LCA a RMQ

V tejto kapitole sa budeme venovať dvom na prvý pohľad úplne odlišným a nesúvisiacim algoritmickým problémom, ktoré sa nám zišli pri riešení úloh na sufixových stromoch.

Prvý z nich je LCA – Lowest Common Ancestor, teda najnižší spoločný predok. Úloha znie nasledovne: Máme dopredu daný koreňový strom, v ktorom sa vrcholy nemenia. Máme tiež čas na jeho predspracovanie, teda môžeme si pripraviť dátové štruktúry a pomocné informácie. Následne budeme opakovane dostávať dotazy, v ktorých dostaneme vždy dvojicu vrcholov a našou úlohou bude čo najrýchlejšie nájsť ich najnižšieho spoločného predka. Ako napovedá už názov, najnižší spoločný predok je taký vrchol v strome, ktorý je predkom oboch vrcholov a zároveň sa nachádza najhlbšie (teda najbližšie k týmto vrcholom).

Druhý problém, ktorému sa budeme venovať, je RMQ – Range Minimum Query, teda dotazy na minimum v rozsahu. V tomto prípade máme dané pole čísel $A[0 \dots n-1]$ a cieľom je opakovane zodpovedať dotazy typu: „Kde sa nachádza najmenšia hodnota v poli medzi indexmi i a j?“ Inými slovami: „Nájdi, kde je minimum v úseku $A[i \dots j]$.“ Rovnako ako pri LCA, aj tu predpokladáme, že pole sa nemení (je statické), a že máme čas na predspracovanie údajov. Po ňom nasledujú dotazy, ktoré chceme zodpovedať čo najrýchlejšie.

Ako by ste takéto úlohy riešili?

Jednoduché riešenia

Postupne sa pozrime na niekoľko základných, čoraz lepších riešení, pričom budeme porovnávať čas zodpovedania dotazu, pamäťovú náročnosť a čas predspracovania.

Začnime problémom RMQ.

RMQ #1. Žiadne predspracovanie: Najjednoduchší prístup je nerobiť nič navyše. Pre každý dotaz RMQ(i, j) jednoducho prejdeme celé pole medzi i a j a nájdeme minimum.

Čas na dotaz	Pamäť	Predspracovanie
$O(n)$	$O(n)$	žiadne

RMQ #2. Predpočítame všetko: Ďalšia možnosť je predpočítať si minimum pre *každú možnú* dvojicu indexov (i, j) a uložiť si ho do tabuľky. Potom môžeme každý dotaz na RMQ zodpovedať v konštantnom čase jednoduchým pohľadom do tabuľky.

Čas na dotaz	Pamäť	Predspracovanie
$O(1)$	$O(n^2)$	$O(n^2)$

RMQ #3. Intervalový strom: Nad poľom si zostrojíme binárny strom (každý vrchol bude zodpovedať nejakému intervalu pod ním). Pre každý vrchol spočítame minimum z jeho dvoch synov (ľavý a pravý syn zodpovedá prvej a druhej polovici intervalu), tým pádom bude mať každý vrchol predpočítané minimum z intervalu pod ním. Keď budeme chcieť nájsť minimum z intervalu (i, j), odpoveď poskladáme z najviac $O(\log n)$ podstromov, ktoré dokopy pokrývajú celý interval.

Čas na dotaz	Pamäť	Predspracovanie
$O(\log n)$	$O(n)$	$O(n)$

Dá sa to lepšie?

Oddýchnime si a pozrime sa na LCA. Prvé dva nápady pre RMQ fungujú aj pre LCA:

LCA #1. Žiadne predspracovanie: Pre každý dotaz $\mathrm{LCA}(u, v)$ jednoducho prejdeme a zapíšeme si cestu od u aj od v ku koreňu. Potom tieto cesty prejdeme v opačnom smere od koreňa k u/v; obe cesty najskôr začínajú spoločne no a my potrebujeme nájsť prvé miesto, kde sa odpoja.

Druhá možnosť je, že si predpočítame hĺbku každého vrcholu. Pri otázke na $\mathrm{LCA}(u, v)$ najskôr dorovnáme hĺbky (toho, kto je hlbšie, posunieme po rodičoch vyššie, kým sa nedostane na rovnakú úroveň) a následne oboch postupne *naraz* posúvame smerom nahor, kým sa nestretnú (prvý spoločný vrchol je práve LCA).

