Johdatus tekoälyyn, syksy 2014, Teemu Roos **Harjoitus 2**

Tehtävä 1. Todennäköisyyslaskentaa. (1 p)

Vinkki: Älä menetä hermojasi. Kohdissa a–f on vain laskettava kuinka monta alkeistapahtumaa (korttiparia) toteuttaa ehdon. Löydät esimerkkejä ratkaistuista tehtävistä viime vuoden kurssin sivulta.

Nostetaan tavallisesta hyvin sekoitetusta korttipakasta (ilman jokereita eli $4 \times 13 = 52$ korttia) kaksi korttia. Ensin nostettu kortti pannaan takaisin pakkaan ja sekoitetaan ennen toisen kortin nostamista, joten sama kortti voi tulla kaksi kertaa. Laske seuraavat todennäköisyydet.

a) P("kumpikaan kortti ei ole pataa")

b) P("molemmat kortit ovat pataa")

c) P("tasan yksi korteista on pataa")

d) P("numeroiden summa suurempi kuin 4")

e) P("numeroiden summa suurempi kuin 4" | "ensimmäisen kortin numero on 2")

f) P("ensimmäisen kortin numero pienempi kuin jälkimmäisen")

Mentalisti Saku Tetja on ennustanut oikein mikä arpa poimitaan sata arpaa sisältäneestä hatusta. Saku on joko oikeasti maaginen ja pystyy poimimaan aina oikean arvan tai sitten hänellä kävi tuuri (1 % todennäköisyys osua oikeaan satunnaisesti). Arvioidaan melko sinisilmäisesti, että etukäteen (priori)todennäköisyys, että Saku on maaginen on yksi kymmenestä tuhannesta.

g) Kuinka todennäköistä on, että Saku on maaginen, sen jälkeen, kun on todettu hänen nostaneen oikean arvan hatusta?

Jos tehtävä tuntuu vaikealta, voit etsiä lisämateriaalia Bayesin kaavasta (engl. *Bayes theorem*). Esimerkiksi oheiset linkit saattavat olla

`http://en.wikipedia.org/wiki/Bayes%27_theorem#Drug_testing`
`http://www.youtube.com/watch?v=tRE6mKAIkno`

Tehtävä 2. Minimax ja alpha-beta. (1 piste)

a) Esitä pelipuu alkaen ristinollatilanteesta

```
O|X|
-+-+-
X| |
-+-+-
X|O|O
```

kun seuraavana vuorossa on 'X'. Lopputilojen kustannus on +1, kun 'X' voittaa, -1, kun 'O' voittaa, ja 0 muuten.

b) Laske solmujen min- ja max-arvot luennolla esitetyn minimax-algoritmin avulla. Mitkä siirrot ovat optimaaliset ja mihin lopputulokseen ne johtavat?

c) Sovella alpha-beta-karsintaa edellisen tehtävän puuhun ja esitä α- ja β-arvojen tila kussakin suorituksen vaiheessa. Aloita suoritus juurisolmussa parametreilla $\alpha = -1, \beta = +1$ (huomaa, että tämä poikkeaa kurssin sivulla olevasta videosta, jossa aloitusarvot ovat $\alpha = -\infty, \beta = +\infty$). Karsitaanko joitain alipuita? Miten lapsisolmujen läpikäyntijärjestys vaikuttaa asiaan?

Tehtävä 3. Alpha-beta. (1 p)

Toteuta (ohjelmoimalla) alpha–beta-karsinta seuraavan pseudokoodin mukaisesti:

```
ALPHA-BETA-ARVO(Solmu):
    return(MAX-ARVO(Solmu,-1,+1))

MAX-ARVO(Solmu,alpha,beta):
    if LOPPUTILA(Solmu) return(ARVO(Solmu))
    v=-Inf
    for each Lapsi in LAPSET(Solmu,'X')
        v=MAX(v,MIN-ARVO(Lapsi,alpha,beta))
        if v>=beta return(v)
        alpha=MAX(alpha,v)
    return(v)

MIN-ARVO(Solmu,alpha,beta):
    if LOPPUTILA(Solmu) return(ARVO(Solmu))
    v=+Inf
    for each Lapsi in LAPSET(Solmu,'O')
        v=MIN(v,MAX-ARVO(Lapsi,alpha,beta))
        if v<=alpha return(v)
        beta=MIN(beta,v)
    return(v)
```

missä `LAPSET(Solmu,'X')` ja `LAPSET(Solmu,'O')` palauttavat tilanteessa `Solmu` kunkin pelaajan vuorolla mahdolliset seuraavat pelitilanteet. Testaa algoritmia ratkaisemalla tehtävä 2.

