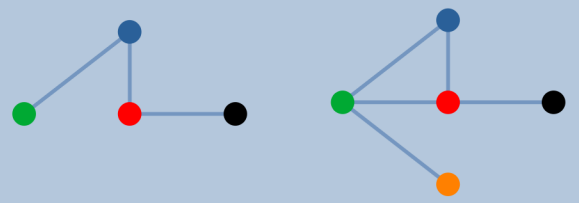
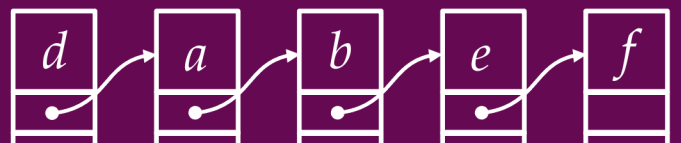
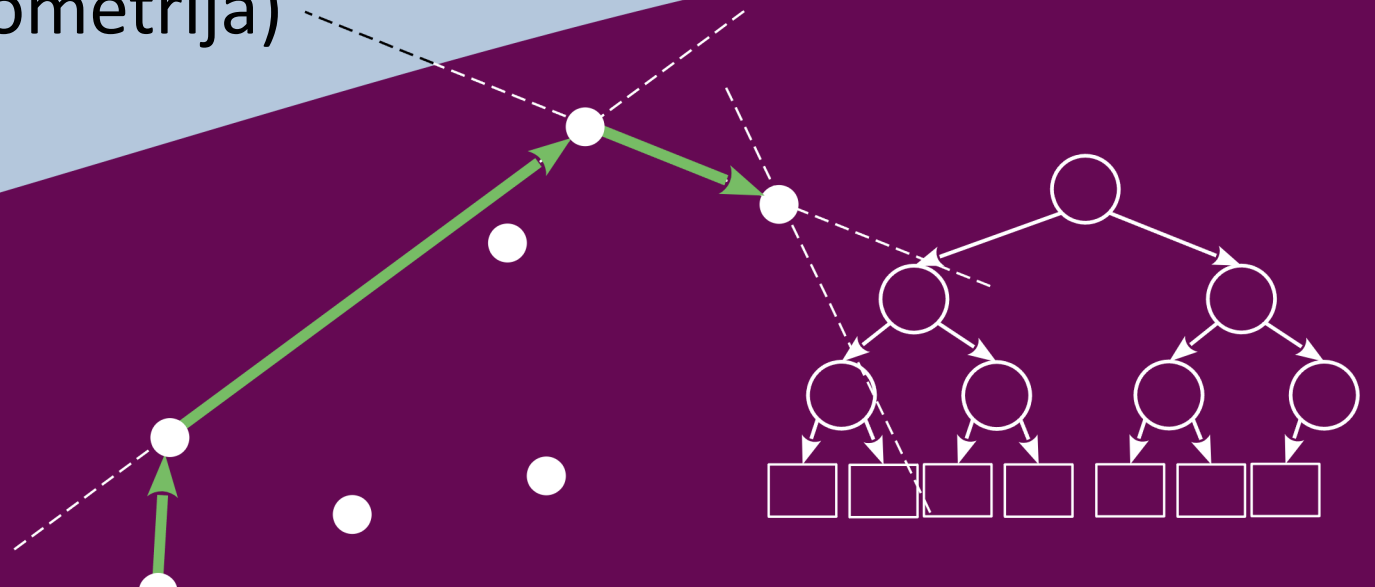
Napredni algoritmi i strukture podataka

Tjedan 4: Geometrijski algoritmi
(računalna geometrija)

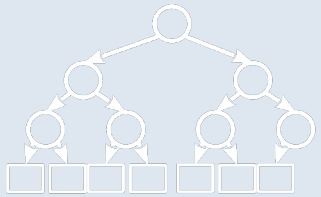




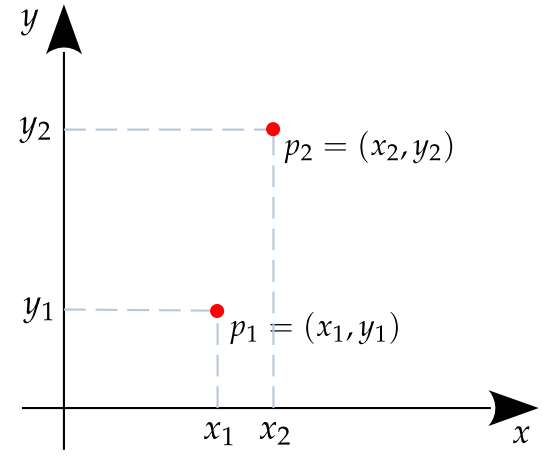


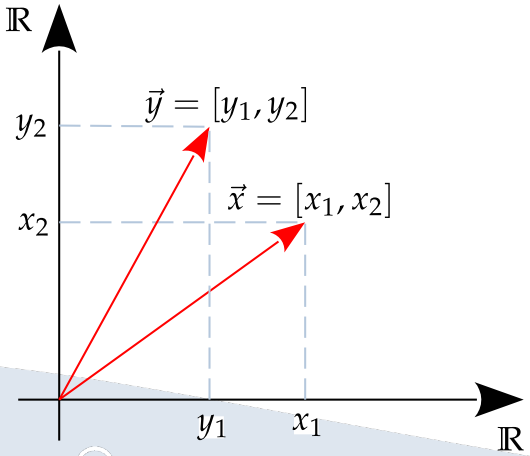
Što je računalna geometrija?

- Definicija: *sistematsko proučavanje struktura podataka i algoritama u geometriji*
- Primjeri:
  - **Računalna grafika** – jedno od najrazvijenijih područja – računalne igre, vizualizacija, modeliranje
  - **Prostorni dizajn produkata** – raspored na elektroničkim pločicama, dizajn integriranih čipova, pakiranja, **CAD/CAM**
  - **Robotika** – izračun pokreta i trajektorija
  - **GIS** – prostorne projekcije, smještanje objekata, proračuni putanja, pretraživanje objekata, presjeci, ...



Osnovni geometrijski elementi - točka





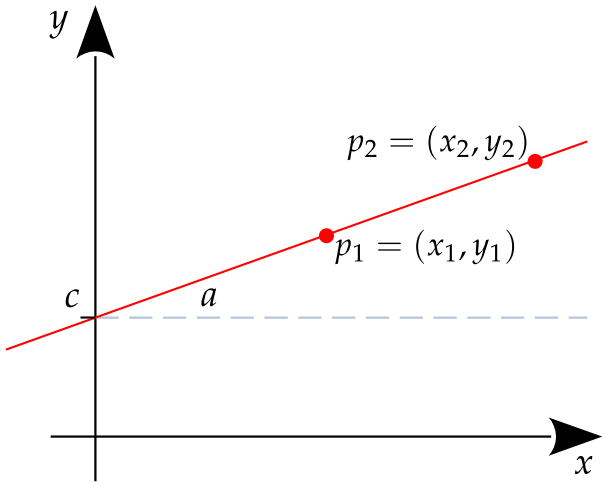
- Točka je infinitezimalni element $\mathbb{R}^n$ prostora

$$\mathbb{R}\times\mathbb{R}\times\mathbb{R}\times\cdots\times\mathbb{R} = (x_1, x_2, x_3, \ldots, x_n) = p$$

- Realni prostor $\mathbb{R}^n$ je apstraktan – preslikavamo ga u Kartezijev koordinatni sustav
- Točka se može napisati i u vektorskom obliku

$$\vec{x} = [x_1\ x_2\ x_3\ \ldots\ x_n]$$

Osnovni geometrijski elementi - linija



- Linija je beskonačni skup točaka u prostoru koje slijede linearnu jednadžbu

$$L = \{(x, y): (x, y) \in \mathbb{R}^2, ax + by = c\}$$

- Linearna jednadžba može se napisati i kao

$$y = c - ax$$

- Ako imamo dvije točke u prostoru kroz koje linija prolazi

$$p_1 = (x_1, y_1), p_2 = (x_2, y_2)$$

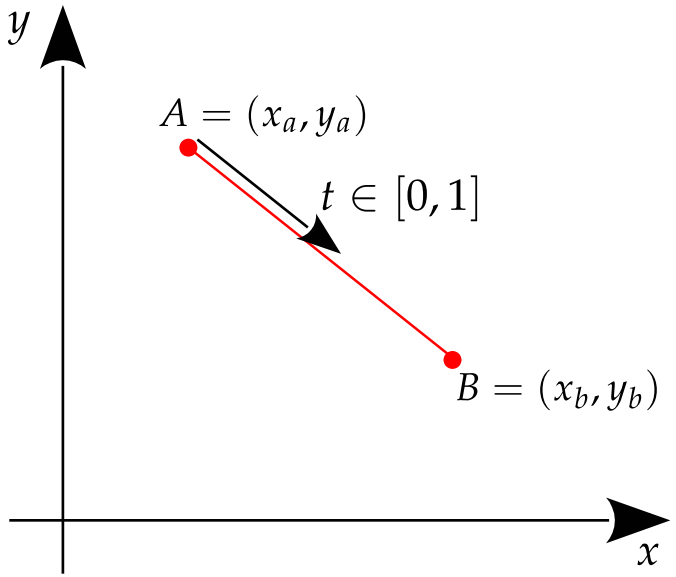
- Linija se može napisati kao

$$a = \frac{y_2 - y_1}{x_1 - x_2}, c = y_1 + \frac{y_2 - y_1}{x_1 - x_2} x_1$$

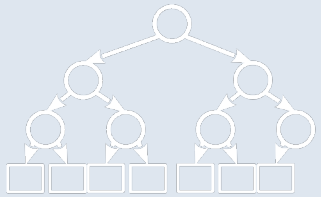
Osnovni geometrijski elementi – linija i segment

- Linija se može parametrizirati

$$\Delta y = y_2 - y_1, \Delta x = x_2 - x_1$$
$$y = y_1 + t\Delta y, x = x_1 + t\Delta x$$



- Parametar $t \in [0,1]$ određuje poziciju točke na liniji:
  - za $t = 0$ nalazimo se u p_1, dok se za $t = 1$ nalazimo u p_2
- Parametarski pristup koristimo za omeđivanje linije
- Linija prolazi točkama $A = (x_a, y_b), B = (x_b, y_b)$
- Ako smatramo da ju točke A i B omeđuju
  - Za vrijednosti parametra $t \in [0,1]$ dobivamo skup točaka koji predstavljaju segment linije ili dužinu $\overline{AB}$.

