CT10A0013 Ohjelmointi Pythonilla

L11: Ohjelmien tehokkuus

Uolevi Nikula

Päivän ohjelma

- Teoria
  - Algoritmi ja pseudokoodi
  - Rekursio
  - Suorituskyky
  - Tulkeista ja kääntäjistä
- Lopuksi

Teoria / Ohjelmien tehokkuus

Ohjelmien rakenne ja monimutkaisuus
Algoritmi ja pseudokoodi
Rekursiivinen algoritmi
Perusohjelman suorituskykyongelmat

Ohjelmien monimutkaisuus

- "Jokaiseen vaikeaan ongelmaan on olemassa helppo ratkaisu – joka on väärä"
- Tietokoneohjelmat ratkaisevat usein vaikeita ongelmia, joten ohjelmat ovat usein monimutkaisia – ja joskus myös vaikeita ymmärtää
- Tietokoneohjelman toteutus voi olla myös tarpeettoman vaikea ymmärtää, jolloin se on käytännössä huono
- Tietokoneohjelmien luonnollinen monimutkaisuus tulee ratkaistavasta ongelmasta ja sen ominaisuuksista
- Ohjelman määrittelyn, suunnittelun ja toteutuksen tarkoitus on varmistaa, että ohjelma täyttää kaikki asetetut vaatimukset
  - Näiden vaiheiden tulee myös tukea mahdollisimman yksinkertaisen, ymmärrettävän ja tehokkaan toteutuksen tekemistä
  - Oppaan L07 kokoava esimerkki -ohjelman rakenne: yksinkertainen ja ymmärrettävä?

Ohjelman rakenne ja tehokkuus

- Ohjelman rakenne vaikuttaa sen tehokkuuteen hyvin paljon
- Käytännössä ohjelman rakenne vaikuttaa tehokkuuteen usein enemmän kuin käytetty laitteisto
  - Huono ohjelman rakenne tekee tehokkaastakin tietokoneesta hitaan
  - Hyvä rakenne saa hitaankin koneen toimimaan suhteellisen hyvin
  - Useat sisäkkäiset silmukat ennakoivat usein suorituskykyongelmia
- Ohjelman rakenne tarkoittaa mm. sen käskyjen suoritusjärjestystä eli algoritmia, mitä ohjelma seuraa

Algoritmi

- **Algoritmi** on tarkasti määritelty joukko yksikäsitteisiä toimenpiteitä, jotka suorittamalla voidaan ratkaista tietty ongelma
- Algoritmit kirjoitetaan usein ensin yleisellä tasolla ja tarkennetaan tarpeen mukaan
- Esimerkki yksinkertaisesta tiedostonkäsittelyalgoritmista
  1. Avaa tiedosto
  2. Lue tiedosto
  3. Sulje tiedosto
  - Lue tiedosto –vaihe pitäisi tarkentaa seuraavaksi; avaus ja sulku on jo selviä asioita
  - Muita esimerkkejä algoritmeista: haku-, pakkaus-, lajittelu- ja salausalgoritmit
- Katso tarkemmin esim. http://fi.wikipedia.org/wiki/Algoritmi

Pseudokoodi

- Pseudokoodi on **ohjelmointikielen tapaista koodia**
  - piilottaa eri ohjelmointikielten väliset syntaksierot
  - jättää jäljelle algoritmin perusrakenteen
- Mahdollistaa keskustelun algoritmeista, vaikka ohjelmointikieli ei olisi tuttu
- Helpottaa
  - algoritmin tila- ja suoritusaikavaatimusten analysointia
  - algoritmin oikeellisuuden varmistamista
- Yleensä Algoliin perustuvaa Pascal- tai Python -tyylistä syntaksia. Erityisesti sijoitusta kuvaa usein merkki := tai ←, jotta se ei sekoittuisi yhtäsuuruuden kanssa
- (Wikipedia)

