Skip Lists רשימות דילוגים

Skip lists: A probabilistic Alternative to Balanced Trees, William Pugh, Communications of the ACM, 33(6):668-676, 1990.

המאמר נמצא גם באתר הקורס תחת skip list animation וכן באתר:
www.cs.umd.edu/users/pugh (selected publications)

©Geiger & Itai, 2001

רשימות דילוגים Skip Lists

עצים מאוזנים, 2-3 ו- AVL שנלמדו בשני השעורים הקודמים, מְמַמְּשִׁים את פעולות המילון (חיפוש, הכנסה, הוצאה) בזמן O(log n) במקרה הגרוע ביותר.

מימוש הפעולות אינו טריוויאלי,

במיוחד כאשר יש צורך לתמוך גם בפעולות הדורשות לשמור בכל צומת אינפורמציה ייחודית (כמו rank) ולעדכן אותה.

נלמד כעת מבנה נתונים בעל מימוש פשוט מאוד, שלא דורש איזונים של מבנה הנתונים לאחר הכנסה/הוצאה.

בִּתְמוּרָה: בצוע פעולות המילון יעשה בזמן O(log n) בממוצע.

למרות זאת, ההסתברות שזמן בצוע פעולה יחרוג בצורה משמעותית (נ˙אמַר פי 3) מהזמן הממוצע היא זניחה (פחות מ- 1 ל 100,000,000).

©cs,Technion

מהו "ממוצע" ?

ראינו כבר (בשיעור 3) שעצי חיפוש ללא פעולות איזון יכולים לשמש למימוש פעולות המילון בזמן $O(\log n)$ ב"ממוצע".

מהו הממוצע הזה ? בניתוח שעשינו הנחנו שהוכנסו n איברים כך שההסתברות לכל אחת מ- $n!$ הפרמוטציות להכנסתם אחידה. הממוצע של גובה $n!$ העצים שנוצרו הוא $O(\log n)$. אבל – ההנחה שעשינו הינה על הקלט!

הבעייתיות בניתוח זה: ההסתברות להופעת פרמוטציה בקלט אינה ידועה ואין שום סיבה להניח יוניפורמיות. אנו מעוניינים במבני נתונים + הבטחות ביצועים ללא תלות בהנחות מְפֻקְפָּקוֹת על הקלט. בפרט היינו מעוניינים להיות חֲסִינִים ל-"יריב" המעוניין להאיט את ריצת האלגוריתמים.

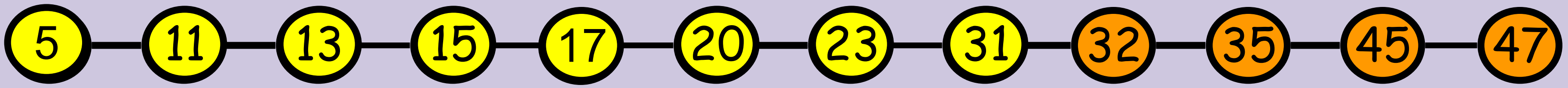
ברשימות דילוגים, שעליהן נלמד היום, נקבל ביצוע בזמן $O(\log n)$ בממוצע.

אולם: המבנה וזמן ביצוע הפעולות תלויים בהגרלות שעושה המחשב ולא בסדר הכנסת הנתונים. הממוצע מחושב לפי ההגרלות ולא לפי הנחות על הסתברות הקלטים. אלגוריתם כזה נקרא **אלגוריתם רנדומלי**.

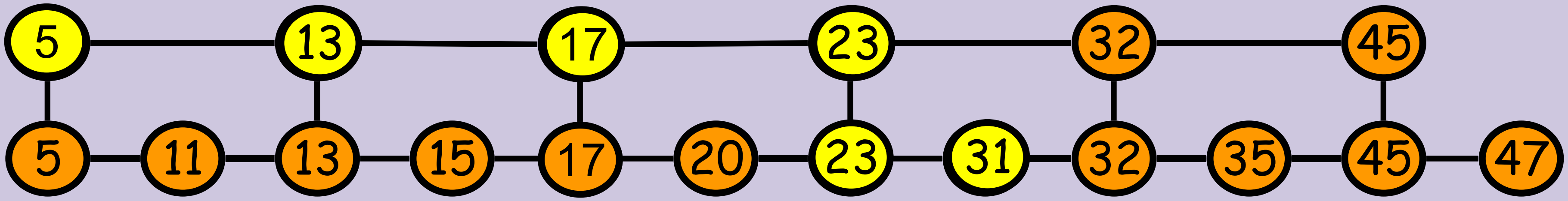
©cs,Technion

הרעיון המרכזי

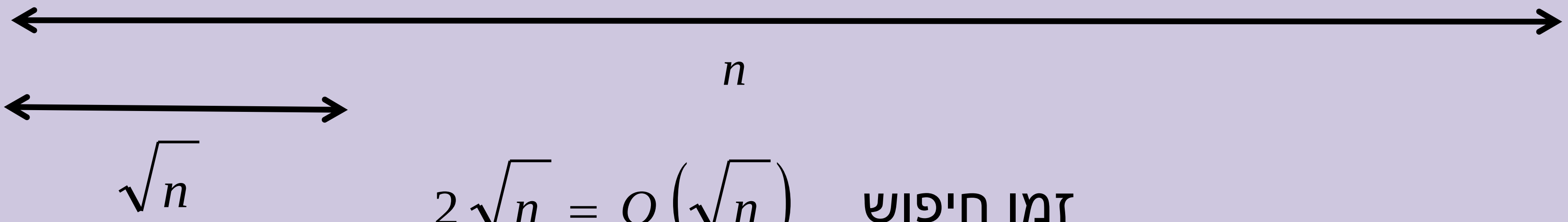
חיפוש איבר בסוף רשימה הוא יקר.



כדי לזרז את החיפוש (פי שניים) נוסיף מַדְרִיךְ- תת רשימה של כל איבר שני:



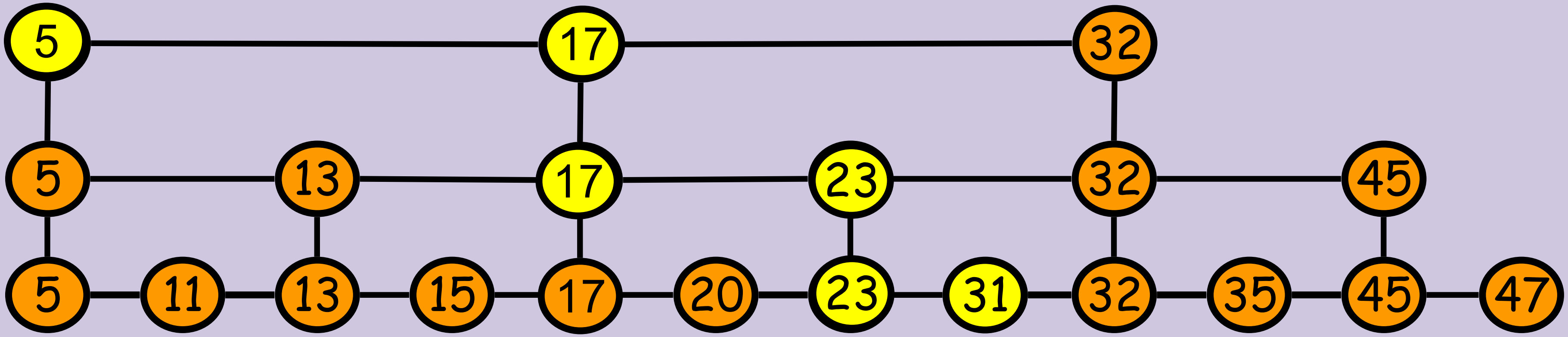
מהו זמן החיפוש המינימלי?



זמן חיפוש $2\sqrt{n} = O\left(\sqrt{n}\right)$

©cs,Technion

כדי לזרז את החיפוש במדריך נוסיף רמה נוספת:

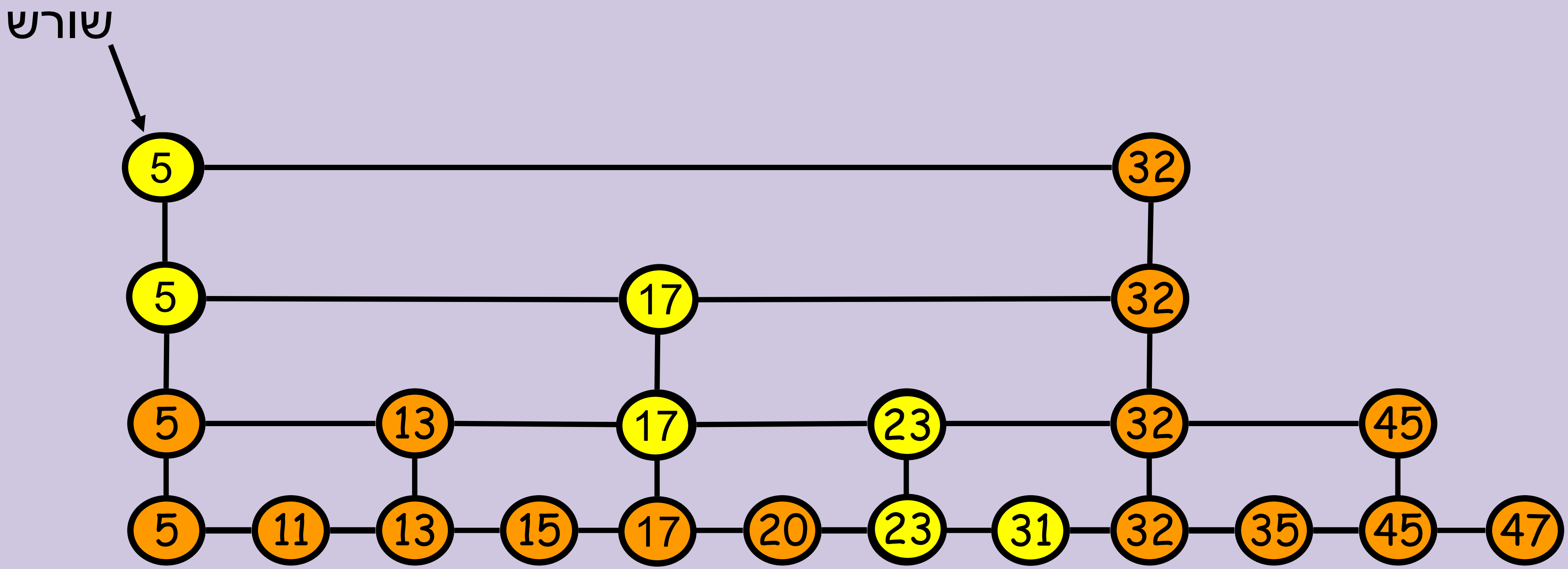


וכך הלאה עד לרמה $\lceil \log n \rceil$ ובה איבר בודד.