Čas na dotaz	Pamäť	Predspracovanie
$O(h)$, kde h je výška stromu, najviac $O(n)$	$O(n)$	žiadne, resp. $O(n)$

LCA #2. Predpočítame všetko: Pre každú dvojicu vrcholov u a v si spočítame výsledok a uložíme do tabuľky. Každý dotaz potom vieme vyriešiť jedným pohľadom do tabuľky. (Skúste si rozmyslieť, ako túto tabuľku spočítať efektívne.)

Čas na dotaz	Pamäť	Predspracovanie
$O(1)$	$O(n^2)$	$O(n^2)$

OK. Vedeli by sme nájsť riešenie, ktoré má čas lepší ako lineárny a pamäť lepšiu ako kvadratickú?

Hint: Binárne vyhľadávanie.

Hint #2: Predstavme si, že naším prvým krokom bude vyrovnať hĺbky oboch vrcholov. To znamená, že hlbšie položený vrchol „vystúpa“ po strome nahor, kým sa nedostane na rovnakú úroveň ako ten druhý. Ako sa to dá lepšie ako v lineárnom čase a zároveň lepšie ako s kvadratickou pamäťou?

LCA #3 Binárny rebrík. Myšlienka je nasledovná: Predpokladajme, že vrcholy už máme dorovnané na rovnakej úrovni a predstavme si, že si pod seba napíšeme obe cesty od vrcholov smerom ku koreňu – napíšeme si zoznam predkov, krok za krokom, až po koreň. Ako nájdeme miesto, kde sa tieto dve cesty prvýkrát „zbehnú“?

Jednoducho, pomocou binárneho vyhľadávania: Pozrieme sa do stredu týchto ciest.

- Ak sú v tomto bode predkovia rovnakí, vieme, že LCA je v tejto výške alebo ešte nižšie (bližšie k vrcholom) – LCA treba hľadať v prvej polovici.
- Ak sú predkovia rozdielni, znamená to, že k zhodnému predkovi sme sa ešte nedostali, a LCA musí byť vyššie v strome – LCA trbea hľadať v druhej polovici.

Takýmto spôsobom dokážeme v logaritmickom počte krokov nájsť najnižší bod, kde sa obe cesty spájajú.

Ostáva ešte vyriešiť dve otázky: Ako dorovnať výšky vrcholov, aby sme mohli hľadať LCA v synchronizovaných úrovniach? A ako sa pri binárnom vyhľadávaní pozrieť do stredu? Postupovať zakaždým krok po kroku by bolo pomalé. Potrebujeme mať možnosť „skákať“ rýchlejšie – nie po jednom, ale po väčších skokoch.

Riešením je predpočítať si pre každý vrchol skoky o 1 predka, o 2 predkov, o 4, o 8, o 16, atď., skoky dĺžky 2^k pre každú mocninu dvojky. Pre každý vrchol v a každé k od 1 po $\lg n$ si uložíme

$$\mathrm{up}[v][k] = \text{predok vo vzdialenosti } 2^k \text{ od } v.$$

(Táto technika sa nazýva *binary lifting*.)

Dorovnanie hĺbok potom dokážeme v logaritmickom čase: Napríklad ak jeden vrchol je v hĺbke 47 a druhý v hĺbke 68, potrebujeme druhým skočiť o 21 vyššie, čo zvládneme jednoducho troma skokmi o $16 + 4 + 1$.

Namiesto klasického binárneho vyhľadávania, ktoré sa pozerá vždy do stredu použijeme variant, ktorý sa zakaždým pozrie na najbližšiu menšiu mocninu dvojky. Takto sa celý dotaz LCA dá vyriešiť v čase $O(\log n)$, pričom predspracovanie trvá $O(n \log n)$ a zaberá rovnaké množstvo pamäte.

Čas na dotaz	Pamäť	Predspracovanie
$O(\log n)$	$O(n \log n)$	$O(n \log n)$

Hmmm... síce sme dorovnali tretie riešenie RMQ (logaritmický čas), ale pamäť a predspracovanie je horšie! Dá sa to lepšie?

RMQ – ešte lepšie riešenie

Späť ku RMQ. V predchádzajúcej časti sme si ukázali viacero riešení – od naivného lineárneho po logaritmické s intervalovým stromom. Zaujímavé však je, že všetky doterajšie riešenia sú oveľa všeobecnejšie, než by sa na prvý pohľad

mohlo zdať. *Nevyužili sme pritom žiadnu špeciálnu vlastnosť operácie minimum!* Každý z uvedených prístupov by úplne rovnako fungoval aj pre iné operácie, napríklad:

- výpočet súčtu alebo súčinu prvkov v intervale,
- bitové operácie ako AND, OR, XOR,
- dokonca aj pre zložitejšie štruktúry ako:
  - matice, kde v intervale počítame súčin matíc (napr. na ľavo asociovaný),
  - alebo funkcie $X \to X$ reprezentované tabuľkou, pričom chceme poskladať funkcie v intervale ($(f \circ g)(x) = g(f(x))$).

