5 Metody vyhledávání

5.1 Asociativní vyhledávání

5.1.1 sekvenční

Pole se sekvenčně prochází = od začátku do konce prvek po prvku :-)))
Pole nemusí být seřazeno.
K urychlení se používá metoda zarážky. Z cyklu while (A[i] ≠ X & $i < n$) můžeme podmínku $i < n$ přesunout za cyklus a to tak, že na konec pole vložíme hledaný prvek, čili ten se vždy najde a nemůžeme z pole s indexem vyjet. Pokud $i = n$ nalezli jsme vložený prvek a tedy X v poli není. Pokud je $i < n$, hledaný prvek je na pozici i.
$T_{min}(n) = 1$, $T_{max}(n) = O(n)$
$T_{avg}(n) = T_{konst} + T_{VAR} * (n+1)/2 = O(n)$, kde $n+1/2$ značí případ, kdy projdeme polovinu všech dat

Pokud se bude vyhledávat často, vyplatí se pole na začátku seřadit a pak rychle vyhledávat (viz další metody).

5.1.2 binárním půlením

Pole musí být seřazeno.
Rekurzivní alg:

1. vezmu prvek v polovině pole
2. když je to hledaný prvek skončil jsem
3. když je hledaný prvek menší než ten co jsem našel v polovině pole, beru jako nové pole pro hledání už jen pouze levou polovinu a pravou zahodím. Když je prvek větší, tak beru jen pravou polovinu původního pole.
4. pokud je nové pole prázdný interval končíme = prvek v poli není
5. pokračuj rekurzivně 1. krokem

Iterační alg. vypadá tak, že mám index levého a pravého kraje pole (L a P). Jedu v cyklu, když je v půlce větší prvek než hledaný A[(L+P)/2] > X, posunu pravý index pole P = (L+P)/2 – 1. V opačném případě posouvám levý L = (L+P)/2 + 1. Cyklus se opakuje dokud nenaleznu a L>=P
$T_{min}(n) = 1$, $T_{max}(n) = O(\log_2 n)$,
$T_{avg}(n)$ se nedá určit jednoznačně průměrem, protože pravděpodobnost nalezení prvku nepřímoúměrně závisí na počtu prvků prohledávaného pole a ten je v každém kroku o polovinu menší. Platí, že $T_{max}(n) - 1 < T_{avg}(n) < T_{max}(n)$. Prakticky to znamená, že se hledaný prvek nalezne v posledním kroku půlení. Toho využil Dijkstra a z cyklu vyhodil test na nalezení prvku A[(L+P)/2] = X (test se přesunul za cyklus = pokud X v poli bylo, je na A[L] nebo A[P] podle toho jaká byla použita podmínka pro posun L a P při půlení) tím se počet operací v cyklu snížil o třetinu a celková časová složitost klesla (nikoliv však řádově).

5.1.3 interpolační

Používá se v setříděné posloupnosti se znalostí pravděpodobnosti rozložení jednotlivých hodnot (interpolační předpokládá rovnoměrné rozložení hodnot, ale může být i jinak). Princip je podobný jako v binárním půlení, jen s tím rozdílem, že hodnotu netestuji v polovině pole, ale v jiném zlomku. Pro rovnoměrné rozložení na indexu: $(X - A[L]) \cdot \frac{(R-L)}{(A[R]-A[L])}$. Ten zlomek udává „hustotu“. Snažíme se to tedy rozdělit někde tam, kde se asi bude nacházet ten prvek s hodnotou X. Složitost je pak $\log_2 \log_2 n$.

5.1.4 binární vyhledávací stromy

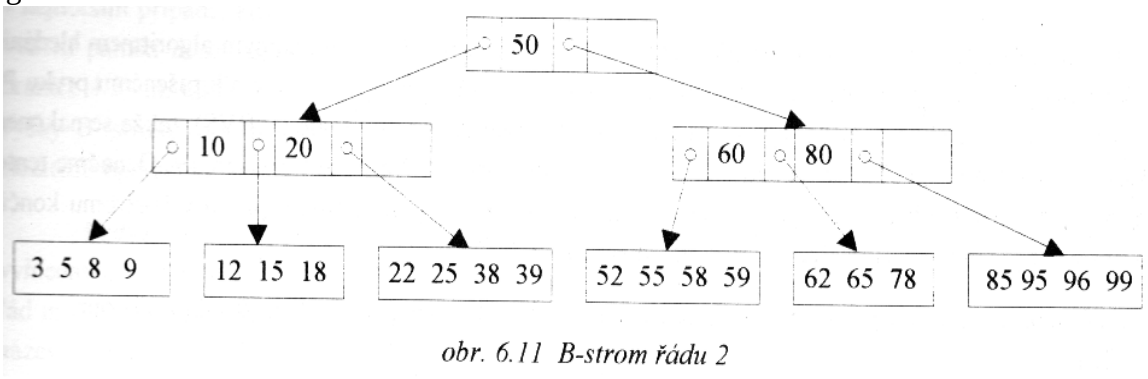
Binární vyhledávací strom je kořenový uzlově ohodnocený binární strom takový, že pro každý vnitřní uzel v s levým synem v_L a pravým synem v_P platí: $H(v_L) <= H(v) <= H(v_P)$. $H(v)$ je ohodnocení uzlu v.
Hledání je asi jasné po 5 letech na FELu ne? To nebudu popisovat :-)))
$T_{min}(n) = 1$, $T_{max}(n) = T_{avg}(n) = O(\log_2 n)$
Toto však platí jen pro výškově vyvážené stromy. Těch je několik: (AVL, B, 1-2)

B-strom je stromová datová struktura, používaná pro uchovávání dat a vyhledávání v nich. Operace přidání, vyjmutí i vyhledávání probíhají v logaritmicky omezeném čase. Tato struktura je často využívána pro databázové aplikace. B-strom je speciální případ (a,b)-stromu, který poskytuje větší volnost ve volbě minimálního a maximálního počtu potomků.

