Hledání Euklidovských konstrukcí

Postrehy

Nemáme dané súradnice bodov A a B ale pomocou operácií rotácie, posunutia a zväčšenia / zmenšienia sa vieme medzi rôznymi polohami týchto bodov ľubovoľne presúvať.

To nám umožňuje si zvoliť súradnice bodov A a B a pracovať s číslami. Zvolíme si teda

```
A = [0, 0]
B = [0, x]
```

Pre nejaké x (konštantu).

Teraz môžeme skúmať rôzne konštrukcie a hľadať takú, ktorá nám rozdelí úsečky AB na dve časti s požadovaným pomerom.

Ak máme dané N a zvolíme si nejaké x, vlastne hľadáme pod S, ktorý má súradnice

```
S = [x / (N+1), 0]
```

Problém v tomto prístupe je so zaokrúhľovaním a porovnávaním reálnych čísel na počítači!

Vyriešiť by sme to mohli vhodnou reprezentáciou bodov. Pri takýchto výpočtoch nám nemôže vzniknúť nič iné, ako odmocniny stupňa 2, pričom celé čísla vieme reprezentovať rovnako (napr. 10 budeme reprezentovať ako 100, pretože 100 = sqrt(10)). Môže sa ale stať, že pri výpočte polohy bodu nám vznikne napr. takýto výraz `sqrt(5) + sqrt(6)`, *??ktorý nevieme dať pod jednu odmocninu??*.

Všetky čísla teda budeme reprezentovať ako zoznamy celých čísel (`int`), kde každé číslo je pod odmocninou stupňa 2, (pričom napr. 10 = sqrt(100)).

Budeme ale musieť implementovať výpočty s číslami v takejto podobe a ich porovnávanie (rovnosť). Túto reprezentáciu čísel nazveme `sqrtint`.

Reprezentácia dat

Všetky čísla budeme reprezentovať pomocou typu `sqrtint`, ktorý sme si definovali

Bod

Bod bude reprezentovaný pomocou class-y `Bod`

```
class Point {
  public sqrtint x;
  public sqrtint y;
  public Point Clone(); // Vytvorí deep kópiu
}
```

Priamka

Priamka bude daná pomocou hodnôt intercept a slope.

```
class Line {
  public sqrtint intercept;
  public sqrtint slope;
}
```

Kružnica

```
class Circle {
  public int center;
  public sqrtint radius;
}
```

Kde center je index bodu v

Stav

Každá konštrukcia (aj čiastočná) bude reprezantovaná pomocou stavu. Stav bude daný bodmi, kružnicami a priamkami.

```
class State {
  public List<Point> points;
  public List<Line> lines;
  public List<Circle> circles;
  public int moves;
  public State Clone();
}
```

Algoritmus

Inicializácia

Algoritmus bude spočívať v prehľadávaní možných stavov. Stavy si rozdelíme podľa počtu krokov. Každé riešenie musí začať takto:

```
(1) p1=AB
(2) k2=A(B)
(3) k3=B(A), {C, D}=k2∩k3
```

To znamená, že počiatočný stav bude mať 4 body, 1 priamku, 2 kružnice a vieme sa doň dostať pomocou 3 krokov. Potom začneme prehľadávať.

Prehľadávanie

Použijeme na to prehľadávanie do šírky. Vytvoríme si frontu. V každom kroku
si vyberieme stav z fronty a vyskúšame všetky možné modifikácie:

1. Pridanie priamky (`Line`)
2. Pridanie kružnice (`Circle`)

To znamená, že musíme vedieť, kde môžeme pridať nové priamky a kde môžeme pridať nové kružnice. Na to si spravíme rozhranie na class-e `State` . Vieme si napríklad spraviť N x N maticu, kde N je počet bodov a do nej si do poľa i,j zapísať 0, ak priamka medzi i-tým a j-tým bodov neexistuje a 1, ak existuje. Podobne vieme riešiť aj kružnice.

Po modifikácii vždy spočítame nové priesečníky (body) a pridáme ich ma koniec listu `state.points`.

Priesečníky

Potrebujeme spočítať všetky priesečníky nového objektu `Line` alebo `Circle` so všetkými (`Line` aj `Circle`) v stave. Môžu teda nastať 3 prípady:

1. `Circle` intersect `Circle`
2. `Circle` intersect `Line` (to je symetrické)
3. `Line` intercest `Line`

Body, ktoré nám vznikli rovno pridávame na koniec zoznamu `points`.

Trackovanie stavu

Teraz potrebujeme zistiť, či sme sa už niekedy predtým do takéhoto stavu nedostali. Stavy, v ktorých sme už boli bude sledovať class-a `StateTracker`, ktorá v sebe bude mať hešovaciu tabuľku a bude mať takéto rozhranie:

- `public void RegisterState(State state)` - zahešuje stav a zaznamená ho
- `public bool StateDiscovered(State state)` - vráti `true` ak takýto stav už nastal; `false` v opačnom prípade.

Hešovacia tabuľka bude amortizovaná, pretože dopredu nevieme, koľko stavov nastane.

Hešovanie stavu

Stav sa dá plne popísať bodmi, ktoré obsahuje a informáciami o kružniciach. Mohli by sme si zostrojiť vektor

```
( points[0].x, points[0].y, points[1].x, points[1].y, ..., lines[0].intercept, lines[0].slope, ..., circles[0
```

A ten zahešovať pomocou skalárneho súčinu. (Ak by rôzna dĺžka vektorov spôsobovala problém, spravíme si pre každú dĺžku jednu priehradku pomocou array a v každej priehradke bude hešovacia tabuľka).

Koniec prehľadávania

Po výpočte nových bodov zvýšime hodnotu `moves` o 1 a skontrolujeme, či niektorý z týchto bodov nemá požadované vlastnosti (viz. Postrehy).

Ak takýto bod vznikol, prehľadávanie skončíme. Inak nový stav **naklonujeme**, pripojíme na koniec fronty a pokračujeme v prehľadávaní (buď pridanie iného objektu alebo ďalší stav).

Dekompozícia programu

Program bude mať takúto štruktúru

```
{
  namespace MainNamespace {
    public class Point { /* ... */ }
    public class Circle { /* ... */ }
    public class Line { /* ... */ }
    public class State { /* ... */ }
    public class StateTracker { /* ... */ }

    public class MainClass {
      static void Main() {
```

```
        const int X = /* value of X */;

        for (int N = 1; N < 20; N++) {
          State state; // Tu bude posledný state
          var tracker = new StateTracker();
          var queue = new Queue<State>();
          var initial = new State();
          // Pridať potrebné objekty do stavu

          while (!queue.Empty) {
            // Do search
          }

          Console.WriteLine("{0}: {1}", N, state.moves);
        }
      }
    }
  }
}
```