ĐẠI HỌC QUỐC GIA HÀ NỘI TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

**KHOA TOÁN - CƠ - TIN HỌC**



Báo cáo Cấu trúc dữ liệu và thuật toán

**Bài toán: Trạm cho thuê xe đạp**

Giảng viên học phần: **PGS.TS. Nguyễn Thị Hồng Minh**

Sinh viên nhóm 8: **1. Nguyễn Thị Minh Hằng**

# 2. Vũ Quang Huy

**Hà Nội - 2023**

ĐẠI HỌC QUỐC GIA HÀ NỘI TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

**KHOA TOÁN - CƠ - TIN HỌC**

Báo cáo Cấu trúc dữ liệu và thuật toán

**Bài toán: Trạm cho thuê xe đạp**

Giảng viên học phần: **PGS.TS. Nguyễn Thị Hồng Minh**

Sinh viên nhóm 8: **1. Nguyễn Thị Minh Hằng**

# 2. Vũ Quang Huy

**Hà Nội - 2023**

**Lời cảm ơn**

Để hoàn thành tốt đề tài này, ngoài sự nỗ lực của các thành viên trong nhóm, nhóm chúng em còn nhận được sự quan tâm giúp đỡ của nhiều tập thể và cá nhân.

Trước hết, chúng em xin gửi tới toàn thể các thầy, cô giáo dạy môn Cấu trúc dữ liệu và thuật toán. Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc tới Giáo viên PGS.TS. Nguyễn Thị Hồng Minh, người đã tận tâm hướng dẫn chúng em trong suốt quá trình hoàn thiện đề tài.

Do kiến thức và kinh nghiệm còn hạn cho nên đề tài của nhóm chúng em còn nhiều thiếu sót, kính mong được sự đánh giá, góp ý của quý thầy cô.

Cuối cùng, chúng em kính chúc quý thầy, cô dồi dào sức khỏe và thành công trong sự nghiệp "trồng người" của mình.

Hà Nội, ngày 12 tháng 10 năm 2023

# Mục lục

|  |  |  |
| --- | --- | --- |
| [**1 Cấu trúc dữ liệu: Priority Queue**](#_bookmark0)  [1.1 Priority Queue là gì?](#_bookmark1) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | .6 | **6** |
| [1.2 Cách triển khai hàng đợi ưu tiên priority queue](#_bookmark2) . . . . . . . . . . . . . | .6 |  |
| [1.2.1 Triển khai hàng đợi ưu tiên bằng mảng](#_bookmark3) . . . . . . . . . . . . . . | .6 |  |
| [1.2.2 Triển khai hàng đợi ưu tiên bằng danh sách liên kết](#_bookmark4) . . . . . . . | .6 |  |
| [1.2.3 Triển khai hàng đợi ưu tiên bằng cách sử dụng đống (Heap)](#_bookmark5) . . | .7 |  |
| [1.2.4 Triển khai hàng đợi ưu tiên bằng cây tìm kiếm nhị phân](#_bookmark6) . . . . | .7 |  |
| [1.3 Ưu điểm của hàng đợi ưu tiên](#_bookmark7) . . . . . . . . . . . . . . . . . . . . . . . | .8 |  |
| [1.4 Nhược điểm của hàng đợi ưu tiên](#_bookmark8) . . . . . . . . . . . . . . . . . . . . . | .8 |  |
| [**2 Thuật toán**](#_bookmark9)  [2.1 Binary Search](#_bookmark10) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | .8 | **8** |
| [2.2 Thuật toán Dijkstra](#_bookmark11) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | .9 |  |
| [**3 Demo**](#_bookmark12) |  | **9** |
| [**Tài liệu tham khảo**](#_bookmark13) |  | **19** |

**Lời mở đầu**

Hiện nay, dự án “Xe đạp đô thị” tại Hà Nội đang rất được mọi người quan tâm và hưởng ứng nhiệt tình. Đây là một hình thức giao thông đô thị mới đề cao sự văn minh, hiện đại, tiện lợi, và đặc biệt là an toàn với sức khỏe người dùng và thân thiện với môi trường.

Dựa trên dự án đó, nhóm chúng em muốn thực hiện một chương trình về “Trạm cho thuê xe đạp” nhằm giúp mọi người có những trải nghiệm tốt hơn và tuyệt vời hơn.

Bằng cách sử dụng cấu trúc hàng đợi ưu tiên, thuật toán Binary Search và Dijkstra vào hệ thống quản lý các trạm và dịch vụ chăm sóc khách hàng. Khách hàng khi đến thuê xe sẽ được gợi ý trạm gần nhất và được sử dụng xe đạp tốt nhất hiện có ở trạm. Và khi khách hàng trả xe hệ thống cũng sẽ đưa ra trạm xe gần nhất để có thể giúp khách hàng tiết kiệm thời gian di chuyển và không bị lạc đường.