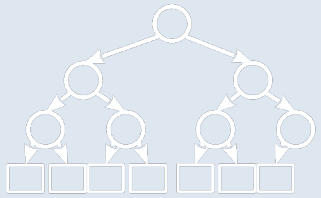


Osnovni geometrijski elementi – ploha

- Promotrimo geometrijski element koji ima jednu dimenziju manje od prostora u kojem se nalazi
  - Na primjer točka (0 dimenzija) na liniji (1 dimenzija)
- U trodimenzionalnom prostoru to je ploha
- Ploha je beskonačan skup točaka koje slijede linearnu jednadžbu

$$P = \{(x, y, z): (x, y, z) \in \mathbb{R}^3, ax + by + cz = d\}$$

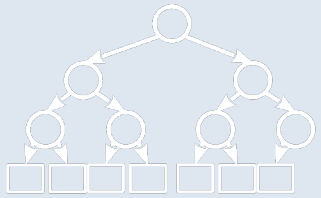
- Podskup točaka plohe nazivamo geometrijskim oblikom



Osnovni geometrijski elementi – poligoni

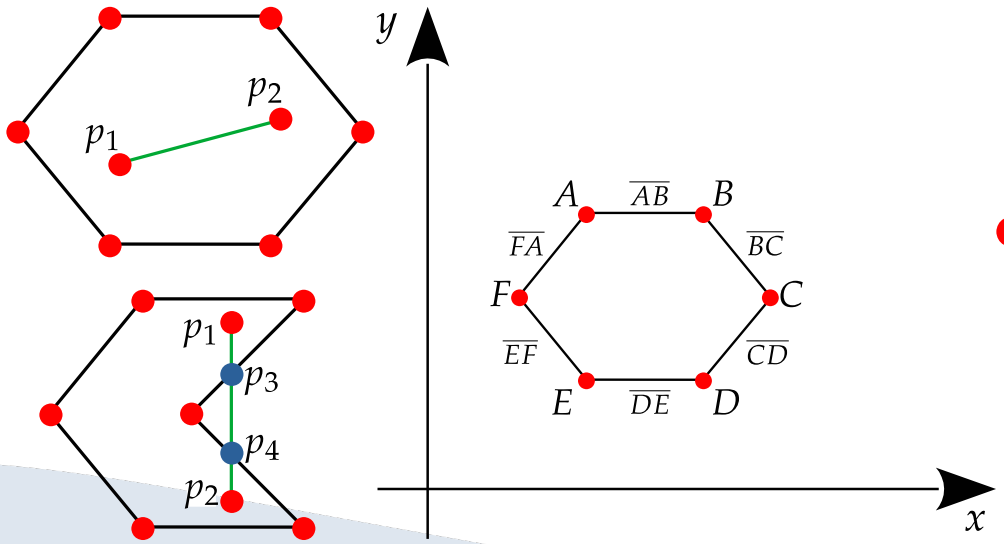
- Podskup plohe $\mathcal{P} \in P$ ograničen skupom točaka i linijskim segmentima između tih točaka naziva se poligon
- Poligon sastavljen od n točaka naziva se n-gon
- Linijski segmenti omeđuju poligon čineći brid *poligona*, dok se točke u poligonu nazivaju *tijelo poligona*
- Kada nabrajamo točke na bridu poligona to uobičajeno činimo u smjeru kazaljke na satu
- Suma unutarnjih kutova poligona je

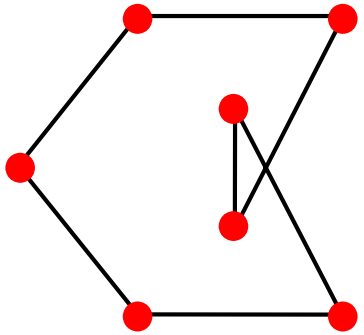
$$S = (n - 2) * 180^{\circ}$$



Osnovni geometrijski elementi – poligoni

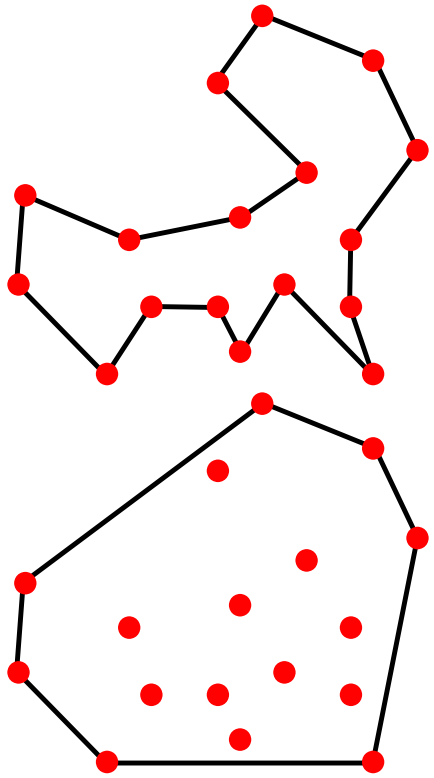
- Poligon se smatra **konveksnim** ako za svaki par točaka $p_1, p_2 \in \mathcal{P}$, dužina $\overline{p_1 p_2}$ prolazi unutar poligona
- Poligon se smatra **jednostavnim** ako nema interakcije sa samim sobom, na primjer križanje sa svojim bridom





- **Regularni** poligoni su ekviangularni i ekvilateralni, što znači da su im unutarnji kutevi i duljine svih linijskih segmenata jednaki
  - **Primjer:** jednostranični trokut, kvadrat, pentagon, hexagon, heptagon, ...

Plošna konveksna ljuska (Convex Hull)



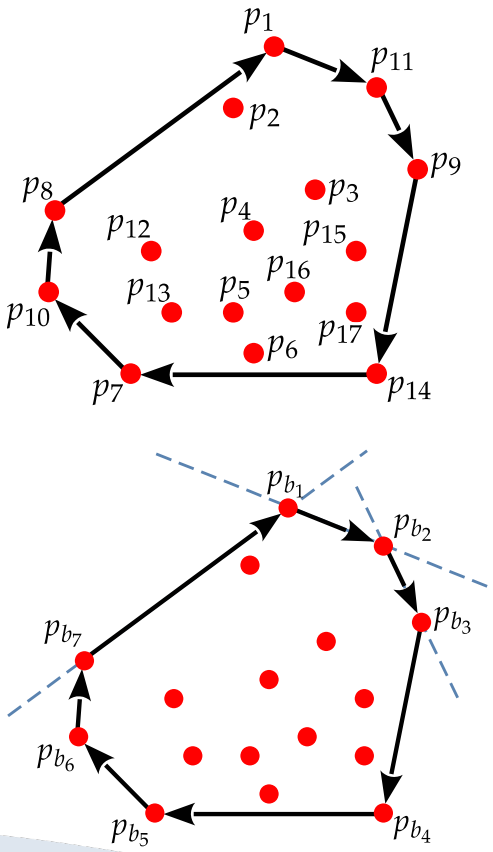
- Zamislimo skup točaka u $\mathbb{R}^2, P_p = \{p_1, p_2, \dots, p_n\}$
- Postoji podskup tih točaka, takav da čini brid poligona $\mathcal{P}$ koji sadržava sve točke iz skupa P_p, tako da vrijedi

$$\nexists p \in P_p : p \notin \mathcal{P}$$

- Ako je poligon $\mathcal{P}$ konveksan, tada se radi o konveksnoj ljusci

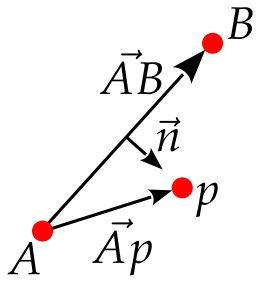
- Zamislimo da je P_p skup čavala zabijen u dasku
- Stavimo gumicu oko tih čavala
- Gumica predstavlja rub poligona koji je konveksna ljuska

Plošna konveksna ljuska I



- Nabrajamo točke koje pripadaju bridu poligona $b(\mathcal{P}) \subseteq P_p$ u smjeru kazaljke na satu
- Jednostavni način utvrđivanja da je $b(\mathcal{P})$ konveksna ljuska
  - Uzmemo svaki linijski segment ruba poligona i gledamo da li su sve ostale točke u poligonu **desno** od njegove linije
  - Ako da, radi se o konveksnoj ljusci
- Kako utvrditi da je točka „**desno**” od linije?
- I što to uopće znači „**desno**”?

Plošna konveksna ljuska I



- Ako imamo dvije točke $A = (x_a, x_b), B = (x_b, y_b)$ i njihov linijski segment $\overline{AB}$, tada se može definirati vektor

$$\overrightarrow{AB} = [x_b - x_a \quad y_b - y_a]$$

- Uz sustav jednadžbi

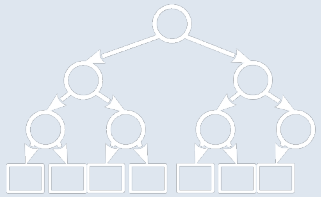
$$\overrightarrow{AB} * \vec{n}^T = 0, D = \vec{n} * \overrightarrow{Ap}^T$$

- Dobivamo

$$D = [y_b - y_a \quad x_a - x_b] \begin{bmatrix} x - x_a \\ y - y_a \end{bmatrix}$$

$$D = (x - x_a)(y_b - y_a) - (y - y_a)(x_b - x_a)$$

- Za sve $D > 0$ smatramo da je točka p **desno** od linijskog segmenta $\overline{AB}$

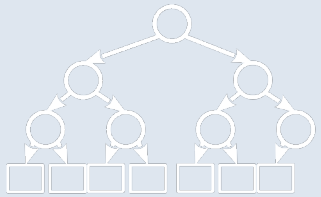


Plošna konveksna ljuska I

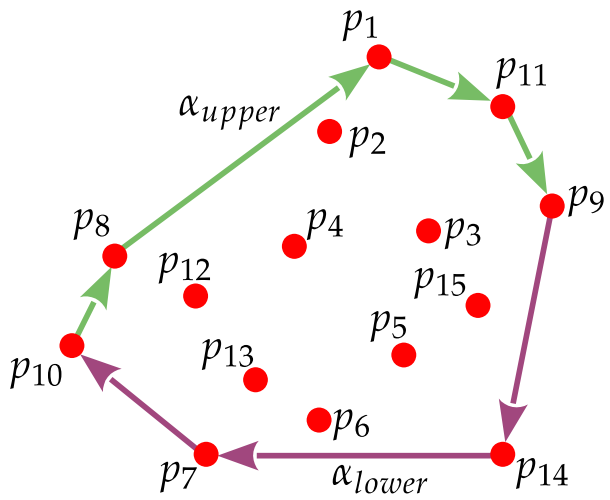
```
function SimpleConvexHull(P_p)
    α(P_p) ← ∅
    for each pair of points p_i, p_j ∈ P_p do
        bound ← 1
        for each point p_k ∈ P_p such that p_k ∉ {p_i, p_j} do
            if p_k is left from the line segment p_i p_j then
                bound ← 0
        if bound is 1 then
            add p_i p_j to α(P_p) taking care of the clockwise order
    return α(P_p)
```

Kompleksnost = $O(n^3)$

- S obzirom da na početku imamo samo P_p, moramo pronaći rub $b(\mathcal{P})$ takav da predstavlja konveksnu ljusku, što je označeno sa $\alpha(\mathcal{P})$
- Uzimamo parove točaka iz $p_i, p_j \in P_p$ i provjeravamo da li su sve ostale točke iz P_p desno od njihovog linijskog segmenta $\overline{p_i p_j}$
- Ako da, onda taj linijski segment $\overline{p_i p_j}$ pripada $\alpha(\mathcal{P})$
- Dva velika problema:
  - Algoritam ne osigurava da je konveksna ljuska zatvorena
  - Visoka kompleksnost rješenja



Plošna konveksna ljuska II



- Da li je moguć drukčiji, brži algoritam za pronalazak konveksne ljuske?
- Razdijelimo konveksnu ljusku na gornji i donji dio

$$\alpha(\mathcal{P}) = \alpha_{lower}(\mathcal{P}) \cup \alpha_{upper}(\mathcal{P})$$

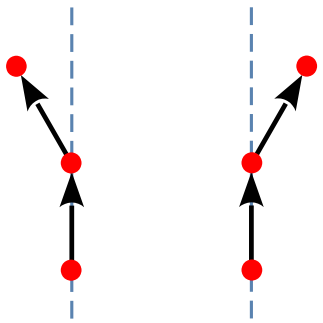
- Horizontalno sortiramo točke u P_p

$$h(P_p) = \{p_{h_i} : p_{h_i} \in P_p\}$$

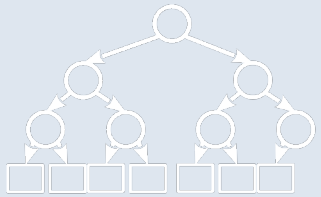
$$\forall p_{h_i}, p_{h_j} \in h(P_p) : i < j, x(p_{h_i}) \leq x(p_{h_j})$$

- Primjećujemo da je prva točka u $h(P_p)$ početak, a zadnja točka u $h(P_p)$ završetak gornjeg dijela konveksne ljuske ili $\alpha_{upper}(\mathcal{P})$
- Istovjetno, samo okrenuto, zadnja točka u $h(P_p)$ početak, a prva točka u $h(P_p)$ završetak donjeg dijela konveksne ljuske ili $\alpha_{lower}(\mathcal{P})$

Plošna konveksna ljuska II



- I dalje oba dijela konveksne ljuske obilazimo u smjeru kazaljke na sati
- To znači da je konveksna ljuska, a time i oba njena dijela **desno naginjuća**
- Da bismo utvrdili da je dio konveksne ljuske desno naginjući, trebamo barem tri točke
- Koristimo se istim principom utvrđivanja da li je treća točka po redu desno ili lijevo od linije koju čine prve dvije točke

