Trinajste vaje APS2: KD-drevo

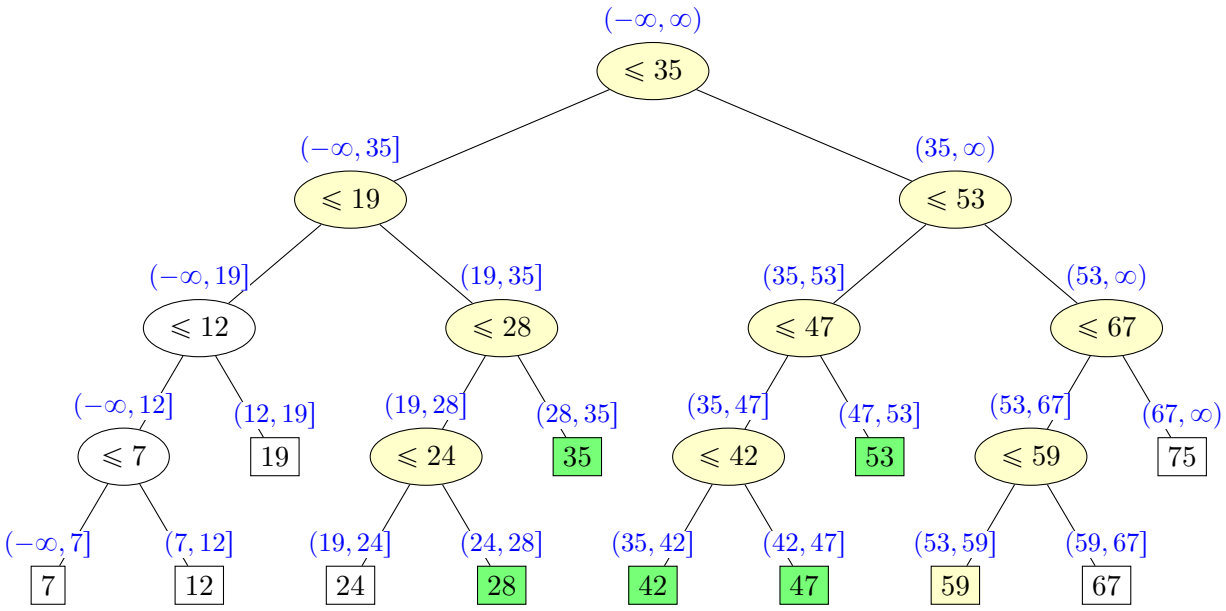
KD-drevo je drevesna podatkovna struktura, ki se uporablja za učinkovito iskanje točk znotraj določenega območja v d-dimenzionalnem prostoru. Pričeli bomo s primerom $d = 1$, nato pa si bomo podrobneje ogledali primer $d = 2$ in nakazali, kako ga lahko posplošimo na večje število dimenzij.

V eni dimenziji

Podana je končna množica podatkovnih točk $P = \{p_1, \ldots, p_n\}$. Predpostavili bomo, da so to kar cela števila, dejansko pa so lahko kakršnikoli elementi, za katere je mogoče definirati popolno urejenost. Skratka, za podano množico P bi radi učinkovito odgovarjali na poizvedbe tipa »poišči vsa števila v množici, ki pripadajo intervalu $[l, r]$«.

Problem je mogoče rešiti preprosto tako, da števila zapišemo v tabelo in jo uredimo. Zatem najprej po dvojiško poiščemo indeks i, na katerem je shranjeno najmanjše število, ki je večje ali enako l, nato pa še indeks j, kjer se nahaja največje število, ki je manjše ali enako r. Nazadnje zgolj izpišemo vsa števila od indeksa i do indeksa j. Časovna zahtevnost je na dlani: $O(\log n + k)$, kjer je $k = j - i + 1$ število števil na intervalu $[l, r]$.