©cs,Technion

אלגוריתם החיפוש

נניח שמחפשים מפתח x. נתחיל משורש המבנה. בכל רמה נצעד ימינה כל עוד המפתחות שנסרקים קטנים או שווים ל-x. עם מיצוי הרמה נרד רמה אחת, וכו'.



©cs,Technion

אורך מסלול החיפוש

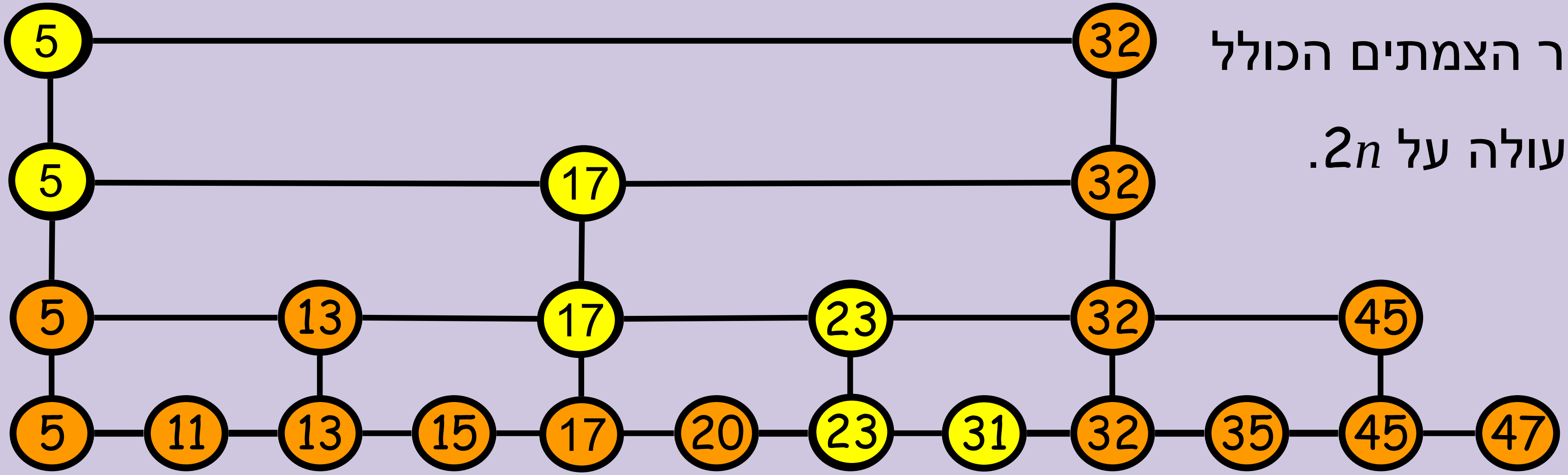
נסתכל על מסלול החיפוש מצומת הסיום ועד צומת ההתחלה.

במעבר מרמה לרמה אנו עוברים על פני 2 מצביעים לכל היותר.

לפיכך אורך המסלול הוא $2\log n$.

מספר הצמתים הכולל

אינו עולה על $2n$.



הבעיה שנותר לפתור היא כיצד לשמור על המבנה שיצרנו בעת הכנסה והוצאה מבלי שיהיה צורך לארגן את כל הרמות מחדש.

נשים לב שאלגוריתם החיפוש נכון גם אם מספר הצמתים של רמה i בין כל שני צמתים של רמה i-1 אינו קבוע.

הכנסה באמצעות הטלת מטבע

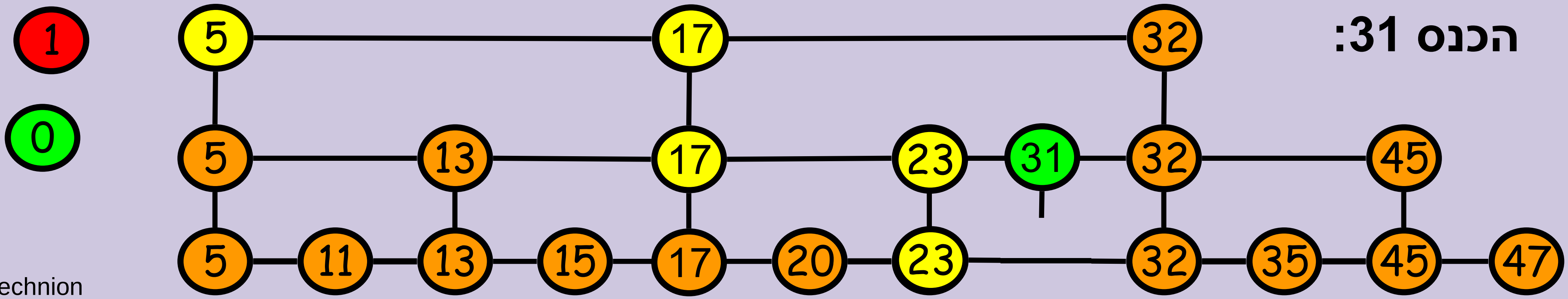
צעדים להכנסת מפתח k.

1. חפש את k. אם k נמצא סיים.
2. הוסף צומת חדש עם מפתח k ברמה התחתונה ביותר (במקום המתאים!)
3. לפי סדר הרמות מלמטה למעלה:

   הגרל מטבע toss( ). אם יוצא 0: הוסף צומת חדש מעל הרמה הנוכחית וקבע את המפתח בו להיות k. אם יוצא 1, עצור.

4. אם ברמה העליונה הוגרל 0, הוסף רמה חדשה.

הגרלה

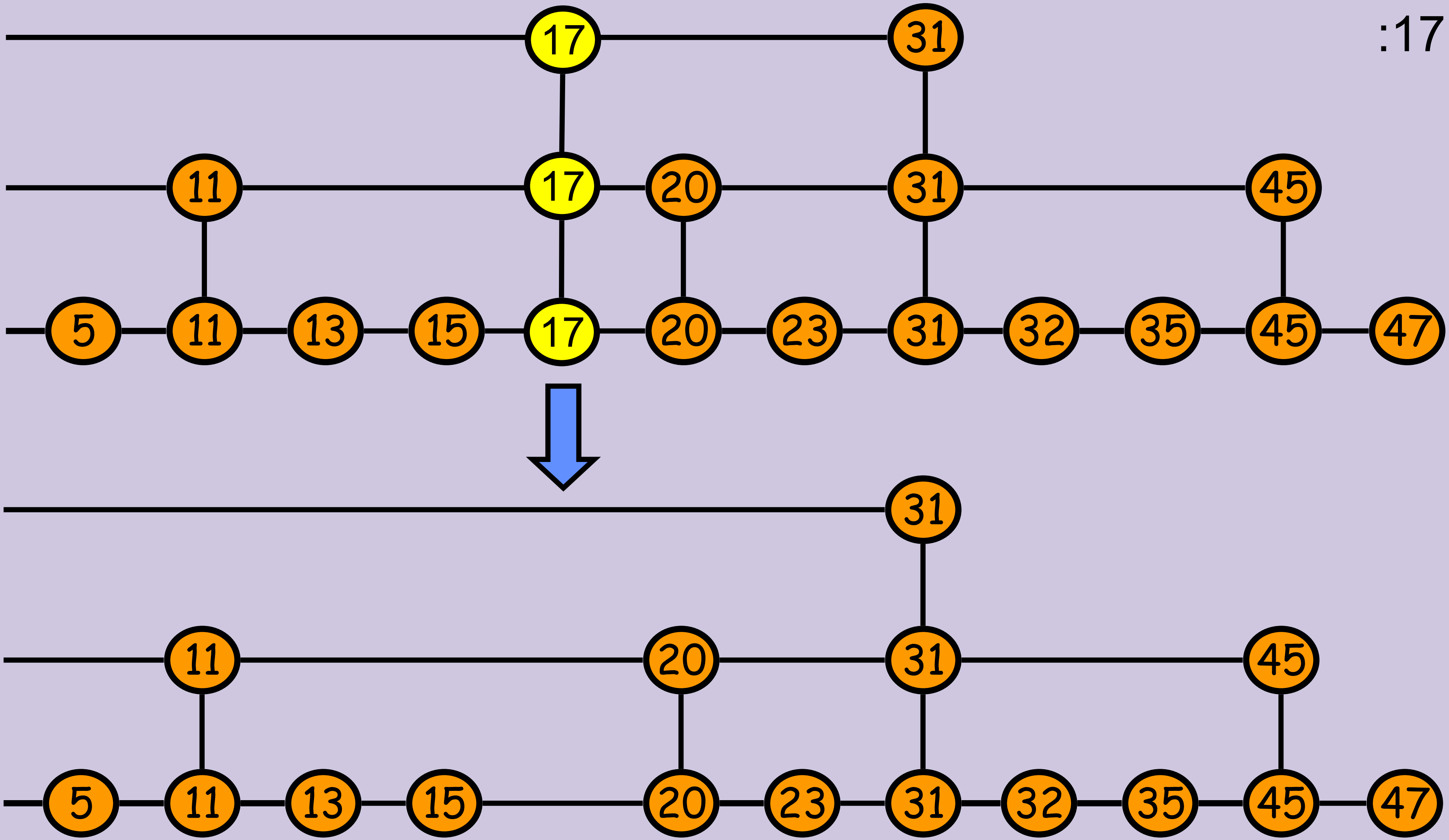


הכנס 31:

הוצאה מרשימת דילוגים

1. מצא את האיבר בעל המפתח k.
2. הוצא איבר זה מכל הרמות בהם הוא מופיע.

