Diskrétne geometrické štruktúry

3.

Martin Florek

florek@sccg.sk

www.sccg.sk/~florek

Oknové a bodové požiadavky

- Či sa bod nachádza v danej množine
- Všetky body v danej množine
- Všetky objekty v danom okne
- Priesečníky okien
- Požiadavka:
  - Rozmer priestoru
  - Prislušná dvojica objektov
  - Typ požiadavky

Intervalové stromy

- Binárny strom pre požiadavku interval/bod
- Vstup: množina *S* uzavretých jednorozmerných intervalov
- Požiadavka: 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 stromu

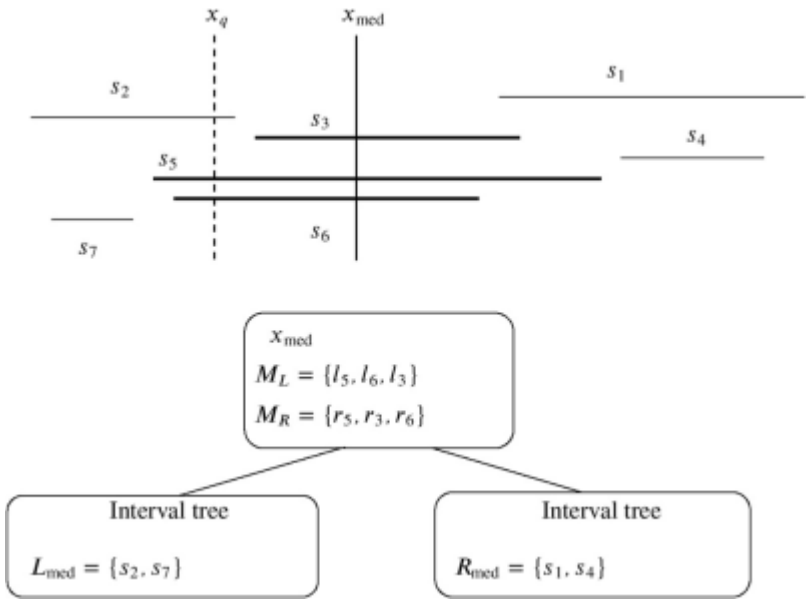
- X_{med} – medián, ktorým bola rozdelená množina intervalov v tomto vrchole
- L_{med}- usporiadaná množina ľavých koncových bodov intervalov, ktoré obsahovali X_{med}
- R_{med}- usporiadaná množina pravých koncových bodov intervalov, ktoré obsahovali X_{med}
- Smerníky na dvoch synov

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

```
struct IntervalTreeNode
{
        float xmed;
        vector<float> Lmed;
        vector<float> Rmed;
        IntervalTreeNode * left;
        IntervalTreeNode * right;
}
```

Vytvorenie stromu

- Pre *n* intervalov, intervalový strom sa dá vytvoriť v čase O(*n*.log *n*) a zaberie O(*n*) pamäte




```
IntervalTreeNodeConstruct(S)
{
        if (S == 0) return NULL;
        v = new IntervalTreeNode;
        v->xmed = Median(S);
        Smed = HitSegments(v->xmed, S);
        L = LeftSegments(v->xmed, S);
        R = RightSegments(v->xmed, S);
        v->Lmed = SortLeftEndPoints(Smed);
        v->Rmed = SortRightEndPoints(Smed)
        v->left = IntervalTreeNodeConstruct(L);
        v->right = IntervalTreeNodeConstruct(R);
        return v;
}
```

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

Vyhľadanie

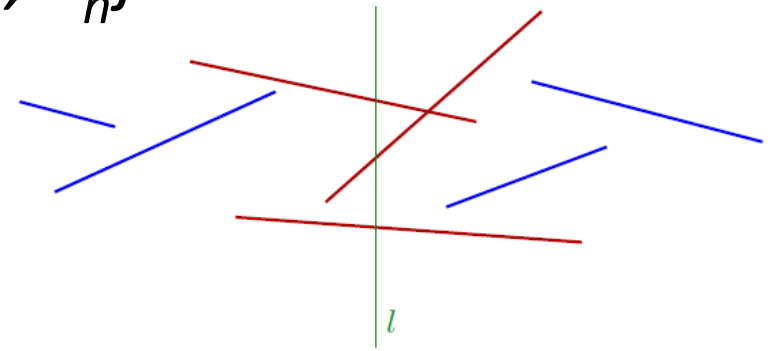
- Pre množinu *n* intervalov, požiadavka na vyhľadanie vráti *k* intervalov v čase O(k + log *n*)

```
IntervalStabbing(v, xq)
{
        D = new List;
        if (xq < v->xmed)
        {
                L = v->Lmed;
                F = L.first;
                while (F != NULL && F < xq)
                {
                        D.insert(Seg(F));
                        F = L.next;
                }
                D1 = IntervalStabbing(v->left, xq);
                D.add(D1);
        }
        else
        {
                R = v->Rmed;
                F = R.first;
                while (F != NULL && F > xq)
                {
                        D.insert(Seg(F));
                        F = R.next;
                }
                D2 = IntervalStabbing(v->right, xq);
                D.add(D2);
        }

        return D;
}
```