S takšnim pristopom seveda ni nič narobe, kljub temu pa si bomo ogledali rešitev z dvojiškim iskalnim drevesom, saj jo je mogoče posplošiti na več dimenzij. Tokratno drevo bo nekoliko drugačno od tistih, ki smo jih vajeni, saj bodo podatki zapisani zgolj v listih, notranja vozlišča pa bodo namenjena le usmerjanju do podatkov.[1] Drevo bomo zgradili tako, da bo vsako notranje vozlišče hranilo največje število iz njegovega levega poddrevesa. Na primer, slika 1 prikazuje dvojiško iskalno drevo za množico $P = \{7, 12, 19, 24, 28, 35, 42, 47, 53, 59, 67, 75\}$. Leva veja notranjega vozlišča ustreza odgovoru »da« na vprašanje v tem vozlišču, desna veja pa odgovoru »ne«.



Slika 1: *Drevo za iskanje točk na enodimenzionalnem intervalu.*

Drevo za množico P zgradimo s postopkom, ki je prikazan kot algoritem 1. V algoritmu je množica predstavljena kot tabela P; njeni indeksi se pričnejo z 1, podtabela $P[a : b]$ pa zajema elemente od vključno indeksa a do vključno indeksa b. Atribut $v.točka$ predstavlja

[1] Podatki bi lahko bili tudi v notranjih vozliščih, s čimer bi prihranili nekaj prostora, a rešitev, ki jo bomo prikazali, bo lažje posplošiti na več dimenzij.

podatek (točko) v listu v, atributi $v.levo$, $v.desno$ in $v.meja$ pa otroka notranjega vozlišča v in mejni element, ki ločuje elemente v levem poddrevesu vozlišča v od tistih v desnem poddrevesu.

Algoritem gradnje deluje takole:

- Če trenutna podtabela $P[a : b]$ vsebuje en sam element (npr. p), ima drevo eno samo vozlišče (torej list) s podatkom p.
- V nasprotnem primeru najprej poiščemo mediano elementov tabele P (recimo, da je to element m). Drevo nato zgradimo tako, da ustvarimo vozlišče z oznako $\leqslant m$, nato pa na njegovo levo vejo obesimo drevo, ki ga rekurzivno zgradimo nad elementi, ki so manjši ali enaki m, na desno vejo pa drevo, ki ga rekurzivno zgradimo nad elementi, večjimi od m.

Če želimo učinkovito poiskati mediano, morajo biti podatki urejeni. Mediana je potem seveda kar element na sredinskem indeksu.

Algoritem 1 *Gradnja drevesa za množico enodimenzionalnih točk.*

```
function ZGRADI1D(P, a, b)
    ▷ Zgradi drevo za podtabelo P[a : b] in vrne kazalec na njegov koren. ◁
    if a = b then
        ustvari nov list v
        v.točka ← P[a]
        return v
    s ← ⌊(a + b)/2⌋
    ustvari novo notranje vozlišče v
    v.meja ← P[s]
    v.levo ← ZGRADI1D(P, a, s)
    v.desno ← ZGRADI1D(P, s + 1, b)
    return v

function ZGRADI1D(P)
    UREDI(P)
    return ZGRADI1D(P, 1, |P|)
```

Ni težko videti, da za časovno zahtevnost gradnje velja rekurenčna enačba $T(n) = 2T(n/2)+\Theta(1)$. Vrednosti parametrov iz krovnega izreka so $A = 2$, $B = 2$ in $D = 0$ (da ne bi bilo zmešnjave z oznako d, ki predstavlja število dimenzij, uporabljamo velike črke namesto malih). Ker je $A > B^D$, je rešitev $T(n) = \Theta(n^{\log_B A}) = \Theta(n)$. No, če v časovno zahtevnost gradnje vštejemo še čas za urejanje podatkov, dobimo $O(n \log n)$.