# Cấu trúc dữ liệu: Priority Queue

## Priority Queue là gì?

Priority queue là loại hàng đợi trong đó các phần tử có giá trị ưu tiên cao hơn được truy xuất trước các phần tử có giá trị ưu tiên thấp hơn. (Khác với queue là phần tử nào vào trước thì truy xuất trước)

Trong hàng đợi ưu tiên, mỗi phần tử có một giá trị ưu tiên gắn liền với nó. Mức độ ưu tiên được xác định tùy vào mục đích sử dụng. Ví dụ: phần tử có giá trị cao nhất thì sẽ có mức độ ưu tiên cao nhất hoặc ngược lại, phần tử có giá trị nhỏ nhất sẽ có mức độ ưu tiên cao nhất.

Các phương thức cơ bản ở trong một hàm đợi ưu tiên:

* + - Chèn phần tử vào hàng đợi ưu tiên: enqueue()/insert()
    - Xóa phần tử trong hàng đợi ưu tiên: dequeue()/remove()
    - Lấy ra phần tử có độ ưu tiên cao nhất trong hàng đợi ưu tiên: getTheHighestPrior- ity()

## Cách triển khai hàng đợi ưu tiên priority queue

Hàng đợi ưu tiên có thể được triển khai bằng cách sử dụng các cấu trúc dữ liệu sau: Mảng; Danh sách liên kết; Cấu trúc dữ liệu đống; Cây tìm kiếm nhị phân.

* + 1. **Triển khai hàng đợi ưu tiên bằng mảng**

Gồm các phương thức:

* + - * insert(k,v): thêm phần tử có độ ưu tiên k và giá trị v vào hàng đợi
      * getTheHighestPriority(): lấy phần tử có độ ưu tiên cao nhất
      * removeTheHighestPriority(); xóa phần tử có độ ưu tiên cao nhất

Trong phần triển khai bằng mảng, ta có thể sử dụng mảng có sắp xếp và mảng không sắp xếp. Trong cả 2 loại thì đều có sự đánh đổi: nếu chèn phần tử mới vào nhanh thì xóa phần tử có độ ưu tiên cao nhất sẽ lâu và ngược lại. Dưới đây là bảng thể hiện độ phức tạp thời gian của các phương thức đối với 2 loại mảng:

|  |  |  |
| --- | --- | --- |
|  | insert(k,v) | removeTheHighestPriority() |
| Mảng có sắp xếp | O(n) | O(1) |
| Mảng không sắp | O(1) | O(n) |

* + 1. **Triển khai hàng đợi ưu tiên bằng danh sách liên kết**

Gồm các phương thức:

* + - * insert(k,v): thêm phần tử có độ ưu tiên k và giá trị v vào hàng đợi
      * getTheHighestPriority(): lấy phần tử có độ ưu tiên cao nhất
      * removeTheHighestPriority(); xóa phần tử có độ ưu tiên cao nhất

Cũng như mảng, đối với triển khai hàng đợi ưu tiên bằng danh sách liên kết ta có thể dùng danh sách liên kết có sắp xếp hoặc danh sách liên kết không sắp xếp. Trong cả 2 loại thì đều có sự đánh đổi: nếu chèn phần tử mới vào nhanh thì xóa phần tử có độ ưu tiên cao nhất sẽ lâu và ngược lại. Dưới đây là bảng thể hiện độ phức tạp thời gian của các phương thức đối với 2 loại danh sách liên kết:

|  |  |  |
| --- | --- | --- |
|  | insert(k,v) | removeTheHighestPriority() |
| Danh sách liên kết  có sắp xếp | O(n) | O(1) |
| Danh sách liên kết  không sắp xếp | O(1) | O(n) |

* + 1. **Triển khai hàng đợi ưu tiên bằng cách sử dụng đống (Heap)**

Đống nhị phân thường được ưa thích để triển khai hàng đợi ưu tiên vì đống cung cấp hiệu suất tốt hơn so với mảng hay danh sách liên kết. Vậy đống là gì?

Đống là một cây nhị phân lưu trữ các khóa tại các nút của nó và phải thỏa mãn 2 tính chất sau:

* + - * Tính chất 1: Là một cây nhị phân đầy đủ
      * Tính chất 2: Mỗi nút trên cây đều chứa 1 nhãn lớn hơn hoặc bằng các con của nó (nếu có) và nhỏ hơn hoặc bằng nút cha (trừ nút gốc vì nút gốc là lớn nhất)

Một cấu trúc như thế được gọi là maxHeap. Tương tự ta có thể thay đổi tính chất 2 là: mỗi nút trên cây đều chứa 1 nhãn nhỏ hơn hoặc bằng nút con của nó (nếu có) và lớn hơn hoặc bằng nút cha (trừ nút gốc vì nút gốc là nhỏ nhất) thì ta sẽ có được minHeap.

Chiều cao của đống: một đống chứa n nút sẽ có chiều cao là h = O(log(n)).

Cách triển khai hàng đợi ưu tiên bằng đống (PQ - Heap): Chúng ta sẽ dùng mảng để lưu trữ các phần tử của Heap mà không phải dùng cấu trúc liên kết vì Heap là một cây nhị phân hoàn chỉnh nên việc dùng mảng để lưu trữ giúp tiết kiệm chi phí lưu trữ con trỏ đến các nút con.

