Diskrétne Geometrické Štruktúry

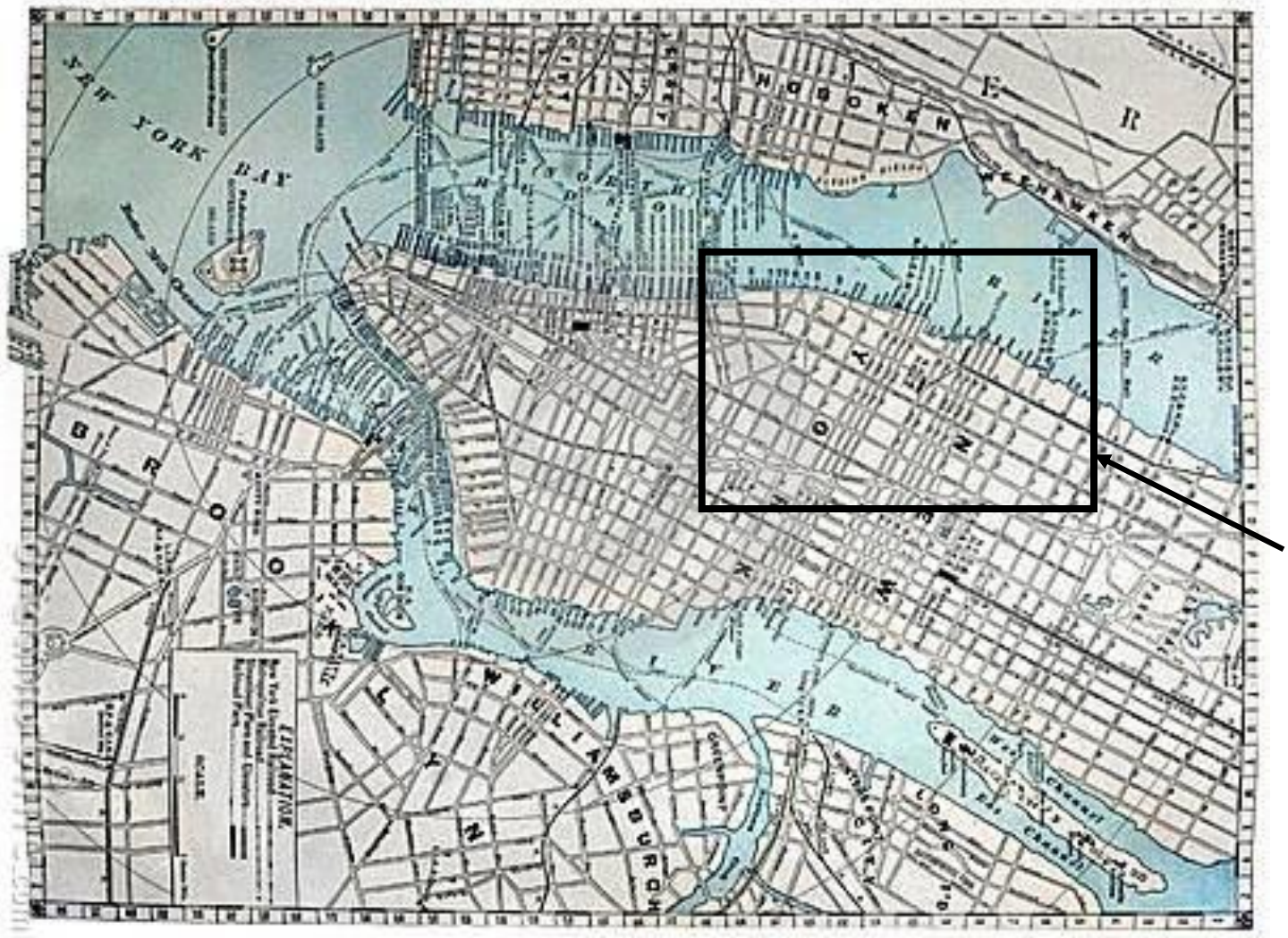
1. Intervalové, segmentové, range stromy

Martin Samuelčík
samuelcik@sccg.sk, www.sccg.sk/~samuelcik, I4

Oknové a bodové požiadavky

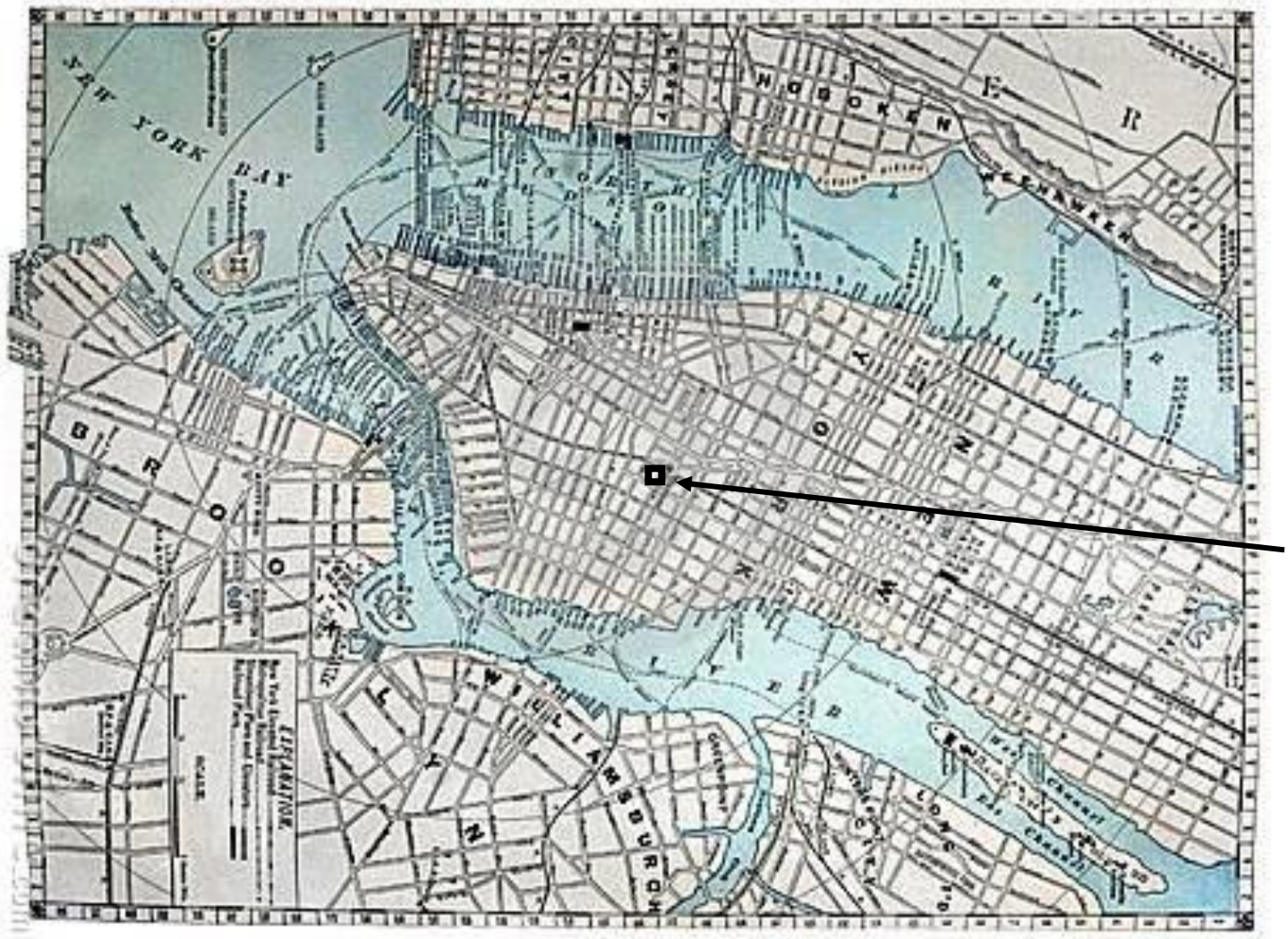
- Pre dané body, nájdi z nich také, ktoré patria danému d-rozmernému intervalu
- Pre dané d-rozmerné intervaly, nájdi z nich také, ktoré obsahujú daný bod
- Pre dané d-rozmerné intervaly, nájdi z nich také, ktoré sa pretínajú s daným intervalom
- d-rozmerný interval: interval, obdlžnik, kváder, ...
- Príklad pre 1D:
  - Vstup: množina n intervalov, reálna hodnota q
  - Výstup: k intervalov obsahujúcich hodnotu q