Rekursio

- ”Rekursiivinen algoritmi on algoritmi, jonka toiminta perustuu rekursion käyttöön.”
  - Lähtökohtaisesti huono määritelmä, koska toistaa itseään
  - Rekursion kohdalla sopiva määritelmä, koska toistaa itseään
- ”[rekursio on] kielen ominaisuus, että *sama rakenne* voidaan *toistaa periaatteessa rajattoman monta kertaa*.” (Tieteen termipankki)
- Matematiikassa rekursio mahdollistaa funktion määrittelyn niin, että ”funktion arvo tietyssä pisteessä riippuu funktion arvosta edellisessä pisteessä.”
- Ohjelmoinnissa rekursio tarkoittaa sitä, että funktio kutsuu itse itseään
  - Erittäin tehokas tapa toteuttaa tiettyjä rekursiivisia algoritmeja
  - Asiaa käsitellään laajemmin Tietorakenteet ja algoritmit –kurssilla
- Kaksi variaatiota
  - Suora rekursio: funktio kutsuu itseään
  - Epäsuora rekursio: funktio A kutsuu funktiota B ja funktio B kutsuu funktiota A

Rekursion toteutuksesta

- Funktioiden eli aliohjelmien toiminta yleisesti ottaen
  - Jokainen funktiokutsu luo uuden kopion funktiosta ja lisää sen pinoon
  - Muuttujien arvot pysyvät tallessa eri kopioissa
- Jossain vaiheessa rekursion on loputtava ja funktion on palautettava arvo. Tämän jälkeen palataan takaisin kutsuvaan funktioon eli otetaan se takaisin pinosta ja tehdään se loppuun jne.
  - Vertaa toistorakenteeseen eli lopetusehdon tarkistus jne.
  - Rekursio varaa muistia jokaisen aliohjelmakutsun yhteydessä, joten sitä ei tule käyttää toistorakenteen tilalla turhaan!
- Rekursio ei ole tämän kurssin ydinasioita, mutta käsitellään lyhyesti koska
  - Demonstroi hyvin algoritmia ja sen toimintaa
  - Asian käsittelyn jälkeen tällä kurssilla ei tule käyttää rekursiota (väärin)
- Esimerkki rekursiivisesta ohjelmasta näkyy Koodiesimerkit-osassa

Lajittelualgoritmien tehokkuus

- Lajittelualgoritmit on tyypillinen esimerkki algoritmeistä
  - Monta hieman erilaista algoritmia
  - Algoritmien ominaisuudet hyvin erilaisia, erityisesti nopeus ja tarvittavan muistin määrä
- Video lajittelualgoritmeistä
  - 15 algoritmia 6 minuutissa
  - https://www.youtube.com/watch?v=kPRA0W1kECg

Perusohjelman suorituskykyongelmat 1

- Myös perusohjelmissa voi olla suorituskykyongelmia ja tällöin kannattaa tarkastaa seuraavat asiat
  - Varaako ohjelma ylimääräisiä resursseja kuten tiedostoja, taulukoita, yms.
    - Nämä eivät yleensä ole ongelmia Python-kielessä
  - Vapauttaako ohjelma resurssit aina käytön jälkeen
  - Tiedostonkäsittely on hidasta ajonaikaiseen muistiin verrattuna
    - Liittyykö tiedostonkäsittelyyn turhia toimenpiteitä, esim. tiedoston avaaminen ja sulkeminen silmukassa joka kierroksella
  - Onko ohjelman toistorakenteissa ongelmia
    - Useat sisäkkäiset toistorakenteet hidastavat ohjelmaa merkittävästi suurilla tietomäärillä
- Ohjelmien suorituskyky ei ole ohjelmoinnin peruskurssin ydinasioita, mutta jatkon ja käytännön kannalta asian perusteet on hyvä ymmärtää
  - Esim. harjoitustyössä joillain on aina joskus suorituskykyongelmia, mutta kaikilla ei
- Suorituskykyä ja algoritmeja käsitellään tarkemmin kurssilla *Tietorakenteet ja algoritmit*

Perusohjelman suorituskykyongelmat 2

- CodeGrade virheitä, jotka viittaavat suorituskykyongelmaan ts. ohjelma on hidas
  - timeout: Ohjelman suoritus vie niin pitkään, että CG olettaa ohjelman olevan ikisilmukassa ja lopettaa sen (timeout)
- Lähtökohta ongelmien välttämiseen on yksinkertainen, selkeä ja suoraviivainen ratkaisu
  - Jokaiselle loogiselle tehtävällä kannattaa olla oma aliohjelma: lue, analysoi, tallenna, jne.
  - Silmukoissa kannattaa pyrkiä aina datajoukon läpikäymiseen yhden kerran alusta loppuun, ts. käsittelyaika riippuu lineaarisesti käsiteltävästä datasta. Tällöin datamäärän tullessa 10-kertaiseksi myös suoritusaika on luokkaa 10-kertainen
  - Kahta (tai useampaa) silmukkaa ei kannata laittaa sisäkkäin, sillä tällöin datamäärän 10-kertaistuessa lajitteluaika on luokkaa 10x10 eli 100-kertainen

