**מגישים :** ארטיום שבלין 321243339, אנה פינצ׳וק 206431082

**מטלה 0**



**אלגוריתם online:**

נתכנן אלגוריתם יעיל במצב און-ליין אשר משתמש במידע הנתון בזה הרגע ולא שומר שום מידע על קריאות המעליות מראש.

נגדיר שבבניין קיימות קומות עם ערך שלילי, כך שהן מהוות קומות כניסה או יציאה מהבניין.

ושאר הקומות אשר גדולות מ-0 הן קומות מגורים/משרדים.

תיאור כללי של האלגוריתם – בהינתן קריאה למעלית כלשהי, המעלית הקרובה והמהירה ביותר תגיע לאותו נוסע ותיקח אותו אל קומת היעד.

אם יש מספר קריאות ליעדים כלשהם, וממוצע זמן סה״כ טיפול בקריאות קטן יותר מזמן טיפול הקריאות של כל אחת בנפרד, אזי נצוות מעלית אחת לכל הקריאות האלו. אחרת נצוות מעליות שונות עבור כל קריאה כזו(במידה ויש מעליות פנויות). אם אין מעליות פנויות נבצע את האיסופים לפי סדר הזימון שלהן(לדוגמא עם יוסי זימן את המעלית לפני דוד וחלי, אזי יוסי יהיה הראשון לטיפול, לאחריו דוד ורק אחריהם חלי).

אלגוריתם שיבוץ מעליות- תחילה נבדוק את סטטוס המעלית ונחלק למקרים:

* המעלית נמצאת במצב מנוחה – נחשב את זמן העצירה + זמן תחילת התנועה + מהירות\*|קומת המקור פחות קומת יעד| (בערך מוחלט).
* המעלית נמצאת בצב ירידה/עליה – נבצע את אותו החישוב, אך נוסיף לשדה שיוך מעלית (תוצאת החישוב של הזימונים הקודמים).

באופן כללי – עבור כל קריאה נבצע את החישוב לכל המעליות שנמצאות בבניין, נשווה עם שדה שיוך המעליות ונבחר את זאת עם זמן ההגעה המינימאלי ביותר ונעדכן את השדות ונבצע את השיוך.

נרצה לבנות מבני נתונים עבור הקריאות כלפי מטה ועבור הקריאות כלפי מעלה.

נשתמש במערך של תורים, ובנוסף נבנה מבני נתונים מסוג מערך עבור כיווני התנועה של המעליות (למעלה/למטה/ללא תנועה).

מקרים לטיפול :

* אם קיים נוסע אשר רוצה לרדת ובדרך ישנו נוסע נוסף אשר מזמן מעלית, נפצל לשתי מקרים:
* הנוסע רוצה לעלות – נחשב את הזמן האופטימאלי עבורו, ובמידה המעלית הזו היא היעילה יותר : נאסוף את הנוסע הזה לאחר שנוריד את הראשון.
* הנוסע רוצה לרדת – נחשב את הזמן האופטימאלי עבורו, ובמידה המעלית הזו היא היעילה יותר : נאסוף את הנוסע **בדרך.**
* אם קיים נוסע אשר רוצה לעלות ובדרך ישנו נוסע נוסף אשר מזמן מעלית, נפצל לשתי מקרים:
* הנוסע רוצה לעלות – נחשב את הזמן האופטימאלי עבורו, ובמידה המעלית הזו היא היעילה יותר : נאסוף את הנוסע **בדרך.**
* הנוסע רוצה לרדת – נחשב את הזמן האופטימאלי עבורו, ובמידה המעלית הזו היא היעילה יותר : נוריד קודם את הנוסע הראשון, ואז נחזור ונאסוף אותו, אחרת נצוות לו את המעלית היעילה ביותר עבורו.

לאחר סיום התהליכים המעליות יישארו במקום האחרון בו הן הורידו את הנוסע ויהיו מוכנות לקריאות הבאות כך שהן יהיו מוגדרות במצב **״מנוחה״**. מכיוון שמדובר באלגוריתם און-ליין אין לנו מידע על הנוסעים הבאים, ולכן לא נרצה לשלוח כל מעלית לקומת הכניסה להמתין (בשונה ממקרה offline).