Naše doterajšie riešenia fungujú pre ľubovoľnú asociatívnu operáciu, pre fanúšikov algebry: pre ľubovoľný monoid. Nevyužívajú žiadnu špeciálnu vlastnosť minima.

Môžeme si teda položiť otázku: *Čo ak nás zaujíma konkrétne práve minimum?* Existujú efektívnejšie riešenia ktoré naplno využívajú vlastnosti operácie min? A čím je vlastne minimum špeciálne?

Nuž, operácia min má hneď niekoľko príjemných vlastností. Je asociatívna (nezáleží na zátvorkách), komutatívna (nezáleží na poradí) a okrem toho má ešte jednu tajnú zbraň:

$$\min(x, x) = x \qquad \text{pre každé } x.$$

V preklade: ak aj nejaký prvok započítame viackrát, výsledné minimum sa nezmení! Ak by ste chceli machrovať v krčme (alebo na skúške), tak táto vlastnosť sa volá *idempotencia*.

V treťom riešení, kde sme použili intervalový strom, sme celý interval rozložili na niekoľko disjunktných podintervalov (vždy s dĺžkami, ktoré sú mocninami dvojky). Pre tieto podintervaly sme mali už vypočítanú odpoveď a stačilo jednotlivé medzivýsledky skombinovať. V najhoršom prípade však bolo potrebných až $O(\log n)$ podintervalov – práve preto, že boli disjunktné a museli pokrývať celý rozsah presne.

Pri operácii minimum máme výhodu: môžeme použiť aj intervaly, ktoré sa prekrývajú (výsledok to nezmení).

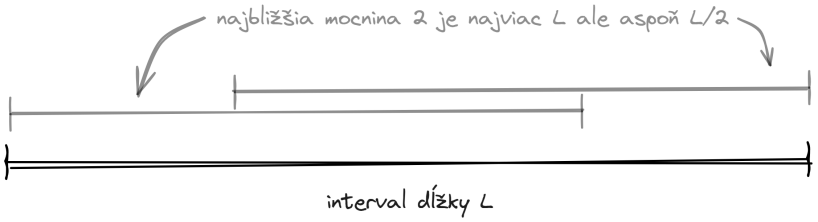
RMQ #4 Riedka tabuľka. Myšlienka riešenia je prekvapivo jednoduchá: predpočítame si minimá pre všetky intervaly, ktorých dĺžka je mocninou dvojky. Konkrétne pre každú začiatočnú pozíciu $i \in [0 \ldots n)$ a pre každé k od 0 po $\lg n$ si spočítame

$$M[k][i] = \min A[i \ldots i + 2^k - 1].$$

Finta spočíva v tom, že *každý* interval $[i, j]$ (dĺžky $\ell = j - i + 1$) vieme pokryť *len dvoma* intervalmi dĺžky 2^k. Ako to funguje? Najskôr si nájdeme najväčšiu mocninu dvojky, ktorá sa ešte do nášho intervalu zmestí:

$$2^k \leq \ell \qquad \text{teda} \qquad k = \lfloor \lg \ell \rfloor.$$

Hodnota 2^k je určite aspoň polovica dĺžky intervalu, $\ell/2 \leq k \leq \ell$, takže celý interval pokryjeme jedným intervalom, ktorý začína na i ($A[i \ldots i + 2^k - 1]$) a jedným, ktorý končí na j ($A[j - 2^k + 1 \ldots j]$).



Výsledok spočítame ako

$$\mathrm{RMQ}(i, j) = \min(M[k][i], M[k][j - 2^k + 1]).$$

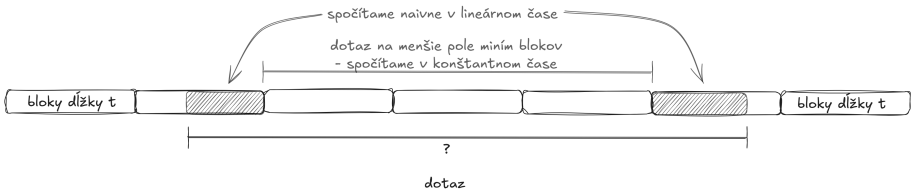
Poznámka k implementácii: hodnotu k prosím vás nepočítajte pomocou matematickej funkcie logaritmus. Počítače prirodzene pracujú v dvojkovej sústave a vďaka tomu môžeme získať k oveľa efektívnejšie ako pozíciu najvyššieho jednotkového bitu. Moderné procesory majú na tento účel špeciálnu inštrukciu s názvom CLZ (Count Leading Zeros), ktorá vráti počet núl na začiatku binárneho zápisu čísla. Hodnotu $k = \lfloor \lg \ell \rfloor$ dostaneme ako $k =$ dĺžka registra $- 1 - \mathrm{CLZ}(\ell)$.

