IN1010 V20, Obligatorisk oppgave 5

Innleveringsfrist: Tirsdag 14.4.2020 kl 23:59

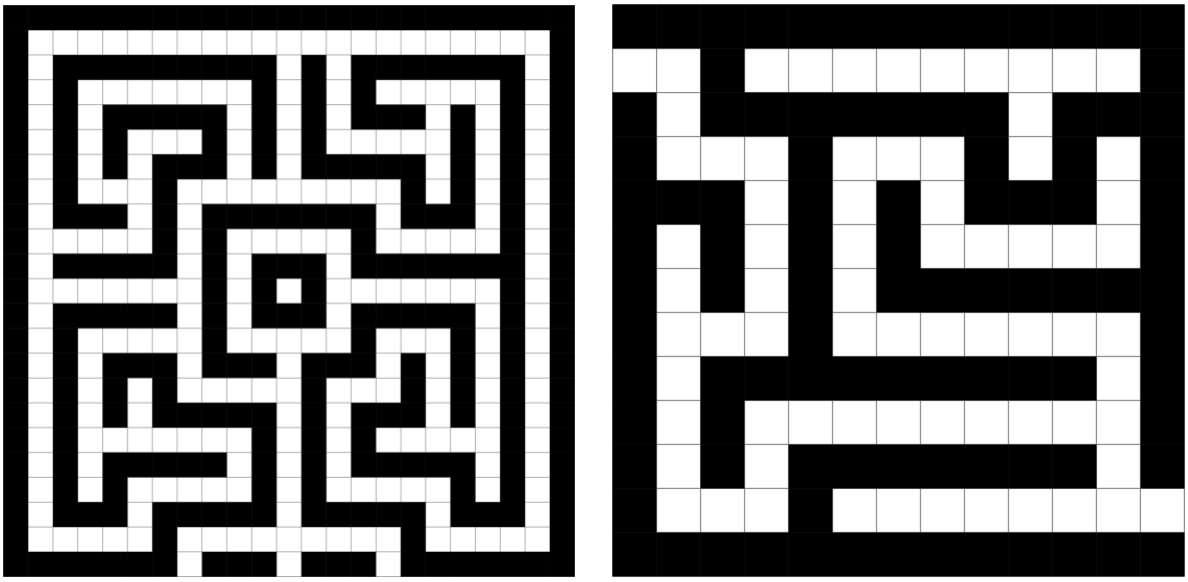
Sist modifisert av Stein Gjessing. Versjon 5.3.2

Innledning

I denne oppgaven skal du bruke rekursjon til å lage et program som er i stand til å finne veien ut av en labyrint. Labyrintene i denne oppgaven er rutenett, bygget opp av kvadratiske ruter som man enten kan gå gjennom eller ikke.

I obligatorisk oppgave 7 skal du lage et grafisk brukergrensesnitt (eng. graphical user interface, ofte forkortet til GUI) til programmet du lager i denne oppgaven.

Figur 1: Labyrint laget av hekker (foto: Karen Blaha, Flickr)



Figur 2: En syklisk (fil 4) og en asyklisk (fil 3) labyrint. Det er kun obligatorisk å løse asykliske labyrinter i denne oppgaven.

Notasjon og terminologi

Under følger to tekster som definerer en del begreper vi får bruk for senere. Den første er strukturert som en serie presise og minimale definisjoner, som i matematikk, og den andre er strukturert som en mer uformell, forklarende tekst.

Formelle definisjoner

- En rute er den minste enheten i labyrinten (hver rute er representert ved ett tegn i filen).
    - Hvite ruter kan man gå gjennom.
    - Sorte ruter kan man ikke gå gjennom.

- To ruter er naboer dersom de har en felles side.

- En vei er en sekvens av hvite ruter slik at hver rute i sekvensen er nabo med den foregående og den etterfølgende ruten, og de to kan ikke være samme rute.

