

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

«МЕТОДИ ОПТИМІЗАЦІЇ

КОМП'ЮТЕРНИХ СИСТЕМ»

для магістрів спеціальності F7 «Комп'ютерна інженерія»

денної форми навчання

Методичні вказівки до виконання лабораторних робіт з дисципліни «Методи оптимізації комп'ютерних систем» для магістрів спеціальності F7 «Комп'ютерна інженерія», денної форми навчання. / Укл. М.Б. Ільяшенко, Т.С. Дьячук – Запоріжжя: НУ «Запорізька політехніка», 2025. - 54с.

Укладачі : М.Б. Ільяшенко, доцент, к.т.н.
 Т.С. Дьячук, асистент

Рецензент : Г.Г. Киричек, доцент, к.т.н.

Відповідальний

за випуск : М.Б. Ільяшенко, доцент, к.т.н.

Дані методичні вказівки до лабораторних робіт розроблені у відповідності з програмою курсу «Методи оптимізації комп'ютерних систем» і призначені для магістрів денної форми навчання.

Затверджено:

На засіданні кафедри

«Комп'ютерні системи та мережі»

Протокол № 2

від 29 серпня 2025 р.

Рекомендовано до видання

НМК факультету КНТ

Протокол № 2

від 10 вересня 2025 р.

ЗМІСТ

С.

Вступ.....	5
1 Лабораторна робота № 1 - Елементарні алгоритми.....	7
1.1 Теоретичні відомості	7
1.2 Завдання до лабораторної роботи	10
1.3 Зміст звіту	13
1.4 Контрольні питання	13
2 Лабораторна робота № 2 - Динамічні інформаційні структури.....	15
2.1 Теоретичні відомості	15
2.2 Завдання до лабораторної роботи	20
2.3 Зміст звіту	24
2.4 Контрольні питання	24
3 Лабораторна робота № 3 - Сортування й пошук	26
3.1 Завдання до лабораторної роботи	26
3.2 Зміст звіту	29
3.3 Контрольні питання	30
4 Лабораторна робота № 4 - Обхід графів.....	31
4.1 Завдання до лабораторної роботи	31
4.2 Зміст звіту	39
4.3 Контрольні питання	39
5 Лабораторна робота № 5 - Задачі на графах.....	40
5.1 Завдання до лабораторної роботи	40
5.2 Зміст звіту	46
5.3 Контрольні питання	46
6 Лабораторна робота № 6 - Метод гілок та меж, жадібні алгоритми, динамічне програмування	47
6.1 Завдання до лабораторної роботи	47
6.2 Зміст звіту	51

6.3 Контрольні питання	51
Перелік джерел посилання	53

ВСТУП

Методичні матеріали з дисципліни «Методи оптимізації комп'ютерних систем» спрямовані на формування у студентів знань і навичок алгоритмічної оптимізації програмного забезпечення. У сучасних умовах розвитку інформаційних технологій усе більшого значення набуває не лише правильність роботи програм, а й ефективність використання часу та обчислювальних ресурсів. Тому курс орієнтований на формування вмінь створювати програмні рішення, що відрізняються високою продуктивністю, надійністю та раціональністю, а також на розвиток здатності до системного аналізу та порівняння різних підходів до розв'язання задач.

Посібник містить шість лабораторних робіт, кожна з яких присвячена окремому класу задач та алгоритмів. Тематика завдань охоплює широкий спектр практично значущих проблем, що виникають у процесі розробки програмних систем. Базовою мовою програмування обрано C++, який забезпечує можливість безпосереднього контролю роботи з пам'яттю, гнучке керування ресурсами та високу продуктивність при реалізації алгоритмів. Це дозволяє студентам поєднати теоретичні знання з практичним досвідом оптимізації й глибше зрозуміти закономірності взаємозв'язку між структурою алгоритму та ефективністю його виконання.

У кожному завданні наведено приклади вхідних і вихідних даних, які ілюструють постановку задачі та результати її розв'язання. Очікується, що студент доповнить їх власними тестами для більш повної перевірки коректності й надійності реалізованої програми. При оцінюванні результатів враховується не лише правильність обчислень, але й оптимальність реалізації — час виконання, ефективність використання пам'яті та інших ресурсів, оскільки саме ці аспекти є центральними для дисципліни.

Методичні матеріали покликані допомогти студентам не тільки оволодіти практичними прийомами алгоритмічної оптимізації, але й розвинути

алгоритмічне мислення, вміння оцінювати ефективність програмних рішень та здатність до подальшого вдосконалення професійних навичок у сфері інформаційних технологій.

1 ЛАБОРАТОРНА РОБОТА № 1 - ЕЛЕМЕНТАРНІ АЛГОРИТМИ

Мета роботи: одержання навичок написання ефективних програм з використанням алгоритмів з низкою обчислювальною складністю [1-6, 8].

1.1 Теоретичні відомості

Консольний ввід-вивід

Для всіх завдань курсу, якщо інше не зазначене в лабораторній роботі окремо, використовується консольне введення-вивід даних. Консольне введення припускає, що дані вводяться із клавіатури й виводяться на дисплей монітора. Для консольного введення-виводу використовуються функції мови C++:

Бібліотека `stdio.h`

- `printf`, `vprintf` - використовуються для виводу в стандартний потік виводу;
- `scanf`, `vscanf` використовуються для введення зі стандартного потоку введення.

Крім того, можна використовувати всі стандартні функції роботи з файлами, указуючи в якості файлу для читання стандартний потік введення – `stdin`, а для виводу стандартний потік виводу – `stdout`, наприклад:

- `fscanf(stdin,"%d",&N);` - читання зі стандартного потоку введення;
- `fprintf(stdout,"%ld",sum);` - запис у стандартний потік виводу.

У такий же спосіб використовуються функції `fread`, `fwrite` і `foef`.

Бібліотека `conio.h`

- `int getch(void)` - Зчитує символ прямо з консолі без використання буфера й `echo`-виводу;
- `int putch(int c)` - Функція `_putch` виводить символ `c` у консоль без буферизації. У випадку успіху повертає `c`. А якщо ні, то повертає `EOF`.

Для роботи з консоллю необхідно використовувати командний рядок. Запуск командного рядка здійснюється через Пуск – Виконати, у вікні запуску ввести команду `cmd`. Відкриється інтерфейс командного рядка.

Альтернативою є використання `Far`. Для того, щоб забрати обидві панелі й переглянути результати виводу на екрані, необхідно натиснути `Ctrl+O`.

Перенаправлення консольного виводу у файл здійснюється за допомогою `>` або `>>`. Команда `>` перенаправляє консольний вивід з екрана у файл, при цьому файл створюється заново (тобто якщо він існував, то всі дані у файлі будуть загублені). Команда `>>` теж перенаправляє консольний вивід у файл, але якщо файл існував, то дані виводу дописуються в кінець файлу. Обидві команди створюють файл якщо він не існував.

Перенаправлення консольного введення здійснюється командою `<`. При цьому необхідно, щоб файл, зазначений у якості параметра команди вже існував.

Приклади використання:

а) Запустити програму `lab1_1.exe`, при цьому здійснювати введення даних з файлу `input.txt`, а вивід перенаправляти у файл `output.txt` (`output.txt` буде створений знову):

```
lab1_1.exe <input.txt >output.txt
```

б) Запустити програму `lab1_2.exe`, при цьому введення не перенаправляється, а результати дописуються у файл `outbuffer.dat`

```
lab1_2.exe >>outbuffer.dat
```

Перенаправлення виводу однієї програми на вхід іншої здійснюється за допомогою команди `|`. Наприклад виконати програму `lab1_3.exe`, консольне введення для якої перенаправити із файлу `input.txt`, а вивід перенаправляти на вхід програми `lab1_4.exe`, вивід якої перенаправляти у файл `output.txt`:

```
lab1_3.exe <input.txt | lab1_4.exe >output.txt
```

Визначення часу виконання програми

Завданням курсу є одержання студентами навичок написання оптимальних по продуктивності й споживанню ресурсів програм. Для оцінки продуктивності програм необхідно вміти досить точно визначати час її виконання.

Для досить тривалих проміжків часу можна використовувати функцію `clock()` мови C++. `clock()` - повертає число тиків таймера з моменту запуску програми. Константа `CLOCKS_PER_SEC` (у нових версіях компіляторів - `CLK_TCK`) містить число тиків таймера в секунду. Таким чином, найпростіший спосіб визначення часу виконання програми може бути реалізований у такий спосіб:

Лістинг 1.1 - Приклад визначення часу виконання програми

```
#include <time.h>

clock_t start, end;

double cpu_time_used;

start = clock();

... /* Тіло програми. */

end = clock();

cpu_time_used = ((double) (end-start)) / CLOCKS_PER_SEC;
```

Для всіх завдань лабораторної роботи необхідно визначити час виконання програми і вивести його на екран.

