# 1 Introduktion

I hver af foelgende opgaver og implementationer har jeg brugt C#. Jeg har parset input dataen til variablen `string mapString`, *uden* den foerste linie (n-linien). `mapString` antages at eksistere i kode eksemplerne. I opgaverne B, C og D er denne streng yderligere parset til variablen `List<char[]> map`, altsaa en todimensionel liste som repraesenterer kortet, og hvor et koordinat tilgaas ved `map[y][x]`, da de indre lister i `map` repraesenterer raekker i stedet for kolonner, grundet simplere parsing.

Parsingen af data'en er ikke medregnet i algoritme analyserne.

# 2 Antal skatte

## 2.1 Beskrivelse

Jeg itererer hver karakter i `mapString`, og for hver af dem som er lig karakteren "$" laegger jeg 1 til variablen `int amount`, som indeholder svaret naar loekken er koert.

## 2.2 Implementation og analyse

Bemaerk at `mapString` her er en streng *uden* linieskift karaktere, altsaa en $n^2$ lang streng.

```
1 var amount = 0;                          // c
2 foreach (var ch in mapString)            // n
3     if (ch == '$')                       // n^2
4         amount++;                        // n^2
5
6 Console.WriteLine(amount);               // c
```

Hver linies antal eksekveringer er noteret som kommentare. Samlet har jeg foelgende udtryk for tidskompleksiteten som funktion af kortets stoerrelse $n$:

$$2n^2 + n + 2c$$

Naar jeg smider konstanten vaek ender jeg op med $n^2$ som den hoejst betydende faktor og dermed er worstcase koeretiden $\theta(n^2)$. Det giver intuitivt mening, da hvert felt i kortet tjekkes praecis een gang, og kortets antal felter er $n^2$, hvilket ogsaa viser at algoritmen er korrekt, da alle mulige skatte itereres.

# 3 Tilgængelige skatte

## 3.1 Beskrivelse

Til opgaven bruger jeg BFS[1] algoritmen. Kortet opfattes som vaerende en graf i sig selv, hvor hvert koordinat er en knude og hver nabo til en knude som ikke er vaeg (`#`), indikerer en kant til foerende naboen.

Jeg bruger en koe (`Queue<Tile> frontier` i C#) som datastruktur for de felter/koordinater (`Tile`s), der skal undersoeges, og jeg markerer et felt udforsket ved at aendre det til en vaeg i kortet (`map`). Paa den maade vil et felt aldrig blive tilfoejet til `frontier` mere end een gang.

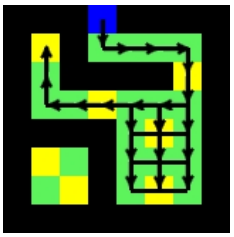
Jeg initialiserer en variabel `found` (linie 16) til at gemme antallet af fundne skatte.

[1] Breadth First Search

Jeg laegger indgangsfeltet i koeen, og saetter en `while` loekke igang (linie 17), som koerer indtil `frontier` er tom. Hver iteration i loekken "Dequeue"er naeste knude (`Tile`) i koeen (foerste gang dermed indgangsfeltet) og laegger dens naboer ind i `frontier`, hvis det er et gyldigt felt (dvs. ikke-vaeg, og inden for kortets rammer). For hver af dets naboer som er en skat, laegges 1 til antal fundne skatte (linie 45).

Dermed bliver hver knude tjekket een gang, men kun de knuder som kan naas fra indgangsfeltet, da knuder som er spaerret inde af vaegge aldrig kan blive tilfoejet til frontier. Det kan de ikke fordi de ikke har nogle kanter som foerer ind til de tilgaengelige knuder/felter.

Figur 3.1 viser hvordan algoritmen gaar fra felt til felt, visualiseret ved en pil. Det bemaerkes at der kun er een pil som pejer paa hvert felt ($deg^-(v) = 1$), men der kan vaere fra $0-3$ ud ($deg^+(v) \in \{0,1,2,3\}$). Dermed er grafen over de tilgaengelige skatte et (orienteret) trae som er et sammenhaengskomponent i den samlede graf over kortet.



Figur 1: Sample02 fra E-Judge; blaa: indgang, groen: gulv, gul: skat, sort: vaeg

## 3.2 Implementation

```
1  var entry = new Tile();
2  // find indgangen, antag at der er praecis en (tager foerste)
3  for (var y = 0; y < n; ++y)
4      for (int x = 0; x < n; ++x)
5          if (map[y][x] == 'I')
6          {
7            entry = new Tile(x, y);
8            break;
9          }
10
11 // initialiser en koe over felter som skal undersoeges
12 var frontier = new Queue<Tile>();
13 // tilfoej indgangen som foerste felt der skal undersoeges
14 frontier.Enqueue(entry);
15 // variabel som bruges til at taelle tilgaengelige skatte
16 var found = 0;
17 while (frontier.Count > 0)
18 {
19     // tag foerste element ud af koeen
20     var curr = frontier.Dequeue();
21     // saet feltet som vaerende vaeg for ikke at undersoege det igen
22     map[curr.Y][curr.X] = '#';
23
24     // undersoeg om felter oven, under, til hoejre
25     // og til venstre er gyldige at gaa videre paa
26     ProcessTile(n, curr.X-1, curr.Y, ref found, frontier, map);
27     ProcessTile(n, curr.X+1, curr.Y, ref found, frontier, map);
28     ProcessTile(n, curr.X, curr.Y+1, ref found, frontier, map);
29     ProcessTile(n, curr.X, curr.Y-1, ref found, frontier, map);
30 }
31
32 Console.WriteLine(found);
```

```
33
34 // funktionen ProcessTile som bruges ovenfor
35 void ProcessTile(int n, int px, int py, ref int found,
36                   Queue<Tile> frontier, List<char[]> map)
37 {
38     // tjek om koordinatet er inden for kortets rammer
39     // og at den i saa fald er gyldig at gaa videre paa (ikke er vaeg)
40     if (!(px < 0 || py < 0 || px > (n-1) || py > (n-1)) && map[py][px] != '#')
41     {
42         var p = new Tile(px, py);
43         // tael en op hvis feltet er en skat
44         if (map[py][px] == '$')
45             found++;
46         // tilfoej feltet til koeen over felter der skal gaas videre paa
47         frontier.Enqueue(p);
48         // saet feltet som vaerende ugyldig at tilfoeje til listen
49         // over utjekkede felter (der er netop blevet gjort og maa ikke goeres igen)
50         map[py][px] = '#';
51     }
52 }
```

### 3.3 Analyse

For simplicitet har jeg analyseret koden i grupper af linier:

**Linie 1** $c$

**Linie 3** $n$

**Linie 4-5** $n^2$

**Linie 12-16** $3c$

**Linie 17-29** $n^2$, da *alle* tiles i worst-case tjekkes

**Linie 40** $4n^2$, da den for hvert felt der tjekkes kaldes 4 gange

**Linie 42-50** $n^2$, da praemissen i linie 40 maksimalt vil opfyldes $n^2$ gange grundet "vaeg-tjekket"

Dermed er den samlede worst-case koeretid

$$7n^2 + n + 4c$$

Og derfor i $\theta$ notation: $\theta(n^2)$. Jvf. slides brugt i undervisningern er koeretiden af BFS

BFS har koeretiden $\theta(v + e)$, hvor $v$: knuder, $e$: kanter

## 4 Den bedste indgang

### 4.1 Beskrivelse

Jeg bruger samme soegemetode som i opgave