Комунальний заклад освіти

«Дніпровський ліцей інформаційних технологій

при Дніпровському національному університеті

імені Олеся Гончара»

Випускна робота

на тему:

**Пошук найбільш безпечного шляху на заражений ділянці**

Виконав:

ліцеїст 11-В-I класу

Єременко Олександр Сергійович

Керівник:

Ентін Йосиф Абрамович

**Дніпро 2021**

Зміст

[Вступ 3](#_Toc90012301)

[Теоретична частина 5](#_Toc90012302)

[Головна частина 12](#_Toc90012303)

[Крупноблочна блок-схема 20](#_Toc90012304)

[Програмно-апаратні вимоги 21](#_Toc90012305)

[Перелік використаних програмних проектів 22](#_Toc90012306)

[Посібник користувача 23](#_Toc90012307)

[Висновки 24](#_Toc90012308)

[Список джерел інформації 25](#_Toc90012310)

# Вступ

Моя випускна робота являє собою рішення задачі. Умов її такі:

Є місцевість, яку має перетнути медична служба. Ця місцевість радіоактивно заражена. Треба визначити на ній такий шлях, на якому одержана доза радіації буде мінімальною.  
Місцевість має форму прямокутника. Радіорозвідка для зручності розбила її на однакові квадрати. У межах кожного з них рівень радіації вважається однаковим. Задана швидкість пересування по місцевості автомобіля з медиками. Доза, одержана на ділянці шляху – це добуток рівня радіації на ньому, помножений на час, протягом якого пройдено дану ділянку. Автомобіль може рухатися в різних напрямках. Заборонено двічі відвідувати одну й ту ж ділянку.

Але в цій задачі є уосконалення:

На ділянці в випадковому порядку створюються певні квадрати. Вони такі ж за розмірами, як і всі інші квадрати на які розбили всю ділянку. Ці квадрати також можна перетнути, але на них радіація значно вище ніж на інших і її рівень змінюється з часом.

В епоху техногенних катастроф дана тема являється найбільш актуальною, так як в наш час дуже часто трапляються неминучі фатальні ситуації, де кожна секунда на рахунку життя, і не завжди допомога приходить вчасно.

У даній роботі розглянуті методи рішення задачі для пошуку оптимального шляху, для цього був використаний "Алгоритм Дейкстри", але дещо модифікований і він має назву «А\*». Під час вирішення основної задачі було вирішено супровідну типову задачу на повний перебір під назвою «Світлофор». У ньому були використані методи за якими у моєму основному проекті будуть застосовани для доповнення.

Мета роботи: створити програму, яка визначає маршрут, на якому доза радіації мінімальна.

Для реалізації мети були поставлені такі завдання:

1. проаналізувати науково-методичну літературу в області методів вирішення завдань для пошуку найкоротшого шляху метою вибору найбільш ефективних і дієвих.

2. Проаналізувати необходні алгоритми для вирішення цих задач.

3. Вирішення задачі «Світлофор»,

3. Написати необхідний код і візуалізувати рішення задачі.

4. Підготувати супровідну документацію.

# Теоретична частина

***Перебір і методи його скорочення***

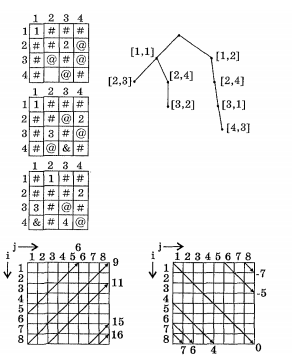
У багатьох прикладних задачах потрібно знайти оптимальне рішення серед дуже великого (але кінцевого) числа варіантів. Іноді вдається побудувати це рішення відразу, але в більшості випадків єдиний спосіб його відшукати полягає в переборі всіх можливих варіантів і порівнянні їх між собою. Тому так важливо навчитися будувати алгоритми перебору різних комбінаторних об'єктів - послідовностей, перестановок, підмножин і т.д.