Завдання із зірочкою:

Завдання, які помічені зірочкою мають підвищену складність і не є обов'язковими, але виконання враховується під час заліку.

1.2 Завдання до лабораторної роботи

1.2.1 Проблема з A і B

Поміняти значення змінних A і B між собою, не заводячи додаткових змінних. Вхідний файл input.txt містить числа a і b ($0 \leq a, b \leq 32767$). У вихідному файлі output.txt повинні міститися значення цих змінних після обміну.

1.2.2 Трамвайні квитки

Написати програму визначення кількості шестизначних "щасливих" трамвайних квитків, у яких сума перших трьох цифр збігається із сумою трьох останніх. Оптимізувати розв'язок за часом виконання. Кількість квитків вивести у файл output.txt.

1.2.2.1 Трамвайні квитки з довільною довжиною номера*

Вхідний файл input.txt містить у кожному рядку одне число N. Необхідно вивести у файл output.txt для кожного вихідного числа N число «щасливих» квитків, що складаються з $2 \cdot N$ цифр.

1.2.3 Копіювання

В офіс поступила нова комп'ютерна програма. Вам, як адміністраторові, треба скопіювати її на всі N наявних комп'ютерів. Зараз програма є тільки на одному комп'ютері, інші комп'ютери не мають дисководів і не об'єднані в локальну мережу. Єдиний спосіб передати інформацію з одного комп'ютера на іншій — скопіювати її, використовуючи нуль-модем (дріт, що з'єднує два комп'ютери безпосередньо). Таким чином, з будь-якого комп'ютера, де вже встановлена програма можна скопіювати її на якийсь іншій (але тільки на один) усього за одну годину. В офісі є всього K нуль-модемних дротів. Ваше завдання по заданих числах N і K оцінити мінімальний час, необхідний для копіювання програми на всі наявні комп'ютери.

Вихідні дані:

У єдиному рядку входу перебувають цілі числа $0 < N \leq 10^9$ і $0 < K \leq 10^9$, розділені пробілом.

Результат:

Вивести мінімальний час (у годинах), що вимагається для копіювання програми на всі комп'ютери.

Приклад:

Вхідні дані	Результат
8 3	4

1.2.4 Сума послідовних чисел**Вхідні дані:**

У єдиному рядку перебуває ціле число N , $0 < N \leq 10^9$.

Результат:

Слід вивести два цілі позитивні числа A і P , таких, що

- $N = A + (A + 1) + \dots + (A + P - 1)$.
- Серед усіх пар чисел A і P , що задовольняють умові 1, у виведеній парі максимальне значення P .

Приклад:

Вхідні дані	Результат
14	2 4

1.2.5 Довжина нитки

У площину вбито N цвяхів. Усі цвяхи мають капелюшки з радіусом R . Усі цвяхи вбиті у вершини опуклого багатокутника. Навколо цвяхів натягнута нитка

(див. рис. 1.1). Визначити довжину нитки, знаючи координати цвяхів і радіус капелюшків.

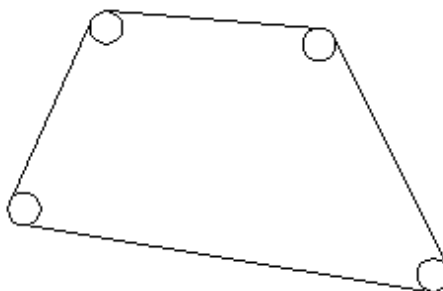


Рисунок 1.1 – Визначити довжину нитки

Вхідні дані:

У першому рядку входу перебувають два числа — кількість цвяхів N , ($1 \leq N \leq 100$), і дійсне число R — радіус капелюшків цвяхів. Усі капелюшки мають однаковий радіус. Далі розташовуються ще N рядків, у кожному з яких записані через пробіл пари дійсних координат центру чергового цвяха; координати не перевершують по абсолютній величині числа 10^5 . Координати цвяхів наводяться в порядку обходу вершин багатокутника за годинниковою стрілкою, починаючи з довільного. Капелюшки різних цвяхів не стикаються.

Результат:

На виході в єдиному рядку повинне знаходитися дійсне число, з точністю два знаки після коми — довжина ниточки, натягнутої навколо всіх цвяхів.

Приклад:

Вхідні дані	Результат
4 1	14.28
0.0 0.0	
2.0 0.0	
2.0 2.0	
0.0 2.0	

1.2.5.1 Довільне розташування цвяхів*

Цвяхи можуть утворювати не опуклий багатокутник, але всі так само перелічуються у вхідному файлі в порядку обходу за годинниковою стрілкою. Нитка натягнута на цвяхи в порядку їх проходження так, що завжди натягнута між сусідніми цвяхами (завжди оминає капелюшки цвяхів таким чином щоб унеможливити провисання). Визначити довжину нитки.

1.3 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Основні функціональні блоки розроблених програм (структурна схема або код програми).
3. Особисті тестові дані для кожного завдання та результати роботи програми для цих даних.
4. Короткі відповіді на основні запитання та висновки за результатами роботи.

1.4 Контрольні питання

1. Що таке часова складність алгоритму?
2. Що таке просторова складність алгоритму?
3. Які алгоритми вважаються ефективними, а які не ефективними (згідно їх обчислювальній складності)?

4. Що таке клас складності P ? Наведіть приклади алгоритмів, які до нього належать.

5. Що таке клас складності NP ? Наведіть приклади алгоритмів, які до нього належать.

6. Яку обчислювальну складність мають запропоновані вами алгоритми розв'язку завдань лабораторної роботи? Обґрунтуйте свою відповідь.

7. Яку просторову складність мають запропоновані вами алгоритми розв'язку завдань лабораторної роботи? Обґрунтуйте свою відповідь.

2 ЛАБОРАТОРНА РОБОТА № 2 - ДИНАМІЧНІ ІНФОРМАЦІЙНІ СТРУКТУРИ

Мета роботи: одержання навичок застосування динамічних інформаційних структур і ефективних алгоритмів роботи з ними.

2.1 Теоретичні відомості

У даній та наступних лабораторних роботах рекомендується використовувати для операцій з динамічними інформаційними структурами бібліотеку STL мови C++, короткий опис та вступ в роботу з якої наведено нижче [7, 12, 13].

Бібліотека STL

Бібліотека STL складається з чотирьох категорій: контейнери, алгоритми, ітератори і функціональні об'єкти (або функтори).

Контейнери — це об'єкти, що зберігають інші елементи і реалізують механізми доступу до них. Прикладами контейнерів є вектори і списки. Кожний контейнер описується шаблонним класом, у якому реалізуються механізми доступу і функція для обробки елементів, що містяться у контейнері. Кожний контейнер має свої ітератори для перебору елементів і функції для їх обробки. Наприклад, клас, що описує роботу із вектором, має відповідний ітератор для прямого доступу до елементів вектору, а також функції для вставки і видалення тощо.

Алгоритми виконують операції над елементами, що містяться у контейнері. Основними алгоритмами є заповнення контейнера за відповідним правилом, сортування за вказаним критерієм, пошук за заданим критерієм, заміна елементів контейнера. Алгоритми визначені в заголовному файлі <algorithm.h>. Нижче приведені імена деяких найбільше часто використовуваних функцій-алгоритмів STL.

Ітератори — це об'єкти, що реалізують концепцію інтелектуального вказівника і дозволяють одержати доступ до елементів контейнера. Оскільки ітератор є аналогом вказівника, до нього можна застосовувати ті ж самі операції, що і до звичайного вказівника: розіменування, інкремента, декремента і порівняння.

