# הגדרות בסיסיות

## מבנה נתונים

מבנה נתונים הוא דגם המגדיר את היחסים בין הנתונים לבין הפעולות המבוצעות עליהם. מבני הנתונים הבסיסיים הם: מערך, רשימה, מחסנית, תור, עצים, וגרפים. בכל פעם שנבוא לאחסן סוג נתונים נבחר את המבנה היעיל ביותר בהתאם לסוג השימוש בו אנו רוצים לאחסן נתונים.

לכל מבנה נתונים חייבים להגדיר שלוש פעולות בסיסיות (יש פעולות נוספות אך הן הבסיסיות):

1. הכנסת איבר.

2. מחיקת איבר.

3. חיפוש איבר.

## אלגוריתם

אלגוריתם הוא דרך שיטתית (כלומר כזו שצעדיה מוגדרים היטב) לביצועה של משימה מסוימת במספר סופי של צעדים. האלגוריתם מקבל ערך/ים כקלט ומפיק ערך/ים כפלט. מספר האיברים בקלט נקרא גם "גודל הקלט" או "אורך הקלט". דוגמה לאלגוריתם: מיון מערך. הקלט: מערך, הפלט: מערך הקלט ממוין.

קיימת חשיבות גדולה ליעילות של האלגוריתם. שאיפתנו היא לממש את האלגוריתם ביעילות הגבוהה ביותר, דהיינו במספר קטן ככל האפשר של צעדים. מדידת היעילות נקראת "סיבוכיות" (complexity), עליה נלמד בפרק הבא.

## פסאודו-קוד

כאשר אנו רוצים להשוות בין שני אלגוריתמים, אם נעשה זאת על ידי כתיבתם בשפת תכנות במחשב, נצטרך לעשות זאת על אותה שפה ועל אותה חומרה מכיוון שהחומרה והתוכנה משפיעות על מדדי הסיבוכיות של האלגוריתם. כדי להיפטר מהתלות של חומרה ותוכנה נכתוב את האלגוריתם בעזרת פסאודו-קוד, שהוא תיאור מופשט של תוכנית מחשב. הדרישה היחידה שהוא יהיה חד משמעי.

הערות לכתיבת פסאודו קוד:

* + לא מעמיסים בפרטים בפסאודו קוד. לדוגמה, לא כותבים טיפוס משתנים.
  + הזחה של הטקסט היא חובה.
  + נניח שהאינדקסים מתחילים מ-1, וקופצים ב-1.
  + AND כותבים בדרך כלל בטקסט, כיוון שבכל שפת תכנות הסימון משתנה.

# סיבוכיות

## הגדרה

כמו שאמרנו לעיל, קיימת חשיבות גדולה בבדיקה האם האלגוריתם ומבנה הנתונים שבחרנו הם הכי יעילים. לשם כך נצטרך לחשב "סיבוכיות של אלגוריתם". כדי למדוד סיבוכיות יש לחשב שני מדדים הנקראים "מדדי הסיבוכיות":

**זמן ריצה** - משך הזמן הנחוץ כדי לבצע את האלגוריתם.

**זיכרון** - גודל זיכרון המחשב שיש להקצות לביצוע האלגוריתם.

באופן מעשי, מכיוון שחישוב מדויק של מדדים אלו אינו אפשרי, שכן הוא משתנה בהתאם לשפת התכנות ולמערכת ההפעלה שאנו כותבים בהם את האלגוריתם, אנו נחשב סיבוכיות לפי סדר הגודל של מספר הצעדים הנדרשים לביצוע האלגוריתם. סיבוכיות בדרך כלל מסומנת באות גדולה O.

למשל אם נכתוב אלגוריתם למציאת האיבר הגדול ביותר במערך בעל n איברים, נוכל להשתמש באחת משתי השיטות הבאות:

1. להשוות כל מספר לכל המספרים האחרים וכך למצוא את הגדול מבניהם.

2. להניח שהמספר הראשון הוא הגדול ביותר, ולהשוותו לבאים אחריו. אם נמצא מספר גדול ממנו, יש לשנות את ההנחה, ולהשוות את כל הבאים אחריו למספר החדש.

באלגוריתם הראשון יש לבצע n\*n פעולות השוואה, ואילו לפי האלגוריתם השני יש לבצע סך הכל n פעולות. לפיכך האלגוריתם השני הוא היעיל יותר.

## כללים

סיבוכיות היא פונקציה של גודל הקלט באלגוריתם.

1. כל אלגוריתם שמבצע מספר קבוע של פעולות ללא קשר למספר האיברים שנקלטו בו, הסיבוכיות שלו היא 1, בסימון (1)O. לדוגמה: פעולת ההחלפה swap למרות שמתבצעות 3 החלפות, הסיבוכיות היא 1.

