# **Quoridor-AI**

## **Graf-Konstruktion**

### **Noder**

Varje ruta på spelbrädet måste läggas till i en lista för användning, detta betyder att antalet noder som kommer läggas till är bredden multiplicerat med höjden vilket är antalet noder. Det ger en tidskomplexitet på O(N). Utrymmeskomplexiteten blir detsamma eftersom den kommer behöva använda (N *×* Nod) mängd utrymme för att lagra alla noder vilket blir O(N).

### **Bågar**

För varje nod som lags till skall det finnas bågar som går emellan varandra som indikerar för spelaren/motståndare vart de kan gå eller var det är mer lämpligt att gå. För att lagra information om alla bågar kan en adjacency lista användas fast istället har jag valt att lagra varje båge som ett unikt objekt i en separat lista. För varje båge innehåller information vilken nod den tillhör och vilken nod den går till.

I ett kvadratiskt fält beroende på om rutan ligger i ett hörn eller vid kanten så kan de antingen ha 4, 3 eller 2 bågar som går emellan varandra. I fallet av Quoridor kommer antalet bågar maximalt bli *((4 × 2) + ((bredd – 2) × 2 × 3) + ((höjd – 2) × 2 × 3) + ((bredd – 2) × (höjd – 2) × 4) | alt. 2(2 \* bredd \* höjd – bredd – höjd) | alt. 4(N – √N)*

Ex. Quoridor har 9x9 rutor som ger 4(81 – √81) = 288 bågar.

Däremot har Quoridor väggar som kan sättas ut på spelplanen vilket kommer ta bort bågar som går mellan noder där väggen blivit placerad. För att konstruera en graf med aktuella bågar krävs en algoritm vilket kollar om det finns en vägg placerad mellan denna nod till en granne, och i så fall, lägg inte till en båge mellan dessa. I Quoridor kommer varje nod ha medelvärdet fyra grannar som kommer behöva kollas för varje ruta. Eftersom varje nod kollas och dess grannar kommer algoritmen få en tidskomplexitet på (~ 4 × N) som ger O(N). För att lagra alla bågar krävs en utrymmeskomplexitet på O(E) där E ≤ 4(N – √N), N är dominerande faktor vilket således betyder att O(E) kan skrivas om till O(N). För mängden noder som växer kommer antalet bågar växa lika snabbt i utrymme.

## **A\***

### **Prioritetskö (Min Heap)**

För att snabbt hitta den bästa noden till målet, används en prioritetskö vilket är en min heap. Den snabbt lagrar det minsta värdet på toppen av heapen vilket gör det effektivt för A\* att alltid välja noden med minst värde i open. Vid insättning och utdrag har den en tidskomplexitet på O(log n) där n är höjden på trädet. Utrymmeskomplexitet för de operationer är O(1) då de inte kräver något extra utrymme för att utföras.

### **Sökning**

Problemet med Quoridor är att spelet har flera mål som spelaren kan gå mot. A\* är optimalt för att hitta den kortaste vägen mellan start och mål där båda oftast bara finns en av. För att få A\* att fungera optimalt med flera mål behövs en omvänd A\*. Där istället startpunkten är målet och alla mål är startpunkten för algoritmen. Det betyder att den kommer i början utvärdera för varje start dess grannar och sedan därifrån ta den mest optimala noden till det satta målet. Detta betyder att tidskomplexitet kommer växa linjärt för antalet mål det finns. I Quoridor är det hur lång en sida är eller √N antal noder. Därför blir tidskomplexiteten för A\* O(M + bd) där M = antalet mål. Utrymmeskomplexiteten blir dock O(bd) eftersom efter varje mål utvärderas tas det bort från kön och därefter fortsätter det som normalt. Dessutom kan storleken för M aldrig bli större än vad det blir för sökningen i värsta fall vilket betyder att bd dominerar över M i utrymme.

Annars fungerar A\* som den skall där den använder viktade bågar som längd från start (standard = 1) och Manhattangeometri som heuristisk för längd till mål.