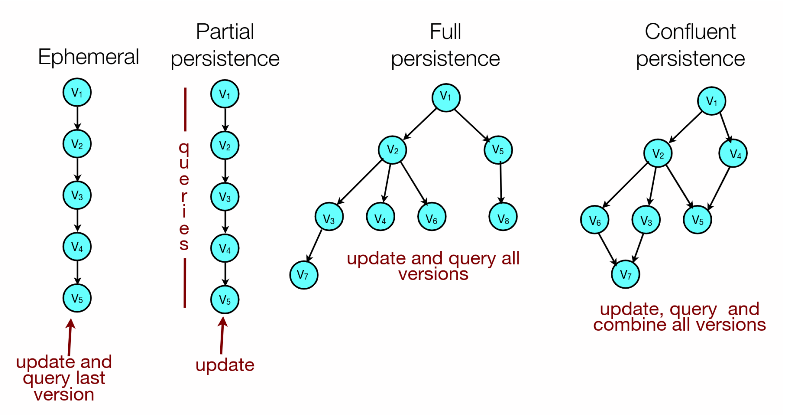
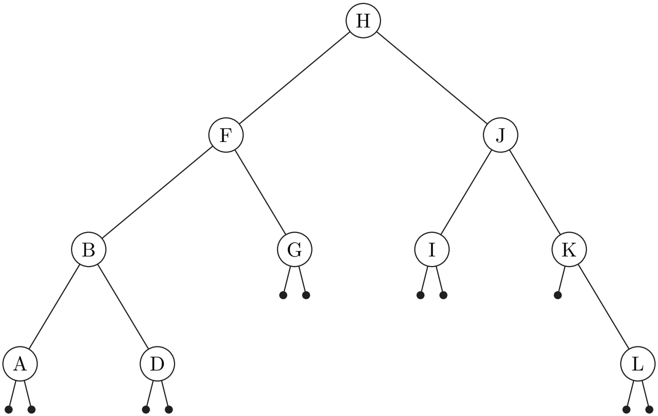
Perzistentné DŠ

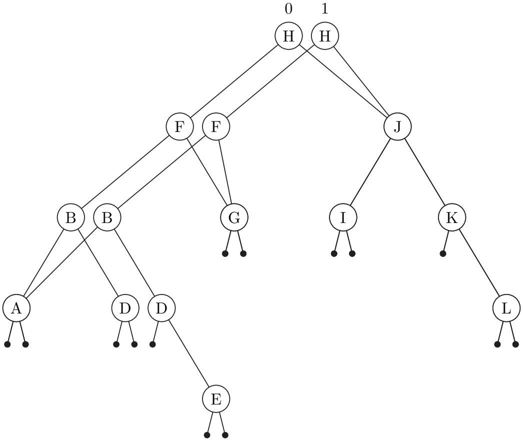
kuko

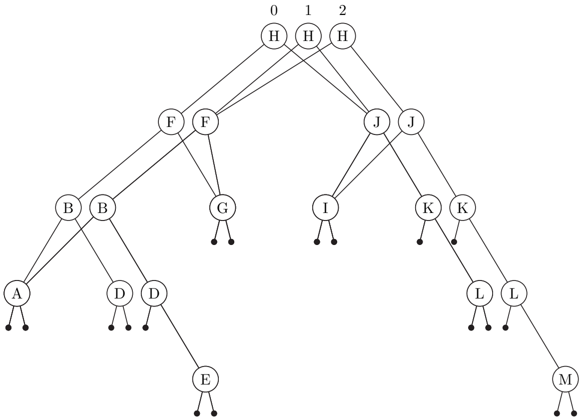
14.10.2020

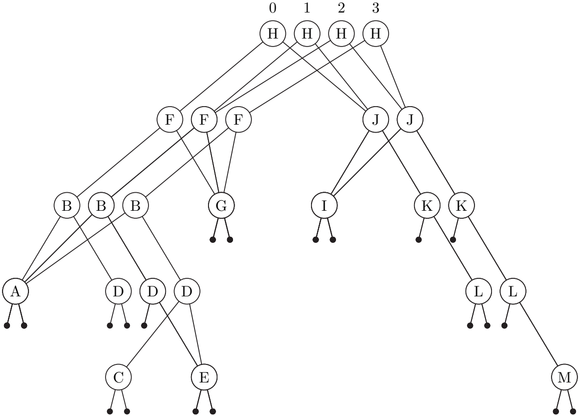
Vybrané partie z dátových štruktúr


Ephemeral
Partial persistence
Full persistence
Confluent persistence
V1
V2
V3
V4
V5
V6
V7
V8
queries
update and query last version
update
update and query all versions
update, query and combine all versions