- Hvis det går en vei mellom to ruter, så er rutene tilkoblet hverandre.

- En vei er syklisk hvis minst én av rutene på veien besøkes flere ganger, ellers er den asyklisk.

- En labyrint er syklisk hvis det finnes en syklisk vei i den, ellers er den asyklisk.

- En åpning er en hvit rute på kanten av labyrinten.

- En utvei fra en bestemt startrute er en vei fra den valgte startruten til en åpning.

- Vi bruker koordinater på formen (kolonne, rad) for å identifisere en rutes plassering i labyrinten (merk at vi teller fra 0).

- Posisjon (0,0) er øvre venstre hjørne i labyrinten på en tegning. Dette samsvarer med hvordan man nummererer pixler i grafiske grensesnitt.

Mindre formelle definisjoner

En labyrint er bygd opp av kvadratiske ruter som alle er like store. Hver rute svarer til et tegn i filen. En rute kan være hvit, som betyr at man kan gå gjennom den, eller sort, som betyr at man ikke kan gå gjennom den. Vi sier at to ruter er naboer hvis de har en felles side. En rute har altså inntil 4 naboer.

Vi kan gå mellom et par av hvite ruter som er naboer. Vi kan sette sammen mange slike veistykker for å danne en vei. Hvis det går en vei mellom to ruter, så er rutene tilkoblet hverandre.

Vi sier at en vei er syklisk hvis en av rutene som ligger på veien besøkes flere ganger, og en vei som ikke er syklisk kalles asyklisk.

Vi sier at en labyrint er syklisk dersom det finnes en syklisk vei i den. Dette innebærer at vi må gjøre noen ekstra sjekker i koden for å unngå å bli gående i en sirkel. Figur 2 inneholder blant annet en syklisk vei rundt midten i labyrinten til høyre. En labyrint som ikke er syklisk, kaller vi asyklisk.

Hvite ruter på kanten av labyrinten kalles åpninger, og en vei fra en gitt rute til en åpning kalles en utvei. Vi kan referere til en bestemt rute ved å bruke koordinatene slik: (kolonne, rad). F.eks. har figur 3 to åpninger; (0, 1) og (12, 11).

Noen forenklende antagelser og nyttige resultater

Forenklende antagelser

- Vi skal i denne oppgaven bare se på rektangulære labyrinter (altså er den ytre grensen alltid et rektangel.)
- Vi skal anta at ingen åpninger er nabo med hverandre. Når vi har funnet en åpning, trenger vi altså ikke å sjekke om det finnes veier videre derfra.

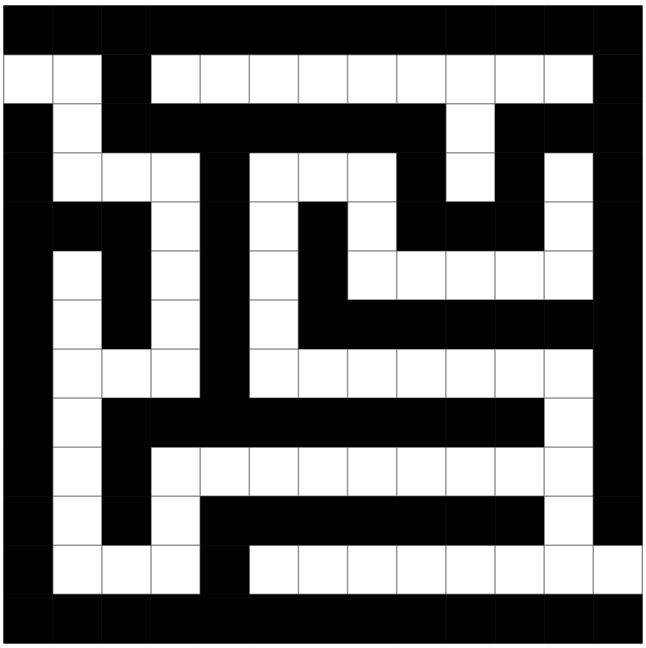
Merk: Vi kan ha områder med hvite ruter som er isolert fra de andre hvite rutene, som f.eks. midten i figur 2 eller området nord-øst i figur 3. Dersom det ikke finnes en vei fra en rute til en åpning vil vi heller ikke finne en løsning.