Čas na dotaz	Pamäť	Predspracovanie
$O(1)$	$O(n \log n)$	$O(n \log n)$

Super – podarilo sa nám vyriešiť problém RMQ v konštantnom čase a zaplatili sme za to len „trochu horšou“ pamäťovou zložitosťou $O(n \log n)$. Dobrá správa: Vyhrali sme, zjavne lepší ako konštantný čas už dosiahnuť nevieme. Zlá správa: Lenže teoreticky by sa stále mohla dať zlepšiť pamäť. A tak ostáva v pozadí trýznivá otázka: Dá sa to lepšie?

RMQ #5. Delenie na bloky. Tu je jeden praktický nápad, ak nám záleží na úspornejšom využití pamäte a zároveň nepotrebujeme úplne najrýchlejšie dotazy. Môžeme urobiť rozumný kompromis a získať lepšiu pamäť za cenu trochu horšieho času: Pole veľkosti n si rozdelíme na bloky veľkosti t (povedzme $t = 10$). Dostaneme tak n/t blokov. Z každého bloku spočítame minimum a zostrojíme nové, t-krát menšie pole týchto miním. Na toto menšie pole aplikujeme predchádzajúce riešenie – budeme tak vedieť v konštantnom čase odpovedať na RMQ dotazy medzi *celými blokmi*.

A čo s intervalmi, ktoré zasahujú čiastnočne do vnútra blokov? V konštantnom čase nájdeme minimum menšieho intervalu zarovnaného na bloky, a zvyšné kúsky na začiatku a na konci intervalu (najviac $2t$ prvkov) jednoducho prejdeme naivne v lineárnom čase.



Celkový čas je $O(t)$, na druhej strane pamäťová náročnosť je t-krát menšia. Jeden extrém je voľba $t = 1$, čo je v podstate to isté ako riešenie s riedkou tabuľkou s časom $O(1)$ a pamäťou $O(n \log n)$). Opačný extrém sú bloky dĺžky $\log n$ – čas narastie na $O(\log n)$, ale pamäť klesne na lineárnu. Dostávame tak alternatívne, úplne nové riešenie, ktoré má rovnakú zložitosť ako intervalové stromy. Avšak môžeme si zvoliť aj ľubovoľné t „medzi tým“, podľa toho, či chceme lepší čas alebo lepšiu pamäť. Dostávame tak celé spektrum riešení a tzv. trade-off (niečo za niečo) medzi časom a pamäťou. Napríklad pre $t = \sqrt{\log n}$ bude čas $O(\sqrt{\log n})$ a pamäť $O(n\sqrt{\log n})$.

Rekapitulácia. Skúsme si zrekapitulovať, kde sme a čo už vieme. Riešenia, ktoré sme zatiaľ vymysleli sú v tabuľke nižšie:

Riešenia RMQ	**Čas na dotaz**	**Pamäť**	**Predspracovanie**
#1 bez predspracovania	$O(n)$	$O(n)$	žiadne
#2 všetko predpočítané	$O(1)$	$O(n^2)$	$O(n^2)$
#3 intervalový strom	$O(\log n)$	$O(n)$	$O(n)$
#4 riedka tabuľka	$O(1)$	$O(n \log n)$	$O(n \log n)$
#5 delenie na bloky	$O(t)$	$O(n \log n/t)$	$O(n \log n/t)$
napr. pre $t = \sqrt{\log n}$	$O(\sqrt{\log n})$	$O(n\sqrt{\log n})$	$O(n\sqrt{\log n})$

Riešenia LCA	**Čas na dotaz**	**Pamäť**	**Predspracovanie**
#1 bez predspracovania	$O(n)$	$O(n)$	žiadne
#2 všetko predpočítané	$O(1)$	$O(n^2)$	$O(n^2)$
#3 binárny rebrík	$O(\log n)$	$O(n)$	$O(n)$

Vzťah LCA a RMQ

Z doterajšieho vývoja by sa mohlo zdať, že problém RMQ je jednoduchší ako LCA. Veď pri RMQ pracujeme len s obyčajným poľom čísel, zatiaľ čo pri LCA riešime dotazy na predkov v stromovej štruktúre, čo už na prvý pohľad vyzerá zložitejšie. A naozaj, pre RMQ sme našli oveľa efektívnejšie algoritmy: $O(1)$ čas, $O(n \log n)$ pamäť alebo $O(\log n)$ čas, $O(n)$ pamäť, zatiaľčo naše doteraz najlepšie riešenie LCA s binárnym rebríkom má $O(\log n)$ čas a $O(n \log n)$ pamäť a celé pôsobí o čosi „komplikovanejšie“ ako riešenia RMQ.