Các phương thức:

* + - * upHeap(): thêm một nút vào cây
      * downHeap(): xóa nút gốc khỏi cây
      * findMin()/findMax(): tìm phần tử có độ ưu tiên lớn nhất

Khi khai triển hàng đợi ưu tiên bằng cấu trúc dữ liệu đống, chúng ta có thể cân bằng thời gian trong cả 2 phương thức thêm phần tử và xóa phần tử mà không cần phải đánh đổi như khi khai triển bằng mảng hay cấu trúc liên kết. Dưới đây là bảng thể hiện độ phức tạp thời gian khi triển khai PQ - queue bằng đống nhị phân:

|  |  |  |
| --- | --- | --- |
|  | upHeap() | downHeap() |
| Đống nhị phân  (Binary Heap) | O(logN) | O(lobN) |

* + 1. **Triển khai hàng đợi ưu tiên bằng cây tìm kiếm nhị phân**

Cây tìm kiếm nhị phân cân bằng AVL tree cũng có thể được sử dụng để triển khai hàng đợi ưu tiên.

Gồm các phương thức:

* + - * insert(k,v): thêm phần tử có độ ưu tiên k và giá trị v vào hàng đợi
      * getTheHighestPriority(): lấy phần tử có độ ưu tiên cao nhất
      * removeTheHighestPriority(): xóa phần tử có độ ưu tiên cao nhất

Bảng thể hiện độ phức tạp thời gian khi triển khai hàng đợi ưu tiên bằng cây tìm kiếm nhị phân:

|  |  |  |
| --- | --- | --- |
|  | insert(k,v) | removeTheHighestPriority() |
| Cây tìm kiếm  nhị phân | O(logN) | O(logN) |

## Ưu điểm của hàng đợi ưu tiên

Hàng đợi ưu tiên giúp truy cập các phần tử một cách nhanh hơn. Điều này là do các phần tử trong hàng đợi ưu tiên được sắp xếp theo mức độ ưu tiên cao nhất mà không cần phải tìm kiếm trong toàn bộ hàng đợi.

Trong hàng đợi ưu tiên việc sắp xếp các phần tử được thực hiện linh hoạt. Các phần tử trong hàng đợi ưu tiên có thể được cập nhật các giá trị ưu tiên, điều này cho phép hàng đợi tự động sắp xếp lại khi mức độ ưu tiên thay đổi.

Hàng đợi ưu tiên được sử dụng trong nhiều thuật toán để cải thiện hiệu quả của chúng, chẳng hạn như thuật toán Dijkstra để tìm đường đi ngắn nhất trong biểu đồ và thuật toán tìm kiếm A\* để tìm đường đi ngắn nhất.

Hàng đợi ưu tiên cho phép bạn nhanh chóng truy xuất phần tử có mức ưu tiên cao nhất, chúng thường được sử dụng trong các hệ thống thời gian thực trong đó thời gian là điều cốt yếu.

## Nhược điểm của hàng đợi ưu tiên

Hàng đợi ưu tiên có độ phức tạp cao. Nó phức tạp hơn các cấu trúc dữ liệu đơn giản như mảng và danh sách liên kết. Đồng thời có thể khó triển khai và bảo trì hơn.

Hàng đợi ưu tiên tiêu thụ bộ nhớ cao. Việc lưu trữ giá trị ưu tiên cho từng phần tử trong hàng đợi ưu tiên có thể chiếm thêm bộ nhớ, điều này có thể là mối lo ngại trong các hệ thống có tài nguyên hạn chế.

Hàng đợi ưu tiên không phải lúc nào cũng là cấu trúc dữ liệu hiệu quả nhất. Trong một số trường hợp, các cấu trúc dữ liệu khác như đống hoặc cây tìm kiếm nhị phân có thể hiệu quả hơn đối với một số thao tác nhất định, như tìm phần tử tối thiểu hoặc tối đa trong hàng đợi.

Thứ tự của các phần tử trong hàng đợi ưu tiên được xác định bởi các giá trị ưu tiên của chúng nên các phần tử được truy xuất có thể khó dự đoán hơn so với các cấu trúc dữ liệu khác như ngăn xếp hoặc hàng đợi (tuân theo nguyên tắc vào trước ra trước (FIFO) hoặc vào sau ra trước (LIFO)).

# Thuật toán

## Binary Search

Ý tưởng của thuật toán là sẽ liên tục chia không gian tìm kiếm thành hai nửa và loại một nửa đi. Thuật toán có thể trình bày như sau: Với A là mảng được sắp xếp cho trước,

* + - Ta duy trì một không gian tìm kiếm S là một dãy con các giá trị có thể là kết quả. Ban đầu, không gian tìm kiếm là toàn bộ các chỉ số của mảng S=0,. . . ,n-1 với n là chỉ số phần tử cuối cùng của A.
    - Ở mỗi bước, thuật toán so sánh giá trị cần tìm với phần tử có chỉ số là trung vị trong không gian tìm kiếm. Dựa trên sự so sánh đó, cộng thêm việc ta biết dãy A có thứ tự, ta có thể loại một nửa số phần tử của S.
    - Lặp đi lặp lại quá trình này, cuối cùng ta sẽ được một không gian tìm kiếm bao gồm một phần tử duy nhất. Khi đó, nếu phần tử duy nhất đó bằng với giá trị cần tìm x thì đó là nghiệm của bài toán, nếu không thì bài toán vô nghiệm.