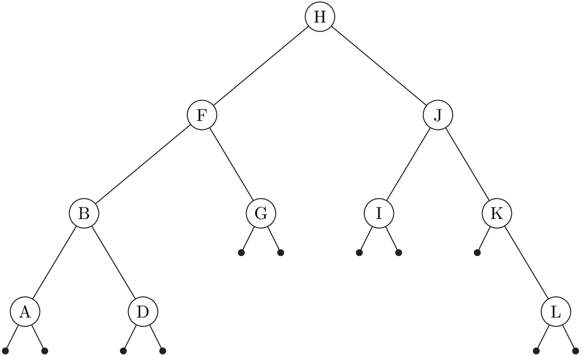

H
F
J
B
G
I
K
A
D
L

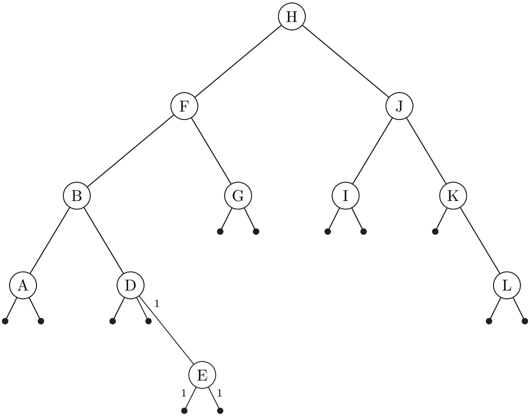

0
1
H
H
F
F
J
B
B
G
I
K
A
D
D
L
E

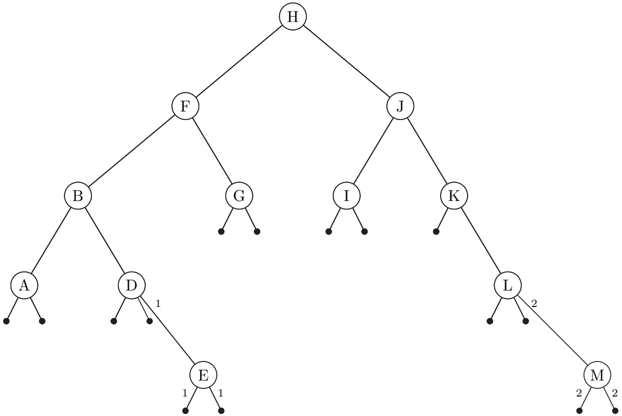

0
1
2
H
H
H
F
F
J
J
B
B
G
I
K
K
A
D
D
L
L
E
M

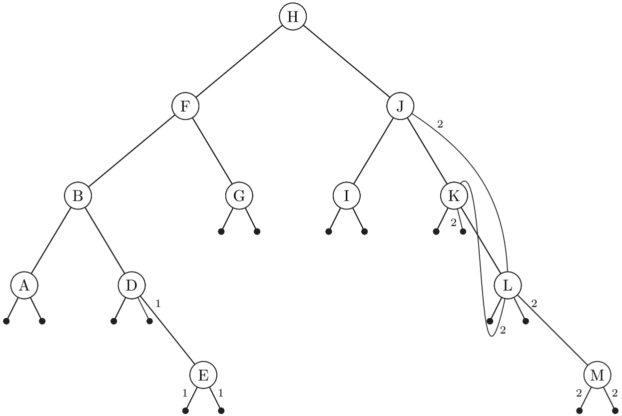

0 1 2 3
H H H H
F F F
J J
B B B
G
I
K K
A
D D D
L L
C
E
M

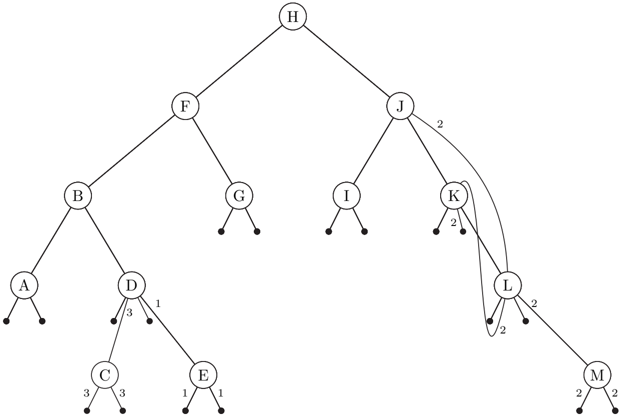
	pamäť	find(x, t)
kopírovanie cesty	$+O(\log n)$	$O(\log n)$


H
F
J
B
G
I
K
A