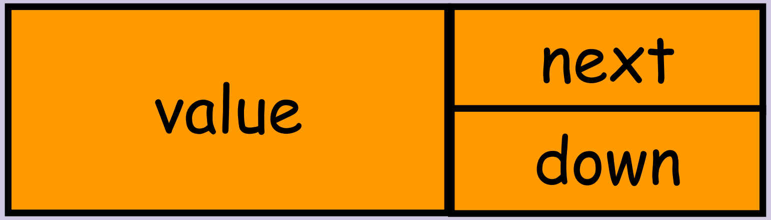
הוצא 17:



תוכניות לרשימת דילוגים

הגדרת צומת ופרוצדורת עזר ליצירתו.

```
typedef  struct  node {
        Key  value ;
        struct node  *next,  *down ;
        } NODE ;
NODE  *top ;
```




```
NODE   *new_node (KEY k, NODE  *n, NODE  *d)
{       NODE  *p ;
        p = (NODE *) malloc( sizeof ( NODE ))  ;
        p → value = k ;
        p → next = n ;
        p → down = d ;
        return p ; }
```

הערות לגבי התוכניות

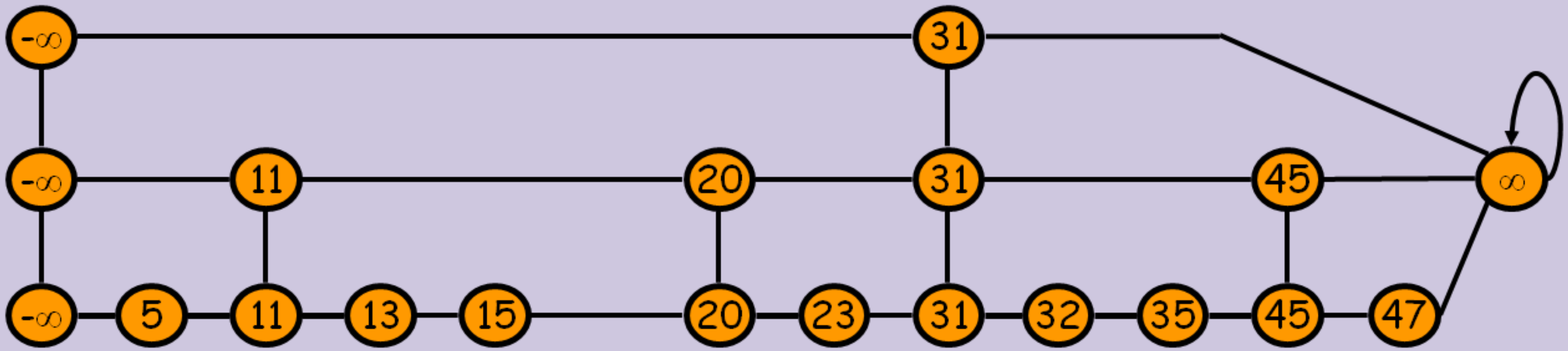
נוח להוסיף בכל רמה צומת אחד בהתחלת הרשימה שמפתחו $-\infty$ וצומת אחד בסוף שמפתחו $+\infty$ (sentinel).

תוכנית ההכנסה רקורסיבית אך ניתן לכתוב אותה גם ללא רקורסיה בקלות רבה.

באתר הקורס תחת skip list animation קיים קוד לא רקורסיבי בשפת C++ ובשפת Java. יש הבדלים מסוימים בין התוכניות השונות.

הפרוצדורה add_after(k,p) מוסיפה צומת חדש בעל מפתח k אחרי p, ומחזירה מצביע אליו.

הפרוצדורה toss( ) מחזירה 0 או 1 בהסתברות שווה.

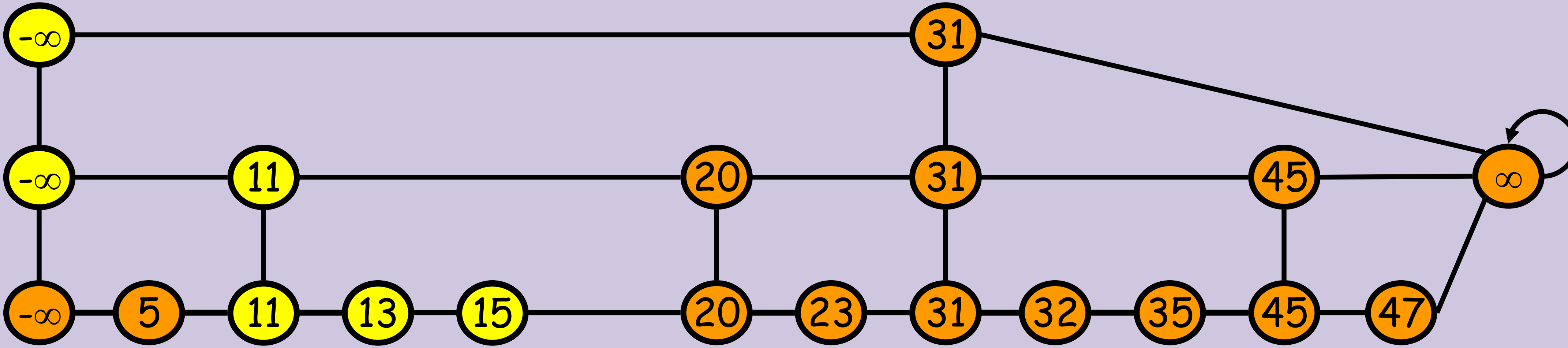


©cs,Technion

תוכנית חיפוש ברשימת דילוגים

```
NODE   *search (KEY k, NODE  *p)
{        /* returns a pointer p to last node ≤ k  */
         while(1) { for ( ; p → next →  value  <= k ; p =  p → next) ;
                  if ( p → down = = NULL)  break;   /* final level */
                  p = p → down ;  }
         return p ; }
```

מצא 17: לא נמצא.

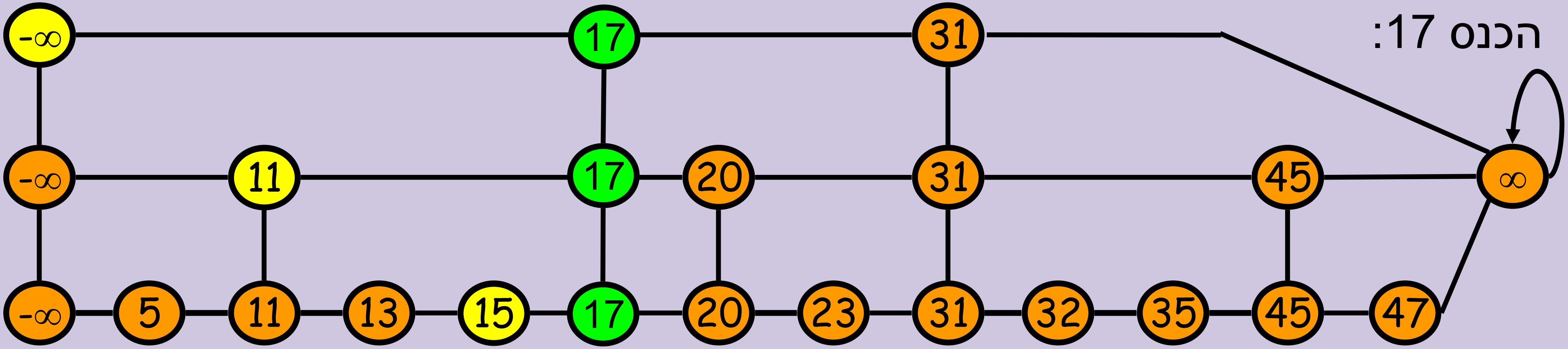


תוכנית להכנסה ברשימת דילוגים

```
NODE   *insert1 (KEY k, NODE  *p)
{          /* returns a pointer to the added node */
   NODE * deeper, *new_node;
   for ( ; p → next!=NULL && p → next→value  <= k ; p =  p→next) ;
   if ( p → down = = NULL)  return add_after(k,p) ;         /* final level */
   deeper = insert1(k, p→down);
   if (( deeper == NULL || toss( ) ) then { return  NULL};
   else {  new_node = add_after(k,p) ;
           new_node → down = deeper ;}
  return new_node;   }
```