Filformat

For å representere labyrinter bruker vi følgende filformat: (dette er fil 3 i testfilmappen)

```
13 13
#############
..#.........#
#.#######.###
#...#...#.#.#
###.#.#.###.#
#.#.#.#.....#
#.#.#.#######
#...#.......#
#.#########.#
#.#.........#
#.#.#######.#
#...#........
#############
```

- Den første linjen inneholder to positive heltall som bestemmer hhv. antall rader og antall kolonner i labyrinten (merk rekkefølgen, her er antall rader først).
- Hvite ruter representeres ved . (punktum)
- Sorte ruter representeres ved # (hashtag/firkant)



Figur 3: Labyrinten i fil 3.

Du finner flere eksempel-labyrinter på infosiden til oppgaven.

Del A: Klasser og datastruktur

Under skisséres klassehierarkiet og datastrukturen som skal brukes. Du skal lage klassene *Labyrint*, *Rute*, *HvitRute*, *SortRute* og *Aapning*. Du står fritt til å utvide denne løsningen slik du vil, men da må du begrunne dette gjennom kommentarer i koden.

Labyrint

Labyrint inneholder et todimensjonalt Rute-array med referanser til alle rutene i labyrinten og bør ta vare på antall rader og kolonner. Merk at det er opptil deg å velge hvilken av de to indeksene i arrayet som representerer kolonne og hvilken som representerer rad, så lenge du bruker dette konsistent inne i klassen. Arrayet er uansett en intern representasjon i klassen Labyrint, som ikke aksesseres direkte utenfra.

I tillegg skal hele labyrinten kunne returneres i et format som enkelt kan skrives ut til terminalen underveis (bruk gjerne toString-metoden til dette). Du kan bruke samme representasjon (de samme tegnene) på terminalen som i filformatet, men dette er valgfritt.

Rute

Klassen Rute skal ta vare på sine koordinater (kolonne og rad), og skal også ha en referanse til labyrinten den er en del av. I tillegg skal klassen ha referanser til sine eventuelle nabo-ruter (nord/syd/vest/øst).

Rute skal ha en abstrakt metode *char tilTegn()* som returnerer rutens tegnrepresentasjon (denne skal følge filformatet som beskrevet lenger opp!). Det skal *ikke* være mulig å opprette et objekt av klassen Rute, kun av subklassene.

HvitRute og SortRute

HvitRute og SortRute er subklasser av Rute. Disse må implementere *char tilTegn()* som deklareres i Rute.

Aapning

Aapning er en subklasse av HvitRute og bruker samme datastruktur som sin superklasse.

Del B: Innlesing fra fil

Du skal nå utvide klassen Labyrint. Du skal opprette en konstruktør som tar imot et array av Rute-objekter samt antall kolonner og rader og setter disse inn i instansen. **Det spesielle med konstruktøren er at den skal være *private*, den skal altså ikke kunne kalles utenfor klassen.**

For å kunne opprette Labyrint-objektet skal du deretter utvide klassen med en statisk metode *Labyrint lesFraFil(File fil).* Denne metoden skal gjøre følgende:

1. Lese inn alle linjene fra filen og opprette Ruter basert på fildata (du kan anta at input-filen er gyldig).

2. Opprette selve Labyrint-objektet på bakgrunn av data fra filen.

3. Returnere Labyrint-objektet med all datastruktur ferdig oppsatt.

> **Tilleggsinformasjon:** Dette er et programmeringsmønster som kalles static factory method. For å opprette nye Labyrint-objekter utenfra, må man kalle lesFraFil(File fil) (siden den eneste konstruktøren er private). Dette er en form for innkapsling som skal garantere at datastrukturen i Labyrint alltid blir satt opp riktig (fordi konstruktøren bare kan kalles fra metoder i Labyrint).

