|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | from ..ch08.linked\_binary\_tree import LinkedBinaryTree | |  |  |  | | --- | | from ..ch10.map\_base import MapBase | |  |  |  | | --- | | class TreeMap(LinkedBinaryTree, MapBase): | |  |  |  | | --- | |  |  |  | | --- | | #---------------------------- menimpa kelas Posisi ---------------------------- | |  |  |  | | --- | | class Position(LinkedBinaryTree.Position): | |  |  |  | | --- | | def key(self): | |  |  |  | | --- | | *//* *Kunci kembali dari pasangan nilai kunci peta* | |  |  |  | | --- | | return self.element().\_key | |  |  |  | | --- | | def value(self): | |  |  |  | | --- | | """Return value of map's key-value pair.""" | |  |  |  | | --- | | return self.element().\_value | |  |  |  | | --- | | #------------------------------- utilitas nonpublik ------------------------------- | |  |  |  | | --- | | def \_subtree\_search(self, p, k): | |  |  |  | | --- | | """ Kembalikan Posisi subpohon p yang memiliki kunci k, atau simpul terakhir yang dicari.""" | |  |  |  | | --- | | if k == p.key(): # menemukan kecocokan | |  |  |  | | --- | | return p | |  |  |  | | --- | | elif k < p.key(): # cari subpohon kiri | |  |  |  | | --- | | if self.left(p) is not None: | |  |  |  | | --- | | return self.\_subtree\_search(self.left(p), k) | |  |  |  | | --- | | else: # search right subtree | |  |  |  | | --- | | if self.right(p) is not None: | |  |  |  | | --- | | return self.\_subtree\_search(self.right(p), k) | |  |  |  | | --- | | return p # pencarian gagal | |  |  |  | | --- | | def \_subtree\_first\_position(self, p): | |  |  |  | | --- | | """Kembalikan Posisi item pertama dalam subpohon yang berakar pada p.""" | |  |  |  | | --- | | walk = p | |  |  |  | | --- | | while self.left(walk) is not None: # terus berjalan ke kiri | |  |  |  | | --- | | walk = self.left(walk) | |  |  |  | | --- | | return walk | |  |  |  | | --- | | def \_subtree\_last\_position(self, p): | |  |  |  | | --- | | """ Kembali Posisi item terakhir di subpohon berakar pada p.""" | |  |  |  | | --- | | walk = p | |  |  |  | | --- | | while self.right(walk) is not None: # terus berjalan ke kanan | |  |  |  | | --- | | walk = self.right(walk) | |  |  |  | | --- | | return walk | |  |  |  | | --- | | #--------------------- metode publik yang menyediakan dukungan "posisional" --------------------- | |  |  |  | | --- | | def first(self): | |  |  |  | | --- | | """ Kembalikan Posisi pertama di pohon (atau Tidak ada jika kosong).""" | |  |  |  | | --- | | return self.\_subtree\_first\_position(self.root()) if len(self) > 0 else None | |  |  |  | | --- | | def last(self): | |  |  |  | | --- | | """ Kembalikan Posisi terakhir di pohon (atau Tidak Ada jika kosong).""" | |  |  |  | | --- | | return self.\_subtree\_last\_position(self.root()) if len(self) > 0 else None | |  |  |  | | --- | | def before(self, p): | |  |  |  | | --- | | """Kembalikan Posisi tepat sebelum p dalam urutan alami.  Return None jika p adalah posisi pertama. |  |  | | --- | | """ | |  |  |  | | --- | | self.\_validate(p) # diwarisi dari LinkedBinaryTree | |  |  |  | | --- | | if self.left(p): | |  |  |  | | --- | | return self.\_subtree\_last\_position(self.left(p)) | |  |  |  | | --- | | else: | |  |  |  | | --- | | # walk upward | |  |  |  | | --- | | walk = p | |  |  |  | | --- | | above = self.parent(walk) | |  |  |  | | --- | | while above is not None and walk == self.left(above): | |  |  |  | | --- | | walk = above | |  |  |  | | --- | | above = self.parent(walk) | |  |  |  | | --- | | return above | |  |  |  | | --- | | def after(self, p): | |  |  |  | | --- | | """Kembalikan Posisi tepat setelah p dalam urutan alami.  Return None jika p adalah posisi terakhir. |  |  | | --- | | """ | |  |  |  | | --- | | self.\_validate(p) # diwarisi dari LinkedBinaryTree | |  |  |  | | --- | | if self.right(p): | |  |  |  | | --- | | return self.\_subtree\_first\_position(self.right(p)) | |  |  |  | | --- | | else: | |  |  |  | | --- | | walk = p | |  |  |  | | --- | | above = self.parent(walk) | |  |  |  | | --- | | while above is not None and walk == self.right(above): | |  |  |  | | --- | | walk = above | |  |  |  | | --- | | above = self.parent(walk) | |  |  |  | | --- | | return above | |  |  |  | | --- | | def find\_position(self, k): | |  |  |  | | --- | | """ Kembalikan posisi dengan kunci k, atau tetangga lain (atau Tidak ada jika kosong).""" | |  |  |  | | --- | | if self.is\_empty(): | |  |  |  | | --- | | return None | |  |  |  | | --- | | else: | |  |  |  | | --- | | p = self.