Motivácia



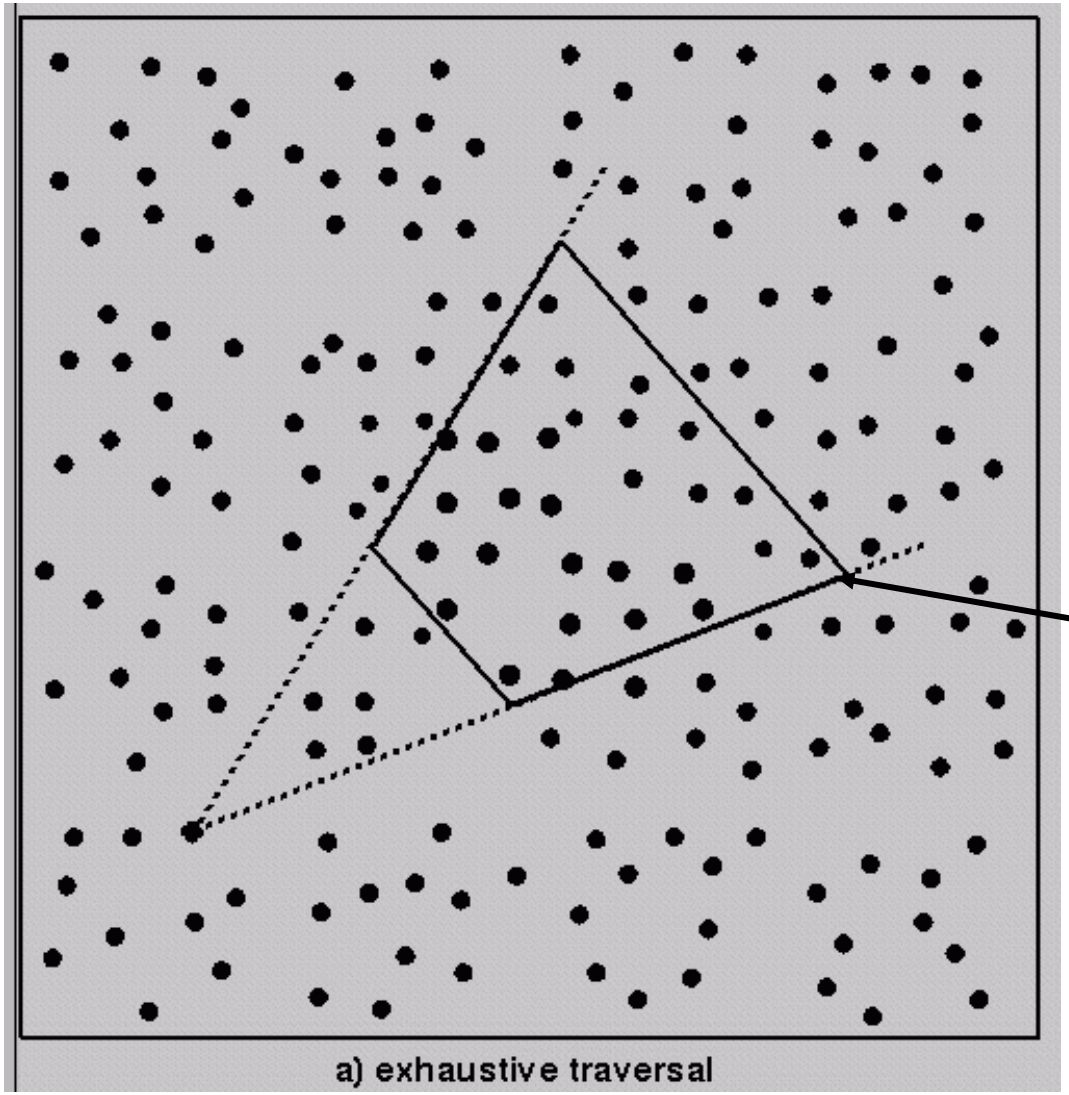
Ktoré budovy sú vo vyznačenom regióne?

Motivácia



Ktorá budova je pod daným bodom?

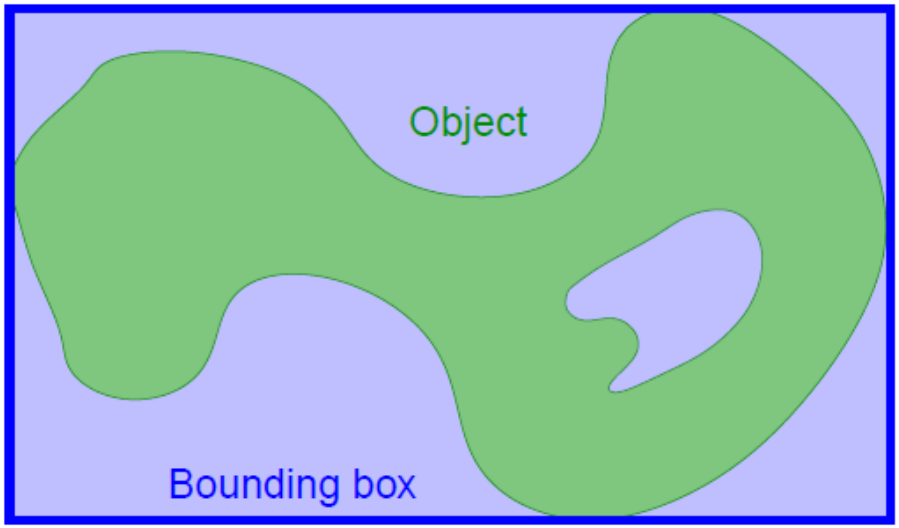
Motivácia

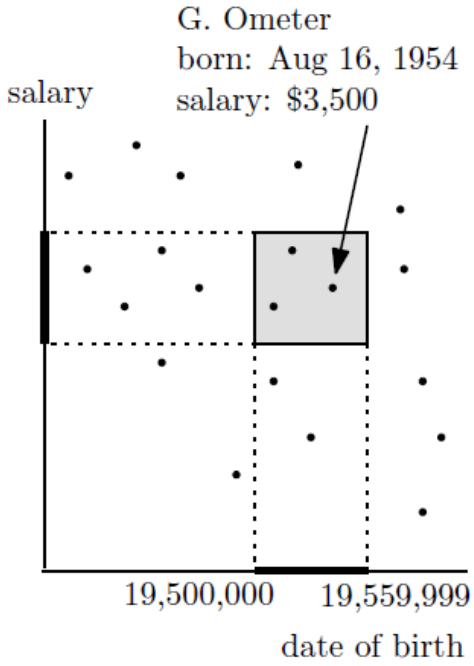


Ktoré objekty sú v pohľadovom objeme?