Metoden *lesFraFil* skal *ikke* håndtere FileNotFoundException, men bare kaste unntaket videre. Da flyttes ansvaret for å håndtere dette unntaket opp til metoden som kaller lesFraFil. Dette blir viktig når du skal lage et GUI til dette programmet i oblig 7.

Hint 1: Før du begynner å lage Rute-objekter bør du tenke over hvilken rekkefølge ting bør gjøres i og hvilke konsekvenser dette får for programmet ditt. For eksempel må du sørge for at alle ruter har kunnskap om sine naboer før Labyrint-objektet returneres.

Hint 2: Det kan være lurt å ha en egen hjelpemetode som avgjør om en rute skal være en åpning eller ikke.

Du bør teste at brettet er lest inn riktig ved å skrive det ut igjen i terminalen før du går videre til neste del.

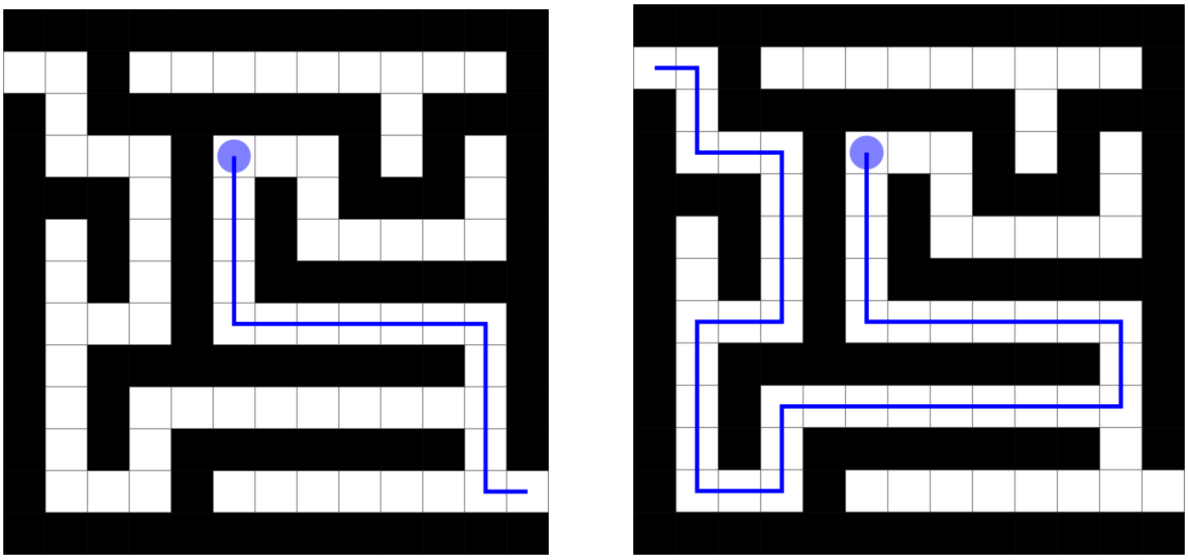
Relevante Trix-oppgaver: 9.01 & 11.01.

Del C: Løsning ved rekursjon

Du skal bruke rekursjon for å finne eventuelle utveier fra en bestemt rute. Denne rutens jobb blir å spørre alle naboruter om å finne en vei videre. Naboruten spør så igjen alle sine naboruter (unntatt den som nettopp spurte) om å finne en vei videre, osv. På denne måten vil vi til slutt komme til en åpning (hvis den finnes).

Kort fortalt er strategien er altså å prøve å gå alle veier bortsett fra den vi kom fra.

Programmet vårt skal finne alle utveier fra en rute i en asyklisk (ikke-syklisk) labyrint. Som en valgfri ekstraoppgave, kan du utvide denne løsningen slik at programmet også kan håndtere sykliske labyrinter.