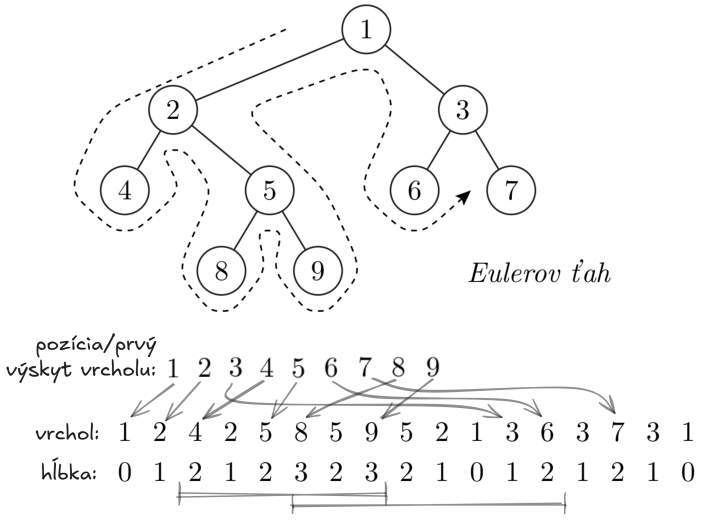
Navyše, netrpezlivým čitateľom už možno vŕta v hlave otázka: *„Prečo sa vôbec obom problémom venujeme v jednej kapitole? Veď spolu vôbec nesúvisia. . . “*

No tu prichádza prekvapenie: LCA a RMQ sú oveľa užšie späté, než sa na prvý pohľad zdá. Ukážeme si, že oba problémy sú algoritmicky ekvivalentné – teda rovnako „ťažké“ (alebo ak chcete: rovnako „ľahké“) a jeden vieme efektívne vyriešiť pomocou druhého.

Redukcia LCA na RMQ

Predstavme si, že máme daný strom z úlohy LCA a našou úlohou je opakovane odpovedať na dotazy typu: „Nájdi najnižšieho spoločného predka vrcholov u a v.“ Ukážeme si, že takúto úlohu vieme pretransformovať na problém RMQ. Cieľom tejto časti bude ukázať, ako zo stromu „vyrobiť“ pole, a ako dotazy typu LCA v tomto strome preložiť na dotazy RMQ, vďaka ktorým už potom nájdeme LCA jednoducho a efektívne. Predstavme si, že niekto za nás už problém RMQ vyriešil. My riešime LCA, ale v našom riešení môžeme pokojne použiť „knižnicu na RMQ“ ako čiernu krabičku.

Ako na to? Strom „rozvinieme do poľa“ pomocou tzv. Eulerovského prechodu okolo stromu (pozri príklad na obrázku) – ide o modifikované prehľadávanie do hĺbky, pričom pri každej návšteve vrcholu si zapíšeme jeho *hĺbku*. Okrem toho si spravíme dve tabuľky, ktoré nám umožnia prevádzať medzi vrcholmi a pozíciami v tomto poli: pre každú pozíciu v poli hĺbok si poznačíme *vrchol*, ktorému zodpovedá a naopak, pre každý vrchol si uložíme *pozíciu* v poli hĺbok, keď sme vrchol *prvýkrát navštívili*. Všimnite si, že vrcholy môžeme navštíviť viackrát, napríklad z koreňa 1 vôjdeme do 2, zídeme do 4, vrátime sa späť do 2, a cez 2 prejdeme ešte raz po tom, čo prejdeme cez podstrom 5 8 9.