Kako pa poiščemo vse točke na intervalu $[l, r]$? Možnih je več pristopov. Eden od njih je predstavljen v knjigi Computational Geometry (M. de Berg *et al.*), osebno pa se mi najenostavnejši zdi postopek, ki je prikazan kot algoritem 2. Parametra l in r predstavljata fiksen interval poizvedbe, parametra l_v in r_v pa interval, ki ga predstavlja trenutno vozlišče (na sliki 1 so ti intervali prikazani z modro). Pri obravnavi trenutnega vozlišča v najprej preverimo, ali ima njegov interval neprazen presek z intervalom $[l, r]$. Če ga nima, ne storimo ničesar, saj vemo, da v poddrevesu vozlišča v ni nobenega elementa znotraj intervala $[l, r]$. V nasprotnem primeru pa preverimo, ali je v list ali notranje vozlišče. Če je list, preverimo njegov podatek in ga izpišemo, če pripada intervalu $[l, r]$. Če je vozlišče v notranje vozlišče, pa rekurzivno obdelamo oba njegova otroka, pri čemer upoštevamo, da otroka predstavljata ožji interval kot trenutno vozlišče.

Na sliki 1 je prikazana situacija pri iskanju podatkov na intervalu $[25, 55]$. Z rumeno ali

zeleno so obarvana vozlišča, pri katerih se interval $[l_v, r_v]$ prekriva z iskalnim intervalom $[l, r]$, z zeleno pa vozlišča, ki jih izpišemo.

Algoritem 2 *Iskanje po drevesu T za množico enodimenzionalnih točk.*

```
function PoiščI1D(v, l, r, l_v, r_v)
    if v obstaja ∧ l_v < r ∧ r_v ⩾ l then
        if v je list then
            if v.točka ⩾ l ∧ v.točka ⩽ r then
                Print(v.točka)
        else
            PoiščI1D(v.levo, l, r, l_v, v.meja)
            PoiščI1D(v.desno, l, r, v.meja, r_v)

function PoiščI1D(T, l, r)
    PoiščI1D(T.koren, l, r, −∞, ∞)
```

Časovna zahtevnost iskanja je $O(\log n + k)$, kjer je k število elementov na intervalu $[l, r]$. Člen k imamo preprosto zato, ker moramo k elementov poiskati in izpisati, člen $\log n$ pa je posledica dejstva, da se moramo v najslabšem primeru po drevesu spustiti vse do listov tudi tedaj, ko noben element ne pripada iskanemu intervalu.

V dveh dimenzijah

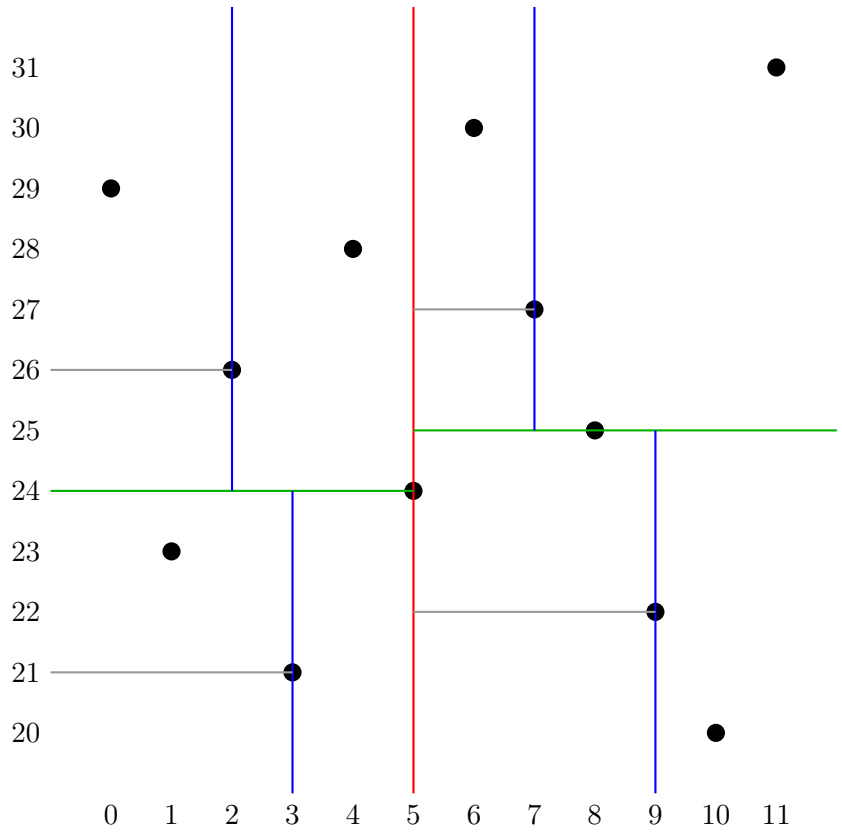
V eni dimenziji se elementi (točke) ločujejo samo po enem atributu (njihovi vrednosti), v dveh dimenzijah pa imamo dva atributa: koordinato x in koordinato y. Ker sta v splošnem obe koordinati enako pomembni, ravnamo takole:

- na globini 0, 2, 4, 6 itd. razdelimo množico točk v (skoraj) enako veliki polovici glede na koordinato x;
- na globini 1, 3, 5, 7 itd. razdelimo množico točk v (skoraj) enako veliki polovici glede na koordinato y.

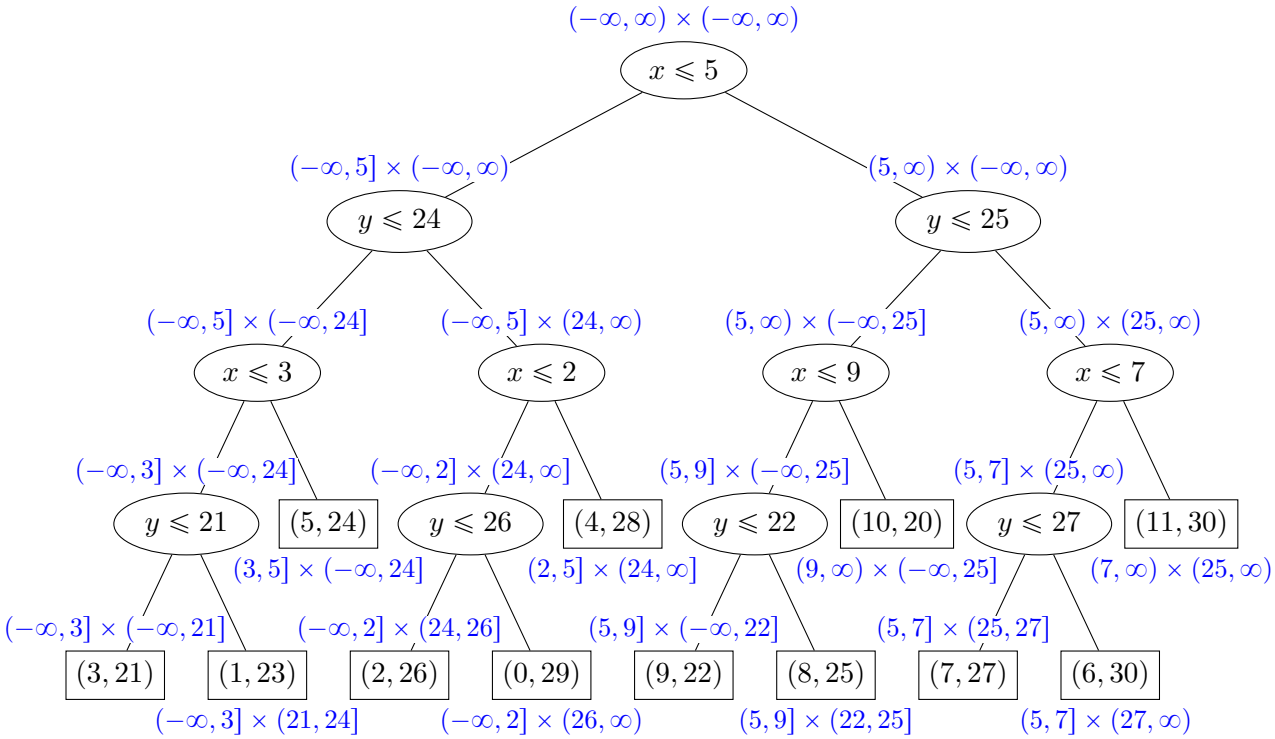
Slika 3 prikazuje dvodimenzionalno KD-drevo za točke na sliki 2. Ta slika prikazuje isto KD-drevo, a na drugačen način. Na primer, levo poddrevo korena vsebuje vse točke, ki so levo od rdeče daljice ali na njej, desno poddrevo pa vse ostale točke. Levo poddrevo levega otroka korena vsebuje vse točke, ki se nahajajo levo od rdeče daljice ali na njej in hkrati pod levo zeleno daljico ali na njej, desno poddrevo pa vse točke, ki se nahajajo levo od rdeče daljice ali na njej in hkrati nad levo zeleno daljico.