Def: B-stromem stupně m nad vyhledávacím prostorem $S = \{p_1, ..., p_n\}$ rozumíme strom, splňující následující podmínky:

1. Každý uzel kromě kořene a listu má nejméně *m* a nejvýše *2m* prvků p_i.
2. Všechny cesty od kořene k listům jsou stejně dlouhé.
3. Pro každé p_j v podstromu odpovídajícímu odkazu $p_i - 1$, platí $p_j < p_i$
4. Pro každé p_k v podstromu odpovídajícím odkazu p_i, platí $p_k > p_i$

V principu se jedná o (*2m*)-ární strom, kde jednotlivé uzly obsahují minimálně *m* a maximálně 2*m* prvků. Každý uzel obsahuje *x* + 1 ukazatelů na podstromy nižší úrovně, kde *x* je počet prvků p_i v uzlu.
Strom je organizován takto:

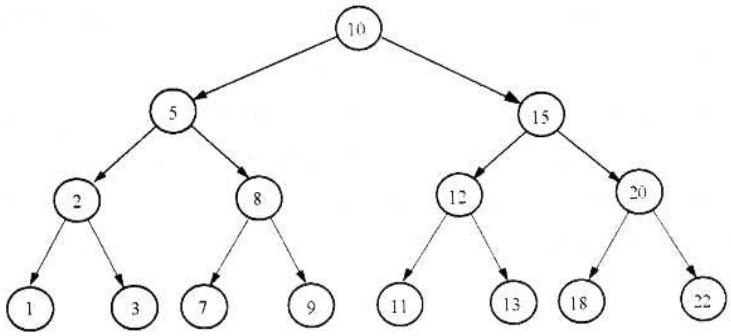


obr. 6.11 B-strom řádu 2

Pěkný animovaný obrázek jak se v něm hledá je na http://www.bluerwhite.org/btree/

Problém je při jejich tvorbě, aby byly vyvážené. U AVL se používá jednoduchá a dvojitá rotace. Vyvažování je úplně brutální na popis, to snad ani nemůžou chtít.

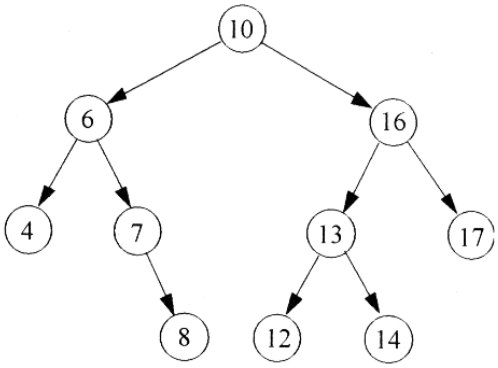
BVS:



obr. 4.2 BVS množiny S = { 1, 2, 3, 5, 7, 8, 9, 10, 11, 12, 13, 15, 18, 20, 22 }

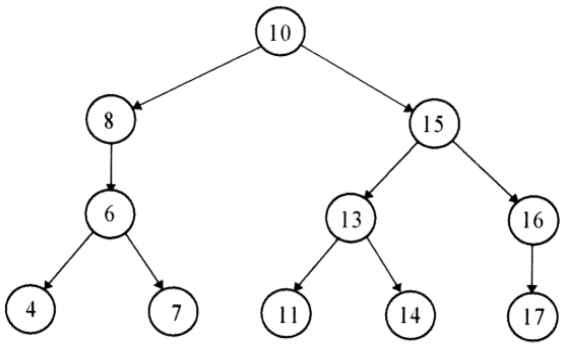
- reprezentují uspořádanou množinu S⊆U, S={k1,k2,...,kn}, je kořenově ohodnocený binární strom s množinou uzlů V, |V|=na ohodnocením β takovým, že: 1) $\beta(v)=k_i$, $v \in V$, $k_i \in S$; 2) Pro každý uzel *v*, který není listem a pro každý uzel v_L, resp. v_P v levém resp. v pravém podstromu s kořenem v platí $\beta(v_L) \leq \beta(v) \leq \beta(v_P)$.
Rušení uzlu: 1) uzel má 1 syna – rušený uzel nahradíme synem. 2) uzel má 2 syny – nahrazujeme nejbližším nižším prvkem (prvkem z nejpravější větve levého podstromu) nebo nejbližším vyšším (z nejlevější větve pravého podstromu)
Složitost: čas. slož. NAJDI,VLOZ,ZRUS v nejhorším případě dána délkou nejdelší větve v BVS. U pravidelného BVS – $Q(n)=D(n)=I(n) \approx \log_2(n+1)$

AVL-strom (71): Výškově vyvážený strom. BVS je AVL právě tehdy, když se výšky levého a pravého podstromu každého uzlu liší nejvýše o 1. Výška AVL stromu je max. o 45% větší než výška BVS.



obr. 4.5 Výškově vyvážený AVL strom

1-2 strom (74): klasický BVS s 1) všechny listy mají stejnou vzdálenost od kořene; 2) uzel, který má jednoho syna, má „bratra“ se dvěma syny.