Рисунок 1 Приклад повного перебіру

Зазвичай коли необхідно розглянути способи скорочення перебору, часто розбирають завдання розстановки 8 ферзів на шахівниці. Оскільки це завдання більше 200 років тому вирішив ще Леонард Ейлер, то навряд чи зараз є, хоч якась можливість помилитися, при описі вирішення цієї проблеми. Але в зв'язку з тим, що технічний прогрес не стоїть на місці і прямий перебір 40320 (= 8!) Варіантів, а саме стільки варіантів розстановки ферзів треба розглянути, щоб вирішити задачу, не є чимось незвичайним і здійснюється, навіть не на найсучаснішому комп'ютері, менше ніж за хвилину, то і скорочення такого перебору прямо скажемо не дуже вражає, але як приклад варто розглядати.

Приклад: «Завдання про розстановку ферзів». На шаховій дошці N \* N потрібно розставити N ферзів таким чином, щоб жоден ферзь не атакувати іншого. Відзначимо наступне. Всі можливі способи розстановки ферзів - З ^ 2 (близько 4,4 \* 109 для N = 8). Кожен стовпець містить щонайбільше одного ферзя, що дає тільки NN розстановок (1,7 \* 107 для iV = 8). Ніякі два ферзя не можна поставити в один рядок, а тому, для того щоб вектор (at, a2, ..., aN) був рішенням, він повинен бути перестановкою елементів (1,2, ..., N), що дає тільки N \ (4,0 \* 104 для N == 8) можливостей. Ніякі два ферзя не можуть перебувати на одній діагоналі, це скорочує число можливостей ще більше (для 2V = 8 в дереві залишається 2056 вузлів). Отже, за допомогою ряду спостережень ми виключили з розгляду велике число можливих розстановок N ферзів на дошці розміром N \* N. Використання подібного аналізу для скорочення процесу перебору називається пошуком з обмеженнями або відсіканням гілок в зв'язку з тим, що при цьому видаляються піддерева з дерева. Друге. Іншим удосконаленням є злиття, або склеювання, гілок. Ідея полягає в тому, щоб уникнути виконання двічі однієї і тієї ж роботи: якщо два або більше піддерев даного дерева ізоморфні, ми хочемо досліджувати лише одна з них. У задачі про ферзів ми можемо використовувати склеювання, помітивши, що якщо af ^ N / 2], то знайдене рішення можна відобразити і отримати рішення, для якого a ^ N / 2]. Отже, дерева, відповідні, наприклад, випадків А1-2 та UJ ^ -N-1, ізоморфні.

В задачах лінійного та нелінійного програмування керований процес, відносно якого приймається рішення, вважається статичним, тобто незалежним від часу, тому оптимальне рішення (керування) знаходиться лише на один етап планування. Такі задачі називаються одноетапними або однокроковими. В задачах динамічного програмування керований процес залежить від часу, тому знаходиться ряд рішень (для кожного етапу), що забезпечують оптимальний розвиток всього процесу в цілому. Задачі динамічного програмування називаються багатоетапними або багатокроковими. Динамічне програмування являє собою математичний апарат, що дозволяє реалізувати оптимальне керування багатокроковими процесами прийняття рішень. Зауважимо, що досить часто шляхом певного перефразування і деякі статичні задачі можуть бути зведені до багатоетапних.