Ohjelmien tehokkuus ja suorituskyky

- Tietokoneohjelmien yhteydessä suorituskyky on yleinen keskustelunaihe/ongelma
  - Tyypillisesti suorituskyky ei ole koskaan liian hyvä ja parhaimmillaan hyväksyttävä/riittävä
- Nopean ja tehokkaan **fyysisen laitteen** lisäksi ohjelmiston toteutustapa ja käytetyt **tietorakenteet sekä algoritmit** vaikuttavat merkittävästi suorituskykyyn
  - Tyypillisesti suunnittelija valitsee algoritmit ja ohjelmoija toteuttaa ne
- Käytetyissä **ohjelmointikielissä** on eroja, mutta tyypillisesti erot ohjelmissa ja ohjelmoijissa ovat merkittävämpiä tekijöitä

Kääntäminen ja tulkkaaminen

Luonnollisen kielen kääntäminen ja tulkkaaminen
Tietokoneohjelmien kääntäminen ja tulkkaaminen
Ohjelmointikielten vertailu ja kääntäjiä

Luonnollisen kielen kääntäminen ja tulkkaaminen

- Luonnollisen kielen (simultaani)**tulkkaus**
  - Tulkki kääntää noin yhden lauseen kerrallaan kieleltä toiselle puhujan pitäessä taukoa
  - Tulkattu lause vastaa mahdollisimman hyvin alkuperäistä lausetta – paikallinen eli lausekohtainen optimointi mahdollista
- Luonnollisen kielen **kääntäminen**
  - Kääntäjä lukee esim. kirjan käsikirjoituksen läpi, kenties useita kertoja
  - Kääntäjä voi tarkistaa asioita ristiin käsikirjoituksessa – globaali optimointi ja tarkastelu mahdollista

Tietokoneohjelman tulkkaaminen ja kääntäminen

- Tulkkaaminen
  - Suoritettava lause luetaan, tarkistetaan syntaksi, tulkitaan merkitys ja suoritetaan lause
- Kääntäminen
  - Koko koodi luetaan, tarkistetaan, käännetään ja mahdollisesti optimoidaan konekieleksi
    - Koko ohjelman (globaali) optimointi mahdollista
  - Tyypillisesti koko ohjelma käydään läpi 2-5 kertaa
  - Lopputuloksena on suoritettavassa kohdeympäristössä toimiva konekielinen ohjelma
  - Lopputulos suoritetaan erillisestä käskystä

Tulkkaamisen etuja ja riskejä

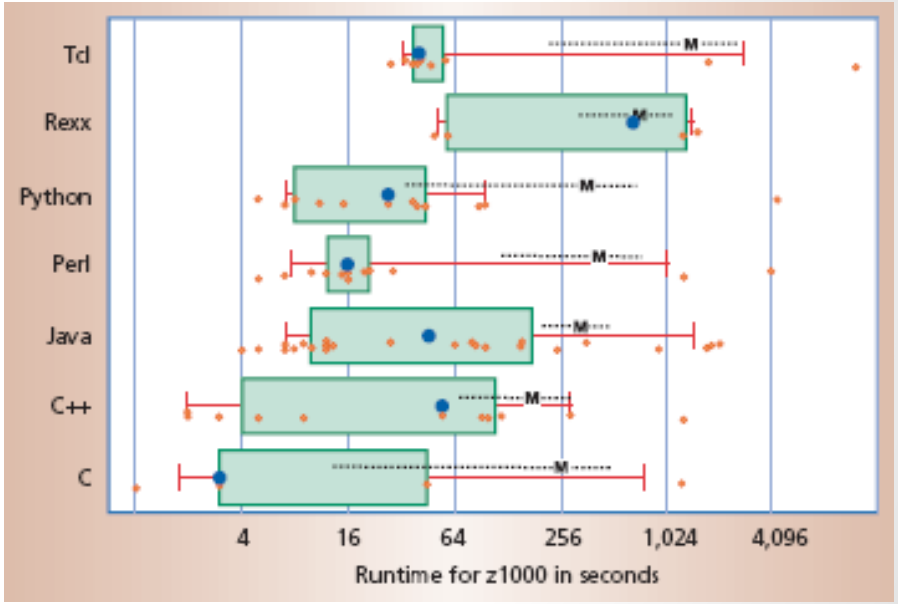
- Etuja
  - Koodin suorittaminen alkaa nopeammin – tarvitaan vain yhden suoritettavan rivin luku ja tulkinta kerrallaan
- Riskejä
  - Tulkki voi olla iso ja hidas käynnistää
  - Koodi voi olla hidasta ja kuluttaa paljon muistia optimoinnin paikallisuuden takia
  - Koodissa voi olla rivejä, joita ei ole suoritettu – ja näillä voi olla virheitä

