C++ -kielen perusteet-kurssin harjoitustyö 2007 – 2008

Lyhin kuninkaan reitti shakkilaudalla

Tehtävän määrittely

Shakkipelin kuningas voi liikkua shakkilaudalla mihin tahansa sijaintiruutunsa viereisistä ruuduista, myös kulmittain siirtyminen on sallittua. On kirjoitettava ohjelma, joka etsii lyhimmän mahdollisen kuninkaan reitin suorakaiteen muotoisen n*k-shakkilaudan vasemmasta yläkulmasta (ruudusta (0,0)) oikeaan alakulmaan (ruutuun (n-1,k-1)). Osa ruuduista on merkitty varatuiksi, tällaisiin ruutuihin kuningas ei pääse. Esimerkiksi alla on kuvattu 6*5-lauta

	0	**1**	**2**	**3**	**4**	**5**
0		**P**	**P**		**P**	**P**
1		**P**		**P**		**P**
2				**P**		**P**
3		**P**	**P**	**P**		**P**
4			**P**	**P**		

jossa eräs reitti on: (0,0), (0,1), (1,2), (2,1), (3,0), (4,1), (4,2),(4,3) ja (5,4). Tämä on myös lyhin mahdollinen reitti.

Kun lauta luetaan tiedostosta, muodostetaan verkko, jossa solmut vastaavat laudan ruutuja. Väli kuvaa sitä, että kuningas voi siirtyä ruudusta toiseen. Kun verkko on muodostettu, **haetaan leveyshakualgoritmilla lyhin reitti vasemmasta yläkulmasta oikeaan alakulmaan**. Mikäli lyhimpiä reittejä on useita, **yhden mahdollisuuden hakeminen riittää**. Myös tilanne, jossa reittiä ei ole, on otettava huomioon. **Leveyshakualgoritmi on kuvattu liitteessä**. Algoritmia käsitellään myös Johdatus tietorakenteisiin-kurssissa sekä tarkemmin Algoritmit-kurssissa.

Ohjelman syötteet

Ohjelma lukee käytettävän laudan tekstitiedostosta; riveille on merkitty ruudut siten, että E on tyhjä ruutu ja P on varattu ruutu.

Yllä olevan esimerkkilaudan esitys olisi siten

EPPEPP
EPEPEP
EEEPEP
EPPPEP
EEPPEE

Ohjelmalle annetaan kolme komentoriviparametria. Ensimmäinen parametri on optio, joko -show tai -quiet. Näistä ensimmäinen optio (-show) tarkoittaa, että myös komentorivi-ikkunaan tulostetaan ja jälkimmäinen (-quiet), että tulostetaan ainoastaan tiedostoon. Toinen komentoriviparametri on syöttötiedoston nimi ja kolmantena komentoriviparametrina annetaan tulostustiedoston nimi. **Kaikki kolme komentoriviparametria on annettava**, ellei näin tehdä, ohjelma tulostaa ohjeen käytöstään. Ohjelman on pystyttävä käsittelemään erikokoisia lautoja, myös virheelliset syötteet (ensimmäinen optio on väärä; syöttötiedostoa joko ei ole olemassa tai tiedosto ei ole oikeaa muotoa) pitää käsitellä.

Ohjelman tulostus

Aluksi ohjelma tulostaa komentorivi-ikkunaan laudan niin, että ensimmäisellä vaakarivillä ja pystyrivillä ovat koordinaatit ja varattujen ruutujen kohdalle tulostetaan P; muut ruudut voi jättää tyhjäksi. Esimerkkilauta tulostettaisiin

	0	1	2	3	4	5
0		P	P		P	P
1		P		P		P
2				P		P
3		P	P	P		P
4			P	P		

Tämän jälkeen tulostetaan komentorivi-ikkunaan reitti tai tieto siitä, ettei reittiä ole. Reitti tulostetaan merkitsemällä labyrinttiin käydyt ruudut numerojärjestyksessä, esimerkiksi aiemmin annettu reitti tulostettaisiin:

	0	1	2	3	4	5
0	**1**	P	P	**5**	P	P
1	**2**	P	**4**	P	**6**	P
2		**3**		P	**7**	P
3		P	P	P	**8**	P
4			P	P		**9**

Mikäli ohjelman komentoriviparametreja puuttuu tai ne ovat virheelliset, ohjelma tulostaa ilmoituksen oikeasta käytöstään. Mikäli syötetiedosto on virheellinen tai se puuttuu, tulostetaan ilmoitus tästä. **Komentorivi-ikkunaan tulostetaan ainoastaan, jos ensimmäinen komentoriviparametri on -show; muuten tulostetaan ainoastaan tiedostoon.**

Ohjelma tulostaa myös tekstitiedostoon, johon tulostetaan seuraavaa:

1. Jos reitti on olemassa, tulostetaan sen koordinaatit välilyönnillä erotettuna siten, että koordinaattiparit ovat kukin omalla rivillään. Vaakarivin koordinaatti on parissa ensin mainittuna.
2. Jos reittiä ei löydy, tulostetaan tiedostoon pari -1 -1
3. Jos syöttötiedosto on virheellinen (tai sitä ei ole olemassa) tulostetaan pari -2 -2
4. Jos tapahtuu ajonaikainen virhe, tulostetaan pari -3 -3.

Esimerkiksi yllä olevassa tapauksessa ohjelma tulostaisi tiedostoon

0 0
0 1
1 2
2 1
3 0
4 1
4 2
4 3
5 4

Tulostustiedoston nimi annetaan ohjelman kolmantena komentoriviparametrina. Mikäli tiedosto on olemassa, sen päälle saa kirjoittaa ilman eri varoitusta.

Toteutusteknisiä seikkoja

1. Shakkilaudan kokoa ei ennalta tiedetä, joten verkon esityksessä on käytettävä dynaamista tietorakennetta. Staattisesti varattuja taulukoita ei sallita. Jotakin C++-standardikirjaston tietosäiliötä voi luonnollisesti halutessaan käyttää.
2. Ohjelmaan on toteutettava jonkinlainen luokkarakenne, **puhtaasti proseduraalista ratkaisua ei sallita**.

Yleisiä vaatimuksia

Harjoitustyö palautetaan pkzip-arkistointiohjelmalla avautuvana tiedostona käyttäen Prolabin Puzzle-järjestelmää. Harjoitustyön on sisällettävä seuraavat asiat:

1. readme.txt- tiedosto, jossa on tekijän tiedot (nimi, laitos, tiedekunta, opiskelijatunnus ja sähköpostiosoite) sekä mahdollisesti muuta työhön liittyvää tietoa, jota tarkastajan on tärkeää tietää
2. Lähdekielinen koodi, joka sisältää Doxygen-dokumentoinnin. Oletetaan, että kooditiedostot ovat samassa hakemistossa.

Tarkastajat kääntävät koodin Microsoftin Visual Studiolla (.NET 2005). Näin ollen työn laatijan on ennen palautusta varmistuttava koodin kääntymisestä Microsoftin Studiossa, mikäli on käyttänyt jotakin muuta kehitysympäristöä. Yleensä tämä ei vaadi koodin laatijalta suuria ponnistuksia, lähinnä voi joutua tekemään joitakin tyyppimuunnoksia tai

lisäämään otsikkotiedostoja. Ohjelmaa ajetaan Visual Studion debuggerilla, joten on varmistuttava myös ohjelman suoritettavuudesta ko. ympäristössä.

Tarkastajat suorittavat ohjelmaa sekä asianmukaisilla että virheellisillä syötteillä. Virheellisiin syötteihin on reagoitava järkevästi: ohjelma ei saa kaatua eikä joutua ikuiseen silmukkaan ja ilmoitus virheellisestä syötteestä on annettava. Kurssin www-sivuilla annetaan joitakin syötetiedostoja, joilla työtä voidaan testata. Lisäksi tarkastetaan, että ohjelma ei vuoda dynaamista muistia. Tämän voi tarkistaa Visual Studiossa lisäämällä pääohjelmatiedoston alkuun (opiskelija voi jättää koodiin seuraavat määreet)

```
#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
```

ja kutsumalla main -funktion lopussa funktiota

```
_CrtDumpMemoryLeaks();
```

Mikäli ohjelmassa on muistivuoto, siitä tulee ilmoitus Studion output-ikkunaan ajettaessa ohjelmaa debug-moodissa.