Контейнери

Нижче наведені основні контейнери бібліотеки STL які знадобляться вам під час виконання лабораторних робіт:

- **vector** - контейнер, що реалізує прямий доступ до послідовно розташованих елементів (аналог масиву).
- **list** - контейнер, що реалізує концепцію двозв'язного списку, тобто забезпечує доступ до елементів(вузлів), що розташовані у довільних комірках пам'яті і пов'язані між собою відповідними посиланнями (вказівниками на вузли).
- **deque** - контейнер, що реалізує концепцію двосторонньої черги, тобто дозволяє вставляти і видаляти елементи з обох кінців.
- **set** - контейнер, що реалізує концепцію множини елементів, які не повторюються і розташовані у певному порядку.
- **multiset** - контейнер, що реалізує концепцію множини елементів, які можуть повторюватися і розташовані у певному порядку.
- **map** - контейнер, що реалізує концепцію асоціативного масиву, тобто забезпечує доступ до елементів за ключами.
- **multimap** Контейнер, що реалізує концепцію асоціативного масиву, забезпечуючи доступ до елементів за ключами, які можуть повторюватися.
- **stack** - контейнер, що реалізує концепцію стеку - лінійну структуру даних, організовану за принципом LIFO (останнім увійшов — першим вийшов).
- **queue** - контейнер, що реалізує концепцію черги - лінійної структури даних, що додаються і видаляються за принципом FIFO (першим увійшов - першим пішов).
- **priority queue** - контейнер, що реалізує концепцію черги за пріорите.

Алгоритми

Алгоритми розподіляються на ті, що не модифікують елементи контейнера і ті, що модифікують. Бібліотека STL базується на принципі інкапсуляції та приховання даних. Всі алгоритми працюють з методами контейнерів, нічого не знаючи про те, як вони реалізовані.

Операції, що немодифікують:

- **for_each()** - виконує операції для кожного елемента послідовності;
- **find()** - знаходить перше входження значення в послідовність;
- **find_if()** - знаходить перша відповідність предикату в послідовності;
- **count()** - підраховує кількість входжень значення в послідовність;
- **count_if()** - підраховує кількість виконань предиката в послідовності;
- **search()** - знаходить перше входження послідовності як підпослідовності;
- **search_n()** - знаходить n-і входження значення в послідовність.

Операції, що модифікують:

- **copy()** - копіює послідовність, починаючи з першого елемента;
- **swap()** - міняє місцями два елементи;
- **replace()** - замінює елементи з заданим значенням;
- **replace_if()** - замінює елементи при виконанні предиката;
- **replace_copy()** - копіює послідовність, замінюючи елементи із заданим значенням;
- **replace_copy_if()** - копіює послідовність, замінюючи елементи при виконанні предиката;
- **fill()** - замінює всі елементи даним значенням;
- **remove()** - видаляє елементи з даним значенням;
- **remove_if()** - видаляє елементи при виконанні предиката;
- **remove_copy()** - копіює послідовність, видаляючи елементи із заданим значенням;

- **remove_copy_if()** - копіює послідовність, видаляючи елементи при виконанні предиката;
- **reverse()** - змінює порядок проходження елементів на зворотний;
- **random_shuffle()** - переміщає елементи відповідно до випадкового рівномірного розподілу (“перетасовує” послідовність);
- **transform()** - виконує задану операцію над кожним елементом послідовності;
- **unique()** - видаляє рівні сусідні елементи;
- **unique_copy()** - копіює послідовність, видаляючи рівні сусідні елементи.

Операції сортування:

- **sort()** - сортує послідовність з гарною середньою ефективністю;
- **partial_sort()** - сортує частина послідовності;
- **stable_sort()** - сортує послідовність, зберігаючи порядок проходження рівних елементів;
- **lower_bound()** - знаходить перше входження значення у відсортованій послідовності;
- **upper_bound()** - знаходить перший елемент, більший чим задане значення;
- **binary_search()** - визначає, чи є даний елемент у відсортованій послідовності;
- **merge()** - зливає дві відсортовані послідовності.

Операції над множинами:

- **includes()** - перевірка на входження;
- **set_union()** - об'єднання множин;
- **set_intersection()** - перетинання множин;
- **set_difference()** - різниця множин.

Операції знаходження мінімумів і максимумів:

- **min()** - менший з двох елементів;
- **max()** - більший з двох;
- **min_element()** - найменше значення у послідовності;
- **max_element()** - найбільше значення у послідовності.

Ітератори

Ізольованого типу ітератора не існує, тому що він визначається у різних контейнерах залежно від організації самого контейнера. Існує п'ять категорій ітераторів.

1. Ітератори вводу призначені для запису елементів у контейнер. Вони підтримують операції рівності, розіменування та інкремента.

`==, !=, *i, ++i, i++, *i++`

Спеціальним випадком ітератора вводу є клас `istream_iterator`.

2. Ітератори виводу призначені для читання елементів із контейнера. Вони підтримують операції розіменування, що можна виконувати лише у лівій частині присвоювання, та інкремента.

`++i, i++, *i = t, *i++ = t`

Спеціальним випадком ітератора виводу є клас `ostream_iterator`.

3. Односпрямовані ітератори призначені для перебору елементів контейнера у визначеному напрямку. Вони підтримують всі операції ітераторів вводу/виводу `i`, крім того, дозволяють без обмеження застосовувати присвоювання.

`==, !=, =, *i, ++i, i++, *i`

4. Двоспрямовані ітератори мають усі властивості односпрямованих ітераторів, а також мають додаткову операцію декремента (`--i, i--, *i--`), що дозволяє їм проходити контейнер в обох напрямках.

5. Ітератор прямого (довільного) доступу мають усі властивості двоспрямованих ітераторів, а також підтримують операції порівняння й адресної арифметики, тобто безпосередній доступ по індексі.

`i += n, i + n, i -= n, i - n, i1 - i2, i[n],`

$$i1 < i2, i1 \leq i2, i1 > i2, i1 \geq i2$$

У бібліотеці STL реалізовані також зворотні ітератори, роль яких можуть відігравати або двоспрямовані ітератори, або ітератори довільного доступу, але обходячи послідовність у зворотному напрямку.

2.2 Завдання до лабораторної роботи

2.2.1 Перевірка правильності розміщення дужок (елементарний синтаксичний верифікатор)

Заданий рядок, що містить арифметичний вираз. У виразі можуть використовуватися символи латинського алфавіту, цифри й знаки операцій, а також дужки трьох видів: ($\{\}$). Необхідно перевірити правильність розміщення дужок у виразі.

Вхідні дані:

У єдиному рядку перебуває арифметичний вираз. У виразі можуть використовуватися символи 0-9, +, -, *, /, = та дужки трьох видів: ($\{\}$).

Результат:

В єдиному рядку виводу вивести «YES», якщо дужки розставлені правильно, або «NO» якщо ні.

Приклад:

Вхідні дані	Результат
126+15*(3625+{7- 15*3}*12+25/2+7*[123- 78/5])	YES

Рекомендації:

Це завдання класично вирішується з використанням стека [1-3, 8, 9-11]. Елементи стека повинні мати символьний тип. Функція перевірки вмісту вершини стека не знадобиться.

Сам алгоритм працює в такий спосіб [4, 5]. Послідовно проглядаються всі символи рядка. Якщо зустрічається символ відкритої дужки (кожного із трьох типів), цей символ заноситься в стек. Якщо зустрічається символ закритої дужки, то зі стека знімається верхній елемент і, якщо це відповідна відкрита дужка, то перегляд рядка триває. Інакше – вихід з алгоритму й вивід повідомлення про те, що правильність розміщення дужок не дотримується.

При реалізації необхідно пам'ятати, що перед тим, як зняти елемент із вершини стека, необхідно перевірити, чи не порожній він. Після закінчення роботи алгоритму необхідно, щоб стек залишився порожнім. Інакше відкритих дужок буде більше, ніж закритих, і правильність розміщення дужок не буде дотримуватися.

2.2.2 Обчислення арифметичного виразу (елементарний синтаксичний інтерпретатор)

Заданий рядок, що містить арифметичний вираз. У записі допускаються цифри, символи '+', '-', '*', '/', '.' та круглі дужки. Обчислити вираз. При обчисленні виразу слід враховувати порядок виконання арифметичних дій. Якщо вираз обчислити неможливо – видати повідомлення про помилку.

Вхідні дані:

У єдиному рядку перебуває арифметичний вираз. У виразу можуть використовуватися символи 0-9, +, -, *, / та пробіл.

Результат:

У єдиному рядку виводу вивести значення виразу або «ERROR» якщо вираз містить помилки або не може бути обчисленим.

Приклад:

Вхідні дані	Результат
$2+2*2-(2/2+2)$	3

Рекомендації:

В алгоритмі буде використовуватися два стека: стек операцій (який зберігатиме інформацію про арифметичні дії, виконані при обчисленні) та стек операндів (який зберігатиме інформацію про числа, над якими проводяться дії). Як впливає з призначення стека операндів, його елементи повинні бути такого типу, який може зберігати дійсні числа (у записі вираза припустима десяткова крапка). Стек операцій зручно зробити символьним.