Voit testata algoritmia myös aloittamalla tyhjästä ruudukosta. Voit halutessasi myös kokeilla poistaa algoritmista rivit, joilla suoritus keskeytyy kesken `for each` -silmukan, jos sen hetkinen paras arvo `v` on vähintään β tai korkeintaan α. Tällöin kyseessä on tavallinen minimax-algoritmi. Onko tällä vaikutusta aikavaativuuteen aloitettaessa tyhjästä ruudukosta?

Tehtävä 4. Reittiopas: A*-haku. (2 p)

HUOM: Joillakin opiskelijoilla oli viime viikolla ongelmia edellisen Reittiopas-tehtävän tehtäväpohjan kanssa. 99% ongelmista ratkeaa painamalla Netbeansissa `Clean&Build`-nappulaa. (Punaisten rivien pitäisi hävitä sillä, vaikka `Clean&Build` saattaakin ilmoittaa jokseenkin hämäävästi `''Build failed''`, koska testit eivät mene läpi.) Loput 1% ongelmista ratkeaa asentamalla uudenaikaisen version Netbeansista ja/tai Maven-pluginin Netbeansiin.

Toteuta edellisen viikon tehtävän ratkaisun pohjalta (tai kokonaan alusta) reittihaku A*-hakualgoritmin avulla. Viime viikon leveyssuuntaiseen hakuun lisätään nyt tieto yksittäisten raitiovaunulinjojen aikatauluista. `Pysakki`-oliot tuntevat viereiset pysäkit ja aikataulujen mukaiset lyhimmät siirtymäajat niille. Mielikuvitusmaailmassamme kaikki raitiovaunulinjat lähtevät 10 minuutin välein linjan ensimmäiseltä pysäkiltä ja matkaan lähtö tapahtuu 0-9 minuutin kohdalla haun lähtöpysäkiltä. Reittikysely annetaan siis muodossa:

`Reittiopas.haku(lähtöpysäkki, maalipysäkki, lähtöaika)`

Jos käytät tehtävänannon valmista Java-pohjaa (mikä on erittäin suositeltavaa, sillä siinä käytännössä kaikki pohjatyö tehty valmiiksi), tehtävä on kolmiosainen (lisää ohjeistusta on tehtäväpohjassa):

1. Toteuta `Vertailija`-luokkaan metodi `heuristiikka(Pysakki p)`, joka laskee parametrina annetun pysäkin `p` arvioidun jäljellä olevan ajan maaliin käyttäen apuna tietoja pysäkin ja maalin koordinaateista. Oletetaan tässä, että raitiovaunun nopeus on 260 koordinaattipistettä minuutissa.

2. Toteuta `Vertailija`-luokkaan myös metodi `compare(Tila t1, Tila t2)`, jota käytetään A*-haun prioriteettijonossa. Tilat tulee laittaa järjestykseen A*-haulle tarkoituksenmukaisella tavalla, jossa suositaan tiloja, joille $g(N) + h(N)$ (eli matka-aika tähän mennessä + heuristinen arvio loppuajasta) on mahdollisimman pieni.

3. Toteuta `Reittiopas`-luokkaan A*-haku. Käytä apuna Javan `PriorityQueue` -tietorakennetta, jolle annat konstruktorin parametrina juuri toteuttamasi `Vertailija`-luokan ilmentymän.

Valmis reitti tulee palauttaa taaksepäin linkitettynä listana `Tila`-olioita, joista palautettu tila osoittaa maaliin ja jokainen osoittaa edelliseen tilaan. `Tila`-oliot ovat muuten samanlaisia kuin viime kerralla, mutta niihin on myös liitetty mukaan tieto kuluneesta ajasta.

`Pysakki`-oliot tuntevat naapuripysäkkinsä ja lyhimmän mahdollisen siirtymisajan naapuripysäkeilleen. Pysäkiltä voi kysyä listaa naapuripysäkeistä kutsumalla sen `getNaapurit()`-metodia ja siirtymisaikoja naapuripysäkeille kutsumalla sen `nopeinSiirtyma(Pysakki p, int aika)` -metodia, jossa `p` on jokin pysäkin naapuripysäkeistä ja aika on tähän mennessä kuljettu aika. (Pysäkillä saatetaan joutua odottamaan eri verran aikaa riippuen siitä, mihin aikaan pysäkille tultiin.)

Tehtäväpohja sisältää valmiit testit vertailijalle, heuristiikalle ja A*-haulle.

Jos teet kaiken itse, verkko.json sisältää tiedot pysäkeistä ja `linjat.json` sisältää tiedot raitiovaunulinjoista.