D
L


H
F
J
B
G
I
K
A
D
1
L
E
1
1

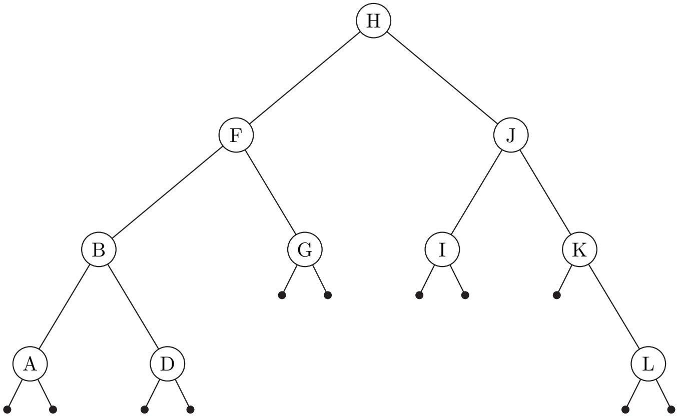

H
F
J
B
G
I
K
A
D
L
E
M
1
1
1
2
2
2

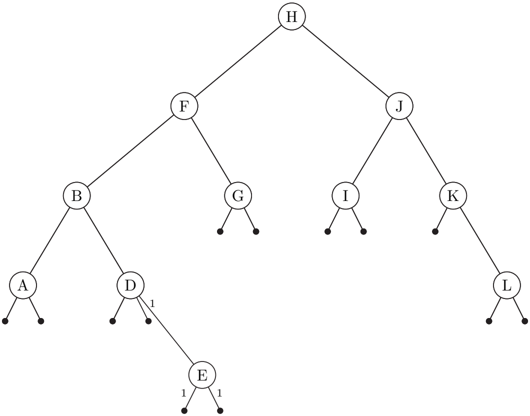

H
F
J
2
B
G
I
K
2
A
D
1
L
2
2
E
1
1
M
2
2

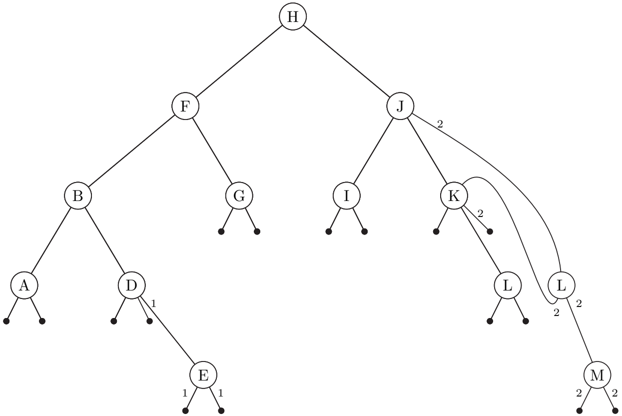

H
F
J
2
B
G
I
K
2
A
D
1
3
L
2
2
C
E
M
3
3
1
1
2
2

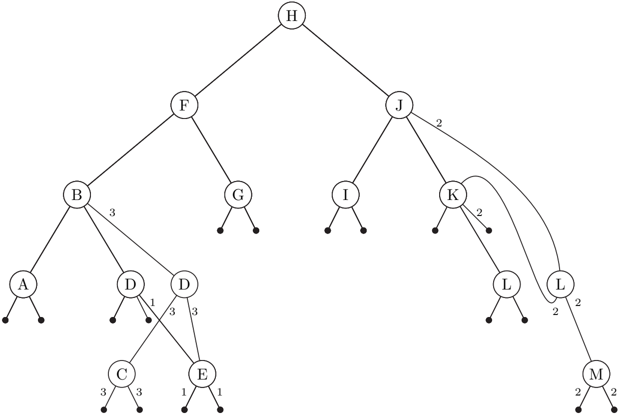
	pamäť	find(x, t)
kopírovanie cesty	$+O(\log n)$	$O(\log n)$
veľké vrcholy	$+O(\#zmien)$	$O(\log t \times \log n)$