**O(constant number) = O(1)**

1. אלגוריתם שמספר האיברים שקלט הוא n ועל כל איבר מבצע פעולה אחת הסיבוכיות שלו היא O(n). גם אם האלגוריתם יבצע מספר פעולות **קבוע** על כל איבר, כיוון שסיבוכיות נמדדת בסדרי גודל המספרים הקבועים אינם משפיעים עליה, והסיבוכיות תישאר O(n). לדוגמה: בהדפסת כל האיברים עד n הסיבוכיות היא O(n), וגם בהדפסת כל האיברים עד 2n הסיבוכיות תישאר O(n).

**O(n\*constant number) = O(n)\*constant number = O(n)**

1. באלגוריתמים המורכבים מכמה שורות קוד, יש לסכום את כל הפעולות שמבצעת כל שורה כדי לקבל את מספר הפעולות שמבצע כל האלגוריתם. אמנם מכיוון שאנו מתעניינים בסיבוכיות של אלגוריתמים הקולטים מספר גדול של קלטים, תמיד נניח שמספר הקלטים שואף לאינסוף, לכן נשאיר את המחובר הדומיננטי ביותר ונשמיט את כל יתר המחוברים, לפי הכללים הבאים:

דוגמאות:

## סוגי סיבוכיות

עד כה שדיברנו על סיבוכיות, לקחנו את מספר הפעולות המקסימלי שהאלגוריתם יבצע לקלט הגרוע ביותר, אמנם זהו לא תמיד המצב כמובן. מספר הצעדים הדרוש לאלגוריתם כדי לבצע את משימתו תלוי בצורה של הקלט שנכנס לאלגוריתם. לדוגמה: באלגוריתם שעובר על כל האיברים במערך עד שמוצא איבר מסוים. המקרה הטוב ביותר הוא שהאיבר נמצא במקום הראשון, והמקרה הגרוע ביותר הוא שהאיבר כלל לא נמצא. לפיכך, הסיבוכיות משתנה בהתאם לקלט. אנו מבחינים בין שלושה סוגי סיבוכיות:

1. המקרה הגרוע (worst case) – המספר המקסימלי של פעולות לכל הקלטים האפשריים.
2. המקרה הטוב (best case) – המספר המינימלי של פעולות לכל הקלטים אפשריים.

3. המקרה הממוצע (average case) – ממוצע הפעולות הנדרשות לסך כל הקלטים האפשריים.

אותנו בדרך כלל מעניין המקרה הגרוע ביותר, ולכן לרוב נחשב את הסיבוכיות לפיו.

## חסמים אסימפטוטיים

כנגד כל אחד משלושת סוגי הסיבוכיות יש חסם אסימפטוטי המתאר את ההתנהגות של פונקציית הסיבוכיות עבור ערכים הולכים וגדלים (שואפים לאינסוף), וזאת באמצעות השוואה לפונקציה אחרת g המהווה לה חסם. היתרון שבשימוש בחסם אסימפטוטי הוא שהוא מאפשר לקבל הערכה טובה על אופן הגידול של ערכי הפונקציה מבלי שיהיה צורך לדעת אותו במדויק.

חסם אסימפטוטי, לדוגמא , הוא אוסף כל הפונקציות (בעלות אותו תחום וטווח כמו g(n)), שמקיימות תנאים מסוימים. במידה ופונקציה f(n) אכן מקיימת תנאים אלו, נאמר כי , אמנם נהוג לכתוב .

### חסם אסימפטוטי עליון -

עבור שתי פונקציות f(n), g(n) נאמר ש-, אם קיימים קבועים חיוביים ו-c כך שלכל מתקיים: . כלומר, ניתן למצוא מכפלה כלשהי של g(n)שהחל מערך מסוים של n תמיד תהיה גדולה יותר מ-f.

במילים אחרות, ככל שערכי n הולכים וגדלים f גדל פחות מהקצב של g, היינו g חוסמת את f מלמעלה. משמעות נוספת היא שמתקיים: .

כאשר אנחנו אומרים כי זמן הריצה של אלגוריתם מסוים הוא , הכוונה היא שזמן הריצה במקרה הגרוע ביותר לקלט הגרוע ביותר - חסום מלמעלה על ידי הפונקציה .

דוגמא: . האם ?

כדי להוכיח את הטענה נצטרך למצוא שני קבועים שיקיימו את אי השוויון *. נבחר . האם קיים*  שהחל ממנו ? כן נבחר .