Алгоритм обчислення виразу будується в такий спосіб [4, 5]. Посимвольно проглядається вхідний рядок. Якщо поточний елемент рядка – цифра, то виділяється число, у запис якого вона входить, і заноситься в стек операндів. Якщо зустрічається символ відкритої дужки або операції множення або ділення, то вони заносяться в стек операцій. Якщо зустрічаються операції додавання, віднімання або закритої дужки, то запускається процедура обчислення вже розібраної частини виразу, що перебуває в стеках. Після цього операції додавання та віднімання заносяться в стек операцій, а закрита дужка видаляється разом з відповідній їй відкритій. Якщо в стеці операцій відповідна відкрита дужка відсутня після часткового обчислення виразу, то в записі виразу допущена помилка та виводиться відповідне повідомлення. При досягненні кінця рядка обчислюється розібрана частина виразу, що залишилася в стеках.

Обчислення частково розібраного виразу відбувається в такий спосіб. Доти, поки в стеку операцій перебувають операції й чергова з них - не відкрита дужка, зі стека операндів знімається два числа (перше число - це другий операнд, а друге із чисел, що знімаються, - перший, тому що в стеку порядок проходження елементів міняється на протилежний) і над ними проводиться відповідна дія. Результат обчислення заноситься в стек операндів і надалі використовується як операнд при обчисленні наступних операцій. Слід звернути увагу на відстеження ділення на 0, тому що ця дія вважається неприпустимою і повинна приводити до видачі повідомлення про помилку. Обчислення проводяться до першої відкритої дужки в стеку операцій через те, що дії в дужках виконуються до всіх інших дій. Таким чином, доки не буде отримане число, рівне результату виконання дій у дужках,

доти виконання інших операцій тимчасово відкладається (вони запам'ятовуються в стеку).

Після досягнення кінця рядка необхідно зняти результат обчислень зі стека операндів і перевірити, чи не залишилося значень у стеках. Якщо стеки не порожні, то у виразу були допущені помилки, та необхідно вивести відповідне повідомлення.

2.2.3 Лічилочка

Навколо людини, що рахує, знаходиться N людей, з яких виділений перший, а інші занумеровані за годинниковою стрілкою числами від 2 до N . Людина, що рахує, починаючи з когось, веде рахунок до M . Людина, на якій зупинився рахунок, виходить із кола. Рахунок триває з наступної людини й так доти, доки не залишиться одна людина.

Визначити номер людини, що залишилася.

Вхідні дані:

У єдиному рядку перебуває два цілі числа N ($1 \leq N \leq 10^6$) і M ($1 \leq M \leq 10^6$) розділені пробілом.

Результат:

Вивести єдине число - номер людини, що залишилася.

Приклад:

Вхідні дані	Результат
5 3	4

Рекомендації:

Необхідно завести чергу [1-3, 9-11] з елементами натурального типу, кожний елемент черги являє собою людину й зберігає його номер. Починаючи з першого елемента черги видаляти кожний M -й елемент, продовжуючи обхід черги спочатку після останнього елемента (можна зробити чергу замкненою, коли

показчик наступного елемента у останнього елемента черги вказує на перший – кільцевий зв'язаний список) доти, доки довжина черги не рівна 1.

2.2.3.1 Лічилочка навпаки*

Формулювання завдання те ж, але необхідно знайти номер людини, з якої починався рахунок, якщо відомі N , M і номер людини, що залишилася L .

2.3 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Основні функціональні блоки розроблених програм (структурна схема або код програми).
3. Особисті тестові дані для кожного завдання та результати роботи програми для цих даних.
4. Короткі відповіді на основні запитання та висновки за результатами роботи.

2.4 Контрольні питання

1. Які переваги використання динамічних інформаційних структур?
2. Коли слід використовувати списки?
3. Які основні операції над списками?
4. Як відбувається швидка навігація по списках з пропусками?
5. Що таке стек, дек і черга? Які основні операції над ними?
6. Що таке асоціативний масив? Які основні операції над ним існують?
7. Що таке хеш таблиця? Як виконується швидкий пошук комплексних типів даних за допомогою хеш таблиці?
8. Які динамічних структури представлені в бібліотеці STL?
9. Для чого призначені контейнери в складі STL?
10. Для чого призначені ітератори в складі STL?
11. Які основні алгоритми контейнерів в STL?
12. Що таке функціональні об'єкти в складі STL?
13. Що таке адаптери в STL?

3 ЛАБОРАТОРНА РОБОТА № 3 - СОРТУВАННЯ Й ПОШУК

Мета роботи: одержання навичок застосування швидких алгоритмів сортування й пошуку в різних ситуаціях [1-3, 19].

3.1 Завдання до лабораторної роботи

3.1.1 Відрізки на прямій

На прямій пофарбували N відрізків. Відомі координата $L[I]$ лівого кінця відрізка й координата $R[I]$ правого кінця I -го відрізка для $I=1, \dots, N$. Знайти суму довжин усіх пофарбованих частин прямої.

Примітка. Число N настільки велике, що на виконання $N*N$ навіть найпростіших операцій бракуватиме часу.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^6$ – число відрізків. У наступних N рядках перебувають пари натуральних чисел $1 \leq L[I] \leq 10^9$ і $1 \leq R[I] \leq 10^9$ – координати початку й кінця відрізків.

Результат:

Вивести єдине натуральне число – суму довжин зафарбованих частин прямої.

Приклад:

Вхідні дані	Результат
3 1 5 3 7 8 9	7

Рекомендації:

Відрізки варто спочатку відсортувати одним з ефективних алгоритмів [19], а потім в один прохід підрахувати суму зафарбованих ділянок на прямій.

3.1.2 Силует міста

Необхідно написати програму, яка допоможе визначити силует міста. Для формалізації завдання передбачається, що всі будинки мають прямокутну форму й розташовані на одній висоті. Кожний будинок задається трійкою чисел L_i , H_i і R_i , де L_i – координата початку будинку, H_i – висота будинку та R_i – координата кінця будинку. Рисунок нижче описує послідовність трійок чисел: (1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28), якої відповідає силует міста: (1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0).

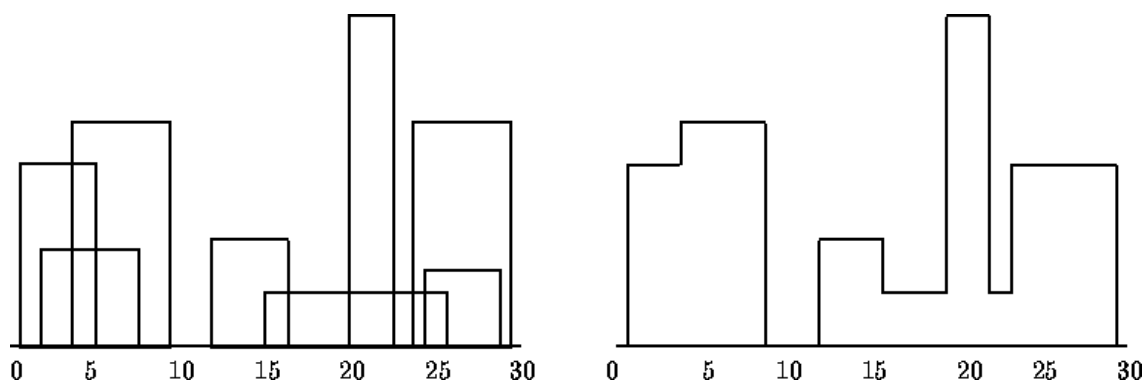


Рисунок 3.1 – Силует міста

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^6$ – число будинків. У наступних N рядках перебувають трійки натуральних чисел $1 \leq l_i \leq 10^9$, $1 \leq h_i \leq 10^9$ та $1 \leq r_i \leq 10^9$ – координати початку висоти й кінця будинків.

Результат:

Необхідно вивести вектор чисел, що описують силует міста, як у наведеному прикладі. Вектор $(v_1, v_2, v_3, \dots, v_n)$ – це послідовність чисел, де непарні члени позначають координати перепаду висот, а парні – висоту чергової горизонтальної

ділянки силуету міста. Вектор повинен починатися з лівого краю найпершого будинку й закінчуватися числом 0 після координат правого краю самого правого будинку. Числа розділені пробілом.

Приклад:

Вхідні дані	Результат
8	1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29
1 11 5	0
2 6 7	
3 13 9	
12 7 16	
14 3 25	
19 18 22	
23 13 29	
24 4 28	

Рекомендації:

Загальні рекомендації повністю аналогічні попередньому завданню з невеликою модифікацією, пов'язаною з необхідністю виводити висоти будинків, а не просто підрахувати сумарну довжину їх ділянок, що перекриваються. Треба врахувати, що один будинок може повністю або частково перекривати декілька інших будинків одночасно.