**אלגוריתם offline :**

נתכנן אלגוריתם יעיל במצב אוף-ליין אשר משתמש במידע הנתון מראש.

נגדיר שבבניין קיימות קומות עם ערך שלילי, כך שהן מהוות קומות כניסה או יציאה מהבניין.

ושאר הקומות אשר גדולות מ-0 הן קומות מגורים/משרדים.

תיאור כללי של האלגוריתם – בהינתן סה״כ הקריאות נחשב את הזמן האופטימאלי עבורן וכך נשייך לכל קריאה מעלית מתאימה.

בהינתן n קריאות – נרצה לתאם לכל מעלית מ 1 עד n קריאות כך שכל הקריאות יבוצעו בזמן הטוב והאופטימאלי ביותר תוך שילוב החלוקה בין סה״כ המעליות שנמצאות בבניין.

נרצה לחשב את כל אפשרויות האיסוף עבור כל מעלית שקיימת בבניין, וכך למצוא עבור כל קריאה את המעלית הטובה ביותר עבורה, כך שיתכנו שעבור מעלית x כלשהי יהיו מספר רב של קריאות.

עבור כל השיוכים ישנם ומכל אפשרות של שיבוץ קריאות למעלית נחשב את סה״כ הזמן המינימאלי בין כל האפשרויות של כל המעליות בדומה לנוסחה ממקרה און-ליין.

בעיה זו דומה לבעיית הסוכן הנוסע, אשר אין לה פתרון חישובי בזמן הגיוני.



**בדיקות ב Junit:**

נרצה לתכנן בדיקות עבור כל המקרים הבאים :

* המעליות לא חורגות מעבר לקומת מקסימום/מינימום
* מעלית אשר נמצאת במצב מנוחה מגיע לקריאה (src) שנמצא מעליה.
* מעלית אשר נמצאת במצב מנוחה מגיע לקריאה (src) שנמצא מתחתיה.
* המעלית מגיעה לקומת היעד(destination).
* בהינתן מספר קריאות, המעלית מטפלת בהם לפי הסדר הכרונולוגי שבו הקריאות בוצעו.

**דיאגרמת המחלקות העיקריות:**

**<<Interface>>ElevatorAlgo**

--------------------------------------------------

+getBuilding():Building

+algoName():String

+allocateAnElevator(CallForElevator c):int

+cmd(int):void

**<<Interface>>CallForElevator**

---------------------------------------------

+up:int=1

+down:int=-1

+INIT:int=0

+Going2Src:int=1

+Goint2Dest:int=2

+Done:int= 3

---------------------------------------------

+getState():int

+allocatedElevator(ind:int):void

+reachedSrc():void

+reachedDest():void

+getTime(state:int):double

+getSrc():int

+getDest():int

+getType():int

+allocatedTo():int

**<<Interface>>Building**

--------------------------------

+getBuildingName():String

+minFloor():int

+MaxFloor():int

+numberOfElevators():int

+getElevator(i:int):elevator

ד

* sdasd\*

**ElevatorAlgo**

---------------------------------------------------------------------------------------------

-building:Building

-go\_up:Queue<Integer>[]

-go\_down:Queue<Integer>[]

-elevDirection:Direction[]

Floor:int

First(true):Boolean

caseOf:int

generator(speed:4):Random

---------------------------------------------------------------------------------------------

+ElevatorAlgoClass(Building b)

+getBuilding():void

+algoName():String

-best(CallForElevator c):int

-arrivingTime(int src,int dest, Elevator e):double

-choose(CallForElevator c):int

+allocateAnElevator(CallForElevator c):int

+cmd(int elev):void

**<<Enumeration>>Direction**

----------------------------------------

Up

Down

None

**<<Interface>>Elevator**

---------------------------------

+up:int= 1

+down:int= -1

+Error:int= -2

+level:int=0

---------------------------------

+getMinFloor(): int

+getMaxFloor():int

+getTimeForOpen():double

+getTimeForClose():double

+getState():int

+getPos():int

+goTo(floor:int):Boolean

+stop(floor:int):Boolean

+geetSpeed():double

+getStartTime():double

+getStopTime():double

