**שפות תכנות מונחי עצמים**

**תרגיל תכנות מספר 3**

לצורך התרגיל Pert הוא טכניקה לחשב סדר משתמע של משימות הנובע מכך שמשמיות מסוימות הם קדם למשימות אחרות. למשל בהקמת בניין בניית הקירות חייבות להקדים את בניית הגג. דוגמא נוספת היא תוכנית לימודים שבה קורסים מסוימים הם קדם לקורסים אחרים. למשל צריך לעבור את "מל"ם" לפני שאפשר "מבנה נתונים".

במושגים של קומבינטוריקה Pert מבוסס על מבנה נתונים הנקרא "גרף מכוון".

טכניקת Pert מבוססת על כך שיש n משימות שיש לבצע על מנת להשלים את המשימה.

יחסי הקדימיות ייוצגו ע"י מערך של n רשימות מקושרות, נניח בשם arcArr.

הרשימות המקושרות יהיו מבוססות על צמתים של המחלקה הבאה:

class Arc

{

public int j;

public Arc next;

} // Arc

אם משימה 5 צריכה להתבצע לפני משימה 3, אזי ברשימה המתחילה ב-arcArr[5] יופיע במקום כלשהוא צומת שבו הערך j יהיה שווה ל-3.

המשימות אינן ממוינות לפי קדימויות.

המערך arcArr בעקיפין מגדיר מסלולים של קדימויות. לא ייתכן לולאה במסלולים הללו, לולאה כזו שקולה למצב, נניח בבניין, שהקירות צריכות להבנות לפני הגג והגג צריך להבנות לפני הקירות.

1. עליך לממש מחלקה בשם Pert המקיים את התנאים הבאים:
2. יכיל מספר n מספר המשמות.
3. יכיל מערך arcArr בגודל n.
4. יכיל בנאי המקבל את מספר המשימות n כפרמטר יחיד.
5. עליך לממש מתודה פומבית

void prerequisite(int i,int j)

המצהירה שמשימה i היא קדם למשימה j. המשמעות היא להוסיף צומת Arc המכילה את הערך j לרשימה של i. סדר של הצמתים ברשימה אינו חשוב. סיבוכיות ההכנסה צריכה להיות O(1).

1. עליך לממש מתודה

public void printStages();

המדפיסה את מספרי המשימות שיכולות להתבצע במקביל בכל שלב.

בשלב הראשון יודפסו כל המשימות שאין להם קדם כלל.

בשלב השני יודפסו כל המשימות שכל ההקדמים שלהם הודפסו בשלב הראשון וכן הלאה.

האתגר הוא לעשות זאת בסיבוכיות O(|V|+|E|) כלומר סיבוככות לינארית במספר המשימות ומספר הקדמים.

את השלבים ממש לפי האלגוריתם הבא:

נגדיר דרגת הקדימויות של משימה i כמספר הפעמים שיש קשת שנכנס אליו, קרי: כמה פעמים קיים x כזה שבו ברשימה של x יש קשת שה-j שלו שווה ל-i.

צור מערך זמניים inDegree המחשב את דרגות הקדימויות הישירות שכל המשימות.

צור מערכים בגודל n בשם stage\_list ו-next\_stage\_list.

בשלב האתחול עבור על inDegree וצבור לתוךstage\_list את כל המשימות שאין להם קדם כלל.

בכל שלב:

* עבור על המשימות i ב-stage\_list והפחת 1 מה-inDegree[j] של כל משימה j שמופיעה ברשימה arcArr[i]. במידה וההפחתה גורמת ל-inDegree[j] **להתאפס** הוסף את j למערך new\_stage\_list.
* לבסוף החלף (או העתק תוכן) בין stage\_list ו-new\_stage\_list.
* אין להקצות את המערכים stage\_list ו-new\_stage\_list בתוך הלולאה כי זה פוגם בסיבוכיות (C# מאפס ערכים של הקצאות דינמיות וזה לוקח סיבוכיות כגודל השטח).
* השלבים יסתיימו כאשר כל המשימות מודפסות.
* את ההדפסה עצמה בצע במתודה show\_assignment.

1. ממש מחלקה Courses כמחלקה **נגזרת** של Pert שעושה את הפעולות על המשימות לפי מחרוזות ולא לפי מספרים. המתודות הנוספות שהמחלקה הזו צריכה לתמוך הם הבאות:
2. בנאי המקבל פרמטר n מספר המשימות.
3. מתודה

void add\_course(String str)

המגדירה שם של משימה.

ההנחה היא שהמתודה הזו תקרא **בדיוק** n פעמים כל פעם עם מחרוזת ייחודית אחרת.

1. מתודה

void prerequisite(String str1, String str2)

המגדירה שהשם str1 הוא קרם של השם str2. ההנחה היא ש-str1, str2 הם בדיוק מחרוזות שעשו להן add\_course.

1. הפעלת printStages() מתוך courses ידפיס בכל שלב את המחרוזות במקום את מספרי הצמתים. **אסור** ש-Couses יממש את !printStages יש למצוא דרך שאפשר יהיה להסתמך על מימוש האלגוריתם של Pert וההתאמה ל-Courses יעשה ע"י דריסת מתודת ההדפסה בלבד.

לדוגמא, הפלט של התוכנית הבאה:

class RunPert

{

public static void Main(String[] args)

{

int n = 7;

Pert p = new Pert(n);

p.prerequisite(3,1);

p.prerequisite(3,6);

p.prerequisite(4,6);

p.prerequisite(4,5);

p.prerequisite(1,2);

p.prerequisite(5,0);

p.prerequisite(6,2);

p.printStages();

n= 12;

Courses c = new Courses(n);

c.addCourse("Malam 61510");

c.addCourse("Matam 61211");

c.addCourse("Atam 61210");

c.addCourse("Data Structures 61120");

c.addCourse("OOP 61310");

c.addCourse("Soft Eng 61101");

c.addCourse("OS 61122");

c.addCourse("OOP LANG 61313");

c.addCourse("Real Time 61133");

c.addCourse("Advanced Programming 61617");

c.addCourse("Parallel Computing 61104");

c.addCourse("Programming Languages 61110");

c.prerequisite("Malam 61510","Matam 61211");

c.prerequisite("Malam 61510","Atam 61210");

c.prerequisite("Matam 61211","Soft Eng 61101");

c.prerequisite("Atam 61210","Data Structures 61120");

c.prerequisite("Matam 61211","Data Structures 61120");

c.prerequisite("OS 61122","Parallel Computing 61104");

c.prerequisite("OS 61122","Real Time 61133");

c.prerequisite("Data Structures 61120",

"Programming Languages 61110");

c.prerequisite("Atam 61210","OS 61122");

c.prerequisite("Data Structures 61120","OS 61122");

c.prerequisite("Matam 61211","Programming Languages 61110");

c.prerequisite("Matam 61211","Advanced Programming 61617");

c.prerequisite("Matam 61211","OOP 61310");

c.prerequisite("OOP 61310","OOP LANG 61313");

c.prerequisite("OOP 61310","Soft Eng 61101");

c.prerequisite("Data Structures 61120",

"Advanced Programming 61617");

c.printStages();

} // main

} // RunPert

**יהיה:**

stage 1

3

4

stage 2

1

5

6

stage 3

0

2

stage 1

Malam 61510

stage 2

Atam 61210

Matam 61211

stage 3

OOP 61310

Data Structures 61120

stage 4

Soft Eng 61101

OOP LANG 61313

Advanced Programming 61617

OS 61122

Programming Languages 61110

stage 5

Real Time 61133

Parallel Computing 61104