Jméno: Petr Valenta **UČO:** 487561

0007 *list* 1 *učo* 4875611 *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

1. [20 bodů] Ulice Algoritmiků je dlouhá a rovná. Na jednom jejím konci stojí Úřad. Na ulici se staví nové a demolují staré bytové domy, které mají různou výšku. Byty v domě jsou prodejné, právě tehdy když mezi domem a Úřadem stojí vyšší dům.
Navrhněte datovou strukturu, ve které lze uchovávat informace o bytových domech na ulici Algoritmiků, zejména jejich výšku a vzdálenost od Úřadu, a nad kterou je možné vykonávat operace:

Construct(b) přidá záznam o nové budově b, která má výšku $b.h$ a jejíž vzdálenost od Úřadu je $b.a$;

Demolish(a) odebere záznam o budově, která stojí ve vzdálenosti a od Úřadu;

Can-see-office(a) vrátí hodnotu False právě tehdy, když mezi budovou b, která stojí ve vzdálenosti a od Úřadu, a Úřadem stojí budova s výškou větší než $b.h$; v opačném případě operace vrátí hodnotu True.

Navrhněte datovou strukturu a implementaci uvedených operací. Požadujeme, aby všechny operace měly složitost v nejhorším případě $\mathcal{O}(\log n)$, kde n je počet domů, které stojí na ulici Algoritmiků.

Příklad. Jestliže na ulici stojí 3 domy ve vzdálenosti 2, 3 a 7 a s výškami 4, 8 a 5 (v daném pořadí), tak operace Can-see-office(7) vrátí hodnotu False. Jestliže nejdříve aplikujeme operaci Demolish(3), tak následná operace Can-see-office(7) vrátí hodnotu True.

Hlavní myšlenka

Použijeme upravenú verziu AVL stromu, kde každý uzol bude reprezentovať jeden dom, pričom úrad stojí na pozícii POS $= 0$. Kľúč podľa ktorého sú uzly v strome usporiadané je pozícia domu $b.a$. V každom uzle si budeme pamätať výšku domu $b.h$ ako h, $lsh = max(h, lsh)$, kde lsh je maximimálna výška ľavého podstramu uzlu a $rsh = max(h, rsh)$, kde rsh je maximálna výška pravého podstromu uzlu. Pre potreby AVL stromu je v každom uzle uložená aj informácia o výške, v ktorej sa uzol nachádza.

Uzol je potom reprezentovaný štruktúrou:
NODE $\{$ $key \leftarrow b.a$, $h \leftarrow b.h$, lsh rsh $\}$

Algoritmus

```
1 function Construct(b)
2 |   newNode ← newNode(key = b.a, h = b.h, lsh = b.h, rsh = b.h)
3 |   root ← insert(root, newNode)
4 |   root ← rebalanceRotation(root)
5 |   adjustHeight(root)                          // standart AVL tree operation
```

Jméno: Petr Valenta **UČO:** 487561

0007 *list* 2 *učo* 4875611 *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

```
1  function insert(root, node)
2      if root = Void then
3          Result ← node
4      else
5          if root.key > node.key then
6              if root.lsh < node.h then
7                  root.lsh ← node.h
8              root.left ← insert(root.left, node)
9          else
10             if root.rsh < node.h then
11                 root.rsh ← node.h
12             root.right ← insert(root.right, node)
13         Result ← rebalanceRotation(root)
14         adjustHeight(Result)
                                          // standart AVL tree operation
15     return Result
```

```
1  function rebalanceRotation(node)
       // same as in standart AVL tree using rotations
2      foreach node which changed his parent do
3          recalculateRotation(node, node.h)
```

```
1  function recalculateRotation(node, h)
2      if node is left child then
3          if parent.lsh < h then
4              parent.lsh ← node.lsh
5          recalculateRotation(node.parent, max(parent.lsh, parent.rsh))
6      else if node is right child then
7          if parent.rsh < h then
8              parent.rsh ← node.rsh
9          recalculateRotation(node.parent, max(parent.lsh, parent.rsh))
10     else
           // node is root
```

Jméno: Petr Valenta **UČO:** 487561

0007 *list* 3 *učo* 487561 *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

```
1  function Demolish(a)
2      if root = Void then
3          Result ← Void
4      else
5          if root.key = a then
6              Result ← root.entry
7              root ← removeNode(node)
8          else
9              Result ← removeRecursive(root, a)
               // if root.key > a then Remove from the left sub-tree
               // else Remove from the right sub-tree
10     adjustHeight(root)
                                                  // standart AVL tree operation
11     root ← rebalanceDeletion(root)
```

```
1  function rebalanceDeletion(node)
       // same as in standart AVL tree using rotations
2      foreach node which changed his parent do
3          recalculateDeletion(node, node.h)
```

```
1  function recalculateDeletion(node, h)
2      if node is left child then
3          if parent.lsh = h then
4              parent.lsh ← max(node.lsh, node.rsh)
5          recalculateDeletion(node.parent, h)
6      else if node is right child then
7          if parent.rsh = h then
8              parent.rsh ← max(node.lsh, node.rsh)
9          recalculateDeletion(node.parent, h)
10     else
           // node is root
```

```
1  function Can-see-office(a)
2      node ← root                                   // start in root
3      pathMax ← −∞
4      Result ← find(node, a, pathMax)
5      return Result
```

Jméno: Petr Valenta **UČO:** 487561

0007 *list* 4 *učo* 4875611 *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

```
1  function find(root, key, pathMax)
2      if root.key = key then
3          if root.h ≥ pathMax and root.h ≥ root.lsh then
4              return True
5          else
6              return False
7      else if root is childless then
8          return NotFound
9      else if root.key > key then
10         find(root.left, key, pathMax)
11     else
           // root.key < a
12         if root.key > pathMax then
13             pathMax ← root.lhs
14         find(root.right, key, pathMax)
```