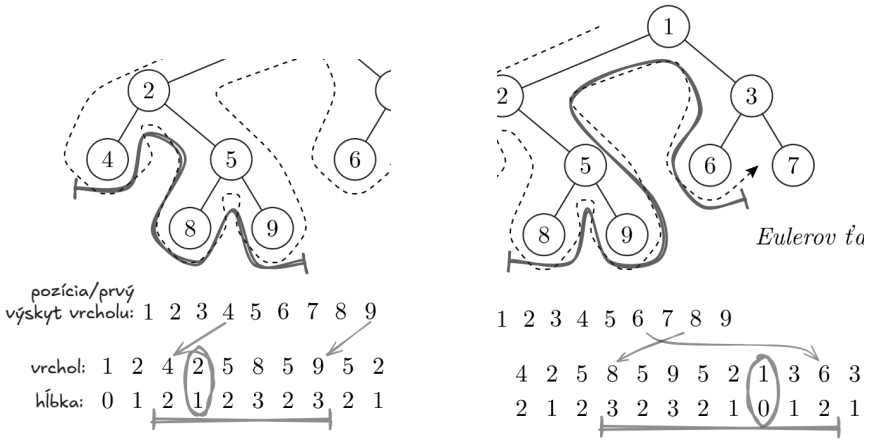
Pole hĺbok si predspracujeme na hľadanie RMQ. Ako teraz nájdeme LCA

dvoch vrcholov? Pre vrcholy u, v platí:

$$\mathrm{LCA}(u,v) = \mathrm{vrchol}[\underset{\text{hĺbok}}{\mathrm{RMQ}}(\text{pozícia } u, \text{pozícia } v)]$$

alebo slovne: najskôr premeníme vrcholy na pozície v poli hĺbok. Potom spočítame RMQ na tomto intervale (pozor: potrebujeme pozíciu minima, nie len jeho hodnotu) a nakoniec túto pozíciu prevedieme naspäť na zodpovedajúci vrchol, čo je najnižší spoločný predok.

Ukážme si to na dvoch príkladoch: $\mathrm{LCA}(4,9)$ a $\mathrm{LCA}(8,6)$. V prvom príklade (pozri obr. vľavo) sa vrcholy 4 a 9 nachádzajú prvýkrát na pozíciách 2 a 7. Všimnite si, že tento interval zodpovedá presne časti Eulerovského ťahu medzi 4 a 9 a obsahuje hĺbky vrcholov, ktoré sme medzi tým navštívili. Z nich najmenšiu hĺbku 1 (index 3) má vrchol číslo 2, čo je naozaj $\mathrm{LCA}(4,9)$. V druhom príklade (pozri obr. vpravo) sa vrcholy 8 a 6 nachádzajú na pozíciách 5 a 12 a tento interval zodpovedá časti Eulerovského ťahu medzi 8 a 6. Najvyšší vrchol, ktorý po ceste navštívime, je koreň 1.



Skúste si rozmyslieť, že celú redukciu (v rámci predspracovania) zvládneme v lineárnom čase. (Vstup sa nám pritom trochu nafúkne, ale iba trochu – aká je dĺžka Eulerovského ťahu?)

Skvelé! Vďaka tejto redukcií už vieme aj LCA riešiť v konštantnom čase a $O(n \log n)$ pamäti! Stačí LCA tranformovať na RMQ a na RMQ použiť riešenie s riedkou tabuľkou a predpočítanými intervalmi dĺžky mocnín dvoch.

Redukcia RMQ na LCA

Teraz sa môžeme pozrieť na opačný smer redukcie: ako previesť problém RMQ na problém LCA. Máme pole čísel a chceme odpovedať na dotazy typu RMQ (minimum v intervale). Ako to prevedieme na dotaz typu LCA v strome?

Použijeme tzv. kartézsky strom – je to binárny strom, ktorý:

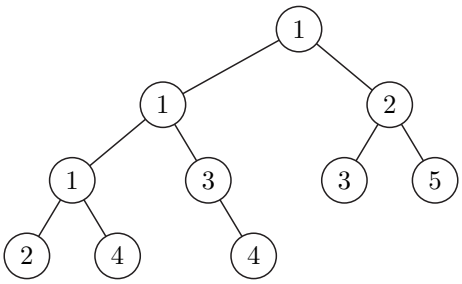
- rešpektuje inorder poradie – ak prvky vypíšeme v inorder poradí (ľavý podstrom → koreň → pravý podstrom), dostaneme pôvodné pole
- a zároveň spĺňa vlastnosť min-haldy: každý vrchol má menšiu hodnotu ako jeho deti.

Ak sú všetky hodnoty v poli rôzne, potom je kartézsky strom nad týmto poľom jednoznačne určený:

- minimum poľa musí byť koreň,
- ľavý podstrom sa skonštruuje rekurzívne z prvkov, ktoré ležia naľavo od minima a
- pravý podstrom sa skonštruuje z prvkov, napravo od minima.

V prípade, že sa v poli nachádzajú opakujúce sa hodnoty, strom už nie je jednoznačne určený, ale stále platí, že *niektorý* prvok s minimálnou hodnotou musí byť v koreni a zvyšok sa skonštruuje rekurzívne (pozri príklad na obrázku).