Yhteenveto

Alla olevaan listaan on koottu tärkeimmät huomioonotettavat seikat:

1. Ohjelmaa ajetaan syöttämällä komentoriville

<ohjelman_nimi> <-show | -quiet> <input_tiedoston_nimi> <output_tiedoston_nimi>

2. Ohjelma tulostaa komentorivi-ikkunaan laudan ja reitin numeroituna, jos reitti löytyi.

3. Ohjelma tulostaa tekstitiedostoon reitin koordinaatit, jos reitti löytyi. Muuten tulostetaan kohdassa ”Ohjelman tulostus” määritelty koodi, joka kertoo epäonnistumisen syyn.

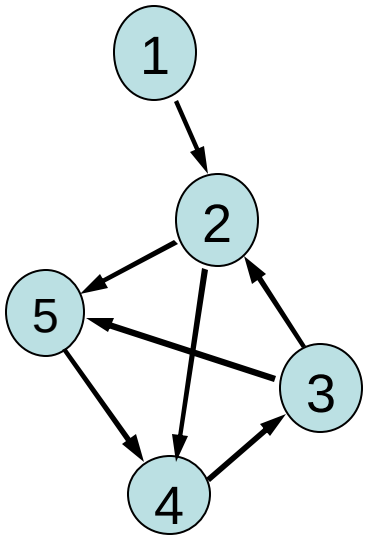
4. Työn on sisällettävä readme.txt-tiedosto ja lähdekoodi, jossa on Doxygen-kommentointi.

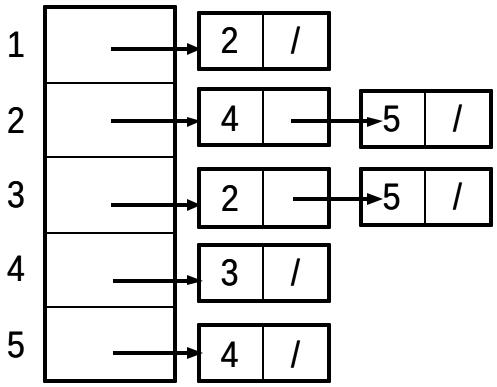
5. Koodin on käännyttävä Visual Studio .NET 2005:llä.

6. Ohjelmasta tarkistetaan muistivuodot.

LIITE: Leveyshakualgoritmi

Verkko on pari G = (V,E), missä V on solmujen epätyhjä joukko ja E on välien joukko, joka on relaatio solmujen joukossa. Joukko E siis määrittelee, mitkä solmut ovat verkossa kytketty toisiinsa. Verkon esittämiseen käytetään tässä vieruslistaesitystä. Jos verkon solmujen lukumäärä on N, vieruslistaesityksessä käytetään taulukkoa Adj, jossa on N linkitettyä listaa. Kun u on solmu, listassa Adj[u] ovat solmun vierustoverit verkossa (ts. solmu v on listassa jos ja vain jos (u,v) on verkon väli). Alla on eräs suunnattu verkko ja sen vieruslistaesitys





Leveyshaku on yksinkertaisimpia menetelmiä käydä verkon solmut systemaattisesti läpi. Sitä voidaan käyttää perustana monenlaisille verkkoalgoritmeille. Olkoon G = (V,E) verkko. Leveyshaussa valitaan jokin solmu s ja käydään välejä systemaattisesti läpi, kunnes kohdataan kaikki solmut, joihin solmusta s voidaan päästä. Samalla lasketaan kunkin solmun etäisyys solmusta s. Leveyshaun nimi johtuu siitä, että algoritmi laajentaa löydettävien solmujen rintamaa tasaisesti koko rintaman leveydeltä. Toisin sanoen ensin löydetään kaikki etäisyydellä 1 olevat solmut, sitten etäisyydellä 2 olevat solmut jne.