Ở mỗi bước, kích thước không gian tìm kiếm bị giảm đi một nửa. Ta dễ thấy rằng độ phức tạp của thuật toán là O(log(N)) với N là số phần tử ban đầu của không gian tìm kiếm.

Hàm log là một hàm tăng rất chậm. Ví dụ như nếu phải tìm kiếm giá trị trong 1 triệu phần tử, với tìm kiếm nhị phân chỉ cần tối đa là 21 bước.

## Thuật toán Dijkstra

Thuật toán Dijkstra dùng để giải quyết bài toán đường đi ngắn nhất một nguồn (Single-source shortest path), đồ thị trọng số không âm. Thuật toán Dijkstra có thể áp dụng cho cả đồ thị có hướng hoặc vô hướng.

Cho một đồ thị vô hướng với N đỉnh (được đánh số từ 0 đến N-1), M cạnh có trọng số, và một đỉnh nguồn S. Trọng số của tất cả các cạnh đều không âm. Yêu cầu tìm ra đường đi ngắn nhất từ đỉnh S tới tất cả các đỉnh còn lại (hoặc cho biết nếu không có đường đi).

Ý tưởng hoạt động của thuật toán Dijkstra như sau:

* + - B1: Đặt giá trị khoảng cách ban đầu của đỉnh xuất phát là 0. Khởi tạo khoảng cách tới các đỉnh khác là +*∞*
    - B2: Chọn đỉnh có khoảng cách nhỏ nhất và chưa được xét.
    - B3: Duyệt qua các đỉnh kề với đỉnh đang xét, nếu tổng khoảng cách từ đỉnh xuất phát đến đỉnh đang xét và trọng số của cạnh nối giữa hai đỉnh là nhỏ hơn khoảng cách hiện tại của đỉnh đó, cập nhật khoảng cách mới.
    - B4: Đánh dấu đỉnh đã xét.
    - B5: Lặp lại các bước 2-4 cho đến khi tất cả các đỉnh đã được xét.

Thuật toán Dijkstra giải quyết bài toán đường đi ngắn nhất một nguồn trong thời gian O((E + V)logV). Với E là số cạnh, V là số đỉnh của đồ thị.

# Demo

Với bài toán "Trạm cho thuê xe đạp" nhóm chúng em sẽ viết dự án BICYCLE\_MANAGEMENT gồm các class sau:

* Bicycle
* BicyclePriorityManager
* Customer
* FormatMoney
* Station
* Test
* Ticket

## Class: Bicycle

1



package BICYCLE\_ MANAGEMENT ;

import java . util . concurrent . atomic . Atomic Integer ; public class Bicycle {

private

private private private

int id;

final long durable Time = 1000000 ;//hour

double

static

time Traveled ;//hour

final String ID\_ FORMAT ="BIKE 04d";

private static Atomic Integer id Counter = new Atomic Integer ( 2300 ) ;

public Bicycle () {

this . id = generate Unique Id (); this . time Traveled = 0;

}

//Functiontocreateauniqueidforeachbicycle private int generate Unique Id () {

return id Counter . get And Increment ();

}

public String get Id () {

return String . format ( ID\_ FORMAT , id);

}

public long get Durable Time () { return durable Time ;

}

public double get Time Traveled () { return time Traveled ;

}

public double update Traveled Time ( double time ) {

return time Traveled += time ;

}

@ Override

public String to String () { return"Bicycle{"+

"id="+ get Id () + ",␣time Traveled="+ time Traveled + ’}’;

}

}

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

## Class: BicyclePriorityManager

1 package BICYCLE\_ MANAGEMENT ;

2

1. public class Bicycle Priority Manager {
2. Bicycle [] station ;
3. int n = 0;
4. protected int default Size = 10000 ;

7

1. public Bicycle Priority Manager () {
2. this . station = new Bicycle [ default Size ];

10 }

1. public boolean is Empty () {
2. return n == 0;

13 }

14

1. //functiontoinsertabicyclefromcustomer
2. public void insert ( Bicycle bicycle ) {
3. if( check To Insert () && check Expired ( bicycle )) {
4. int insertion Index = binary Search (( int ) bicycle . get Time Traveled ());
5. shift And Insert ( insertion Index , bicycle );

20 }

21 }

22

1. //functiontoexportbicyclewhencustomerrent
2. public Bicycle export () {

25 if ( n >= 0) {

1. Bicycle bicycle Export = station [0];
2. shift To Left ();

28 n - -;

1. return bicycle Export ;
2. } else {
3. return null ;

32 }

33 }

34

1. //functiontochecktheexpiredbicycle
2. public boolean check Expired ( Bicycle bicycle ) throws Runtime Exception {
3. if ( bicycle . get Time Traveled () > bicycle . get Durable Time ()) {
4. return false ;

39 }

40 return true ;

