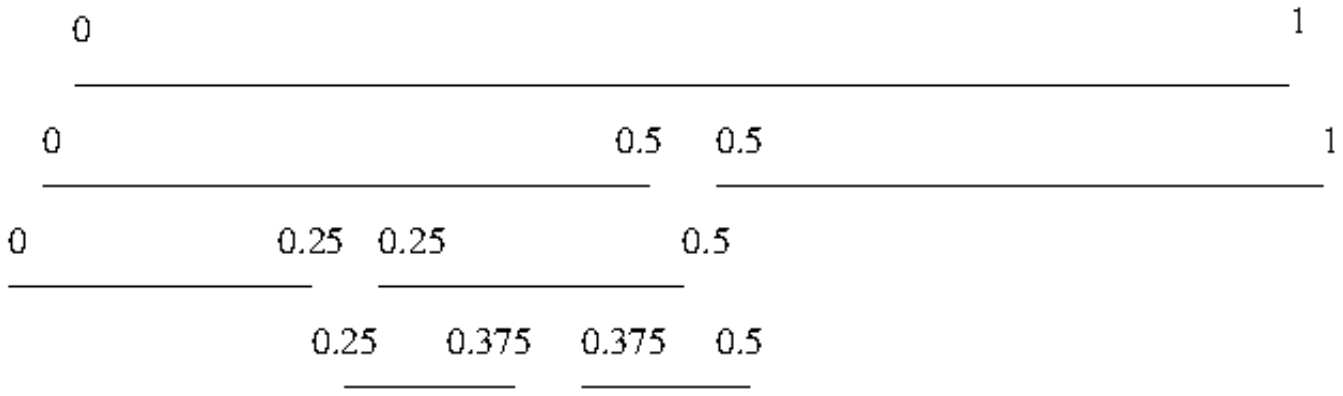
Zadatak: Particija segmenta (a,b)

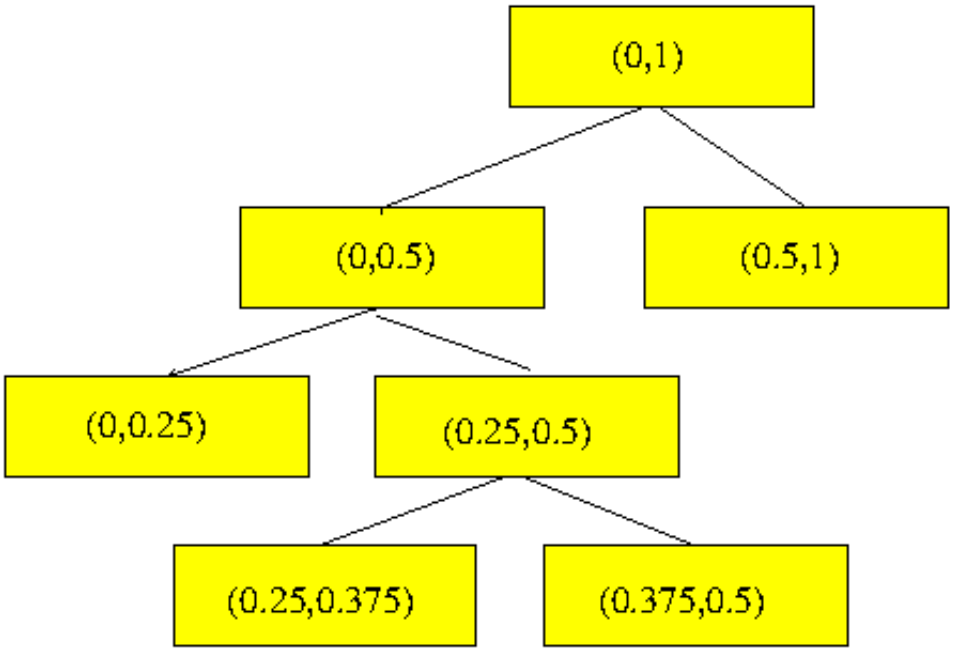
Na zadanom segmentu (a,b) možemo napraviti subdiviziju ako uzastopno polovimo segment. Na primjer, uzmimo segment (0,1):

1. Jednim polovljenjem segmenta dobivamo (0,0.5), (0.5,1)
2. Još jednim polovljenjem prvog segmenta dobivamo (0,0.25), (0.25, 0.5), (0.5,1).
3. Još jednim polovljenjem drugog segmenta dobivamo (0,0.25), (0.25, 0.375), (0.375,0.5), (0.5,1).

Sljedeća slika prikazuje postupak koji se može nastaviti u nedogled proizvodeći subdiviziju sa sve više segmenata.



(0,1) = (0,0.25) , (0.25,0.375), (0.375,0.5), (0.5,1)



Struktura koja pamti sve generirane segmente ima formu binarnog stabla. Profinjeni segment nazivamo roditeljem, a segmente nastale profinjenjem njegovom djecom. Listovi stabla su segmenti bez djece. Listovi stabla čine subdiviziju polaznog segmenta.

Zadatak je implementirati klasu `Grid` koja predstavlja cijelo binarno stablo segmenata i implementirati iterator koji iterira samo kroz segmente listove. Ukupno treba implementirati 3 klase:

1. Klasu `Segment` koja modelira jedan segment i drži koordinate rubnih točaka segmenta.
2. Klasu `Grid` koja predstavlja binarno stablo svih segmenata i nudi iterator kroz segmente listove.
3. Klasu `GridLeafIterator` koja implementira iterator kroz segmente listove.

