בס״ד טבת התשע״ב - 12/2011

מבוא לאלגוריתמים גמישים

אליעזר גנסבורגר, עזרא דשט, ר.ב. יחזקאל

תוכן עניינים

פרק ראשון - מבוא
מטרות הקורס
אלגוריתם סדרתי ואלגוריתם גמיש
דוגמא של קניות בצרכניה לפי רשימה באמצעות אלגוריתם גמיש
דוגמא נוספת של בדיקת עבודות בע״פ באמצעות אלגוריתם גמיש

פרק שני - אלגוריתמים גמישים בסגנון של פונקציות ושיטות ביצוע בסגנון של קבוצות
אלגוריתם גמיש rev1 , rev3 , rev5
שיטות ביצוע בסגנון של קבוצות
אלגוריתם גמיש reverse
סגנונות שונים של כתיבת פרמטרים
אלגוריתם גמיש swap
אלגוריתם גמיש gcd

פרק שלישי - המרת אלגוריתם גמיש לתוכנית סדרתית

פרק רביעי - אלגוריתמים גמישים שונים
אלגוריתם גמיש לחיבור 8 מספרים
אלגוריתם גמיש לחיבור מערך של n מספרים כאשר n חזקה של 2
אלגוריתם גמיש להזזה מעגלית
אלגוריתם גמיש למיזוג (merge)
אלגוריתם גמיש למיון מערך לפי שיטת ״מיון זוגי אי-זוגי״ (Even-Odd Sort)
אלגוריתם גמיש למיון ע״י מיזוג (Merge Sort)
אלגוריתם גמיש לחיפוש בינארי (Binary Search)

פרק חמישי - פיתוח דיאגרמות בלוקים של חומרה
חיפוך ווקטור בגודל קבוע
חיבור של שני מספרים עם נשא (ספרה ספרה או ביט-ביט)

פרק שישי - הכרה ראשונית של המושג מערכת והקשר עם אלגוריתמים גמישים

פרק שביעי - נכונות של אלגוריתמים
חזרה על אינדוקציה
אינדוקציה חישובית

חומר נוסף מתקדם באנגלית

פרק ראשון - מבוא

מטרות הקורס

1) להקנות ידע על הליכי ביצוע שונים של אלגוריתמים (סדרתי, מקבילי, קריאות מידיות/דחויות)
2) הצגת אלגוריתם כשלב ראשוני בפיתוח תוכניות.
3) הצגת שיטה מתמטית לניתוח אלגוריתם / תוכניות.
4) הכרה ראשונית של המושג "מערכת".

אלגוריתם סדרתי ואלגוריתם גמיש

מקובל להגדיר אלגוריתם **כתהליך** סידרתי המתבצע צעד אחר צעד. האלגוריתם הסדרתי איננו מהווה גולת הכותרת של הקורס שלנו ואנחנו נתמקד בעיקר בנושא של "אלגוריתמים גמישים". "אלגוריתם גמיש" הוא תהליך שתוצאותיו הסופיות מוגדרות באופן חד משמעי, אבל מניחים שייתכנו כמה תהליכי ביצוע עד לקבלת התוצאות הסופיות.

דוגמא של קניות בצרכניה לפי רשימה באמצעות אלגוריתם גמיש

א) הקונה מגיע עם רשימת קניות לצרכניה.
ב) הקונה נכנס לצרכניה לוקח עגלה ומכניס את המוצרים לעיל לעגלה. הקונה יכול לגשת למוצרים לפי סדר הופעתם ברשימת הקניות או לפי סדר מקומם של המוצרים על גבי המדפים, או לפי סדר אחר. (כאן יש גמישות בביצוע המשימה.)
ג) הקונה מגיע לקופה עם העגלה ומשלם עבור המוצרים ויוצא מהצרכניה.

דוגמא נוספת של בדיקת עבודות בע"פ באמצעות אלגוריתם גמיש

א) המרצה פותח את דלת משרדו על מנת שתלמידים יוכלו להבחן.
ב) התלמידים מתייצבים ע"י דלת המשרד כשבידיהם העבודה שהם צריכים להבחן עליה.
ג) התלמידים נכנסים ונבחנים.
ד) המרצה\המרצים בוחנים.
ה) לאחר המבחן הציון הסופי מעודכן.
ו) העבודה שניבדקה מצורפת לערמת העבודות להחזרה.

מדוע ניתן לזהות את ביצועו של התהליך הנ"ל כ-"אלגוריתם גמיש" ?

- שלב ו' מציין את התוצאה הסופית של תהליך הבדיקות שבו העבודות שנבדקו נערמות יחד בערמת העבודות להחזרה כלומר <u>יש דרישה לתוצאה סופית שמוגדרת באופן חד-משמעי</u> **!**
- את כל אחד מן השלבים ב' - ה' ניתן לבצע בצורה שונה כדלהלן:
- בשלב ב' ניתן לקבוע שהתלמידים מתייצבים בדלת המשרד בצורה נפרדת כל תלמיד לחוד או לחילופין כל פעם מגיעים X תלמידים ביחד.
- בצורה שווה ניתן גם לממש את שלב ג'. התלמידים נכנסים למשרד כל אחד לחוד או X תלמידים ביחד.
- בשלב ד' ניתן לבצע את הבדיקה של אותה עבודה ע"י כמה בודקים כך שכל בודק יבדוק שאלה מסוימת או כמה עבודות ע"י כמה בודקים כאשר כל בודק יבדוק עבודה שלימה. או שמרצה אחד עובר בצורה סדרתית על כל עבודה אחת אחר השניה.
- את שלב העדכון ניתן לבצע בכמה אופנים. לדוגמא ניתן לעדכן כל ציון סופי של עבודה באופן ניפרד בתוך גיליון הציונים. או ניתן לעדכן כל ציון של שאלה בנפרד ע"י המרצה האחרון שבדק בתוך גיליון הציונים. אפשרות אחרת שכל מרצה יעדכן את הציון שלו מיד אחר הבדיקה בגיליון הציונים.

בסיכומו של דבר עד לשלב הסופי ניתן לבצע את כל אחת מן המשימות בצורה שונה - מקבילית או סדרתית.

לדיון בכיתה

5 באיזה סדרים ניתן לחשב את הביטוי (((1-2)/(3-5)) / ((5-6)/(7-9))) ?

פרק שני - אלגוריתמים גמישים בסגנון של פונקציות ושיטות ביצוע בסגנון של קבוצות

אלגוריתם גמיש rev1 , rev3 , rev5

```
function a', b', c', d', e' = rev5(a, b, c, d, e);
/*
SPECIFICATION:
IN - values a, b, c, d, e
OUT - values a', b', c', d', e' like a, b, c, d, e but in reverse order
*/
{ a'=e;
  b', c', d'=rev3(b, c, d);
  e'=a;
} /* end rev5 */

function a', b', c' = rev3(a, b, c);
/*
SPECIFICATION:
IN - values a, b, c
OUT - values a', b', c' like a, b, c but in reverse order
*/
{ a'=c;
  b' =rev1(b);
  c'=a;
} /* end rev3 */

function a' = rev1(a);
/*
SPECIFICATION:
IN - value a
OUT - value a' like a but in reverse order - that is a' is a
*/
{ a'=a;
} /* end rev1 */
```

שיטות ביצוע בסגנון של קבוצות

נוכל לממש את הביצוע (או החישוב) באמצעות אחת משלושת השיטות הבאות:

א) ביצוע מקבילי.

ב) ביצוע סידרתי כאשר הקריאה \ ההפעלה של פונקציה מתבצעת **מיד** ללא דיחוי.

ג) ביצוע סידרתי כאשר הקריאה \ ההפעלה של פונקציה **נדחית**.

שלוש שיטות הביצוע יוצגו בצורה הבאה:

set of statements values of results.

הצגת שלש שיטות ביצוע בסגנון של קבוצות

נניח שאנחנו רוצים לבצע (או לחשב):

a', b', c', d', e' = rev5 (1, 2, 3, 4, 5);

Parallel execution

set of statements	a', b', c', d', e'
{ a', b', c', d', e' = rev5(1, 2, 3, 4, 5) }	_, _, _, _, _
{ a' = 5 ; b', c', d' = rev3(2, 3, 4) ; e' = 1 }	_, _, _, _, _
{ b' = 4, c' = rev1(3) ; d' = 2 }	5, _, _, _, 1
{ c' = 3 }	5, 4, _, 2, 1
{ }	5, 4, 3, 2, 1

Sequential execution left to right with <u>immediate</u> execution of function call at left

set of statements	a', b', c', d', e'
{ a', b', c', d', e' = rev5(1, 2, 3, 4, 5) }	_, _, _, _, _
{ a' = 5 ; b', c', d' = rev3(2, 3, 4) ; e' = 1 }	_, _, _, _, _
{ b', c', d' = rev3(2, 3, 4) ; e' = 1 }	5, _, _, _, _
{ b' = 4, c' = rev1(3) ; d' = 2 ; e' = 1 }	5, _, _, _, _
{ c' = rev1(3) ; d' = 2 ; e' = 1 }	5, 4, _, _, _
{ c' = 3 ; d' = 2 ; e' = 1 }	5, 4, _, _, _
{ d' = 2 ; e' = 1 }	5, 4, 3, _, _
{ e' = 1 }	5, 4, 3, 2, _
{ }	5, 4, 3, 2, 1

Sequential execution left to right with <u>delayed</u> execution of function call at left

set of statements	a', b', c', d', e'
{ a', b', c', d', e' = rev5(1, 2, 3, 4, 5) }	_, _, _, _, _
{ a' = 5 ; b', c', d' = rev3(2, 3, 4) ; e' = 1 }	_, _, _, _, _
{ b', c', d' = rev3(2, 3, 4); e' = 1 }	5, _, _, _, _
{ b', c', d' = rev3(2, 3, 4) }	5, _, _, _, 1
{ b' = 4, c' = rev1(3) ; d' = 2 }	5, _, _, _, 1
{ c' = rev1(3) ; d' = 2 }	5, 4, _, _, 1
{ c' = rev1(3) }	5, 4, _, 2, 1
{ c' = 3 }	5, 4, _, 2, 1
{ }	5, 4, 3, 2, 1

תרגילים

1) כתוב פונקציות rev2, rev4 דומות לפונקציות הקודמות, כולל מפרטים.
הצג את הביצוע של a', b', c', d' = rev4 (2, 3, 4, 5); לפי שלש השיטות לעיל.