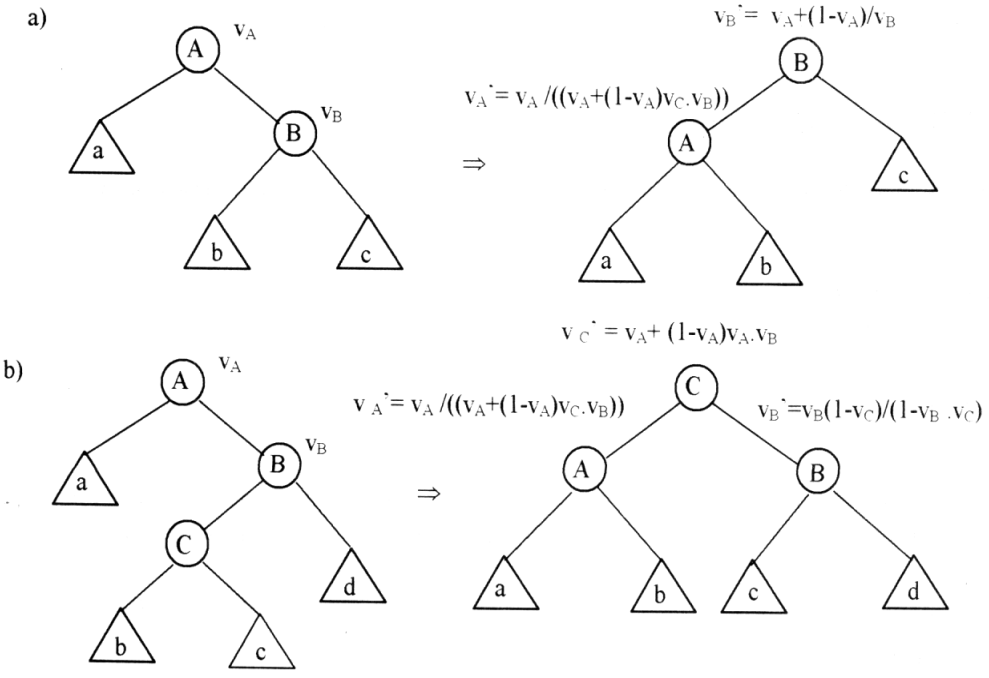
obr. 4.8 1-2 strom

Váhově vyvážené stromy. Spočívá ve vyvážení počtu uzlů levého a pravého podstromu každého uzlu.
Váha uzlu se definuje jako:

1. v(u) = ½ pro list
2. v(u) = (1 + počet_uzlů_levého_podstromu) / (1 + počet_uzlů_stromu_s_kořenem_u)

BVS má ohraničené vyváření a, 0 <= a <= 0.5, jestliže pro každý uzel platí a <= v(u) <= 1-a
Ideálně vyvážený strom má a = ½ (pak se jedná o „pravidelný“ BVS). Např. kořen na obrázku 4.8 (1-2 stromy) má váhu v(u_{10}) = 5/12 nebo v(u_{16}) = 1/3, v(u_8) = 4/5.

Váhově vyvážený strom (75): vyvážení počtu uzlů levého a pravého podstromu.



obr. 4.9 Transformace na váhově vyvážených stromech a) jednoduchá rotace, b) dvojitá rotace

Ohraničené vyvážení α, $0 \le \alpha \le 0.5$, jestliže pro každý uzel platí: $\alpha \le \rho(u) \le 1 - \alpha$
Složitost hledání max.2x ideálně vět. BVS. Čím menší α tím míň vyvažování, ale delší hledání.

5.2 Adresní vyhledávání

Adresní metody jsou založeny na výpočtu adresy hledaného prvku z klíče.

5.2.1 přímý přístup

Def: Nechť vyhledávací prostor S je v souvislém intervalu adres A[a_{min}, ..., a_{max}] a hodnoty klíčů tvoří souvislý interval K[k_{min}, ..., k_{max}]. Pak zřejmě můžeme nalézt funkci f: K–>A, f(k)=a, která jednoznačně přidělí klíči z K adresu z A.
f je např: f(k) = (k - k_{min})+a_{min}

Takovouto implementaci nazýváme přímým přístupem. Je to ideální implementace neboť všechny operace (najdi,vlož,smaž) mají konstantní časovou složitost T(n) = O(1). Bohužel reálné úlohy málokdy vyhovují podmínkám přímého přístupu neboť rozsah identifikačních klíčů (mohutnost universa) je mnohem větší než adresní prostor.

5.2.2 rozptylovací funkce

Def: Nechť K je množina klíčů a A je souvislý adresní prostor. |K| >> |A|. Pak zobrazeni h(k) = a, $k \in K$, $a \in A$, nazveme **rozptylovací funkcí** (hash function = hešovací funkce) . Různé klíče k_1 a k_2 pro něž platí h(k_1) = h(k_2) nazveme **synonymy** a jejich zobrazení **kolizí**.

Problém je jak navrhnout hashovací funkci aby generovala co nejméně synonym při optimalnim využití paměťového prostoru.

5.2.3 zřetězené a otevřené rozptylování

Kolize se řeší v zásadě 2 způsoby:
Zřetězením synonym (pole neobsahuje prvek, ale frontu prvků)
$T_{max}(n) = O(n)$ = v nejhorším případě rozptylovací fce. generuje samá stejná synonyma = úloha přejde v případ hledání v seznamu.
Vyhledavaci prostor je reprezentovan řetezy synonym a polem V. První prvek h(k)=a se nachazi v prvku V(a).
Alg. hledani: 1) vypocet rozptyl. fce h(k) $\Rightarrow$ adresa zacatku retezu, 2) vyhledani prvku v retezu. Časová složitost bude dána časem výpočtu funkce h(k) a časem k nalezení. Asympt. Složitost operace je v $Q_{max}(n)$ = O(n), n=|S|
Pruměrná: při rovnoměrném pokryti budou mit vsechny retezy delku n/m. Q(n)=t_h+t_c(n/2m), asymptoticky:
$Q_{avg}(n)=Q(t_n+t_c n/2m)=O(n)$. Prumerna delka synonym **= poměr plnění $\beta=n/m$**, m=poc. retezu synonym (velikost V), n=počet klicu, napr. β=0.5 bude Q_{avg}= 1.25

