*Изпитна тема № 18:* **Алгоритми и структури от данни**

Алчни алгоритми – понятие, примери, решаване на задачи с алчни алгоритми. Рекурсия и рекурсивни алгоритми – понятие, примери за рекурсивни алгоритми и алгоритми по схемата на пълното изчерпване (backtracking). Комбинаторни съединения и алгоритми за генерирането им: вариации, комбинации, пермутации – понятия, алгороитми, прилики и разлики. Динамично оптимиране – техника, едномерни и двумерни задачи. Теория на графите – основни понятия, представяне на графи в програмен код. Алгоритми върху графи – алгоритми за обхождане на граф, графично представяне на алгоритмите и имплементация с програмен код.

1. Описва понятието алчен алгоритъм. Дава пример за алчни алгоритми.
   1. Алчни алгоритми са клас от алгоритми които са добре в това да намерят локално оптимизирани решения, като те във всяка дадена стъпка, правят най-доброто потенциално решение до сега, и никога не се връщат назад.
   2. Примери за такъв вид алгоритъм би бил, алгоритъма за монетите и алгоритъма за раницата.

**Алчни алгоритми (greedy)** - Greedy подходът включва изграждане на решение чрез избиране на последователни стъпки, всяка от които произвежда частично решение на задачата, до получаване на цялостното решение. В същото време на всяка стъпка изборът трябва да бъде:

* допустим, т.е. да отговаря на ограниченията на задачата;
* локално оптимален, т.е. да е най-добрият локален избор между всички възможни варианти, налични на всяка стъпка;
* окончателен, т.е. веднъж направен, не може да променя следващите стъпки на алгоритъма.

**Оптимизационни решения**

В компютърните науки задачите, свързани с оптимизацията са такива, в които е необходимо да се намери най-доброто решение от всички възможни решения.

* Greedy алгоритмите се използват за решаване на оптимизационни задачи;
* Обикновено са по-ефективни от другите алгоритми, но може да доведат и до не толкова оптимален резултат;
* Алчните алгоритми избират най-доброто локално решение;
* Алчните алгоритми предполагат, че винаги изборът на локално оптимално решение води до глобално такова, но понякога не е така.

1. Описва понятието рекурсия. Дава пример за рекурсивни алгоритми с пълно изчерпване.
   1. Рекурсия е термина което се ползва за клас от методи и функции, които директно (пряко) или индиректно (косвено) извикват себе си.

Това е техника за решаване на задачи, където се разделя задачата на подзадачи от същия тип, включва функция, извикваща себе си, функцията трябва да има граничен случай и всяка стъпка от рекурсията трябва да се движи към граничния случай

**Видове рекурсия**

*Пряка рекурсия*

Метода А директно се обръща към себе си.

*Непряка рекурсия*

А се обръща към В, В се обръща към А.

Възможно е и А->В->С->А

**Рекурсията е:**

* Метод, който се обръща към себе си;
* Много мощна техника за прилагане на комбинаторни алгоритми;
* Всяка рекурсия трябва да има граничен случай;
* Рекурсията може да бъде вредна, ако не се използва правилно.
  1. Пример за такива методи/функции е намиране на Факториела на число, тъй като всяко число на Факториел, се състои от продукта на всички предишни числа на факториел: 5! = 5 \* 4!, 4! = 4\* 3!, и т.н.

static int Factorial(int input)  
{  
 return (input < 2) ? 1 : (input \* Factorial(input-1));  
}

**Какво е backtracking? -** Клас алгоритми за намиране на всички решения на някаква комбинаторна задача. В това число намиране на всички пътища от София до Варна

**Как работи backtracking?**

* На всяка стъпка рекурсивно се преглеждат всички перспективни възможности
  + Всички не перспективни възможности се отхвърлят възможно най-рано
  + Времето на работа е експоненциално

**Backtracking поставя въпроси като:**

* Колко начина за…съществуват?
* Има ли начин за…
* Кои са всички възможни решения?
* Кое е оптималното решение?