2) נניח שנשנה את הגדרת הפונקציות **rev1, rev3, rev5** כדלהלן:

```
function a', b', c', d', e' = new_rev5(a, b, c, d, e);
{
  b', c', d'= new_rev3(b, c, d);
  a'=e;    // שים לב שעכשיו שתי ההשמות מופיעות יחד
  e'=a;
}

function a', b', c' = new_rev3(a, b, c);
{
  b' = new_rev1(b);
  a'=c;    // שים לב שעכשיו שתי ההשמות מופיעות יחד
  c'=a;
}

function a' = new_rev1(a);
{
  a'=a;
}
```

הצג את הביצוע של a', b', c', d', e' = new_rev5(1, 2, 3, 4, 5); לפי שלש השיטות לעיל.

אלגוריתם גמיש reverse

```
function v' = reverse(v, low, high);
/*
```

מפרט:

הפונקציה **reverse** מקבלת ווקטור v ושני ערכים low ו- high שמציינים מיקומים בתוך הווקטור.
אם low≤high,
הפונקציה **reverse** מחזירה בווקטור " 'v " בתחום שבין low ו - high,
אותם איברים של ווקטור " v ", רק שהם יופיעו בסדר הפוך.
שאר איברי " 'v " לא מקבלים ערכים ע"י בצוע פונקציה זאת.
אם low>high, הפונקציה לא תשים דבר ב- " 'v ".

SPECIFICATION:

IN - "v" is a vector and "low", "high" are positions within the vector v.
OUT - If low≤high, then within the range "low" to "high", v' is like v but reversed.
Other elements of v' are not given values by this function.
If low>high, then the function does nothing to v'.

```
*/
{
if (low<high)
  {v'high = vlow;
   v' = reverse (v, low+1, high-1);
   v'low = vhigh;}
else if (low==high)
      {v'high = v high;};
} /* end reverse */
```

הערות

1) המיקום הראשון בווקטור הינו מיקום 0.
2) השמה תסומן ע"י " = " , ואילו שוויון יסומן ע"י " = = ".
3) הסימון " IN " פירושו הפרמטרים שהפונקציה **מקבלת**, ואילו הסימון " OUT " פירושו התוצאות שהפונקציה **נותנת**.
4) פרמטרים שמופיעים עם גרש לימינם מסווגים כ - " OUT ".
5) אסור לשנות ערך של משתנה כלשהו אבל מותר ליצור משתנה **חדש** (עם אותו שם) ולשים ערך חדש בתוכו. תכונה זאת הנה הכרחית על מנת לאפשר ביצוע גמיש של אלגוריתם (פונקציה).

סגנונות שונים של כתיבת פרמטרים

אם נשים לב הרי בצענו קריאה \ הפעלה לפונקציה reverse מתוך הפונקציה עצמה. ניתן לרשום את הקריאה בכמה סגנונות שקולים כדלהלן:

א) סגנון של פונקציה: *v' = reverse (v,low+1,high-1)*

ב) סגנון של פרוצדורה: reverse(v,low+1,high-1,v')

ג) סגנון של השמות: reverse(v=v,low=low+1,high=high-1,v'=v')

ד) כמו ג' אבל רשום רק את השינויים: *reverse(low=low+1,high=high-1)*

אנחנו נשתמש בשיטות א' ו- ד' בהמשך.

הערה

בהתאם להנחה הקודמת אזי לדוגמא בצוע ההשמות הבאות שמופיעות בפונקציה הנ"ל:

high=high-1 , low=low+1 **אינו משנה** את הערך של- low ושל- high הקיימים, אלא שהמשתנים המופיעים בצד ימין של פעולת ההשמה הם המשתנים ***הקיימים*** והמשתנים המופיעים משמאל לפעולת ההשמה הם משתנים ***חדשים*** שנוצרים כל פעם מחדש עבור כל ביצוע חוזר של הפונקציה reverse.

הצגת שלש שיטות ביצוע בסגנון של קבוצות

נניח ש - v=(1,2,3,4,5) ואנחנו רוצים לבצע (או לחשב):

r' = reverse (v,0,4);

Parallel execution

set of statements	r'
{ r' = reverse (v, 0, 4); }	(_, _, _, _, _)
{ $r'_4 = v_0$; r' = reverse (v, 1, 3); $r'_0 = v_4$; }	(_, _, _, _, _)
{ $r'_3 = v_1$; r' = reverse (v, 2, 2); $r'_1 = v_3$; }	(5, _, _, _, 1)
{ $r'_2 = v_2$; }	(5, 4, _, 2, 1)
{ }	(5, 4, 3, 2, 1)

Sequential execution left to right with <u>immediate</u> execution of function call at left

set of statements	r'
{ r' = reverse (v, 0, 4); }	(_, _, _, _, _)
{ $r'_4 = v_0$; r' = reverse (v, 1, 3); $r'_0 = v_4$; }	(_, _, _, _, _)
{ r' = reverse (v, 1, 3); $r'_0 = v_4$; }	(_, _, _, _, 1)
{ $r'_3 = v_1$; r' = reverse (v, 2, 2); $r'_1 = v_3$; $r'_0 = v_4$; }	(_, _, _, _, 1)
{ r' = reverse (v, 2, 2); $r'_1 = v_3$; $r'_0 = v_4$; }	(_, _, _, 2, 1)
{ $r'_2 = v_2$; $r'_1 = v_3$; $r'_0 = v_4$; }	(_, _, _, 2, 1)
{ $r'_1 = v_3$; $r'_0 = v_4$; }	(_, _, 3, 2, 1)
{ $r'_0 = v_4$; }	(_, 4, 3, 2, 1)
{ }	(5, 4, 3, 2, 1)

Sequential execution left to right with <u>delayed</u> execution of function call at left

set of statements	r'
{ r' = reverse (v, 0, 4); }	(_, _, _, _, _)
{ $r'_4 = v_0$; r' = reverse (v, 1, 3); $r'_0 = v_4$; }	(_, _, _, _, _)
{ r' = reverse (v, 1, 3); $r'_0 = v_4$; }	(_, _, _, _, 1)
{ r' = reverse (v, 1, 3); }	(5, _, _, _, 1)
{ $r'_3 = v_1$; r' = reverse (v, 2, 2); $r'_1 = v_3$; }	(5, _, _, _, 1)
{ r' = reverse (v, 2, 2); $r'_1 = v_3$; }	(5, _, _, 2, 1)
{ r' = reverse (v, 2, 2); }	(5, 4, _, 2, 1)
{ $r'_2 = v_2$;}	(5, 4, _, 2, 1)
{ }	(5, 4, 3, 2, 1)

תרגילים

1) נניח ש - v=(1,2,3,4,5,6,7) ורוצים לבצע: { v' = reverse (v,0,3); v' = reverse (v,4,6); } .
הצג את כל אחת משלושת אפשרויות הבצוע הנ"ל עבור ההגדרה הקודמת של reverse.

2) נניח שנשנה את הגדרת הפונקציה **reverse** כדלהלן:

```
function v' = reverse(v, low, high);
{
if (low<high)
 {  v'low = vhigh;
    v' = reverse (v, low+1, high-1);
    v'high = vlow;}
else if (low==high)
        {v'high = v high;};
} /* end reverse */
```

הצג את כל אחת משלושת אפשרויות הבצוע הנ"ל עבור ההגדרה החדשה של reverse כאשר v=(1,2,3,4,5) והקריאה הראשונה הינה r'=reverse(v,0,4); .
השווה את הביצועים שרשמת עם הביצועים של rev5(1,2,3,4,5) שרשום בדפים קודמים.

3) נניח שנשנה את הגדרת הפונקציה **reverse** כדלהלן:

```
function v' = reverse(v, low, high);
{
if (low<high)
 {
 v' = reverse (v, low+1, high-1);
    v'high = vlow; v'low = vhigh;  // שים לב שעכשיו שתי ההשמות מופיעות יחד
}
else if (low==high)
        {v'high = v high;};
} /* end reverse */
```

הצג את כל אחת משלושת אפשרויות הבצוע הנ"ל עבור ההגדרה החדשה של reverse כאשר v=(1,2,3,4) והקריאה הראשונה הינה r'=reverse(v,0,3); .

4) כתוב פונקציה שמקבלת מחרוזת או מילה כווקטור של תווים. הפונקציה בודקת האם המחרוזת מהווה פלינדרום, ומחזירה תוצאה אמת או שקר בהתאם. (פלינדרום הנו מילה או מחרוזת שנקראת ישר והפוך בסדר זהה.) הפונקציה צריכה לכלול מפרט (specifcation).

