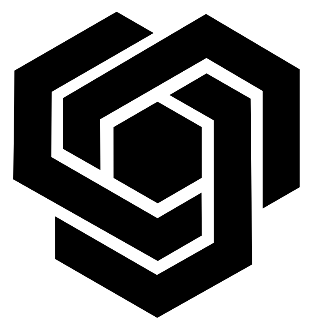
**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ СОФИЯ**

КУРСОВА РАБОТА ПО

Синтез и анализ на алгоритми (САА)

на тема:

Алгоритми за графи. Алгоритъм на Dijkstra.

Динамична визуализация.

Разработил:Мехмед Весалов Проверил:

Специалност: ИТИ

Курс: II

Фак.№: 501223036

София2024

Съдържание

[Увод 2](#_Toc180201774)

[Рекурентни числови редици: 3](#_Toc180201775)

[Алгоритъм на Dijkstra: 3](#_Toc180201776)

[Рекурентна числова редица 3](#_Toc180201777)

[Дефиниция на рекурентна числова редица 4](#_Toc180201778)

[Рекурсивни функции 4](#_Toc180201779)

[Пример за изчисляване на елементи от рекурентна числова редица с nnn от интервала [0...20] 5](#_Toc180201780)

[Алгоритъм на Dijkstra 5](#_Toc180201781)

[Динамична визуализация 6](#_Toc180201782)

[Заключение 6](#_Toc180201783)

Увод

Целта на тази курсова работа е да се изследва и анализира ролята на рекурентните числови редици и алгоритъма на Dijkstra в областта на компютърните науки, като се обърне внимание на тяхната важност и приложение в съвременните алгоритми и задачи за графи.

Рекурентни числови редици:

Рекурентните числови редици са важна част от математиката и компютърните науки, тъй като те представляват последователност от числа, където всеки елемент се определя на база на предишните елементи чрез определена формула. Такива редици намират приложение в различни алгоритмични задачи, свързани с оптимизация, симулация и прогнозиране. Чрез използването на рекурсивни функции, ние можем ефективно да изчислим всеки елемент от редицата, като това предоставя основа за изграждане на по-сложни алгоритми и решения на проблеми.

Алгоритъм на Dijkstra:

Алгоритъмът на Dijkstra е един от най-важните алгоритми за намиране на най-краткия път в граф. Той се използва широко в приложения като компютърни мрежи, географски информационни системи и различни форми на оптимизация. Този алгоритъм има критично значение в ситуации, където трябва да се намери оптимален маршрут или минимален разход в системи, представени като графи (например транспортни мрежи, маршрути за доставка или комуникационни мрежи).

Важност и приложение:

Рекурентните числови редици и алгоритъмът на Dijkstra демонстрират същността на рекурсията и графовите структури в компютърните науки. Те са основни компоненти в задачи за анализ на данни, решаване на оптимизационни проблеми и визуализиране на сложни системи. Като изучаваме тези концепции, ние придобиваме по-задълбочено разбиране за принципите на алгоритмите и тяхната роля в развитието на ефективни решения на реални проблеми.

Тази курсова работа изследва и анализира значението на рекурентните числови редици и алгоритъма на Dijkstra в контекста на компютърните науки, като се поставя акцент върху тяхната важност и практическо приложение в съвременните алгоритми и задачи, свързани с графи.

Рекурентната числова редица представлява последователност от числа, при която всеки елемент се изчислява на базата на предходните елементи, използвайки дадена формула. Такива редици са широко използвани в математиката и алгоритмите за решаване на задачи, свързани с оптимизация и симулация.

Алгоритъмът на Dijkstra е предназначен за намиране на най-краткия път между възли в граф с неотрицателни тегла. Той се използва в различни приложения, като мрежова оптимизация и навигационни системи, за ефективно определяне на оптималния маршрут.

Динамичната визуализация е мощен метод за представяне на алгоритми, който позволява на потребителите да видят как стъпка по стъпка се извършват операциите. Тя е особено полезна за илюстриране на сложни алгоритми като Dijkstra, помагайки за по-добро разбиране на техния работен процес.

Рекурентна числова редица

Рекурентна числова редица е последователност от числа, в която всеки следващ елемент се определя на база на един или повече предишни елементи чрез фиксирана математическа връзка или формула. Рекурентните редици често се дефинират с начален или базов елемент (или няколко) и рекурентна формула, която описва как се генерира всеки следващ елемент.