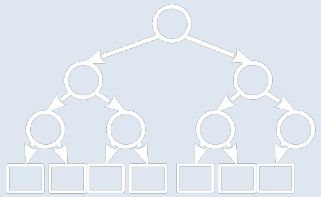


Plošna konveksna ljuska II

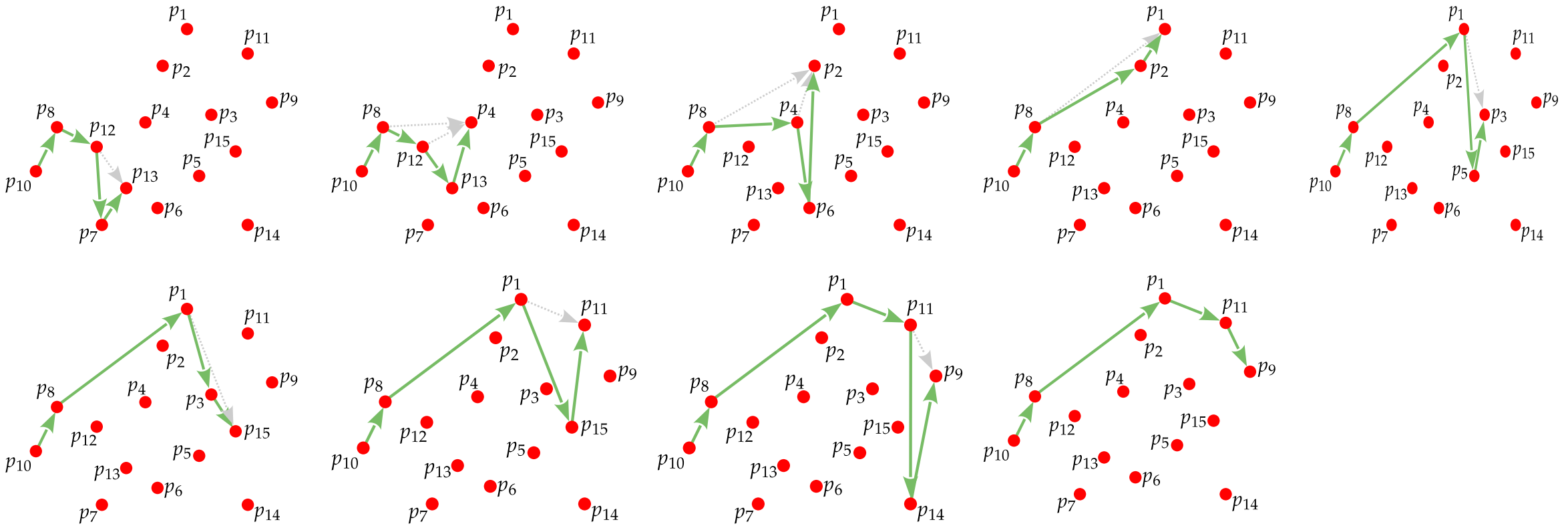
function ConvexHull(P_p)
  create $h(P_p)$ as the sorted list of points from P_p
  $\alpha_{upper}(P_p) \leftarrow \{p_{h_1}, p_{h_2}\}$ from $h(P_p)$
  for $i \leftarrow 3$ **to** n **do**
    add p_{h_i} to $\alpha_{upper}(P_p)$
    while $|\alpha_{upper}(P_p)| \geq 3$ and last three points incline left **do**
      remove the point before the last from $\alpha_{upper}(P_p)$
  $\alpha_{lower}(P_p) \leftarrow \{p_{h_n}, p_{h_{n-1}}\}$ from $h(P_p)$
  for $i \leftarrow n-2$ **downto** 1 **do**
    add p_{h_i} to $\alpha_{lower}(P_p)$
    while $|\alpha_{lower}(P_p)| \geq 3$ and last three points incline left **do**
      remove the point before the last from $\alpha_{lower}(P_p)$
  return $\alpha_{upper}(P_p) \cup \alpha_{lower}(P_p)$

Kompleksnost = $O(n \log n)$

- Prvo horizontalno sortiramo listu točaka
- Dodamo prve dvije točke iz $h(P_p)$ u gornju konveksnu ljusku
- U petlji krećemo od treće točke
  - Ukoliko zadnje tri točke naginju lijevo, uklanjamo srednju od njih iz gornje konveksne ljuske
- Kada stignemo do zadnje točke u $h(P_p)$, dobili smo gornju konveksnu ljusku
- Za donju se radi isti postupak, ali obrnutim redoslijedom kroz $h(P_p)$

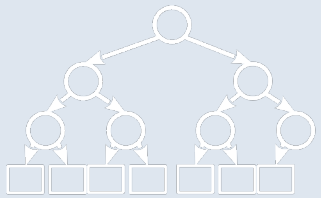


Plošna konveksna ljuska II

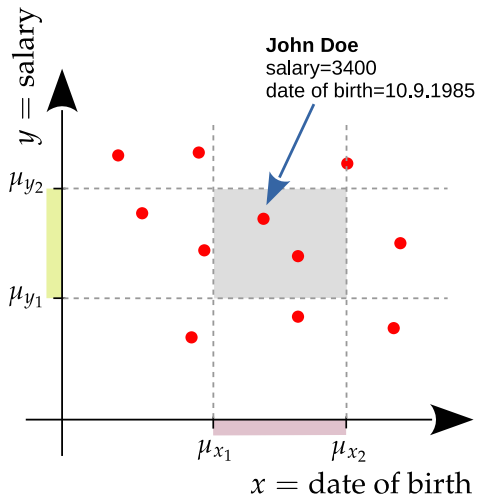


Plošna konveksna ljuska II

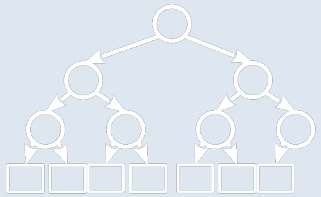
- Algoritam prolazi kroz svih n točaka iz P_p, čime bi kompleksnost bila $O(n)$
- Kako imamo potrebu koristiti neki od algoritama za sortiranje, za stvaranje $h(P_p)$, tako nam kompleksnost algoritma za sortiranje prevladava i ukupna kompleksnost je $O(n \log n)$
- Koncept konveksne ljuske koristi se kasnije u linearnom programiranju



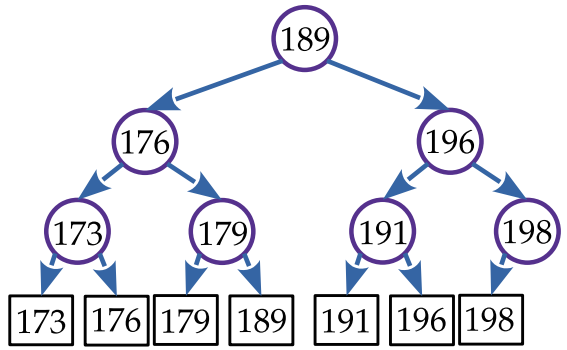
Geometrijsko pretraživanje



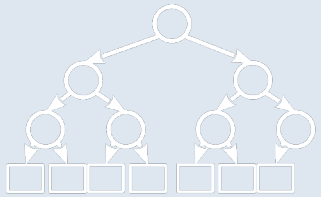
- Zamislimo da u bazi podataka imamo skup podataka koji možemo smjestiti u n-dimenzionalni prostor
- Neke od dimenzija mogu biti *visina*, *težina*, *plaća*, itd...
- Zaboravimo na trenutak sve mogućnosti baze podataka
- B^+ indeksna stabla nam pomažu dohvatiti podatke za točno određenu vrijednost atributa
- Kako dohvatiti skup podataka koji ima određeni atribut u određenom rasponu vrijednosti?
- Na primjer: pronađimo sve osobe koje su visine od 165 do 184 centimetara
- Generalno, imamo skup vrijednosti (1-dimenzionalan) iz kojeg što je brže moguće želimo pronaći vrijednosti u traženom rasponu



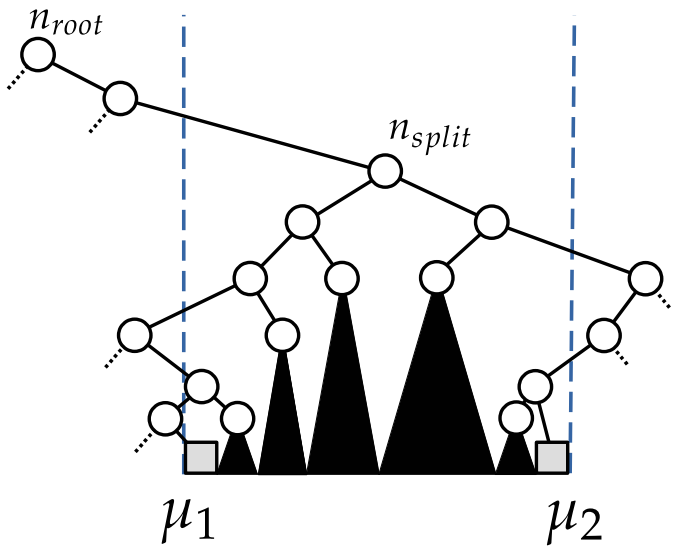
Pretraživanje raspona (1D)



- Imamo skup vrijednosti P, te raspon koji tražimo $[\mu_1, \mu_2]$
- Na primjer $P = \{173, 176, 179, 189, 191, 196, 198\}$
- Kako organizirati skup vrijednosti P tako da imamo mogućnost pronalaska raspona $[\mu_1, \mu_2]$ na najefikasniji mogući način?
- Pribjegavamo korištenju binarnih stabala
- U ovom slučaju koristimo binarna stabla koja u listovima imaju konkretne vrijednosti – zovemo ih **stabla raspona**
  - Listovi imaju vrijednosti iz P
  - Unutarnji čvorovi su navodeći (*guiding*) i ne trebaju imati iste vrijednosti kao listovi
  - Princip sličan kao i kod B^+ stabala

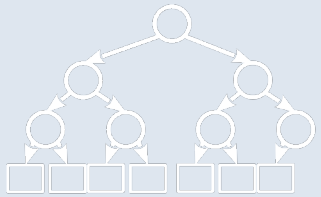


Pretraživanje raspona (1D)

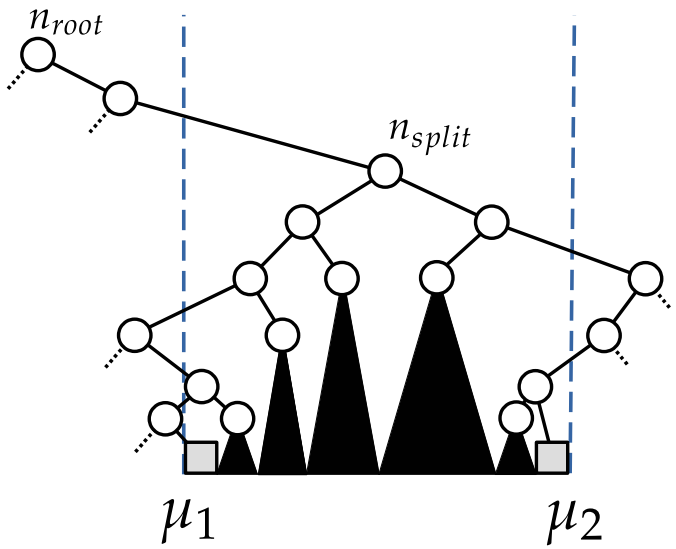


- Imamo binarno stablo τ, takvo da svaki unutarnji čvor ima vrijednost da vrijedi:
  - U lijevom podstablu su sve vrijednosti koje su $\leq$
  - U desnom podstablu su sve vrijednosti koje su $>$

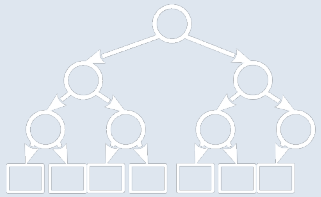
1. Krenemo od korijenskog čvora tražeći prvi čvor koji je u traženom rasponu $[\mu_1, \mu_2]$, takozvani *razdvajajući čvor* n_{split}
2. Od razdvajajućeg čvora n_{split} krećemo se na lijevu i na desnu stranu, tako dobivamo dvije vertikalne putanje od čvora do lista: $\mathcal{V}_L$ i $\mathcal{V}_R$