אלגוריתם גמיש swap

```
function a', b' = swap (a, b);
/*
SPECIFICATION:
IN - "a","b" are integers.
OUT - a' will get the value of b and b' will get the value of a.
*/
{
   a' = b;
   b' = a;
} /* end swap */
```

הצגת שלש שיטות ביצוע בסגנון של קבוצות

נניח ש- a=1, b=2 ואנחנו רוצים לבצע:

```
a',b' = swap (a,b);
```

Parallel execution

set of statements	a', b'
{ a' ,b' = swap (a,b); }	_, _
{ a' = b; b' = a; }	_, _
{ }	2, 1

Sequential execution left to right with <u>immediate</u> execution of function call at left

set of statements	a', b'
{ a' ,b' = swap (a,b); }	_, _
{ a' = b; b' = a; }	_, _
{ b' = a; }	2, _
{ }	2, 1

Sequential execution left to right with <u>delayed</u> execution of function call at left

set of statements	a', b'
{ a' ,b' = swap (a,b); }	_, _
{ a' = b; b' = a; }	_, _
{ b' = a; }	2, _
{ }	2, 1

אלגוריתם גמיש gcd

להלן אלגוריתם גמיש לחשב את המחלק המשותף המירבי (Greatest Common Divisor - gcd)
עבור שלמים חיוביים m , n כך ש- m, n > 0.
אוקלידס המציא אלגוריתם זה.

```
function r' = gcd(m,n)
/*
        מפרט: m ,n הם כנ"ל ו- r' יקבל את השלם החיובי הגדול ביותר שמחלק גם את m ואת n.
*/
{ if (m=n)
    {r'=m};
  if (m>n)
    {gcd(m=m-n);};
  if (n>m)
    {gcd(n=n-m);};
}
```

תרגילים

1. הצג את הביצוע של r'=gcd(6,14); לפי השיטה מקבילית.
2. ניתן לשפר יעילות הנ"ל ע"י שימוש בפעולות חילוק ושארית, כלומר "%".
   כתוב אותו מחדש כפונקציה אחת בעזרת פעולות אלו.
3. ניתן לשפר עוד את היעילות ע"י שימוש בפונקציה נוספת.
   כתוב אותה מחדש ע"י שתי פונקציות.

פרק שלישי - המרת אלגוריתם גמיש לתוכנית סדרתית

תוכנית סדרתית הרבה פעמים מקבלת ערך במשתנה מסוים ועל מנת לחסוך מקום היא מחזירה את הערך באותו משתנה קלט שהיא קבלה.

דוגמא: הפונקציה reverse שלמדנו לעיל מחזירה תוצאות במשתנה 'v. ניתן להמיר את האלגוריתם הגמיש reverse לתוכנית סדרתית כשהתוצאה הסופית תינתן באותו משתנה v. יש שלשה שלבים להמרה.

התבונן שוב באלגוריתם reverse הגמיש. להלן שלבי ההמרה.

<u>שלב 1</u>

1) מחוק את הכותרת "function reverse(...)" ובמקום זה רשום "reverse:".
2) מחוק כל גרש.
3) במקום הפעלה חוזרת עשה שימוש בפקודה goto.

אחר ביצוע הנ"ל האלגוריתם הסדרתי ייראה:

```
reverse: {if low<high
           {
                                        /*  לקמן יש בעיות עם סדר הכתיבה.  */
                                        /*  חסר משתנה עזר לצורך החלפה. */
            v[high]=v[low];
            low=low+1; high=high-1;
            goto reverse;
            v[low]=v[high]
           }

                                        /* הקטע לקמן מיותר */
          else
           if low==high
              v[high]=v[high]
        }
```

<u>שלב 2</u>

1) בדוק את סדר הכתיבה אם היא מתאימה לתוכנית סדרתית.
2) השתמש במשתנה עזר לפי הצורך.
3) מחוק חלקים מיותרים.

אחר ביצוע הנ"ל האלגוריתם הסדרתי ייראה:

```
reverse: { if low<high
           { temp1=v[low];temp2=v[high];
             v[low]=temp2; v[high]=temp1;
             low=low+1; high=high-1;
             goto reverse;
           }
         }
```

שלב 3

ניתן לעשות שימוש בלולאת while במקום בפקודת goto.

```
while low<high
{
  temp1=v[low]; temp2=v[high];
  v[low]=temp2; v[high]=temp1;
  low=low+1;  high=high-1;
}
```

הערה

ניתן לבצע את הנ"ל באופן סדרתי פעולה אחר פעולה. אבל גם אפשר לבצע באותה שורה פעולות במקביל.

לדיון בכיתה

בפתרון קודם בעזרת לולאת while, השתמשנו בשני משתנה עזר. התבונן בפתרון נוסף להלן שבו השתמשנו במשתנה עזר אחד בלבד.

```
while low<high
{
  temp=v[low];
  v[low]= v[high];
  v[high]=temp;
  low=low+1;  high=high-1;
}
```

מה היתרונות וחסרונות של הפתרון לעיל?

תרגילים

1) לפניך אלגוריתם גמיש שמחשב את סכום השלמים מאפס עד n.

```
function s'=sigma(n);
{ loop(i=0;s=0;);}
function s'=loop(n,i,s);
{
 if i>n
    {s'=s}
  else
    {loop(i=i+1;s=s+i);}
}
```

המר את האלגוריתם הגמיש לעיל לתוכנית סדרתית לפי השלבים שהסברנו לעיל.

2) לפניך אלגוריתם גמיש שמחשב את החזקה הרבועית עבור כל אחד מסדרת מספרים נתונה.

```
function v' = squares (v);
/*
SPECIFICATION: v, v' are vectors having the same length. Each element of v' is the
square of the corresponding element of v.
*/
{ loop(i=0); }; /* squares */

function v'=loop(v, i);
/*
SPECIFICATION: Like squares, but puts values only at places i onwards.
*/
{ if (i< length of v)
    { loop(i=i+1); v'i=vi**2; };
}; /* loop */
```

א. המר את האלגוריתם הגמיש לעיל לתוכנית סדרתית לפי השלבים שהסברנו לעיל כשהתוצאה הסופית תינתן באותו משתנה v.

ב. נניח שחלוקת העבודה מתבצעת ע"י "מנהל" וחישוב הרבועים מתבצע ע"י ארבעה "פועלים". המנהל מפעיל את כל שלב ונותן הוראה לפועל כל 10 שניות והפועל מחשב את הרבוע בדקה. רשום את התרשים של הפתוח המקבילי והוסף לו את זמני הבצוע. בסוף סכם את הזמן הכולל של הבצוע עבור הדוגמא: v=(1,-2,3,-4). ציין זמנים של התחלה וזמנים של סיום כל שלב בפתוח הנ"ל.

3) <u>כתוב אלגוריתם גמיש</u> שמחשב את סכום השלמים מאחד עד n אבל עם סימן מתחלף, כלומר 1-2+3-4...

5) לפניך האלגוריתם copy שמעתיק מערך או חלק ממנו למערך תוצאה.

```
function v'=copy(v,low,high,index);
 /*
 SPECIFICATION
IN - "v" is a vector and "low" and "high" are positions within the vector v where we
assume low ≤ high.
"index" is a position within the vector v' indicates the beginning of the copied block.
OUT - From the position "index", v' is a copy of v in the range "low" to "high". Other
elements of v' are not given values by the function.
*/
{
  v'index=vlow;
   if(low < high)
     {
       copy(low=low+1, index=index+1);
     }
} /*End copy*/
```

שנה את הפונקציה כך שתעבוד גם אם high<low. השתמש בפונקצית עזר אם אכן יש צורך בכך.

נגדיר אלגוריתם גמיש שמקבל מערך של מספרים ומסכם אותם

```
function s' = sum(v);
/* SPECIFICATION: מפרט
     Input:   v is a vector of numbers.
     Output:  s' is the sum of the elements of the vector v.
              s'= v0 + v1 +... + v (length of v-1)
*/
{ sum1(i=0;s=0); }

function s' = sum1(v, i, s);
/* SPECIFICATION:  s'= s + vi + vi+1 ... + v (length of v - 1)  : מפרט */
/* NOTE that when i ≥ length of v, this means that  s'= s.  */
{
  if (i<length of v)
     {sum1(i=i+1;s=s+vi)};
  else
     { s'=s; }
}
```

המרת האלגוריתם הגמיש הנ"ל לתוכנית סדרתית כאשר התוצאה תהיה ב- s ולא ב-s'

הצעה לפתרון ראשון

```
i=0; s=0;
while i<length of v
{
  new_i=i+1; new_s=s+v[i]
  i=new_i; s=new_s;
}
```

הצעה לפתרון שני

```
i=0;s=0;
while i<length of v
{
  s=s+v[i];
  i=i+1;
}
```

פרק רביעי - אלגוריתמים גמישים שונים

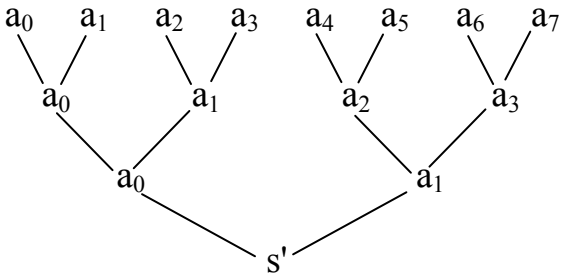
אלגוריתם גמיש לחיבור 8 מספרים

כאשר כותבים $s' = a_0 + a_1 + \dots a_7$

כאילו כתוב $s' = (((\dots(a_0 + a_1) + a_2) + a_3) + \dots a_7)$, ובשיטת החישוב הזה לא ניתן לבצע פעולות במקביל.