Otevřeným rozptylováním = rozptylování s otevřenou adresací (prvky se sekvenčně ukládají do pole na první volnou pozici) hešovací fce není již proměnou klíče, ale ještě kroku opakování. Když chci vložit prvek a zjistím, že už tam je synonymum, zvednu i a znovu počítám hash fci. Nejjednodušší fce vypadá takto:
h(k,i) = (k + i) mod m
někdy také h(k,i) = (h_1(k) + i) mod m
nejčastěji však h(k,i) = (h_1(k) + i* h_2(k)) mod m např: h(k,i) = ((k mod 7) + 3i) mod 7

Průměrná složitost vyhledání s otevřeným rozptylováním v poli s plněním **$\beta=n/m$** je:
O(T_{avg}(n,m)) = 1/β * ln(1 / (1 - β))

Výhody: Není potřeba dynamicky přidělovat paměť, vytvářet nový prvky pro reprezentaci zřetězeného seznamu. Vystačíme si pouze s polem V.
Nevýhody: Horší T_{avg} , Počet klíčů je omezen m (velikostí V)

5.3 Vícerozměrné vyhledávání

K - rozměrný vyhledávací prostor:
Nechť U = U1 x U2 x ... x Uk, |Ui| = u_i , i=1...k
K-rozměrný vyhledávací prostor je množina S = {p_1, ... , p_k}, p_j = [k_{1j}, ..., k_{kj}, h_j], k_{ij} je z U_i , j = 1...n, i=1...k
K = rozměr vyhl.prostoru
n = mohutnost vyhl.prostoru
k_{ij} = i-tý klíč prvku p_j

h_j = hodnota prvku p_j

Lexikografické uspořádání
Porovnávám klíčů od začátku dokud jsou stejné, až na pozici *i* narazím na různé klíče, tak *p* je menší nežli *q* ⇔ klíč p_i je menší nežli klíč q_i
Příklad: „karel“ je menší než „karla“

5.3.1 vyhledávání na úplnou shodu
Dotaz, zda prvek s klíči [k_1 ... k_k] je prvkem vyhled.prostotu S.
Analogie s geometrii = zda je prvek v prostoru

Nejjednodušší je sekvenčně T(*n*,*k*) = O(*n*,*k*)
Za použití lexikografického uspořádání jsme K rozměrný problém převedli na jednorozměrný a můžeme použít metodu binárního půlení.
Pak $T(n,k) = O(k * \log_2 n)$

5.3.2 částečnou shodu
Jsou dány hodnoty některých z klíčů, např. [? k_2 ? k_4], hledají se všechny prvky z S, jejichž klíče jsou shodné se zadanými klíči.
Analogie ve 2D prostoru = [3 ?] všechny body v přímce se souřadnicí $x = 3$

Nejjednodušší je sekvenčně T(*n*,*q*) = O(*n*,*q*), kde *q* = počet zadaných klíčů k porovnávání. Pokud je jedním z daných klíčů první klíč a pole je lexikograficky uspořádané, můžeme k omezení výsledku použít nejprve binární hledání podle prvního klíče a teprve poté z výsledku sekvenčně hledat podle ostatních klíčů.
Pokud na začátku srovnáme jednotlivé klíče odděleně – vytvoříme indexy – můžeme pro každý hledaný klíč samostatně použít binární půlení. Tím dostaneme množinu potencionálních výsledků a z ní pak hledáme průnik přes všechny dané klíče.

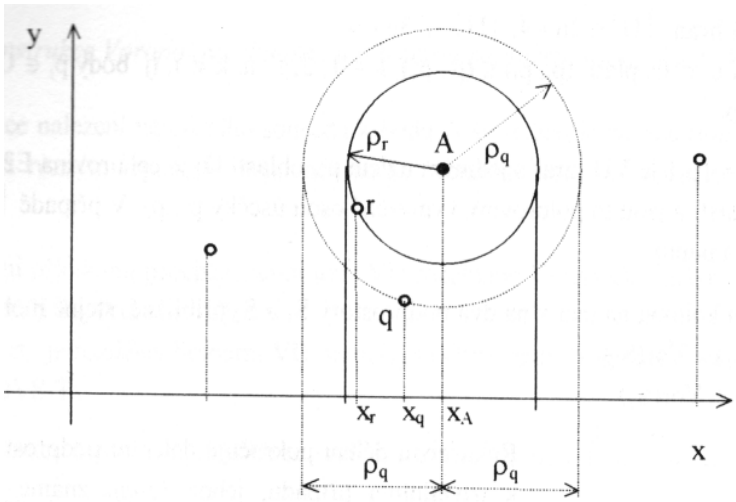
5.3.3 intervalovou shodu
Jsou dány intervaly hodnot klíčů $<k_{min}$; $k_{max}>$ všech nebo opět jen některých klíčů a hledají se prvky S jejichž klíče leží v zadaných intervalech.
Analogie 2D = [<1;3> ; ?] všechny body se souřadnicí *x* mezi 1 a 3 (úsečka)

Nejjednodušší je sekvenčně T(*n*,*q*) = O(*n*,*q*), *q* = počet zadaných intervalů klíčů k porovnávání. Stejné jako na částečnou shodu, akorát výsledek není vždy jeden prvek ale interval. Hledáme buďto stejnou hodnotu nebo pro dolní mez nejbližší vyšší klíč a pro horní nejbližší nižší.