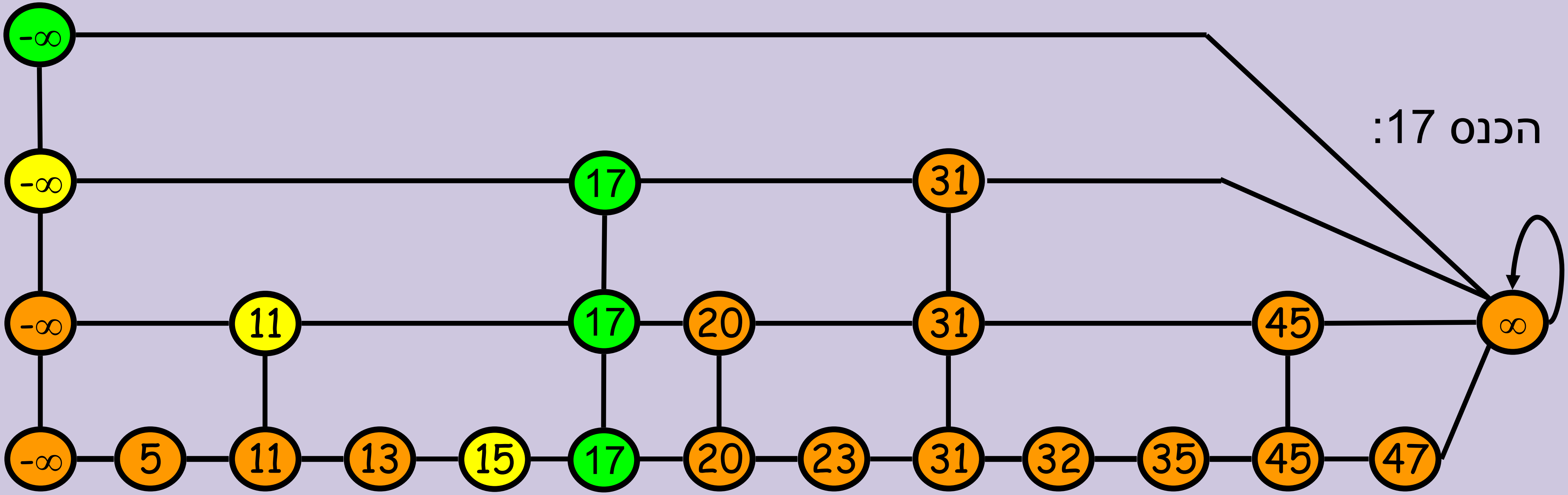
הפרוצדורה add_after(k,p) מוסיפה צומת חדש בעל מפתח k אחרי p, ומחזירה מצביע אליו.

הכנס 17:



הכנסה ברמה העליונה

```
NODE   *insert (KEY k, NODE  *old_list)
{          /* returns a pointer to the added node */
   NODE * deeper ;
   deeper = insert1(k, old_list);
   if (( deeper == NULL || toss( ) ) return  old_list ;
   return   new_node(-INFINITY, inf-addrs, old_list);   }
```

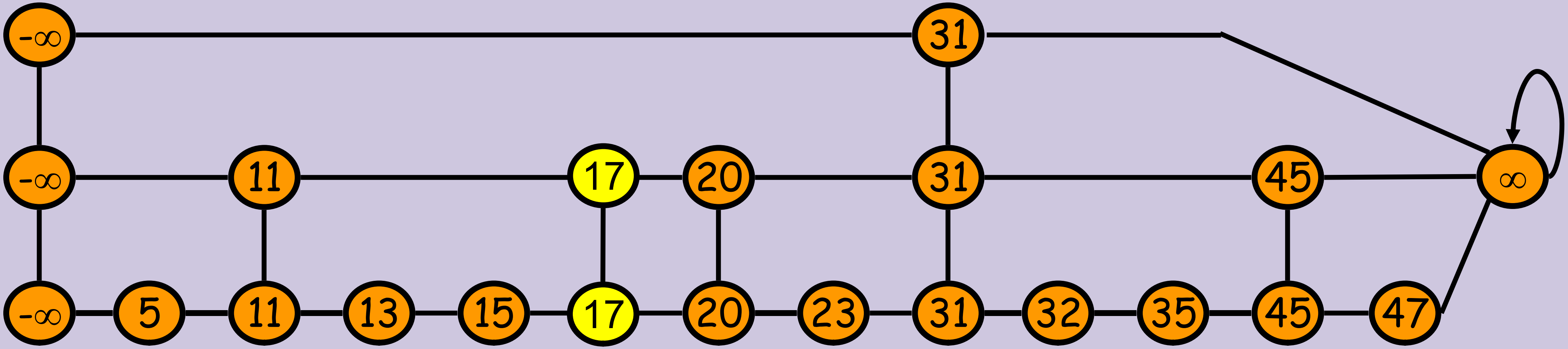


בכמה רמות בממוצע מופיע כל מפתח ?

בכל רמה האלגוריתם מטיל מטבע כדי לקבוע אם יוצר העתק נוסף מעליה.

מספר הרמות הממוצע למפתח שווה למספר הפעמים הממוצע שיש להטיל מטבע הוֹגֵן עד שיצא 1.

מספר הפעמים הממוצע שיש להטיל מטבע הוגן עד שיצא 1 הוא 2. (הוכחה בשקף הבא. אינטואיציה:בממוצע מטבע נופל פעם על צד 0 ופעם על צד 1).



<u>מסקנה (לא פורמלית)</u>: מספר הצמתים הממוצע הוא 2*n* כאשר *n* הוא מספר המפתחות במבנה (כל מפתח מופיע בממוצע פעמיים).

©cs,Technion

מהו אורך מסלול חיפוש ממוצע ?

משפט: האורך הממוצע L של מסלול חיפוש ברשימת דילוגים עם n מפתחות מקיים $L \le 2\log_2(n) + 2$.

הוכחה: נסתכל על מסלול חיפוש מנקודת הסיום אֲחוֹרַנִּית לנקודת ההתחלה. המסלול מורכב מתנועה מעלה (כשאפשר) ותנועה שמאלה (כשחייבים). כאשר מגיעים לקצה השמאלי ביותר, התנועה ממשיכה כלפי מעלה בלבד.

נסמן ב- $c(k)$ את אורך המסלול הממוצע הַמְּטַפֵּס k רמות.

מתקיים: $c(0) = 0$

$$c(k) \le \tfrac{1}{2}(1+ c(k)) + \tfrac{1}{2}(1 + c(k-1))$$

הסבר: צעד שמאלה בהסתברות $\tfrac{1}{2}$ (אם אפשר) וצעד מעלה בהסתברות $\tfrac{1}{2}$.

לפיכך: $c(k) \le 2 + c(k-1)$

ולכן: $c(k) \le 4 + c(k-2) \le \dots \le 2k$

כלומר אורך מסלול ממוצע מרמה 1 לרמה $\log_2(n)$ מקיים: $L \le 2(\log_2(n)-1)$

אורך מסלול חיפוש (המשך)

המשך ההוכחה: נותר לקחת בחשבון את המקרים שבהם מספר הרמות גדול מ- $log_2(n)$.

מעל הרמה ה- $log_2(n)$ מספר הצעדים הממוצע שמאלה חסום ע"י מספר האיברים הממוצע ברשימת הדילוגים שהוכנסו מעל רמה זו. מספרם הממוצע חסום ע"י 2 שכן בכל רמה מספר האיברים הממוצע קטן פי 2.

מהו מספר הצעדים כלפי מעלה?

ניתן להראות שמספר הרמות המקסימלי הממוצע עבור רשימת דילוגים עם n איברים קטן מ- $log_2(n) + 2$

כלומר מספר הצעדים הממוצע כלפי מעלה מעל הרמה $log_2(n)$ הוא 2.

לפיכך סך אורכו הממוצע של מסלול חיפוש מקיים: $L \leq 2(log_2(n)-1)+4$

כמצוין במשפט.

©cs,Technion

זמן בצוע הפעולות

מסקנה מהמשפט: חיפוש, הכנסה, והוצאה מרשימת דילוגים נעשים בזמן ממוצע $O(\log n)$ כיון שבכל צעד על מסלול החיפוש מתבצעים $O(1)$ צעדים.

הטלת מטבע

נניח שלמטבע הסתברות p לצאת "זנב" (נסמן צד זה באפס) והסתברות 1-p לצאת "ראש" (נסמן צד זה באחד).

כמה פעמים בממוצע L יש להטיל את המטבע עד שנקבל "ראש" ?

נסמן ב- Q_i את ההסתברות שנקבל "ראש" לראשונה בהטלה ה- i.

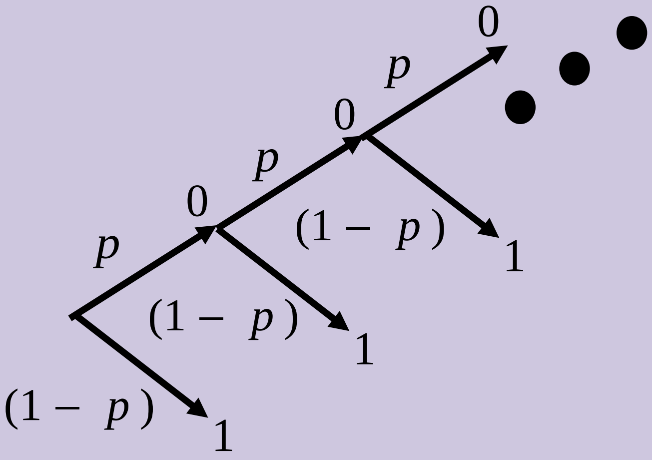
מתקיים: $Q_0=0$ $\quad Q_i = p^{i-1}(1-p)$

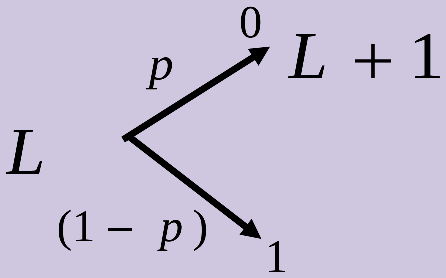
$$L = \sum_{i=1}^{\infty} i \cdot Q_i = \sum_{i=1}^{\infty} i \cdot p^{i-1}(1-p) = (1-p)\sum_{i=1}^{\infty} i \cdot p^{i-1} = \frac{1-p}{(1-p)^2} = \frac{1}{1-p}$$

כלומר בממוצע נדרשות $L = \frac{1}{1-p}$ הטלות עד שמקבלים "ראש".

ועבור מטבע הוגן, כאשר $p = \frac{1}{2}$, נדרשות L=2 הטלות בממוצע.