בשיטת החישוב הבאה ניתן לבצע פעולות במקביל.

$$s' = (((a_0 + a_1) + (a_2 + a_3)) + ((a_4 + a_5) + (a_6 + a_7)))$$



צורת חישוב עבור אלגוריתם גמיש - פונקציות.

```
function s' = add8 (a0, a1 , a2, a3, a4, a5, a6, a7)
{ add4 (a0=a0+a1; a1=a2+a3; a2=a4+a5; a3=a6+a7);}

function s' = add4(a0,a1,a2,a3);
{add2(a0=a0+a1; a1=a2+a3);}

function s' = add2(a0, a1);
{s'=a0+a1};
```

אלגוריתם גמיש לחיבור מערך של n מספרים כאשר n חזקה של 2

```
function s'=addn(v,n)
/* SPECIFICATION: s' = v0 + v1 ... + vn-1    :מפרט          */
/* n is the size of vector v which must be a power of 2.     */
{
if (n==1)
      {s' = v0}
else addn (    n=n/2;
               v0=v0+v1;
               v1=v2+v3
               .
               .
               .
               vn/2-1 = vn-2 + vn-1   );
}
```

דוגמא של בצוע

set of statements	**s'**
{s' = addn((1,2,3,4,-1,-2,-3,-4),8)}	_
{s' = addn((3,7,-3,-7),4)}	_
{s' = addn((10,-10),2)}	_
{s' = addn((0),1)}	_
{}	0

תרגילים

1) כתוב פונקציה (דומה במבנה ל- addn) שתמצא את המספר הקטן ביותר ב- v.
2) כתוב את הקטע הבא ע"י פונקצית עזר

$v_0=v_0+v_1$;
$v_1=v_2+v_3$;
.
.
.
$v_{n/2-1}=v_{n-2}+v_{n-1}$;

v=addpairs(v);

3) כתוב גירסה כללית של הפונקציה addn שתעבוד עבור כל שלם n כך ש- $n\geq 1$.

אלגוריתם גמיש להזזה מעגלית

להלן אלגוריתם גמיש (שתי פונקציות) להזזה מעגלית מיקום אחד שמאלה.

```
function v' = slc(v);          // Shift Left Circular - פונקציה ראשית
//
// מפרט: v ,v' הם מערכים //
// אבל האיברים הוזזו מיקום אחד שמאלה למעט מיקום אפס //
// האיבר במיקום אפס במערך v יופיע בסוף המערך v' //
{
  v'(length of v –1) = v0;
  moveleft(n=1);
}

function v' = moveleft(v,n)
//
// מפרט: v ,v' הם מערכים. האיברים מהמיקום //
// n והלאה ב- v יועברו ל- v' עם הזזה //
// n>=1 : מיקום אחד לשמאל. נדרש ש //
{
  if (n < length of v)
     {
        v'n-1 = vn;
         moveleft(n=n+1);
     }
}
```

תרגילים

1) המר את הנ"ל לתוכנית סדרתית כאשר התוצאה תופיע באותו מערך v.
2) כתוב את הפונקציות לעיל מחדש כך שאין צורך לטיפול מיוחד במיקום אפס.
   רמז: השתמש בפונקציה שארית, כלומר "%".
3) הכלל את הפונקציה slc כדי שתקבל **פרמטר נוסף** שיציין כמה מיקומים צריך להזיז שמאלה.
4) כתוב פונקציה src להזזה מעגלית של מיקום אחד **ימינה**.

אלגוריתם גמיש למיזוג (merge)

נניח שקיימים שני מערכים ממוינים לפי סדר "לא יורד". אנחנו מעוניינים למזג את שני המערכים למערך חדש ממוין כנ"ל שכולל את האיברים של שני המערכים הנתונים לדוגמא:

v' = merge ((1,1,3,4,9),(-1,3,5,7)) = (-1,1,1,3,3,4,5,7,9)

```
function v'=merge(v1,v2);
/*
```

מפרט: v',v1,v2, הם מערכים.
מניחים ש- v1 ו- v2 ממויינים לפי סדר "לא יורד".
מניחים שהאורך של v' שווה לאורך של המערך v1 + האורך של המערך v2.
v' גם יהיה ממוין כנ"ל ויכיל את האיברים של v2,v1.

```
*/

{loop (i=0;j=0;k=0)}

function v'=loop(v1,v2,i,j,k)
/*
```

מפרט: כנ"ל ובנוסף ב- v1 פועלים בתחום i והלאה, ב- v2 פועלים בתחום j והלאה, ב- v' פועלים בתחום k והלאה.
מניחים שאורך של v' פחות k שווה לאורך של v1 פחות i + אורך של v2 פחות j

```
*/
{
  if (i<length of v1) and (j < length of v2)
     if (v1_i ≤ v2_j)
        {v'_k=v1_i; loop(i=i+1;k=k+1);}
      else
        {v'_k=v2_j; loop(j=j+1;k=k+1);}
  if (i = length of v1 ) and (j < length of v2)
     { v'_k=v2_j; loop(j=j+1;k=k+1);}
  if (i < length of v1 ) and (j = length of v2)
     { v'_k=v1_i; loop(i=i+1;k=k+1);}
}
```

תרגילים

1) לבצע: v'=merge((1,2,4),(3,5))
2) לכתוב את הנ"ל מחדש בצורה יעילה יותר.
3) להמיר את האלגוריתם הגמיש הנ"ל לתוכנית סדרתית כך שהתוצאה תתקבל במערך חדש vv.

4) נניח שקיימת פונקציה copy עם מפרט כדלהלן.

```
function v' = copy(v,low,high,index)
/*
    מפרט: v הינו מערך ו- high , low הינם אינדקסים בתוך המערך v כך ש: low ≤ high.
    התאים במערך v יועתקו בטווח low עד high לתוך מערך התוצאה v' מהאינדקס index.
*/
```

כתוב מחדש את הפונקציה loop לעיל בעזרת הפונקציה copy.

5) כתוב את הפונקציה merge1 (דומה לפונקצית merge) שתמזג שני קטעים ממוינים בתוך המערך v ותשים את התוצאה באיזה מקום שרוצים בתוך מערך התוצאה v'. להלן מפרט של הפונקציה.

```
function v' = merge1(v,low1,high1,low2,high2,index);
/*
    מפרט: קיימים שני קטעים ממוינים במערך v בטווח שבין low1 עד high1 וכן בטווח שבין
    low2 עד high2, כאשר low1 ≤ high1 ו- low2 ≤ high2. התוצאה תופיע במערך v'
    מהמיקום index והלאה.
    אין חפיפה בין שני הטווחים האלה, כלומר או high1 < low2 או high2 < low1.
*/
```

6) המר את הפונקציהmerge1 שכתבת בסעיף 5 לתוכנית סדרתי.

אלגוריתם גמיש למיון מערך לפי שיטת "מיון זוגי אי-זוגי" (Even-Odd Sort)

נתאר שיטת מיון (לפי סדר "לא יורד") שמאפשרת ביצוע מקבילי שנקראת "מיון זוגי אי-זוגי" (Even-Odd Sort).

שיטת המיון: משווים זוגות במיקומים זוגיים ומחליפים לפי הצורך ומכניסים לתוך מערך תוצאה. את מערך התוצאה מעבירים תהליך מיון כנ"ל והפעם עבור הזוגות שמתחילים במקומות האי-זוגיים ומכניסים לתוך מערך תוצאה. חוזרים על הפעולה עבור הזוגות במקומות הזוגיים כנ"ל וחוזר חלילה. סיום התהליך יתאפשר כאשר מערך התוצאה ממוין. את המיון הנ"ל מבצעים באמצעות פונקציות עזר כאשר העיקריות שבהן הן: Is_sorted,opcs,epcs.

```
function v'=sort(v)
/*
                    מפרט: v' כמו v אבל ממוין לפי סדר "לא יורד"
*/
{if  (is_sorted(v))
    {v'=v}
    else {sort(v=opcs(epcs(v)))};
}

function v'=epcs(v)
/*Even Pair Compare Swap*/
/*
                    מפרט: v' כמו v אבל אם i זוגי ו- v_i>v_{i+1} אזי v'_{i+1}=v_i ו- v'_i=v_{i+1}
*/
{loop1(i=0)};

function v'=loop1(v, i);
                    /* מפרט: כמו epcs אבל פועלת על המיקומים i והלאה. */
{
 if ( i ≤ length of v-2 )
    {
      if (v_i > v_{i+1})
         { v'_i=v_{i+1}; v'_{i+1}=v_i}
      else
         { v'_i=v_i; v'_{i+1}=v_{i+1}};
      loop1(i=i+2);
    }
 else if length of v is odd - /* אורך אי זוגי, טיפול באיבר בודד */
         {v'_{length of v –1} = v_{length of v-1}};
}
```

דוגמת הרצה:

```
v'=sort ((5,4,3,2,1))
                epcs
                    (4,5,2,3,1)
                opcs
=sort((4,2,5,1,3))
                epcs
                    (2,4,1,5,3)
                opcs
=sort((2,1,4,3,5)
                epcs
                    (1,2,3,4,5)
                opcs
=sort((1,2,3,4,5))
=(1,2,3,4,5)
```

להלן הגדרת הפונקציה is_sorted

```
function  b'=is_sorted(v)
/*
                    b'=false אחרת  b'=true אזי v ממוין אם :מפרט
*/
{loop2(i=0)};

function b'=loop2(v,i);
{
  if ( i ≤ length of v-2 )
    if (v_i > v_i+1)
      {b'=false}
    else
      {loop2(i=i+1)};
  else
    {b'=true};
}
```

תרגיל

הפונקציה opcs דומה לפונקציה epcs, כתוב את הפונקציה כולל המפרט.