41 }

1. public boolean check To Insert (){
2. if ( n >= default Size ) {
3. return false ;

45 }

46 return true ;

47 }

48

1. public void check To Remove () throws Array Index Out Of Bounds Exception {
2. if ( is Empty ()) {
3. throw new Array Index Out Of Bounds Exception ("Priority␣queue␣is␣empty")

;

52 }

53 }

1. private int binary Search ( int key ) {
2. int low = 0;
3. int high = n - 1;

57

1. while ( low <= high ) {
2. int mid = ( low + high ) / 2;
3. int mid Key = ( int ) station [ mid ]. get Time Traveled ();

61

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 62 |  | | if | ( mid Key == key ) { | |
| 63 |  | |  | return mid ; | |
| 64 |  | | } | else if ( mid Key < key ) { | |
| 65 |  | |  | low = mid + 1; | |
| 66 |  | | } | else { | |
| 67 |  | |  | high = mid - 1; | |
| 68 |  | | } |  | |
| 69 |  | | } |  | |
| 70 |  | | return | low ; | |
| 71 | } | |  |  | |
| 72 |  | |  |  | |
| 73 |  | //Shiftelementstomakespaceforthenewentryandinsertit | | |  |
| 74 |  | private void shift And Insert ( int index , Bicycle bicycle ) | | | { |
| 75 |  | for ( int i = n - 1; i >= index ; i - -) { | | |  |
| 76 |  | station [ i + 1] = station [ i]; | | |  |
| 77 |  | } | | |  |
| 78 |  | station [ index ] = bicycle ; | | |  |
| 79 |  | n ++; | | |  |
| 80 |  | } | | |  |
| 81 |  | private void shift To Left () { | | |  |
| 82 |  | for ( int i = 1; i < n; i ++) { | | |  |
| 83 |  | station [ i - 1] = station [ i]; | | |  |
| 84 |  | } | | |  |
| 85 |  | } | | |  |
| 86 |  | int size (){ | | |  |
| 87 |  | return n; | | |  |
| 88 |  | } | | |  |
| 89 |  |  | | |  |
| 90 |  | Bicycle get Bike ( int i){ | | |  |
| 91 |  | if( i >=0 && i< n){ | | |  |
| 92 |  | return station [ i]; | | |  |
| 93 |  | } else { | | |  |
| 94 |  | return null ; | | |  |
| 95 |  | } | | |  |
| 96 |  | } | | |  |
| 97 | } |  | | |  |

## Class: Customer

1



package BICYCLE\_ MANAGEMENT ;

import java . time . Duration ; import java . time . Local Time ; import java . time . Local Date Time ;

import java . time . temporal . Chrono Unit ; import java . util . Scanner ;

import java . util . concurrent . atomic . Atomic Integer ;

public class Customer { private String name ; private int id Name ;

private static final String ID\_ FORMAT ="2200 4d"; private Ticket current Ticket ;

private Local Time bicycle Rental Time ; private Bicycle bicycle Rented ;

private static Atomic Integer id Counter = new Atomic Integer ( 22000000 ) ;

public Customer ( String name ) { this . name = name ;

this . id Name = generate Unique Id ();

}

//FunctiontocreateanuniqueIdforeachcustomer; private int generate Unique Id () {

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27 }

28

return id Counter . get And Increment ();

1. public void buy Ticket For Hour ( int time ) {
2. Ticket ticket = new Ticket ();
3. this . current Ticket = ticket . create Ticket For Hour ( time );

32 }

33

1. public void buy Ticket For Daily () {
2. Ticket ticket = new Ticket ();
3. this . current Ticket = ticket . create Ticket For Daily ();

37 }

38

1. public void rent Bicycle ( Station station ) {
2. Local Time current Time = Local Time . now (). truncated To ( Chrono Unit . MINUTES )

;

1. //Checkifcustomerhasn’tticketsorticketisexpired
2. if ( this . current Ticket == null ) {
3. System . out . println ("Please␣purchase␣a␣new␣ticket␣!!!");
4. } else if ( this . current Ticket . is Expired ( current Time )) {
5. System . out . println ("Your␣rental␣period␣has␣expired,␣please␣purchase

␣a␣new␣ticket␣!!!");

1. } else {
2. this . bicycle Rented = station . rent Bicycle ();
3. //Rentingsuccess
4. this . bicycle Rental Time = current Time ;
5. //updatebicycletimetraveled
6. Duration duration = Duration . between ( bicycle Rental Time , current Ticket . get Valid Time ());
7. long update Time = duration . get Seconds ();
8. double traveled Hours = update Time \* 1.0 / 60 / 60;
9. this . bicycle Rented . update Traveled Time ( traveled Hours );
10. System . out . println ("RENTING␣SUCCESS!!!!");
11. //System.out.println("Timeavailable:"+current Ticket. get Valid Time());
12. //System.out.println("Rentedbikewithrunningtime:"+this.

bicycle Rented.get Time Traveled());//printtimetraveledofbicycle

58 }

59 }

60

1. public void return Bicycle ( Station station ) {
2. Local Time current Time = Local Time . now (). truncated To ( Chrono Unit . MINUTES )