הוכחה פשוטה:





$$L = p(L+1) + (1-p)1$$

$$L(1-p) = 1 \quad \Rightarrow \quad L = \frac{1}{1-p}$$

©cs,Technion

שימוש במטבע מֻטֶּה

כאשר משתמשים במטבע עם הסתברות לקבלת "זנב" p, אזי מספר ההעתקים הממוצע לכל מפתח הוא $1/(1-p)$ ומספר הרמות הממוצע הוא $\log_{1/p}(n)$.

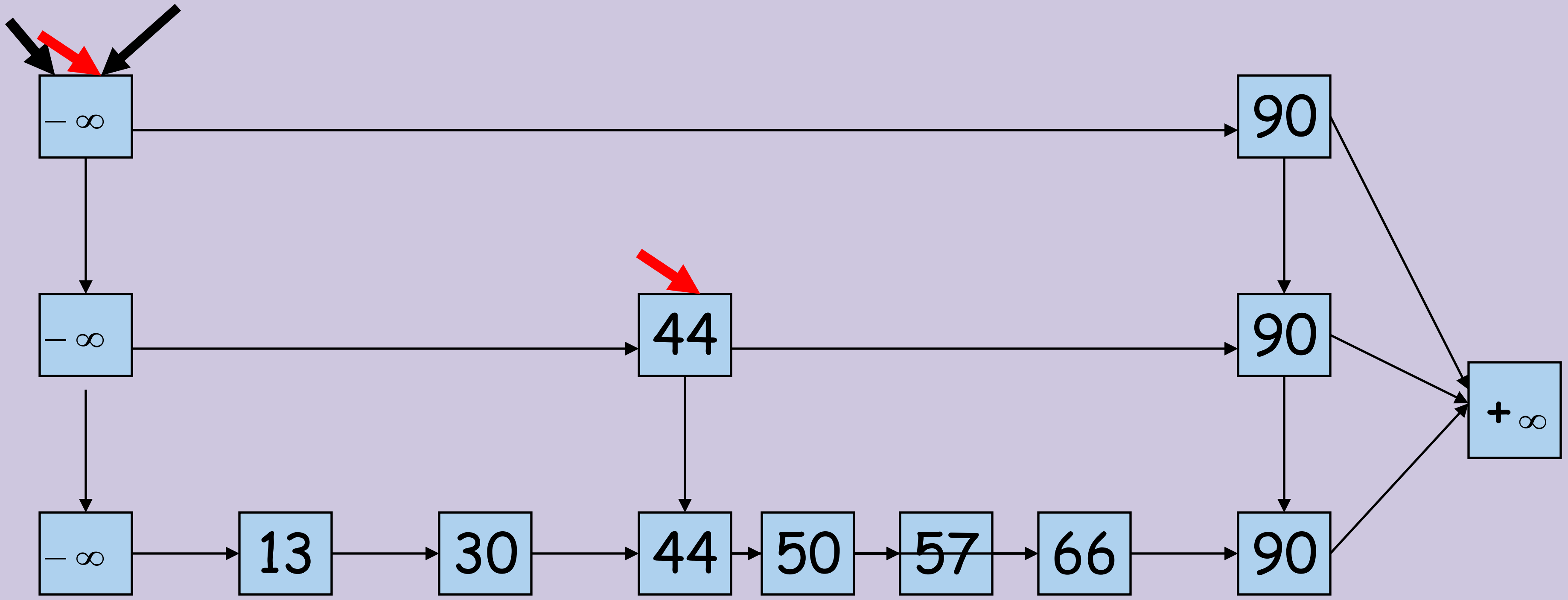
<u>הכללת המשפט</u>: האורך הממוצע L של מסלול חיפוש ברשימת דילוגים עם n מפתחות כאשר למטבע הסתברות p לצאת "זנב" מקיים

$L \le 1/p \cdot \log_{1/p}(n) + 1/p$. (כלומר 2 מוחלף ב- $1/p$).

לפיכך, לפרמטר p אין השפעה גדולה (בגבולות סְבִירִים) על האורך הממוצע של מסלול חיפוש.

אלגוריתם הכנסה לא רקורסיבי ברשימת דילוגים

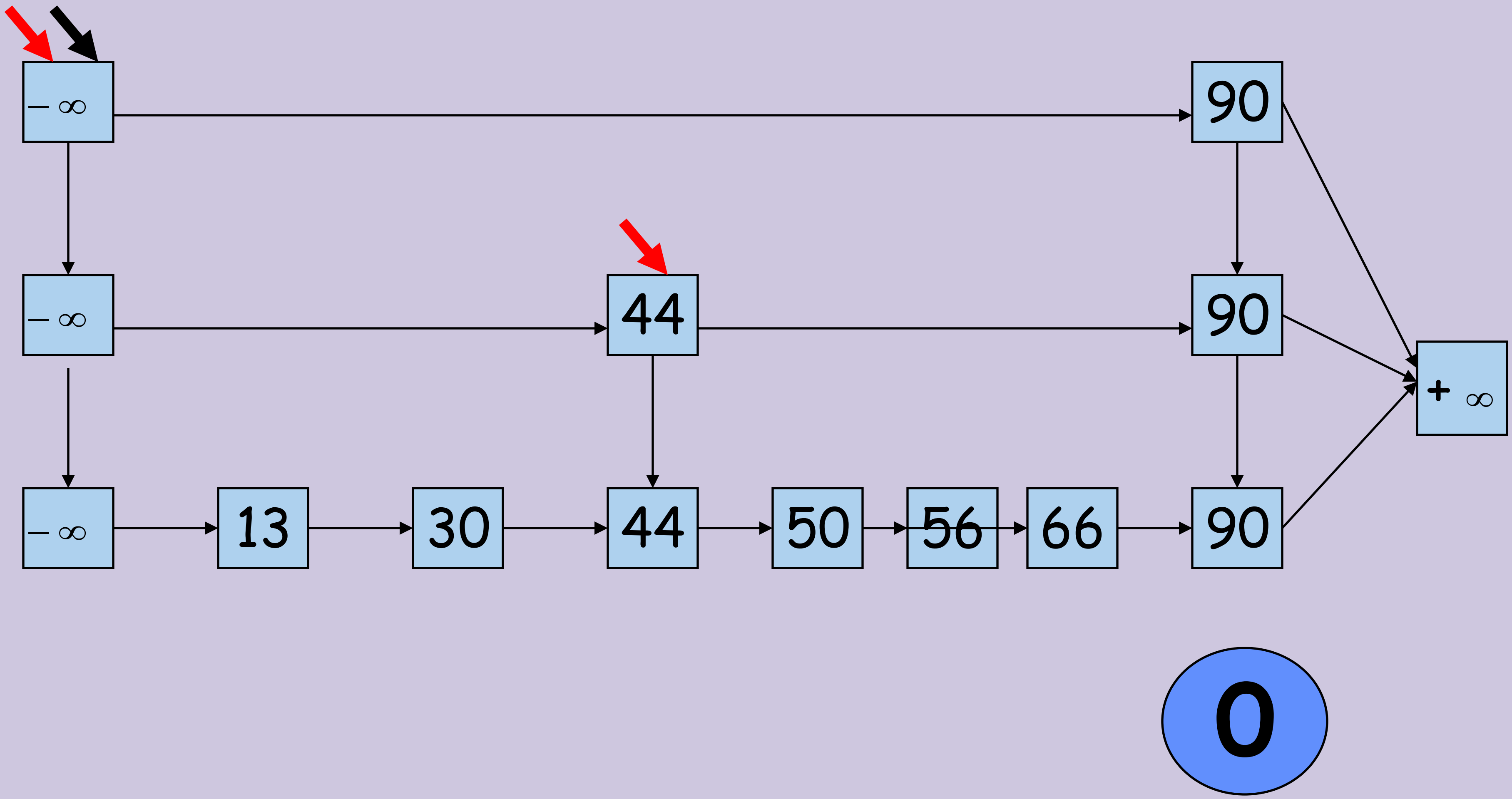
Insert(50)

