Oblig 3 for INF 4130, 2008

Leveringsfristen er mandag. 17. november

NB: Det blir nokså strengt med leveringsfrister denne gangen, siden all godkjenning må være klar til 28. november, slik at Fakultetet kan sette opp eksamenslister etc. Det kan komme presiseringer til oppgaven, så følg med.

Merk at noe av stoffet til oppgave 2 ikke er forelest ennå, men det blir forelest godt før leveringsfristen.

Levering skjer som på tidligere obliger (spesielt skal programmet ha navn nøyaktig “Oppgave1”, på den måten det blir mest naturlig innenfor språket en bruker).

Oppgave 1

Oppgaven er å skrive et program for løsing av "15-spillet", som også kommer som 8-spillet og generelt (NxN-1)-spillet (på et NxN-brett). Dette er diskutert i læreboka som "8-puzzle game" på side 717. Programmet skal lages generelt for NxN-brett (men du kan i implementasjonen forutsette at N<=10). En tilstand for spillet med n=3 kan vises slik:

```
1 2 3
0 4 5
7 8 6
```

Her angir 0 det tomme feltet. Programmet skal da finne frem til en måte å flytte brikkene (et antall ”trekk”) slik at man kommer til måltilstanden:

```
1 2 3
4 5 6
7 8 0
```

(og tilsvarende for NxN-brett). Et lovlig trekk er å la den tomme posisjonen (0-en) ”bytte” posisjon med en av de (maksimalt 4) nabobrikkene. Et slikt trekk angis med hvordan *den tomme posisjonen* flytter seg, med V, H, O, N (for Venstre, Høyre, Opp og Ned). En løsning på problemet over er derved: HHN, og denne skal skrives ut på skjermen om programmet finner en løsning. Merk at løsningen man finner skal være optimal, i den forstand at det ikke finnes noen løsning som har færre trekk. Programmet skal bruke A*-søking, og kan passelig bruke en rett fram Manhattan-heuristikk (se oppgave 23.7).

Programmet bør kunne løse alle 8-spill-problemer på rimelig tid, og noe mer vil ikke bli forlangt. Det er jo imidlertid morsomt å se om det også kan løse enklere 15-spill-problemer (slike som kan løses i få trekk), og det er morsomt om dere legger ved i leveringa en kommentar om hva programmet klarer, og på hvilken tid.

Input og output

Programmet skal få to filnavn ved oppstart, en input-fil og en output-fil. Første linje av input-fila skal inneholde verdien av N, og startilstanden er så angitt på N linjer slik:

```
3
1 2 3
0 4 5
7 8 6
```

Første linje i output-fila skal være antall trekk løsningen er på, denne verdien er entydig bestemt for hver oppgave. Andre linje skal være en sekvens av O, N, V og H, som angir løsningen slik som angitt over (på noen oppgaver er det flere mulige svar her). Tredje til femte linje skal angi hvor mange tilstander en måtte innom etc. (disse verdiene er ikke entydig bestemt oppgaver og vil variere bl.a. avhengig av heuristikk). Nærmere bestemt skal de angi henholdsvis:

- Antall tilstander en besøkte, inklusive starttilstanden og sluttilstanden (altså omtrent antall kall til extractMin i prioritetskøen, litt avhengig av implementasjonen).

- Antall ganger det skjer at en oppdager en ny tilstand (= antall insert i prioritetskøen av en tilstand som ikke ligger i køen fra før).

- Antall ganger en modifiserer kosten til en allerede oppdaget (men ikke besøkt) tilstand (= antall decreaseKey/re-insert i prioritetskøen).

Eksempel på output som løser oppgave over:

```
3
HHN
4
8
0
```

Implementasjonstips

Hver tilstand må representeres på en eller annen måte. Det enkleste er å bruke objekter av en egen klasse State med en byte[]-array, og gjør gjerne det (det går fint siden N<=10). Det finnes også mer plassbesparende representasjoner, f.eks. som ett eller flere ”lange” tall, eller ved å la hver byte representere to posisjoner på brettet. Men disse vil jo bruke mer tid.

A*-søk krever en del mer avanserte datastrukturer enn i foregående oppgaver; en trenger i alle fall en prioritetskø til å lagre køen, og en effektiv oppslagsstruktur (hashmap, splaytre el.l.) til å lagre allerede besøkte noder i. Du kan her bruke en passende innebygd struktur eller et valgfritt open source-bibliotek. Se

http://www.uio.no/studier/emner/matnat/ifi/INF4130/h08/faq.html

for flere tips til hvordan oppgaven kan implementeres.

Oppgave 2

2.a

Bevis at 2-HAMILTONBARHET (definert under) er NP-komplett. Du kan anta som kjent at vanlig HAMILTONBARHET er NP-komplett.

Definisjon av 2-HAMILTONBARHET: Gitt en sammenhengende graf G som input. Bestem om det finnes to enkle løkker i grafen som ikke har noen felles node, og som til sammen inkluderer alle nodene.

2.b

Drøft kompleksiteten til følgende problemer. Du trenger ikke angi nøyaktig kompleksitet, det holder å angi kompleksitetsklasse; for eksempel "polynomisk" eller "NP-komplett". I alle problemene er input en graf *G* og et par av noder (*x*, *y*). Skisser bevis for dine påstander.

Definisjoner: En enkel vei i en graf er en vei hvor ingen noder forekommer to ganger. Lengden av en vei defineres som antall kanter langs veien.

(i) SSP (*G*, *x*, *y*) = Lengden av den korteste vei mellom *x* og *y* (SSP = Shortest Simple Path).

(ii) LSP (*G*, *x*, *y*) = Lengden av lengste enkle vei mellom *x* og *y*. (LSP = Longest Simple Path.)

(iii) SLSP (*G*, *x*, *y*) = En lengste enkle vei mellom *x* og *y* i *G*. (SLSP = Some Longest Simple Path). Algoritmen returnerer en sekvens noder som svarer til en eller annen lengste enkle vei fra *x* til *y*.

(iv) ALSP (*G*, *x*, *y*) = Alle lengste enkle veier mellom *x* og *y*. Algoritmen returnerer alle sekvenser av noder som svarer til lengste enkle veier fra *x* til *y*.