Segmentové (úsečkové) stromy

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



Prehľadávací strom

- Usporiadame x-ové súradnice koncových bodov úsečiek, $E = \{e_1, e_2, \ldots, e_{2n}\}$
- Rozdelíme E na intervaly $[-\infty, e_1], \ldots, [e_{2n-1}, e_{2n}], [e_{2n}, \infty]$
- Základné intervaly budú listy prehľadávacieho stromu

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

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

Segmentový strom

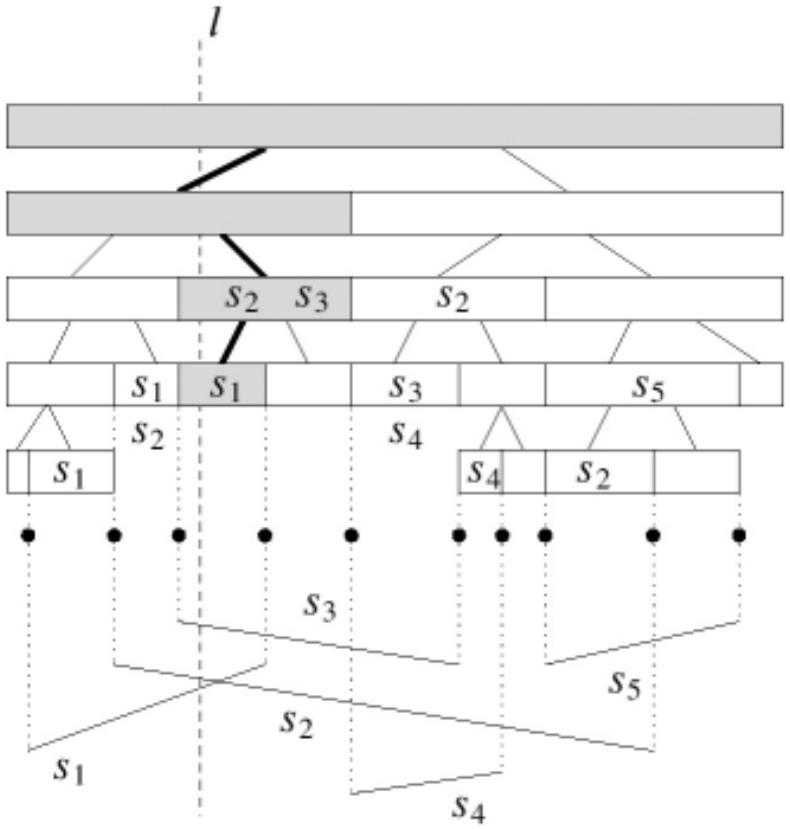
- Máme malé intervaly uložené v stromovej štruktúre
- Naplníme tento strom príslušnosťou ku úsečkám tak, aby každá úsečka bola pokrytá čo najmenším počtom intervalov (vrcholov) prehľadávacieho stromu
- Vytvorenie seg. stromu má časovú zložitosť O(n.log$n$), strom využije O(n.log$n$) pamäte