Postopek gradnje dvodimenzionalnega KD-drevesa je prikazan kot algoritem 3. Kot vidimo, je algoritem v osnovi podoben njegovi enodimenzionalni različici, dva vidika pa sta nekoliko bolj zapletena:

- V enodimenzionalnem primeru smo točke uredili, nato pa smo v vsakem rekurzivnem klicu v času $O(1)$ poiskali mediano in izvedli delitev na levo in desno podtabelo. V dveh dimenzijah moramo točke na začetku posebej urediti po koordinati x in po koordinati y (izdelamo tabeli X in Y), v vsakem rekurzivnem klicu pa potrebujemo $O(n)$ časa, da na podlagi tabel X in Y izdelamo tabele X_L, X_R, Y_L in Y_R. Ker sta tabeli X in Y urejeni, bodo urejene tudi tabele X_L, X_R, Y_L in Y_R. Na primer, pri izdelavi tabel Y_L in Y_R na sodi globini točke iz tabele Y enostavno obravnavamo po vrsti in nastali tabeli bosta urejeni brez dodatnega dela.
- V dvodimenzionalnem primeru lahko zabredemo v težave, če ima več točk isto koordinato x ali isto koordinato y. (Kako bi, denimo, izvedli delitev na sodi globini, če



Slika 2: *Množica točk in njena razdelitev glede na KD-drevo na sliki 3. Rdeča daljica pripada globini 0, zeleni pripadata globini 1, modre pripadajo globini 2, sive pa globini 3. Točke na navpičnih daljicah pripadajo levim odsekom, točke na vodoravnih pa spodnjim odsekom.*



Slika 3: *Dvodimenzionalno KD-drevo za množico točk na sliki 2.*

imajo vse točke isto koordinato x?) Zato bomo zaenkrat predpostavili, da noben par točk nima iste koordinate x in da noben par točk nima iste koordinate y, kasneje pa bomo nakazali, kako lahko to omejitev odpravimo.

Algoritem 3 *Gradnja KD-drevesa za množico dvodimenzionalnih točk, predstavljeno s tabelo P.*

```
function ZGRADI2D(X, Y, g)                          ▷ g je trenutna globina
    ▷ Zgradi drevo za točke v tabeli X oz. Y. Obe tabeli vsebujeta iste točke, le da
      so v tabeli X urejene po koordinati x, v tabeli Y pa po koordinati y. Indeksi se
      pričnejo z 1.                                                          ◁
    n ← |X|
    if n = 1 then
        ustvari nov list v
        v.točka ← X[0]
        return v
    s ← ⌈n/2⌉
    m ← X[s]
    if g mod 2 = 0 then
        X_L = X[1 : s]
        X_R = X[s + 1 : n]
        Y_L = tabela s točkami p ∈ Y, za katere velja p.x ⩽ m
        Y_R = tabela s točkami p ∈ Y, za katere velja p.x > m
    else
        Y_L = Y[1 : s]
        Y_R = Y[s + 1 : n]
        X_L = tabela s točkami p ∈ X, za katere velja p.y ⩽ m
        X_R = tabela s točkami p ∈ X, za katere velja p.y > m
    ustvari novo notranje vozlišče v
    v.meja ← m
    v.levo ← ZGRADI2D(X_L, Y_L, g + 1)
    v.desno ← ZGRADI2D(X_R, Y_R, g + 1)
    return v

function ZGRADI2D(P)                                ▷ P je tabela točk (x, y)
    X ← kopija tabele P, urejena po koordinatah x
    Y ← kopija tabele P, urejena po koordinatah y
    return ZGRADI2D(X, Y, 0)
```