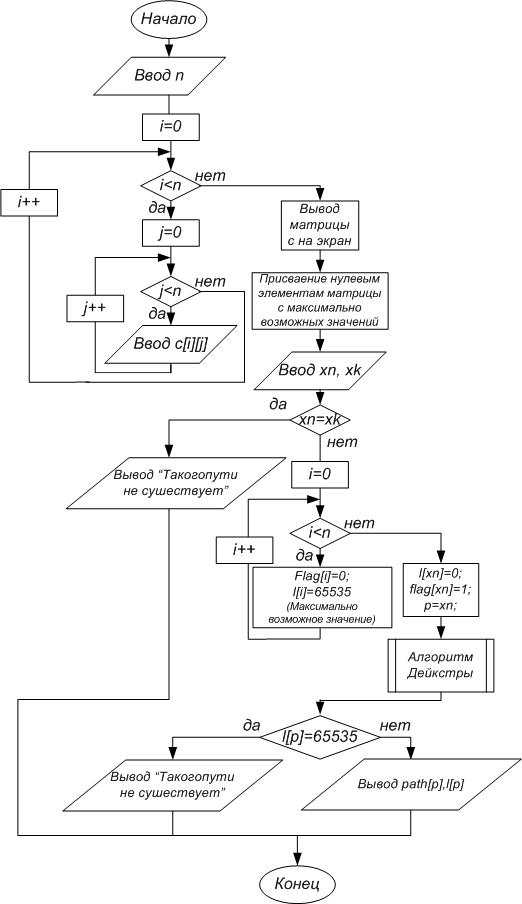
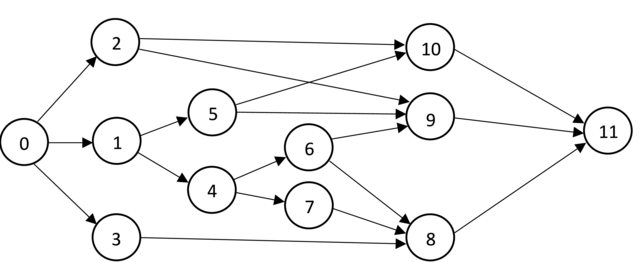


Рисунок 2Приклад вирішення задачі на динамічне програмування за допомогою блок-схеми

* **Алгоритм Дейкстри.**

Алгоритм Дейкстри (також званий пошуком з рівномірною вартістю) дозволяє нам ставити пріоритети дослідження шляхів. Замість рівномірного дослідження всіх можливих шляхів він віддає перевагу шляхах з низькою вартістю.

Алгоритм голландського вченого Едсгер Дейкстри знаходить все найкоротші шляхи з однієї наперед заданій вершини графа до всіх інших.

Мінусом даного методу є неможливість обробки графів, в яких є ребра з негативним вагою. Якщо, наприклад, деяка система передбачає збиткові для Вашої фірми маршрути, то для роботи з нею варто скористатися відмінним від алгоритму Дейкстри методом.

Рисунок 3 Граф

Для програмної реалізації алгоритму знадобитися два масиви: логічний **visited** - для зберігання інформації про відвідані вершинах і чисельний **path\_lenght**, в який будуть заноситися знайдені найкоротші шляхи. Отже, є граф **matrix** = (V, E). Кожна з вершин входять в безліч V, спочатку відзначена що не відвідана, т. Е. Елементам масиву visited присвоєно значення false.

Оскільки найвигідніші шляхи тільки належить знайти, в кожен елемент вектора distance записується таке число, яке свідомо більше будь-якого потенційного шляху (зазвичай це число називають нескінченністю, але в програмі використовують, наприклад максимальне значення конкретного типу даних). В якості вихідного пункту вибирається вершина start і йому приписується нульовий шлях: path\_lenght [start] = 0, оскільки немає ребра з start в start.

Алгоритм Дейкстри будує найкоротші шляхи, що ведуть з вихідної вершини графа до решти вершин цього графа (якщо такі є).

У процесі роботи алгоритму послідовно позначаються розглянуті вершини графа. Причому вершина, позначена останньої (на даний момент) розташована ближче до вихідної вершині, ніж всі непомічені, але далі, ніж всі помічені.Спочатку позначається вихідна вершина; Наступного, очевидно, буде позначена вершина, найближча до вихідної, і суміжна з нею.

Нехай на якомусь етапі вже позначено кілька вершин. Відомі найкоротші шляхи, що ведуть з вихідної вершини до поміченим. Для кожної з непомічених вершин проробимо наступне:

1. Розглянемо всі дуги, провідні з помічених вершин в одну непоміченими. Кожна така дуга є останньою дугою на шляху з вихідної вершини в цю непоміченими.

2. Виберемо з цих шляхів найкоротший. А потім виберемо серед них самий короткий до всіх непомічених вершин, і позначимо вершину, до якої він веде.

Алгоритм завершиться, коли будуть помічені всі досяжні вершини.

