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

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.

1 Cấu trúc dữ liệu: Priority Queue

1.1 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: getTheHighestPriority()

1.2 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.

1.2.1 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)

1.2.2 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)

1.2.3 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)

1.2.4 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)

1.3 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.

1.4 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)).

2 Thuật toán

2.1 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.

2.2 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 $+\infty$
- 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.

3 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

```
package BICYCLE_MANAGEMENT;

import java.util.concurrent.atomic.AtomicInteger;

public class Bicycle {

    private int id;
    private final long durableTime = 1000000;  //hour
    private double timeTraveled; //hour
    private static final String ID_FORMAT = "BIKE%04d";

    private static AtomicInteger idCounter = new AtomicInteger(2300);



    public Bicycle() {
        this.id = generateUniqueId();
        this.timeTraveled = 0;
    }

    // Function to create a unique id for each bicycle
    private int generateUniqueId() {
        return idCounter.getAndIncrement();
    }

    public String getId() {
        return String.format(ID_FORMAT, id);
    }

    public long getDurableTime() {
        return durableTime;
    }

    public double getTimeTraveled() {
        return timeTraveled;
    }


    public double updateTraveledTime(double time) {
        return timeTraveled += time ;
    }

    @Override
    public String toString() {
        return "Bicycle{" +
                "id=" + getId() +
                ", timeTraveled=" + timeTraveled +
                '}';
    }
}
```

Class: BicyclePriorityManager

```
1   package BICYCLE_MANAGEMENT;
2
3   public class BicyclePriorityManager {
4       Bicycle[] station;
5       int n = 0;
6       protected int defaultSize = 10000;
7
8       public BicyclePriorityManager() {
9           this.station = new Bicycle[defaultSize];
10      }
11      public boolean isEmpty() {
12          return n == 0;
13      }
14
15      // function to insert a bicycle from customer
16      public void insert(Bicycle bicycle) {
17          if(checkToInsert() && checkExpired(bicycle)) {
18              int insertionIndex = binarySearch((int) bicycle.getTimeTraveled());
19              shiftAndInsert(insertionIndex, bicycle);
20          }
21      }
22
23      // function to export bicycle when customer rent
24      public Bicycle export() {
25          if (n >= 0) {
26              Bicycle bicycleExport = station[0];
27              shiftToLeft();
28              n--;
29              return bicycleExport;
30          } else {
31              return null;
32          }
33      }
34
35      // function to check the expired bicycle
36      public boolean checkExpired(Bicycle bicycle) throws RuntimeException{
37          if (bicycle.getTimeTraveled() > bicycle.getDurableTime()) {
38              return false;
39          }
40          return true;
41      }
42      public boolean checkToInsert(){
43          if (n >= defaultSize) {
44              return false;
45          }
46          return true;
47      }
48
49      public void checkToRemove() throws ArrayIndexOutOfBoundsException {
50          if (isEmpty()) {
51              throw new ArrayIndexOutOfBoundsException("Priority queue is empty")
                    ;
52          }
53      }
54      private int binarySearch(int key) {
55          int low = 0;
56          int high = n - 1;
57
58          while (low <= high) {
59              int mid = (low + high) / 2;
60              int midKey = (int) station[mid].getTimeTraveled();
61
```

```
            if (midKey == key) {
                return mid;
            } else if (midKey < key) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return low;
    }

    // Shift elements to make space for the new entry and insert it
    private void shiftAndInsert(int index, Bicycle bicycle) {
        for (int i = n - 1; i >= index; i--) {
            station[i + 1] = station[i];
        }
        station[index] = bicycle;
        n++;
    }
    private void shiftToLeft() {
        for (int i = 1; i < n; i++) {
            station[i - 1] = station[i];
        }
    }
    int size(){
        return n;
    }

    Bicycle getBike(int i){
        if(i>=0 && i<n){
            return station[i];
        } else{
            return null;
        }
    }
}
```