Figur 4 og 5: De to mulige utveiene fra (5, 3) i labyrinten i fil 3.

Utveier i asykliske labyrinter

Lag en metode gaa i Rute. Denne metoden skal kalle gaa på alle naboruter unntatt den som er i den retningen kallet kom fra (for da ville vi gått tilbake til der vi nettopp var). **Merk:** Det forventes at du implementerer dette uten bruk av metoden *instanceof*.

Hint: Det kan være lurt å lage en hjelpemetode for hver av retningene man kan komme fra. Her bør du også vurdere å skrive ut rutens koordinater ved hvert kall på *gaa* for å se hva som skjer.

Lag så metoden *void finnUtvei()* i Rute som finner alle utveier fra ruten ved hjelp av kall på gaa.

Test programmet før du går videre, for eksempel ved å lese inn fil 3 (figur 3) og kalle finnUtveiFra(5, 3). De to åpningene er (0, 1) og (12, 11) og begge kan nås fra (5, 3). Utveiene er tegnet opp i figur 4 og 5. Sjekk så at det ikke finnes noen utveier fra (5, 1).

Lagring av utvei i en String

Utvid gaa slik at den tar inn et ekstra parameter som inneholder koordinatene til rutene på den foreløpige veien. Når du kommer til en ny rute, legger du denne rutens koordinater til strengen før du kaller den rekursive metoden i nabo-rutene. La strengen være på formen (kolonne, rad) --> (kolonne, rad) --> (kolonne, rad).

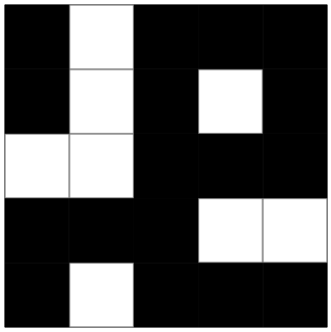
Merk: Vanligvis når vi sender med en referanse som parameter til en metode, så kan metoden forandre det objektet som parameteren refererer til. Men dette gjelder ikke objekter av klassen String, siden disse er ikke-foranderlige (eng. immutable). Når et Rute-objekt legger til sine koordinater på slutten av strengen, lager den i realiteten et helt nytt String-objekt.

Når du har funnet en utvei, skal strengen med utveien legges inn i en Liste<String> (du kan selv velge hvilken av subklassene fra obligatorisk oppgave 4 som skal brukes). Du kan lagre

listen med utveier som en instansvariabel i Labyrint og endre denne ved behov. Lag metoden finnUtveiFra(int kol, int rad) i Labyrint til å returnere denne listen. Husk at det skal være mulig å kalle denne metoden flere ganger på forskjellige koordinater.

Hovedprogram

Det er gitt et hovedprogram på infosiden. Dette programmet leser inn filnavnet til en labyrint fra kommandolinjeargumentene og oppretter et Labyrint-objekt. Deretter lar programmet bruker oppgi koordinater til én og én rute og skriver ut utveiene fra disse. Dette gjør at man enkelt kan teste utveier fra mange ruter i samme labyrint. Se følgende eksempel på bruk (husk at koordinatene er med kolonne først, så rad).



Figur 6: Labyrinten i fil 7.

```
$ java Oblig5 7.in

1 1
(1, 1) --> (1, 2) --> (0, 2)
(1, 1) --> (1, 0)

3 1
Ingen utveier.

3 3
(3, 3) --> (4, 3)
```

Det kan hende at ditt program gir en annen rekkefølge på utveiene fra én bestemt rute, men det vesentlige er at det finner de samme utveiene.