Пример за рекурентна числова редица е редица на Фибоначи

* Базови случаи: F(0) = 0, F(1)=1
* Рекурентна формула: F(n) = F(n-1) + F (n-2) за n ≥ 2

Тази редица започва с числа 0 и 1, а всяко следващо число е сумата на предишните две. Например: 0, 1, 1, 2, 3, 5, 8, 13 и т.н

Дефиниция на рекурентна числова редица

Рекурентна числова редица е последователност от числа, в която всеки следващ елемент се определя въз основа на един или няколко предходни елемента чрез математическа формула. Тези редици се характеризират с рекурентни връзки, които описват зависимостите между елементите на редицата. Те могат да имат базови случаи (начални стойности), които задават първите един или няколко елемента от редицата.

**Ако an е елемент от редицата на позиция n, тогава рекурентната формула може да бъде представена като: an​=f(an−1​,an−2​,...,an−k​)**

където f е някаква функция на предходните k елемента от редицата, а n≥k.

Примери за рекурентни числови редици:

Редица на Фибоначи

Базови случаи: F(0) = 0, F(1) =1

Рекурентна формула: F(n) = F(n−1) + F(n−2), за n≥2

Пример: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

Рекурентните редици играят важна роля в компютърните науки, особено в контекста на алгоритми, свързани с оптимизация, анализ на алгоритми и решаване на задачи, които могат да бъдат представени чрез рекурсивни структури. Те предоставят основа за разширяване на концепции като динамично програмиране, където решенията на подзадачите се съхраняват и използват за решаване на по-големи проблеми.

Рекурсивни функции

Рекурсивна функция е функция, която извиква самата себе си, за да реши по-малка версия на същия проблем. Тази техника се използва широко за задачи, които могат да бъдат разделени на подпроблеми с подобна структура, какъвто е случаят с изчисляването на елементи от рекурентна числова редица.

Как работят рекурсивните функции?

Рекурсивните функции работят, като следват два основни принципа:

1. Базов случай (край на рекурсията): Това е условие, което определя кога рекурсията трябва да спре. Без базов случай функцията би продължила да извиква сама себе си безкрайно, което води до грешка (стеков препълване).
2. Рекурсивен случай (разделяне на проблема): Това е случаят, при който проблемът се разбива на по-малки подпроблеми. Функцията се извиква със стойности, които приближават решението до базовия случай.

Предимства и недостатъци на рекурсивните функции:

**Предимства**:

* Кратък и ясен код, който точно следва логиката на рекурентната формула.
* Лесно за разбиране при проблеми, които естествено се дефинират рекурсивно (например рекурентни числови редици, дървовидни структури и др.).

**Недостатъци**:

* Икономия на паметта: Рекурсивните функции използват системния стек за всяко рекурсивно извикване, което може да доведе до стеково препълване при дълбока рекурсия.
* Ефективност: Без оптимизация, като мемоизация или динамично програмиране, рекурсивните функции могат да извършват многократни преизчисления на едни и същи стойности, което води до неефективност.

Пример за изчисляване на елементи от рекурентна числова редица с nnn от интервала [0...20]

Да вземем за пример редицата на Фибоначи, която е класическа рекурентна числова редица, и да изчислим конкретни стойности на nnn, където nnn е в интервала [0...20].

Когато n=20

Изчисляваме:F (20):

F(20)=F(19)+F(18)

Продължаваме, като използваме рекурентната формула за всяка следваща стойност до n=20. След изчисленията получаваме:

F(0) = 0, F(1) = 1, F(2) = 1, F(3) = 2, F(4) = 3, F(5)=5

F(6) = 8, F(7) = 13, F(8) = 21, F(9) = 34, F(10) = 55

F(11) = 89, F(12) = 144, F(13) = 233, F(14) = 377

F(15) = 610, F(16) = 987, F(17) = 1597, F(18) = 2584, F(19) = 4181

F(20) = 6765

Резултат:F(20) = 6765

Тези пример демонстра как, чрез рекурентната формула и базовите случаи, могат да се изчислят стойностите на рекурентната числова редица за всяко n от интервала [0...20].