5.3.4 vyhledání nejbližšího souseda
Předpokládejme že na univerzu U je dána metrika ρ, to je relace pro kterou platí: když p_1, p_2, p_3 jsou z U
$\rho(p_1,p_2) = \rho(p_2,p_1)$
$\rho(p_1,p_2) \geq 0$
$\rho(p_1,p_3) \leq \rho(p_1,p_2) + \rho(p_2,p_3)$ (to jest trojúhelníková nerovnost)

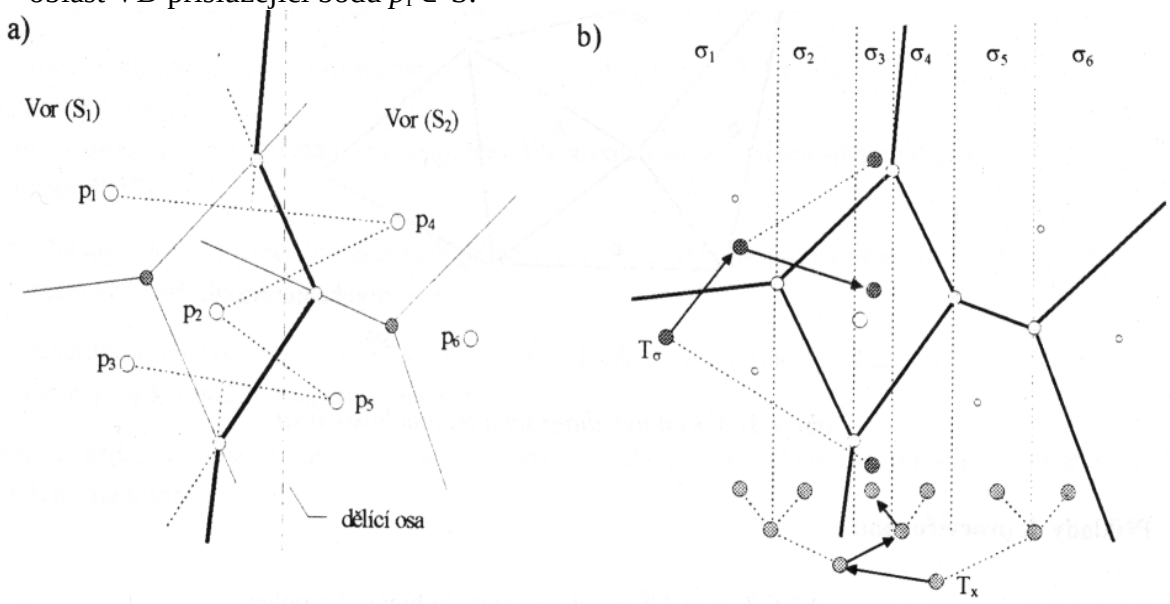
Je dán prvek *p* z U a hledáme prvek *q* z S, který je v dané metrice ρ nejblíže prvku *p*, to jest pro každé $r \neq p$ je $\rho(p,q) \leq \rho(p,r)$

K vyhledávání nejbližšího souseda se používá:
Naivní řešení: hledáme-li nejbližšího souseda uzlu *u* můžeme vypočítat vzdálenost od každého jiného bodu náležejícího naší množině. Složitost takového počínání je O(*n*), kde *n* je počet prvků.
Metoda projekce: každá snaha o urychlení naivního řešení vede před omezení prohledávacího prostoru. Jednou možností je projekce do jedné ze souřadnicových os. Celý algoritmus sestává ze tří kroků. 1) najdeme nejbližšího souseda v projekci do některé ze součadnicových os. 2) vybereme dalšího souseda v projekci x tak, že $(|x_A - x_r| \leq \rho_{min}) \,\&\, (\rho_r < \rho_{min})$. 3. Jestliže takový bod neexistuje, hledaným nejbližším sousedem je poslední nalezený soused z bodu 2. V opačném případě obsahuje graf jediný bod a nejbližší soused neexistuje. První krok realizujeme v $O(\log_2 n)$ krocích. Složitost druhého kroku je konstantní. Celková složitost je tedy O(*k* + $\log_2 n$), kde *k* je počet opakování kroku 2. Zřejmě $k_{min} = 1$ a $k_{max} = n$.



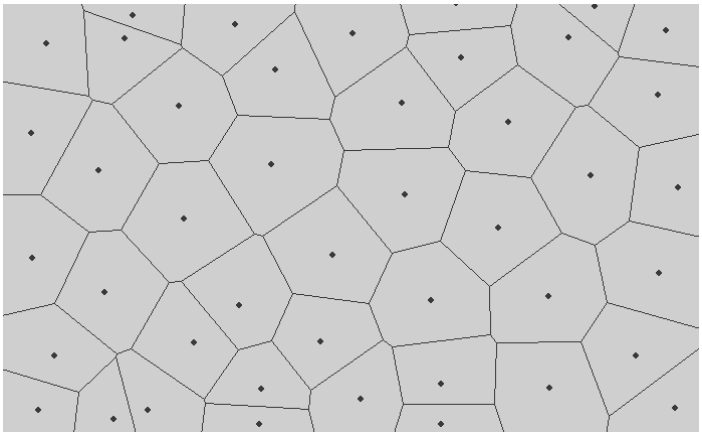
obr.4.8 Hledání nejbližšího souseda metodou projekci

Voroniuv diagram (VD), konstrukce, jak zjistim nejblizsi body noveho vlozeneho bodu S={p_1,...p_n} = mnozina bodu. H(p_i,p_j) = polorovina ohranicena osou usecky p_i,p_j obsahujici bod p_i. Prunik polorovin V(i)=$\cap_{i\neq j}$ H(p_i , p_j) je Voronoiuv polygon. Soustava Vor(S) je VD. Je to 1 planárni graf 2 každá oblast O_i je konvexni 3 obsahuje |V|=n oblastí 4 každý uzel má stupeň alespoň 3 5 obsahuje O(n) uzlů a hran 6 pro všechny body plati (p_i,p_j)<=(p_j,p_k) , k=1,2,..n, $k\neq i$, tj. body $P_j \in O_i$ mají nejbližšího souseda bod p_i .
Konstrukce: 1) rozdělit prostor přímkou kolmou na osu x na s_1 a s_2 stejné mohutnosti 2) rekurzivně vytvoř Vor(s_1) a Vor(s_2) 3) vytvoř Vor(s)=Vor(s_1)+Vor(s_2). **Nejbližši soused**: 1) vertikalnimi primkami prochaz. uzly sestavit pasy a nad nimi vyvazeny BVS Tx 2) každý pas se rozdeli hranami Nad temito castmi sestavit vyvazene BVS Tσ. I) ve stromu Tx vyhledej k danemu Ax pas σi 2) ve stromu Tσ vyhledej část obsahujici bod A=> oblast VD prisluzejici bodu $p_i \in$ S.