	pamäť	find(x, t)
kopírovanie cesty	$+O(\log n)$	$O(\log n)$
veľké vrcholy	$+O(\#zmien)$	$O(\log t \times \log n)$
limitované vrcholy	$+O(\#zmien)$ amort.	$O(\log n)$


H
F
J
B
G
I
K
A
D
L


H
F
J
B
G
I
K
A
D
L
1
E
1
1


H
F
J
2
B
G
I
K
2
A
D
L
L
1
2
2
E
M
1
1
2
2


H
F
J
2
B
G
I
K
3
2
A
D
D
L
L
1
3
3
2
2
C
E
M
3
3
1
1
2
2

Veta

Ak pôvodný algoritmus spraví Z zmien, perzistentný algoritmus potrebuje $O(Z)$ pamäte.

Každý vrchol má 1 extra políčko, kde si vie zapísať zmenu (čo sa zmenilo, ako a kedy)

INVARIANT: Zaplnený vrchol má našetrený 1$.

$$\Phi(D) = \#\text{zaplnených vrcholov}$$

Veta

Ak pôvodný algoritmus spraví Z zmien, perzistentný algoritmus potrebuje $O(Z)$ pamäte.

Každý vrchol má 1 extra políčko, kde si vie zapísať zmenu (čo sa zmenilo, ako a kedy)

INVARIANT: Zaplnený vrchol má našetrený 1$.

$$\Phi(D) = \#\text{zaplnených vrcholov}$$

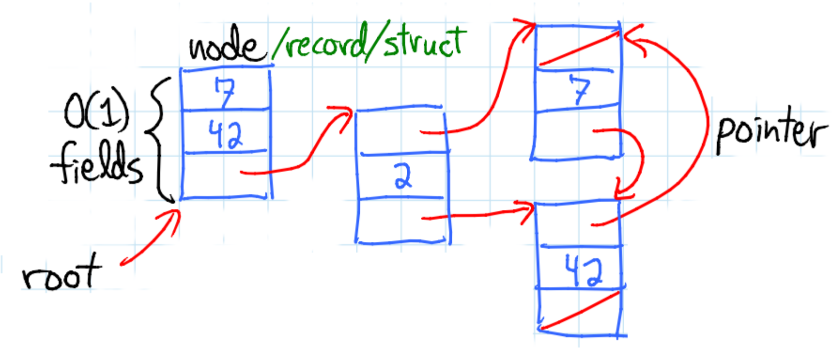
Veta

Ak pôvodný algoritmus spraví Z zmien, perzistentný algoritmus potrebuje $O(Z)$ pamäte.

Každý vrchol má 1 extra políčko, kde si vie zapísať zmenu (čo sa zmenilo, ako a kedy)

INVARIANT: Zaplnený vrchol má našetrený 1$.

$$\Phi(D) = \#\text{zaplnených vrcholov}$$



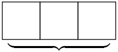
Pointer machine:

- máme vrcholy (record/struct)
- každý môže mať len konštantnú veľkosť (žiadne polia/stringy)
- v kažom políčku je buď hodnota (int/char/bool) alebo pointer

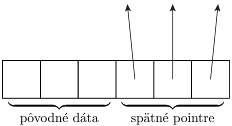
Veta

Ľubovoľnú dátovú štruktúru pre pointer machine, kde na každý vrchol ukazuje najviac $p = O(1)$ pointrov vieme prerobiť na čiastočne perzistentnú, pričom

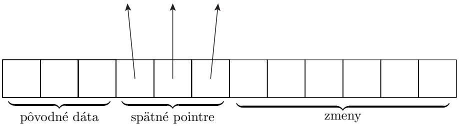
- *čas bude asymptoticky rovnaký a*
- *pamäť je $+O(1)$ za každú zmenu.*



