив ребер e. Потім змінюємо порядок ребер в цьому масиві. Правило впорядкування полягає в знаходженні часу відправлення вершин у початковому графі за допомогою алгоритму пошуку в глибину.

За умови, що граф описаний з використанням списку суміжності, алгоритм Косараджу виконує два повних обхода графа і тому виконується за $O(V + E)$ (лінійний) час, який є оптимальним, тому що існує відповідна нижня межа (будь-який алгоритм повинен обходити всі вершини і ребра). Це концептуально найпростіший і одночасно ефективний алгоритм.

3.4 Алгоритм знаходження простих циклів в орієнтованому графі

Для пошуку простих циклів в орієнтованому графі використовувався алгоритм Джонсона. Алгоритм виконується за $O((n + e)(c + 1))$ час і використовує $O(n + e)$ місця для зберігання.

Побудований алгоритм на основі поняття, що прості цикли складаються з кореневої вершини s у підграфі, індукованому s та вершин більших за s у певному

впорядкуванні вершин. Таким чином, відповідь групується за найменшими вершинами циклів.

Щоб уникнути дублювання циклів, вершина v блокується коли додається до деяких елементарних шляхів, що починаються в s. Вона залишається заблокованою до тих пір, поки кожен шлях від v до s не перетне поточний елементарний шлях у вершині, відмінній від s. Крім того, вершина не стає кореневою, для побудови елементарних шляхів, якщо вона не є найменшою вершиною хоча б в одному елементарному ланцюзі. Ці дві особливості запобігають безрезультатному пошуку, який присутній в інших алгоритмах.

Алгоритм приймає граф G, представлений структурою суміжності A_G, яка являє собою поєднання списку суміжності $A_G(v)$ для кожного $v \in V$. Список $A_G(v)$ містить u, якщо ребро $(v, u) \in E$. Алгоритм передбачає, що вершини представлені цілими числами від 1 до n.

Алгоритм виконується завдяки побудові елементарних шляхів з s. Вершина поточного елементарного шляху зберігаються в стеку. Вона додається до шляху викликом процедури CIRCUIT і видаляється при поверненні з виклику. Коли вершина v додається до шляху, вона блокується, встановивши параметр заблоковано $(v) = true$, так що v не можна використовувати двічі на одному й тому ж шляху. По поверненню з виклику, який блокує v, причому v не обов'язково розблоковується. Розблокування завжди затримано настільки, що будь-які два розблокування v розділені або виходом з нового циклу, або повернення до основної процедури виконання.

```
begin
    integer list array A_K(n), B(n); logical array blocked (n); integer s;
    logical procedure CIRCUIT (integer value v);
        begin logical f;
            procedure UNBLOCK (integer value u);
                begin
                    blocked (u) := false;
                    for w∈B(u) do
                        begin
                            delete w from B(u);
                            if blocked(w) then UNBLOCK(w);
                        end
                end UNBLOCK;
            f := false;

        stack v;
        blocked(v) := true;
L1:     for w∈A_K(v) do
            if w=s then
                begin
                    output circuit composed of stack followed by s;
                    f := true;
                end
            else if ¬blocked(w) then
                    if CIRCUIT(w) then f := true;
L2:     if f then UNBLOCK(v)
        else for w∈A_K(v) do
                    if v∉B(w) then put v on B(w);
        unstack v;
        CIRCUIT := f;
    end CIRCUIT;
empty stack;
s := 1;
```

```
begin
    A_K := adjacency structure of strong component K with least
           vertex in subgraph of G induced by {s, s+1, · · · , n};
    if A_K ≠ ∅ then
        begin
            s := least vertex in V_K;
            for i∈V_K do
                begin
                    blocked(i) := false;
                    B(i) := ∅;
                end;
L3:         dummy := CIRCUIT(s);
            s := s+1;
        end
    else s := n;
end
```

Рисунок 3.6 – Алгоритм Джонсона

Коректність алгоритму залежить від того, яка вершина не заблокована коли існує шлях від вершини до s, який перетинає стек лише в s. З іншого боку, обмеження часу роботи залежить від усіх вершин, що залишаються заблокованими допоки можливо, відповідно до вимог коректності. Список B використовується для запам'ятовування інформації, отриманої в результаті пошуку частин графа, які не утворюють елементарних циклів. Процедура UNBLOCK має властивість: якщо є виклик UNBLOCK(x) і вершина y знаходиться у списку B(x), тоді буде викликано UNBLOCK(y), після чого blocked(y) буде помилковим.