Motivácia

- V grafike, databázach, GIS systémoch sú objekty aproximované ohraničujúcimi viacrozmernými intervalmi (AABB)
- Naprv sa problém rieši pre AABB a až potom pre samotný objekt





Intervalový strom

- Binárny strom obsahujúci intervaly
- Zostrojený pre nájdenie intervalov obsahujúci danú hodnotu
- Vstup: množina S uzavretých jednorozmerných intervalov, reálna hodnota x_q
- Výstup: všetky intervaly $I \in S$ také, že $x_q \in I$
- $S = \{[l_i, r_i] \text{ pre } i = 1, \ldots, n\}$

Vrcholy intervalového stromu

- X – hodnota, ktorou bola rozdelená množina intervalov v tomto vrchole
- M_l – usporiadaná množina intervalov, ktoré obsahovali X, usporiadaná vzostupne podľa ľavých koncových bodov intervalov
- M_r – usporiadaná množina intervalov, ktoré obsahovali X, usporiadaná zostupne podľa pravých koncových bodov
- Smerník na ľavého syna, obsahujúceho intervaly s hodnotami menšími ako X
- Smerník na pravého syna, obsahujúceho intervaly s hodnotami väčšími ako X

Príklad

$M_l = (s_4, s_6, s_1)$
$M_r = (s_1, s_4, s_6)$

Interval tree on s_3 and s_5

Interval tree on s_2 and s_7

Konštrukcia stromu

- Rekurzívna konštrukcia
- Výber rozdeľovacej hodnoty X
  - Pre vyvážený strom – medián krajných hodnôt intervalov
  - Medián – pre statickú množinu intervalov, bez insert/delete
- Pri vytváraní vrcholu stromu z množiny intervalov S sa najprv nájde X, potom sa rozdelí S na 3 množiny podľa toho, či intervaly obsahujú X, či sú hodnoty v intervale menšia ako X alebo väčšie ako X

Konštrukcia stromu

```
struct Interval
{
        float left;
        float right;
}
```

```
struct IntervalTree
{
        IntervalTreeNode* root;
}
```

```
struct IntervalTreeNode
{
        float x;
        vector<Interval*> Ml;
        vector<Interval*> Mr;
        IntervalTreeNode * left;
        IntervalTreeNode * right;
}
```

```
LeftSegments(x, S)
{
        list<Interval*> result;
        Interval* I;
        for (each I in S)
        {
                if (I->right < x)
                        result.add(I);
        }
        return result;
}
```

```
HitSegments(x, S)
{
        list<Interval*> result;
        Interval* I;
        for (each I in S)
        {
                if (I->left <= x && x <= I->right)
                        result.add(I);
        }
        return result;
}
```

```
IntervalTreeConstruct(S)
{
        T = new IntervalTree;
        T = IntervalTreeNodeConstruct(S);
        return T;
}
```

```
IntervalTreeNodeConstruct(S)
{
        if (Size(S) == 0) return NULL;
        v = new IntervalTreeNode;
        v->x = Median(S);
        Sx = HitSegments(v->x, S);
        L = LeftSegments(v->x, S);
        R = RightSegments(v->x, S);
        v->Ml = SortLeftEndPoints(Sx);
        v->Mr = SortRightEndPoints(Sx)
        v->left = IntervalTreeNodeConstruct(L);
        v->right = IntervalTreeNodeConstruct(R);
        return v;
}
```

Vlastnosti

- Každý interval je uložený práve v jednom vrchole stromu
- Počet vrcholov stromu je O(n)
- Suma veľkostí zoznamov v M_l a M_r je O(n)
- Pri použití mediánu pri delení je výška stromu O(log(n))
- Pre n intervalov, intervalový strom sa dá vytvoriť v čase O(n.log(n)) a zaberie O(n) pamäte

Vyhľadávanie

- Nájdi všetky intervaly $I \in S$ také, že $x_q \in I$
- Rekurzívny prechod
- Pri použití mediánu, pre množinu n intervalov, požiadavka na vyhľadanie vráti k intervalov v čase O(k + log(n))

```
IntervalStabbing(IntervalTreeNode* v, float xq)
{
	list<Interval*> D;
	if (xq < v->x)
	{
		Interval* F = v->Ml.first;
		while (F != NULL && F->left <= xq)
		{
			D.insert(F);
			F = v->Ml.next;
		}
		D1 = IntervalStabbing(v->left, xq);
		D.add(D1);
	}
	else
	{
		Interval* F = v->Mr.first;
		while (F != NULL && F->right >= xq)
		{
			D.insert(Seg(F));
			F = v->Mr.next;
		}
		D2 = IntervalStabbing(v->right, xq);
		D.add(D2);
	}

	return D;
}
```

Prekrývanie intervalov

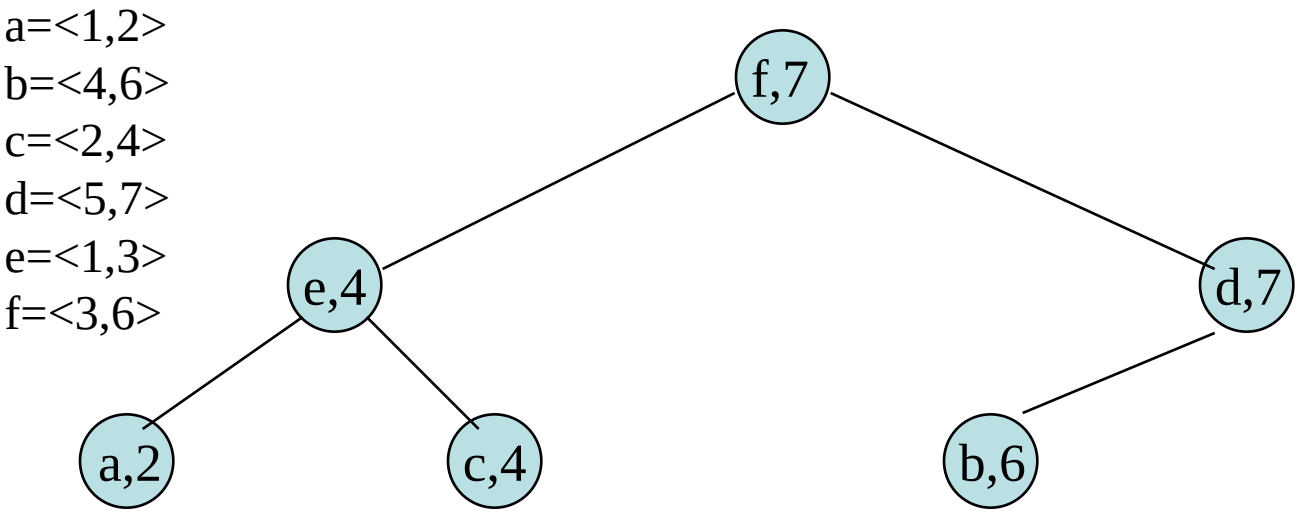
- Pre daný interval $I=[l,r]$, nájdi v množine S intervaly, ktoré majú prienik s I
- Rekurzívne prehladávanie
  - Ak $v->x \in I$, potom všetky intervaly uložené vo $v->M_l$ majú prienik s I, ďalej treba prehladať podstromy $v->left$ aj $v->right$
  - Ak $v->x < l$, potom intervaly z $v->M_r$ pre ktoré $r_i >= l$ majú prienik s I a treba prehladať podstrom $v->right$
  - Ak $v->x > r$, potom intervaly z $v->M_l$ pre ktoré $l_i <= r$ majú prienik s I a treba prehladať podstrom $v->left$
- Zložitosť O(k + log(n)), k je počet nájdených intervalov

Iný intervalový strom

- Vráti iba jeden interval z množiny *S* ktorý ma prienik s daným intervalom
- V každom vrchole *v* je uložený 1 interval z *S* (*v->int*) a maximálna hodnota z koncových hodnôt intervalov z podstromu s koreňom *v* (*v->max*)
- Pridané usporiadanie na intervaloch, napr. $[l_i, r_i] < [l_j, r_j] \iff l_i < l_j \mid (l_i = l_j \,\&\, r_i < r_j)$
- Vytvorený binárny prehľadávací strom (červeno-čierny strom)