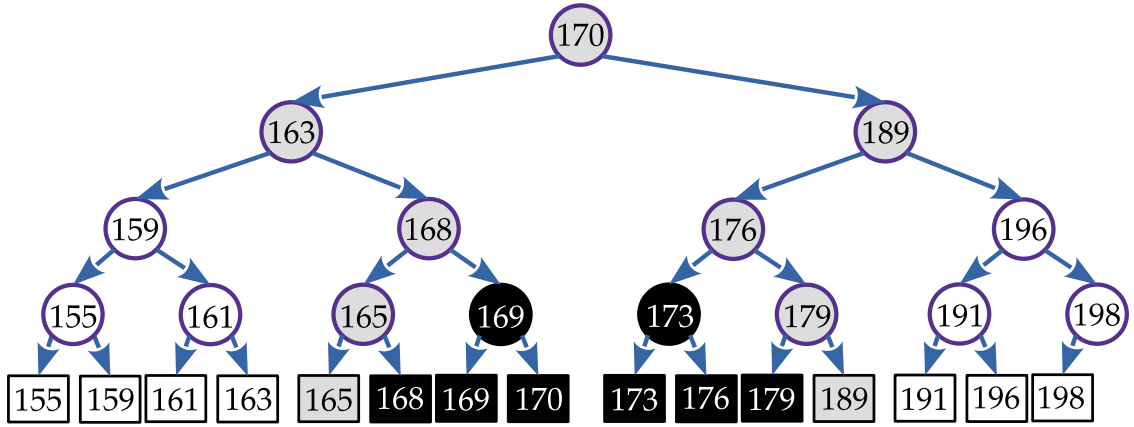
Pretraživanje raspona (1D)



3. Krećući se na lijevo, znamo da su čvorovi manji od n_{split}, te imamo dvije opcije:
   a) Čvor je unutar traženog raspona $[\mu_1, \mu_2]$, što znači
      - Njegovo desno podstablo je sigurno u traženom rasponu i sve vrijednosti automatski dodajemo rezultatu (*pruning*)
      - Za njegovo lijevo podstablo nismo sigurni, pa se krećemo na lijevo dijete
   b) Čvor nije unutar traženog raspona, što znači
      - Njegovo lijevo podstablo sigurno nije u rasponu i ignoriramo ga
      - Za njegovo desno podstablo nismo sigurni, pa se krećemo na desno dijete
4. Kretanjem u desno izvodimo zrcalne operacije



Pretraživanje raspona (1D)



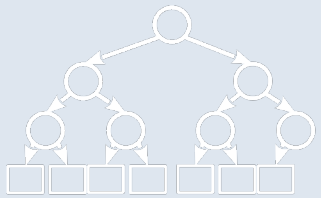
Kompleksnost = $O(k + \log n)$

- Primjer stabla i pretraživanja raspona $[165,184]$
- Korijenski čvor je ujedno i n_{split}
- Sivi čvorovi su testirani
- Crni čvorovi su automatski dodani u rezultat *(pruned)*
- Bijeli čvorovi su odbačeni kao van raspona
- Kompleksnost pretraživanja je $O(k + \log n)$, gdje je k broj automatski dodanih vrijednosti, a $\log n$ su dva prolaska kroz binarno stablo

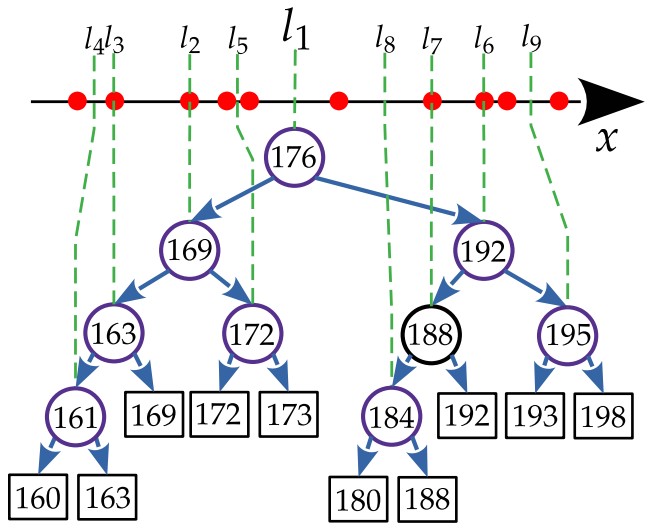
Pretraživanje raspona (1D)

```
function 1DRangeQuery(τ, μ1, μ2)
    rv ← ∅
    n_split ← FindSplittingNode(τ, μ1, μ2)
    if n_split is leaf then
        if v(n_split) is in the range [μ1, μ2] then
            add n_split to rv
    else
                                                ▷ left traversal
        n ← leftChild(n_split)
        while n is not leaf do
            if μ1 ≤ v(n) then
                add ReportSubtree(rightChild(n)) to rv
                n ← leftChild(n)
            else
                n ← rightChild(n)
        if n is leaf and v(n) is in the range [μ1, μ2] then
            add n to rv
                                                ▷ right traversal
        n ← rightChild(n_split)
        while n is not leaf do
            if v(n) ≤ μ2 then
                add ReportSubtree(leftChild(n)) to rv
                n ← rightChild(n)
            else
                n ← leftChild(n)
        if n is leaf and v(n) is in the range [μ1, μ2] then
            add n to rv
    return rv
```

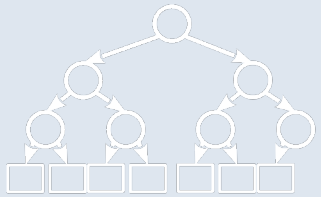
```
function FindSplittingNode(τ, μ1, μ2)
    n ← root(τ)
    while n is not leaf and (μ1 ≥ v(n) or μ2 ≤ v(n)) do
        if μ2 ≤ v(n) then
            n ← leftChild(n)
        else
            n ← rightChild(n)
    return n
```



Pretraživanje raspona (1D)



- Kako stvoriti binarno stablo koje reflektira određeni skup vrijednosti P?
- Kod binarnih stabala učili smo kako stvoriti stablo na temelju sortiranog skupa vrijednosti - promišljeno
- Sličan princip koristi se i ovdje
  - Stvoreni čvor sadrži medijan svih vrijednosti koje su u njegovom podstablu – želimo da stablo bude uravnoteženo
  - Njegovo lijevo podstablo sadrži samo vrijednosti koje su $\leq$ od vrijednosti čvora
  - Njegovo desno podstablo sadrži samo vrijednosti koje su $>$ od vrijednosti čvora
  - Rekurzivnim postupkom stvaramo binarno stablo sve do njegovih listova – konkretnih vrijednosti

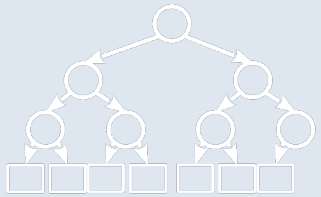


Pretraživanje raspona (1D)

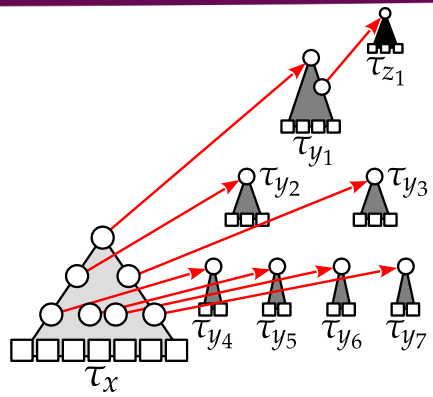
```
function CREATE1DRANGETREE(P_p)
    if |P_p| = 1 then
        return n ← leaf having the value from P_p
    else
        v_m ← median(P_p)
        P_{p_left} ← {p_i : p_i ∈ P_p ∧ p_i ≤ v_m}
        P_{p_right} ← {p_i : p_i ∈ P_p ∧ p_i > v_m}
        n_left ← CREATE1DRANGETREE(P_{p_left})
        n_right ← CREATE1DRANGETREE(P_{p_right})
        n ← create node having value v_m
        leftChild(n) ← n_left
        rightChild(n) ← n_right
    return n
```

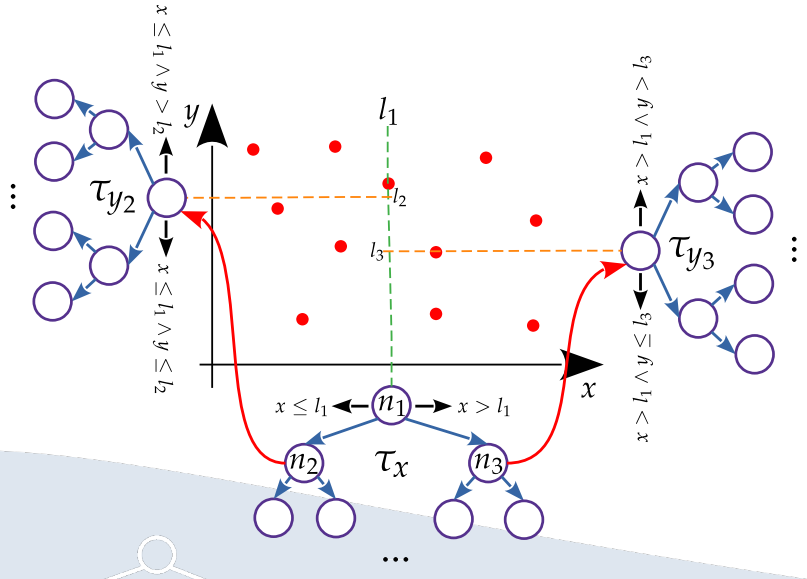
1. Ako smo dobili samo jednu vrijednost
   a) Vratimo čvor s tom vrijednošću (list)
2. Ako smo dobili više vrijednosti
   a) Izračunamo medijan za korijenski čvor
   b) Podijelimo vrijednosti na lijeve ($\leq$) i desne ($>$)
   c) Rekurzivno stvorimo lijevo i desno podstablo
   d) Postavimo stvorena podstabla kao lijevo i desno dijete korijenskog čvora

- Kompleksnost stvaranja stabla raspona je $O(n \log n)$ radi sortiranja zbog traženja medijana



Pretraživanje raspona (2D)

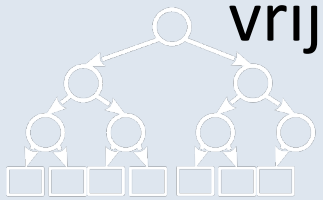




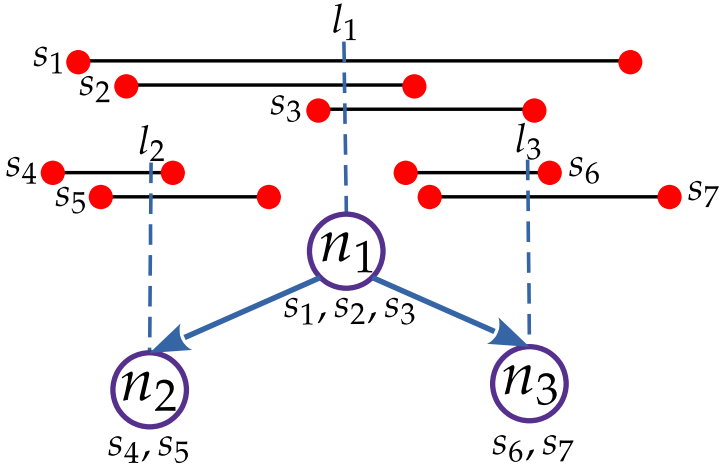
- Svakom čvoru stabla raspona dodajemo podstablo τ_{assoc} za dodatnu dimenziju (y)
  - Ovo ulančavanje se može nastaviti za ostale dimenzije
- τ_{assoc} predstavlja stablo raspona za sve točke koje predstavlja čvor n_i, ali za višu dimenziju – ako je n_i za x-os, tada se τ_{assoc} formira za y-os
- Kompleksnost pretraživanja za dvije dimenzije je sada $O(k + \log^2 n)$ radi ulančavanja stabala
- Kompleksnost stvaranja je i dalje $O(n \log n)$ radi sortiranja

Pretraživanje intervala (1D)