3.1.3 Сорткування файлу

Необхідно просто відсортувати по зростанню числа у файлі. Особливість завдання тільки одна – файл із даними дійсно великий і не міститься повністю в оперативній пам'яті.

Вихідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 2^{32}$ – число значень у файлі. У кожному з наступних рядків перебуває натуральне число $1 \leq V \leq 2^{16}$.

Результат:

Необхідно вивести вихідну послідовність чисел упорядкованої по зростанню по одному числу в рядку.

Приклад:

Вхідні дані	Результат
4	1 3 2 4
4 3 6 5	3 2 4 5
1 3 2 4	4 3 5 6
4 3 5 6	4 3 6 5
3 2 4 5	

Рекомендації:

Необхідно використовувати один з алгоритмів зовнішнього сортування [19].

3.1.3.1 Сортування файлу за лінійний час*

Завдання те ж, але необхідно відсортувати числа за лінійний час.

3.1.3.2 Сортування файлу із довільними числами*

Завдання те ж, але числа із плаваючою крапкою (тип float) та необхідно відсортувати числа за лінійний час.

3.2 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Основні функціональні блоки розроблених програм (структурна схема або код програми).

3. Особисті тестові дані для кожного завдання та результати роботи програми для цих даних.

4. Короткі відповіді на основні запитання та висновки за результатами роботи.

3.3 Контрольні питання

1. Наведіть класифікацію алгоритмів сортування.
2. Опишіть алгоритм сортування бульбашкою. Які його недоліки?
3. Опишіть алгоритм сортування підрахунком, його переваги.
4. Опишіть алгоритм сортування злиттям та його переваги.
5. Опишіть алгоритм швидкого сортування та його переваги.
6. Які існують алгоритми зовнішнього сортування?

4 ЛАБОРАТОРНА РОБОТА № 4 - ОБХІД ГРАФІВ

Мета роботи: одержання навичок застосування швидких алгоритмів на графах та основ побудови оптимальних алгоритмів на графах [14-18].

У цій лабораторній роботі й далі рекомендується застосовувати для розв'язку завдань Boost Graph Library.

4.1 Завдання до лабораторної роботи

4.1.1 Компіляція програми

Сучасні програми складаються з безлічі файлів, кожний з яких використовує у свою чергу при компіляції інші файли, утворюючи в такий спосіб «дерево залежностей». Наприклад:

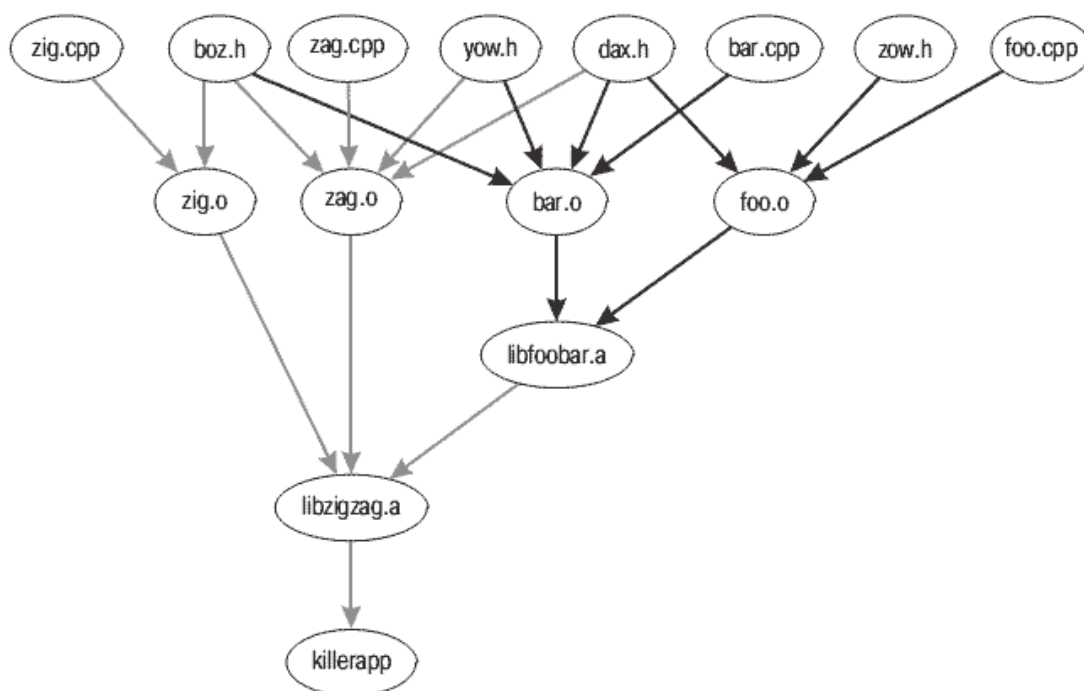


Рисунок 4.1 – Дерево залежностей

Необхідно визначити порядок компіляції (компонування) файлів для складання програми.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 1000$ – число файлів. У наступних N рядках перебувають серії з імен файлів, записані в строку. Перший файл у рядку – ім'я файлу в проекті. Інші імена файлів у рядку – список файлів проекту, які використовують поточний файл при компіляції (список залежностей). Імена файлів не містять пробілів.

Результат:

Вивести N рядків, у кожному з яких міститься ім'я файлу в порядку їх компіляції з урахуванням залежностей між файлами.

Приклад:

Вхідні дані	Результат
15	zig.cpp
killerapp libzigzag.o	boz.h
libzigzag.a zag.o zig.o libfoobar.a	zag.cpp
zig.o zig.cpp boz.h	yow.h
zag.o boz.h zag.cpp yow.h dax.h	dax.h
libfoobar.a bar.o foo.o	bar.cpp
bar.o boz.h yow.h dax.h bar.cpp	zow.h
foo.o dax.h zow.h foo.cpp	foo.cpp
zig.cpp	zig.o
boz.h	zag.o
zag.cpp	bar.o
yow.h	foo.o
dax.h	libfoobar.a
bar.cpp	libzigzag.a
zow.h	killerapp
foo.cpp	

Рекомендації:

Використовувати алгоритм пошуку в глибину для топологічного сортування орієнтованого графа (дерева) залежностей.

4.1.2 Компіляція на паралельному комп'ютері

Умова завдання 1 доповнене інформацією про час компіляції (компонування для об'єктних модулів) кожного файлу. Необхідно визначити мінімальний сумарний час складання програми на кластері комп'ютерів, що працюють паралельно. Передбачається, що комп'ютерів наперед достатньо для будь-якого ступеня паралелізму (необмежена кількість). Наприклад (у дужках зазначений час компіляції кожного файлу й час компіляції при паралельному складанні, критичний шлях позначений жирною лінією).

Необхідно визначити час компіляції програми на кластері паралельних комп'ютерів.

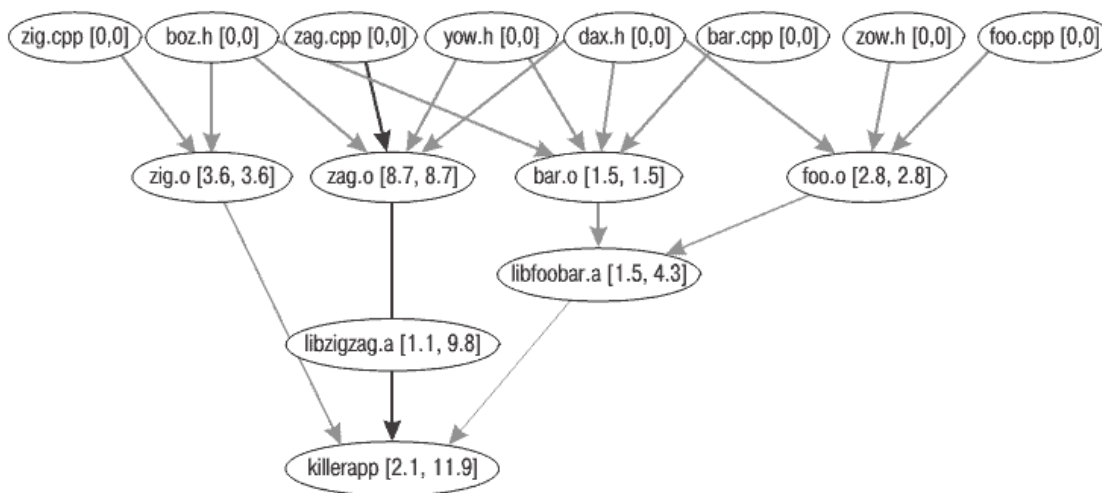


Рисунок 4.2 – Дерево залежностей

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 1000$ – число файлів. У наступних N рядках перебувають серії з імені файлу проекту, часу його компіляції й імен файлів, від яких він залежить, записаних у рядок. Імена файлів не містять пробілів. Час компіляції – невід'ємне число із плаваючою крапкою.