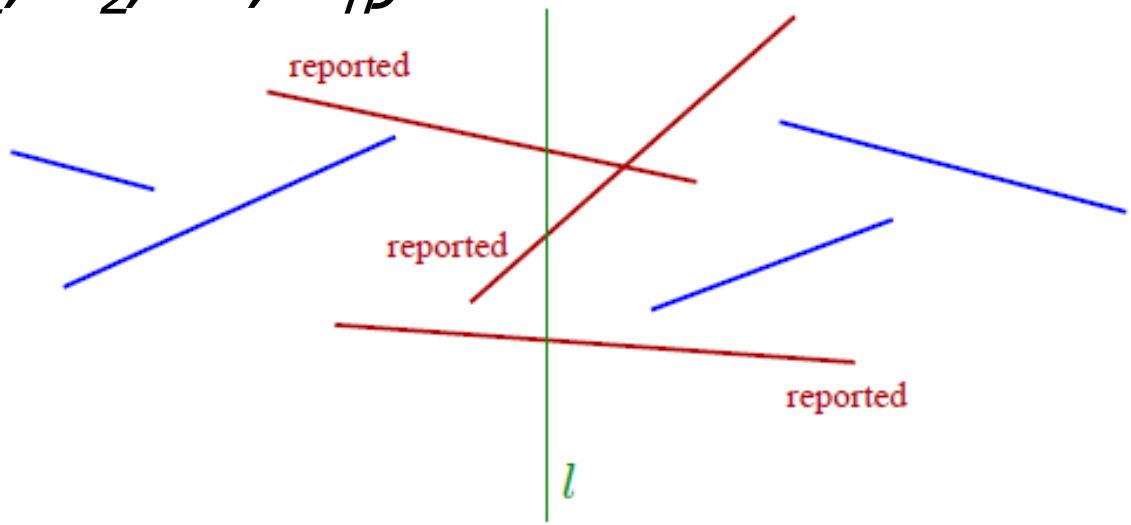
Iný intervalový strom

- Rekurzívne prehľadávanie pre interval *I=[l,r]*
  - Ak *v->int* má prienik s *I*, vráť *v->int*
  - Inak ak *v->left->max >= l*, prehľadaj *v->left*
  - Inak prehľadaj *v->right*



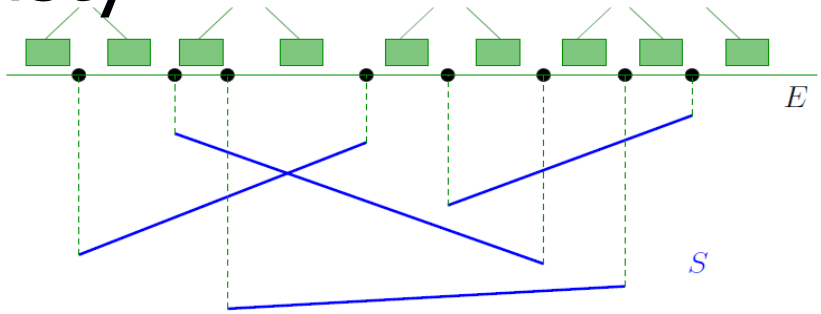
Segmentové (úsečkové) stromy

- Vyhľadanie všetkých úsečiek z danej množiny, ktoré sa pretínajú s danou zvislou priamkou
- Vstup: množina úsečiek S v rovine, zvislá priamka l
- Výstup: všetky úsečky $s \in S$, ktoré pretínajú l
- $S = \{s_1, s_2, \ldots, s_n\}$



Prehľadávací strom

- Pomocná štruktúra
- Usporiadame x-ové súradnice koncových bodov úsečiek z S, $E = \{e_1, e_2, \ldots, e_{2n}\}$
- Predpokladáme všeobecnú pozíciu
- Rozdelíme E na atomické intervaly $[-\infty, e_1]$, $\ldots$, $[e_{2n-1}, e_{2n}]$, $[e_{2n}, \infty]$
- Atomické intervaly budú listy prehľadávacieho stromu



Prehľadávací strom

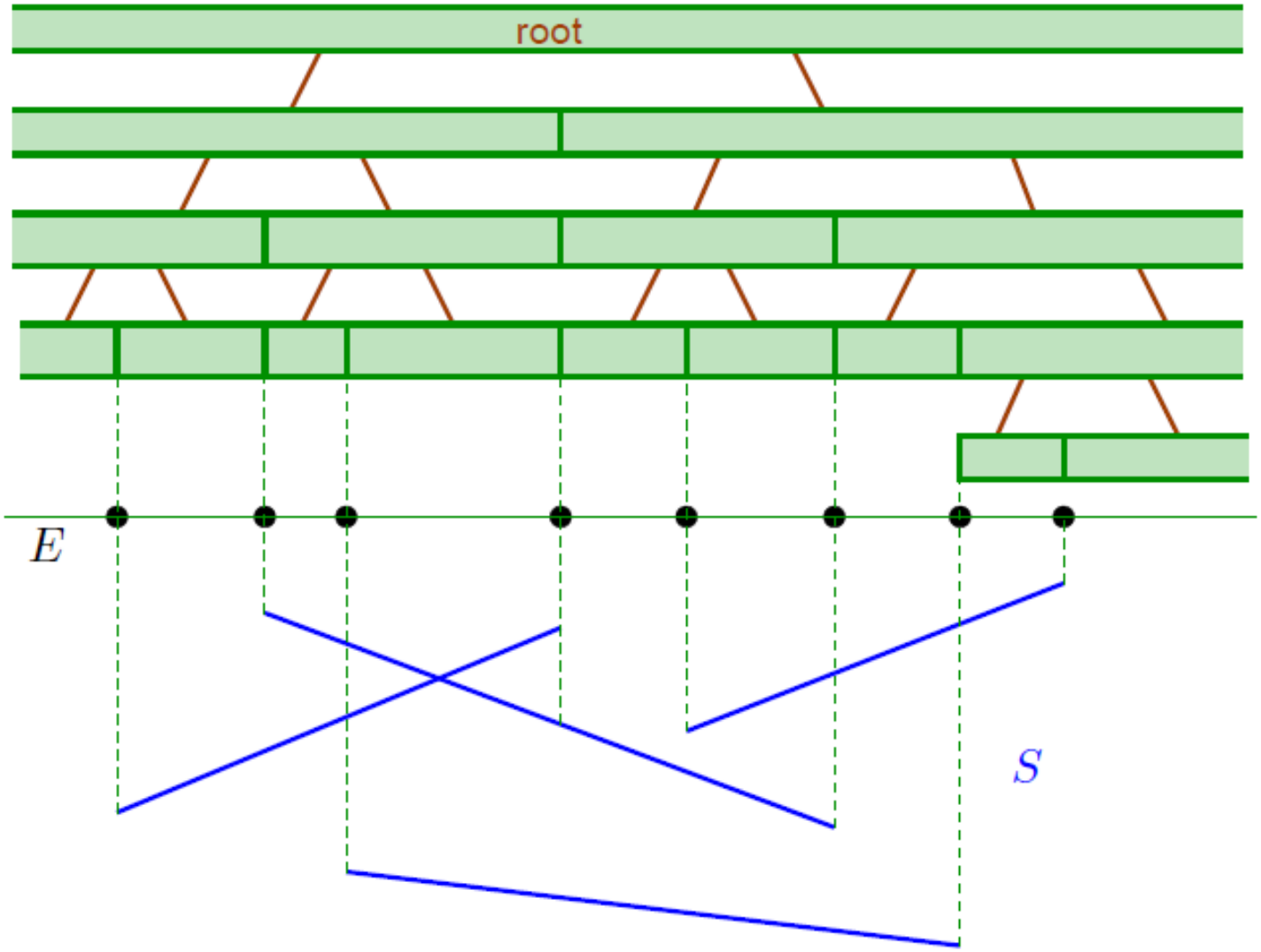
- Vytvoríme vyvážený binárny strom obsahujúci intervaly, v koreni celá reálna os
  - V každom kroku rozdelíme množinu atomických intervalov na dve skoro rovnaké polovice, jedna ide do ľavého podstromu, druhá do pravého

```
struct SegmentTreeNode
{
        float istart;
        float iend;
        List L;
        IntervalTreeNode* left;
        IntervalTreeNode* right;
}
```

```
struct SegmentTree
{
        SegmentTreeNode* root;
}
```