### חסם אסימפטוטי תחתון -

עבור שתי פונקציות f(n), g(n) נאמר ש-, אם קיימים קבועים חיוביים ו-c כך שלכל מתקיים: . כלומר, ניתן למצוא מכפלה כלשהי של g שהחל מערך מסוים של n תמיד תהיה קטנה יותר מ-f.

במילים אחרות, ככל שערכי n הולכים וגדלים f גדל יותר מהקצב של g, היינו g חוסמת את f מלמטה. משמעות נוספת היא שמתקיים: .

לעיתים נרצה לדבר על חסם תחתון לאלגוריתם, כדי לדעת מהו מינימום הפעולות שיצטרך לבצע לקלט הטוב ביותר. נשתמש בחסם כדי לתאר זאת. לדוגמא, ברור שמיון מערך של n מספרים דורש לפחות n פעולות, שהרי רק בדיקה אם המספרים ממוינים עולה סדר גודל של n פעולות. לכן זמן הריצה הוא *, כלומר שזמן הריצה הטוב ביותר למיון מערך -* חסום מלמטה על ידי הפונקציה .

דוגמא: . האם ?

כדי להוכיח את הטענה נצטרך למצוא שני קבועים שיקיימו את אי השוויון *. נבחר , במצב זה אי השוויון נכון לכל* n.

### חסם אסימפטוטי הדוק -

עבור שתי פונקציות f(n), g(n) נאמר ש-, אם קיימים קבועים חיוביים כך שלכל מתקיים: . המשמעות היא שיש מכפלה כלשהי של g שהחל מערך מסוים של n תמיד תהיה קטנה יותר מ-f, וכן יש מכפלה נוספת של g שהחל מאותו ערך n תמיד תהיה גדולה יותר מ-f. נשים לב כי מתקיים: וגם .

במילים אחרות, ככל שערכי n הולכים וגדלים f גדל באותו הקצב של g, היינו g ו-f מאותו סדר גודל. משמעות נוספת היא שמתקיים: .

### חסם אסימפטוטי - o

עבור שתי פונקציות f(n), g(n) נאמר ש-, אם **לכל קבוע חיובי** קיים קבוע חיובי כך שלכל מתקיים: . כלומר, לכל מכפלה חיובית של g, החל מערך מסוים של n, g תמיד תהיה גדולה יותר מ-f.

במילים אחרות, ככל שערכי n הולכים וגדלים f גדל בקצב שקטן ממש מהקצב של g, היינו g שולט אסיפטוטית על f. משמעות נוספת היא שמתקיים: .

### חסם אסימפטוטי -

עבור שתי פונקציות f(n), g(n) נאמר ש-, אם **לכל קבוע חיובי** קיים קבוע חיובי כך שלכל מתקיים: . כלומר, לכל מכפלה חיובית של g, החל מערך מסוים של n, g תמיד תהיה קטנה יותר מ-f.

במילים אחרות, ככל שערכי n הולכים וגדלים f גדל בקצב שגדול ממש מהקצב של g, היינו g זניח אסיפטוטית ביחס ל-f. משמעות נוספת היא שמתקיים: .

## סיכום גבולות

## תכונות

טרנזיטיביות:

רפלקסיביות:

סימטריה:

השלמה:

# רקורסיה

## הגדרה

פונקציה רקורסיבית היא פונקציה אשר על מנת לפתור בעיה מסוימת קוראת לעצמה (או לפונקציה שקוראת לה, במקרה של "רקורסיה הדדית"). השימוש ברקורסיה בעצם מאפשר לנו במקום להתמודד עם הבעיה הגדולה, לחלק את הבעיה לתתי בעיות שעליהם נפעיל שוב ושוב את הפונקציה הרקורסיבית. כל פונקציה רקורסיבית חייבת לקבל פרמטר מסוים, הנקרא "בסיס הרקורסיה", שבו הפונקציה לא קוראת לעצמה אלא מחזירה ערך, שאם לא כן תהיה זו לולאה אינסופית רקורסיבית.

בזמן הריצה, בכל קריאה לפונקציה רקורסיבית נשמרת כתובת החזרה ואוסף המשתנים המקומיים של הפונקציה במחסנית קריאות. בסוף ביצוע כל פונקציה, הערכים האלה נשלפים מהמחסנית כך שבסיום הרקורסיה, כל הזיכרון שנדרש לחישובה, מתפנה. מאחר שכל קריאה רקורסיבית נשמרת במחסנית המחשב, בזמן החישוב, דורשת הרקורסיה זיכרון בגודל פרופורציונלי לעומקה.

דוגמאות לפונקציות רקורסיביות נמצאות בקובץ הרצאה מספר 2.