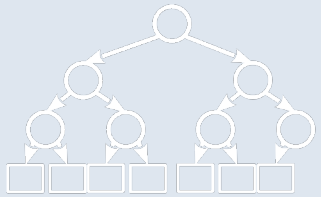
- Zamislimo skup intervala $I = \{s_i = ([x_{i_1}, x_{i_2}], y_i) : x_{i_1}, x_{i_2}, y_i \in \mathbb{R}\}$ koji su paralelni sa x-osi – zapravo horizontalni linijski segmenti
- Točke (x_{i_1}, y_i) i (x_{i_2}, y_i) zovemo krajnjim točkama intervala
- Želja nam je imati strukturu u koju spremamo skup intervala, tako da brzo i efikasno možemo pronaći intervale koji se nalaze u prozoru upita $W_q = [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$
- Rješenje se nalazi u kompozitnom binarnom stablu koje vrijednosti sprema u čvorovima – zovemo ih **stabla intervala**
- U ovom slučaju nemamo listove koji predstavljaju konkretne vrijednosti



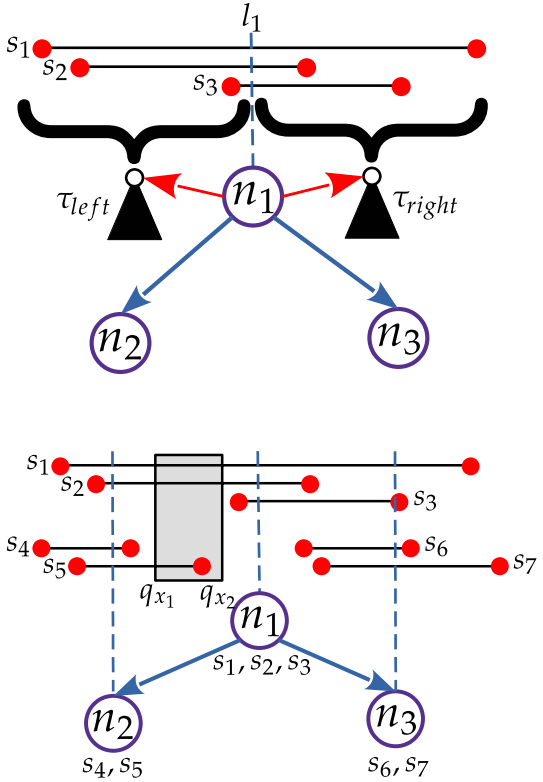
Pretraživanje intervala (1D)



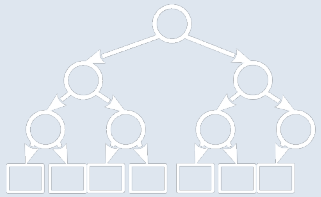
- Izračunamo medijan v_m svih krajnjih točaka intervala iz skupa intervala za koji stvaramo stablo
  - Za korijenski čvor skup intervala je I
  - Za svaki čvor ispod korijenskog to je podskup od I – skupa intervala spremljenog u roditelja
- Stvorimo čvor s vrijednošću izračunatog medijana v_m
  - U čvor spremimo sve intervale koji presijecaju v_m
  - U n_1 tako spremamo intervale $\mathrm{I}(n_1) = \{s_1, s_2, s_3\}$
- Rekurzivno stvaramo lijevo i desno podstablo
  - U lijevo podstablo idu svi intervali čije su obje krajnje točke $< v_m$
  - U desno podstablo idu svi intervali čije su obje krajnje točke $> v_m$



Pretraživanje intervala (1D)



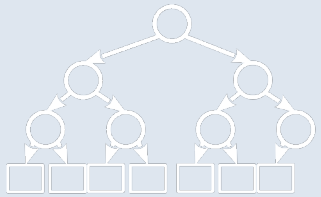
- Svakom čvoru se dodaju dvije dodatne strukture podataka (2D stabla raspona recimo)
  - U lijevom τ_{left} se čuvaju lijeve krajnje točke intervala spremljenih u taj čvor ($\leq$)
  - U desnom τ_{right} se čuvaju desne krajnje točke intervala spremljenih u taj čvor ($>$)
  - Krajnje točke u stablima τ_{left} i τ_{right} imaju povratne reference na intervale u čvorovima kako bi se izbjeglo bilo kakvo dodatno pretraživanje
- Ovo radimo zato što pretraživanje rezultira sa dva moguća scenarija:
  1. Interval u potpunosti presijeca $[q_{x_1}, q_{x_2}]$, što znači da mu je lijeva krajnja točka $< q_{x_1}$, a desna $> q_{x_2}$
  2. Interval ima početak ili kraj u $[q_{x_1}, q_{x_2}]$, što znači da mu je lijeva ili desna krajnja točka u tom rasponu
  3. Trebamo paziti i da je interval u rasponu $[q_{y_1}, q_{y_2}]$
- Ovo se može utvrditi kroz dodatne strukture τ_{left} ili τ_{right}
  - Zato se stablo intervala i naziva kompozitnim



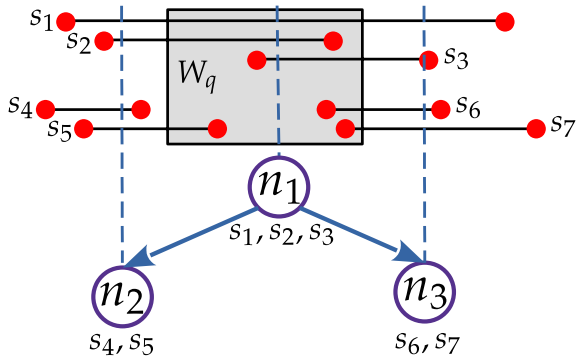
Pretraživanje intervala (1D)

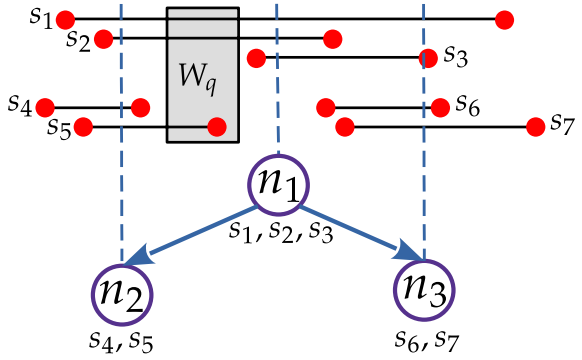
function CreateIntervalTree(I)
  if $I = \emptyset$ **then**
    $n \leftarrow$ empty leaf
  else
    $x_{med} \leftarrow median(endpoints(I))$
    $I_{left} \leftarrow \{s_i : s_i \in I \wedge x_{i_1}(s_i) < x_{med} \wedge x_{i_2}(s_i) < x_{med}\}$
    $I_{right} \leftarrow \{s_i : s_i \in I \wedge x_{i_1}(s_i) > x_{med} \wedge x_{i_2}(s_i) > x_{med}\}$
    $I_{med} \leftarrow I \setminus (I_{left} \cup I_{right})$
    $n \leftarrow$ create a node having value x_{med}
    $intervals(n) \leftarrow I_{med}$
    $leftChild(n) \leftarrow$ CreateIntervalTree(I_{left})
    $rightChild(n) \leftarrow$ CreateIntervalTree(I_{right})
    if $I_{med} \neq \emptyset$ **then**
      $P_{left} \leftarrow \{((x_{i_k}(s_i), y_i(s_i)), s_i) : s_i \in I_{mid} \wedge x_{i_k}(s_i) \leq x_{med}\}$
      $P_{right} \leftarrow \{((x_{i_k}(s_i), y_i(s_i)), s_i) : s_i \in I_{mid} \wedge x_{i_k}(s_i) > x_{med}\}$
        ▷ We include the back reference to s_i
      $\tau_{left}(n) \leftarrow$ Create2DRangeTree(P_{left})
      $\tau_{right}(n) \leftarrow$ Create2DRangeTree(P_{right})
  return n

- Zbog potrebe za sortiranjem, osnovno stablo intervala ima kompleksnost $O(n \log n)$
- Dodamo li tome dva stabla raspona po čvoru, dobivamo $O\big((n \log n)(2 \log n)\big)$
- Što je u konačnici $O(n \log^2 n)$

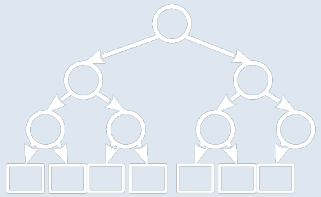


Pretraživanje intervala (1D)

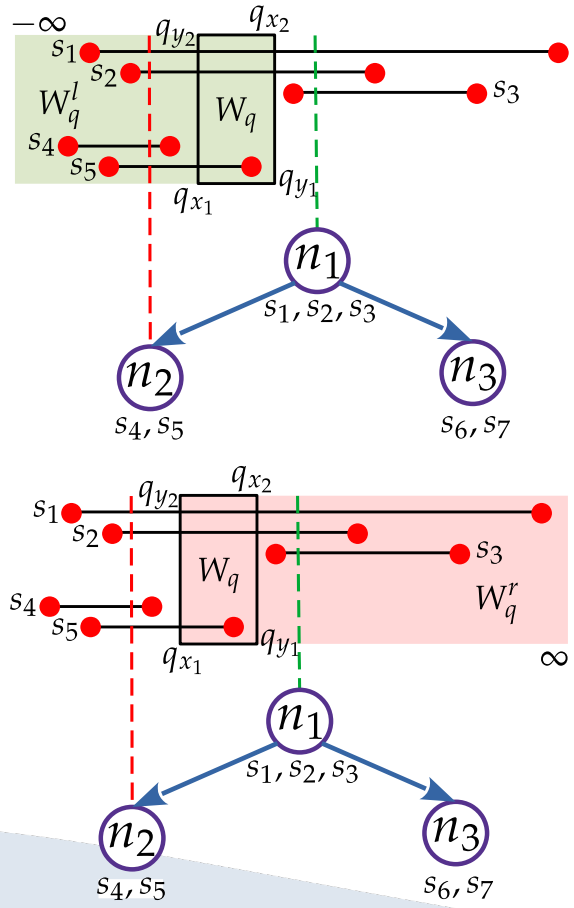




- Pretraživanje stabla intervala slično je pretraživanju stabla raspona. Ako se trenutno nalazimo u čvoru n_i (krećemo od korijenskog čvora), tada imamo tri osnovna slučaja:
  - medijan čvora $x_{med}(n_i)$ je unutar raspona $[q_{x_1}, q_{x_2}]$
    - konzultiraju se oba stabla τ_{left} i τ_{right}, kako bi se selektirali intervali koji završavaju u prozoru $[q_{x_1}, q_{x_2}]\times[q_{y_1}, q_{y_2}]$
  - medijan čvora $x_{med}(n_i)$ s lijeve je strane raspona $[q_{x_1}, q_{x_2}]$
    - konzultira se desno dodatno stablo τ_{right}, kako bi se selektirali intervali koji desnom stranom završavaju u prozoru $[q_{x_1}, q_{x_2}]\times[q_{y_1}, q_{y_2}]$
    - za lijevo podstablo čvora n_i smatramo da nema traženih intervala
    - nastavljamo u desnom podstablu
  - medijan čvora $x_{med}(n_i)$ s desne je strane raspona $[q_{x_1}, q_{x_2}]$
    - konzultira se lijevo dodatno stablo τ_{left}, kako bi se selektirali intervali koji lijevom stranom završavaju u prozoru $[q_{x_1}, q_{x_2}]\times[q_{y_1}, q_{y_2}]$
    - za desno podstablo čvora n_i smatramo da nema traženih intervala
    - nastavljamo u lijevom podstablu



Pretraživanje intervala (1D)



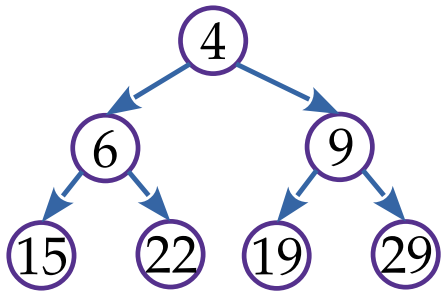
- No, što se dešava s intervalima koji u potpunosti presijecaju raspon, kao recimo s_1 i s_2 u primjeru?
- Umjesto da krajnje točke pretražujemo isključivo u rasponu $[q_{x_1}, q_{x_2}]$, pretraživanje proširimo
  - Lijeve krajnje točke pretražujemo u prozoru $(-\infty, q_{x_2}] \times [q_{y_1}, q_{y_2}]$
  - Desne krajnje točke pretražujemo u prozoru $[q_{x_1}, \infty) \times [q_{y_1}, q_{y_2}]$