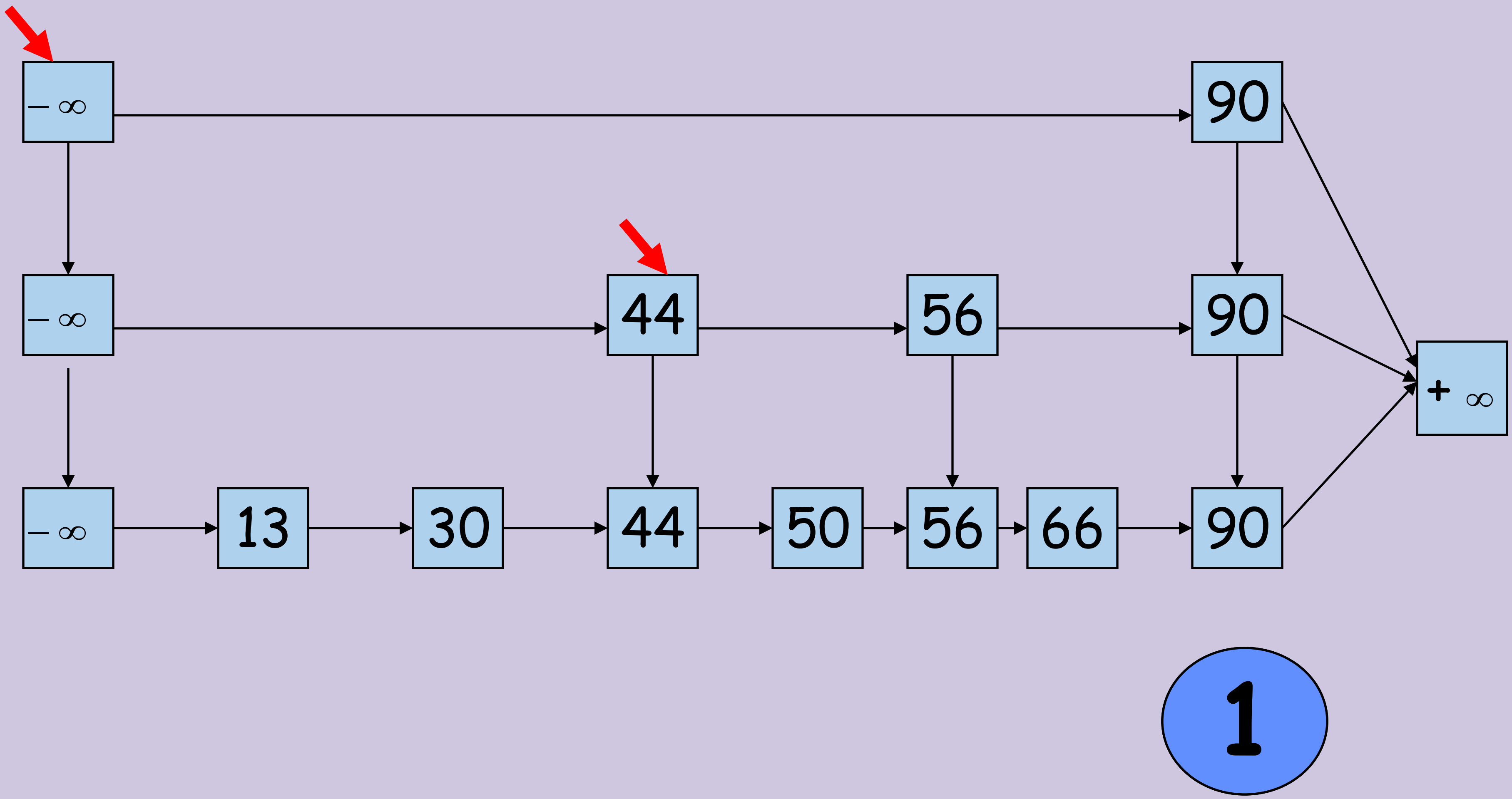


Delete(57)

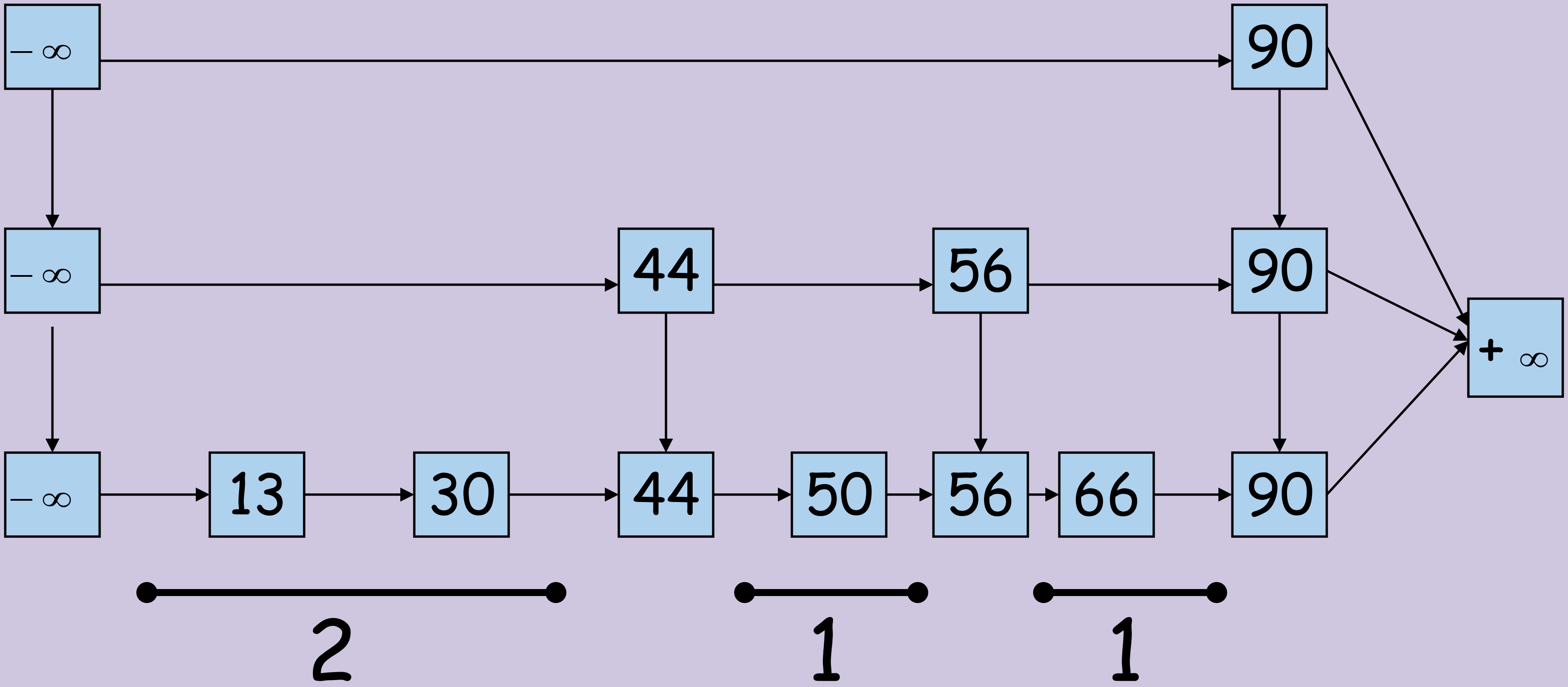
1

Insert(56)

