Pathfinding
Breadth-First Search

by F. Sørensen frederik@patch.dk

26. december 2005

1 Forord

Denne artikel er baseret på et afsnit i rapporten webspil som jeg har været med til at skrive på 1.Semester, Datalogi på Aalborg universitet. Den er omskrevet en smule for at passe til en artikel. Rapporten kan findes på www.patch.dk/files/rap/1.sem.dat.pdf

Eksempel kode der understøtter denne artikel kan findes på www.patch.dk/files/pathfinding/pathfinding.zip Eksempel koden er skrevet i PHP5 og kræver derfor en webserver med PHP5. Koden kan dog ses på www.patch.dk/files/example/pathfinding/driver.php man kan vælge labyrint ved at skrive følgende url, www.patch.dk/files/example/pathfinding/driver.php?maze=1 der er fra 1-4 labyrinter.

2 Pathfinding

Formålet med pathfinding eller grafteori som pathfinding er, er at se på forskellige metoder, til at finde vej mellem to punkter på et kort.

Pathfinding er proceduren til at finde en vej fra ét sted til et andet sted på den mest effektive måde. Det kan f.eks være den korteste, billigste eller mindst farefulde vej. Hvis man kigger på spil, er det meget vigtigt, at en computerspiller kan finde frem hurtigt og effektivt. Her kan der være tale om at få flyttet sin helt gennem et landskab, og undgå at han bliver angrebet på ruten. Derfor kan ruten optimeres efter sandsynligheden, for at han bliver angrebet. Ofte vil det dog være forvirrende for en spiller at ruten findes ud fra farer, da det kan lede til umiddelbart ulogiske stier. I stedet benytter spil ofte kun at det er den korteste vej der findes uden hensyn til andre faktorer.

Vi starter med at definere nogle elementer i grafteori:

Definition 2.1 *En knude er et basiselement. Et unikt "punkt" på en graf.*

Definition 2.2 *En kant er et sæt af to knuder.*

Vi kan nu definere hvad en graf er. Der findes to forskellige typer grafer, orienteret og ikke-orienteret:

Definition 2.3 *En ikke-orienteret graf G er defineret ved et ordnet par $G = (V, E)$ hvor:*

- *V er en ikke tom mængde af knuder, $V \neq \varnothing$ hvor det gælder at $< v_1, v_2, \cdots, v_i > \in V$*
- *E er en mændge af uordnede par af knuder. De knuder som tilhører en kant kaldes også endeknuder*

V og E er endelige mængder.

Definition 2.4 *En retningsbestemt graf G er defineret ved et ordnet par $G = (V, A)$ hvor:*

- *V er mængde af knuder $< v_1, v_2, \cdots v_i > \in V$*
- *E er et ordnet mængde af knuder også kaldet pile.*
- *En kant $e = (x, y)$ har retning fra x til y. y er kaldet hovedet og x halen af kanten.*

V og A er endelige mængder.

For at kunne tage højde for forskellige typer underlag på f.eks et spilkort kan man bruge vægtede grafer. Med vægtede grafer har man mulighed for at tillægge en vægt til hver kant.

Definition 2.5 $w : E \longrightarrow \mathbb{R}_\infty$*, hvor* $\mathbb{R}_\infty = \mathbb{R}_{V\{\infty\}}$
En vægt er en værdi, der knytter sig til en kant og viser, hvad det koster at følge kanten.

2.1 Algoritmer

Herefter beskrives én algoritme inden for grafteori, der specifikt handler om at finde den korteste vej imellem to knuder. Breadth-First Search (BFS), er af de simpleste algoritmer, som kan benyttes til at løse korteste-vej-problemer. Der er mange forskellige algoritmer til at bestemme forskellige ting i grafer, for at give et overblik over andre mulige algoritmer er der kort beskrevet nogle få af dem her.