Pretraživanje intervala (1D)

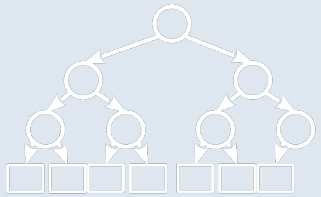
function IntervalQuery($n, [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$)
  $iv \leftarrow \emptyset$
  if $q_{x_1} \leq x_{med}(n) \leq q_{x_2}$ **then**
    $p \leftarrow$ 2DRangeQuery($\tau_{left}(n), (-\infty, q_{x_2}] \times [q_{y_1}, q_{y_2}]$)
    $p \leftarrow p \cup$ 2DRangeQuery($\tau_{right}(n), [q_{x_1}, \infty) \times [q_{y_1}, q_{y_2}]$)
    $iv \leftarrow$ all intervals of the endpoints in p
    $move \leftarrow both$
  else if $x_{med}(n) < q_{x_1}$ **then**
    $p \leftarrow$ 2DRangeQuery($\tau_{right}(n), [q_{x_1}, \infty) \times [q_{y_1}, q_{y_2}]$)
    $iv \leftarrow$ all intervals of the endpoints in p
    $move \leftarrow right$
  else if $q_{x_2} < x_{med}(n)$ **then**
    $p \leftarrow$ 2DRangeQuery($\tau_{left}(n), (-\infty, q_{x_2}] \times [q_{y_1}, q_{y_2}]$)
    $iv \leftarrow$ all intervals of the endpoints in p
    $move \leftarrow left$
  if $move \in \{both, left\}$ and exists $lc \leftarrow leftChild(n)$ **then**
    $iv \leftarrow iv \cup$ IntervalQuery($lc, [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$)
  if $move \in \{both, right\}$ and exists $rc \leftarrow rightChild(n)$ **then**
    $iv \leftarrow iv \cup$ IntervalQuery($rc, [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$)
  return iv

- S obzirom da imamo dvije razine stabala, u prvoj razini je stablo intervala, a u drugoj su dva dvodimenzionalna stabla raspona, kompleksnost pretraživanja
$$O\big((2\log^2 n)(\log n)\big) = O(\log^3 n)$$
- Za neke druge strukture podataka u τ_{left} i τ_{right} to može biti drukčije. Recimo za običnu listu je to $O(n \log n)$

Stablo prioriteta *(priority tree)*

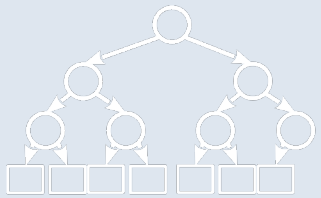


- Korištenje dvodimenzionalnog stabla raspona u stablu intervala čini se neefikasno – dvodimenzionalno stablo raspona ima dva sloja (tri ukupno sa stablom intervala)
- Upiti za stablo intervala su specifični: $(-\infty, q_{x_2}]\times[q_{y_1}, q_{y_2}]$ i $[q_{x_1}, \infty)\times[q_{y_1}, q_{y_2}]$
- Za ovu se namjenu može koristiti gomila *(heap)* kao podatkovna struktura
  - Gomila je sortirana struktura, gdje je korijen gomile najmanji ili najveći član
  - Govorimo o uzlazno ili silazno sortiranoj gomili
- Ukoliko točke sortiramo po x-osi, možemo ih spremiti u gomilu
  - To nam rješava horizontalni dio upita $(-\infty, q_{x_2}]$ i $[q_{x_1}, \infty)$
  - Za $(-\infty, q_{x_2}]$ trebamo uzlazno sortiranu gomilu – spuštamo se od korijena gomile, pa sve do dok ne naiđemo na $x(n) > q_{x_2}$



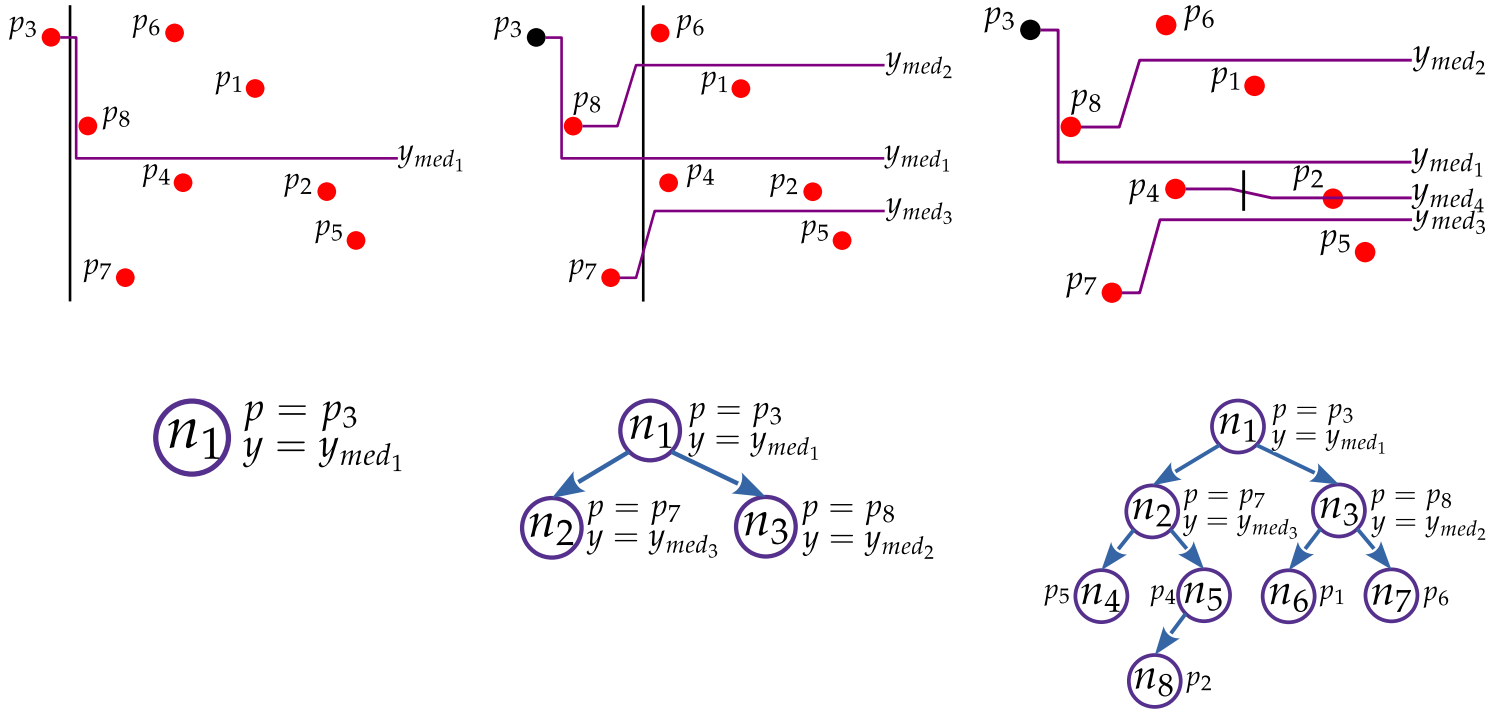
Stablo prioriteta

- Što je s y-osi? Gomila sama po sebi neće podržati tu drugu dimenziju.
- Koristimo činjenicu da particioniranje gomile može biti proizvoljno tako dugo dok je gomila sortirana – roditelj manji ili veći od djece
- U gomilu ugrađujemo koncept binarnog stabla – **stablo prioriteta**
  - Vrijednost čvora $p(n)$ predstavlja točku spremljenu u čvor n – slijedi princip gomile, uvijek uzimamo slijedeću točku s najmanjom $x(p)$ vrijednosti
  - Čvoru dodajemo parametar $y(n)$
    - Struktura stabla prioriteta po parametru $y(n)$ slijedi princip binarnog stabla
      - Lijevo dijete n_L ima parametar $y(n_L) \leq y(n)$ - manji ili jednak od roditelja
      - Desno dijete n_R ima parametar $y(n_R) > y(n)$ - veći od roditelja
    - Kada stvaramo stablo prioriteta i odlučujemo u koje podstablo stavljamo točke
      - Iz skupa točaka maknemo točku čvora
      - Izračunamo medijan y_{med} vrijednosti preostalih točaka – parametar $y(\mathrm{n})$ postavimo na y_{med}
      - Preostale točke s $y(p) \leq y_{med}$ idu u lijevo podstablo
      - Preostale točke s $y(p) > y_{med}$ idu u desno podstablo

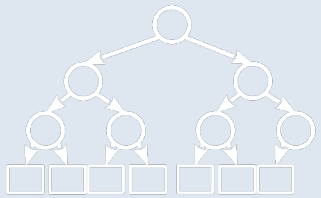


PAZITI! $y(p(n)) \neq y(n)$ – y vrijednost točke u čvoru i y parametar čvora nisu isto!

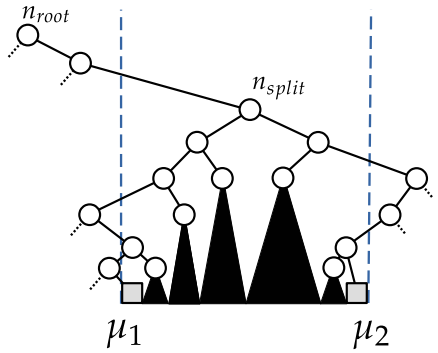
Stablo prioriteta



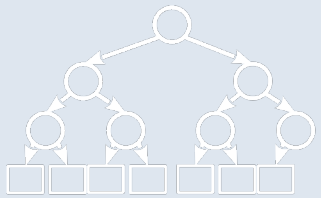

```
function CreatePriorityTree(P)
    p_min = arg min_{p_i ∈ P} x(p_i)
    n ← new node
    p(n) ← p_min
    if P \ p_min ≠ ∅ then
        y_med ← median(P \ p_min)
        P_L ← {p : p ∈ P \ p_min, y(p) ≤ y_med}
        P_R ← {p : p ∈ P \ p_min, y(p) > y_med}
        y(n) ← y_med
        if P_L ≠ ∅ then
            leftChild(n) ← CreatePriorityTree(P_L)
        if P_R ≠ ∅ then
            rightChild(n) ← CreatePriorityTree(P_R)
    return n
```



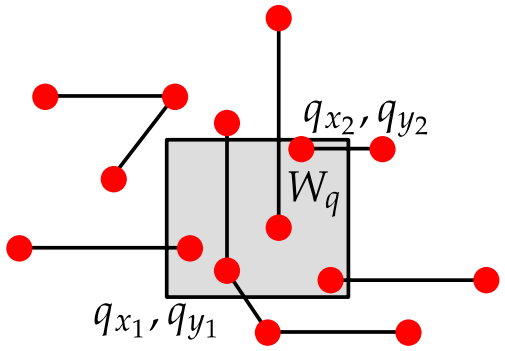
Stablo prioriteta



- S obzirom da y vrijednost točke čvora ($y(p(n))$) stabla nije isto što i y parametar čvora ($y(n)$) – y parametar čvora je korišten u formiranju binarnog stabla
  - $y(p(n))$ se koristi kada se provjerava da li je konkretna točka u prozoru upita, to jest unutar $[q_{y_1}, q_{y_2}]$
  - $y(n)$ se koristi za kretanje po stablu prioriteta
- Potražimo čvor razdvajanja n_{split} – prvi ispod korijena koji je unutar intervala $q_{y_1} \leq y(n_{split}) \leq q_{y_2}$
- Pretraživanje razdvajamo u dvije vertikalne putanje
  - Koncept je isti kao i kao pretraživanja raspona
- Cijelo vrijeme pratimo da je $x(p(n)) \leq q_{x_2}$ - princip gomile
- Za čvorove svih putanja koje prolazimo od korijena, uključujući i čvorove svih podstabala koje automatski dodajemo, svaku točku čvora provjeravamo da li je unutar prozora upita$(-\infty, q_{x_2}] \times [q_{y_1}, q_{y_2}]$