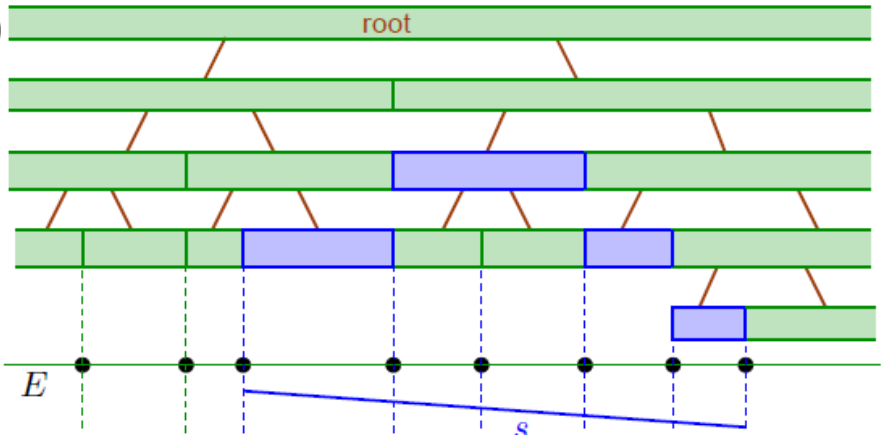
```
SearchTreeNodeConstruct (E)
{
        if (|E| == 0) return NULL;
        v = new SegmentTreeNode;
        n = |E|;m = n / 2;
        (L, R) = Split(E, m);
        v->istart = E.get(1);
        v->iend = E.get(n);
        v->left = SearchTreeNodeConstruct(L);
        v->right = SearchTreeNodeConstruct(R);
        return v;
}
```

Prehľadávací strom



Segmentový strom

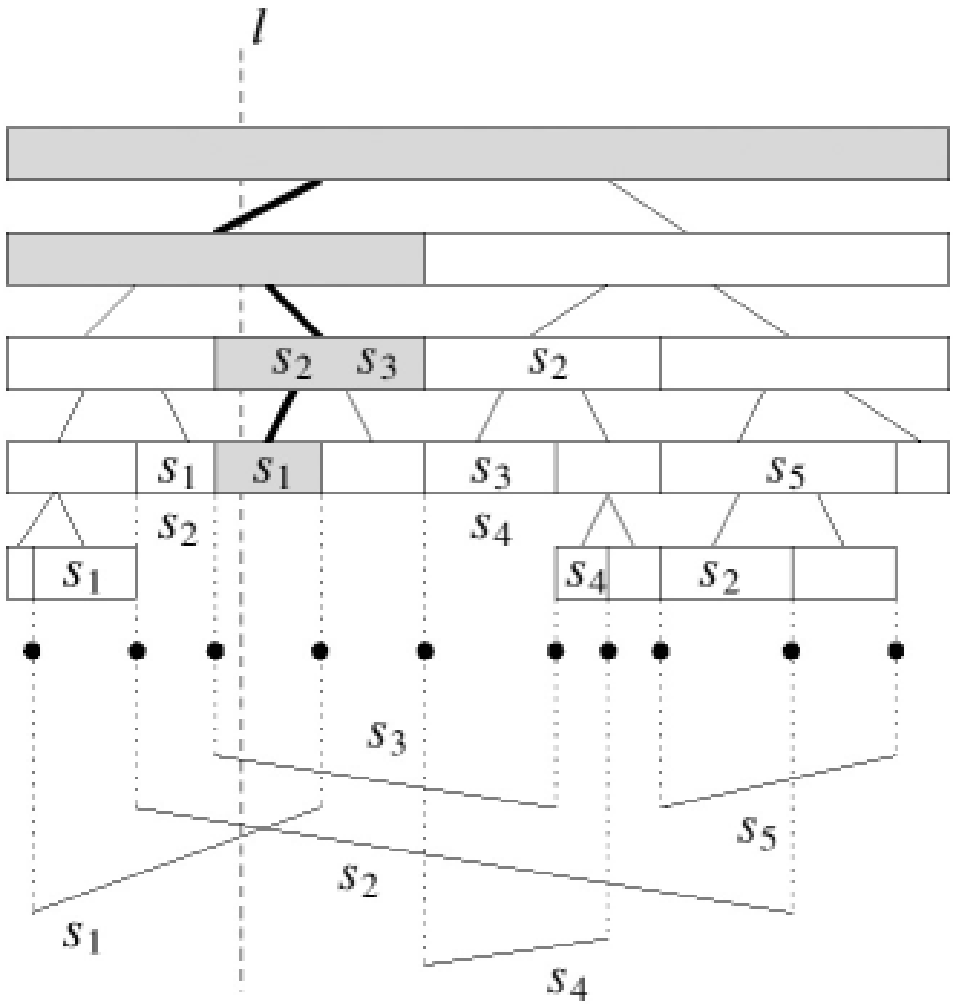
- Máme malé (atomické) intervaly uložené v stromovej štruktúre = prehľadávací strom
- Naplníme tento strom príslušnosťou ku úsečkám tak, aby každá úsečka bola pokrytá čo najmenším počtom intervalov (vrcholov) prehľadávacieho stromu
- Vytvorenie: O(n.log(n))
- Pamät: O(n.log(n))



Vytvorenie segmentového stromu

```
InsertSegment(v, s)
{
        if (v == NULL) return;
        if ([v->istart, v->iend] ∩ s == 0) return;
        if ([v->istart, v->iend] ⊂ s)
        {
                v->L.add(s);
        }
        else
        {
                InsertSegment(v->left, s);
                InsertSegment(v->right, s);
        }
}
```

```
BuildSegmentTree(S)
{
        Sx = S.SortX();
        Sx.ExtendInfinity();
        T = new SegementTree;
        T->root = SearchTreeNodeConstruct(Sx);
        while (S.first != 0)
        {
                InsertSegment(T->root, S.first);
                S.deleteFirst();
        }
}
```



Vyhľadanie

- Pre n úsečiek v rovine, nájdenie všetkých úsečiek ktoré pretínajú danú zvislú priamku má časovú náročnosť O(k + log(n)), kde k je počet nájdených úsečiek

```
StabbingQuery(T, l)
{
        Return StabbingQueryNode(T->root, l);
}
```

```
StabbingQueryNode(v, l)
{
        List L;
        If (v != NULL && v->istart <= l && l <= v->iend])
        {
                L.add(v.L);
                L1 = StabbingQueryNode(v->left, l);
                L2 = StabbingQueryNode(v->right, l);
                L.add(L1);
                L.add(L2);
        }
        return L;
}
```

Vlastnosti

- Usporiadanie koncových bodov O(n.log(n))
- Vytvorenie prehľadávacieho stromu O(n)
- Vloženie jedného segmentu O(log(n))
- Vloženie n segmentov O(n.log(n))
- Pamäťová náročnosť O(n.log(n))