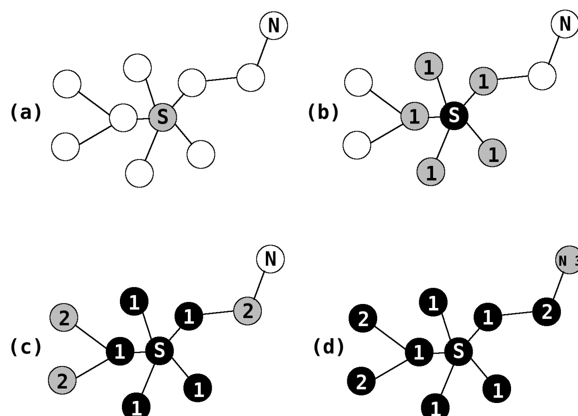
Dijkstra's algoritme minder meget om BFS med den store forskel, at den arbejder på grafer, der har tilknyttet en vægt til hver kant. Formålet med vægte, er at man har mulighed for at beskrive et spil kort mere præcist, end man kan uden. F.eks. kan en forbindelse imellem to steder, der går hen over en flod, uden en bro, have en uendelig positiv værdi, den samme forbindelse, blot med en bro, vil have en lavere positiv værdi, fordi nu er der en mulighed for at bevæge sig over floden. Algoritmen vil forstå at den positive værdi er lavere hen over broen, frem for hvis broen ikke var til stede, så vil denne sti være en mulighed. Ligeledes, vil en sti der forbinder de to punkter på hver side af floden, også være en mulighed, men hvis den samlede værdi på stien der går over broen er lavere end den samlede værdi af stien der går over land, så vil Dijkstra's algoritme forstå at stien hen over broen er den korteste af de to stier.

A* algoritmen, som udtales A star, benytter sig af informationer, den er blevet forsynet med, til at træffe nogle beslutninger omkring, hvor den skal hen efter det næste punkt. Det vil sige, slutknuden er kendt fra starten, og den har også en idé omkring, hvad der er den korteste vej. A* vil f.eks., hvis den skal finde frem til et hus, der er nord fra den nuværende position, vil den vide, at det er smartest at prøve at gå nord først, og se hvad der sker. Dette er anderledes end Dijkstra's algoritme, den vil kigge på alle knuder i alle retninger indtil den finder den korteste vej frem til slutknuden. Der vil A* kigge i den retning der er mest sandsynlig at have den korteste vej.

2.2 Breadth-First Search

Breadth-First Search(BFS) er en af de mest grundlæggende og simple algoritmer, man kan bruge til at søge på en graf. Ideen i BFS er at søge gennem alle kanter på grafen, for at finde alle knuder, der kan tilgås fra rodknuden. Ved at algoritmen søger alle kanter igennem, og den finder alle knuderne, så er det herefter muligt at opbygge et Breadth-First Tree(BFT).

Når man har søgt gennem alle kanter og har fundet alle knuder, har man opbygget et Breadth-First Tree(BFT), og kan herefter bestemme den korteste rute fra rodknuden til en hvilken som



Figur 1: BFS søgning

helst anden knude. Kortest muligt betyder her mindst antal kanter. Søgningen forgår ved, at man fra rodknuden går til alle knuder, der er mulige gennem én kant, først når alle disse knuder er fundet fortsætter søgningen med de knuder, der er to kanter væk osv.

Givet en graf G = (V,E) og en rodknude s undersøger BFS systematisk kanter for at finde samtlige knuder, der kan nås fra s. BFS danner herved et BFT over alle knuder, tilgængelige med s som rodknude. For at holde styr på hvor langt man er kommet, benytter BFS sig af farvekoder. Hvide knuder er ikke opdagede knuder, grå er opdagede ydreknuder og sort er knuder, der er opdaget, men ikke er ydreknuder. Når BFS-algoritmen har opdaget alle knuder, er det derefter muligt at bestemme afstanden til en vilkårlig knude, givet den er fundet under søgningen, ved at tælle antal kanter imellem startknuden, og en vilkårlig knude.

Ofte vil man finde korteste vej fra en bestemt knude til en anden bestemt knude. Hvis man ser på figur 1, kan man se at først findes knuder, der er én kant fra rodknuden, derefter udvides der til knuder, der er to kanter fra rodknuden osv. Til sidst findes slutknuden N, som er tre kanter fra startknuden, og dermed den korteste vej fra start til slut. Metoden der benyttes til at beregne den korteste vej, er at ligge kanterne sammen og derefter sammenligne hvilken rute der er kortest, for at nå til slutknuden.

Definition 2.6 *G er en graf, G = (V,E), s er begyndelsesknuden.*

Definition 2.7 *color[u], er knuden u's farve. Der er tre farver defineret, hvid er en endnu ikke besøgt knude, grå er en kantknude og sort er en allerede besøgt knude.*

Definition 2.8 *d[u] er afstanden fra knuden s(begyndelses knuden) til knuden u.*