שיפור יעילות

אם נכתוב את הפונקציות epcs ו-opcs כך שבנוסף הן תחזירנה ערך בוליאני שמציין שהתבצעה החלפה או לא אזי ניתן לשפר את היעילות באופן משמעותי כך:

```
function v'=newsort(v)
{ ezer1(  v1,b1=newepcs(v) ) };
```

הערה: להלן v1 ו- v2 הינם מערכי עזר. b1 ו- b2 הינם משתנים בוליאניים

```
function v'=ezer1(v1, b1);
{ ezer2( v2,b2=newopcs(v1) ) };

function v'=ezer2(b1, v2, b2);
{
   if(b1 or b2)
     {newsort(v=v2)}
   else
    v'=v2;
}
```

תרגילים

1) בפתרון הנ"ל מספיק לבדוק b2 בלבד. למה?
2) כתוב את הפונקציות newepcs ו- newopcs כך שבנוסף הן תחזירנה ערך בוליאני שמציין שהתבצעה החלפה.

אלגוריתם גמיש למיון ע"י מיזוג (Merge Sort)

דוגמא

הערה: למען הפשטות נניח שגודל המערך הינו חזקה של 2.
בתחילה מתייחסים לכל מספר בודד כקבוצה נפרדת.

(8, 1 , 7, 2 , 6, 3 , 5, 4)

ממזגים כל זוג מספרים לחוד ומקבלים:

(1,8 , 2,7 , 3,6 , 4,5)

ממזגים כל שני זוגות מספרים ומקבלים:

(1,2,7,8 , 3,4,5,6)

ממזגים את שתי הקבוצות שנשארו ומקבלים:

(1,2,3,4,5,6,7,8)

הסבר

בשלב ראשון ממזגים ארבע זוגות של איברים בודדים. אחר כך ממזגים שתי זוגות של זוגות. אחר כך ממזגים זוג אחד של רביעיות. כשעוברים משלב לשלב גודל הקטעים שממזגים מוכפל בשתים ומספר הקטעים שממזגים קטנה כפול 2.

הערה

השיטה הנ"ל מאפשרת מקבילות ודומה לפונקציה addn שמאפשרת מקבילות.

תרגיל

כתוב פונקציה לפי המפרט:

```
function v'=m2 (v,size,place);
/*
```

מפרט: v הינו מערך שמכיל שני קטעים ממויינים לפי סדר "לא יורד". בגודל size כל אחד מהמיקום place והלאה. הפונקציה תמזג שני קטעים אלה לקטע ממוין אחד במערך התוצאה v' מהמיקום place והלאה באורך של 2*size. שאר אברי v' לא יקבלו ערכים.

```
*/
```

עתה נעשה שימוש בפונקציה m2 הנ"ל כדי למיין ע"י מיזוג.

```
function v'=mergesort(v)
/*
```

מפרט: v' כמו v אבל ממוין לפי סדר "לא יורד", בהנחה שגודל המערך הינו חזקה של 2.

```
*/

{loop(size=1);}


function v'=loop(v, size)
{
 if (size==length of v)
   {v'=v}
  else
    loop ( size=size*2;
           v=m2( v, size ,0 );
           v=m2( v, size, 2*size );
           v=m2( v, size, 4*size );
           v=m2( v, size, 6*size );
           .
           .
           .
           v=m2( v, size, length of v – (2*size) );
         );
 }
```

תרגילים

1) המר את הפונקציות הנ"ל לתוכנית סדרתית.
2) כתוב את הפונקציה mergesort מחדש כך שתעבוד עבור גודל מערך כלשהו.

רמז: השתמש בפונקציה merge1 במקום הפונקציה m2. הפונקציה merge1 מוגדרת בשאלה 5 בתרגילים שאחרי הסעיף "אלגוריתם גמיש למיזוג (merge)", בפרק זה.

אלגוריתם גמיש לחיפוש בינארי (Binary Search)

להלן פונקציה לחיפוש איבר בקטע בקטע ממוין בווקטור (לפי סדר "לא יורד").

```
function place'= bsearch(v,low,high,value)
/*
  מפרט: נחפש את האיבר value בקטע ממוין שבין low עד high במערך v. אם הערך value
      נימצא במערך v בקטע שבין low עד high אזי 'place יכיל את המיקום (האינדקס) שבו
נימצא הערך. אם value לא נימצא בקטע הנ"ל או אם low>high אזי נציב את -1 ב- 'place.
*/
{
  if low>high
    { place'= -1; } /* code for not found */
  else
    {  if value==v(low + high)/2
         {place'=(low + high)/2}
       else
         if value < v(low + high)/2
           {bsearch( high=((low + high)/2)-1) );}
         else
           {bsearch( low=((low + high)/2+1) );}
     }
  }
```

לדיון בכיתה

כיצד לשנות את הפונקציה לעיל כדי לחפש בקטע לא ממוין?

תרגילים

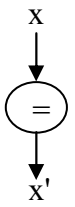
1) הפונקציה bsearch כתובה בצורה לא יעילה, כי הערך (low + high)/2 יחושב כמה פעמים. השתמש בפונקצית עזר עבור הקטע אחרי ה- else הראשון, כדי שערך זה יחושב לא יותר מפעם אחת.

2) ניתן לכתוב את הפונקציה bsearch בלי שני ה- "else" האחרונים אבל עם "if" נוסף. כתוב הפונקציה הנ"ל מחדש. האם יש יותר אפשרויות לבצוע מקבילי בפונקציה החדשה? אם כן, מה בדיוק?

פרק חמישי - פיתוח דיאגרמות בלוקים של חומרה

חיפוך ווקטור בגודל קבוע

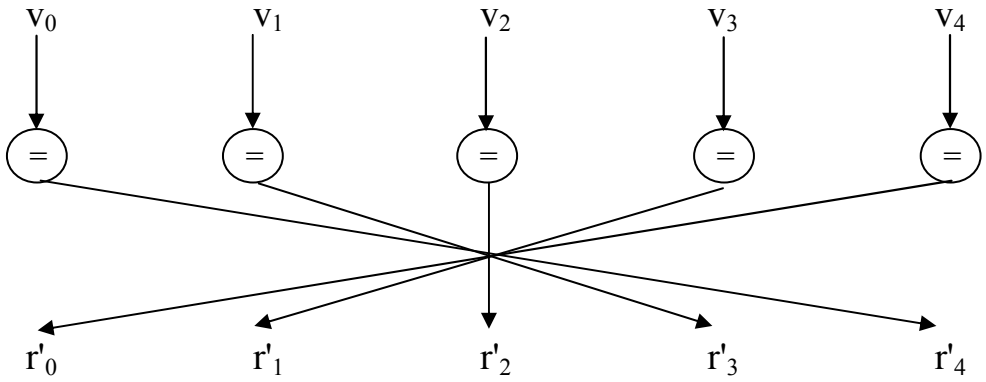
Let us return to the function reverse we presented in chapter 1. Let us say we have a circuit for performing a copy operation x' = x.



We can produce a block diagram of a circuit for reversing a vector of five elements from the set { r' = reverse (v, 0, 4); } by symbolically executing the program (i.e. substituting using function definitions as in mathematics). The execution is shown as a sequence of equivalent sets of statements as follows:

{ r' = reverse (v, 0, 4); }
≡ { $r'_4 = v_0$; r' = reverse (v, 1, 3); $r'_0 = v_4$; }
≡ { $r'_4 = v_0$; $r'_3 = v_1$; reverse (v, 2, 2); $r'_1 = v_3$; $r'_0 = v_4$; }
≡ { $r'_4 = v_0$; $r'_3 = v_1$; $r'_2 = v_2$; $r'_1 = v_3$; $r'_0 = v_4$; }

As the last line contains only primitive operations, it essentially describes the following block diagram for carrying out the reverse. (As no output is being generated, there is no need give the state of the output variables, which we gave in previous examples of actual computations.)



Of course this technique can be used for any constant size vector.

חיבור של שני מספרים עם נשא (ספרה ספרה או ביט-ביט)

האלגוריתם הגמיש

נניח שקיימת פונקציה a3b שמחברת שלוש ספרות (או ביטים) ונותנת שתי תוצאות: ספרה אחת והיא הסכום וספרה אחת והיא הנשא. לדוגמא בבצוע של
c' , s' = a3b(9,3,1) יגרום ש- s'=3 ו- c'=1 (s' הינו סכום ו- c' הינו נשא. כל אחד ספרה אחת בלבד)
להלן אלגוריתם (הפונקציה add) גמיש לחיבור שני מספרים של כמה ספרות (או ביטים) עם נשא קיים (ספרה אחת או ביט אחד)
הפונקציה add נותנת שתי תוצאות: הסכום שהוא מספר של כמה ספרות ונשא שהוא ספרה אחת.
שים לב, בשני המספרים והסכום, יש אותו מספר של ספרות (n+1 ספרות), כאשר n הנו המיקום של הספרה הימנית. שים לב שמספר מוצג ע"י מערך של ספרות (או ביטים), ומיקום הספרה השמאלית הינו אפס.

```
function c', s' = add(u,v,c,n);
/*
SPECIFICATION
IN - "n" denotes the position of the least significant bits of u,v, to be added, where n≥ 0.
The position of the most significant bit "0". Bit "c" is also added at the position of the
least significant bits of u, v.
OUT - the carry of the addition is produced in c' and the sum itself in s'.
*/
{
 if (n>0)
    add( c, s'n= a3b(un, vn, c); n=n-1);
    else c', s'0=a3b(u0, v0, c);
} /* add */
```

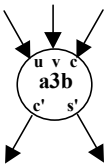
לדוגמא, הביצוע של "c',s'=add(123,987,6,2);" יגרום שבסופו של דבר s'=116 ו- c'=1.