3.5 Задача перерозподілу вступників

Маємо поданий орієнтований мультиграф вершини якого - це спеціальність на факультеті навчального закладу, а дуги – побажання студента змінити поточне місце розподілу на більш пріоритетне для нього.

Наша задача знайти множину абітурієнтів, яким можна запропонувати змінити поточне місце розподілу визначене алгоритмом адресного розміщення, тобто помінятися місцями з іншим студентом, так щоб кожен потрапив на більш пріоритетне місце, без змін кількості студентів на спеціальностях.

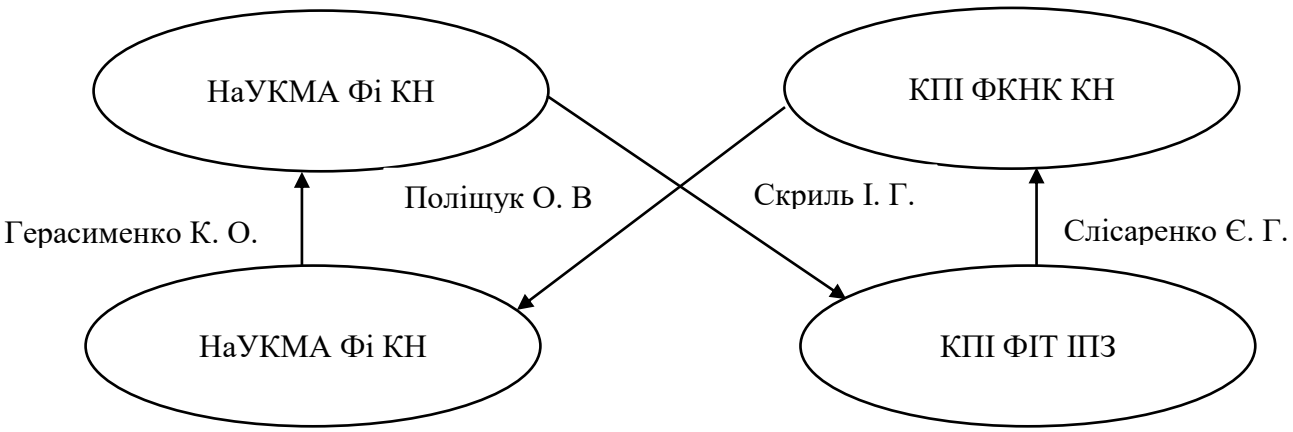

Рисунок 3.7 – Приклад циклу замін студентів

З прикладу на рисунку 3.7 бачимо, що є чотири студента, які б бажали змінити місце розподілу. Ми можемо задовольнити їхні побажання виконавши обмін між університетами і факультетами.

Отже, задача перерозподілу полягає в знаходженні множини циклів без спільних ребер максимальної довжини в заданому орієнтованому мультиграфі.

Так як кількість студентів, які бажають змінити своє місце розподілу, може досягати десятки тисяч, то вихідний граф буде мати завеликі розміри. Тому ми розбиваємо початковий граф на компоненти сильної зв’язності, використовуючи алгоритм Косараджу, який наведений в розділі 3.3. В результаті ми отримаємо масив сильно зв’язних графів. Далі на кожному з них потрібно знайти множину циклів без спільних дуг максимальної довжини.

Так як кількість студентів на спеціальностях має залишатися незмінна, то для кожної вершини, із знайдених циклів, кількість вхідних і вихідних ребер має бути однакова.

Дана задача зводиться до того, що в сильно зв’язному графі $G = (V; E)$ потрібно знайти підмножину вершин, де жодні дві не є суміжними.

3.6 Алгоритм розв’язання задачі

Спершу розглянемо алгоритм пошуку циклів в сильно зв’язному графі:

1. Маємо сильно зв’язний мультиграф $G = (V; E)$.
2. Шукаємо множину простих циклів C в графі G за допомогою алгоритму Джонсона.
3. В масиві C шукаємо множину циклів без спільних дуг C_{opt}.
4. C_{opt} – шукана множина переходів.

Найгірша часова складність алгоритму пошуку простих циклів Джонсона складає $O((V + E)C)$, де V – кількість вершин, E – кількість ребер і C – кількість простих циклів у графі. Для пошуку множин циклів без спільних дуг використовується цикл for всередині іншого циклу for - $O(C^2)$.