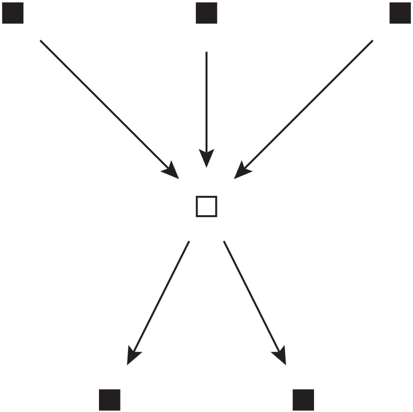
pôvodné dáta

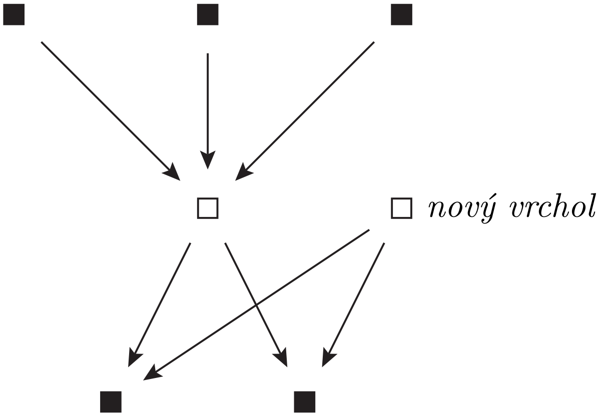


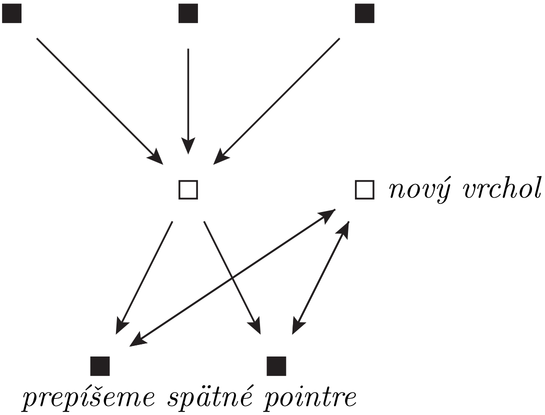
- pridáme max. p spätných pointrov
- stačí ich udržiavať len pre poslednú verziu

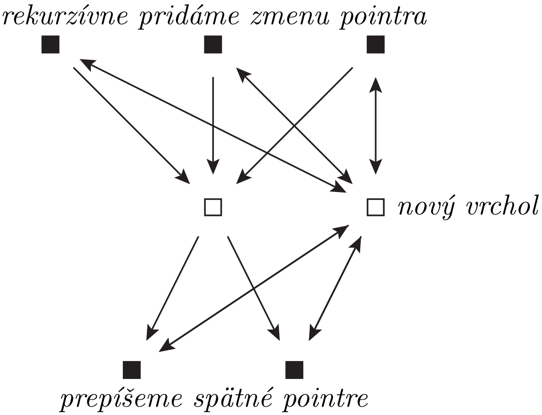


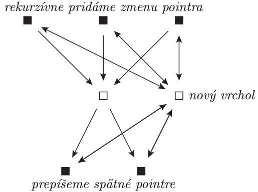
- pridáme $2p$ extra políčok na zmeny
- zmena $=$ (čo meníme, nová hodnota, čas)




nový vrchol

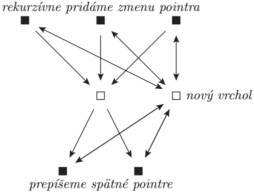

nový vrchol
prepíšeme spätné pointre


rekurzívne pridáme zmenu pointra
nový vrchol
prepíšeme spätné pointre



INVARIANT: Každý vrchol má našetrených toľko \$, koľko má zmenených políčok.