Kääntämisen etuja ja riskejä

- Etuja
  - Koodi voidaan optimoida vähän tilaa vieväksi ja nopeaksi
  - Koko koodi voidaan tarkastaa syntaksin osalta
  - Käännetyn koodin suorittaminen nopeampaa, koska tarkastukset on tehty käännösvaiheessa
    - Vrt. Python ja kirjastojen pyc-tiedostot
- Riskejä eli huonoja puolia
  - Kääntäminen vie oman aikansa (1 min – 30 min – yö – …, vrt. eräajot)

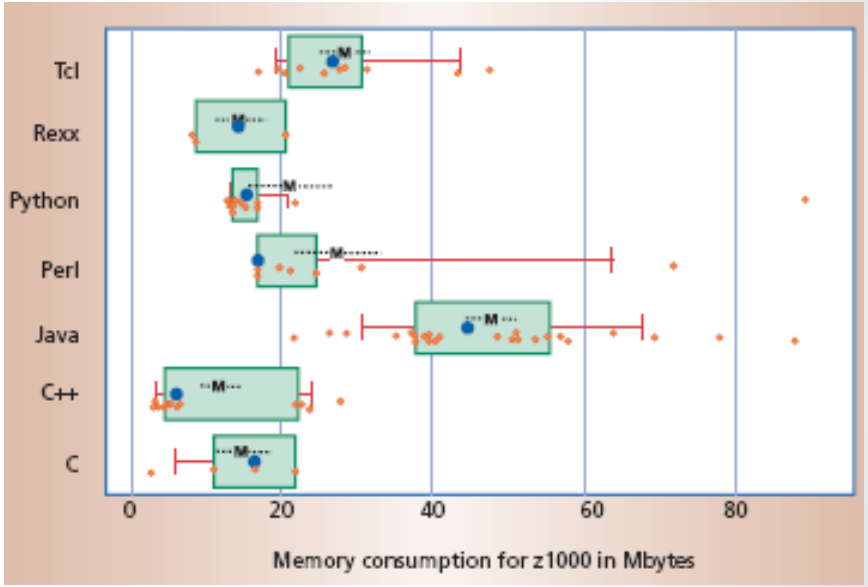
Ohjelmointikielivertailu: nopeus

- Python ei ole merkittävästi hitaampi kieli kuin muut
- Prechelt, L. (2000). "An empirical comparison of seven programming languages." Computer 33(10): 23-29.



Ohjelmointikielivertailu: muistintarve

- Python ei tarvitse suorituksen aikana merkittävästi enempää muistia kuin muut ohjelmointikielet



Ilmaisia kääntäjiä Internetissä

- **PyPy3.9,** http://pypy.org/
  - “PyPy is a replacement for CPython. It is built using the RPython language that was co-developed with it. The main reason to use it instead of CPython is speed: it runs generally faster.”
  - "PyPy is an alternative implementation of the Python programming language to CPython (which is the standard implementation). PyPy often runs faster than CPython because PyPy is a just-in-time compiler while CPython is an interpreter. Most Python code runs well on PyPy except for code that depends on CPython extensions, which either doesn't work or incurs some overhead when run in PyPy." Wikipedia
- **Psyco**, http://psyco.sourceforge.net/
  - "High-level languages need not be slower than low-level ones.”
  - ”Psyco is a Python extension module which can greatly speed up the execution of any Python code.”
  - ”12 March 2012: Psyco is unmaintained and dead. Please look at PyPy for the state-of-the-art in JIT compilers for Python.”
- **C/C++** kääntäjä: Dev-C++, http://www.bloodshed.net/devcpp.html
  - Ilmainen C/C++ kääntäjä ja kehitysympäristö, ei ole tämän kurssin asioita
- **C/C++ kehitysympäristö LUTin C-kurssilla**: Windows/WSL/Ubuntu-Linux; gcc ja VSC