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->I.add(s);
        }
        else
        {
                InsertSegment(v->left, s);
                InsertSegment(v->right, s);
        }
}
```

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

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



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

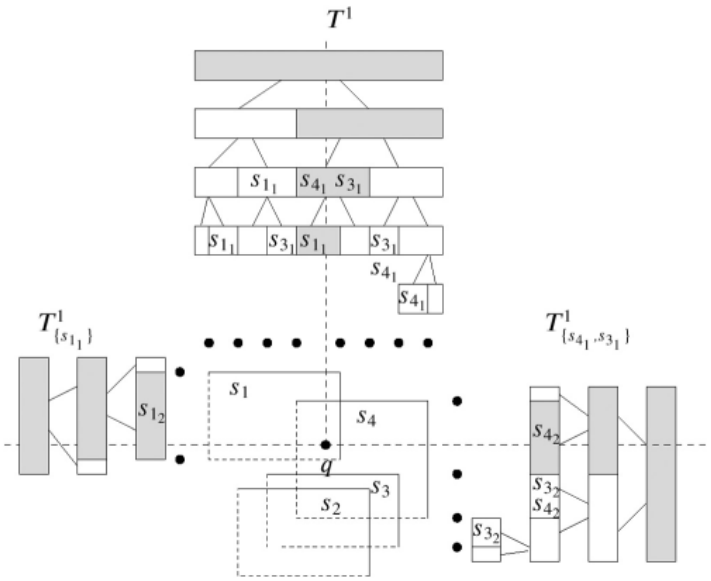
Viacrozmerné segmentové stromy

- Vstup: množina *d*-rozmerných obdĺžnikov *S* rovnobežných so súradnicovými osami
- Požiadavka: bod *q* z R^d
- Výstup: množina boxov 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 2

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



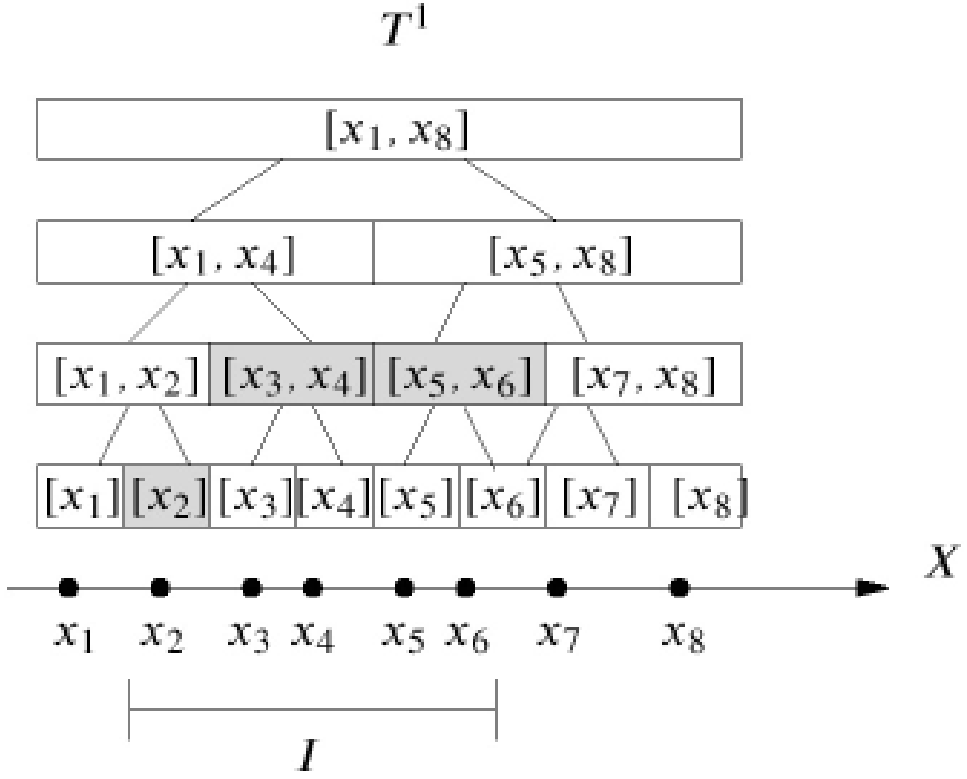

```
MLSegmentTreeQuery(T, q, d)
{
        if (d == 1)
        {
                L = StabbingQuery(T, q);
                return L;
        }
        else
        {
                List A;
                L = SearchTreeQuery(T, q.First);
                while (|L| != 0)
                {
            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

Range stromy

- Vstup: množina bodov *S* v R^d
- Požiadavka: obdĺžnik *B* z R^d rovnobežný so súradnicovými osami
- Výstup: Množina bodov z *S*, ktoré patria obdĺžniku B
- Podobná metóda ako pri segmentových stromoch
- Vytvorenie prehľadávacieho stromu na základe súradníc bodov
- Väčšinou $d \geq 2$