Алгоритъм на Dijkstra

Алгоритъмът на Dijkstra е един от най-известните алгоритми за намиране на най-краткия път в граф. Той решава проблема за намиране на най-краткия път от един изходен връх до всички останали върхове в граф, като приема, че всички ръбове имат неотрицателни тегла.

Как работи алгоритъмът?

* За всеки връх в графа се поддържа минимално разстояние d(v), което първоначално се задава като безкрайност (∞) за всички върхове, освен за началния връх s, за който d(s) = 0.
* Избиране на връх. Избира се връх u, който все още не е обработен и има минимално разстояние d(u) от началния връх. Този връх се маркира като обработен, защото вече знаем, че не съществува по-кратък път до него.
* Актуализиране на съседите. За всички съседни върхове на uuu (които все още не са обработени), се актуализират разстоянията d(v) Ако разстоянието до съседен връх v през u е по-кратко от текущо известното разстояние d(v), стойността на d(v)се обновява.
* Повторение. Стъпките се повтарят, докато всички върхове не бъдат обработени или докато минималните разстояния до целевите върхове са установени.

Динамична визуализация

Динамичната визуализация е техника, която използва визуални инструменти и анимации, за да илюстрира процеса на работа на даден алгоритъм в реално време. Тя позволява да се видят междинните стъпки на алгоритъма, как той взима решения и как се променя състоянието на входните данни с всяка операция. Основната ѝ цел е да направи сложните алгоритми по-лесни за разбиране, като показва визуално промените в данните. Тази техника е особено полезна за студенти, преподаватели и програмисти, които искат да анализират работата на алгоритмите.

Динамичната визуализация показва как алгоритъмът преминава през различните си стъпки, като например прилагане на операции върху данни, избор на върхове и ръбове при графи или разпределяне на памет при сортиращи алгоритми. В някои случаи се осигурява и интерактивност, като потребителите могат да променят параметрите в реално време и да наблюдават как това влияе върху изпълнението на алгоритъма.

Един от често срещаните примери за използване на динамична визуализация е в алгоритмите за графи. Например, при алгоритъма на Dijkstra визуализацията може да покаже как се избира начален връх, как се актуализират разстоянията до съседните върхове и как постепенно се изгражда най-краткият път. Всеки нов връх и ръб, които се включват в пътя, могат да бъдат подчертани или анимирани. При рекурсивните алгоритми, като тези за изчисляване на елементи от рекурентни редици, визуализацията може да покаже дървото на рекурсията и как всяка функция извиква себе си, както и как се връщат резултатите към предходните нива.

Заключение

В заключение, настоящата курсова работа разглежда две важни теми в компютърните науки: рекурентните числови редици и алгоритъма на Dijkstra. Чрез детайлно описание на рекурентните редици, тяхното изчисляване с рекурсивни функции и конкретни примери за различни стойности на nnn, се подчертава значението на тези математически концепции за алгоритмичните решения. Рекурентните редици са основен инструмент в различни области на програмирането и алгоритмите, особено когато се изисква ефективно решаване на задачи с повтарящи се структури.

Алгоритъмът на Dijkstra, от своя страна, е ключов за решаването на проблеми, свързани с графи и оптимизация, като намирането на най-кратки пътища. Неговата ефективност и широкото му приложение го правят незаменим в множество практически задачи, като навигационни системи и мрежови протоколи. Разгледаната динамична визуализация допълнително подчертава значението на визуализацията за по-доброто разбиране на алгоритмите и техните междинни стъпки, което е от голяма полза за образованието и анализа на алгоритмични решения.

Съчетанието на тези теми показва как сложните теоретични концепции могат да бъдат приложени на практика в реални задачи, като същевременно да се представят по начин, който е лесно достъпен и разбираем чрез визуализация.

Обяснение на кода за алгоритъма на Dijkstra.

Клас Graph (Граф): Представлява графа като списък на съседите, където всеки ключ е връх, а стойността е списък от ръбове към съседните върхове.

A screen shot of a computer

Description automatically generated

Създаваме клас Vertex (Връх): Представлява връх (възел) в графа. Съхранява уникална стойност и имплементира методи \_\_hash\_\_ и \_\_eq\_\_, за да бъде използван като ключ в речници и за сравняване на обекти.