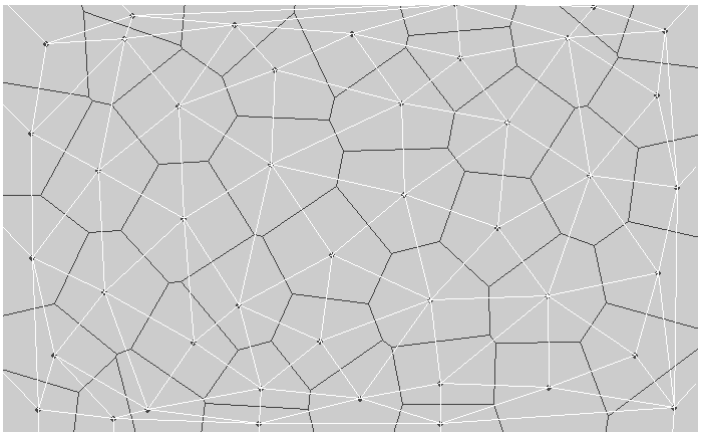


4.30 Konstrukce Voronoiova diagramu a hledání nejbližšího souseda ve Voronoiově diagramu

Pamet slozitost je n^2,
T(n) = $\log_2 n$



Voronoi diagram



Triangulace z voroniova diagramu

5.3.5 vyhledávací stromy pro vícerozměrné vyhledávání

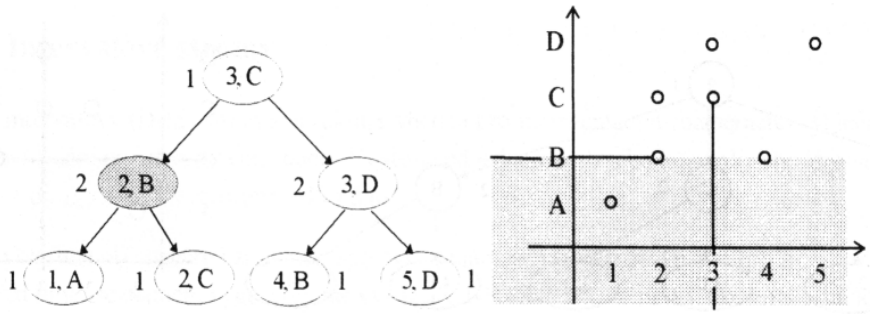
BVS založený na lexikograf.uspoř. – klasický BVS – Dobrý pro hledání na úplnou shodu a na částečnou podle prvního klíče.

$T(n) = \log_2 n$

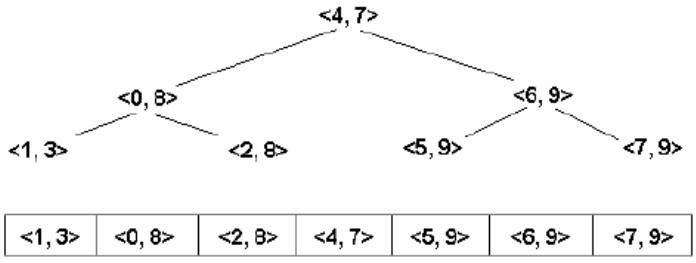
K–BVS strom = při stavbě se porovnávají jen *d*-té složky prvku (*d*-té klíče), této hodnotě *d* se říká **diskriminátor**. Diskriminátor je součástí ohodnocení uzlu a odvozuje se ze vzdálenosti uzlu od kořene L. Ohodnocení uzlu je pak $\gamma(u) = [p_u, d]$, $p_u = [\ [p_1 \ldots p_d \ldots p_k]\ ,\ d\ ] = (L \bmod k) + 1$

V každém hladině stromu se tedy v uzlu rozhodujeme podle *d*-tého klíče. Pokud je v uzlu *d*-tý klíč stejný rozhodujeme se podle dalšího tedy $(d+1 \bmod k)$

Hledání probíhá tak, že hledáme stejné ohodnocení v *d*-tém klíči a když ho naleznem, tak testujeme ostatní klíče.



obr. 4.22 D-BVS strom a odpovidajici dělení prostoru



<1, 3>	<0, 8>	<2, 8>	<4, 7>	<5, 9>	<6, 9>	<7, 9>

2-BVS strom pro body ve 2D

V nejhorším případě, kdy se uzly v každém *d* shodují a průměrně se porovnává $(k+1) / 2$ (polovina) ze zbylých *k* klíčů, je $O(n) = (k+1) / 2 * \log_2(n+1)$

Pokud se však v *d* nikdy neshodují, bude

$O(n) = \log_2(n+1)$

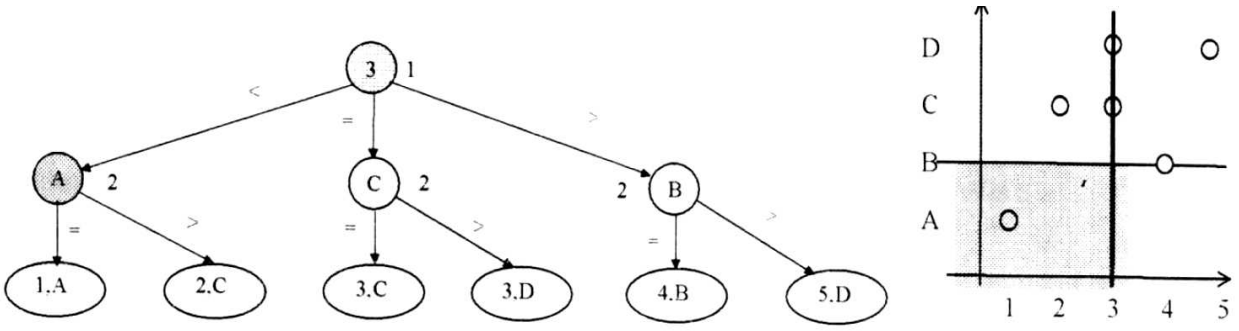
DD-strom (alias KD strom)