Pretraživanje linijskih segmenata



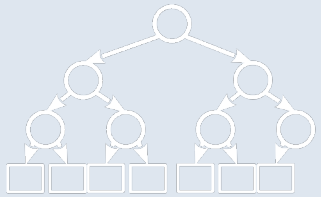
- Zamislimo dvodimenzionalni primjer u kojem umjesto intervala imamo linijske segmente

$$S = \left\{ s_i = \overline{(x_{i_1}, y_{i_1})(x_{i_2}, y_{i_2})} : (x_{i_k}, y_{i_k}) \in \mathbb{R}^2 \right\}$$

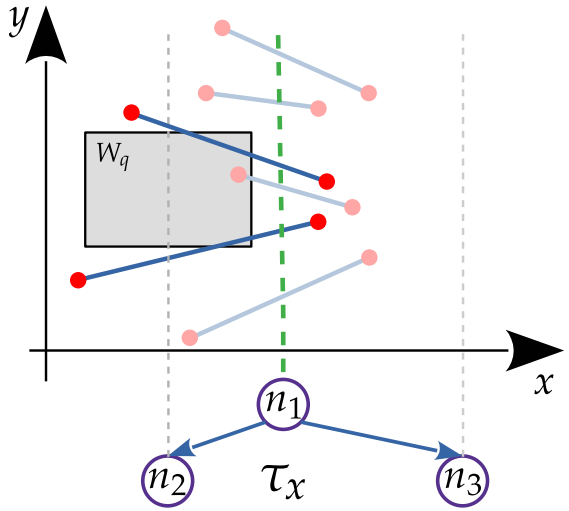
- Zatim definiramo područje pretraživanja

$$W = [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$$

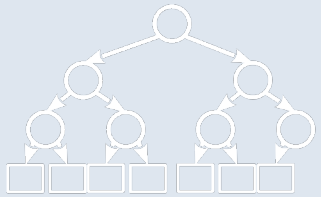
  - Primjer štampane elektroničke pločice na kojoj želimo pronaći sve vodove koji prolaze kroz određeno područje
  - Primjer plana grada na kojem se žele pronaći ulice koje se nalaze unutar nekog područja pretraživanja



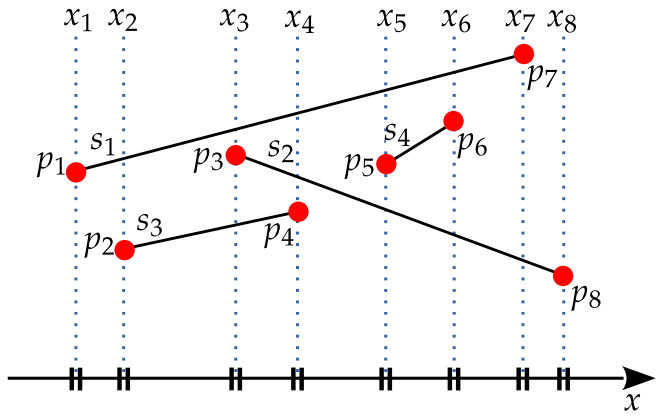
Pretraživanje linijskih segmenata



- Da li za takvo pretraživanje možemo koristiti stablo intervala?
- Zamislimo da imamo prozor upita koji je lijevo od medijana čvora (n_1 na primjeru).
  - Provjeravamo da li su lijevi krajevi linijskih segmenata čvora n_1 unutar $(-\infty, q_{x_2}]\times[q_{y_1}, q_{y_2}]$
  - Na primjeru vidimo barem dva linijska segmenta svojim lijevim krajevima nisu u traženom području, no ipak presijecaju prozor upita W_q - 😕
- Iako će stablo intervala ispravno raditi za većinu slučajeva, neki će ipak proizvesti *false negative* rezultat



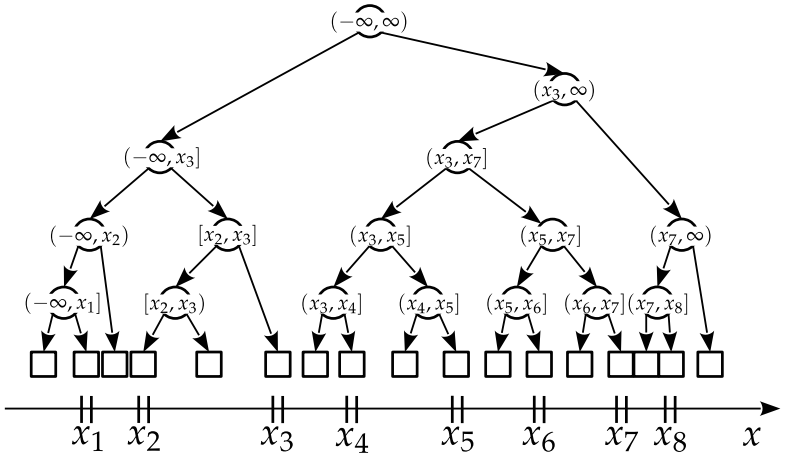
Pretraživanje linijskih segmenata



- Umjesto toga primijenimo pozicijski pristup (*locus approach*)
- Krenemo od x-osi
  - Skup linijskih segmenata razbijemo u elementarne intervale – to su intervali po x-osi u kojima nema promjena broja linijskih segmenata
  - Promjene se dešavaju u krajnjim točkama linijskih segmenata – koristimo se projekcijama krajnjih točaka na x-os : $x_i = x(p_i)$
  - Primjer na slici razbijamo u slijedeće elementarne intervale

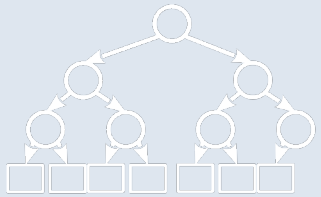
$$(-\infty, x_1), [x_1, x_1], (x_1, x_2), [x_2, x_2], \dots, (x_7, x_8), [x_8, x_8], (x_8, \infty)$$

Pretraživanje linijskih segmenata

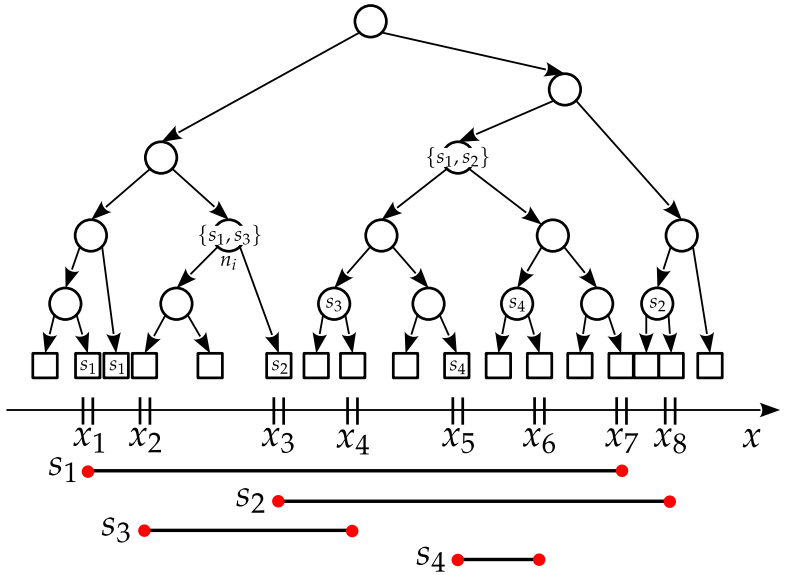


$(-\infty, x_1), [x_1, x_1], (x_1, x_2), [x_2, x_2], \ldots, (x_7, x_8), [x_8, x_8], (x_8, \infty)$

- Svaki od elementarnih intervala pretvorimo u jedan list uravnoteženog binarnog stabla – listovi su vrijednosti (koncept B⁺ indeksnog stabla), a unutarnji čvorovi samo su za navođenje
- Stablo iznad tih listova napravimo na promišljeni način
  - Želimo da stablo bude uravnoteženo kako bi pretraživanje bilo $O(k + \log^2 n)$ - opet se vraćamo na koncept pretraživanja za raspon
- Unutarnji čvorovi stabla agregiraju intervale svojih podstabala $I(n) = I(n_L) \cup I(n_R)$
  - Ovo rezultira time da se intervali u čvorovima iste razine ne preklapaju i nemaju razmaka
- Korijenski čvor agregira sve elementarne intervale, što rezultira sa $(-\infty, \infty)$

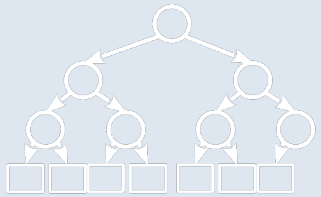


Pretraživanje linijskih segmenata

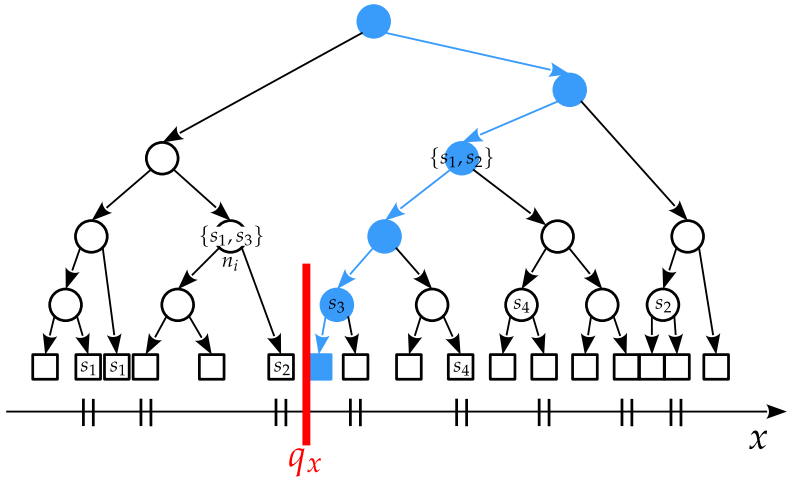



```
procedure InsertSegment(n, [x1, x2])
    if I(n) ⊆ [x1, x2] then
        add [x1, x2] to S(n)
    else
        if I(leftChild(n)) ∩ [x1, x2] ≠ ∅ then
            InsertSegment(leftChild(n), [x1, x2])
        if I(rightChild(n)) ∩ [x1, x2] ≠ ∅ then
            InsertSegment(rightChild(n), [x1, x2])
```

- Zatim pridjeljujemo linijske segmente čvorovima stabla
  - Logika nalaže da se svaki linijski segment u elementarnom intervalu pridijeli svojem listu
  - Moguće je (i vjerojatno) da više listova sadrži isti linijski segment s_i - time uzrokujemo visoku kompleksnost spremanja stabla (*storage complexity*)
  - Takav linijski segment pridijelimo na unutarnji čvor koji sadrži sve listove koji imaju taj linijski segment s_i
  - Korištenjem intervala čvorova $I(n)$, za takvo pridjeljivanje možemo krenuti od korijenskog čvora
  - Za n_i, interval je $I(n_i) = [x_2, x_3]$, što u je u potpunosti sadržano u linijskim segmentima s_1 i s_3
  - $S(n_i) = \{s_1, s_3\}$ predstavlja kanonski skup linijskih segmenata koji u potpunosti prolaze intervalom $I(n_i)$
- Dobivamo **stablo segmenata**