<u>תזכורת</u>:
c,n באגף ימין הם משתנים קיימים ואילו c,n באגף שמאל הם משתנים חדשים עבור הביצוע החוזר הבא.

הפונקציה add עושה שימוש בפונקציה a3b. להלן המפרט של הפונקציה a3b.

```
function c', s' = a3b(u, v, c)
/*
SPECIFICATION:
IN - u, v, c, are the bits to be added.
OUT - c' is the carry and s' is the sum.
*/
```

This operation can be represented diagrammatically as follows:



להלן ביצוע מפורט לפי השיטה המקבילית:

set of statements		c'	s'
u v c n			
{ c',s' = add(123, 987, 6, 2) }	→	_, _, _, _,	
{ c',s' = add(123, 987, 1, 1) }	→	_, _, _, 6,	
{ c',s' = add(123, 987, 1, 0) }	→	, _, 1, 6,	
{ c',s'$_0$ = a3b(1, 9, 1) }	→	, _, 1, 6,	
{ }	→	1, 1, 1, 6,	

Develop a block diagram of a circuit for a 4 bit serial adder.

{c',s'=add(u,v,c,3);}

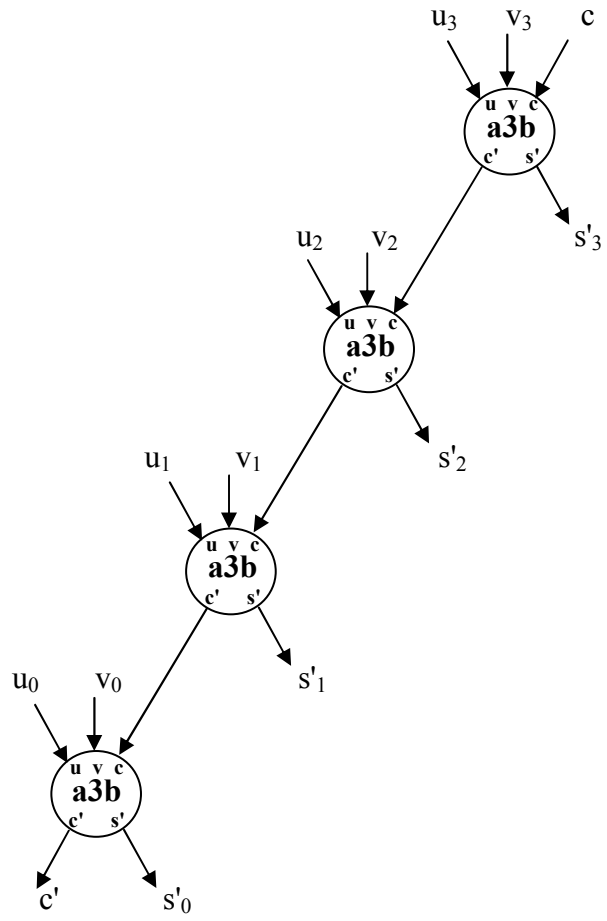
≡{add(c,s'_3=a3b(u_3,v_3,c);n=2);}

≡{add(c,s'_2=a3b(u_2,v_2;c,s'_3=a3b(u_3,v_3,c));n=1);}

≡{add(c,s'_1=a3b(u_1,v_1;c,s'_2=a3b(u_2,v_2;c,s'_3=a3b(u_3,v_3,c)));n=0);}

≡{c',s'_0=a3b(u_0,v_0;c,s'_1=a3b(u_1,v_1;c,s'_2=a3b(u_2,v_2;c,s'_3=a3b(u_3,v_3,c))));}

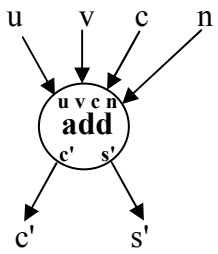
As the last line contains only primitive operations, it essentially describes the following block diagram for this serial adder.



The above development was hard to follow.

Here is a diagrammatic development that is much easier to understand.

A call c', s'=add(u, v, c, n) can be represented by:



This diagram is equivalent to one of the following depending whether or not n>0.

Diagram for n>0

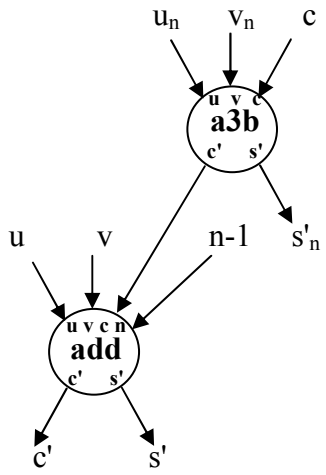
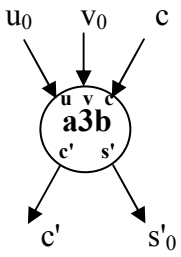
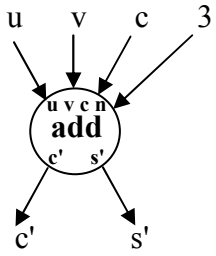


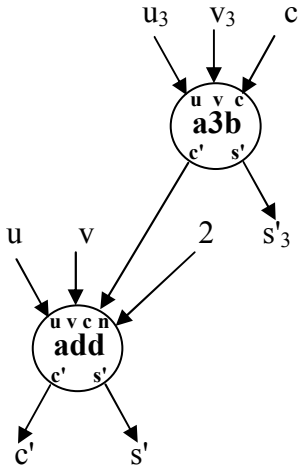
Diagram for n=0



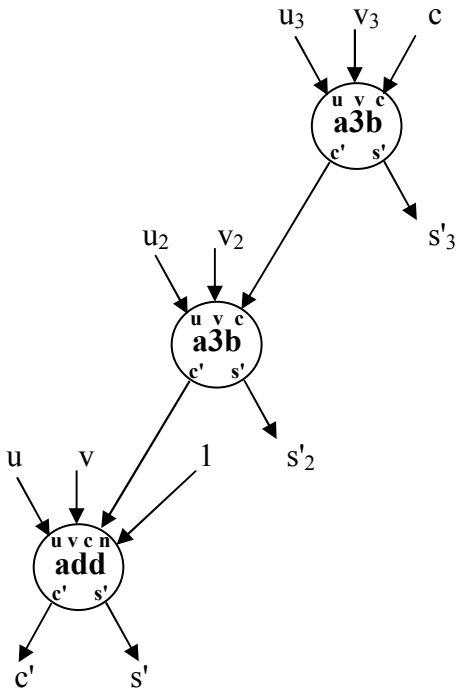
The call for a 4 bit adder is c', s'=add(u, v, c, 3); which is represented by:



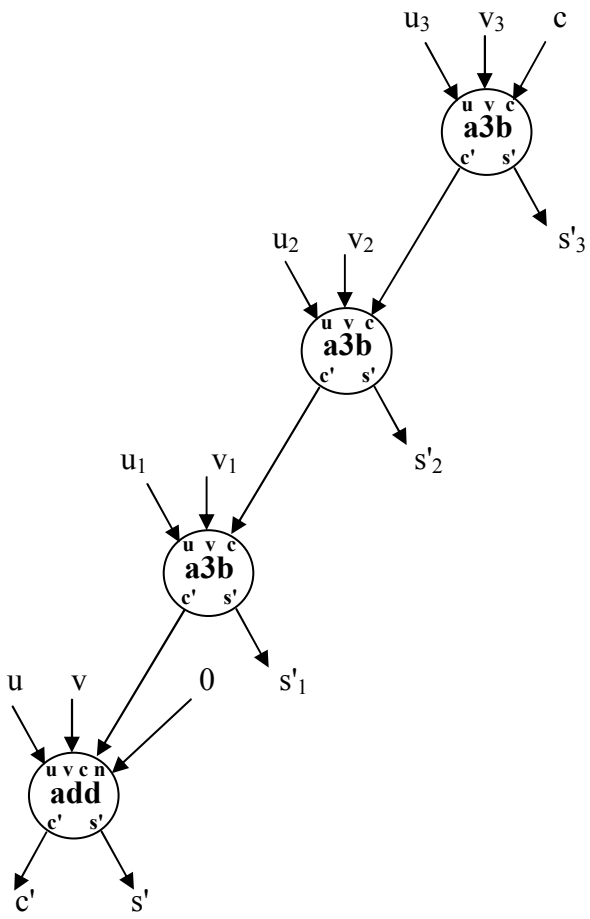
Equivalent to:



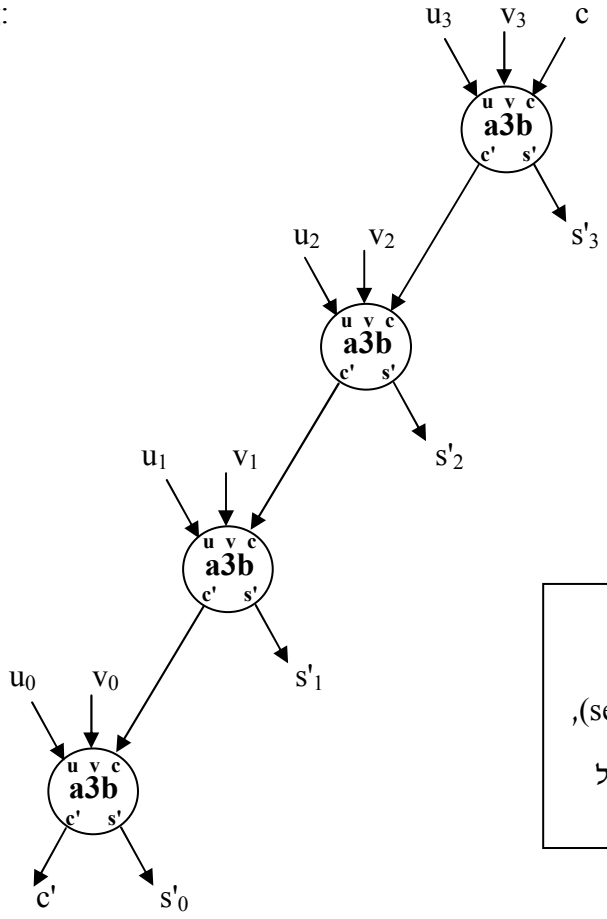
Equivalent to:



Equivalent to:



Finally giving:



שים לב:

שרטוט סופי זה מייצג מעגל לחיבור סידרתי (serial adder), שקיבלנו מאלגוריתם גמיש. תראו אותו בקורס או ספר על לוגיקה ומערכות ספרתיות.