В результаті роботи алгоритму Дейкстри будується Дерево найкоротших шляхів.

* **Жадібний алгоритм(пошук)**

Є багато методів вирішення тих чи інших завдань: динамічне програмування, перебір. Не менш відомими та досить поширеними є жадібні алгоритми.

Джудеа Перл описав пошук «Найкращий — перший», взявши як оцінку вузла n значення деякої «евристичної функції оцінки f(n), яка, взагалі кажучи, може залежати від опису вузла n, опису мети, інформації, яка зібрана пошуком на даний момент, і головне, від будь-яких додаткових знань про предметну галузь».

Деякі автори використовували пошук «Найкращий — перший» спеціально для опису пошуку з евристикою, щоб спробувати передбачити, наскільки близько знаходимося до фінального стану, так що шляхи, які мають найкращу евристичну оцінку, розглядаються першими. Цей специфічний тип пошуку називається «жадібним пошуком».

Простими словами, жадібний алгоритм - це алгоритм, який на кожному кроці робить локально найкращий вибір, сподіваючись, що підсумкове рішення буде оптимальним.

Наприклад, алгоритм Дейкстри знаходження найкоротшого шляху у графі цілком собі жадібний, тому що ми щокроку шукаємо вершину з найменшою вагою, у якій ми ще бували, після чого оновлюємо значення інших вершин.

Жа́дібний алгори́тм — простий і прямолінійний евристичний алгоритм, який приймає найкраще рішення, виходячи з наявних на кожному етапі даних, не зважаючи на можливі наслідки, сподіваючись урешті-решт отримати оптимальний розв'язок. Легкий в реалізації і часто дуже ефективний за часом виконання. Багато задач не можуть бути розв'язані за його допомогою.

Зазвичай, жадібний алгоритм базується на п'яти принципах:

1. Набір можливих варіантів, з яких робиться вибір;
2. Функція вибору, за допомогою якої знаходиться найкращий варіант, який буде додано до розв'язку;
3. Функція придатності, яка визначає придатність отриманого набору;
4. Функція цілі, передає значення розв'язку або частковому розв'язку;
5. Функція розв'язку, яка вказує на те, що кінцевий розв'язок знайдено.

Придатний набір варіантів — такий, що обіцяє не просто отримання розв'язку, а отримання оптимального розв'язку задачі.

На відміну від динамічного програмування, за якого задача розв'язується знизу догори, за жадібної стратегії це робиться згори донизу, шляхом здійснення одного жадібного вибору за іншим, зведенням великої задачі до малої.

Жадібний алгоритм добре розв'язує деякі задачі, а інші — ні.

Принцип жадібного вибору:

До оптимізаційної задачі можна застосувати принцип жадібного вибору, якщо послідовність локально оптимальних виборів дає глобально оптимальний розв'язок. В типовому випадку доведення оптимальності здійснюється за такою схемою: спочатку доводиться, що жадібний вибір на першому етапі не унеможливлює шляху до оптимального розв'язку: для будь-якого розв'язку є інший, узгоджений із жадібним і не гірший від першого. Далі доводиться, що підзадача, яка виникла після жадібного вибору на першому етапі, аналогічна початковій, і міркування закінчується за індукцією. Інакше кажучи, за жадібного алгоритму ніколи не переглядаються попередні вибори для здійснення наступного, на відміну від динамічного програмування.

**А\***

Алгоритм пошуку А\* належить до евристичних алгоритмів пошуку. Використовується для пошуку найкоротшого шляху між двома вершинами графу з додатніми вагами ребер. Описаний 1968 р. Пітером Хартом, Нільсом Нільсоном та Бертрамом Рафаелєм.

Алгоритм використовує допоміжну функцію (евристику), аби скеровувати напрям пошуку та скорочувати його тривалість. Алгоритм повний в тому сенсі, що завжди знаходить оптимальний розв'язок, якщо він існує.

Алгоритм A \* також оптимально ефективний для заданої евристики h.