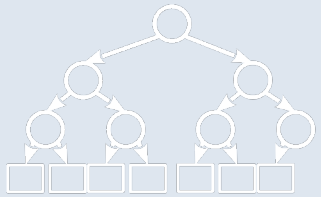
Pretraživanje linijskih segmenata



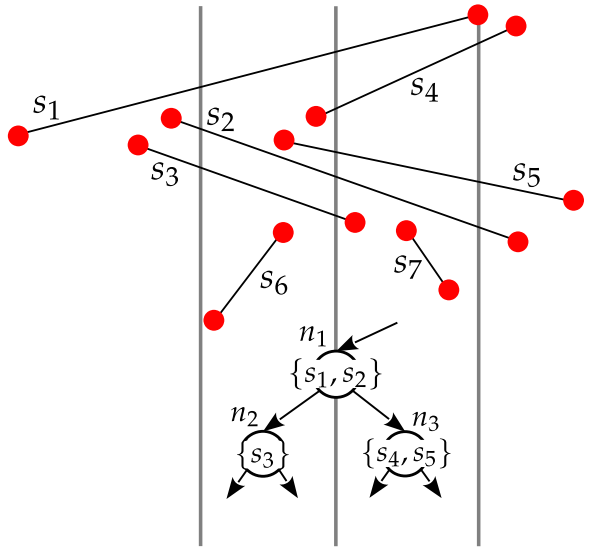
- Ukoliko želimo pronaći linijske segmente iz S koji prolaze kroz $I_q = q_x \times [q_{y_1}, q_{y_2}]$
- Trebamo pronaći elementarni interval kroz koji prolazi q_x
- Izvodimo klasični prolaz kroz stablo, prolazeći kroz čvorove gdje je $q_x \in I(n)$
- List predstavlja traženi elementarni interval
- Vertikalnu putanju od korijenskog čvora do lista označimo kao $\mathcal{V}$
- Skup svih linijskih segmenata koji prolaze kroz q_x su

$$\bigcup_{n \in \mathcal{V}} S(n) \subseteq S$$

- U našem primjeru $\{s_1, s_2, s_3\}$

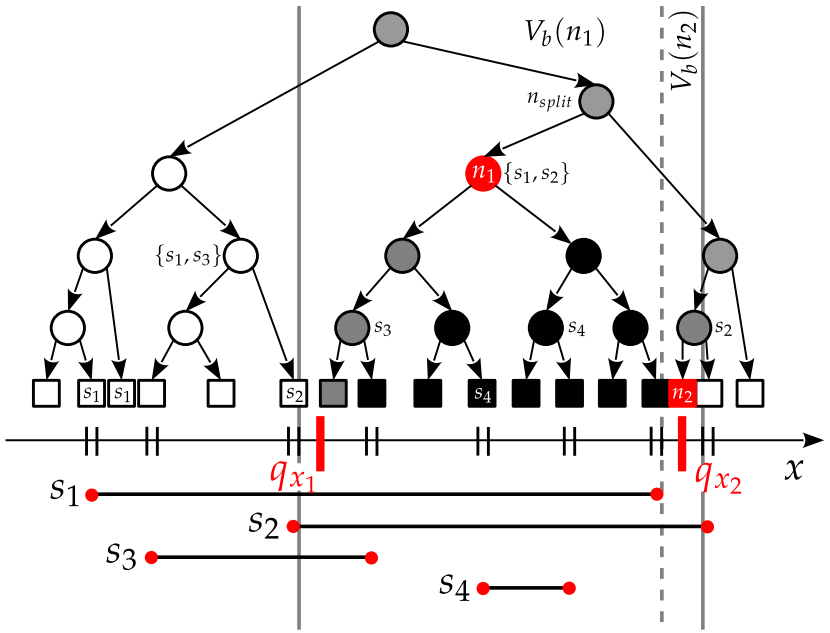


Pretraživanje linijskih segmenata

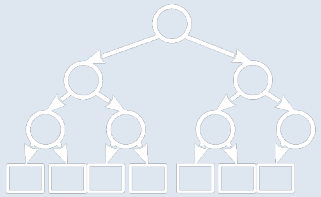


- Ukoliko uzmemo interval nekog čvora $I(n)$ on definira vertikalni blok $V_b(n) = I(n) \times (-\infty, \infty)$
- Svi linijski segmenti u $S(n)$ u potpunosti horizontalno prolaze kroz cijeli $V_b(n)$
- U primjeru na slici primjećujemo i jedno drugo svojstvo koje je vezano uz princip agregacije intervala
  - $V_b(n_1) = V_b(n_2) \cup V_b(n_3)$
  - Linijski segmenti $S(n_1)$ u potpunosti prolaze kroz $V_b(n_1)$, a time i kroz $V_b(n_2)$ i $V_b(n_3)$
  - Linijski segmenti koji u potpunosti prolaze kroz $V_b(n_i)$, gdje je n_i u podstablu čiji je korijen n_j, **djelomično prolaze** kroz $V_b(n_j)$

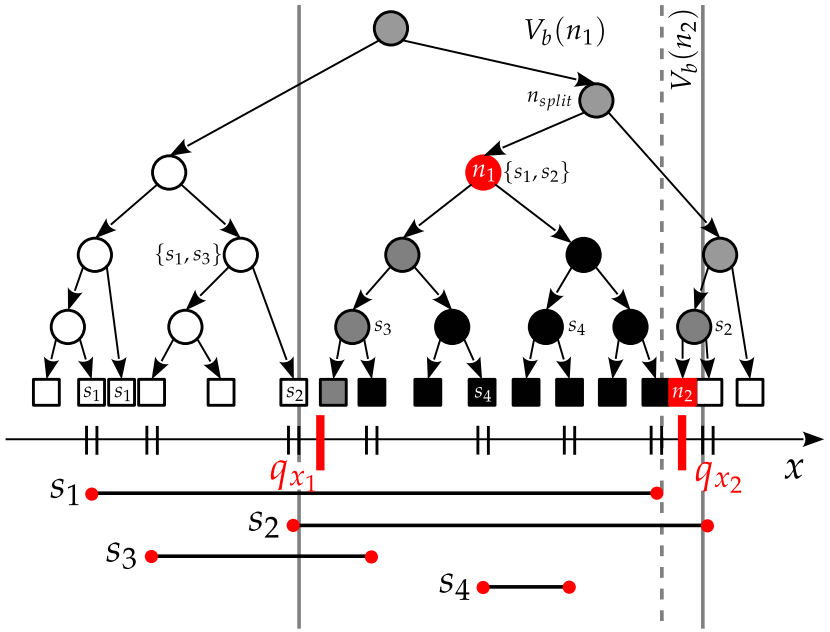
Pretraživanje linijskih segmenata



- Vratimo se pretraživanju prozora $W_q = [q_{x_1}, q_{x_2}] \times [q_{y_1}, q_{y_2}]$
- Koncept je isti kao i kod pretraživanja raspona
- Pronađemo čvor razdvajanja n_{split} koji je u ovom slučaju:
  - Sadrži cijeli interval $I_{q_x} = [q_{x_1}, q_{x_2}]$
  - Lijevo dijete n_L sadrži $q_{x_1} \in I(n_L)$
  - Desno dijete n_R sadrži $q_{x_2} \in I(n_R)$
- Nakon toga dobivamo:
  - Dvije vertikalne putanje, lijevu $\mathcal{V}_L$ i desnu $\mathcal{V}_R$ - u primjeru čvorovi sive boje
  - Skup podstabala N_T koje su sigurno u intervalu I_{q_x} - u primjeru čvorovi crne boje
- Rezultat je skup vertikalnih blokova

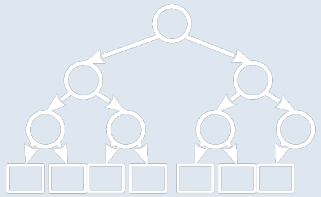


Pretraživanje linijskih segmenata

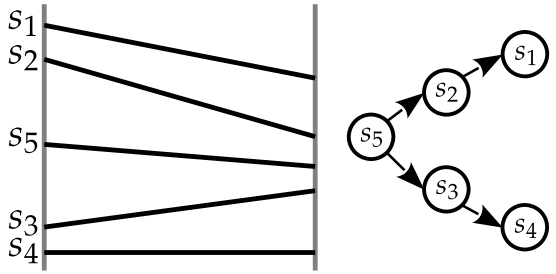


- U skupu vertikalnih blokova koji su rezultat:
  - Možemo imati samo jedan vertikalni blok
    - Kada je n_{split} u elementarnom intervalu – u listu
    - Kada je n_{split} korijen podstabla čiji su svi elementarni intervali u I_{q_x}
  - Prvi i posljednji vertikalni blok sadrže q_{x_1} i q_{x_2}
- Skup linijskih segmenata koji u potpunosti ili djelomično presijeca I_{q_x} je unija svih kanonskih skupova linijskih segmenata podstabala koja čine skup vertikalnih blokova – vidi crvene čvorove

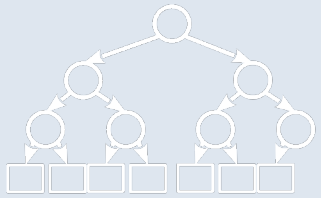
$$\bigcup_{n \in \mathcal{V}_L \cup \mathcal{V}_R \cup N_T} S(n) \subseteq S$$



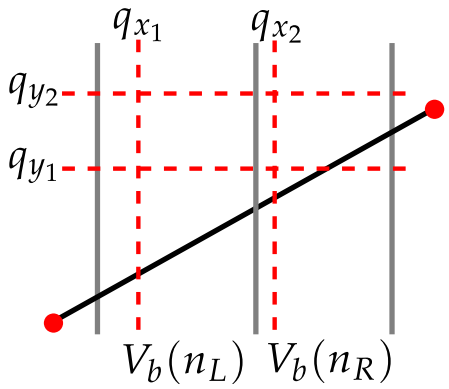
Pretraživanje linijskih segmenata



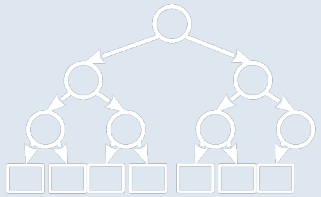
- Što je s vertikalnim pretraživanjem, kako pronalazimo $I_{q_y} = [q_{y_1}, q_{y_2}]$?
- Do sada znamo da kanonski skup linijski segmenata $S(n)$ sadrži linijske segmente koji u potpunosti horizontalno presijecaju vertikalni blok $V_b(n)$
- Ako kanonski skup segmenata organiziramo u uravnoteženo binarno stablo, možemo koristiti isti princip geometrijskog pretraživanja kao i kod raspona
  - Problem: Da bismo jasno znali vertikalni raspored linijskih segmenata, oni se **ne smiju presijecati** !
- Takvo stablo pridružimo čvoru koji ima $S(n) \neq \emptyset$ i označujemo sa $\tau(n)$



Pretraživanje linijskih segmenata



- Prvi i posljednji vertikalni blok sadrže q_{x_1} i q_{x_2}
- U tim vertikalnim blokovima ne smijemo gledati samo mjesto gdje linijski segment presijeca rubove vertikalnog bloka
- Mora se uzeti u obzir sjecište linijskog segmenta sa q_{x_1}, odnosno q_{x_2}, kako ne bi došlo do *false positive* pribrajanja linijskog segmenta konačnom rezultatu



Pretraživanje linijskih segmenata

```
function ReportSubtree(n, W_q = [q_x1, q_x2] × [q_y1, q_y2])
    rv ← ∅
    if n is not nil then
        add VerticalQuery(τ(n), n, W_q) to rv
        add ReportSubtree(leftChild(n), W_q) to rv
        add ReportSubtree(rightChild(n), W_q) to rv
    return rv
function FindSplittingNode(τ, I_q = [q_x1, q_x2])
    n ← root(τ)
    while n is not leaf do
        lc ← leftChild(n), rc ← rightChild(n)
        if q_x1 ∈ I(lc) and q_x2 ∈ I(rc) then
            return n
        else
            if I_q ⊆ I(lc) then
                n ← lc
            else
                n ← rc
    return n
```

```
function SegmentTreeQuery(τ, W_q = [q_x1, q_x2] × [q_y1, q_y2])
    rv ← ∅
    n_split ← FindSplittingNode(τ, [q_x1, q_x2])
    if n_split is leaf then
        if S(n_split) ≠ ∅ then
            add VerticalQuery(τ(n_split), n_split, W_q) to rv
    else
        n ← leftChild(n_split)
        while n is not leaf do
            add VerticalQuery(τ(n), n, W_q) to rv
            lc ← leftChild(n), rc ← rightChild(n)
            if q_x1 ∈ I(lc) then
                add ReportSubtree(rc, W_q) to rv
                n ← lc
            else
                n ← rc
        add VerticalQuery(τ(n), n, W_q) to rv
        n ← rightChild(n_split)
        while n is not leaf do
            add VerticalQuery(τ(n), n, W_q) to rv
            lc ← leftChild(n), rc ← rightChild(n)
            if q_x2 ∈ I(rc) then
                add ReportSubtree(lc, W_q) to rv
                n ← rc
            else
                n ← lc
        add VerticalQuery(τ(n), n, W_q) to rv
    return rv
```

Pitanja