**Backtracking – алгоритъм**

*1 стъпка:* Конструира се едно частично решение.

*2 стъпка:* Проверява се дали частичното решение е общо (търсеното). Ако е така, решението се запомня или извежда и процесът на търсене или завършва, или продължава по същата схема, докато бъдат генерирани всички възможни решения.

*3 стъпка:* В противен случай се прави опит текущото частично решение да се продължи (според условието на задачата).

*4 стъпка:* Ако на някоя стъпка се окаже невъзможно ново разширяване, извършва се връщане назад към предишното частично решение и се прави нов опит то да се разшири (продължи) по друг, различен от предишния, начин.

**Backtracking намира:**

* Всички решения/оптимално решение на комбинаторна задача чрез генериране на всички възможности
* Отстранява не перспективните възможности

1. Разработва програми за решаването на задачи с помощта на алчни алгоритми.
   1. Пример за това е задачата за монети, задачата изисква видове монети с техните стойности, и желана сума, решението, при всяка стъпка, взима най-голямата възможна монета, което може да си позволи, и го слага към текущата сумата, разликата между желаната сума и текущата е останалата сума.  
      Този алгоритъм приключва когато двете суми са равни.
2. Дефинира понятията от комбинаториката: вариации, комбинации и пермутации. Различава вариации, комбинации и пермутации.
   1. **Вариация** - Брой уникални избора на K елемента от група с N елемента, където К е по-малко от N. Реда на елементите **НЕ** е от значение.  
      Примерно: Кои 10 от 100 човека ще бъдат избрани да участват в шоу.

**Вариации без повторения - Определение:** Всички различни наредени извадки без повторение на n елемента от k-ти клас наричаме вариации без повторение на n елемента от k -ти клас. Две вариации без повторение се различават една от друга или по реда на участващите в тях елементи или по елементите, участващи в тях.

**Теорема:** Броят на всички различни вариации без повторение на n елемента от k-ти клас е:



* 1. **Комбинация** – Брой уникални подреждания на K елемента от група с N елемента, където К е по-малко от N. Реда на елементите е от значение.  
     Пример: Кои 3 три участника ще спечелят на в състезание (1во, 2ро и 3то място) от 10 участника.

**Комбинации без повторения - Определение:** Всички различни не наредени извадки без повторение на n елемента от k-ти клас се наричат комбинации без повторение на n елемента от k-ти клас. Две комбинации без повторение се различават една от друга по елементите, участващи в тях.

**Теорема:** Броят на всички различни комбинации на n елемента от k-ти клас е:



Някои по-често използвани свойства на биномните коефициенти са:



* 1. Пермутация – Брой уникални подреждания на N елемента, без повторения.  
     По колко различни начина може да се подредят всички 10 книги.

**Множества -** Съвкупност от обекти, обединени по някакъв общ признак.

Обектите, от които се състои множеството, се наричат **елементи.**

Символният запис **a∊A** означава, че*елементът a принадлежи на множеството A***.**

Множеството А се нарича **крайно**, ако се състои от краен брой елементи.

Означаваме A={a1, a2, …, an} и пишем |A|=n.

Множеството Ø, което не съдържа нито един елемент, се нарича **празно множество**.

* Пермутации - начини за подреждане на N елемента
* Вариации - начини за подреждане на K от N елемента
* Комбинации - начини за избор на K от N елемента
* Триъгълник на Паскал
  + Биномни коефициенти

1. Обяснява техниката на динамичното оптимиране.
   1. В програмирането, Динамично Оптимиране, също се нарича Динамично Програмиране, то е начин на решаване на проблеми като рекурсивно се пречупват на по прости задачи, чиито решение водят во холистично решение, най-често като се изключват потенциални подходи които не водят до оптимално решение, ранно в етапите на рекурсията.( Метод за решаване на задачи с припокриващи се подзадачи.