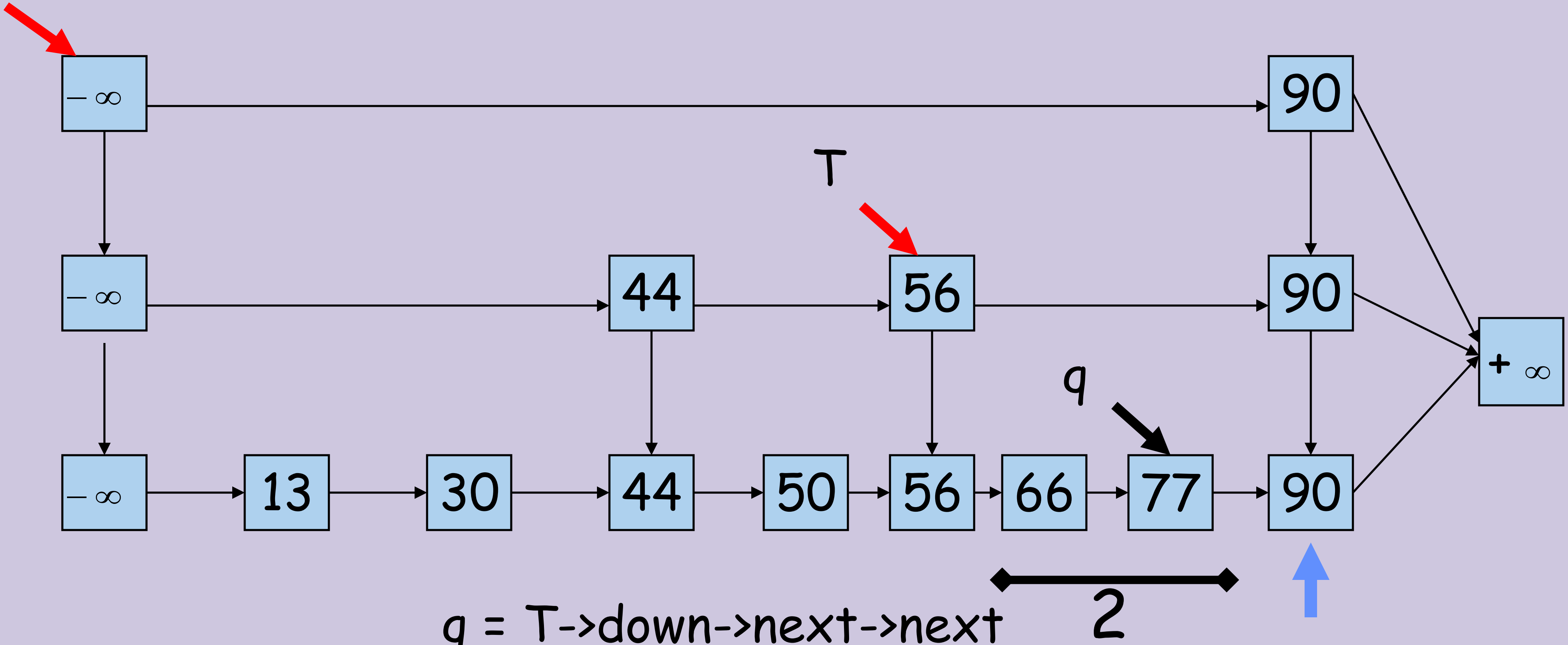



– ∞
90
– ∞
44
56
90
+ ∞
– ∞
13
30
44
50
56
66
90
1

Deterministic Insert and Delete



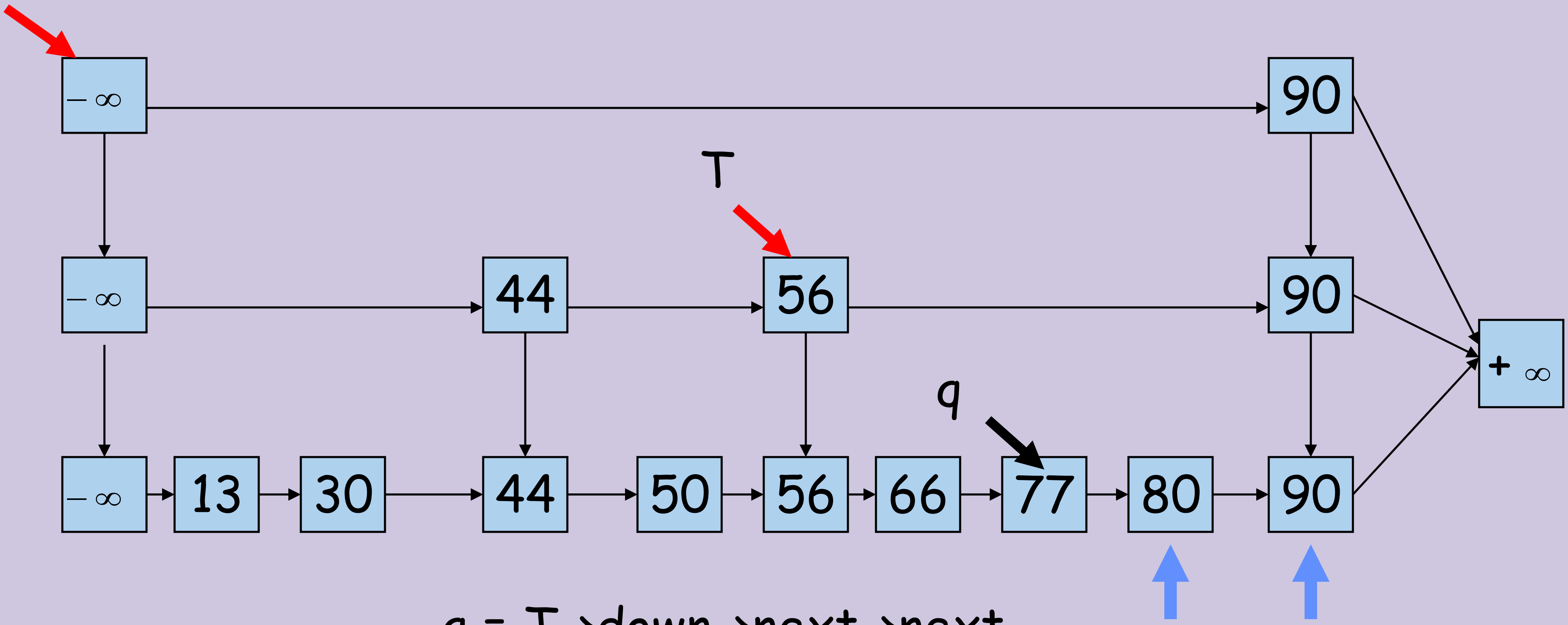
Insert(77)



q = T->down->next->next

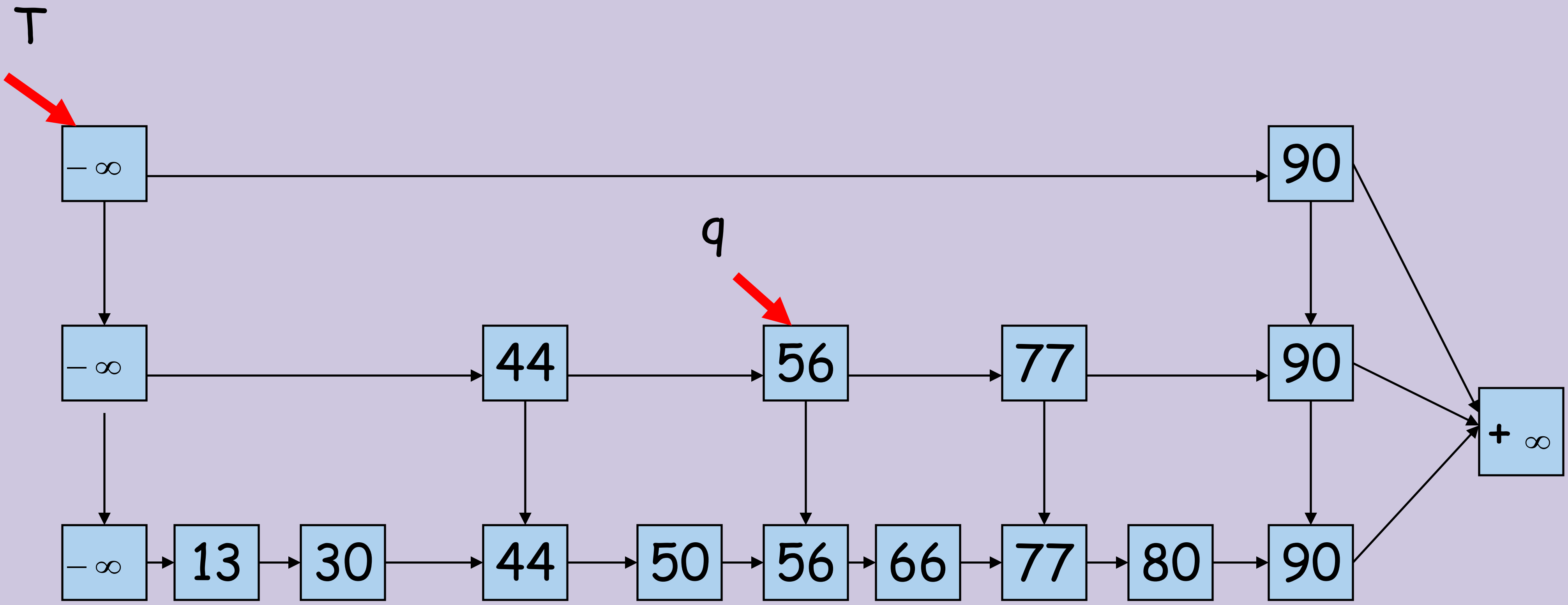
T->next->down is q->next or q

Insert(80)