\_subtree\_search(self.root(), k) | |  |  |  | | --- | | self.\_rebalance\_access(p) # kait untuk subkelas pohon seimbang | |  |  |  | | --- | | return p | |  |  |  | | --- | | def delete(self, p): | |  |  |  | | --- | | """ Hapus item pada Posisi yang diberikan.""" | |  |  |  | | --- | | self.\_validate(p) # diwarisi dari LinkedBinaryTree | |  |  |  | | --- | | if self.left(p) and self.right(p): # p punya dua anak | |  |  |  | | --- | | replacement = self.\_subtree\_last\_position(self.left(p)) | |  |  |  | | --- | | self.\_replace(p, replacement.element()) # dari LinkedBinaryTree | |  |  |  | | --- | | p = replacement | |  |  |  | | --- | | # sekarang p memiliki paling banyak satu anak | |  |  |  | | --- | | parent = self.parent(p) | |  |  |  | | --- | | self.\_delete(p) # diwarisi dari LinkedBinaryTree | |  |  |  | | --- | | self.\_rebalance\_delete(parent) # jika root dihapus, induknya adalah None | |  |  |  | | --- | | #--------------------- metode publik untuk antarmuka peta (standar) --------------------- | |  |  |  | | --- | | def \_\_getitem\_\_(self, k): | |  |  |  | | --- | | """ Mengembalikan nilai yang terkait dengan kunci k (menaikkan KeyError jika tidak ditemukan).""" | |  |  |  | | --- | | if self.is\_empty(): | |  |  |  | | --- | | raise KeyError('Key Error: ' + repr(k)) | |  |  |  | | --- | | else: | |  |  |  | | --- | | p = self.\_subtree\_search(self.root(), k) | |  |  |  | | --- | | self.\_rebalance\_access(p) # kait untuk subkelas pohon seimbang | |  |  |  | | --- | | if k != p.key(): | |  |  |  | | --- | | raise KeyError('Key Error: ' + repr(k)) | |  |  |  | | --- | | return p.value() | |  |  |  | | --- | | def \_\_setitem\_\_(self, k, v): | |  |  |  | | --- | | """ Tetapkan nilai v ke kunci k, timpa nilai yang ada jika ada.""" | |  |  |  | | --- | | if self.is\_empty(): | |  |  |  | | --- | | leaf = self.\_add\_root(self.\_Item(k,v)) # dari LinkedBinaryTree | |  |  |  | | --- | | else: | |  |  |  | | --- | | p = self.\_subtree\_search(self.root(), k) | |  |  |  | | --- | | if p.key() == k: | |  |  |  | | --- | | p.element().\_value = v # ganti nilai item yang ada | |  |  |  | | --- | | self.\_rebalance\_access(p) # kait untuk subkelas pohon seimbang | |  |  |  | | --- | | return | |  |  |  | | --- | | else: | |  |  |  | | --- | | item = self.\_Item(k,v) | |  |  |  | | --- | | if p.key() < k: | |  |  |  | | --- | | leaf = self.\_add\_right(p, item) # diwarisi dari LinkedBinaryTree | |  |  |  | | --- | | else: | |  |  |  | | --- | | leaf = self.\_add\_left(p, item) # | |  |  |  | | --- | | self.\_rebalance\_insert(leaf) # kait untuk subkelas pohon seimbang | |  |  |  | | --- | | def \_\_delitem\_\_(self, k): | |  |  |  | | --- | | """ Hapus item yang terkait dengan kunci k (angkat KeyError jika tidak ditemukan.""" | |  |  |  | | --- | | if not self.is\_empty(): | |  |  |  | | --- | | p = self.\_subtree\_search(self.root(), k) | |  |  |  | | --- | | if k == p.key(): | |  |  |  | | --- | | self.delete(p) # rely on positional version | |  |  |  | | --- | | return # successful deletion complete | |  |  |  | | --- | | self.\_rebalance\_access(p) # hook for balanced tree subclasses | |  |  |  | | --- | | raise KeyError('Key Error: ' + repr(k)) | |  |  |  | | --- | | def \_\_iter\_\_(self): | |  |  |  | | --- | | """ Hasilkan iterasi semua kunci di peta secara berurutan.""" | |  |  |  | | --- | | p = self.first() | |  |  |  | | --- | | while p is not None: | |  |  |  | | --- | | yield p.key() | |  |  |  | | --- | | p = self.after(p) | |  |  |  | | --- | | #--------------------- public methods for sorted map interface --------------------- | |  |  |  | | --- | | def \_\_reversed\_\_(self): | |  |  |  | | --- | | """ Hasilkan iterasi semua kunci di peta dalam urutan terbalik.""" | |  |  |  | | --- | | p = self.last() | |  |  |  | | --- | | while p is not None: | |  |  |  | | --- | | yield p.key() | |  |  |  | | --- | | p = self.before(p) | |  | |