Ker v vsakem rekurzivnem klicu potrebujemo $\Theta(n)$ časa, da ugotovimo, katere točke sodijo v katero poddrevo, se rekurenčna enačba sedaj glasi $T(n) = 2T(n/2) + \Theta(n)$, njena rešitev pa je $T(n) = \Theta(n \log n)$ ($A = 2$, $B = 2$, $D = 1$, $A = B^D$). Ta čas vključuje čas, potreben za urejanje točk po koordinati x in po koordinati y.

Postopek iskanja točk na območju S je prikazan kot algoritem 4. Podobno kot v enodimenzionalnem primeru vzdržujemo dvodimenzionalni interval I, ki ga predstavlja trenutno vozlišče (na sliki 3 so ti intervali prikazani z modro). Območje S je v splošnem lahko kakršnekoli oblike, če pa gre za pravokotnik, ni težko preveriti, ali se seka z intervalom trenutnega vozlišča.

Na prvi pogled časovna zahtevnost tudi tokrat znaša $O(\log n + k)$, vendar pa so razmere nekolikanj zapletenejše. Poskušajmo oceniti, koliko intervalov $I(v)$, ki pripadajo posameznim vozliščem v KD-drevesa s korenom K, seka poljubna navpična (ali vodoravna) premica p. Na ta način bomo namreč dobili oceno števila vozlišč, ki jih bomo obiskali, tudi če ne

Algoritem 4 *Iskanje točk v območju S.*

```
function PoiščiKD(v, S, I, g)
    if v obstaja ∧ S in I se sekata then
        if v je list then
            if v.točka ∈ S then
                Print(v.točka)
        else
            if g mod 2 = 0 then
                I_L ← I ∩ (−∞, v.meja] × (−∞, ∞)
                I_R ← I ∩ (v.meja, ∞) × (−∞, ∞)
            else
                I_L ← I ∩ (−∞, ∞) × (−∞, v.meja]
                I_R ← I ∩ (−∞, ∞) × (v.meja, ∞)
            PoiščiKD(v.levo, S, I_L, g + 1)
            PoiščiKD(v.desno, S, I_R, g + 1)
```

najdemo nobene točke znotraj intervala poizvedbe.

Osredotočimo se torej na poljubno navpično premico p. Ta premica seka bodisi interval $I(K.levo)$ bodisi interval $I(K.desno)$. Brez izgube splošnosti predpostavimo, da seka interval $I(K.levo)$. No, če p seka interval $I(K.levo)$, potem seka tudi intervala $I(K.levo.levo)$ in $I(K.levo.desno)$ (torej spodnji in zgornji podinterval intervala $I(K.levo)$; upoštevajmo, da levo poddrevo vozlišča na lihi globini predstavlja zgornji, desno pa spodnji podinterval). To pomeni, da p seka $M_0 = 2$ intervala od štirih na prvih dveh nivojih KD-drevesa, od globine 2 naprej pa se situacija rekurzivno ponovi, saj interval zopet razdelimo z navpično daljico (tako kot na globini 0). Sedaj moramo še ugotoviti, koliko intervalov vozlišč v poddrevesih vozlišč $K.levo.levo$ in $K.levo.desno$ seka premica p. Ker gre za dve vozlišči, od katerih ima vsako $n/4$ vozlišč, je število intervalov, ki jih seka premica p, enako $M(n) = M_0 + 2M(n/4) = 2M(n/4) + 2 = 2M(n/4) + \Theta(1)$, rešitev te rekurenčne enačbe pa je $M(n) = \Theta(n^{\log_B A}) = \Theta(\sqrt{n})$. Časovna zahtevnost iskanja v KD-drevesu potemtakem znaša $O(\sqrt{n} + k)$.