Jako K-BVS (D-BVS) se v každé hladině mění diskriminant. Každý uzel má (může mít) 3 syny – menší, větší, roven (menší a větší se označují jako vlastní synové). Listy reprezentují vyhledávací prostor. Směrem k listům se snižuje v menších a větších podstromech mohutnost vyhled. prostoru. V podstromu rovnosti se snižuje dokonce řád.

Pro ideálně vyvážený KD-strom platí hledání na úplnou shodu

$T(n) = O(k + \log_2 n)$

Ideální KD-strom lze zkonstruovat v čase $n \log_2 n$



obr. 4.23 DD-strom a odpovidajici děleni prostoru

Intervalove stromy (K-IS)

Je vhodny pro intervalovou shodu.

Priklad jednorozmerneho intervaloveho stromu (1-IS)

1. vytvorime obousmerne zretezeny spojovy seznam serazenych prvku
2. nad timto seznamem vygenerujeme vyvazeny BVS, prvky seznamu budou listy v case $O(n\log_2 n)$

1-BVS ma paměťovou slozitost $S(n) = O(2n) = O(n)$

Hledani intervalu:

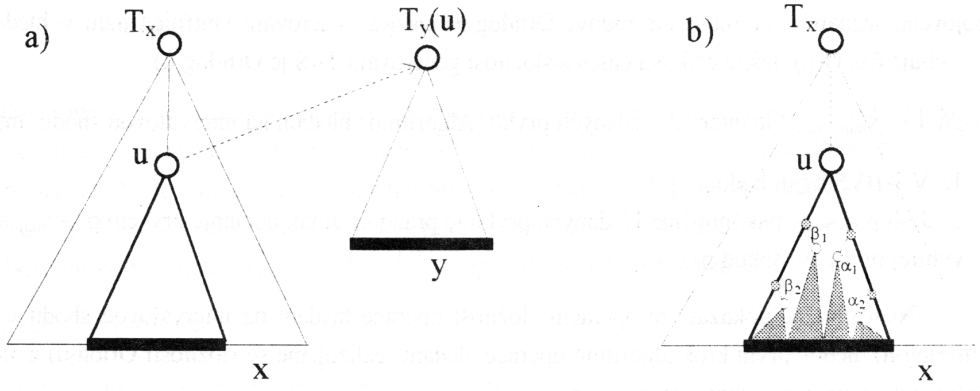
1. Najdi hodnotu $p <= K_{min}$
2. `if` $p > K_{max}$, pak je množina hledaných prvků prázdná, jinak počínaje K_{min} vybíráme prvky do K_{max}

$T(n) = O(m+\log_2 n)$, kde m je pocet nalezenych prvku

Generovani vicerozmerneho K-IS

2-IS:

1. Vygeneruj 1-IS strom T1 dle souradnice K1
2. Kazdemu vnitrnimu uzlu u prirad jednorozmerny IS T2 s korenem v *u* generovany dle K2 nad mnozinou bodu prislusejicich listum podstromu T1 v *u*



obr. 4.25 Dvojrozměrný intervalový strom - a) konstrukce stromu, b) hledání na intervalovou shodu

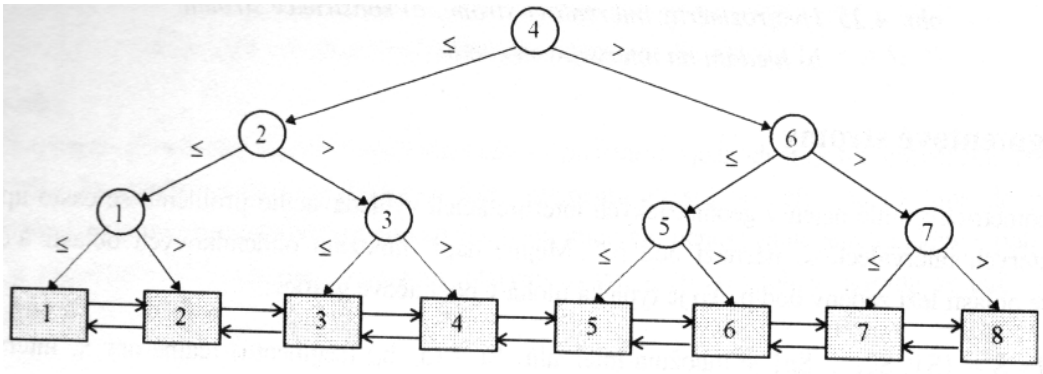
Hledani v K-IS:

1. Vyhledej v T_1 dolni a horni mez intervalu K_{1min},K_{1max} (zajima nas uzel *u*, ve kterem se dělí cesty k K_{1min} a K_{1max})
2. Oznacme A_1 az A_S leve syny uzlu lezicich na ceste k K_{1min} pod uzlem *u*
3. Oznacme B_1 az B_S pravé syny uzlu ležících na ceste k K_{1max} pod uzlem *u*
4. Intervalové stromy $T_2(A_i)$ a $T_2(B_i)$ zaručeně pokrývají interval K_{1min}, K_{1max}
5. Hledej na shodu intervalu K_{2min}, K_{2max} v jednorozmenych stromech $T_2(A_i)$ a $T_2(B_i)$