Class: Customer

```
    package BICYCLE_MANAGEMENT;

import java.time.Duration;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;
import java.util.Scanner;
import java.util.concurrent.atomic.AtomicInteger;

public class Customer {
    private String name;
    private int idName;
    private static final String ID_FORMAT = "2200%4d";
    private Ticket currentTicket;
    private LocalTime bicycleRentalTime;
    private Bicycle bicycleRented;
    private static AtomicInteger idCounter = new AtomicInteger(22000000);

    public Customer(String name) {
        this.name = name;
        this.idName = generateUniqueId();
    }

    // Function to create an unique Id for each customer;
    private int generateUniqueId() {
```

```
26          return idCounter.getAndIncrement();
27      }
28
29      public void buyTicketForHour(int time) {
30          Ticket ticket = new Ticket();
31          this.currentTicket = ticket.createTicketForHour(time);
32      }
33
34      public void buyTicketForDaily() {
35          Ticket ticket = new Ticket();
36          this.currentTicket = ticket.createTicketForDaily();
37      }
38
39      public void rentBicycle(Station station) {
40          LocalTime currentTime = LocalTime.now().truncatedTo(ChronoUnit.MINUTES)
                ;
41          // Check if customer hasn't tickets or ticket is expired
42          if (this.currentTicket == null) {
43              System.out.println("Please purchase a new ticket !!!");
44          } else if (this.currentTicket.isExpired(currentTime)) {
45              System.out.println("Your rental period has expired, please purchase
                    a new ticket !!!");
46          } else {
47              this.bicycleRented = station.rentBicycle();
48              // Renting success
49              this.bicycleRentalTime = currentTime;
50              // update bicycle time traveled
51              Duration duration = Duration.between(bicycleRentalTime,
                    currentTicket.getValidTime());
52              long updateTime = duration.getSeconds();
53              double traveledHours = updateTime * 1.0 / 60 / 60;
54              this.bicycleRented.updateTraveledTime(traveledHours);
55              System.out.println("RENTING SUCCESS!!!!");
56  //              System.out.println("Time available: " + currentTicket.
    getValidTime());
57  //              System.out.println("Rented bike with running time: " + this.
    bicycleRented.getTimeTraveled()); // print time traveled of bicycle
58          }
59      }
60
61      public void returnBicycle(Station station) {
62          LocalTime currentTime = LocalTime.now().truncatedTo(ChronoUnit.MINUTES)
                .plusHours(4).plusMinutes(20);
63          System.out.println("Time to return bike: " + currentTime);
64          if (this.currentTicket == null) {
65              System.out.println("There is no car to return");
66          } else {
67              // check expired ticket
68              if (!this.currentTicket.isExpired(currentTime)) {
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.nextInt();
74                  if (choose == 1) {
75                      station.insertBicycle(this.bicycleRented);
76                  } else if (choose == 0) {
77                      return;
78                  } else {
79                      System.out.println("Please choose your option!!");
80                  }
81              } else {
```

```
82                  station.insertBicycle(this.bicycleRented);
83                  Duration ticketDuration = Duration.between(currentTicket.
                        getPurchaseTime(), currentTime);
84                  if (ticketDuration.toHours() < 0) {
85                      ticketDuration.plusHours(24);
86                  }
87                  Duration maxDuration = Duration.between(currentTicket.
                        getPurchaseTime(), currentTicket.getValidTime());
88                  if (ticketDuration.compareTo(maxDuration) > 0) {
89                      Duration overdue = ticketDuration.minus(maxDuration);
90                      long hours = overdue.toHours();
91                      long minutes = overdue.toMinutes() % 60;
92                      System.out.println("The ticket is overdue: " + hours + "
                            hours, " + minutes + " minutes");
93                      double extraMoney = ((hours + (minutes * 1.0 / 60)) * this.
                            currentTicket.TICKET_FOR_HOUR);
94                      System.out.println("You need to pay extra: " + new
                            FormatMoney(extraMoney).format());
95                  }
96              }
97          }
98      }
99
100
101     // to extend ticket validity
102     public void renewalTicket(int extraHours) {
103         this.currentTicket.getValidTime().plusHours(extraHours);
104     }
105
106     public String getName() {
107         return name;
108     }
109
110     public int getIdName() {
111         return idName;
112     }
113
114     public String getCurrentTicketInfor() {
115         if (this.currentTicket != null) {
116             String ticketType = "";
117             int ticketPrice = this.currentTicket.getPrice();
118             if (this.currentTicket.getValidTime().equals(LocalTime.MAX)) {
119                 ticketType = "Daily Ticket";
120             } else {
121                 ticketType = "Hourly Ticket";
122             }
123
124             return "#########TICKET INFORMATION#########\n" +
125                     "ID: " + this.idName +
126                     "\nName: " + this.name +
127                     "\nTicket Type: " + ticketType +
128                     "\nTicket Price: " + new FormatMoney(ticketPrice).format()
                            +
129                     "\nPurchase Time: " + this.currentTicket.getPurchaseTime()
                            +
130                     "\nValid Until: " + this.currentTicket.getValidTime() + "\n
                            " +
131                     "#####################################";
132         }
133         return "No ticket purchased yet.";
134     }
135 }
```