. plus Hours (4) . plus Minutes (20) ;

1. System . out . println ("Time␣to␣return␣bike:␣"+ current Time );
2. if ( this . current Ticket == null ) {

|  |  |  |
| --- | --- | --- |
| 65 |  | System . out . println ("There␣is␣no␣car␣to␣return"); |
| 66 | } | else { |
| 67 |  | //checkexpiredticket |
| 68 |  | if (! this . current Ticket . is Expired ( current Time )) { |
| 69 |  | System . out . println ("The␣current␣ticket␣has␣not␣expired.␣Do␣you␣ |
|  |  | want␣to␣return␣the␣car?\n"+ |
| 70 |  | "Yes␣:␣Please␣enter␣1.\n"+ |
| 71 |  | "No␣:␣Please␣enter␣0␣to␣exit.\n"); |
| 72 |  | Scanner sc = new Scanner ( System . in); |
| 73 |  | int choose = sc. next Int (); |
| 74 |  | if ( choose == 1) { |
| 75 |  | station . insert Bicycle ( this . bicycle Rented ); |
| 76 |  | } else if ( choose == 0) { |
| 77 |  | return ; |
| 78 |  | } else { |
| 79 |  | System . out . println ("Please␣choose␣your␣option!!"); |
| 80 |  | } |
| 81 |  | } else { |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 82  83  84  85  86  87  88  89  90  91  92 | station . insert Bicycle ( this . bicycle Rented );  Duration ticket Duration = Duration . between ( current Ticket . get Purchase Time () , current Time );  if ( ticket Duration . to Hours () < 0) { ticket Duration . plus Hours (24) ;  }  Duration max Duration = Duration . between ( current Ticket . get Purchase Time () , current Ticket . get Valid Time ());  if ( ticket Duration . compare To ( max Duration ) > 0) {  Duration overdue = ticket Duration . minus ( max Duration ); long hours = overdue . to Hours ();  long minutes = overdue . to Minutes ()  60;  System . out . println ("The␣ticket␣is␣overdue:␣"+ hours +"␣  hours,␣"+ minutes +"␣minutes"); | | | |
| 93 | double extra Money = (( hours + ( minutes \* 1.0 / | 60) ) | \* | this . |
|  | current Ticket . TICKET\_ FOR\_ HOUR ); |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 94  95 |  | |  |  | } | System . out . println ("You␣need␣to␣pay␣extra:␣"+ new  Format Money ( extra Money ). format ()); | |
| 96 |  | |  | } |  |  | |
| 97 |  | | } |  |  |  | |
| 98 | } | |  |  |  |  | |
| 99 |  | |  |  |  |  | |
| 100 |  | |  |  |  |  | |
| 101 | //toextendticketvalidity | | | | | |  |
| 102 | public void renewal Ticket ( int extra Hours ) { | | | | | |  |
| 103 | this . current Ticket . get Valid Time (). plus Hours ( extra Hours ); | | | | | |  |
| 104 | } | | | | | |  |
| 105 |  | | | | | |  |
| 106 | public String get Name () { | | | | | |  |
| 107 | return name ; | | | | | |  |
| 108 | } | | | | | |  |
| 109 |  | | | | | |  |
| 110 | public int get Id Name () { | | | | | |  |
| 111 | return id Name ; | | | | | |  |
| 112 | } | | | | | |  |
| 113 |  | | | | | |  |
| 114 | public String get Current Ticket Infor () { | | | | | |  |
| 115 | if ( this . current Ticket != null ) { | | | | | |  |
| 116 | String ticket Type =""; | | | | | |  |
| 117 | int ticket Price = this . current Ticket . get Price (); | | | | | |  |
| 118 | if ( this . current Ticket . get Valid Time (). equals ( Local Time . MAX )) | | | | | | { |
| 119 | ticket Type ="Daily␣Ticket"; | | | | | | |
| 120 | } else { | | | | | | |
| 121 | ticket Type ="Hourly␣Ticket"; | | | | | | |
| 122 | } | | | | | | |
| 123 |  | | | | | | |
| 124 | return"#########TICKET␣INFORMATION#########\n"+ | | | | | | |
| 125 | "ID:␣"+ this . id Name + | | | | | | |
| 126 | "\n Name:␣"+ this . name + | | | | | | |
| 127 | "\n Ticket␣Type:␣"+ ticket Type + | | | | | | |
| 128  129 | "\n Ticket␣Price:␣"+ new Format Money ( ticket Price ). format ()  +  "\n Purchase␣Time:␣"+ this . current Ticket . get Purchase Time () | | | | | | |
| 130 | +  "\n Valid␣Until:␣"+ this . current Ticket . get Valid Time () +"\n | | | | | | |
|  | "+ | | | | | | |
| 131 |  |  | "#####################################"; | | | | |
| 132 |  |  | } | | | | |
| 133 |  |  | return"No␣ticket␣purchased␣yet."; | | | | |
| 134 |  | } |  | | | | |
| 135 | } |  |  | | | | |

## FormatMoney

1

package BICYCLE\_ MANAGEMENT ;