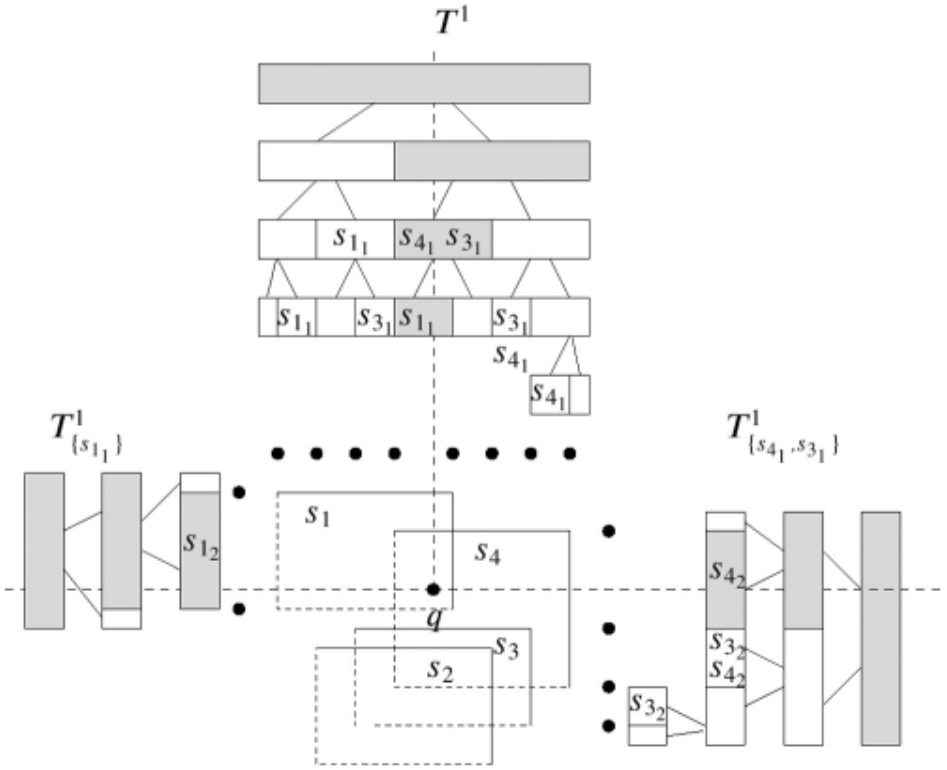
Viacrozmerné segmentové stromy

- Vstup: množina d-rozmerných obdĺžnikov *S* rovnobežných so súradnicovými osami, bod *q* z R^d
- Výstup: množina obdĺžnikov z *S* ktoré obsahujú bod *q*
- V každom vrchole d-rozmerného stromu sa môže nachádzať d-1 rozmerný strom

```
MLSegmentTree(B, d)
{
        S = B.FirstSegmentsExtract;
        T = SegmentTreeConstruct(S);
        T.dim = d;
        if (d > 1)
        {
                N = T.GetAllNodes;
                while (|N| != 0)
                {
                        u = N.First;
                        N.DeleteFirst;
                        L = u->L;
                        List B;
                        while (|L| != 0)
                        {
                                s = L.First;
                                L.DeleteFirst;
                                B.add(s.Box(d − 1));
                        }
                        u->tree = MLSegmentTree(B, d − 1);
                }
        }
        return T;
}
```

Viacrozmerné segmentové stromy

- Čas zostrojenia: O(n.log^d(n))
- Pamäť: O(n.log^d(n))
- Požiadavka: O(k + log^d(n))



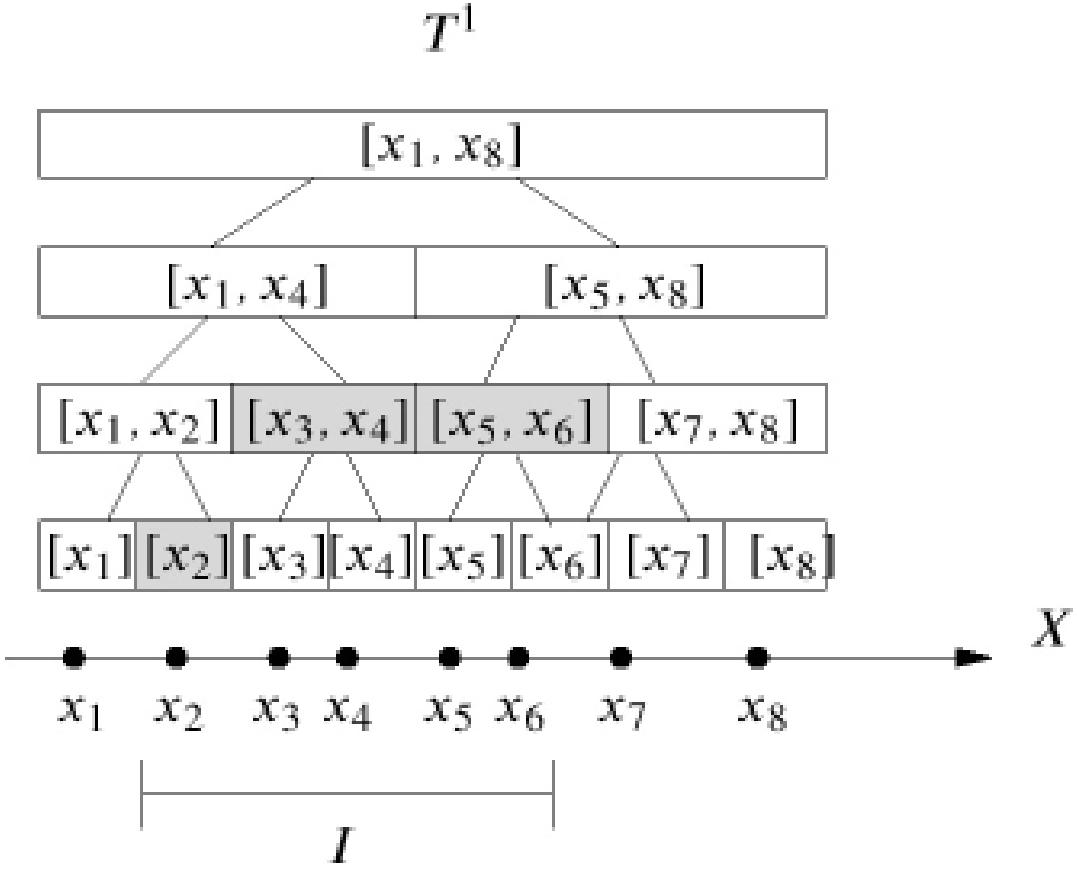

```
MLSegmentTreeQuery(T, q, d)
{
        if (d == 1)
        {
                L = StabbingQuery(T, q);
                return L;
        }
        else
        {
                List A;
                L = SearchTreeQuery(T, q.First);
                while (|L| != 0)
                {
                        t = (L.First)->tree;
                        B = MLSegmentTreeQuery(t, q.Rest, d − 1);
                        A.ListAdd(B);
                        L.DeleteFirst;
                }
                return A;
        }
}
```

Range strom

- Vstup: množina bodov S v R^d, väčšinou $d \geq 2$
- Požiadavka: d-rozmerný interval B z R^d rovnobežný so súradnicovými osami
- Výstup: Množina bodov z S, ktoré sa nachádzajú v B
- Podobná metóda ako pri segmentových stromoch - vytvorenie prehľadávacieho stromu na základe súradníc bodov
- Pre jednorozmerný prípad – hľadanie bodov vo vnútri intervalu

