DM574: INTRODUKTION TIL PROGRAMMERING
Projekt: Del III
afleveringsfrist: fredag, d.5. januar, kl.18:00

Oversigt

I denne fase af projektet implementeres den automatiske spiller, som bruger minimax-algoritmen til at finde ud af, hvad sit næste træk skal være.

Minimax algoritme

Minimax er en algoritme, der kan bruges til at vælge det bedste træk til den næste spiller i en 2-personerssspil. Princippet bag den er, at man bygger et *spiltræ* hvor alle mulige sekvenser af træk er repræsenteret, for at kunne vælge det træk, der garanterer at man vinder (hvis muligt). Dette kan dog kun gøres for meget simple spil (fx. den "internationale" udgave af Tic-Tac-Toe), da der for mere komplicerede spil er for mange mulige sekvenser, man skal generere.

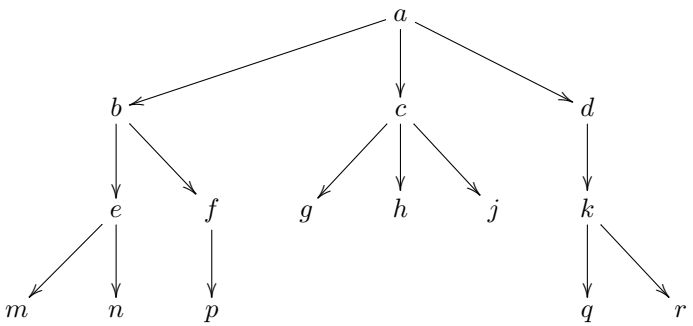
I praksis vælger man en *dybde*, der begrænser hvor længe disse skvenser skal være – dvs, hvor mange træk frem man vil analysere. De mulige brætstater bliver så vurderet ved brug af en *heuristik* – en funktion, der vurderer hvor god en stat er, hvor en bedre stat er en stat, hvor der er større sandsynlighed for at vinde spillet fra.

Spiltræet bygges i to trin. I det første trin bygges der et træ, hvor hver knude er en stat af spillet. Roden er den nuværende stat. For hver knude n og alle mulige træk fra n, er der en knude der er barn til n, og hvis stat er spillets stat efter trækket spilles fra ns stat. Træts højde er begrænset til at være den valgte dybde, hvilket betyder at en knude, hvis afstand fra roden er lige med dybden ikke får børn.[1]

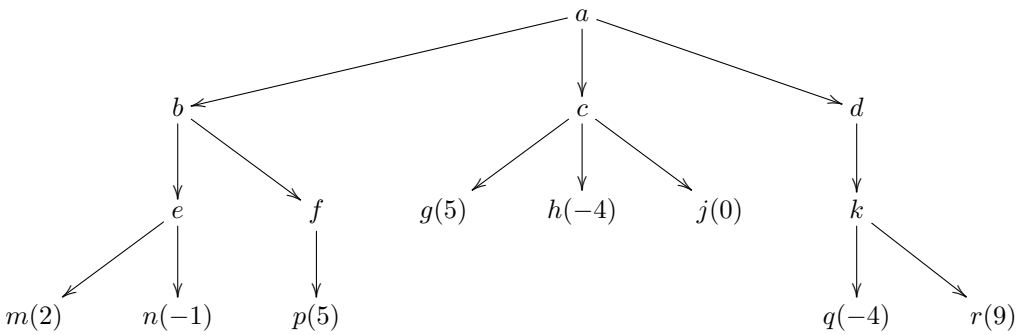
I det andet trin får alle knuder en værdi. Alle knuder, der ikke har børn bliver vurderet ved brug af heuristikken. De andre knuder er enten maxknuder eller minknuder: roden er en maxknude, rodens børn er minknuder, deres børn er igen maxknuder, osv. (Dvs at det er "spillerens" tur i alle maxknuder, og "modstanderens" tur i alle minknuder.) Værdien af en maxknude er maksimum af alle dens børns værdier, og værdien af en minknude er minimum af alle dens børns værdier.

Eksempel

Lad os sige, at det første trin dannede dette minimaxtræ, hvor a, b, ..., r er brætstater:

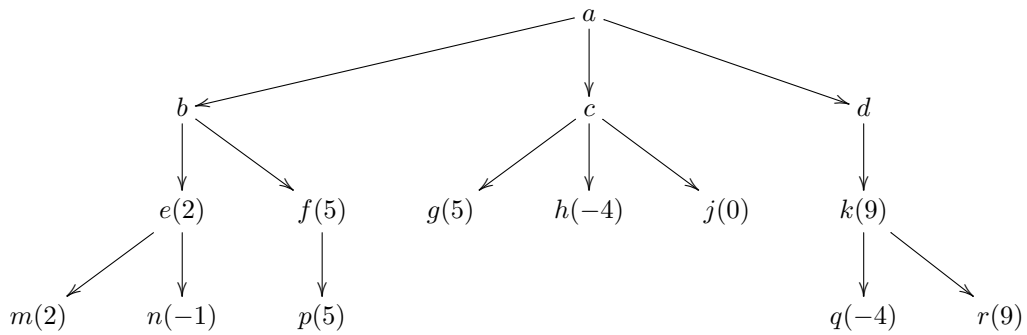


Vi starter med at bruge heuristikken til at give værdier til alle knuder, der ikke har børn – dvs g, h, j, m, n, p, q og r. Lad os antage at vores beregnede værdier er dem som kan ses på næste billede i parenteserne.