2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | import | | java . text . Number Format ; | | | |
| 4 | import | | java . util . Locale ; | | | |
| 5 |  | |  | | | |
| 6 | public | | class Format Money { | | | |
| 7 |  | private double money ; | | | | |
| 8 |  | public Format Money ( double money ) { | | | | |
| 9 |  | this . money = money ; | | | | |
| 10 |  | } | | | | |
| 11 |  |  | | | | |
| 12 |  | String format (){ | | | | |
| 13 |  | Locale locale VN = new Locale ("vi","VN"); | | | | |
| 14 |  | Number Format currency VN = Number Format . get Currency Instance ( locale VN ); | | | | |
| 15 |  | String result = currency VN . format ( this . money ); | | | | |
| 16 |  | return result ; | | | | |
| 17 |  | } | | | | |
| 18 | } |  | | | | |
| **Class: Station** | | | | | | |
| 1 | package BICYCLE\_ MANAGEMENT ; | | | |  |  |
| 2 |  | | | |  |  |
| 3 | public class Station { | | | |  |  |
| 4 | private Bicycle Priority Manager station ; | | | |  |  |
| 5 | private String name ; | | | |  |  |
| 6 |  | | | |  |  |
| 7 | public Station ( String name ) { | | | |  |  |
| 8 | this . station = new Bicycle Priority Manager (); | | | |  |  |
| 9 | this . name = name ; | | | |  |  |
| 10 | } | | | |  |  |
| 11 |  | | | |  |  |
| 12 | public Station ( Bicycle Priority Manager station , String | | | | name ) | { |
| 13 |  | | this . station = station ; | | | |
| 14 |  | | this . name = name ; | | | |
| 15 | } | |  | | | |
| 16 |  | |  | | | |
| 17 |  | public void insert Bicycle ( Bicycle | | bicycle ) { | | |
| 18 |  | station . insert ( bicycle ); | |  | | |
| 19 |  | } | |  | | |
| 20 |  |  | |  | | |
| 21 |  | public Bicycle rent Bicycle () { | |  | | |
| 22 |  | return station . export (); | |  | | |
| 23 |  | } | |  | | |
| 24 |  |  | |  | | |
| 25 |  | public boolean is Station Empty () { | |  | | |
| 26 |  | return station . is Empty (); | |  | | |
| 27 |  | } | |  | | |
| 28 |  |  | |  | | |
| 29 | } |  | |  | | |

## Class: Ticket

1

package BICYCLE\_ MANAGEMENT ;

import java . time . Local Date Time ; import java . time . Local Time ;

import java . time . temporal . Chrono Unit ;

public class Ticket {

private Local Time purchase Time ;//timewhencustomerbuyticket private Local Time valid Time ;//timevalidity

private int price ;//Ticketprice

2

3

4

5

6

7

8

9

10

|  |  |  |
| --- | --- | --- |
| 11  12  13  14  15  16  17 | protected final int TICKET\_ FOR\_ HOUR = 20000 ;  protected final int TICKET\_ FOR\_ DAILY = 150000 ;  public Ticket create Ticket For Hour ( int time ) { Ticket ticket = new Ticket ();  ticket . purchase Time = Local Time . now (). truncated To ( Chrono Unit . MINUTES ); ticket . valid Time = ticket . purchase Time . plus Hours ( time ). truncated To (  Chrono Unit . MINUTES ); | |
| 18 |  | ticket . price = TICKET\_ FOR\_ HOUR \* time ; |
| 19 |  | return ticket ; |
| 20 |  | } |
| 21 |  |  |
| 22 |  | public Ticket create Ticket For Daily () { |
| 23 |  | Ticket ticket = new Ticket (); |
| 24 |  | ticket . purchase Time = Local Time . now (). truncated To ( Chrono Unit . MINUTES ); |
| 25 |  | ticket . valid Time = Local Time . MAX . truncated To ( Chrono Unit . MINUTES ); |
| 26 |  | ticket . price = TICKET\_ FOR\_ DAILY ; |
| 27 |  | return ticket ; |
| 28 |  | } |
| 29 |  |  |
| 30 |  | //functiontoextendticketvalidity |
| 31 |  | public void extend Validity ( int additional Hours ) { |
| 32 |  | this . valid Time = this . valid Time . plus Hours ( additional Hours ); |
| 33 |  | } |
| 34 |  |  |
| 35 |  | public int get Price () { |
| 36 |  | return this . price ; |
| 37 |  | } |
| 38 |  |  |
| 39 |  | public Local Time get Purchase Time () { |
| 40 |  | return this . purchase Time ; |
| 41 |  | } |
| 42 |  |  |
| 43 |  | public Local Time get Valid Time () { |
| 44 |  | return this . valid Time ; |
| 45 |  | } |
| 46 |  |  |
| 47 |  | public boolean is Expired ( Local Time current Time ) { |
| 48 |  | return current Time . is After ( valid Time ); |
| 49 |  | } |
| 50 | } |  |