Загальний алгоритм розв'язання задачі перерозподілу вступників має вигляд:

1. З вхідних даних генеруємо мультиграф переходів $G = (V; E)$, де V – множина вершин (спеціальність навчального закладу), E – ребер (бажані переходи вступників).
2. Розбиваємо граф G на компоненти сильної зв’язності, отримуємо масив з n графів $GG[\,]: \{G_i = (V_i; E_i) | i = \underline{1, n}\}$.
3. Для кожного графа з масиву GG виконуємо алгоритм пошуку циклів в сильно зв’язному графі наведений на почаику підрозділу і знаходимо C_{opt_i}.
4. З’єднуємо отримані результати $\mathrm{C}_{opt} = \bigcup_{i=1}^{n} \mathrm{C}_{opt_i}$.
5. C_{opt} – шукана множина переходів.

4. Практичне застосування алгоритму

4.1 Дані вступної кампанії 2020

Для отримання даних вступної кампанії 2020 року була розроблена програма парсингу рейтингових списків студентів з ресурсу abit-poisk.org.ua, який в свою чергу використовує інформацію, взяту з ресурів Vstup.edbo.gov.ua, IC Конкурс та Vstup.osvita.

Парсинг сторінок з інформацією про навчальні заклади, напрямки і списки зарахованих студентів відбувався з допомогою бібліотеки jsoup. Однак дана бібліотека не надає доступу до динамічно завантажуваного контенту, яким виявилася сторінка де знаходиться інформація про бажані переходи студентів, тому додатково використовувалася бібліотека Selenium.

Отримані дані зберігаються в MySQL базі даних (рисунок 4.1), яка складається з чотирьох сутностей: університет, напрямок, студент та бажані переходи студентів. Скрипт для створення наведено в додатку А.

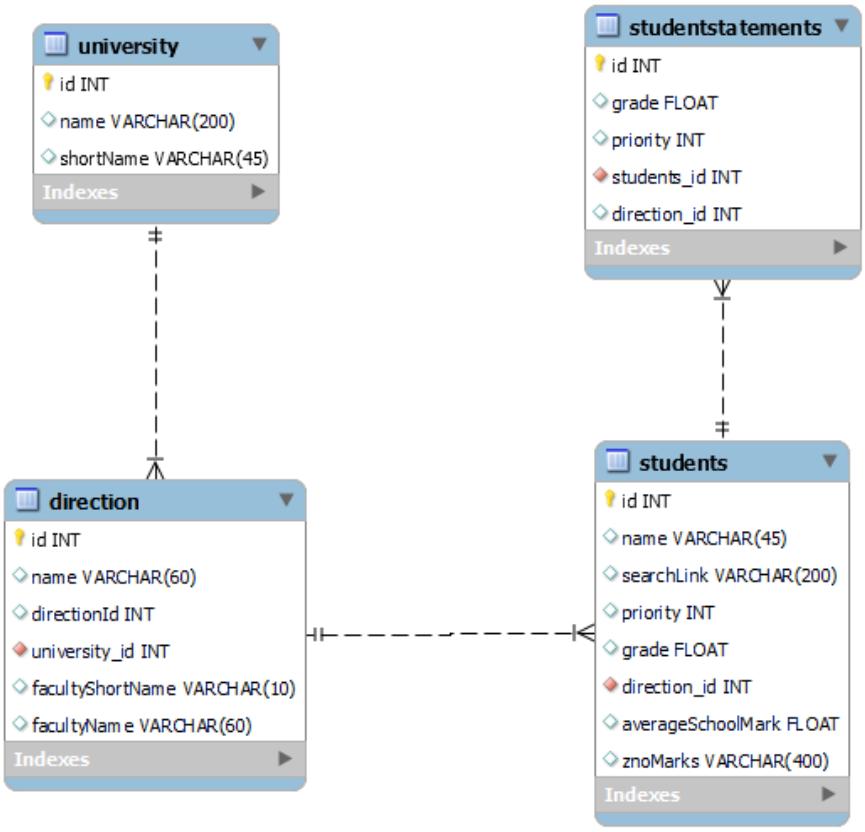

Рисунок 4.1 - MySQL база даних програми

Було розпарсено дані трьох університетів.