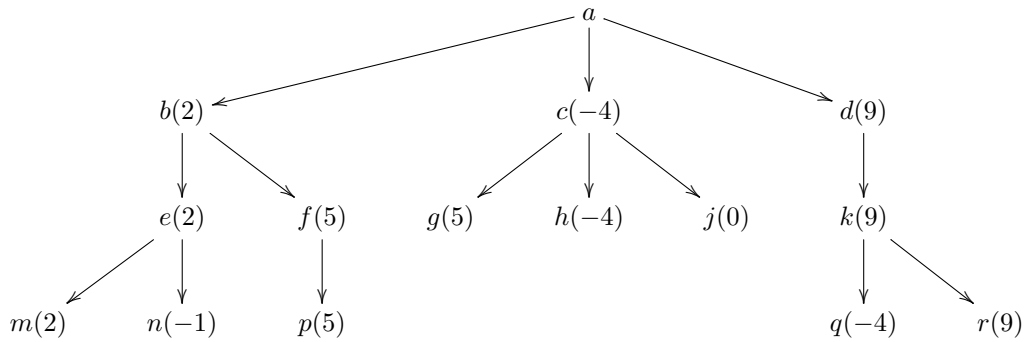


[1] I denne sammenhæng er "dybde" og "højde" mærkeligt nok synonymer – "dybde" er terminologi fra søgningsalgoritmer, "højde" er et standard begreb ifm træer.

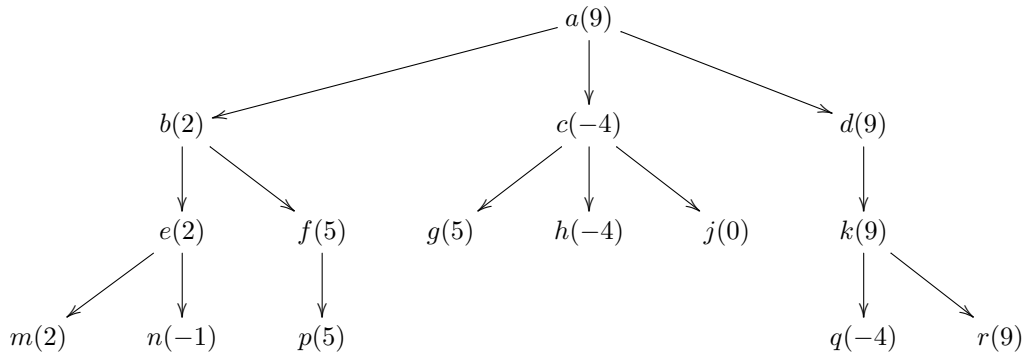
Knude e, f og k er maxknuder, så de får den største af deres børns værdier.



Nu er knude b, c og d minknuder, så deres værdier bliver den mindste af deres børns værdier.



Til sidst er rodknuden a igen en maxknude, så den får den højeste af sin børns værdier.



Algoritmen returnere det træk, der fører os fra brætstat a til brætstat d (da d har den højeste værdi af as børn). Hvis der er flere knuder der har den samme, højeste, værdi, vælges der en af dem.

Implementering

Implementering af minimax kræver en dedikeret datastruktur til minimaxtræer, som dog ikke er relevant for klienter – de skal kun bruge en funktion, der tager brættet og dybden som argument, og returnerer det næste træk. Denne funktion skal så lave et nyt træ, beregne det bedste træk ifølge den algoritme beskrevet ovenpå, og returnere det.

Modul `minimax.py`

Dette modul indeholder som sagt kun en metode:

- `next_move(b: Board, depth: int) -> Move`, som returnerer det bedste træk for den næste spiller – fundet ved at bygge et minimaxtræ med de givne bræt og dybde. Parametren `depth` er valgfri.

Implementeringstips

Det meste af arbedjet er at bygge minimaxtræt. Det kan være en god idé at definere nogle hjælpemetoder til at gøre jobbet mere håndterligt.

Kvaliteten på det svar fundet af minimax er afhængig af, hvor god en heuristik man bruger. Det er en god idé at starte med en simpel heuristik, og når alt er på plads kan der eksperimenteres med mere og mere sofistikere heuristikker. På den måde kan man sørge for, at projektet er færdigt til aflevering, og at man har mulighed for at lave så god en autospiller som muligt.

Forventede resultater

Hver gruppe skal aflevere en zipfil uden mapper, som indeholder de følgende filer.

- En python sourcefil `minimax.py` som implementerer den beskrevne funktionalitet.
- Et pdf dokument `rapportXX.pdf`, hvor `XX` erstattes af gruppenummer, som indeholder en rapport, der beskriver bl.a.: den implementerede algoritme, alle relevante designvalg, eksempler, overvejelser ifm heuristikken. Alt kildekode skal også inkluderes i appendix.

Hvis disse regler ikke følges, kan projektet blive afvist automatisk. Rapporten er basis på evaluering.