פרק שישי - הכרה ראשונית של המושג מערכת והקשר עם אלגוריתמים גמישים

מערכת היא אוסף של רכיבים שפועלים ביחד
(סידרתי/מקבילי) יש פער גדול בין אלגוריתם סידרתי
לבין מערכת. אבל מול אלגוריתם גמיש יש דברים דומים, והאופן שתיארנו אלגוריתם גמישים, מתאים באופן סביר לתיאור כללי של מערכת.

דוגמא: מכונה/מערכת לבדיקת דם יכולה לבדוק כמות הסוכר כולסטרול בדם (Sugar/Cholesterol tast) להלן התיאור כאלגוריתם גמיש.
הגישה הגמישה לתיאור אלגוריתם, מאפשר כמה אפשרויות לממש אותם. להלן אלגוריתם גמיש וכמה אפשרויות של תכנון מכונה/מערכת כזו, עם פרוט פנימי של המכונה או מערכת

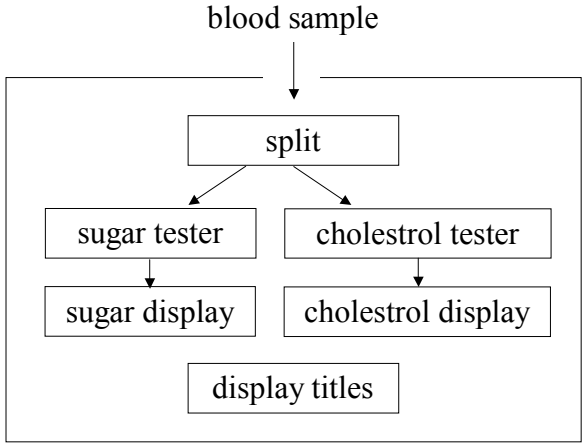
אלגוריתם גמיש:

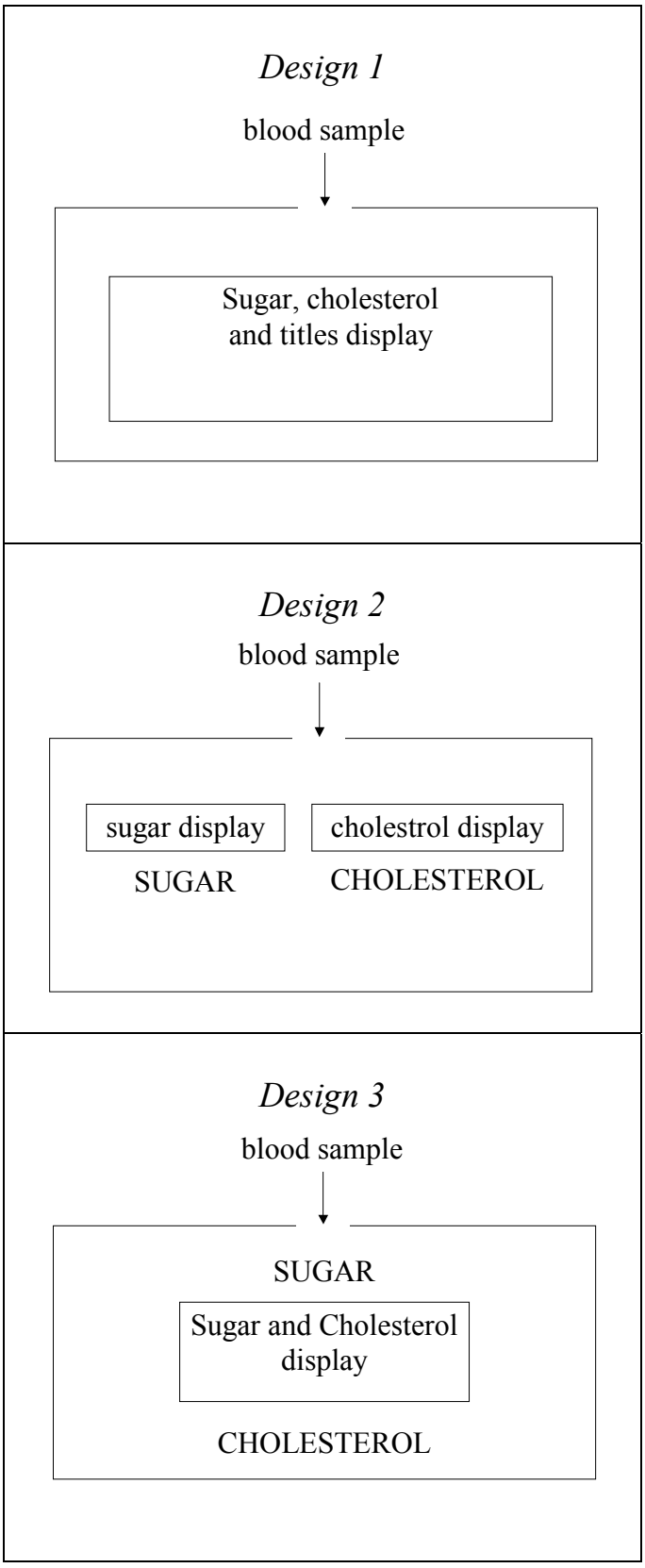
```
function title1', sugar', title2', cholesterol' =
bloodtester(blood_sample);
{ two_tests( sample1, sample2 = split(blood_sample) );
}; /* end bloodtester */

function title1', sugar', title2', cholesterol' =
two_tests (sample1, sample2);
{ title1'="SUGAR";
  sugar' = sugartester(sample1);
  title2'="CHOLESTEROL";
  cholesterol' = cholesteroltester(sample2); }
}; /* end two_tests */
```

Example of Blood test machine

Machine can test Sugar, Cholesterol in a blood sample.




Design 1
blood sample
Sugar, cholesterol
and titles display
Design 2
blood sample
sugar display
cholestrol display
SUGAR
CHOLESTEROL
Design 3
blood sample
SUGAR
Sugar and Cholesterol
display
CHOLESTEROL

פרק שביעי - נכונות של אלגוריתמים

הרצת אלגוריתם על מספר דוגמאות של קלט נותנת ביטחון מסוים שהאלגוריתם תקין אבל אין בזה עדיין להוכיח שהאלגוריתם יעבוד עבור כל קלט אפשרי. אנחנו זקוקים לשיטה מתמתית לניתוח בצוע של כל קלט ולפיכך נציג בהמשך שיטה שנקראת "אינדוקציה חישובית" (Computational Induction)

חזרה על אינדוקציה

להלן שתי שיטות אינדוקציה על שלמים לא שליליים.

אינדוקציה פשוטה על שלמים לא שליליים (לא שימושי לצורכנו)

1. הוכח $p(0)$.
2. הוכח שאם $p(k)$ אזי $p(k+1)$.

המסקנה: $p(n)$ נכון עבור כל שלם לא שלילי.

אינדוקציה משוכללת על שלמים לא שליליים (כן שימושי לצורכנו)

1. הוכח $p(0)$.
2. הניח $p(m)$ עבור m כך ש- $n>m$ והוכח $p(n)$.

(ניסוח אחר: הוכח שאם $p(0), p(1)\ldots p(n-1)$ אזי $p(n)$.)

המסקנה: $p(n)$ נכון עבור כל שלם לא שלילי.

דוגמא של אינדוקציה משוכללת

נניח ש- $S_0=2, S_1=5, S_n=5S_{n-1}-6S_{n-2}$ כאשר $n>1$. נוכיח ש- $S_n=2^n+3^n$.

1. כאשר $n=0$, $2^0+3^0=2=S_0$ וזה מתאים לטענה.
2. כאשר $n=1$, $2^1+3^1=5=S_1$ וזה מתאים לטענה.
3. נטפל עכשיו במקרה $n>1$.

מפני ש- $n>n-1$, אז נניח ש- $S_{n-1}=2^{n-1}+3^{n-1}$

מפני ש- $n>n-2$, אז נניח ש- $S_{n-2}=2^{n-2}+3^{n-2}$

ולכן:

$$\begin{aligned}S_n&=5S_{n-1}-6S_{n-2}=5(2^{n-1}+3^{n-1})-6(2^{n-2}+3^{n-2})\\&=10*2^{n-2}-6*2^{n-2}+15*3^{n-2}-6*3^{n-2}\\&=4*2^{n-2}+9*3^{n-2}=2^2*2^{n-2}+3^2*3^{n-2}=2^n+3^n\end{aligned}$$

מ.ש.ל.