2 1 4 1 3 4 1 3 2 5



K tomuto stromu opäť pripravíme dve pomocné tabuľky, ktoré nám umožnia prevádzať medzi pozíciami v poli a vrcholmi v strome.

TODO: prečo to funguje

TODO: lineárna konštrukcia

Optimálne riešenie: konštantný čas a lineárna pamäť

Doteraz sme si ukázali mnoho prístupov k problému RMQ, pričom najrýchlejší v konštantnom čase potreboval pamäť $O(n \log n)$. Od toho momentu nás trápi otázka: *Dá sa dosiahnuť ešte lepší algoritmus – konštantný čas a iba lineárna pamäť?*

Odpoveď je prekvapivo áno. A cesta k nemu je ešte prekvapivejšia a zdanlivo úplne nelogická:

RMQ najskôr zredukujeme na LCA a potom naspäť na RMQ.

(Prosím?!) Áno, čítate správne: pole, na ktorom chceme riešiť RMQ, najskôr transformujeme na kartézsky strom a úlohu premeníme na LCA. Následne spravíme Eulerovský prechod tohto stromu, vďaka čomu vznikne úplne nové pole (pole hĺbok), pričom RMQ pôvodného poľa vieme riešiť pomocou RMQ zodpovedajúcich intervalov v novom poli.

Znie to ako šialenstvo, pretože sme práve úlohu RMQ pretransformovali na *inú* úlohu RMQ – a ešte k tomu na väčšom poli. Namiesto toho, aby sme si pomohli, sme si zdalo by sa pridali prácu. Vrátili sme sa tam, odkiaľ sme prišli, len s väčším vstupom.

A predsa je tu jeden podstatný detail: nové pole má veľmi špecifickú štruktúru, keďže vzniklo z Eulerovského prechodu binárneho stromu:

- hodnoty zodpovedajú hĺbkam vrcholov, tzn., že sú to celé čísla od 0 po n,
- pole začína nulou a končí nulou, pretože Eulerovský prechod sa začína a končí v koreni, ktorý má hĺbku 0,
- Každé dva susedné prvky sa líšia presne o ± 1, keďže pri pohybe stromom buď zostupujeme o úroveň nižšie (hĺbka $+1$), alebo sa vraciame späť nahor (hĺbka -1).

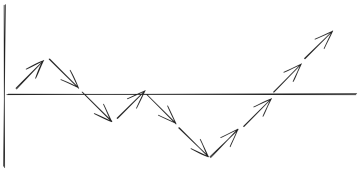
Zatiaľčo pôvodné pole mohlo obsahovať ľubovoľne veľké, aj reálne, aj záporné hodnoty, nové pole už obsahuje iba prvky od 0 po n. Zatiaľčo v pôvodnom poli sa susedné prvky mohli líšiť neobmedzene, v novom poli sa líšia vždy o 1. Táto redukovaná úloha sa preto nazýva tiež RMQ$\pm$1.

Optimálne riešenie RMQ$\pm$1 začína podobne ako riešenie #5 rozdelením celého poľa na bloky dĺžky zhruba $\log n$. Z každého bloku spočítame minimum a tieto minimá uložíme do poľa B, ktoré má dĺžku približne $n/\log n$. Na pole B aplikujeme riešenie s riedkou tabuľkou, ktoré síce používa $O(n \log n)$ pamäte, ale teraz pracuje na $\log n$-krát menšom poli, takže celkové predspracovanie zaberie

$$O\left(\frac{n}{\log n} \times \log n\right) = O(n).$$

a odpovede na dotazy medzi celými blokmi budú v konštantnom čase. Rozdiel oproti predchádzajúcim riešeniam je v tom, ako budeme riešiť prečnievajúce časti, ktoré zasahujú iba čiastočne do vnútra blokov.

Postupnosť hodnôt v jednom bloku si môžeme lepšie predstaviť ako graf hĺbok. Jeden blok môže vyzerať napríklad takto:



Všimnite si, že ak nás zaujímajú iba minimá v intervaloch v rámci jedného bloku, nezáleží na tom, v akej hĺbke blok začína. Nezáleží ani na tom, aké konkrétne

hodnoty blok obsahuje. Záleží len na *tvare* tejto krivky, nie na jej zvislej polohe. Ak celý blok zvýšime alebo znížime o konštantu, pozície miním v ľubovoľnom intervale sa nezmenia. Každý blok môžeme bez straty informácie normalizovať tak, aby začínal nulou.