OBS: Det er svært nyttig å gjøre utskrifter underveis i testingen av programmet, og du oppfordres derfor til å gjøre det, for eksempel ved å skrive ut stegene i letingen etter en utvei. **Men:** Når klassene dine er ferdige skal det være mulig å kjøre hovedprogrammet *uten disse utskriftene* (med unntak av eventuelle feilmeldinger). Du kan løse dette ved å fjerne alle utskriftene før du leverer oppgaven, men et annet forslag er å gi brukeren mulighet til å legge inn et ekstra argument når programmet kjøres og kun vise test-utskrifter dersom dette argumentet er sendt ved (for eksempel nøkkelordet “detaljert”. Et eksempel på kjøring kan se slik ut:

```
$ java Oblig5 7.in detaljert
```

Oppsummering av del C

- Programmet skal ved hjelp av rekursjon klare å finne alle mulige utveier fra en vilkårlig rute.
- Vi skal bare finne asykliske veier.
- For hver utvei vi finner, skal vi lagre en String som beskriver utveien. Disse strengene skal returneres i en Liste<String>.
- Det skal være mulig å skru av all utskrift utenom feilmeldinger.

Relevante Trix-oppgaver: Alle oppgaver om rekursjon, men spesielt 8.02, 8.04, 8.06 & 8.07

Oppsummering

Du skal levere alle klasser som skal til for at hovedprogrammet skal fungere (inkludert det utgitte hovedprogrammet og nødvendige liste-klasser du har laget tidligere).

Alle delene av programmet må kompilere og kjøre på Ifi-maskiner for å kunne få oppgaven godkjent. Unngå bruk av packages (spesielt relevant ved bruk av IDE-er som IntelliJ). *Ikke levér zip-filer!* Det går an å laste opp flere filer samtidig i Devilry.

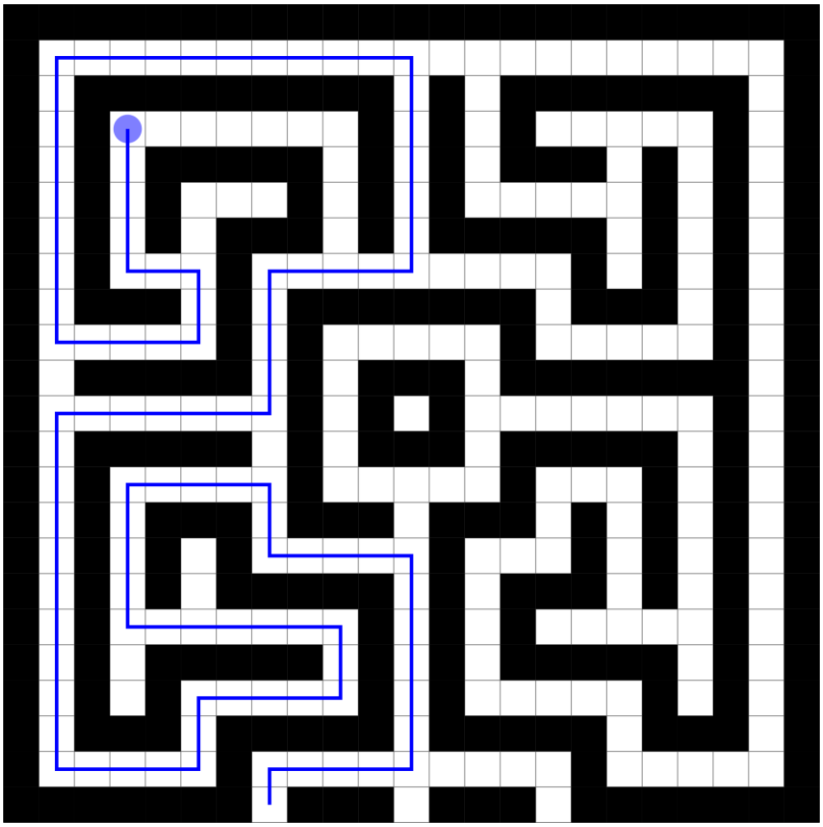
Merk: Resten av oppgaven er frivillig (men anbefalt).