FormatMoney

```
    package BICYCLE_MANAGEMENT;

import java.text.NumberFormat;
import java.util.Locale;

public class FormatMoney {
    private double money;
    public FormatMoney(double money) {
        this.money = money;
    }

    String format(){
        Locale localeVN = new Locale("vi","VN");
        NumberFormat currencyVN = NumberFormat.getCurrencyInstance(localeVN);
        String result = currencyVN.format(this.money);
        return result;
    }
}
```

Class: Station

```
    package BICYCLE_MANAGEMENT;

public class Station {
    private BicyclePriorityManager station;
    private String name;

    public Station(String name) {
        this.station = new BicyclePriorityManager();
        this.name = name;
    }

    public Station(BicyclePriorityManager station, String name) {
        this.station = station;
        this.name = name;
    }

    public void insertBicycle(Bicycle bicycle) {
        station.insert(bicycle);
    }

    public Bicycle rentBicycle() {
        return station.export();
    }

    public boolean isStationEmpty() {
        return station.isEmpty();
    }

}
```

Class: Ticket

```
    package BICYCLE_MANAGEMENT;

import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;

public class Ticket {
    private LocalTime purchaseTime; // time when customer buy ticket
    private LocalTime validTime; // time validity
    private int price; // Ticket price
```

```
    protected final int TICKET_FOR_HOUR = 20000;
    protected final int TICKET_FOR_DAILY = 150000;

    public Ticket createTicketForHour(int time) {
        Ticket ticket = new Ticket();
        ticket.purchaseTime = LocalTime.now().truncatedTo(ChronoUnit.MINUTES);
        ticket.validTime = ticket.purchaseTime.plusHours(time).truncatedTo(
            ChronoUnit.MINUTES);
        ticket.price = TICKET_FOR_HOUR * time;
        return ticket;
    }

    public Ticket createTicketForDaily() {
        Ticket ticket = new Ticket();
        ticket.purchaseTime = LocalTime.now().truncatedTo(ChronoUnit.MINUTES);
        ticket.validTime = LocalTime.MAX.truncatedTo(ChronoUnit.MINUTES);
        ticket.price = TICKET_FOR_DAILY;
        return ticket;
    }

    // function to extend ticket validity
    public void extendValidity(int additionalHours) {
        this.validTime = this.validTime.plusHours(additionalHours);
    }

    public int getPrice() {
        return this.price;
    }

    public LocalTime getPurchaseTime() {
        return this.purchaseTime;
    }

    public LocalTime getValidTime() {
        return this.validTime;
    }

    public boolean isExpired(LocalTime currentTime) {
        return currentTime.isAfter(validTime);
    }
}
```