לכן הנוסחה נכונה עבור כל שלם n לא שלילי.

אינדוקציה חישובית

זו היא שיטה להוכחת נכונות חלקית של אלגוריתם, כלומר אם הביצוע יעצור אזי התוצאות שתתקבלנה מהאלגוריתם תהיינה נכונות לפי המפרט. מפרט פירושו: תיאור מדויק של התוצאות האמורות להתקבל מהאלגוריתם. אינדוקציה חישובית מבוססת על אינדוקציה משוכללת לעיל כאשר n הנו מספר הקריאות (או הצעדים) שבוצעו במהלך בצוע האלגוריתם. $p(n)$ פירושו שתוצאות האלגוריתם נכונות כאשר יש n קריאות (או צעדים) לבצוע.

הערות:

1) שיטה זאת אינה מוכיחה עצירת בצוע.

2) בקורס זה לא נדון בנושא של "נכונות מליאה" כלומר שהאלגוריתם יעצור תמיד ויתן תוצאות נכונות לפי המפרט.

המבנה של אינדוקציה חישובית

1) בודקים שכל תוצאה שרשום במפורש, תואמת למפרט.
2) מניחים שכל קריאה פנימית פועלת לפי המפרט ומוכיחים שהתוצאות המתקבלות בסוף מתאימות למפרט, לפי שרשומים בכותרת הפונקציה.

מסקנה: בהנחה שהבצוע יעצור, התוצאות תהיינה נכונות לפי המפרט.

דוגמת reverse

נוכיח עכשיו שהאלגוריתם\פונקציה reverse שכבר ראינו נכונה חלקית ע"י אינדוקציה חישובית (Computational Induction)

בהתחלה כל האברים במערך 'v ריק, כלומר:

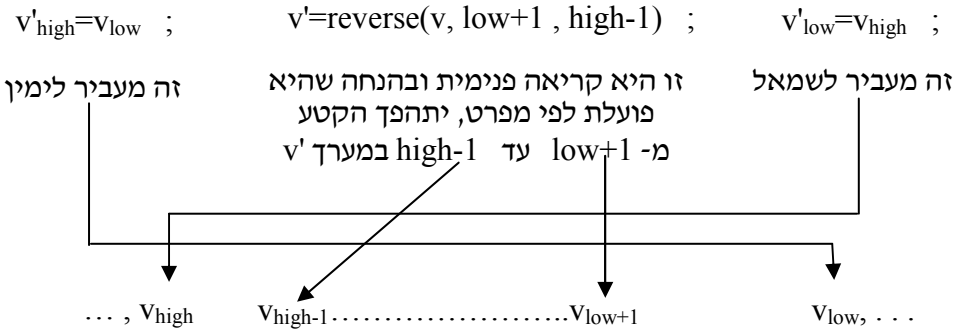
____ , ... , ____ , ____ , ... , ____ , ____ , ... , ____

ישנם שלושה מקרים כדלהלן:

- low<high
- low=high
- low>high

המקרה של low<high

כאן צריך לבצע:



והרי תוצאה זו תואמת למפרט של v'=reverse(v,low,high).

הערה: הקשר עם אינדוקציה משוכללת הוא שמספר הצעדים או קריאות בצוע של v'=reverse(v,low+1,high-1) קטן יותר מהמספר בביצוע של v'=reverse(v,low,high) ולכן מותר להניח שהקריאה v'=reverse(v,low+1,high-1) פועלת לפי המפרט, וכל זה בהנחה שהבצוע יעצור.

המקרה של low=high

כאן רק v_{high} עובר לאותו מקום ב- 'v, כלומר 'v יהיה:

____ , ____ , ____ , ... v_{high} , ____

והרי זה תואם למפרט.

המקרה של low>high

כאן אף ערך לא עובר ל - 'v והרי זה תואם למפרט. מסקנה מהנ"ל: בהנחה שהבצוע יעצור, התוצאות תהיינה נכונות לפי המפרט של reverse.

מ.ש.ל.

דוגמת bugreverse

התבונן בגירסא הבאה של אלגוריתם שאמור להפוך קטע בווקטור אבל יש בו טעות.

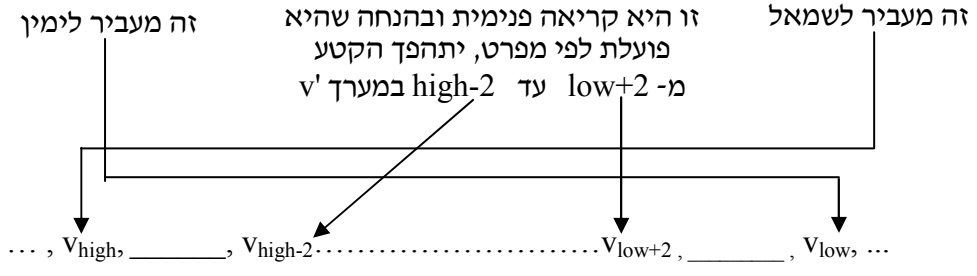
```
function v'=bugreverse(v,low,high);
/*
reverse מפרט: כמו
*/
if (low<high)
{
  v'high=vlow;
  v'=bugreverse(v,low+2,high-2);   <-- טעות כאן
  v'low=vhigh;
}
else
  if (low==high)
    { v'high=vhigh; }
} /*end bugreverse */
```

עתה נבדוק את הפונקציה הנ"ל ע"י אינדוקציה חישובית ונגלה שיש טעות. בהתחלה המערך 'v ריק.

המקרה של: low<high

כאן צריך לבצע:

$v'_{high}=v_{low}$; v'=bugreverse(v, low+2 , high-2) ; $v'_{low}=v_{high}$;



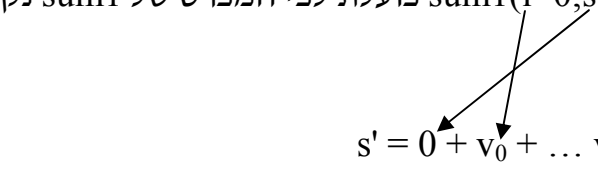
אנחנו נוכחים לראות שלא כל הקטע מתהפך כלומר שיש כאן טעות !

דוגמת sum , sum1

נוכיח עתה שהפונקציות sum ו- sum1 שכבר ראינו נכונות באופן חלקי באמצעות השימוש באינדוקציה חישובית.

הפונקציה sum

בהנחה שהקריאה הפנימית sum1(i=0;s=0) פועלת לפי המפרט של sum1 נקבל:



והרי זה תואם למפרט של sum.

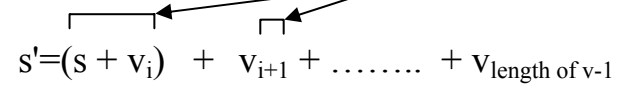
הפונקציה sum1

יש שני מקרים:

- i < length of v
- i ≥ length of v

המקרה של i<length of v

כאן מבצעים: sum1(i=i+1;s=s+vi) ובהנחה שקריאה פנימית זאת פועלת לפי המפרט של sum1 נקבל:



והרי זה תואם למפרט של: s'=sum1(v,i,s)

המקרה של i ≥ length of v

במקרה זה s'=s והרי זה תואם למפרט.

המסקנה: בהנחה שהבצוע יעצור, התוצאות של sum ושל - sum1 תהיינה נכונות לפי המפרט.

דוגמת bugsum1

התבונן בגירסא הבאה של הפונקציה sum1 שיש בה טעות.

```
function s'=bugsum1(v,i,s);
/* sum1 כמו : מפרט */
{
 if i < length of v
   { bugsum1 (i=i+1;s=i+v_i) }
 else
   { s'=s; }
}
```

טעות כאן

עתה נעשה שימוש באינדוקציה חישובית כדי לגלות שיש טעות. יש שני מקרים:

המקרה של i < length of v

כאן מבצעים: bugsum1(i=i+1;s=i+v_i) ובהנחה שקריאה פנימית זאת פועלת לפי המפרט של sum1 נקבל:

$$s'=(i + v_i) + v_{i+1} + \ldots\ldots\ldots + v_{\text{length of v-1}}$$

צריך להיות s

והרי זה לא תואם למפרט של s'=bugsum1(v,i,s). כלומר יש טעות!

תרגילים

1) הוכח ע"י אינדוקציה חישובית (Computational Induction) שהפונקציה bsearch נותנת תוצאה נכונה לפי המפרט בהנחה שהבצוע יעצור.

2) התבונן בפונקציה:

```
function s'=sq1(n,s);
/*
                    מפרט: כאן n הנו מספר שלם, ואם n>0 אזי s'=n^2+s אחרת s'=s
*/
{
if n > 0
  {sq1(n=n-1; s=s+2*n+1);}
else
  {s'=s};
}
```

א. בדוק את הפונקציה הנ"ל ע"י אינדוקציה חישובית. (שים לב שיש טעות)
ב. תקן את הפונקציה הנ"ל. יש לשנות רק סימן אחד.
ג. בדוק את התיקון ע"י אינדוקציה חישובית.

3) *[קשה]* הוכח שהפונקציה addn נכונה חלקית ע"י אינדוקציה חישובית (Computational Induction).

חומר נוסף מתקדם באנגלית

ראה בדף ב- Internet באתר: http://cc.jct.ac.il/~rafi

א. SFL - Simple Flexible Language

ב. First Steps in Computer Science: Sequential or Parallel ?