DM536/DM574: INTRODUKTION TIL PROGRAMMERING
Gruppeprojekt — Efterår 2024 — Fase I
Afleveringsfrist: d. 8. november 2024, kl.18

Oversigt

Formålet med den første fase af projektet er at udvikle de grundlæggende moduler, som resten af programmet bygger på: `comparator.py` og `network.py`.

Modul `comparator.py`

Dette modul definerer en datatype `Comparator`, hvis instanser repræsenterer de gates, som sorteringsnetværk bygges fra. Modulet skal derudover indeholde:

- en funktion `make_comparator(i: int, j: int) -> Comparator`, som danner en ny comparator mellem kanaler `i` og `j`;
- funktioner `min_channel(c: Comparator) -> int` og `max_channel(c: Comparator) -> int`, som returnerer hhv. den kanal, hvor den laveste/højeste værdi bliver lagt på af `c`;
- en funktion `is_standard(c: Comparator) -> bool`, som tjekker om `c` er en standard comparator (dvs. at den lægger den laveste værdi på den laveste kanal);
- en funktion `apply(c: Comparator, w: list[int]) -> list[int]`, som anvender en comparator på en liste af heltal;
- en funktion `all_comparators(n: int) -> list[Comparator]`, som returnerer en liste med alle mulige comparators på `n` kanaler;
- en funktion `std_comparators(n: int) -> list[Comparator]`, som returnerer en liste med alle standard comparators på `n` kanaler;
- en funktion `to_program(c: Comparator, var: str, aux: str) -> list[str]`, som returnerer en sekvens af instruktioner, der vil simulere comparatoren i Python (se forklaring nedenunder).

Modul `network.py`

Dette modul definerer en datatype `Network` hvis instancer er comparatornetværk. Modulet skal derudover indeholde:

- en funktion `empty_network() -> Network`, som danner et tomt comparatornetværk;
- en funktion `append(c: Comparator, net: Network) -> Network`, som tilføjer en comparator i enden af et netværk;
- en funktion `size(net: Network) -> int`, som returnerer antallet af comparators i `net`;
- en funktion `max_channel(net:Network) -> int`, som returnerer den højeste kanal, som netværket rører ved;

- en funktion `is_standard(net: Network) -> bool`, som tjekker om `net` kun indeholder standard comparators;
- en funktion `apply(net: Network, w: list[int]) -> list[int]`, som anvender et comparatornetværk på en liste af heltal;
- en funktion `outputs(net: Network, w: list[list[int]]) -> list[list[int]]`, som anvender `net` på alle input i `w` og returnerer en liste uden duplikater;
- en funktion `all_outputs(net: Network, n: int) -> list[list[int]]`, som returnerer alle mulige binære outputs af længde `n` fra `net`;
- en funktion `is_sorting(net: Network, size: int) -> bool`, som tjekker om `net` er et sorteringsnetværk til `n` input;
- en funktion `to_program(net: Network, var: str, aux: str) -> list[str]`, som returnerer en sekvens af instruktioner, der vil simulere netværket i Python (se forklaring nedenunder).

Kommentarer, forklaringer og tips til implementering

- For at teste om et comparatornetværk er et sorteringsnetværk for n input skal man i princippet tjekke om det sortere alle mulige sekvenser af længde n. Det kan tjekkes ved at teste, at netværket sorterer alle permutationer af de heltal $1, \ldots, n$. Der er dog en bedre måde – en klassisk resultat fra 1960erne viser, at det er nok at teste alle *binære* sekvenser (dvs, sekvenser af 0er og 1er) af længde n. Det reducerer det antal tests, der skal laves fra $n!$ til 2^n, som betyder meget. Det er også vist, at man ikke kan nøjes med at teste færre input. Benyt dig derfor af funktionen `all_outputs` når du skal implementere `is_sorting`.
- De to funktioner med navn `to_program` har som formål at give en implementering af et sorteringsnetværk i Python. De returnerer hver en liste af `str`. Ideen er, at hvis man skriver den liste ud, en element i hver linje, så får man et kort program, der vil påvirke en liste på den samme måde, som comparatoren/netværk. Sådan et program skal vide, hvilken variabel indeholder den liste, der skal ændres – det er formål med argumentet `var`. For at bytte værdien på to variabler har man brug for en hjælpevariabel – dens navn er argumentet `aux`.
- Det er mening, at der bruges kontraktbaseret programmering til udvikling af de her moduler. Det betyder nemlig, at alle antagelser om inputtet skal dokumenteres tydeligt, sådan at klienter til disse moduler kan være sikker på, at de opfylder dem.

Aflevering

Hver gruppe skal aflevere en zipfil med navn `groupXX.zip` (hvor `XX` er gruppens nummer), som indeholder:

- filer `comparator.py` og `network.py`, som implementerer de kontrakter beskrevet tidligere;
- en fil `reportXX.pdf`, hvor `XX` er gruppens nummer.

Rapporten skal bl.a. beskrive, hvordan modulerne er designet – de valg, der er blevet taget ifm definitionen af datatyperne, samt alle relevante overvejelser ifm implementering af funktioner. Det er vigtigt at inkludere eksempler og forklare, hvordan modulerne er blevet testede. Rapporten skal også inkludere kildekode for de to moduler som bilag. Grupper skal vælge en fasekoordinator, som skal identificeres tydeligt på rapportens forside.

Vigtige pointer.

- Hvis instruktioner om formattet til aflevering ikke følges, kan projektet blive afvist.
- Rapporten er basis for evalueringen.
- I takt med kursets formål er det mening at alle funktionalitet implementeres af gruppen. Det betyder at det ikke er tilladt at importere nogen af Pythons biblioteker, undtaget `functools` og `dataclasses`.