Class: Test

```
    package BICYCLE_MANAGEMENT;

public class Test {
    static void test1(){
        Station station1 = new Station("Thanh␣Xu  n");
        Station station2 = new Station("   ng   ␣ a ");
        Station station3 = new Station("Ho  n␣K i ỉm ");

        for (int i = 0; i < 100 ; i++) {
            Bicycle bicycle = new Bicycle();
            station1.insertBicycle(bicycle);
        }

        for (int i = 0; i < 100 ; i++) {
            Bicycle bicycle = new Bicycle();
            station2.insertBicycle(bicycle);
        }

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

```
            station3.insertBicycle(bicycle);
        }


        System.out.println(station3.isStationEmpty());
        Customer customer1 = new Customer("Vu␣Quang␣Huy");
        customer1.buyTicketForHour(3);
        System.out.println(customer1.getCurrentTicketInfor());
        customer1.rentBicycle(station1 );
//        Customer customer2 = new Customer("Vu quang huy");
//        customer2.rentBicycle(customer2, station1);
        customer1.returnBicycle(station1);
    }

    static void test2(){
        BicyclePriorityManager bicyclePriorityManager1 = new
            BicyclePriorityManager();
        Station s1 = new Station(bicyclePriorityManager1,"station1");
        Bicycle b1 = new Bicycle();
        Bicycle b2 = new Bicycle();
        Bicycle b3 = new Bicycle();
        Bicycle b4 = new Bicycle();
        Bicycle b5 = new Bicycle();
        b1.updateTraveledTime(230);
        b2.updateTraveledTime(123);
        b3.updateTraveledTime(33);
        b4.updateTraveledTime(34);
        b5.updateTraveledTime(90);
        bicyclePriorityManager1.insert(b1);
        bicyclePriorityManager1.insert(b2);
        bicyclePriorityManager1.insert(b3);
        bicyclePriorityManager1.insert(b4);
        bicyclePriorityManager1.insert(b5);
        System.out.println("======================");
        for (int i = 0; i < bicyclePriorityManager1.size(); i++) {
            System.out.println(bicyclePriorityManager1.getBike(i)+"␣");
        }
        System.out.println("======================");
        Customer c1 = new Customer("hang");
        c1.buyTicketForHour(5);
        System.out.println(c1.getCurrentTicketInfor());
        c1.rentBicycle(s1);
        c1.returnBicycle(s1);
        System.out.println("########################");
        for (int i = 0; i < bicyclePriorityManager1.size(); i++) {
            System.out.println(bicyclePriorityManager1.getBike(i)+"␣");
        }
        System.out.println("########################");
    }
    public static void main(String[] args) {
        test2();
    }
}
```

Kết quả khi chạy thử test1:

```
   ======================
Bicycle{id=BIKE2302, timeTraveled=33.0}
Bicycle{id=BIKE2303, timeTraveled=34.0}
Bicycle{id=BIKE2304, timeTraveled=90.0}
Bicycle{id=BIKE2301, timeTraveled=123.0}
Bicycle{id=BIKE2300, timeTraveled=230.0}
======================
#########TICKET INFORMATION#########
```

```
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, timeTraveled=34.0}
Bicycle{id=BIKE2302, timeTraveled=38.0}
Bicycle{id=BIKE2304, timeTraveled=90.0}
Bicycle{id=BIKE2301, timeTraveled=123.0}
Bicycle{id=BIKE2300, timeTraveled=230.0}
########################
```

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`

[2] Techie, *"Thuật toán dijkstra và ứng "*,
`https://techie.vn/thuat-toan-dijkstra-va-ung-dung/`

[3] Hackerearth, *"Dijkstra algoritm"*,
`https://www.hackerearth.com/practice/notes/dijkstras-algorithm/`