Yhteenveto kääntäjä-tulkki aiheesta

- Molemmilla on etuja ja rajoitteita
- Tyypillisesti kaupalliset ohjelmistot ovat käännettyjä
- Tulkattavat kielet ovat suosittuja loppukäyttäjäohjelmoinnissa eli "helpoissa" ohjelmointikielissä, kuten Visual Basic ja Python
- Pythonille ei ole tällä hetkellä saatavilla kääntäjää konekielisten ohjelmien tekoon
  - PyPy saattaa olla sellainen, mutta sopivuus pitää tarkistaa tapauskohtaisesti

Koodiesimerkkejä

Rekursio

Rekursiivinen ohjelma

```
# Rekursiivinen ohjelma eli ohjelma kutsuu itseään
def kertoma(x):
    if (x > 0):                     # Lopetusehtoa ei saavutettu vielä
        return (x * kertoma(x-1)) # Kutsuu itseään, parametri muuttuu
    else:             # lopetuksen paluuarvo, tyypillisesti kiinteä arvo
        return 1

def paaohjelma():
    Luku = int(input("Minkä luvun kertoman haluat laskea: "))
    print(kertoma(Luku)) # ensimmäinen rekursiivisen ohjelman kutsu
    return None

paaohjelma()
```

Lopuksi

Osaamistavoitteet

Ohjelmointivideot

Osaamistavoitteet

- Yleiskuva ohjelmien suorituskyvystä, ohjelmistot vs. rauta
- Perustiedot omien ohjelmien suorituskykyyn vaikuttavista tekijöistä, mm. toistorakenteet ja tiedostot
- Algoritmi ja pseudokoodi: mitä ne ovat
- Rekursio: itseään kutsuva ohjelma ja sen turhan käytön välttäminen
- Käännettävien ja tulkattavien ohjelmien periaatteelliset erot

Ohjelmointivideot

- Esimerkki suorituskyvystä erilaisilla aineistoilla, tehtäväksianto alla
- Virhe tulkattavassa ohjelmassa
- Minimaalinen ohjelma Pythonilla ja C-kielellä -vertailua

- Videolla olevan ohjelman tehtäväksianto
  - Tee ohjelma, joka lajittelee tiedoston rivit aakkosjärjestykseen
  - Kokeile ohjelmaa eri kokoisilla tiedostoilla, esim. 10, 100, 1000 ja 10000 riviä
    - Mikäli et löydä sopivia testitiedostoja, tee apuohjelma, joka generoi sellaisia

Täydennyksiä oppaan lukuun 11

Tyyliohjeita pienille Python-ohjelmille
ASPAssa ei enää uusia tarkistuksia
Oppaan esimerkit ja käsitellyt asiat

Pienen Python-ohjelman tyyliohjeet

- Ohjelmat kannattaa muodostaa aliohjelmista, jotka muodostava selkeän ja loogisen kokonaisuuden
  - Näin jo aliohjelmien nimistä näkee, mitä ne sisältävät ja aliohjelmien rakennetta voi miettiä myös suorituskyvyn kannalta
  - Selkeät pienet aliohjelmat ohjaavat myös laajentamaan ohjelmaa tarvittaessa vastaavilla pienillä aliohjelmilla
- Jokaisessa aliohjelmassa kannattaa tavoitella selkeää lineaarista rakennetta, joka etenee alusta loppuun yhdellä kertaa
  - Useita sisäkkäisiä toistorakenteita kannattaa välttää
  - Turhia toistorakenteita kannattaa välttää
  - Rekursiota voi käyttää, jos ymmärtää oikeasti sen toiminnan – tämä käydään tarkemmin läpi *Tietorakenteet ja algoritmit* -kurssilla
- Lähtökohtaisesti kannattaa käyttää vain rakenteita ja funktioita, joita on käytetty kurssilla ja jotka ymmärtää
  - Tämä mahdollistaa selkeiden, yksinkertaisten, ymmärrettävien ja myös tehokkaiden ohjelmien tekemisen

Käsitellyt asiat oppaan luvussa 11

- Pseudokoodiesimerkki
- Ongelmasta ohjelmaksi –esimerkki: Esimerkki 11.1
- Rekursiivinen ohjelma: Esimerkki 11.3
  - Rekursion vaihtoehtona toistorakenne: Esimerkki 11.2