Це означає, що будь-який інший алгоритм досліджує не менш вузлів, ніж A\* (За винятком випадків, коли існує кілька приватних рішень з однаковою евристикою, точно відповідної вартості оптимального шляху). Алгоритм пошуку шляху з пункту А в пункт Б.

У той час як алгоритм A\* оптимальний для «випадково» заданих графів, немає гарантії, що він зробить свою роботу краще, ніж більш прості, але і більш інформовані щодо проблемної області алгоритми. Наприклад, в якомусь лабіринті може знадобитися спочатку йти у напрямку від виходу і тільки потім повернути назад. В цьому випадку обстеження спочатку тих вершин, які розташовані ближче до виходу (по прямій дистанції), буде втратою часу.

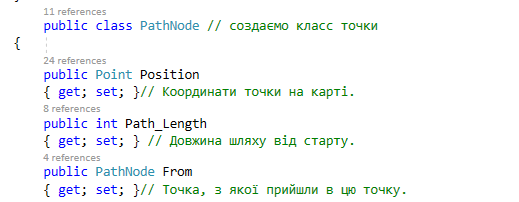
Алгоритм Дейкстри хороший в пошуку найкоротшого шляху, але він витрачає час на дослідження всіх напрямків, навіть безперспективних. Жадібний пошук досліджує перспективні напрямки, але може не знайти найкоротший шлях. Алгоритм A \* використовує і справжнє відстань від початку, і оцінене відстань до цілі. Код дуже схожий на алгоритм Дейкстри.

# Головна частина

Для вирішення даного завдання найвигідніше використовувати алгоритм А \*, так як він не виконує зайві перебори, як Алгоритм Дейкстри. А \* це суміш кращих характеристик 2 алгоритмів (Алгоритму Дейкстри і Жадібного пошуку). Для його реалізації знадобилося у використанні деяких систем, такі як System.Linq і System.Collections.ObjectModel. Ці системи допомагає в роботі з колекціями, це і є головна відмітна риса від Алгоритму Дейкстри (в А \* робота зі списками, в алгоритм Дейкстри з масивами). Вони являються більш зручними при роботі з з пошуком самого найкоротшого і найбільш оптимального шляху.

Для того щоб його реалізувати в простір C # Windows Forms Applications, я вирішив прочитати інформацію у деяких книгах і сайтах. Приведенний нижче код я використовував для знаходження очікуваного шляху (в незваженому орієнтованому графі):

1. По-перше,я створив клас точки, за допомогою яких ми будемо знаходити сусідні точки.

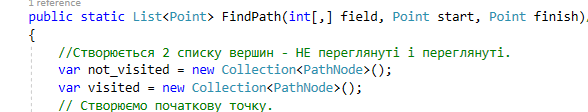


1. По-друге, я створив 2 змінних, які будуть показувати очікуване і реальну відстань до цілі(відстані між вершинами однакові).

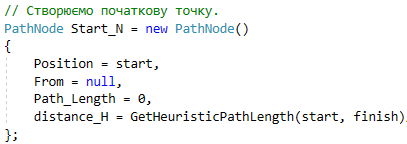
Зображення, що містить текст

Автоматично згенерований опис

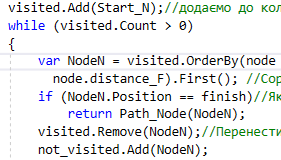
1. Створюю список, в який я буду записувати знайдений шлях, а також створюємо 2 колекції в які будемо записувати переглянуті і не переглянуті (те ж саме в звичайному алгоритму Дейкстри, тільки з масивами).



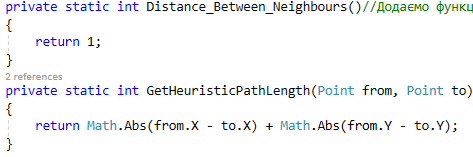
1. Створюємо початкову точку, і говоримо що відстань від неї дорівнює 0, так як ми ще не почали рухатися. Паралельно з цим викликаємо метод GetHeuristicPathLength(Еврестичний метод), який знаходить відстань між 2 точками, але цей метод не завжди знаходить найкоротший шлях, тому ми записуємо його в distance\_H (очікуваний шлях).