Таблиця 4.1 – Дані університетів

id	name	shortName
41	Київський національний університет імені Тараса Шевченка	КНУТШ
79	Національний університет "Києво-Могилянська академія"	НаУКМА
174	Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»	КПІ ім. Ігоря Сікорського

Для кожного з університетів отримано дані по напрямках:

1. 113 Прикладна математика
2. 121 Інженерія програмного забезпечення
3. 122 Комп'ютерні науки
4. 125 Кібербезпека
5. 126 Інформаційні системи та технології

Таблиця 4.2 – Дані напрямків

id	name	directionId	university_id	facultyShortName	facultyName
675589	Інженерія програмного забезпечення	121	41	ФКНК	Факультет комп`ютерних наук та кібернетики
676415	Інформаційні системи та технології	126	41	ФІТ	Факультет інформаційних технологій
698753	Інженерія програмного забезпечення	121	79	ФІ	Факультет інформатики
715657	Комп'ютерні науки	122	174	ТЕФ	Теплоенергетичний факультет
715733	Інженерія програмного забезпечення	121	174	ТЕФ	Теплоенергетичний факультет
715769	Інженерія програмного забезпечення	121	174	ФІОТ	Факультет інформатики та обчислювальної техніки
715770	Інженерія програмного забезпечення	121	174	ФПМ	Факультет прикладної математики
719169	Інформаційні системи та технології	126	174	ФІОТ	Факультет інформатики та обчислювальної техніки

732914	Кібербезпека	125	41	ФІТ	Факультет інформаційних технологій
737688	Прикладна математика	113	79	ФІ	Факультет інформатики
740947	Комп'ютерні науки	122	41	ФІТ	Факультет інформаційних технологій
740948	Комп'ютерні науки	122	41	ФКНК	Факультет комп`ютерних наук та кібернетики
742411	Прикладна математика	113	174	ФПМ	Факультет прикладної математики
742412	Прикладна математика	113	174	ФТІ	Фізико-технічний інститут
746063	Комп'ютерні науки	122	79	ФІ	Факультет інформатики
747440	Комп'ютерні науки	122	41	ФІТ	Факультет інформаційних технологій
747441	Комп'ютерні науки	122	41	ФІТ	Факультет інформаційних технологій
762463	Комп'ютерні науки	122	174	ІПСА	Інститут прикладного системного аналізу
762464	Комп'ютерні науки	122	174	ФБІ	Факультет біомедичної інженерії
764311	Прикладна математика	113	41	ФКНК	Факультет комп`ютерних наук та кібернетики

В результаті знайдено 624 студента, які потрапили на бюджетні місця не за першим пріоритетом, приклад даних наведено в таблиці 4.3.

Таблиця 4.3 –Деякі дані студентів

id	name	searchLink	priority	grade	direction_id	averageSchoolMark	znoMarks
1	Поліщук О. В.	/#search-%D0%9F%D0%BE%D0%BB%D1%96%D1%89%D1%83%D0%BA+%D0%9E.+%D0%92.+2020+11.1	2	194,8	675589	11,1	[{"name":"Українська мова та література","value":194.0},{"name":"Математика","value":195.0},{"name":"Іноземна мова","value":195.0}]
2	Герасименко К. О.	/#search-%D0%93%D0%B5%D1%80%D0%B0%D1%81%D0%B8%D0%BC%D0%B5%D0%BD%D0%BA%D0%BE+%D0%9A.+%D0%9E.+2020+10.9	2	194,8	675589	10,9	[{"name":"Українська мова та література","value":185.0},{"name":"Математика","value":199.0},{"name":"Іноземна мова","value":192.0}]
3	Телепенко І. Ю.	/#search-%D0%A2%D0%B5%D0%BB%D0%B5%D0%BF%D0%B5%D0%BD%D0%BA%D0%BE+%D0%86.+%D0%AE.+2020+11.4	2	194,3	675589	11,4	[{"name":"Українська мова та література","value":197.5},{"name":"Математика","value":195.0},{"name":"Іноземна мова","value":189.0}]

4	Слісаренко Є. Г.	/#search-%D0%A1%D0%BB%D1%96%D1%81%D0%B0%D1%80%D0%B5%D0%BD%D0%BA%D0%BE+%D0%84.+%D0%93.+2020+10.5	3	193,8	675589	10,5	[{"name":"Українська мова та література","value":188.0},{"name":"Математика","value":199.0},{"name":"Фізика","value":184.0}]
5	Димарчук Д. В.	/#search-%D0%94%D0%B8%D0%BC%D0%B0%D1%80%D1%87%D1%83%D0%BA+%D0%94.+%D0%92.+2020+11.2	4	190,4	676415	11,2	[{"name":"Українська мова та література","value":193.0},{"name":"Математика","value":190.0},{"name":"Іноземна мова","value":189.0}]

А також список бажаних переходів (місця з більшим пріоритетом за поточне місце розподілу), що складається з 1059 елементів, приклад даних наведено в таблиці 4.4.

