Тема7: Алгоритми и структури от данни

**Greedy algorithm** – „алчните алгоритми“ представляват стъпкови алгоритми, които довеждат до решение на поставения проблем за разумно време. Не винаги решението, постигнато с „алчен алгоритъм“, е най-доброто. На всяка стъпка алгоритъмът избира локалното оптимално решение, но не е задължително множеството локални оптимални решения да доведат до глобалното оптимално решение.

**Рекурсия** и рекурсивни алгоритми – рекурсията представлява метод за решаване на проблеми чрез структура, която се извиква множество пъти, докато не приключи изпълнението на програмата. Рекурсията дава възможност да се опише цикъл без да се използва оператор за цикъл. Рекурсията се дели на два вида – **директна и индиректна**. При директната рекурсия даденият метод извиква сам себе си, а при индиректната рекурсия няколко метода се извикват последователно. За да се избегне влизането в безкраен цикъл се дефинира **дъно на рекурсията**, при което извикването на следващия метод спира. Действията, изпълнявани в рекурсия, могат да се зададат в **прав или обратен ход**. При правия ход на рекурсията действието се извършва преди следващото извикване на функцията толкова пъти, колкото извиквания настъпят и резултатът се връща в същата последователност. При обратния ход действието се извършва след рекурсивното извикване/достигане на дъното/ и резултатът се връща в обратен ред.

**Backtracking** - На всяка стъпка рекурсивно се преглеждат всички перспективни възможности. Конструира се едно **частично решение**. Проверява се дали частичното решение е търсеното. Ако на някоя стъпка се окаже невъзможно продължаването на решението, извършва се връщане назад към предишното частично решение и се прави нов опит то да се продължи по друг, различен от предишния, начин.

**Комбинаторни алгоритми** – използват се, за да се решават комбинаторни задачи – комбинации, вариации, пермутации. Имплементират се с помощта на рекурсия.

**Динамично оптимиране** – метод за **опростяване на сложни проблеми** чрез разделянето им на по-малки. При решаването на дадена част от проблема резултатът се запазва в подходяща структура от данни. Ако отново се достигне до същия проблем се изважда **готовият резултат**. Така се **пести време**, но за сметка на изразходването на **повече памет**. Техниката за запомняне на решенията на подзадачите, вместо повторното им решаване, се нарича „**мемоизация**“.

Тема8: Алгоритми и структури от данни

**Дървовидни структури от данни –дървета и графи** – Дървовидните структури от данни са разклонени йерархични структури от данни, изградени от възли. Всеки възел е свързан с други възли (разклонения на дървото) чрез дъга.

* Всяко дърво се образува от възли и дъги, които ги свързват
* Върховете могат да бъдат два вида:
* Родител
* Наследник
* Върхът без родител се нарича “корен”
* Всяко дърво има само един корен
* Връх без наследници се нарича “листо”

**Двоични дървета**

* Възлите на двоичните дървета имат по не повече от две разклонения

**Двоични дървета за търсене** (частен случай на сортирани дървета):

* Лявото разклонение на всеки възел има по-малка стойност от стойността на възела
* Дясното разклонение на всеки възел има по-голяма стойност от стойността на възела.

**Обхождане на дървовидни структури**

Обхождане на дърво представлява посещаването на всеки негов възел точно по веднъж

Последователността на обхождането може да варира, в зависимост от алгоритъма за обхождане:

* Обхождане в дълбочина (DFS):
  + Първо се посещават наследниците на възела
  + Стандартна реализация - чрез рекурсия
* Обхождане в ширина (BFS):
  + Първо се посещава най-близкия възел
  + Стандартна реализация - чрез опашка

**Обхождане в дълбочина (DFS)** - за всеки възел:

* Посещават се всички негови деца
* Ако възела няма деца или всички негови деца са вече обходени се обработва стойността му

**Обхождане в ширина (BFS)** - за всеки възел:

* Обработва се стойността на възела
* Посещават се всички съседни възли

**Двоичните дървета за търсене** са подредени

За всеки възел x:

* Елементите в лявото поддърво на x са по-малки от x
* Елементите в дясното поддърво на x са по-големи от x

**B-trees**

**Червено-черно дърво**

**Хеширане и хеш таблици** – хеширането представлява конвертирането на сложен тип данни в целочислен тип данни. Хеш таблицата е структура, която пази двойка данни ключ-стойност (dictionnary).

**Хеширането** може да се представи в масив. Чрез **хеш функция** сложният тип данни се свежда до цяло число и се поставя на съответния индекс в масива. Перфектно хеширащата функция свързва всеки ключ към уникално цяло число в рамките на конкретен интервал. В повечето случаи перфектното хеширане е невъзможно – възникват **колизии**. Колизия настъпва, когато хеш функцията генерира един и същ хеш за различни ключове.

Абстрактния тип данни “**речник**” асоциира стойности с уникални ключове. Тази структура е позната като MAP или **асоциативен масив**. Съдържа набор от наредени двойки от тип **{key, value}**.

**Графи** – върхове, свързани помежду си с ребра.

ориентиран/неориентиран граф – определя се от посоката на ребрата

претеглен/непретеглен граф – определя се от тежестта на ребрата

цикличен/нецикличен граф – определя се от начина на свързване на ребрата

**Алгоритъм на Дейкстра** – служи за намиране на най-късия път между два върха на граф. Може да се имплементира с **priority\_queue**. Във всеки момент от работата на алгоритъма за всеки връх се пази информация за най-късия път, намерен до момента. Ако след това алгоритъмът намери по-добър път, тази информация се актуализира.

**Алгоритъм на Прим** – намиране на път между всички върхове с минимално тегло. Прилага се в претеглени неориентирани графи.

**Алгоритъм на Крускал** – намира пътят с минимално тегло между два върха в граф. Прилага се в претеглени графи.