Tehtävä 1. Todennäköisyyslaskentaa. (1 p)

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("kaksi pataa")

b) P("kaksi samaa maata")

c) P("kaksi samaa numeroa (tai kuvakorttia)")

d) P("kaksi samaa numeroa" | "ensin poimittu ei ole kuvakortti")

e) P("numeroiden summa suurempi kuin 3")

Rikoksesta epäillään herra Imlekiä, jonka sormenjälki täsmää ainoan rikospaikalta löytyneen sormenjäljen kanssa. Rikoslaboratorion mukaan sormenjälkianalyysi tuottaa virheellisen tunnistuksen (väärän positiivisen) yhdessä tapauksessa tuhannesta, ja oikea sormenjälki jää tunnistamatta (väärä negatiivinen) yhdessä tapauksessa sadasta.

f) Jos herra Imlekin syyllisyydeksi arvioidaan aluksi 0.001, kuinka todennäköinen se on sormenjälkitunnistuksen jälkeen?

Jos tehtävä tuntuu vaikealta, voit etsiä lisämateriaalia Bayesin kaavasta (engl. *Bayes theorem*). Esimerkiksi oheiset linkit saattavat olla hyödyllisiä:
`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|
-+-+-
O| |X
-+-+-
X| |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 olevan videon ratkaisusta, 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 1. Voit testata algoritmia myös aloittamalla tyhjästä ruudukosta.

Tehtävä 4. Reittiopas: A*-haku. (2 p)

Toteuta edellisen viikon tehtävän ratkaisua muokkaamalla (tai kokonaan alusta) A*-haku luentokalvojen pseudokoodin mukaisesti. Viime viikon leveyssuuntaiseen hakuun lisätään nyt tieto yksittäisistä raitiovaunulinjoista ja niiden aikatauluista. Raitiovaunulinjojen tietoja käytetään laskemaan pysäkiltä toiselle kulkemiseen kuluva aika. Pysäkki-oliot tietävät viereiset pysäkit ja kaikki niille matkustavat linjat. Linja-oliot tietävät kunkin pysäkin välisen ajan (voi vaihdella linjakohtaisesti). Mielikuvitusmaailmassamme kaikki raitiovaunulinjat lähtevät 10 minuutin välein linjan ensimmäiseltä pysäkiltä ja matkaan lähtö tapahtuu 0-9 minuutin kohdalla haun lähtöpysäkiltä. Reittikysely annetaan siis muodossa AStar(lähtöpysäkki, maalipysäkki, lähtöaika int(0-9)).

Kurssisivuilta löytyvät uudet tehtävässä tarvittavat tiedostot. Tiedostojen lukuoperaatiot löytyvät 'AStar.java' tiedostosta ja valmiit Java-luokkien toteutukset 'Pysakki.java' sekä 'Linja.java' tiedostoista. 'verkko.json' (huom. muutettu viime viikosta, kuten myös 'rplot.txt'!) ja 'linjat.json' tiedostot sisältävät pysäkki- ja linjatiedot:

verkko.json - pysäkit

- String koodi - pysäkin yksiselitteinen koodi, käytettävissä pysäkin tunnistamiseen
- String nimi - pysäkin nimi, useilla pysäkeillä saattaa olla sama nimi
- String osoite - pysäkin osoite
- int x - pysäkin x-koordinaatti
- int y - pysäkin y-koordinaatti
- dictionary naapurit - key: naapuripysäkin koodi, value: **Linja.koodi-taulukko**

linjat.json - linjat

- String koodi - linjan yksiselitteiden koodi, käytettävissä linjan tunnistamiseen
- String koodiLyhyt - linjat lyhyt koodi, sama molempiin suuntiin (esim. 4)
- String nimi - esim. Kaivopuisto - Kallio - Eläintarha
- int[] x - kaikkien linjan pysäkkien x-koordinaatit taulukkona
- int[] y - kaikkien linjan pysäkkien y-koodinaatit taulukkona
- String[] psKoodit - kaikkien linjan pysäkkien koodit taulukkona
- int[] psAjat - ajat, jolloin linja on pysäkeillä lähdön jälkeen taulukkona

Leveyssuuntaisesta hausta poiketen sinun tarvitsee tällä kertaa pitää muistissa omassa hakutila-oliossasi ainakin:

- hakutila, josta tähän tilaan on tultu
- aika, joka on jo kulunut lähdöstä
- tämän hetkinen kellon aika (kulunut aika + lähtöaika)
- heuristinen arvio tästä hakutilasta maaliin kuluvasta ajasta (voit olettaa raitiovaunujen maksiminopeuden olevan 526 koordinaattipistettä minuutissa.)

Lisäksi reitin tulostusta varten on hyödyllistä muistaa:

- millä linjalla tähän tilaan tultiin
- kuinka kauan odotettiin viime pysäkillä
- kuinka kauan matkustettiin viime pysäkiltä

Käytä hakutilojen (solmujen) listaukseen prioriteettijonoa ja järjestä ne kuluneen ajan + heuristisen arvion perusteella nopein ensimmäiseksi. Maaliin päästyäsi tulosta reitin jokaisen pysäkin koodi, nimi ja aika, jolloin ko. pysäkillä ollaan sekä millä linjalla pysäkkien väli matkustetaan (odotusaikaa ei ole välttämätöntä erotella erikseen, joskin se voi helpottaa virheiden metsästystä). Leveyssuuntaisen haun tapaan voit halutessasi visualisoida kuljettua reittiä rLine-metodin avulla.