Таблиця 4.4 – Деякі дані бажаних переходів студентів

id	grade	priority	students_id	direction_id
1	194,4	1	1	683628
2	193,8	1	2	698753
3	193,6	1	3	732916
4	192,4	1	4	698753
5	192,4	2	4	746063

4.2 Результати роботи створеної програми

В результаті виконання створеної програми, на основі даних наведених в попередньому розділі, було побудовано орієнтовний мультиграф, вершини якого – це напрямки в університетах, а дуги – бажання студентів змінити поточне місце зарахування.

Використовуючи алгоритм, наведений у розділі 3.6 (Алгоритм розв’язання задачі), граф було розбито на компоненти сильної звэ’язності. Отримано 13 підграфів, а потім знайдено і проаналізовано цикли в кожному з них. Дані цикли - множина переходів, яку можуть здійснити студенти. При цьому кількість

бюджетних місць в університетах залишиться без зміни. Деякі результати роботи створеної програми наведені на рисунках 4.2 - 4.6.

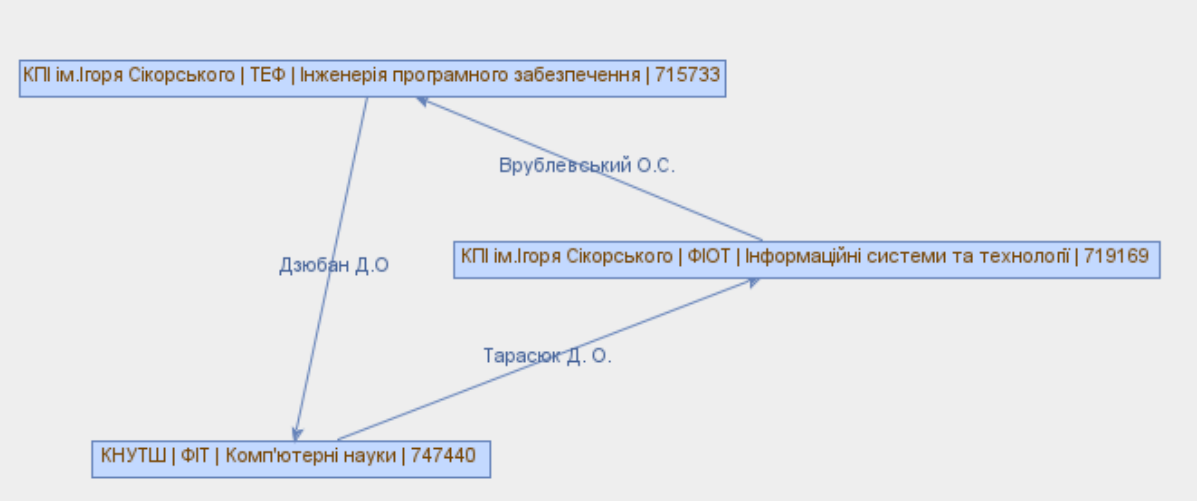

Рисунок 4.2 – Приклад результатів виконання програми

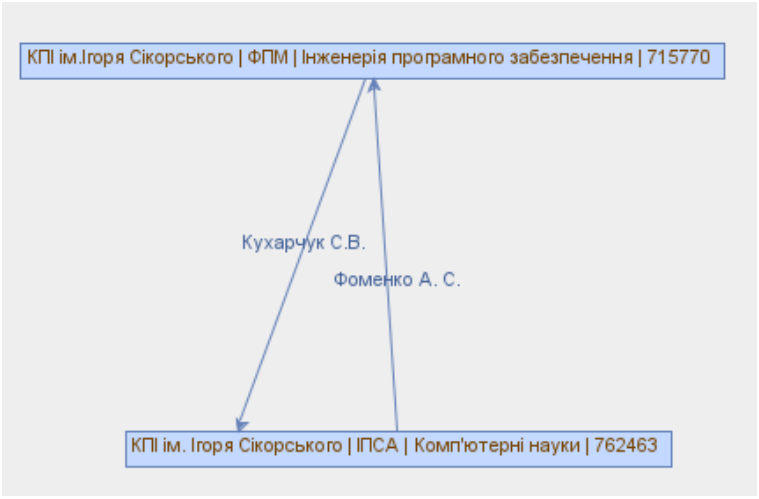

Рисунок 4.3 – Приклад результатів виконання програми

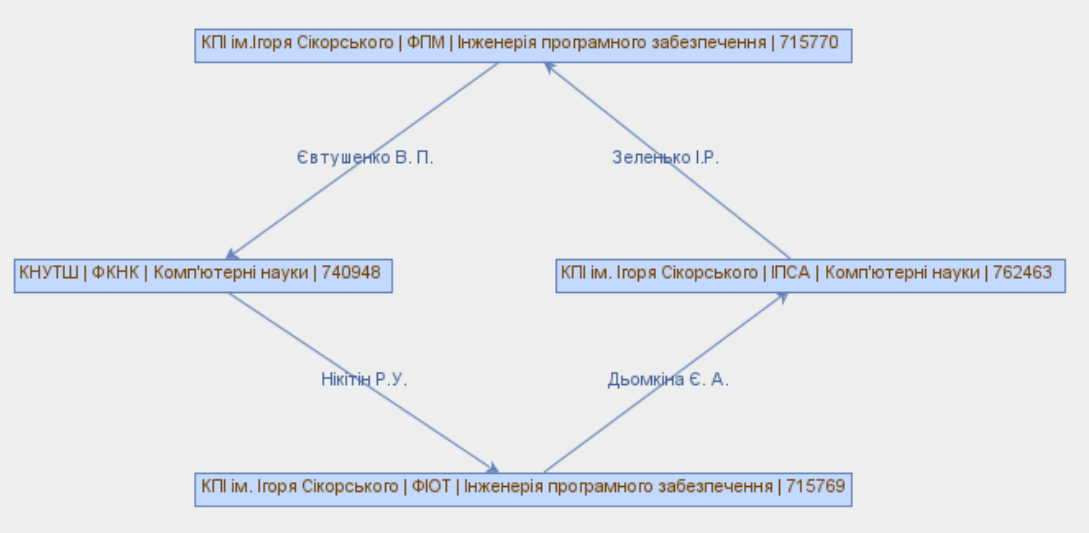

Рисунок 4.4 – Приклад результатів виконання програми

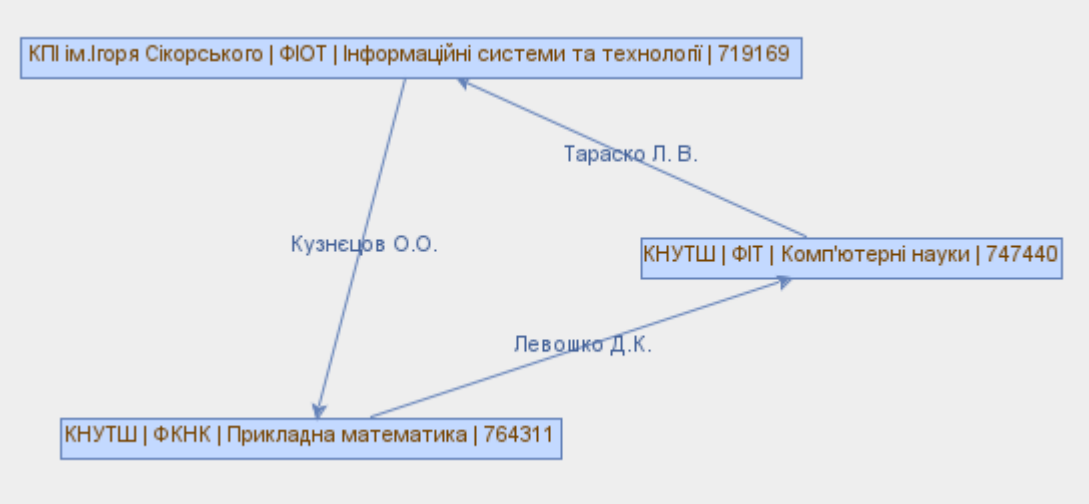

Рисунок 4.5 – Приклад результатів виконання програми

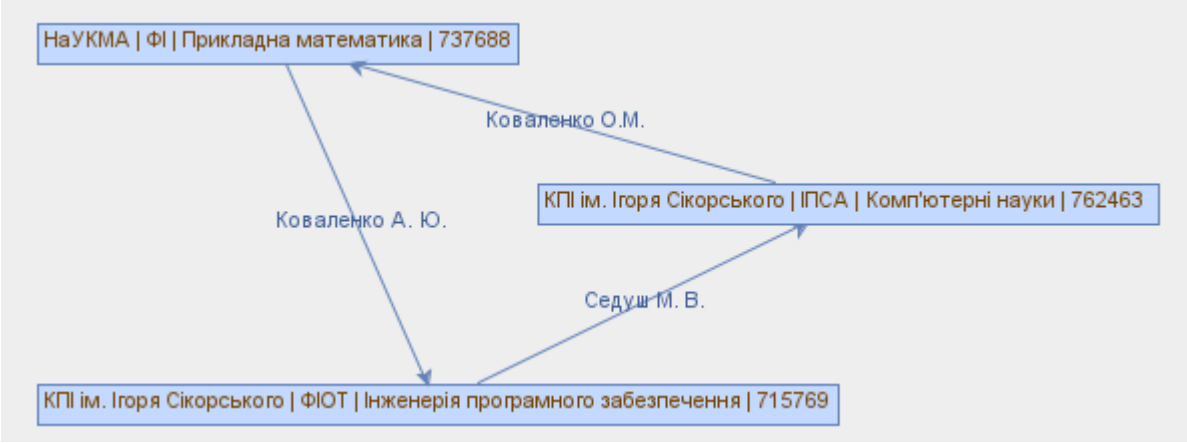