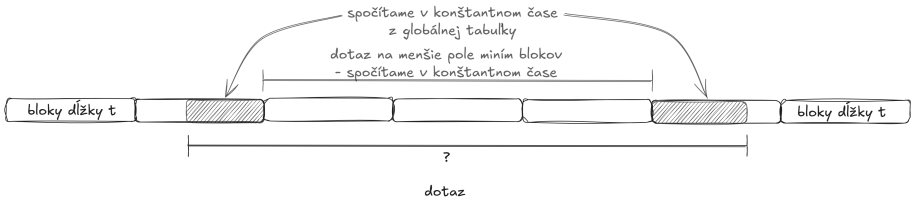
Koľko existuje rôznych blokov? V skutočnosti relatívne málo. Blok dĺžky t vieme opísať ako postupnosť krokov $\uparrow, \downarrow$ dĺžky $t-1$ a teda celkový počet rôznych tvarov blokov je 2^{t-1}. Príklad: majme pole dĺžky jeden milión a rozdelíme ho na bloky dĺžky 11 – takto získame zhruba 90-tisíc blokov. Avšak keď sa pozrieme na počet *rôznych tvarov* blokov po normalizácii, zistíme, že existuje iba 1024 rôznych tvarov! Medzi vyše 90 000 blokmi sa teda neustále opakuje len 1024 rôznych tvarov. Vďaka tomu si môžeme dovoliť pre každý možný tvar bloku vopred predpočítať všetky možné odpovede na všetky RMQ dotazy v rámci tohto bloku. Pre blok dĺžky 11 existuje $11 \times 12/2 = 66$ možných podintervalov, krát 1024 rôznych tvarov – to je len zhruba 68-tisíc hodnôt, ktoré si môžeme predpočítať. Zhruba 68-tisíc hodnôt je úplne zanedbateľné množstvo v porovnaní s pôvodným poľom veľkosti milión. Predpočítané hodnoty si uložíme do jednej globálnej tabuľky vďaka ktorej budeme vedieť spočítať minimum ľubovoľného intervalu v rámci jedného bloku v konštantnom čase.

Finálny algoritmus (Bender a Farach-Colton). Rozdelíme pole na bloky dĺžky $t = \lfloor \frac{1}{2} \log n \rfloor$. Vznikne tak približne $2n/\log n$ blokov. Z každého bloku spočítame minimum a tieto minimá uložíme do poľa B dĺžky $O(n/\log n)$. Na pole B aplikujeme riešenie #4, riedku tabuľku. Výsledkom je štruktúra, ktorá dokáže nájsť minimum medzi celými blokmi v konštantnom čase a lineárnej pamäti.

Jednotlivé bloky znormalizujeme tak, aby začínali od nuly a zakódujeme ich ako binárne reťazce (napríklad `0` znamená klesanie a `1` stúpanie). Počet všetkých možných tvarov blokov je

$$2^{t-1} \approx 2^{\frac{1}{2}\log n} = n^{1/2} = \sqrt{n}.$$

Počet možných podintervalov v jednom bloku je $\binom{t}{2} = O(\log^2 n)$. Pre každý tvar bloku a každý podinterval si spočítame pozíciu minima – dostaneme globálnu tabuľku veľkosti $O(\sqrt{n} \cdot \log^2 n) = o(n)$.



Ako odpovedáme na dotazy RMQ(ℓ, r)? Najskôr interval $[\ell, r]$ v pôvodnom poli transformujeme na zodpovedajúci interval $[i, j]$ v poli RMQ± 1, ktoré

vzniklo Eulerovským prechodom kartézskeho stromu. Ak celý interval $[i, j]$ leží v rámci jedného bloku, predpočítanú odpoveď pre daný tvar bloku a konkrétny podinterval jednoducho nájdeme v globálnej tabuľke. V opačnom prípade si interval rozdelíme na tri časti: kúsok bloku na začiatku (ak i nezačína presne na hranici bloku), kúsok bloku na konci (ak j nekončí presne na hranici bloku) a stredná časť, ktorá sa skladá iba z celých blokov. Na začiatok a koniec použijeme globálnu tabuľku, kde nájdeme príslušný tvar bloku a konkrétny podinterval, na stred použijeme pole B a riešenie s riedkou tabuľkou. Tieto medzivýsledky porovnáme a vyberieme ten najmenší z nich. Tým získame pozíciu minima v poli RMQ$\pm$1 – tú ešte musíme preložiť naspäť na pozíciu v pôvodnom poli.

Každú z týchto častí vyriešime v konštantnom čase – stačí sa pozrieť na správne miesto v príslušnej tabuľke, takže celý dotaz zodpovieme v $O(1)$. Navyše celková pamäť je len lineárna.