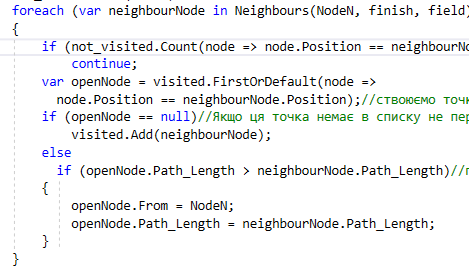
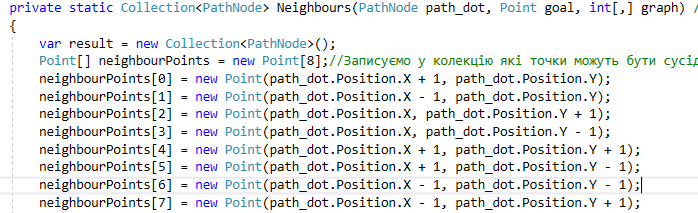


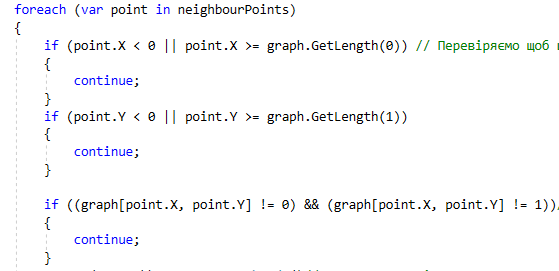
1. У колекцію "переглянуті вершини" додаємо початкову точку і ставимо її найпершою за допомогою методу OrderBy, так як в даний момент ми на ній знаходимося. Також сосдаем умова, що якщо поточна позиція це координата кінцевої точки, то шлях знайдений.



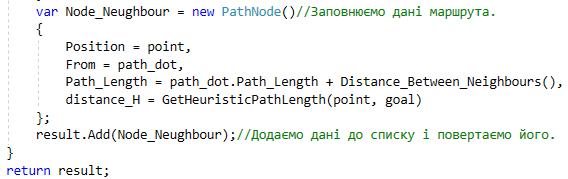
1. Переглядаю сусідні точки в колекції, і створюю умова: якщо поточна позиція це сусідні координати, то скористаємося методом FirstOrDefault, який вирощує перший елемент колекції. І записуємо координати в окрему змінну. Продовжуємо переглядати колекцію. Створюємо умова, що якщо точка, в яку ми записали координати 1 відвідується вершини, не перебувати в колекції "переглянуті вершин", то ми її туди додаємо, а інакше ми перевіряємо, чи рухаємося ми до мети найкоротшим шляхом, якщо немає, то ми записуємо шлях який пройшли, як найкоротший шлях.

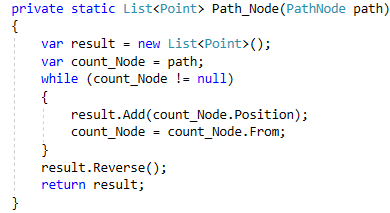


1. Додаємо функцію відстані між сусідніми точками і заповнюємо її значення. Також додаємо функцію приблизної оцінки відстані до цілі.
2. Утворюємо колекцію в якій знаходиться сусідні точки. Записуємо у колекцію які точки можуть бути сусідніми. 
3. Перевіряємо щоб ці точки не виходили за межу. Перевіряємо, щоб можна було переходити з однієї вершини на іншу.



1. Створюємо нову змінну, і заповнюємо туди дані маршруту, який був побудований.Повертаємо дані.



1. Створюємо новий список, в який заповнюємо координати виконаного маршруту і повертаємо його. 
2. На самому початку проекту створюємо список, в який заповнюємо 2 координати (початкова та кінцева).

Також я використав інший алгоритм Дейкстри, наведений нижче:

1. Створюємо декілька класів для заповнення координат(назви) вершин, дуги та їх вагу, відвідані/невідвідані вершини.