Рисунок 4.6 – Приклад результатів виконання програми

Висновки

Під час виконання роботи був досліджений процес формування рейтингових списків і розглянутий алгоритм адресного розміщення державного та регіонального замовлення 2020, який розроблений на основі алгоритму пошуку стабільних пароутворень - алгоритм Гейла-Шеплі.

Алгоритм Гейла-Шеплі забезпечує правдивий механізм з точки зору груп які роблять вибір (університетів). Тобто, жоден член групи не може домогтися кращої відповідності для себе, спотворюючи свої переваги. Але це не стосується членів другої групи (студентів), вони не завжди отримують найбажанішу пару. В свою чергу, задача розподілу вступників додала певні обмеження і модифікації до алгоритму:

1. студент може вибрати тільки 5 навчальних закладів
2. навчальний заклад вибирає не одного студента, а деяку обмежену їх кількість

Дані доповнення призводять до певного спотворення уподобань студентів. Тому що, вищі навчальні заклади надають перевагу студентам , які мають вищі бали, в той час, як вступники малоймовірно будуть допущені до вузу, якому вони віддали перевагу згідно з власними інтересами, тому що вони виберуть більш реалістичні напрямки, в яких вони будуть займати вищі місця.

Так в результаті виконання алгоритму адресного розміщення державного та регіонального замовлення, який використовується, на сьогоднішній день в Україні, для формування рейтингових списків, існують деякі студенти які незавжди задоволені місцем свого розподілу.

Для вирішення даної проблеми був запропонований і розроблений алгоритм перерозподілу вступників до вищих навчальних закладів на місця державного замовлення. Він знаходить студентів, яким можна запропонувати покращити їх місце розподілу, відповідно до визначених пріоритетів. І в результаті виконання замін, запропонованих даною програмою, студенти потраплять на бажаніші місця

навчання. При цьому, це не вплине негативно ні на списки бюджетників, ні на роботу державного алгоритму адресного розміщення.

Список літератури

1. Про затвердження Умов прийому на навчання для здобуття вищої освіти в 2021 році [Електронний ресурс]: Наказ МОН № 1274 від 15.10.2020 року. – Режим доступу до ресурсу: http://osvita.ua/legislation/Vishya_osvita/9990/.

2. Рейтинговий відбір вступників у 2021 році. [Електронний ресурс] – Режим доступу до ресурсу: https://osvita.ua/consultations/bachelor/43472/.

3. Додаток 6 до Умов прийому на навчання до закладів вищої освіти України в 2020 році (пункт 5 розділу IX) [Електронний ресурс]: Наказ МОН № 1285 від 11.10.2019 року. – Режим доступу до ресурсу: https://mon.gov.ua/storage/app/media/vishcha-osvita/vstup-2020/2019/12/0umovi-priyomu2020chinni.docx.