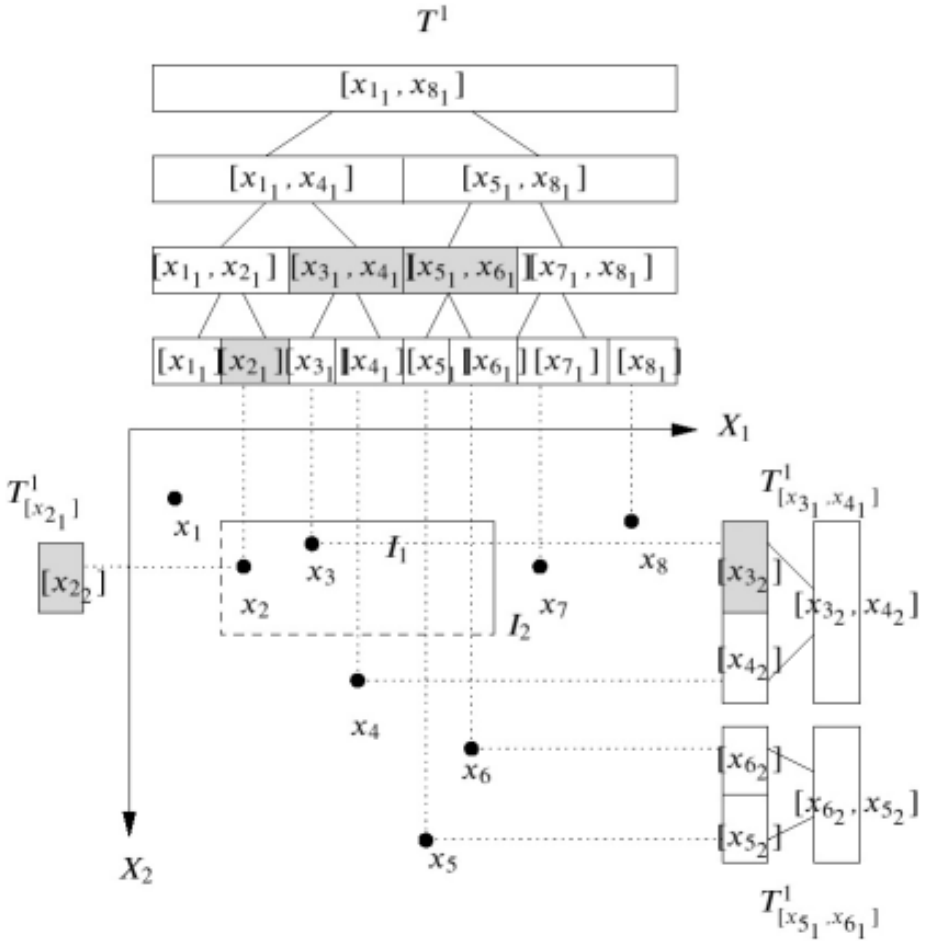
Range strom

- Vytvorenie prehľadávacieho stromu
- Vo vrchole podstromu je uložený interval bodov v listoch



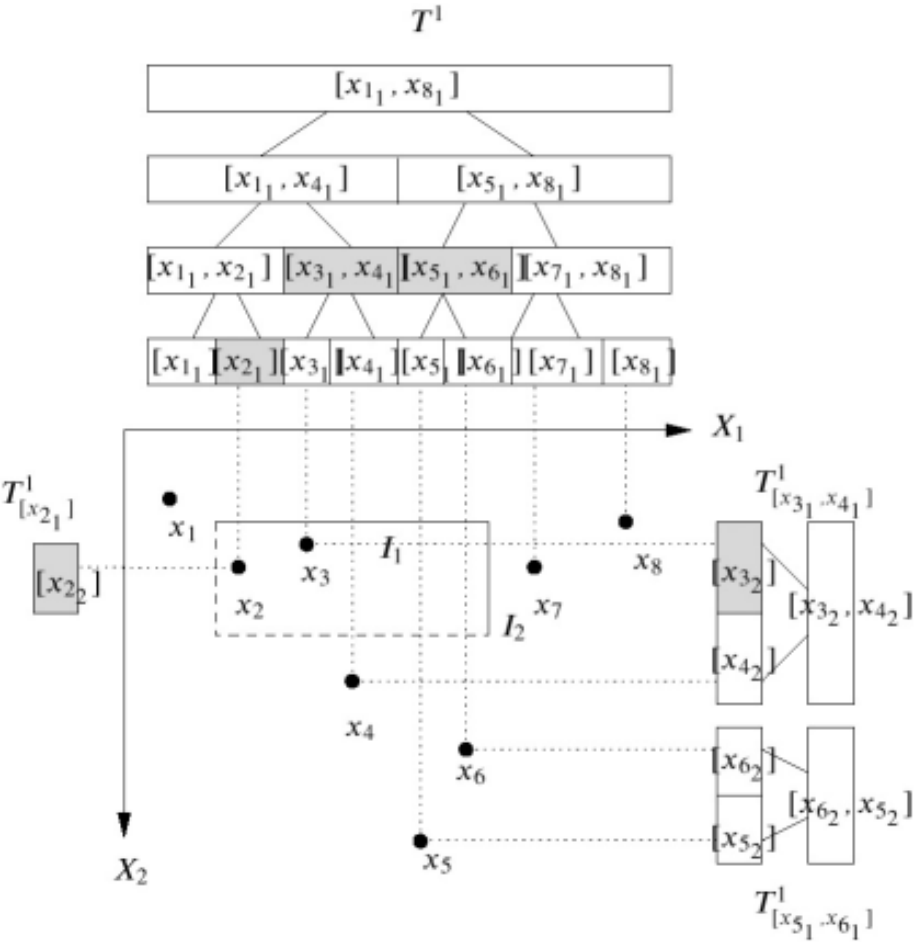
Range strom

- Vrchol d-rozmerného range stromu môže obsahovať d-1 rozmerný range strom




```
RangeTreeConstruct(S, d)
{
        S_f = S.FirstCoordElements();
        S_f.Sort;
        T = SearchTree(S_f);
        T->dim = d;
        if (d > 1)
        {
                N = T->GetAllNodes();
                while (|N| != 0)
                {
                        u = N.PopFirst;
                        L = u->GetAllPoints();
                        List D;
                        while (|L| != 0)
                        {
                                x = L.PopFirst;
                                D.add(x.Point(d − 1));
                        }
                        u->tree = RangeTreeConstruct(D, d − 1);
                }
        }
        return T;
}
```

Range stromy



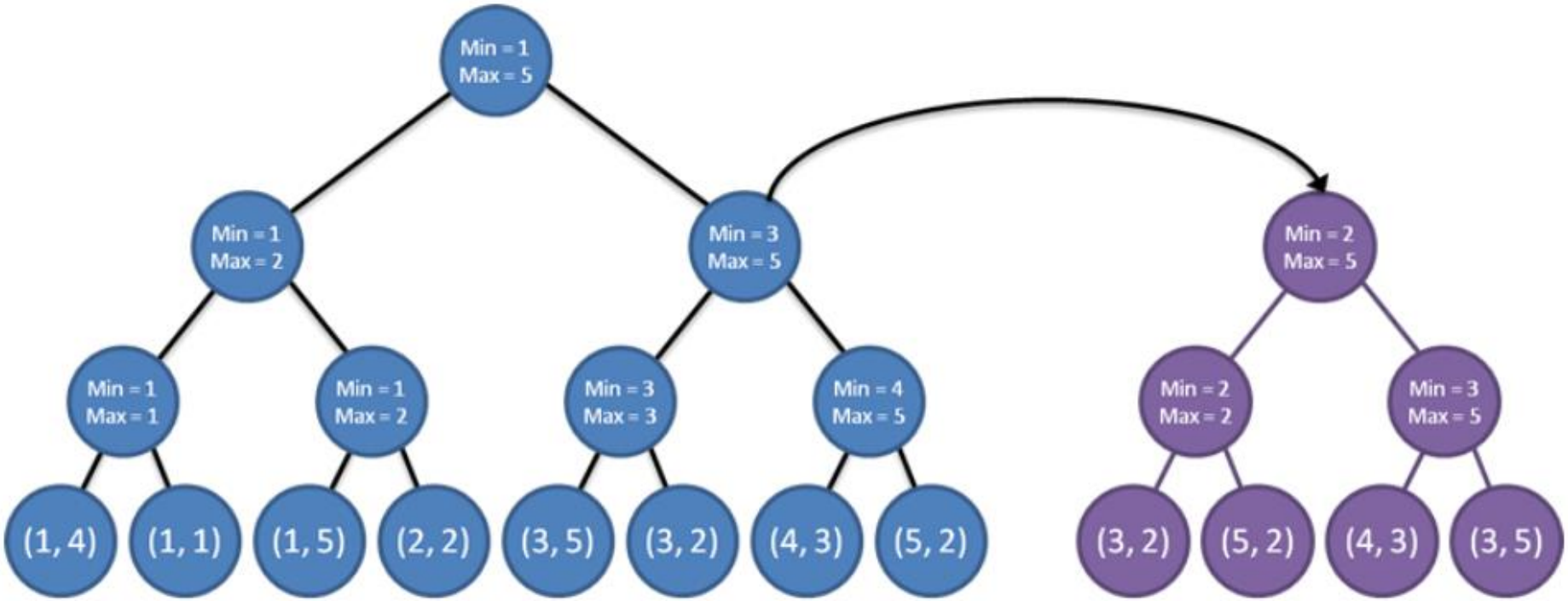

```
RangeTreeNodeQuery(v, B, d)
{
        List result;
        if (d == 0)
           return result;
        (min, max) = GetRangeInDimension(B, d);

        if (v->min > max || v->max < min)
           return result;
        If (v->IsLeaf() && v->point in B)
           result.Add(v->point);
        else if (min <= v->min && max >= v->max)
           result.Add(RangeTreeNodeQuery(v->tree, B, d-1));
        else
        {
           result.Add(RangeTreeNodeQuery(v->left, B, d));
           result.Add(RangeTreeNodeQuery(v->right, B, d));

        }
        return result;
}
```