Del D: Valgfri del

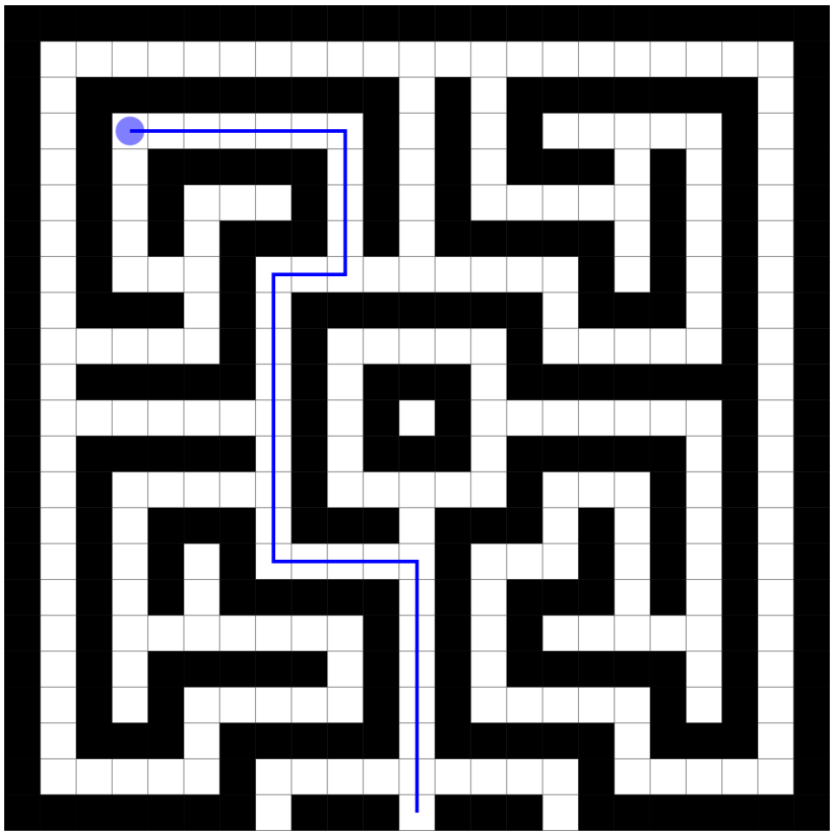
Utveier i sykliske labyrinter

For å kunne håndtere sykler, må vi tilføre algoritmen vår et nytt element – vi trenger å merke veien vi har gått slik at vi kan snu når vi kommer til en rute som allerede er på veien. Vi kan se for oss at vi ruller ut en snor etter oss, og når vi går tilbake ruller vi opp igjen snoren slik at den kun ligger innenfor rutene som ligger på veien. Denne teknikken kalles rekursiv tilbakesporing (eng. recursive backtracking).

Det er greit å ha i bakhodet at vi skal finne enhver utvei. I sykliske labyrinter vil hver sykel kunne gi opphav til mange utveier. Noen av disse er åpenbart ikke optimale, men de er likevel korrekte. Se på figur 7 og 8. Utveien i figur 7 er åpenbart fryktelig ineffektiv – faktisk er den 3 ganger så lang som den i figur 8!



Figur 7: En tilfeldig utvei fra (3, 3) i labyrinten i fil 4 (figur 2)



Figur 8: Den korteste utveien fra (3, 3) i labyrinten i fil 4 (figur 2)

Utfordring: Klarer du å se hvor mange utveier som finnes fra (4, 4) i figur 2?

Modifiser koden din slik at vi holder styr på om vi har vært innom rutene under letingen etter en utvei og ikke går gjennom samme rute flere ganger. Det kan også være lurt å gjøre noen endringer i representasjonen av rutene slik at letingen kan visualiseres fortløpende.

Korteste utvei (valgfri)

Utvid programmet ditt slik at det finner den (eller én av de) korteste utveien(e) (hvis det finnes noen utveier). Skriv også ut hvor mange utveier som ble funnet.