4. Зубова В. В. Теорія матчингу в економіці: прикладні аспекти застосування. [Електронний ресурс] / АГРОСВІТ № 13—14, 2019. С. 53 - 59. – Режим доступу до ресурсу: http://www.agrosvit.info/pdf/13-14_2019/9.pdf.

5. Боднарчук Ю.В., Олійник Б.В. Основи дискретної математики [Електронний ресурс]: навч. посіб. Київ - 2007. С. 111 - 120. – Режим доступу до ресурсу: https://www.ukma.edu.ua/~bogd/Discrete%20Mathematics/PosibnykNew.pdf.

6. Кузьменко В. В., Швачич Г. Г., Рижанкова Г. І, Пасинков В. М. Основи дискретної математики. Розділ “Елементи теорії графів” [Електронний ресурс]: Конспект лекцій. – Дніпропетровськ: НМетАУ, 2004. – 38с. – Режим доступу до ресурсу: https://nmetau.edu.ua/file/072.pdf.

7. Balinski M. and Sonmez T. A tale of two mechanism: Student placement. J. Econom. Theory, 84 (1): 73—94, 1999.

8. Дискретна Математика: Зв’язність графів. Тема 26. Зв’язність графів. С. 152 – 156. [Електронний ресурс] – Режим доступу до ресурсу: https://evgavrilenko.ucoz.ru/DS/LEKCIYA_3.pdf.

9. Roberto-Medina A. Implementation of stable solution in a restricted matching market. Review of Economic Design, 3 (2): 137—147, 1998.

10. Сильно связанный компонентный алгоритм - алгоритм Косараджу, алгоритм Тарьяна и алгоритм Габоу [Електронний ресурс] – Режим доступу до ресурсу: https://russianblogs.com/article/99001565235/.

11. D. B. Johnson, Finding all the elementary circuits of a directed graph, SIAM J. Comput., 4 (1975), pp. 77-84.

Додаток А

Скрипт створення MySQL бази даних

```
-- MySQL Script generated by MySQL Workbench
-- Wed May 26 16:32:17 2021
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_
ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema abitpoisk
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema abitpoisk
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `abitpoisk` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `abitpoisk` ;

-- -----------------------------------------------------
-- Table `abitpoisk`.`university`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `abitpoisk`.`university` (
  `id` INT NOT NULL,
  `name` VARCHAR(200) NULL DEFAULT NULL,
  `shortName` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `abitpoisk`.`direction`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `abitpoisk`.`direction` (
  `id` INT NOT NULL,
  `name` VARCHAR(60) NULL DEFAULT NULL,
  `directionId` INT NULL DEFAULT NULL,
  `university_id` INT NOT NULL,
  `facultyShortName` VARCHAR(10) NULL DEFAULT NULL,
  `facultyName` VARCHAR(60) NULL DEFAULT NULL,
```

```
  PRIMARY KEY (`id`),
  INDEX `fk_direction_university_idx` (`university_id` ASC) VISIBLE,
  CONSTRAINT `fk_direction_university`
    FOREIGN KEY (`university_id`)
    REFERENCES `abitpoisk`.`university` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `abitpoisk`.`students`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `abitpoisk`.`students` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NULL DEFAULT NULL,
  `searchLink` VARCHAR(200) NULL DEFAULT NULL,
  `priority` INT NULL DEFAULT NULL,
  `grade` FLOAT NULL DEFAULT NULL,
  `direction_id` INT NOT NULL,
  `averageSchoolMark` FLOAT NULL DEFAULT NULL,
  `znoMarks` VARCHAR(400) NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
  INDEX `fk_students_direction1_idx` (`direction_id` ASC) VISIBLE,
  CONSTRAINT `fk_students_direction1`
    FOREIGN KEY (`direction_id`)
    REFERENCES `abitpoisk`.`direction` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 625
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `abitpoisk`.`studentstatements`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `abitpoisk`.`studentstatements` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `grade` FLOAT NULL DEFAULT NULL,
  `priority` INT NULL DEFAULT NULL,
  `students_id` INT NOT NULL,
  `direction_id` INT NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,
  INDEX `fk_studentstatements_students1_idx` (`students_id` ASC) VISIBLE,
  CONSTRAINT `fk_studentstatements_students1`
    FOREIGN KEY (`students_id`)
    REFERENCES `abitpoisk`.`students` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 1074
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```