Результат:

Вивести єдине число – час складання програми на кластері паралельних комп'ютерів.

Приклад:

Вхідні дані	Результат
15 killerapp 2.1 libzigzag.a zig.o libfoobar.a libzigzag.a 1.1 zag.o libfoobar.a 1.5 bar.o foo.o zig.o 3.6 zig.cpp boz.h zag.o 8.7 boz.h zag.cpp yow.h dax.h bar.o 1.5 boz.h yow.h dax.h bar.cpp foo.o 2.8 dax.h zow.h foo.cpp zig.cpp 0.0 boz.h 0.0 zag.cpp 0.0 yow.h 0.0 dax.h 0.0 bar.cpp 0.0 zow.h 0.0 foo.cpp 0.0	11.9

Рекомендації:

Необхідно знайти критичний шлях у дереві залежностей (шлях з максимальною сумою часів компіляції) методом пошуку в глибину або зворотним ходом хвильового алгоритму.

4.1.3 Оптимальна маршрутизація

Деяка кількість гравців відіграють в MMORPG на сервері у вашій мережі. Відомо, що чим менше затримка при передачі даних між сервером і клієнтом – тим приємніше гравцям грати. Необхідно сформулювати такий план маршрутизації, щоб для кожного користувача затримка в грі була мінімальною.

Наприклад, заданий граф мережі (латентність з'єднань зазначена в мілісекундах) наведено на рис. 4.3.

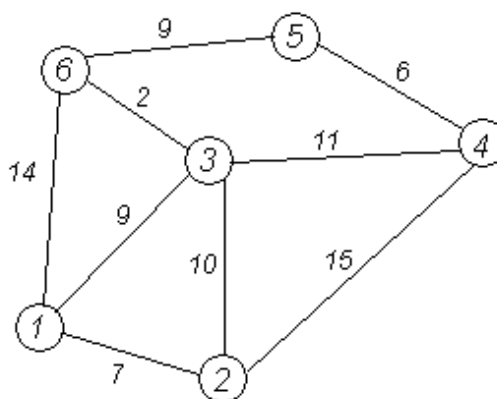


Рисунок 4.3 – Граф мережі

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 1000$ – число вузлів мережі. Далі в кожному рядку першим знаходиться число V_i – кількість фізичних з'єднань даного вузла мережі, а слідом V_i пар чисел $E_{i,j}$ – номер вузла з яким зв'язаний даний та $L_{i,j}$ – латентність з'єднання.

Сервер перебуває у вузлі з номером 1. Вузли нумеруються з 1.

Результат:

Вивести N чисел, кожне в новому рядку – мінімальну затримку при передачі даних для кожного вузла мережі.

Приклад:

Вхідні дані	Результат
6	0
3 6 14 3 9 2 7	7
3 1 7 3 10 4 15	9
4 1 9 2 10 4 11 6 2	20
3 2 15 3 11 5 6	20
2 4 6 6 9	11
3 1 14 3 2 5 9	

Рекомендації:

Використовувати алгоритм Дейкстри або Флойда-Белмана для пошуку найкоротшої відстані між вузлами мережі.

4.1.3.1 Маршрутизація із мінімальною затримкою пакетів*

Вивести план маршрутизації пакетів, при якому кожний з користувачів буде грати з мінімальною затримкою.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 1000$ – число вузлів мережі. Далі в кожному рядку першим знаходиться число V_i – кількість фізичних з'єднань даного вузла мережі, а слідом V_i пара чисел $E_{i,j}$ – номер вузла з яким зв'язаний даний та $L_{i,j}$ – латентність з'єднання.

Сервер перебуває у вузлі з номером 1. Вузли нумеруються з 1.

Результат:

У кожному з $N-1$ рядків вивести через пробіл номера вузлів, через які буде проходити трафік для даного користувача (нагадаємо, що вузол з номером 1 – це сервер, для нього маршрутизація не потрібна).

Приклад:

Вхідні дані	Результат
6	1 2
3 6 14 3 9 2 7	1 3
3 1 7 3 10 4 15	1 3 4
4 1 9 2 10 4 11 6 2	1 3 6 5
3 2 15 3 11 5 6	1 3 6
2 4 6 6 9	
3 1 14 3 2 5 9	

4.1.4 Прокладка мережі

Ви збираєтеся розгорнути комп'ютерну мережу мікрорайону, використовуючи вже наявні колодязі й підземні кабельні шахти Укртелекома. Укртелеком надає вам можливість самому вибрати, які з кабельних шахт використовувати. Для кожної кабельної шахти відома вартість прокладки вашого кабелю в ній. У вас є план кабельних шахт Укртелекома у вашому районі. Вам необхідно скласти план створення мережі мікрорайону для досягнення повного покриття території з мінімальними витратами. Приклад графа кабельних шахт Укртелекома (вартість зазначена в тисячах у.о., оптимальний план прокладки зазначений жирними лініями) наведено на рис. 4.4.

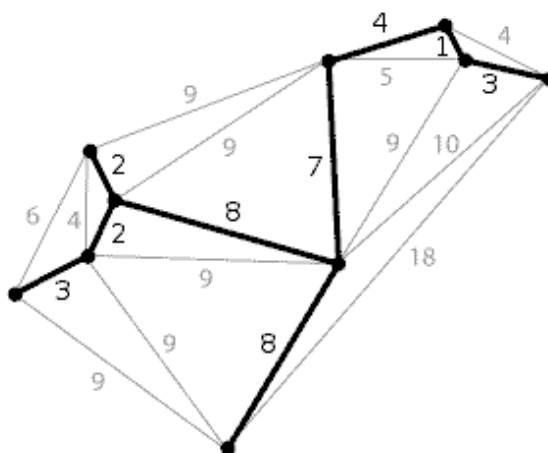


Рисунок 4.4 – Граф кабельних шахт Укртелекома

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^5$ – число кабельних шахт. У кожному з наступних N рядків перераховано 3 числа: номер колодязів (вузлів мережі), між якими прокладена шахта й вартість прокладки вашого кабелю в ній.

Приклад:

Вхідні дані	Результат
21	1 3
1 2 6	3 5
1 3 3	5 2
1 4 9	5 7
2 3 4	7 4
2 5 2	7 6
2 6 9	6 8
3 5 2	8 9
3 4 9	9 10
3 7 9	38
4 7 8	
4 10 18	
5 6 9	
5 7 8	
6 7 7	
6 8 4	
6 9 5	
7 9 9	
7 10 10	
8 9 1	
8 10 4	
9 10 3	

Результат:

Вивести план прокладки мережі, що покриває всі колодязі (вузли мережі) з мінімальними витратами на прокладку. Для цього в кожному з N-1 рядків вивести два числа – номери колодязів (вузлів мережі), шахта між якими буде використовуватися.

В останньому рядку вивести вартість розгортання мережі.

Рекомендації:

Необхідно знайти мінімальне остовне дерево графа.

4.2 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Основні функціональні блоки розроблених програм (структурна схема або код програми).
3. Особисті тестові дані для кожного завдання та результати роботи програми для цих даних.
4. Короткі відповіді на основні запитання та висновки за результатами роботи.

4.3 Контрольні питання

1. Опишіть метод пошуку в глибину.
2. Опишіть метод пошуку завширшки.
3. Опишіть двонаправлений пошук завширшки.
4. Алгоритм Декстри і алгоритм Белмана — Форда — яку задачу вони вирішують і в чому між ними відмінність?
5. Як працює хвильовий алгоритм?
6. Що таке мінімальне основне дерево графа?
7. Як працює алгоритм Прима для знаходження остовного дерева графа?

5 ЛАБОРАТОРНА РОБОТА № 5 - ЗАДАЧІ НА ГРАФАХ

Мета роботи: одержання навичок застосування швидких алгоритмів на графах і основ побудови оптимальних алгоритмів на графах [14-19].

У цій лабораторній роботі й далі рекомендується застосовувати для розв’язку задач Boost Graph Library [13].

5.1 Завдання до лабораторної роботи

5.1.1 Надійність мережі

Надійність комп’ютерної мережі визначається тим, скільки сегментів мережі можуть одночасно вийти з ладу, але всі комп’ютери мережі все ще будуть з’єднані між собою. Сучасні мережі, як правило, дублюють найбільш важливі магістральні з’єднання, як для підвищення пропускної здатності, так і для надійності мережі в цілому.