Algoritmissa käytetään hyväksi (FIFO-tyyppistä) jonoa Q. Alkioita siis lisätään aina jonon häntään ja otetaan pois jonon keulasta. Tyhjää jonoa merkitään symbolilla EMPTY, solmun u jonoon lisääminen tapahtuu funktion PUSH(Q,u) kutsulla ja funktion POP(Q) kutsu poistaa jonosta solmun ja palauttaa sen. Symboli INF tarkoittaa ”ääretöntä” lukua, joka on suurempi kuin mikä tahansa positiivinen luku. Käytännösssä INF voi olla esimerkiksi -1. Algoritmissa päivitetään kahta taulukkoa d ja p, joissa kummassakin on paikka jokaista verkon solmua kohti. Taulukkoon d päivitetään kunkin solmun etäisyys annetusta syötesolmusta ja taulukkoon p solmun edeltäjä jossakin syötesolmusta lähtevästä lyhimmässä polussa. Näin ollen algoritmin suorituksen jälkeen voidaan taulukosta lukea yksi (mahdollisesti monista) lyhimmistä poluista mihin tahansa solmuun. Apuna käytetään lisäksi taulukkoa color, johon solmut merkitään valkoisiksi,

harmaiksi tai mustiksi (WHITE, GRAY, BLACK). Jos solmu on valkea, sitä ei ole vielä löydetty. Mikäli solmu on musta, sen kaikki naapurit on jo löydetty ja ovat joko mustia tai harmaita. Sen sijaan harmaa solmu on löydetty, mutta sillä voi olla löytämättömiä naapureita (jotka ovat siis valkoisia). Näin ollen harmaat solmut esittävät löydettyjen ja löytämättömien solmujen rajapintaa.

Alla on esitetty leveyshaku pseudokoodina, kun oletetaan käytettävän vieruslistaesitystä verkolle.

```
Syöte: Verkko G=(V,E) ja sen solmu s. Verkolle oletetaan
käytettävän vieruslistaesitystä.
Tulostus: Olkoon N solmujen lukumäärä. N-paikkaiset taulukot p,d
ja color indeksoidaan verkon solmuilla. Taulukkoon d lasketaan
solmun etäisyys syötesolmusta s. Arvo INF tarkoittaa, ettei
solmuun ole polkua. Taulukkoon p lasketaan solmun edeltäjäsolmu
jossakin lyhimmässä polussa syötesolmusta s. Arvo NIL tarkoittaa,
ettei polkua ole.

BFS(G,s)
1.   for each u in V do
2.      color[u] = WHITE
3.      d[u] = INF
4.      p[u] = NIL
5.   end for
6.   d[s] = 0
7.   color[s] = GRAY
8.   Q = EMPTY
9.   PUSH(Q,s)
10.  while Q != EMPTY do
11.    u = POP(Q)
12.    for each v in Adj[u] do
13.        if color[v] == WHITE then
14.           color[v] = GRAY
15.           d[v] = d[u]+1
16.           p[v] = u
17.           PUSH(Q,v)
18.       end if
19.    end for
20.    color[u] = BLACK
21.  end while
22.  return
```

Lyhin polku lähtösolmusta tiettyyn solmuun voidaan lukea edeltäjätaulukosta p käyttäen seuraavaa algoritmia:

```
Syöte: Verkko G=(V,E) ja sen solmut s ja v. Algoritmissa
oletetaan ensin ajettavan BFS(G,s), jonka tuottamaa taulukkoa p
käytetään.
Tulostus: Lyhin polku solmusta s solmuun v tai ilmoitus siitä,
että polkua ei ole. Huomaa, että verkossa voi olla monia lyhimpiä
polkuja. Algoritmi tulostaa jonkin niistä.

BFS_POLKU(G,s,v)
1.   if v==s then
2.      tulosta s
3.   else
4.      if p[v] == NIL then
5.         tulosta “Ei polkua solmusta” s “solmuun” v
6.      else
7.          BFS_POLKU(G,s,p[v])
8.          tulosta v
9.      end if
10.  end if
11.  return
```