Definition 2.9 *pi[u] er knuden der er opdaget lige før knuden u.*

Definition 2.10 *Q er er liste af knuder, der søges i.*

Definition 2.11 *head[Q] retunerer første element fra listen Q.*

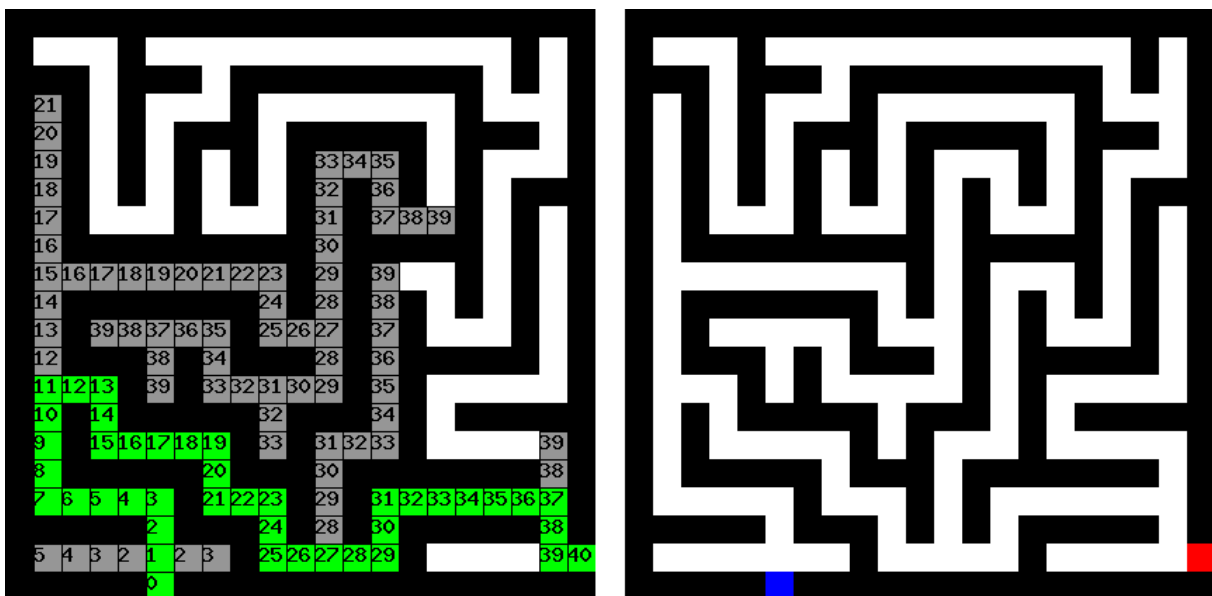
Definition 2.12 *ENQUEUE(Q, v), lægger knuden v ind i listen Q. DEQUEUE(Q) fjerner en knude fra listen Q.*

Definition 2.13 *Adj[u] er liste af knuder der kan tilgås fra knuden u.*

Følgende viser et eksempel på pseudo-kode på en BFS algoritme:

```
1  Begin(G, s)
2  for each vertex u in V[G] - {s}
3    do color[u] <- WHITE
4       d[u] <- infinite
5       pi[u] <- NIL
6  color[s] <- GREY
7  pi[s] <- NIL
8  Q <- {s}
9
10 while Q not equal empty
11   do u <- head[Q]
12      for each v in Adj[u]
13        do if color[v] = WHITE
14           then color[v] <- GREY
15              d[v] <- d[u] + 1
16              ENQUEUE(Q, v)
17      DEQUEUE(Q)
18      color[u] <- BLACK
```

Vi har implementeret BFS i PHP, for at vise hvordan man kan bruge den til at finde vej på et kort.



Figur 2: BFS søgnings eksempel

BFS søger ud i en stjerne form, indtil den finder slutknuden.

3 Eksempel koden

Eksempel koden er skrevet i PHP, det kan sagtens argumenteres for at det er et mindre optimalt sprog at lave pathfinding i. Det har dog nogle fordele der gør at det er udemærket til at bygge eksempler i. Primært er det hurtigt at teste i, og de fleste har adgang til en webserver med php så det er let at bruge eksempel koden.

Koden er delt op i fire klasser og lidt driver kode. Koden i driver.php er den del der opretter objekter og kalder de overordnede metoder i objekterne. Først oprettes et ImageAnalyzer objekt der sørger for at billedet af labyrintern kan analyseres. ImageAnalyzer objektet gives som argument til Map objektet som henter det info ud af billedet som der skal bruges for at oprette en intern datastruktur over labyrinten. Tilsidst får BFS objektet Map som argument og søgningen kan startes.

Hvis man ser på billederne af labyrinterne kan man se at de er sorte og hvide, de hvide felter kan man gå på de sorte er vægge. Der er også et blå og et rød felt. Det blå felt er start feltet og det blå er slut feltet.

Når søgningene startes i BFS::startSearch() metoden bruges BFS algoritmen til at søge gennem datastrukturen som er oprettet i Map objektet. Alle besøgte felter vil blive tegnet med grå og afstanden fra startknuden i sort tekst. Hvis der findes en sti fra start til slut punktet vil det blive tegnet med grønne felter med afstanden skrevet i sort i dem.