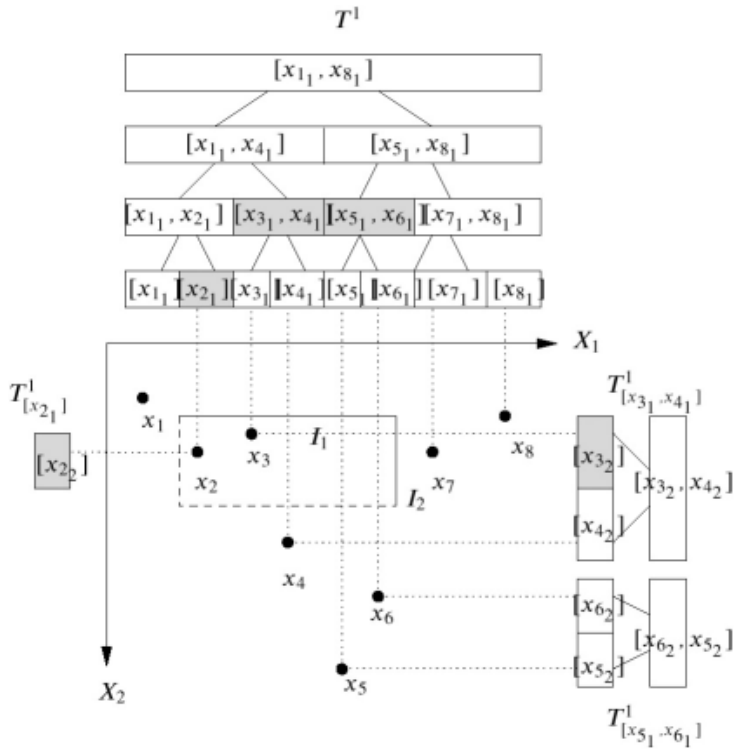
Range stromy 2

- Pre jednorozmerný prípad – hľadanie bodov vo vnútri intervalu
- Vytvorenie prehľadávacieho stromu



Range stromy 3

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



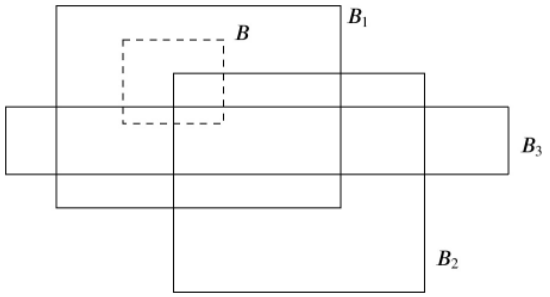

```
RangeTreeConstruct(S, d)
{
        S_f  = S.FirstCoordElements;
        S_f.Sort;
        T = SearchTree(S_f);
        T->dim = d;
        if (d > 1)
        {
                N = T->GetAllNodes;
                while (|N| != 0)
                {
                        u = N.First; N.DeleteFirst;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

Zložitosť

- *d*-rozmerný range strom pre množinu *n* bodov v R^d môže byť vytvorený v čase $O(n.\log^{(d-1)}n)$ a spotrebuje $O(n.\log^{(d-1)}n)$ pamäte
- Požiadavka na zistenie bodov vnútri *d*-rozmerného boxu má časovú náročnosť $O(k + \log^d n)$

AABB/AABB

- Vstup: množina *S* 2D obdĺžnikov
- Požiadavka: 2D obdĺžnik *B*
- Výstup: všetky AABB 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*



AABB/AABB 2

- Prípad 1. – 2D segmentový strom. Hľadáme do ktorých boxov 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 boxov 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 horizontálnych čiar v 2D
  - Požiadavka: Vertikálna čiara ***s*** v 2D
  - Output: Všetky úsečky pretínajúce ***s***

AABB/AABB – Prípad 3

- V smere *x* hľadanie častí s priesečníkom, v smere *y* časti patriace do intervalu
- Kombinácia intervalového a range stromu
- Najprv vytvoríme intervalový strom
- Lmed a Rmed nahradíme prislušnými 2D range stromami
- Čas $O(k_3 + \log^2 n)$, pamäť $O(n.\log n)$

Rozšírenie

- Nielen pre geometrické dáta
- Vyhľadávanie v *d*-rozmerných dátach
- Podľa požiadaviek, hľadanie dát v danom intervale, hľadanie približných dát
- Porovnávanie objektov

koniec (-:

florek@sccg.sk