Ваша задача визначити чи існує у вашій мережі такий сегмент, виведення з ладу якого порушить цілісність мережі (тобто розіб’є мережу на дві підмережі так, що комп’ютери однієї підмережі не будуть мати доступу до комп’ютерів іншої підмережі).

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^5$ – число сегментів мережі. У кожній з наступних N рядків перераховані пари чисел: номери вузлів, які з’єднують сегмент мережі.

Результат:

Вивести всі сегменти, у вигляді пари чисел – номерів вершин, які вони з’єднують, по одному сегменту в рядку, видалення кожного з яких приведе до порушення цілісності мережі. Або вивести “NO” якщо таких сегментів немає.

Приклад:

Вхідні дані	Результат
9	4 7
1 3	4 5
1 4	5 2
3 4	
4 5	
4 7	
5 2	
5 6	
5 8	
6 8	

Рекомендації:

По черзі видаляти сегменти мережі й перевіряти число компонентів зв'язності графа будь-яким доступним методом, наприклад хвильовим алгоритмом або пошуком у глибину.

5.1.1.1 Критичні вузли мережі*

Необхідно визначити наявність вузлів мережі, виведення з ладу яких спричиняє порушення цілісності мережі.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^5$ – число сегментів мережі. У кожному з наступних N рядків перераховані пари чисел: номери вузлів, які з'єднують сегменти мережі.

Результат:

Вивести номери вузлів мережі, порушення працездатності яких спричиняє порушення цілісності мережі, кожний вузол з нового рядка. Або вивести “NO”, якщо таких вузлів немає.

5.1.2 Максимальний трафік між вузлами

Вам необхідно передати величезний об’єм даних по мережі від одного вузла до іншого. Номер вузлів джерела й приймача даних відомі. Так само для кожного мережного з’єднання відома його пропускна здатність. Необхідно визначити максимальну швидкість передачі даних між двома заданими вузлами.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^5$ – число сегментів мережі. У кожному з наступних N рядків перераховано 3 числа: номери вузлів мережі, які з’єднують сегмент та пропускна здатність з’єднання.

В останньому рядку зазначено два числа – номери вузла джерела даних та одержувача.

Результат:

У першому рядку вивести одне значення – максимальну пропускну здатність мережі між заданими вузлами.

У наступних рядках вивести інформацію про сегменти мережі, що брали участь у передачі даних у вигляді пар номерів вузлів мережі, які вони з’єднують, кожна пара чисел з нового рядка.

Приклад:

Вхідні дані	Результат
8	4
1 2 3	1 2
1 3 1	2 4
2 4 3	4 3
3 4 5	4 7
3 5 4	1 3
4 7 2	3 5
5 6 2	5 6
6 7 3	6 7
1 7	

Рекомендації:

Використовувати алгоритми Форда-Фалкенсона або Едмондса-Карпа для пошуку максимального потоку в транспортній мережі.

5.1.2.1 Мінімальна вартість передачі даних*

Задача та ж, але крім пропускної здатності кожного сегмента, зазначена так само вартість передачі одиниці даних через сегмент. Визначити перелік сегментів мережі, що брали участь у передачі даних, максимізувавши пропускну здатність, але мінімізувавши вартість передачі даних.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 10^5$ – число сегментів мережі. У кожному з наступних N рядків перераховано 4 числа: номери вузлів мережі, які з'єднує сегмент, пропускна здатність з'єднання й вартість передачі одиниці інформації.

В останньому рядку зазначено два числа – номери вузла джерела даних і одержувача.

Результат:

У першому рядку вивести максимальну пропускну здатність мережі між заданими вузлами й сумарну мінімальну вартість передачі даних між вузлами.

У наступних рядках вивести інформацію про сегменти мережі, що брали участь у передачі даних у вигляді пар номерів вузлів мережі, які вони з'єднують, кожна пара чисел з нового рядка.

5.1.3 З'єднання P2P

Злісні охоронці авторських прав нарешті повністю перекрили кисень традиційним P2P сервісам, таким як torrent. Але наші співвітчизники не сумують і платити за «біти в байтах» не збираються. Придумана нова захищена схема обміну файлами, закрита від стороннього втручання й заснована на VPN тунелях (закритих шифрованих з'єднаннях між двома заданими комп'ютерами з використанням VPN).

Схема працює в такий спосіб. Для кожного користувача відомо, які файли він прагне завантажити і які готовий надати. Кожний користувач може встановити одне вхідне й одне вихідне закрите з'єднання з іншим користувачем для передачі файлів. На сервері є інформація про список файлів, які кожний з них готовий надати й завантажити. Необхідно скласти план P2P з'єднань між користувачами, при якому число передач файлів буде максимальним.

Вхідні дані:

У першому рядку перебуває натуральне число $1 \leq N \leq 1000$ – число користувачів. Далі для кожного користувача зазначено 2 рядка з даними. У першому рядку через кому зазначені імена файлів, які він готовий надати. У другому рядку через кому зазначені імена файлів, які він прагне завантажити.

Результат:

У кожному рядку вивести два числа – номери користувача джерела, номер користувача приймача файлу, між якими слід установити пряме VPN з'єднання. Число прямих з'єднань при цьому повинне бути максимальним.

Приклад:

Вхідні дані	Результат
4	1 4
1.txt, 1.out	2 3
2.txt, 3.out	3 1
2.txt, 2.out	4 2
1.out, 4.txt	
3.txt, 3.out	
2.txt	
4.txt, 4.out	
1.out, 3.txt	

Рекомендації:

Задачу можна звести до задачі пошуку максимального паросполучення у дводольному графі.

5.1.4 Віртуальна організація

Вам необхідно організувати кластер для паралельних обчислень на базі мережі університету із заданою топологією. Необхідно знаючи топологію мережі університету й необхідну топологію кластера знайти список комп'ютерів мережі, які будуть задіяні під віртуальну організацію кластера.

Вхідні дані:

Мережі університету й кластера задані таблицями зв'язків їх графів. Спочатку йде граф мережі університету, за ним граф планованої топології кластера.

Результат:

Вивести номери вузлів (вершин графа) мережі університету, які будуть задіяні для організації кластера. Або вивести “NO”, якщо організувати кластер заданої топології на базі існуючої мережі університету неможливо.

Рекомендації:

Задачу можна звести до задачі пошуку граф-підграф ізоморфізму.

5.1.4.1 Розміщення кластера в мережі університету*

Для кожного вузла мережі університету задана продуктивність установленого у вузлі комп'ютера. Необхідно знайти варіант розміщення кластера в мережі, при яким, обчислювальна потужність кластера буде максимальної при дотриманні планованої топології кластера.

5.2 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Основні функціональні блоки розроблених програм (структурна схема або код програми).
3. Особисті тестові дані для кожного завдання та результати роботи програми для цих даних.
4. Короткі відповіді на основні запитання та висновки за результатами роботи.

5.3 Контрольні питання

1. Які існують алгоритми перевірки графа на дводольність?
2. Як працюють алгоритми пошуку мостів і точок зчленування?

3. Кратко опишіть алгоритми розв'язку задачі про максимальний потік. Алгоритм Форда-Фолкенсона. Алгоритм Едмондса-Карпа. В чому між ними різниця?
4. Опишіть алгоритми Куна для розв'язку задачі про максимальне паросполучення.
5. Як працюють рекурсивні алгоритми на графах? Як вирішити задачу комівояжера рекурсивним алгоритмом?
6. Топологічне порівняння графів. Розв'язок задачі про ізоморфізм графів. Задача про граф-підграф ізоморфізм.

6 ЛАБОРАТОРНА РОБОТА № 6 - МЕТОД ГІЛОК ТА МЕЖ, ЖАДІБНІ АЛГОРИТМИ, ДИНАМІЧНЕ ПРОГРАМУВАННЯ

Мета роботи: одержання навичок застосування рекурсивних алгоритмів на основі методу гілок та меж, евристики, жадібних алгоритмів та динамічного програмування для побудови ефективних програм розв'язку широкого кола оптимізаційних задач [1-4, 20].

6.1 Завдання до лабораторної роботи

6.1.1 Збір статистики в мережі

У комп'ютерній мережі послідовно кожний комп'ютер одержує маркер – спеціальний пакет даних, куди кожний комп'ютер дописує свій стан, статус, завантаження й інші системні параметри. Подібні системи періодичного опитування вузлів мережі скрізь використовуються в системах розподілених обчислень та комплексах з підвищеними вимогами до надійності (наприклад, системи реального часу). Необхідно, щоб при тій самій завантаженості мережі, інформація про стан вузлів оновлялася якнайчастіше. Для кожного мережного з'єднання задана латентність (затримка передачі даних). Необхідно знайти такий маршрут інформаційного пакета даних, щоб він проходив через усі вузли мережі рівно по одному разу й сумарна затримка одного циклу опитування вузлів була мінімальною.