A black background with white text

Description automatically generated

Клас Edge (Ръб): Представлява ръб между два върха. Съхранява distance (теглото на ръба) и vertex (свързания връх).

A computer screen shot of a black background

Description automatically generated

Тази приоритетна опашка използва (heap) за управление на върховете по приоритет, като по-ниските разстояния имат по-висок приоритет.

* add\_task добавя задача с определен приоритет. Ако задачата вече съществува, приоритетът се актуализира.
* update\_priority актуализира приоритета на съществуваща задача.
* pop\_task премахва и връща задачата с най-нисък приоритет (най-кратко разстояние).

A screenshot of a computer program

Description automatically generated

A computer screen with text on it

Description automatically generated

**Функция dijkstra за намиране на най-краткия път**

Тази функция прилага алгоритъма на Дейкстра, за да намери най-краткия път между два върха:

* Начални структури: Инициализира previous (предходни върхове), visited (посетени върхове) и distances (разстояния от началния връх).
* Цикъл за търсене на най-кратък път: Обработва върховете в приоритетна опашка, добавя съседите в опашката с актуализирани разстояния, ако се намери по-кратък път.
* Край: Когато се достигне крайният връх, функцията извежда най-краткото разстояние и пътя.

A computer screen shot of a program code

Description automatically generatedA computer screen shot of text

Description automatically generated

**Пример за използване:**

В примера се създава граф със списък на съседите и се изпълнява алгоритъмът на Дейкстра, за да се намери най-краткият път от връх A до връх H.

A screen shot of a computer program

Description automatically generated

Обяснение на кода за рекурентна числова редица.

В тази курсова работа се разглежда ефективен метод за изчисляване на числата на Фибоначи чрез използване на матрично умножение. Методът се базира на линейно-алгебрични свойства, които позволяват значително оптимизиране на изчисленията. Благодарение на това оптимизиране, алгоритъмът достига логаритмична времева сложност О(log n), което прави възможно бързото и ефективно намиране на числата на Фибоначи дори при много големи стойности на n

**Функция за умножение на матрици**

A group of numbers on a black background

Description automatically generated

Тази функция приема две матрици (m1 и m2) и връща нова матрица, която е резултат от умножението им.

**Матрици**: Всяка матрица е представена като двумерен списък. В случая работим с 2x2 матрици.

**Формула**: Умножението на матриците следва стандартните правила:

* (a11 \* b11 + a12 \* b21) за елемента на първия ред и първата колона.
* (a11 \* b12 + a12 \* b22) за елемента на първия ред и втората колона.
* Аналогично за втория ред.

### **Функция за повдигане на матрица на степен**

A computer screen shot of a black screen

Description automatically generated

Тази функция приема матрица и число power и я повдига на power.

**Единична матрица**: Започваме с единичната матрица, която е неутралният елемент за матричното умножение.

**Бинарно повдигане**:

* Проверява дали power е нечетно. Ако е, умножава текущия резултат с матрицата.
* Независимо от проверката, матрицата се умножава сама със себе си, за да се подготви за следващата итерация.
* Степента power се дели на 2 (цялостно).

**Връщане на резултата**: В края функцията връща получения резултат.

**Функция за взаимодействие с потребителя**

**A screenshot of a computer program

Description automatically generated**

Тази функция осигурява взаимодействие с потребителя, като получава вход и извършва изчисления.

**Потребителски вход**: Извиква input() за въвеждане на число или 'q' за изход.

**Проверка на входа**:

* Проверява дали потребителят е въвел 'q' и, ако е, изходи от функцията.
* Превръща входа в цяло число и проверява дали е отрицателно.

**Изчисляване и показване на резултата**: Ако входът е валиден, извиква fibonacci\_matrix и извежда резултата.

**Обработка на грешки**: При невалиден вход (например, нечислови символи) извежда съобщение за грешка.

**Рекурсивно извикване**: В края функцията се извиква отново, позволявайки на потребителя да направи нова заявка.

**Стартиране на програмата**

A screen shot of a computer

Description automatically generated

Извиква функцията run\_fibonacci\_program, за да стартира програмата.

Този код е организиран в логически структурирани функции, които позволяват лесно изчисляване на числата от редицата на Фибоначи, използвайки матрично умножение. Взаимодействието с потребителя е интуитивно, а обработката на грешки осигурява устойчивост на програмата.