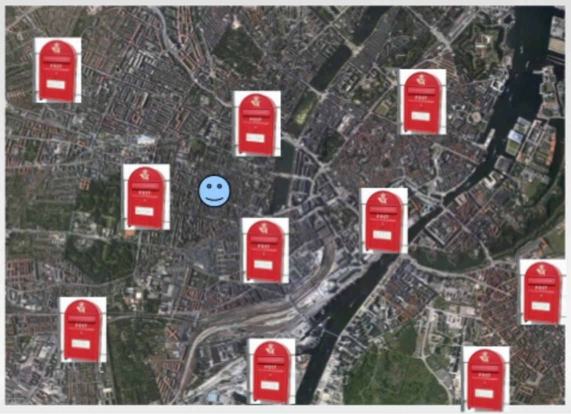
$$\Phi(D) = \#\text{zmien v poslednej verzii} = 2p - \#\text{voľných políčok}$$



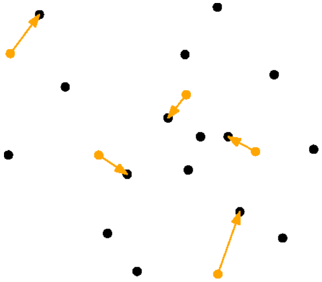
INVARIANT: Každý vrchol má našetrených toľko \$, koľko má zmenených políčok.

$$\Phi(D) = \#\text{zmien v poslednej verzii} = 2p - \#\text{voľných políčok}$$

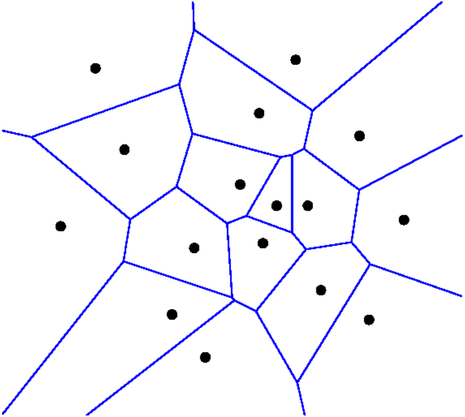
Späť ku geometrii



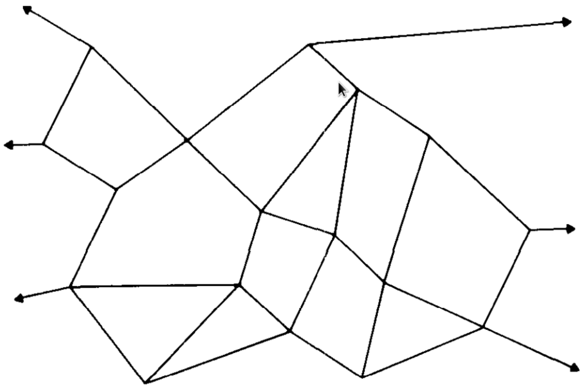
- kde je najbližšia pošta?



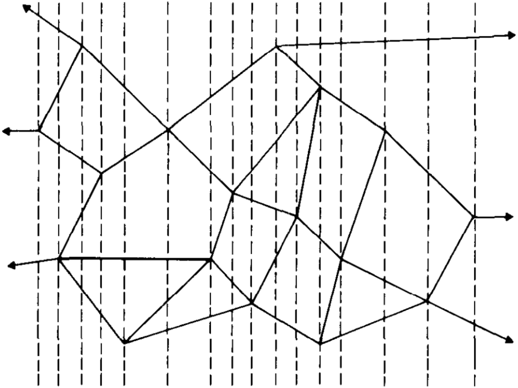
- dané body $p_1, \ldots, p_n$
- môžeme si ich predspracovať
- query: k danému bodu q nájdi najbližší p_i



- Voronoiov diagram: pre každý bod máme oblasť, v ktorej je daný bod najbližšie
- dá sa spočítať v $O(n \log n)$



- dané je rozdelenie plochy na mnohouholníkové oblasti
- môžeme si ich predspracovať
- query: pre daný bod q nájdi oblasť, kam patrí



- rozdelíme na pásy podľa x
- v každom páse zotriedime úsečky podľa y

- pamäť $O(n^2)$
- predspracovanie: $O(n^2 \log n)$
- query: $2\times$ binsearch – $O(\log n)$

- pamäť $O(n^2)$
- predspracovanie: $O(n \log n)$ – zľava doprava `insert`/`delete`
- query: $2\times$ binsearch – $O(\log n)$

- pamäť $O(n)$ – použijeme perzistentný strom
- predspracovanie: $O(n \log n)$ – zľava doprava `insert/delete`
- query: $2\times$ binsearch – $O(\log n)$