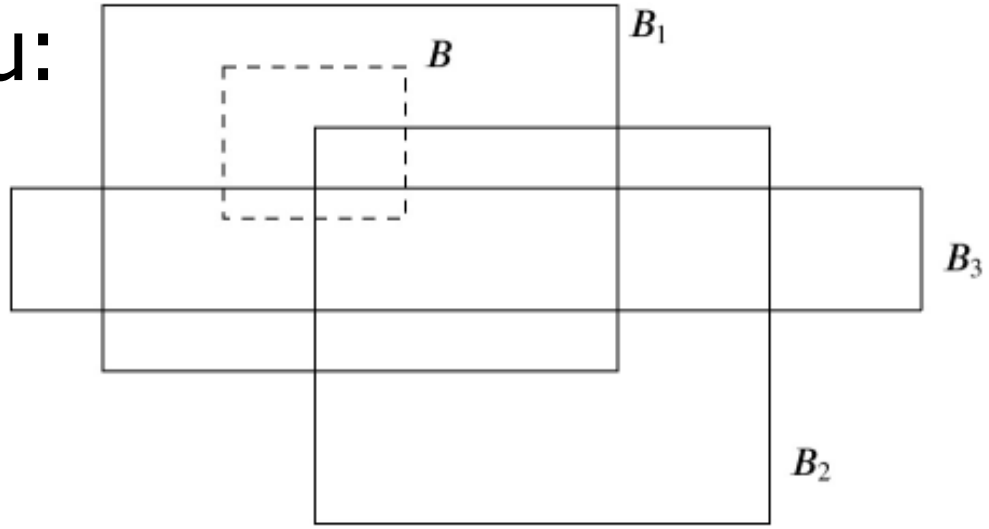
Range stromy

- Konštrukcia: $O(n.\log^{d-1}(n))$
- Pamäť: $O(n.\log^{d-1}(n))$
- Vyhľadanie: $O(k + \log^{d}(n))$



AABB/AABB

- Vstup: množina S 2D obdĺžnikov (intervalov), 2D obdĺžnik B
- Výstup: Intervaly z S ktoré majú prienik s B
- Tri možnosti prieniku:



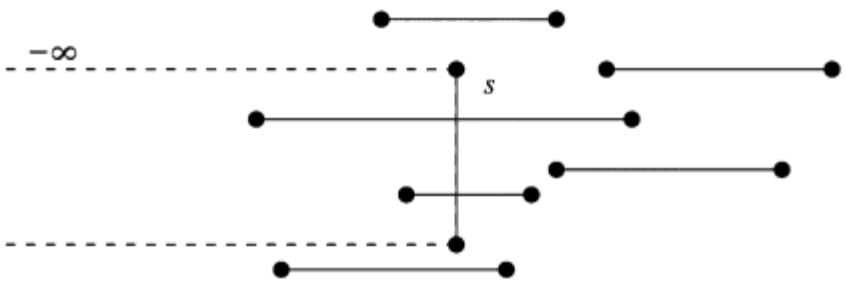
  - B je vnútri B_i
  - Roh B_i leží v B
  - Strana B_i pretína B a žiadny koncový bod B_i nie je v B

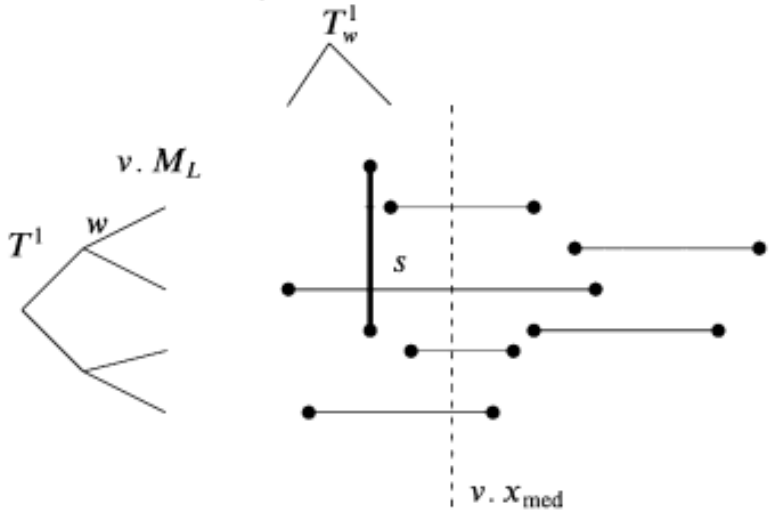
Riešenie

- Prípad 1. – 2D segmentový strom. Hľadáme do ktorých intervalov B_i z S padnú štyri rohy B
  - čas $O(k_1 + \log^2(n))$ a pamäť $O(n + \log^2(n))$
- Prípad 2. – Range strom v 2D. Hľadáme rohy intervalov B_i ktoré padnú do obdĺžnika B
  - čas $O(k_2 + \log^2(n))$ a pamäť $O(n.\log(n))$.
- Prípad 3. – Nová požiadavka:
  - Vstup: Množina S horizontálnych úsečiek v 2D
  - Požiadavka: Vertikálna úsečka s v 2D
  - Výstup: Všetky úsečky pretínajúce s
  - + otočenie o 90º

Prípad 3 - konštrukcia

- Kombinácia intervalového a range stromu
- Najprv vytvoríme intervalový strom z vodorovných hraníc (intervalov)
- M_l a M_r vo vrchole intervalového stromu nahradíme prislušnými 2D range stromami
- Pamäť: O(n.log(n))





Prípad 3 - hľadanie

- V smere x hľadanie horizontálnych hraníc intervalov z S ktoré obsahujú hodnotu x-ovej súradnice s – výsledkom je vrchol intervalového stromu v
- Vo vrchole v sa podľa x a y súradníc s začne prehľadávať M_l resp. M_r (čo sú 2D segmentové stromy) za účelom nájdenia bodov v range strome ktoré sa nachádzajú v príslušnom rozšírenom 2D intervale (do $+\infty$ alebo $-\infty$)
- Čas $O(k_3 + \log^2(n))$

Otázky?