Вхідні дані:

Комп'ютерна мережа задана матрицею суміжності. У першому рядку перебуває єдине значення N – число вузлів мережі. Далі в N рядках записані елементи матриці суміжності, розділені пробілом, що представляють собою латентність у мілісекундах.

Результат:

У першому рядку вивести число T – мінімальну сумарну затримку одного циклу опитування мережі. У наступному рядку вивести через пробіл номери вершин графа в порядку їх обходу інформаційним пакетом.

Приклад:

Вхідні дані	Результат
5	100
0 25 15 73 46	5 4 1 2 3 5
53 0 24 36 75	
32 72 0 88 24	
11 35 29 0 38	
22 63 34 16 0	

Рекомендації:

Задача являє собою варіацію на тему задачі комівояжера.

6.1.2 Кластер суперкомп'ютерів

Вам необхідно зібрати суперкомп'ютер. Для цього вам виділений фіксований бюджет. Ви зібрали інформацію о вартості та продуктивності різних комп'ютерів, з яких можна зібрати кластер (кластер може бути гетерогенним, тобто містити різні про продуктивності комп'ютери). Вам необхідно зібрати кластер максимальної продуктивності не виходячи за рамки бюджету.

Вхідні дані:

У першому рядку перебуває число B – ваш бюджет. У другому рядку перераховані через кому A_i - продуктивність кожного з варіантів комп'ютерів з яких можна скласти кластер. У третьому рядку перераховані через кому C_i - вартість кожного з варіантів комплектації комп'ютерів.

Результат:

У першому рядку вивести одне значення – максимальну продуктивність, яку можна досягти створивши кластер у рамках виділеного бюджету. У наступному рядку вивести число комп'ютерів кожного виду, які будуть використані при створенні кластера.

Приклад:

Вхідні дані	Результат
1000	11512
134, 1354, 567, 789, 987, 56, 345, 4567, 5555	3 0 0 0 0 0 0 0 2
16, 900, 440, 250, 667, 43, 333, 857, 474	

Рекомендації:

Варіація задачі про рюкзак. Вирішується методами динамічного програмування або жадібним алгоритмом.

6.1.3 Порівняння геномів

Як відомо, геном може бути представлений у вигляді послідовності нуклеотидів. Кожний нуклеотид у ланцюжку може бути позначений буквою латинського алфавіту. Ступінь схожості геномів двох різних істот будемо визначати як максимально можливу загальну послідовність нуклеотидів.

Вхідні дані:

У першому рядку перебуває геном першої істоти представлений рядком заголовних латинських букв. У другому рядку перебуває геном другої істоти так само у вигляді рядка.

Результат:

Вивести найбільшу загальну послідовність нуклеотидів двох геномів.

Приклад:

Вхідні дані
FJDNBTNFKJSISK TJNNFJDKOEJHTNDNSJSHTNGVNSDKA
ADNTNFNNDKJSHNNFDFOEKRTJYJSHNRVGN
Результат
DNTNFKJSNNFDOEJSHNV

Рекомендації:

Задача динамічного програмування про пошук найбільшої загальної підпослідовності.

6.1.4 П'ятнашки

Ви – ледачий програміст. Вам хочеться зібрати п'ятнашки, але лінь сидіти й перебирати варіанти – нехай це робить машина.

Вхідні дані:

У чотирьох рядках задані через пробіл числа, що утворюють поле гри в п'ятнашки. Порожня клітка представлена числом 0.

Результат:

Вивести послідовність дій, що приводять до розв'язку головоломки мінімальної довжини. Послідовність виводиться у вигляді напрямків переміщення порожньої клітки – U-вгору, D-вниз, L-вліво, R-вправо. Якщо задача не може бути вирішена – вивести NO.

Приклад:

Вхідні дані	Результат
13 10 11 6 5 7 4 8 1 12 14 9 3 15 2 0	NO

Приклад:

Вхідні дані	Результат
2 5 4 11 1 8 3 7 14 6 10 12 9 13 15 0	UUULDLULDRDLDRURRULDRD

Рекомендації:

Розв'язок методом гілок та меж.

6.2 Зміст звіту

1. Формулювання мети й задачі лабораторної роботи.
2. Основні функціональні блоки розроблених програм (структурна схема або код програми).
3. Особисті тестові дані для кожного завдання та результати роботи програми для цих даних.
4. Короткі відповіді на основні запитання та висновки за результатами роботи.

6.3 Контрольні питання

1. Опишіть коротко метод гілок та меж.
2. Що таке евристичні алгоритми? Яка їх область застосування, особливості?
3. Що таке жадібні алгоритми?
4. Опишіть метод динамічне програмування. Де він застосовується?

5. Які існують методи вирішення задачі комівояжера?
6. Які існують методи вирішення задачі про рюкзак?
7. Як оптимально вирішується задача про максимальну загальну підпоследовність?
8. Як вирішити задача про максимальну загальну підпоследовність що не убиває, в чому особливість?

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Donald Knuth. Art of Computer Programming, The, Volumes 1-4B, Boxed Set (Art of Computer Programming, 1-4). - Addison-Wesley Professional; 1st edition (October 15, 2022). - 736 p.
2. Robert Sedgewick. Algorithms in Java, Parts 1-8: Complete Version. - Addison Wesley; 3rd edition (February 2, 2020). - 1312 p.
3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. Introduction to Algorithms. - The MIT Press; 4th edition (April 5, 2022). - 1312 p.
4. Panos Louridas. Algorithms. - The MIT Press (August 18, 2020). - 320 p.
5. Theophilus Edet. Algorithms and Data Structures (Computer Science Fundamentals). - CompreQuest Books (September 23, 2023). - 265 p.
6. Shubham Gupta. Data Structures and Algorithms. - Toronto Academic Press (January 10, 2024). - 239 p.
7. John Farrier. Data Structures and Algorithms with the C++ STL: A guide for modern C++ practitioners. - Packt Publishing (February 29, 2024). - 458 p.
8. Introduction to Algorithms: A Comprehensive Guide for Beginners: Unlocking Computational Thinking / Quantum Technologies. - Independently published (July 30, 2023). - 267 p.
9. Algorithms and Data Structures with Python: An interactive learning experience: Comprehensive introduction to data structures and algorithms / Quantum Technologies. - Independently published (November 14, 2023). - 406 p.
10. Maxwell Rivers. Data Structures and Algorithms in C: A Comprehensive Guide. - Independently published (September 30, 2023). - 131 p.
11. Maxwell Rivers. Data Structures and Algorithms in Java: A Comprehensive Guide. - Independently published (October 1, 2023). - 129 p.
12. Bill Weinman. C++20 STL Cookbook: Leverage the latest features of the STL to solve real-world problems. - Packt Publishing; 1st edition (May 27, 2022). - 450 p.

13. Peter Van Weert, Marc Gregoire. C++17 Standard Library Quick Reference: A Pocket Guide to Data Structures, Algorithms, and Functions. - Apress; 2nd ed. edition (July 31, 2019). - 316 p.
14. Miklos Bona. A Walk Through Combinatorics: An Introduction to Enumeration, Graph Theory, and Selected Other Topics. - World Scientific Pub Co Inc; 5th edition (August 29, 2023). - 613 p.
15. William Kocay, Donald L. Kreher. Graphs, Algorithms, and Optimization (Discrete Mathematics and Its Applications). - Chapman and Hall/CRC; 2nd edition (January 21, 2023). - 566 p.
16. Christopher Griffin. Applied Graph Theory: An Introduction with Graph Optimization and Algebraic Graph Theory - World Scientific Publishing Company (August 8, 2023). - 304 p.
17. Jeremy Kubica. Graph Algorithms the Fun Way. - No Starch Press (November 19, 2024). - 264 p.
18. Aditya Y Bhargava. Grokking Algorithms. - Manning; 2nd edition (March 26, 2024). - 320 p.
19. Venky Karukuri. Sorting Algorithms Handbook: A Practical Approach with CPP, Java and Python Implementations and Interview Questions. - Amazon Kindle (September 21, 2023). - 230 p.
20. Robert Setiadi. Making Sense of Programming Algorithms Foundations. - Independently published (October 10, 2023). - 295 p.