Odprava omejitve glede medsebojne različnosti koordinat

Predpostavili smo, da imajo vse točke medsebojno različne koordinate x in medsebojno različne koordinate y. Ker je ta predpostavka v praksi prehuda omejitev, jo bomo sedaj odpravili.

Naj bo $p = (x, y)$ in $q = (x', y')$. Vemo, da je točka p manjša od točke q po prvi komponenti ($p[1] < q[1]$) le v primeru, če je $x < x'$, v primeru $x = x$ in $y < y'$ (v katerem je točka p prav tako leksikografsko manjša od točke q) pa seveda velja $p[1] = q[1]$. Sedaj bi radi točki p in q pretvorili v točki $\hat{p}$ in $\hat{q}$, tako da bo točka $\hat{p}$ manjša od točke $\hat{q}$ po prvi komponenti ne samo v primeru $x < x'$, ampak tudi v primeru $x = x' \wedge y < y'$. Prav tako bi radi zagotovili, da je točka $\hat{p}$ manjša od točke $\hat{q}$ po drugi komponenti ($\hat{p}[2] < \hat{q}[2]$) tako v primeru $y < y'$ kot v primeru $y = y' \wedge x < x'$. Ta cilj lahko dosežemo s preprostim trikom: vsako točko $p = (x, y)$ pretvorimo v $\hat{p} = ((x|y), (y|x))$, komponente pretvorjenih točk pa primerjamo leksikografsko ($(a|b)$ je samo nekoliko drugačen zapis para (a, b)). Na primer točke $p = (1, 3)$, $q = (1, 4)$ in $r = (2, 3)$ pretvorimo v $\hat{p} = ((1|3), (3|1))$, $\hat{q} = ((1|4), (4|1))$ in $\hat{r} = ((2|3), (3|2))$. Zlahka vidimo, da velja $\hat{p}[1] < \hat{q}[1] < \hat{r}[1]$ in $\hat{p}[2] < \hat{r}[2] < \hat{q}[2]$.

Seveda moramo poleg točk pretvoriti tudi intervale. Interval $I = [x, x'] \times [y, y']$ tako pretvorimo v interval $\hat{I} = [(x|-\infty), (x'|\infty)] \times [(y|-\infty), (y'|\infty)]$. Lahko preverimo, da velja

$(x, y) \in I$ natanko tedaj, ko velja $(\hat{x}, \hat{y}) \in \hat{I}$.

V najmanj dveh dimenzijah

Če imamo d dimenzij, kjer je $d \geqslant 2$, potem na globinah i, $d + i$, $2d + i$ ($i \in \{0, \ldots, d - 1\}$) itd. točke delimo glede na njihove koordinate v dimenziji $i + 1$. Časovna zahtevnost gradnje je še vedno $O(n \log n)$, pri določanju časovne zahtevnosti iskanja pa za število intervalov vozlišč, ki jih seka poljubna navpična premica p, pridelamo rekurenčno enačbo $M(n) = 2^{d-1} + 2^{d-1}M(n/2^d) = 2^{d-1}M(n/2^d) + \Theta(1)$, ki ima rešitev $M(n) = \Theta(n^m)$, kjer je $m = \log_{2^d} 2^{d-1} = \lg 2^{d-1} / \lg 2^d = (d - 1)/d = 1 - 1/d$. Časovna zahtevnost iskanja v d-dimenzionalnem KD-drevesu je torej $O(n^{1-1/d} + k)$.