Klasa Segment. Pored krajeva segmenta ova klasa drži pokazivače na svoju djecu (dva segmenta dobivena vlastitim polovljenjem) te pokazivač na segment roditelj. Ukoliko je segment list, onda su pokazivači na njegovu djecu jedanki `nullptr`. Klasa segment je zadužena za polovljenje segmenta (metoda `refine()`). Ona može i okrupniti mrežu eliminiranjem svoje djece (ako su djeca listovi). Kako klasa `Segment` mora kontrolirati vijek trajanja svoje djece ona neće držati obične pokazivače na djecu već iz zamata u `unique_ptr`. Izgled klase Segment dan je ovdje:

```
class Segment{
    public:
        // Konstruktor. Postavlja lijevi i desni kraj intervala i pokazivač na
roditelja.
        Segment(double left, double right, Segment * parent) : mpt_l(left),
mpt_r(right), mparent(parent) {}
        // Da li je segment list (nema djece) ?
        bool isLeaf() const{ return mseg_l.get() == nullptr; }
        // Profini segment. Ako već ima djecu ne radi ništa.
        void refine();
        // Okrupni segment, tj. eliminiraj djecu segmenta. Time segment postaje
        // list. Segment mora imati djecu koja su listovi, u suprotnom izbaciti
izuzetak.
        void coarse();
        // Lijevi podsegment ili nullptr.
        Segment * getLeft()  { return mseg_l.get(); }
        // Desni podsegment ili nullptr.
        Segment * getRight() { return mseg_r.get(); }
        // Ispiši segment.
        void print(){std::cout << "(" << mpt_l <<","<<mpt_r<<") ";}
        // Vrati pokazivač na roditelja.
        Segment * parent() { return mparent; }
    private:
      // lijevi kraj segmenta
      double mpt_l;
      // desni kraj segmenta
      double mpt_r;
      // djeca segementa
      std::unique_ptr<Segment> mseg_l;  // lijevi podsegment
      std::unique_ptr<Segment> mseg_r;  // desni podsegment
      // segment roditelj
      Segment * mparent;
      friend class GridLeafIterator;
};
```

Klasa Grid predstavlja binarno stablo svih segmenata i nudi iterator po segmentima listovima. Pored toga nudi par metoda koje testiraju iteratore. Tipično, `Grid` će držati samo root-segment, tj. segment koji je roditelj svih drugih segmenata. Najvažnija funkcija `Grid` klase je što nudi `begin()` i `end()` metode za iteriranje kroz segmente listove. (Radi jednostavnosti nudi se samo nekonstantan iterator te stoga metode koje ga koriste nisu konstantne.)

```
class Grid {
    public:
       using iterator = GridLeafIterator;

       Grid(double a, double b): root(a,b, nullptr) {}

       iterator begin();
       iterator end();

       // Profini svaki segment list u trenutnoj mreži.
```

```
        void uniform_refine();
        // Ispiši sve segmente listove mreže.
        void print();
        // Broj segmenata listova u mreži.
        int  nOfSegments();
        // Profini segment list koji je noSeg po redu.
        // Ako je noSeg >=  nOfSegments() izbaci izuzetak
        void refine_selected(int noSeg);
        // Eliminiraj  segment list koji je noSeg po redu ukoliko
        // su oba djeteta njegovog roditelja listovi.
        void coarse_selected(int noSeg);
    private:
        Segment root;
};
```

Napomena. Poredak segmenata listova određuje iterator.

Kao primjer upotrebe iteratora pokazujemo implementaciju jedne metode iz klase `Grid`. Treba voditi računa o tome da profinjenje i okrupnjenje obezvrijeđuju iteratore.

```
int Grid::nOfSegments(){  // Ne može biti konstantna jer nemamo konstantan
iterator
    int n = 0;
    auto it = begin();
    auto end_it = end();
    for( ; it != end_it; ++it) n++;
    return n;
}
```

Klasa GridLeafIterator. Minimalna implementacija ove klase bi nudila sljedeće metode.

```
// Iterator koji iterira samo po djeci listovima mreže (segmentima koji nemaju
djecu).
class GridLeafIterator{
    public:
        GridLeafIterator(Segment* root);
        // operator inkrementiranja (dovoljna je prefiks verzija)
        // operatori dohvata - dereferenciranje i operator strelica
    private:
        Segment *  mRoot;
        // implementacijski detalji
        // ...
        friend
        bool operator==(GridLeafIterator const & lhs, GridLeafIterator const &
rhs);
        friend
        bool operator!=(GridLeafIterator const & lhs, GridLeafIterator const &
rhs);
};
```

Implementacija klase GridLeafIterator. Implementacija se može bazirati na nerekurzivnom *in-order* obilasku binarnog stabla. Implementacija nerekurzivnog obilaska najjednostavnija je pomoću stoga. U operatoru `++` obilazak se zaustavi uvijek kada se naiđe na segment list.

Sljedeći glavni program mora davati odgovarajući ispis:

```
#include "BT.h"
#include <iostream>

int main()
{
    Grid grid(0.0,1.0);
    grid.print();
```

```
    std::cout<< std::endl;

    grid.uniform_refine();
    grid.print(); std::cout<< std::endl;

    grid.uniform_refine();
    grid.print(); std::cout<< std::endl;
    std::cout << "no of segments = " << grid.nOfSegments() << std::endl;

    grid.refine_selected(2);
    std::cout << "refine segment 2\n";
    grid.print(); std::cout<< std::endl;
    std::cout << "no of segments = " << grid.nOfSegments() << std::endl;

    grid.coarse_selected(1);
    std::cout << "eliminiraj segment 1\n";
    grid.print(); std::cout<< std::endl;
    std::cout << "no of segments = " << grid.nOfSegments() << std::endl;

    return 0;
}
```