APS1

# Algoritem

## Kaj je algoritem?

Algoritem je

* **Jasen** (= enostavno razumljiv, skladen z okoljem oz. znanjem)
* **Nedvoumen** (= ne moremo, ga zamenjati s nečim podobnim)
* **Mehaničen** (= uporablja le elementarne operacije)
* **Postopek** (= izvajanje korak za korakom)

postopek za reševanje računskega problema.

## Kaj je to računski problem?

Computation vs. calculation (računanje z računalnikom npr. iskanje najkrajše poti, urejanje podatkov… vs. računanje v smislu aritmetike)

Pri APS nas zanima v večini computation.

Računski problem rešimo tako da

* Za **vse** možne **naloge** problema
* poiščemo ustrezno **rešitev.**

V splošnem računski problem natačno opisuje odnos med **nalogo** in njenimi **rešitvami**.

Če povežemo algoritme z računskimi problemi, algoritem rešuje računski problem, kadar zna

* za **vsako nalogo**
* izračunati **pravilno rešitev**

Graphical user interface, text, application, chat or text message

Description automatically generated

### Vrste računskih problemov

* **Iskalni** (= rešitev problema je objekt, ki zadošča danim omejitvam, npr. za dani števili izračunaj vsoto, uredi seznam elementov …)
* **Odločitveni** (= rešitev je oblike da/ne, npr. ali velja enakost x + y = z? Ali je zaporedje števil urejeno? …)
* **Preštevalni** (= podobno kot iskalni, le da nas zanima število objektov, npr. na koliko načinov lahko seštejemo dve naravni števili da dobimo 42 …)
* **Naštevalni** (= iskalni in preštevalni skupaj, npr. katera števila so manjša od 5? …)
* **Optimizacijski** (= iskanje najboljše rešitve izmed vseh možnih; naprej delimo še na kriterijski, kjer iščemo vrednost rešitve in konstrukcijski, kjer želimo rešitev konstruirati ali preoblikovati)

### Formalna definicija računskega problema

Računski problem je množica vseh parov naloga : rešitev.

### Church-Turingova teza

S Turingovim strojem se da izračunati vse, kar se sploh izračunati da.

# Abstraktni podatkovni tipi

Algoritmi + podatkovne strukture = program -> BISTVO

## Podatkovni tip

Je model za podatke in pove način uporabe nekega podatka.

Podatkovni tip poda opise za

* **Množico vrednosti**, ki jo lahko podatek zavzame
* **Množico operacij**, s katero lahko podatek operira
* **Predstavitev podatka**

S podatkovnim tipom omogočimo **razvijalski pogled na podatke**.

## Abstraktni podatkovni tip ali ADT/APT

Vsebuje množico vrednosti in operacij, ne vsebuje pa implementacije oz. ga ne zanima kako so podatki predstavljeni (abstract = povzetek). Analogija na OOP => class implements interface (razred implementira množico vrednosti in operacij, doda pa še predstavitev podatkov).

## Podatkovne tipe lahko razdelimo

Razdelimo jih na

* **Enostavne** podatkovne tipe (primitivni, atomični, npr. cela števila, števila s plavajočo vejico …)
* **Sestavljene** podatkovne tipe (sestavljeni iz drugih tipov, ki ali zavzemajo njihovo predstavitev (struct), ali pa njihovo obnašanje (class), npr. kompleksna števila, nizi, zaporedje …)

## Podatkovna struktura

Je sestavljen podatkovni tip in omogoča učinkovito uporabo podatkov (iskanje, dostop, spreminjanje podatkov, tudi predstavitev in organizacija podatkov)

### Osnovne podatkovne strukture:

* Množica (vsebuje enolične elemente in je brez vrstnega reda)
* Vreča (= množica, le da dovoljuje ponavljanje elementov)
* Sklad
* Vrsta
* Vrsta z dvema koncema
* Vrsta s prednostjo
* Zaporedje
* Slovar

## Polje

Polje je podatkovna struktura, ki vsebuje zaporedno shranjene elemente, do katerih lahko dostopamo z indeksi.

### Zaporedno hranjenje elementov

Elementi si v pomnilniku sledijo zaporedno, dostop pa je zaradi tega zelo hiter (naključni dostop). Vstavljanje in brisanje je lahko hitro (ali je na koncu ali na začetku polja) ali pa počasno (nekje v sredini polja). Predpomnilnik je lahko dobro izkoriščen v primeru da v njem hranimo vrednosti, če hranimo naslove pa je slabo izkoriščen (manjša lokalnost).

Loitering ali postopanje se zgodi pri odstranjevanju podatka iz pomnilnika (npr. iz polja). Programski jeziki, ki uporabljajo garbage collector se pri tem še posebej zatikajo, saj če obstaja referenca na nek podatek, ki smo ga že odstranili, bo GC smatral, da podatek še vedno obstaja. Moramo ga dereferencirati (to naredimo, tako da naslov, na katerem se je podatek nahajal, nastavimo na **null**).

# Dinamično polje

Dinamično polje je podatkovna struktura, ki ima sposobno se prilagajati na količino podatkov v polju. Na primer pri dodajanju novega elementa v polje, ki zasedenost 100% se pokliče funkcija (tipično resize), ki ustvari novo polje neke kapacitete, tipično 2 \* or. kapaciteta, elementi iz originalnega polja pa se samo kopirajo v novo ustvarjeno polje. Poznamo tudi druge strategije spreminjanja kapacitete, ampak to je glavna, ki smo jo pri predmetu obravnavali.

# Povezani seznami

## Kazalčne podatkovne strukture

Kaj je kazalec oz. Pointer? **Pointer** (ki se sicer v Javi ne uporablja, je pa zelo pomemben koncept v programiranju) je objekt, ki kaže na naslov nekega podatka v pomnilniku. Uporablja se v bolj nižjenivojskih programskih jezikih kot so C, C++, Rust itd., Java pa uporablja precej podoben koncept, **reference** ki je avtomatiziran in malo bolj neprilagodljiv.

Kazalčna podatkovna struktura je sestavljena podatkovna struktura (sestavljena iz vozlišč in kazalcev). Je dinamična podatkovna struktura, saj je spreminjanje njene velikosti preprosto.

#### Vozlišče (node) :

* **Element** oz. **podatek** + en ali več **kazalcev**
* Vozlišča med seboj **povezana** s **kazalci**
* Posebni kazalci:
  + **Null** … ne kaže nikamor
  + **First** … kaže na prvo vozlišče
  + **Last** … kaže na zadnje vozlišče

Text

Description automatically generated

#### EPS (enojno povezani seznam)

Vozlišče v EPS vsebuje **element** in **kazalec (povezavo)** na naslednika.