Зображення, що містить текст

Автоматично згенерований опис

Зображення, що містить текст

Автоматично згенерований опис

Зображення, що містить текст

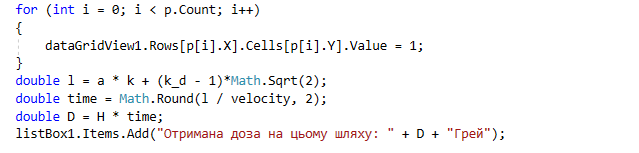
Автоматично згенерований опис

1. Зображення, що містить текст

   Автоматично згенерований описЗображення, що містить текст

   Автоматично згенерований опис Після отримання інформації про найближчі вершини, створюється їх список і викликається клас Algoritm, де в методі FindShortestPath знаходяться найкоротші вершини шляхом порівняння ваги цих вершин.
2. Обчислення суми ваг ребер для наступної вершини, де info - інформація про поточну вершинуЗображення, що містить текст

   Автоматично згенерований опис.



1. Знаходження шляху та повернення отриманого результату (цикл триватиме, доки знайде потрібний шлях чи поверне значення null, якщо такого немає).

Зображення, що містить текст

Автоматично згенерований опис

1. Порівнюємо результати отримані з використанням евристики та використанням Алгоритму Дейкстри.

Зображення, що містить текст

Автоматично згенерований опис

6.

Підраховуємо скільки разів ми робили кроки, щоб обчислити пройдений шлях, після чого знаходимо час шляху і множимо його на рівень радіації, щоб вивести в listbox отриману дозу на цьому шляху. Зображення, що містить текст

Автоматично згенерований опис

# Крупноблочна блок-схема

Все, що було описано вище показано на блок-схемі:

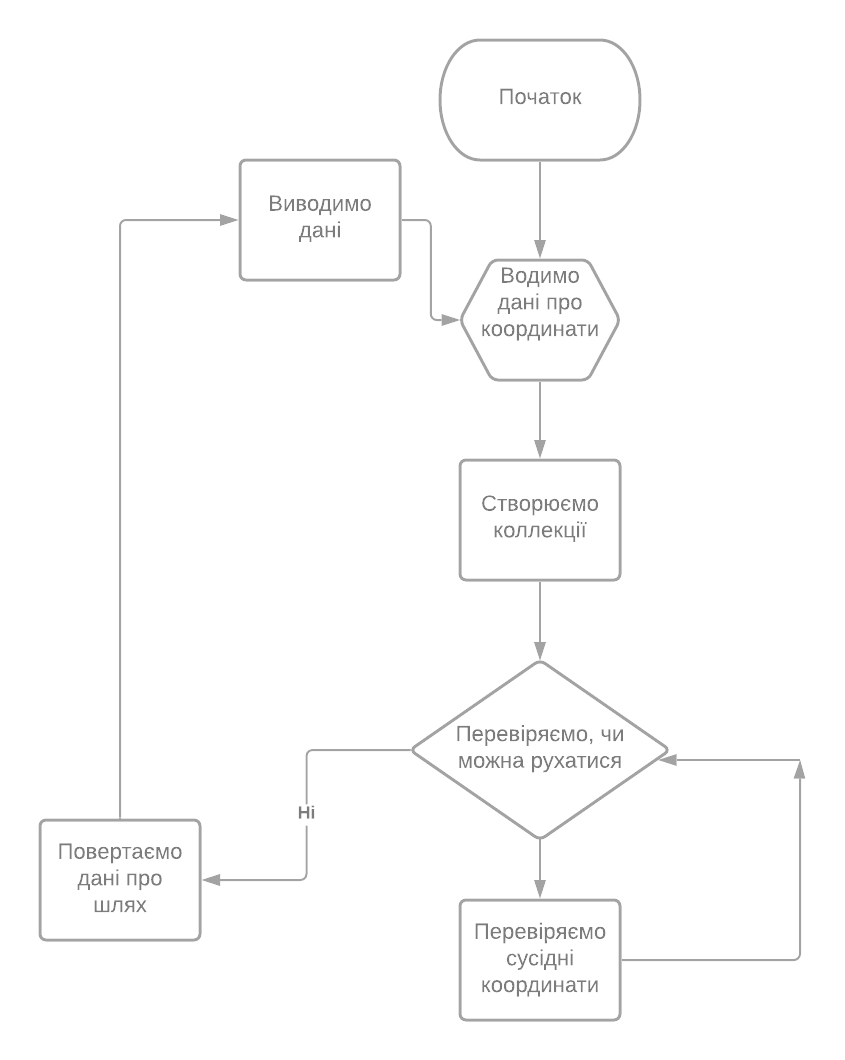


Рисунок 4Блок-Схема вирішення задачі за допомогою алгоритма А\*

# Програмно-апаратні вимоги

* Процесор частотою 1.5 GHz або вище
* 4096 MB RAM
* 100 MB вільного місця на жорсткому диску
* Роздільна здатність монітору 1440 х 900 або вище

Підтримування операційні системи:

* Windows 8(64 біт)
* Windows 10(64 біт)

# Перелік використаних програмних проектів

У цьому проекті, я використав Visual Studio 2017 і Microsoft Word 2016 (для написання супровідної документації). Мова програмування використаний для написання даної програми - С # .NET Core 4.7.2.

Використані програми:

1. Visual Studio 2017
2. Microsoft Word 2016
3. Adobe Photoshop 2016

# Посібник користувача

**Граф** являє собою кінцеву сукупність об'єктів, які з'єднуються між собою зв'язками. Об'єктами, з яких складається граф, є вершини, а зв'язку представлені у вигляді ребер.

**Вершина** є точкою, в якій можуть сходитися чи з якої можуть виходити ребра.

**Ребро** є лінією, яка з'єднує пару суміжних вершин графа.

Якщо вершини графа з'єднані ребром, то вони називаються суміжними, в іншому випадку - несуміжні.

**Евристика** — термін стосується способу вибору цілі або напрямку в розв'язуванні задачі, правильність якого на кожному кроці невідома або не може бути підтверджена. Такі методи як генетичний алгоритм або нейронна сітка використовують для приймання рішень евристичні способи, які можуть ґрунтуватися на суто емпіричній інформації, що не піддається суворій раціоналізації.

# Висновки

# Підбиваючи підсумки даної роботи, поставлене завдання було вирішено. Але я зіткнувся з багатьма проблемами, в тому числі і з тим, що даний алгоритм не є найкращим у пошуку найкоротшого шляху, оскільки сам по собі алгоритм Дейкстри є жадібним (тобто він шукає найкоротший шлях між двома сусідніми вершинами у матриці суміжності). Локально він знаходить правильні шляхи, але в деяких випадках цей шлях не є оптимальним, тому що він може робити зайві кроки. Хорошою заміною даного алгоритму є алгоритм Флойда-Уоршелла, але він також має свої недоліки: оскільки він шукає відстань між усіма вершинами, то йому потрібний складний алгоритм з використанням 5 і більше вкладених циклів з обмежуючими умовами на сусідні точки (ми не можемо перейти з початкової точки відразу в кінцеву), були проведені тести і при запуску програми вона могла вантажитися близько 15 хвилин для показу шляху матриці 10Х10. Хоча алгоритм Дейкстри робить це за кілька секунд. Кожен з алгоритмів має свої недоліки та переваги, але все ж таки вони є зручним інструментом для вирішення багатьох завдань, які необхідні для моделювання реальних ситуацій.

# Список джерел інформації

* «ЗБІРКА НАУКОВИХ ПРАЦЬ НГТУ»:

<https://viduus.net/wp-content/uploads/2018/02/Rihter-Dzh.-CLR-via-C.-Programmirovanie-na-platforme-Microsoft-.NET-Framework-4.5-na-yazyke-C-Master-klass-2013.pdf>

[file:///C:/Users/1/Downloads/sbornik\_2\_88\_13824.pdf](C://Users/1/Downloads/sbornik_2_88_13824.pdf)

* Сайт для пояснення, що таке А\*: <https://habr.com/ru/post/444828/>
* Книга «Программирование в алгоритмах»:

<https://algo.at.ua/_ld/0/3_....pdf>