Изграждане на рекурентни връзки, свързващи решението на задачата с решенията на по-малки подзадачи от същия тип.

Решаване на всяка една от по-малките подзадачи само веднъж, записване на резултата в таблица, от която след това се получава решение на първоначалната задача.)

**Методът “разделяй и владей” -** Един от най-важните и най-широко приложим метод за проектиране на ефективни алгоритми

* Метод на декомпозицията
* Разделя дадена задача с размер N на по-малки задачи,
* На основата на решенията на по-малките задачи се получава решението на първоначалната задача.

**Защо динамичното решение е по-добро? –**

* Рекурсивното решение за големи стойности на N е твърде нерационално
* Налага се твърде много пъти да се пресмятат някои от стойностите на редицата .
* Динамичното решение използва вече пресметнатите стойности за по-малки стойности на търсения аргумент, без да се пресмята наново.
* Използват се рекурсивни формули, но стойностите се вземат от някаква структура от данни, в която са били попълнени в момента на пресмятането си.
* Сложността на такива задачи е линейна, с изключение на задачите, които изискват поддържането на матрица за запазване на стойности, за които сложността е квадратна.

**Динамичното оптимиране**

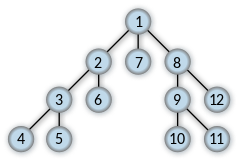
* + решава подзадачи, които се припокриват
  + избира оптималните решения на подзадачите
  + комбинира оптималните решения на подзадачите и получава оптимално общо решение

1. Разработва програми за решението на едномерни и двумерни задачи с помощта на техники от динамичното оптимиране (задача за раницата, най-малка подредица, оптимален път в лабиринт и др.).
2. Дефинира основни понятия от теория на графите: граф, ориентиран и неориентиран граф, връх, ребро, път в граф, цикъл в граф.
   1. Граф е структура, състоящ се от точки и връзки помежду им. То най-често се реализира като списък на съседство или матрица на връзки.
   2. Ориентиран граф е граф където връзките между точките на графа са принципно еднопосочни.
   3. Неориентиран граф е граф където връзките м-ду точките на графа са **ВИНАГИ** двупосочни.
   4. Възел, Node е точка/дестинация в графа.
   5. Ребро, Edge, e връзка между две точки в графа.
   6. Тежест, Weight, е цената на изминаване през дадено ребро.
   7. Цикъл, Loop, е ребро който започва и завършва на един и същ възел.
   8. Път в граф е описание на точките и връзките през които трябва да се премине от една точка до друга. **Дървовидните структури от данни са:**
   * Разклонени йерархични структури от данни
   * Изградени от възли
   * Всеки възел е свързан с други възли (разклонения на дървото)

***Дървета*** - двоични, балансирани, подредени и др.

***Графи*** - ориентирани, неориентирани, с тегла и др.

***Мрежи*** - графи с особени свойства

1. Обяснява и представя графично алгоритми за обхождане в ширина и дълбочина на графи (BFS и DFS).
   1. DFS, Depth-First Search, е алгоритъм за търсене в дълбочина за дървовидни структури от данни. При този метод на търсене, рекурсивно се гледат децата на всеки възел, до като има такива, след което се гледат съседите, преди да се връща назад към родителите.  
      
   2. BFS, Breadth-First Search, е алгоритъм за търсене в широчина за дървовидни структури от данни. При този метод на търсене, се започва като добавим началния възел в опашка, и започваме, циклично да обработваме елемента което на върха на опашката. Добавят се всички деца на елемента на върха, обработва се самия елемент, и се преминава на следващия, този алгоритъм продължава до като има елементи в опашката.  
      
2. Анализира фрагмент/и от код и идентифицира и поправя правилно грешките в написания програмен код, така че да реши поставената задача. Допълва кода, ако и когато това е необходимо.