Pametova slozitost 2-IS je $O(n\log_2 n)$

$T(n)=O(m+\log_2^2 n)$

V obecnem K-IS je $S(n)=O(n*\log_2^{k-1} n)$ a $T(n)=O(m+\log_2^k n)$



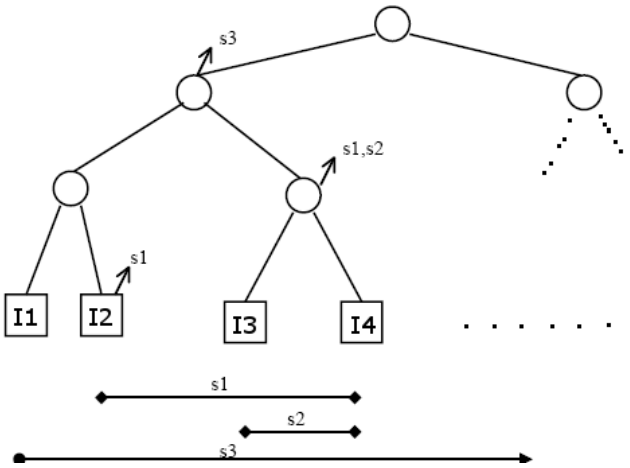
obr. 4.24 Jednorozměrný intervalový strom

Segmentove stromy

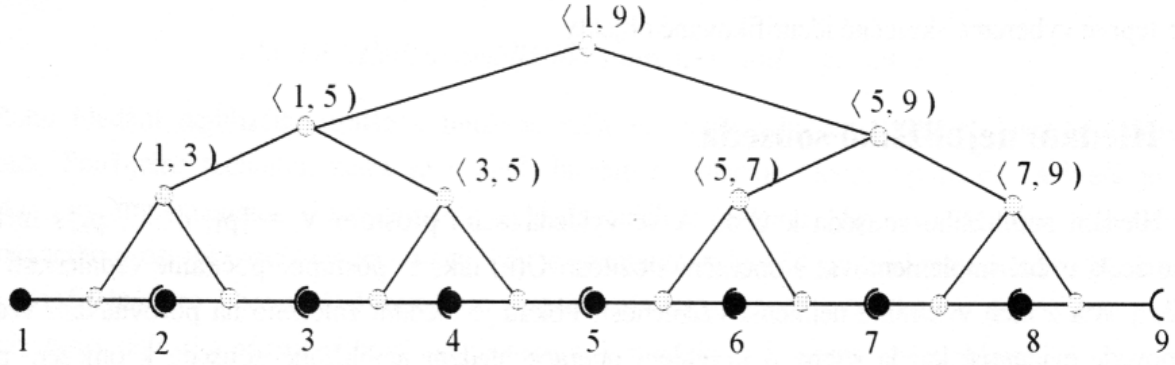
Prevazne v grafice se uplatnuji dotazy „*ve kterych intervalech se nachazi bod P*“ ?

Necht S je mnozina intervalu $<A_i,B_i>$(segmentu) osy *x*, intervaly nemuseji být disjunktni, segmentový strom nad S sestrojime takto:

1. Seřadíme všechny leve i prave koncove body A_i, B_i
2. Oznacime serazenou posloupnost P=(x_1, x_2, ..., x_{2n}), P dělí osu *x* na intervaly
3. Nad intervaly sestrojime vyvazeny BVS tak, že listy stromu budou jednotlive intervaly, kazdemu vnitrnimu uzlu u priradime interval I_u, ktery vznikne sjednocenim intervalu vsech potomku *u*
4. Uzel u ohodnotime mnozinou vsech intervalu z S takových, že
   a) obsahují celý interval I_u
   b) neobsahují celý interval I otce uzlu *u*



Ohodonceni uzlu lidskou reci :-) „Uzel *u* je ohodnocen segmentem s_i pokud je v nem cely obsazen, ale zaroven otec uzlu *u* v segmentu s_i celý obsazen neni. Např I_2 je cely v s_1 a s_3. Je však ohodnocen pouze s_1 protože jeho otec cely v s_1 neni. Nemuze byt ohodnocen s_3 protoze otec je cely v s_3 (ba dokonce jeho děd :-). To ze je cely obsazen v segmentu s_i znamena, ze jeho nejlevejsi a nejpravejsi list jsou intervaly v segmentu s_i.”



obr. 4.27 BVS nad normalizovanými intervaly

5.4 Operační a paměťová složitost algoritmů vyhledávání

	$O(T_{avg}(n))$	$O(S(n))$
Asociativni		
sekvencni	N	N
bin. Puleni	$\log_2 n$	N
Interpolacni	$\log_2 \log_2 n$	N
Adresni		
Prima adresace	1 = konst.	|U| = cele universum = vsechny mozne klice
Zretezene rozptyl.	**n/2m=O(n)**	n + m*velikost(ukazatele)
Otevrene rozptyl.	$1/\beta * \ln(1/(1-\beta))$	m = pokryje pouze m klicu
Vicerozmerne	$O(T_{avg}(n,k))$	O(S(n,k))
Uplna shoda		
Sekvencni	n,k	N
Binarne podle lexikal. Usporadani	$k*\log_2 n$	N
Castecnou shodu		
Sekvencne	n,q (q=pocet zadanych klicu)	N
intervalovou shodu		
Sekvencne	n,q (q=pocet zadanych intervalu)	N
nejbližšího souseda		
Naivní řešení	*n*	
Projekce	$k + \log_2 n$	
Voroniuv diagram	$\log_2 n$	n^2
stromy		
K-strom (D-strom)	$\log_2(n+1)$	???
KD-strom (DD-tree)	$k+ \log_2 n$	???
K-IS intervalovy	$m+\log_2^k n$	$n*\log_2^{k-1} n$
Segmentovy strom	$q+\log_2 n$ (q je pocet nalezenych reseni)	$n* \log_2 n$