Zdůvodnění korektnosti

Operácia CAN-SEE-OFFICE je založená na vyhľadávaní v binárnom vyhľadávacom strome, preto ak existuje uzol s kľúčom a, potom ho určite nájde. Budova Úradu stojí na pozícii POS $= 0$ preto pre každú budovu potrebujeme porovnať veľkosť všetkých budov od dotazovaného domu n až po Úrad, teda interval $(0, n)$. Keďže domy sú usporiadané v štruktúre binárneho vyhľadávacieho stromu, je potrebné porovnať uzol n s uzlami v ľavom podstrome rodiča n a ľavých podstromoch rodičov jeho rodiča a s potomkami v ľavom podstrome uzlu n. Vďaka uloženým informáciam o maximálnej výške budovy v ľavom a pravom podstrome pre každý uzol vieme jednoduchým porovnávaním pri prechode zistiť, či velkosť dotazovanej budovy je najvyšia v intervale $(0, n)$.

Operácia CONSTRUCT vloží uzol ako list podľa kľúča. Pri prechode stromom sa provnáva výška vkladaného domu s maximami uloženými v uzloch, cez ktoré nový uzol prepadáva na svoje miesto a aktualizuje ulozené hodnoty podľa potreby, čim sa zabezpečuje integrita uložených dát aj po vložení nového uzlu. Operácia však vykonáva aj rotácie, ktoré menia štruktúru stromu. Rotáciou sa mení len "vertikálne" poradie uzlov v strome. Poradie "horizontálne", teda to, kde sú uzly radené podľa kľúča je zachované. Vďaka tejto vlastnosti máme zachované poradie a je nutné opraviť len uložené maximá v uzloch. Maximá ktoré boli zneplatnené sú v uzloch ktoré prišli o potomka a v uzloch, ktoré získali nového potomka. Preto stačí, že zmena príde od každého uzlu, ktorý zmenil svojho rodiča. Týmto spôsobom sa nové maximá upravia zdola nahor pre celý strom.

Operácia DEMOLISH zmaže uzol zo stromu. Pred samotným zmazaním sa upravia uložené maximá. Po odstránení uzlu sa AVL strom vyvažuje rotáciami ako v prípade CONSTRUCT. Maximá sa upravujú už pred odstránením a to z dôvodu, že nemusia byť potrebné žiadne rotácie. Pri úprave maxím sa výška odstraňovaneho domu propaguje smerom do koreňa stromu a v každom uzle, kde bola táto výška lokálnym maximom sa nahradí novou hodnotou. V prípade, že prišla hodnota od pravého potomka nastaví sa nové maximum pre pravý podstrom na $max(h, child.lsh, child, rsh)$. Ekvivalent pre ľavého potomka je potom úprava maxima ľavého podstromu na $max(h, child.lsh, child, rsh)$.

Časová analýza složitosti

Zložitosť vychádza zo zložitosti AVL stromu. Pri použití neupravenej verzii stromu majú operácie INSERT, DELETE, FIND časovú zložitosť $\mathcal{O}(log(n))$ vďaka seba-vyvažovaniu.

Jméno: Petr Valenta **UČO:** 487561

0007 *list* 5 *učo* 4875611 *body*

Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte. 0123456789

Operácia CAN-SEE-OFFICE je ekvivalentom operácie FIND binárneho vyhľadávacieho stromu obohatená o porovnávania, ktoré sú vykonávané v konštantnom čase. Preto môžme povedať, že CAN-SEE-OFFICE má časovú zložitosť $\mathcal{O}(log(n))$.

Operácia CONSTRUCT je ekvivalentom operácie INSERT AVL stromu. Úprava samotného vkladania uzlu do stromu zvyšuje časovú zložitosť o konštantné operácie. Seba-vyvažovanie zabezpečujú rotácie stromu, ktoré sú volané podľa potreby vo funkcii REBALANCE. Úprava rotácií spočíva v pridaní funkcie RECALCULATEROTATION, ktorá je pre volaná pre každý uzol, ktorý zmenil svojho rodiča. Prepočítanie maximálnych hodnôt pre jeden zmenený uzol má prinajhoršom časovú zložitosť $\mathcal{O}(log(n))$, čo je vlastne prebublanie nových hodnôt do koreňa. To vyplýva z vlastností seba-vyrovnávacieho AVL stromu. Možné kombinácie rotácií sú: LEFT-LEFT, LEFT-RIGHT, RIGHT-RIGHT, RIGHT-LEFT. Počet uzlov, ktoré zmenia svojho rodiča a musia informovať nového o svojich lokálnych maximách je prijanhoršom 5. Z toho vyplýva, že v najhoršom prípade nás bude RECALCULATEROTATION stáť $\mathcal{O}(6log(n))$.

Operácia DEMOLISH je ekvivalentom operácie DELETE AVL stromu. Úpravy operácií DELETE a RECALCULATEDELETION sú ekvivalentné ako pri operácií CONSTRUCT.

Priestorová analýza složitosti

Pre každý uzol si uchovávame informácie o jeho polohe B.A, výške B.H, maximálnej výške v ľavom a pravom podstrome a výšku v strome pre potreby vyvažovania stromu. Dáta su uložené v uzle a preto je priestorová zložoitosť $\mathcal{O}(n \times C)$, kde n je počet uzlov (domov) a C je priestor potrebný pre uchovávané informácie v uzle.