## Class: Test

1

package BICYCLE\_ MANAGEMENT ;

public class Test { static void test1 (){

Station station 1 = new Station ("Thanh␣X u n"); Station station 2 = new Station ("ng␣a") ; Station station 3 = new Station ("H o n␣K i ỉm") ;

for ( int i = 0; i < 100 ; i ++) { Bicycle bicycle = new Bicycle (); station 1 . insert Bicycle ( bicycle );

}

for ( int i = 0; i < 100 ; i ++) { Bicycle bicycle = new Bicycle (); station 2 . insert Bicycle ( bicycle );

}

for ( int i = 0; i < 100 ; i ++) { Bicycle bicycle = new Bicycle ();

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

|  |  |  |
| --- | --- | --- |
| 21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | station 3 . insert Bicycle ( bicycle );  }  System . out . println ( station 3 . is Station Empty ()); Customer customer 1 = new Customer ("Vu␣Quang␣Huy"); customer 1 . buy Ticket For Hour (3) ;  System . out . println ( customer 1 . get Current Ticket Infor ()); customer 1 . rent Bicycle ( station 1 );  //Customercustomer 2=newCustomer("Vuquanghuy");  //customer 2.rent Bicycle(customer2,station 1); customer 1 . return Bicycle ( station 1 );  }  static void test2 (){  Bicycle Priority Manager bicycle Priority Manager 1 = new Bicycle Priority Manager ();  Station s1 = new Station ( bicycle Priority Manager 1 ,"station 1"); Bicycle b1 = new Bicycle ();  Bicycle b2 = new Bicycle (); Bicycle b3 = new Bicycle (); Bicycle b4 = new Bicycle (); Bicycle b5 = new Bicycle (); b1 . update Traveled Time ( 230) ; b2 . update Traveled Time ( 123) ; b3 . update Traveled Time (33) ; b4 . update Traveled Time (34) ; b5 . update Traveled Time (90) ;  bicycle Priority Manager 1 . insert ( b1 ); bicycle Priority Manager 1 . insert ( b2 ); bicycle Priority Manager 1 . insert ( b3 ); bicycle Priority Manager 1 . insert ( b4 ); bicycle Priority Manager 1 . insert ( b5 );  System . out . println ("=======================");  for ( int i = 0; i < bicycle Priority Manager 1 . size (); i ++) { | |
| 55 |  | System . out . println ( bicycle Priority Manager 1 . get Bike ( i)+"␣"); |
| 56 |  | } |
| 57 |  | System . out . println ("======================="); |
| 58 |  | Customer c1 = new Customer ("hang"); |
| 59 |  | c1 . buy Ticket For Hour (5) ; |
| 60 |  | System . out . println ( c1 . get Current Ticket Infor ()); |
| 61 |  | c1 . rent Bicycle ( s1 ); |
| 62 |  | c1 . return Bicycle ( s1 ); |
| 63 |  | System . out . println ("########################"); |
| 64 |  | for ( int i = 0; i < bicycle Priority Manager 1 . size (); i ++) { |
| 65 |  | System . out . println ( bicycle Priority Manager 1 . get Bike ( i)+"␣"); |
| 66 |  | } |
| 67 |  | System . out . println ("########################"); |
| 68 |  | } |
| 69 |  | public static void main ( String [] args ) { |
| 70 |  | test2 (); |
| 71 |  | } |
| 72 | } |  |

Kết quả khi chạy thử test1:

1

=======================

Bicycle { id= BIKE2302 , time Traveled = 33 . 0 } Bicycle { id= BIKE2303 , time Traveled = 34 . 0 } Bicycle { id= BIKE2304 , time Traveled = 90 . 0 } Bicycle { id= BIKE2301 , time Traveled = 123 . 0 } Bicycle { id= BIKE2300 , time Traveled = 230 . 0 }

=======================

######### TICKET INFORMATION #########

2

3

4

5

6

7

8

9

ID: 22000000

Name : hang

Ticket Type : Hourly Ticket Ticket Price : 100 . 000 Purchase Time : 18: 51

Valid Until : 23: 51 ##################################### RENTING SUCCESS !!!!

Time to return bike : 23: 11

The current ticket has not expired . Do you want to return the car ? Yes : Please enter 1.

No : Please enter 0 to exit .

1 ########################

Bicycle { id= BIKE2303 , time Traveled = 34 . 0 } Bicycle { id= BIKE2302 , time Traveled = 38 . 0 } Bicycle { id= BIKE2304 , time Traveled = 90 . 0 } Bicycle { id= BIKE2301 , time Traveled = 123 . 0 } Bicycle { id= BIKE2300 , time Traveled = 230 . 0 } ########################

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

Hiện tại nhóm em đang trong quá trình hoàn thiện phần gợi ý trạm xe gần nhất và test các hàm đã viết được và sẽ báo cáo trong bản báo cáo cuối ạ.

# Tài liệu tham khảo

* 1. Vnoi, *"Binary Search"*,

<https://vnoi.info/wiki/algo/basic/binary-search.md>

* 1. Techie, *"Thuật toán dijkstra và ứng "*,

<https://techie.vn/thuat-toan-dijkstra-va-ung-dung/>

* 1. Hackerearth, *"Dijkstra algoritm"*,

<https://www.hackerearth.com/practice/notes/dijkstras-algorithm/>