q = T->down->next->next

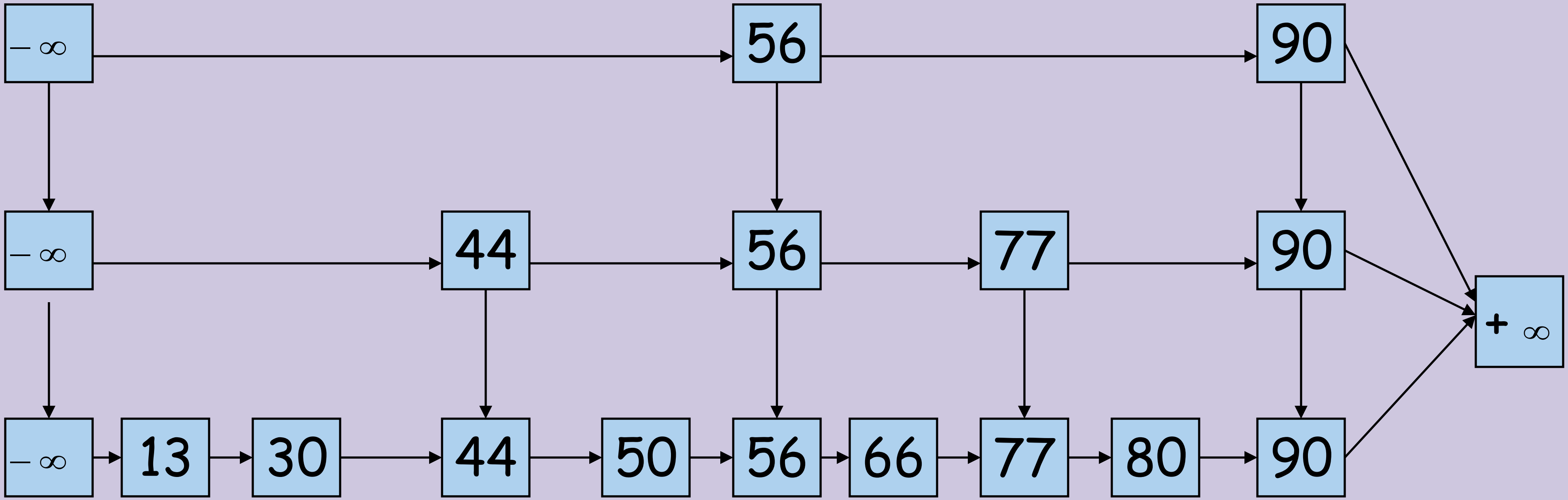
T->next->down is q->next or q



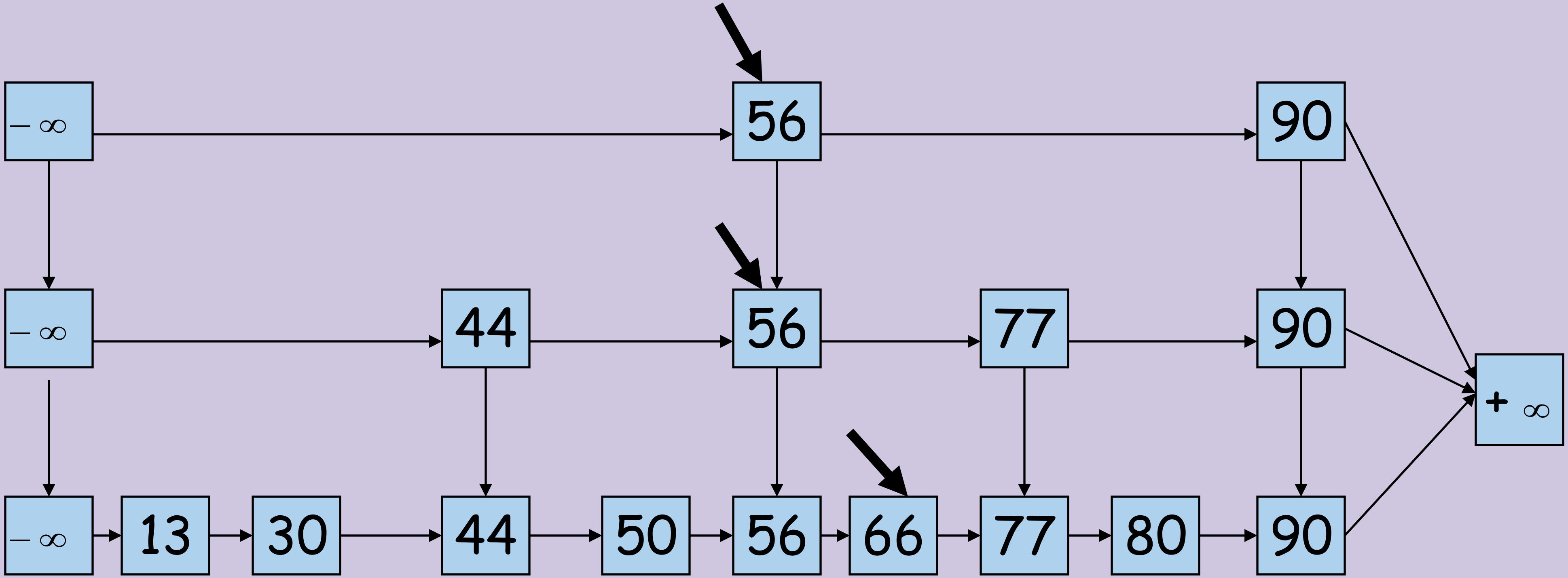
q = T->down->next->next

T->next->down **is** q->next **or** q

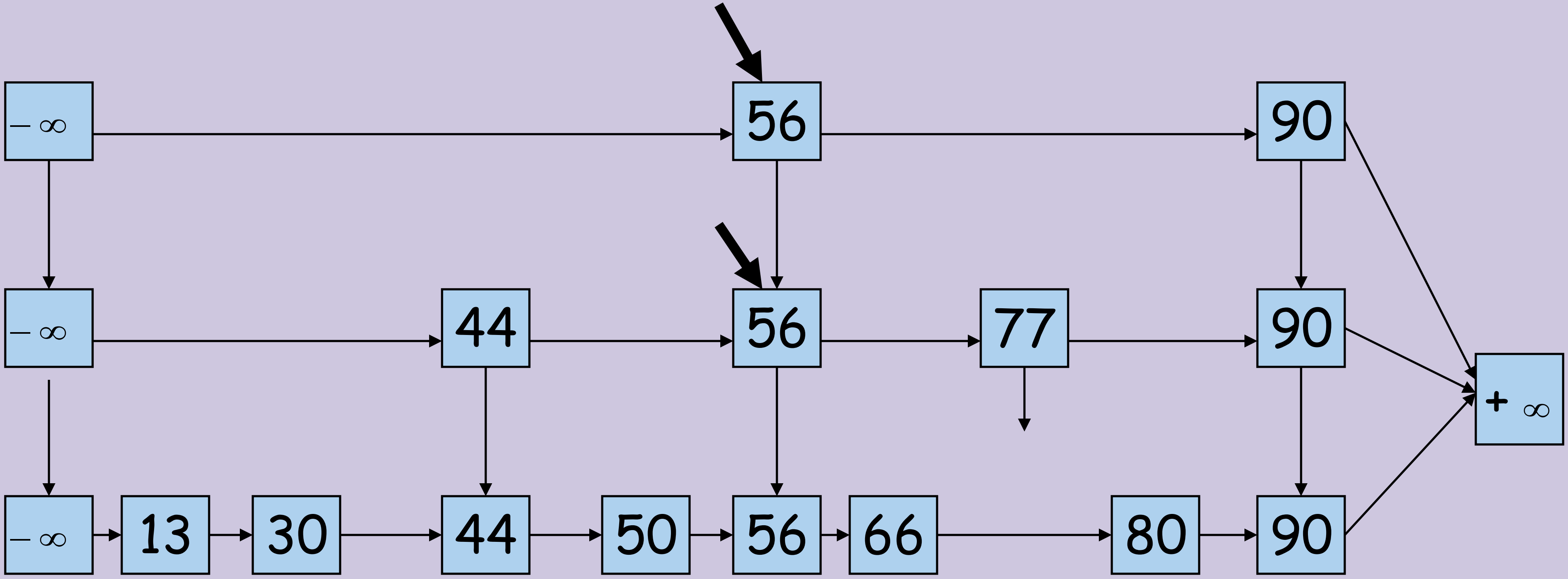
Delete(77)



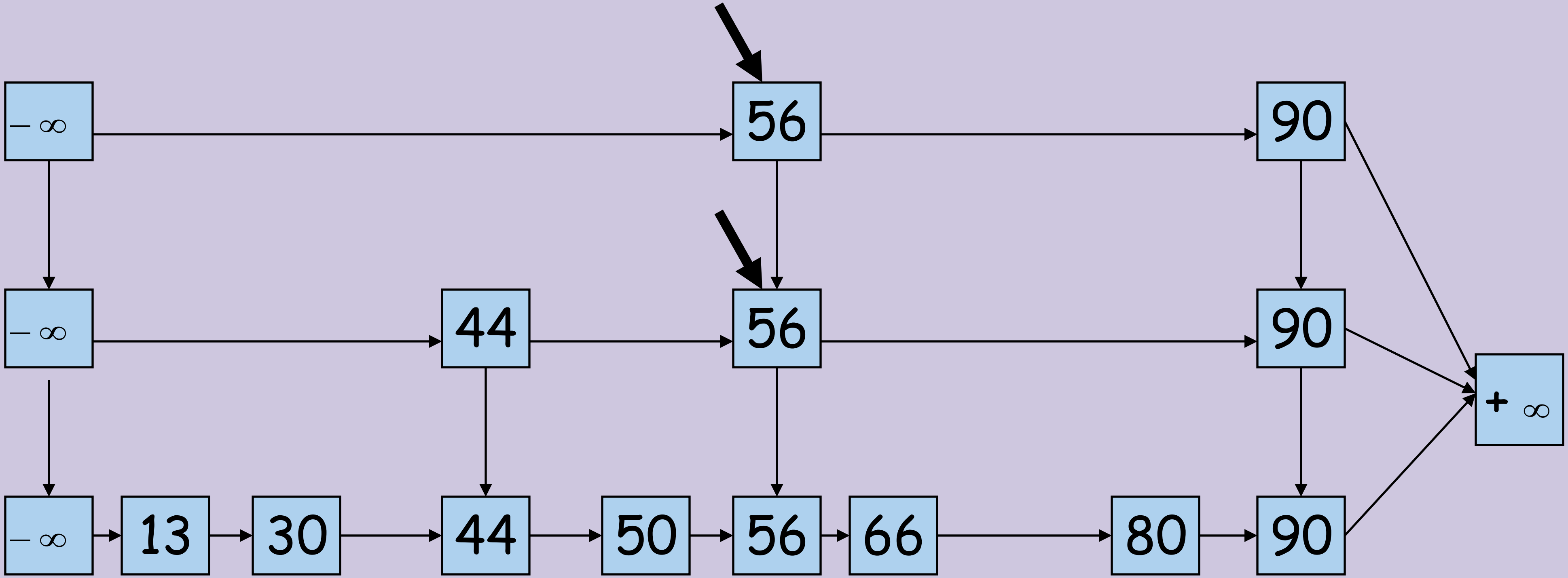
Delete(77)



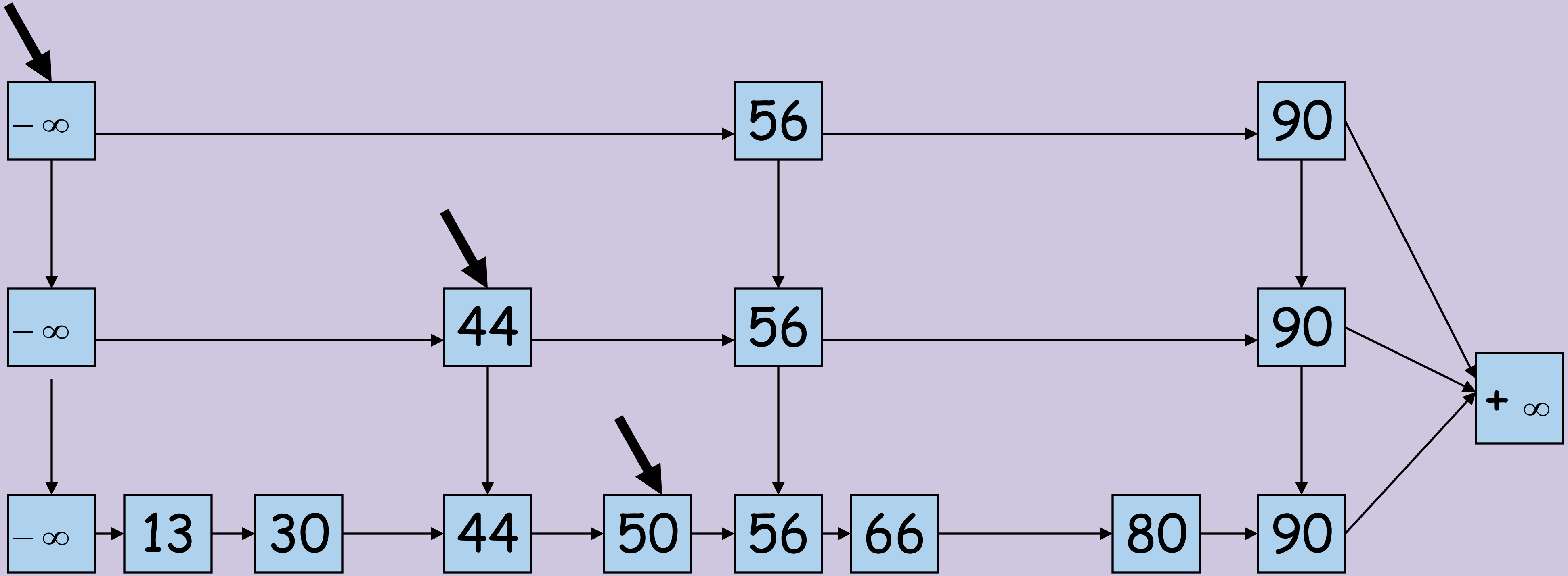
Skip lists



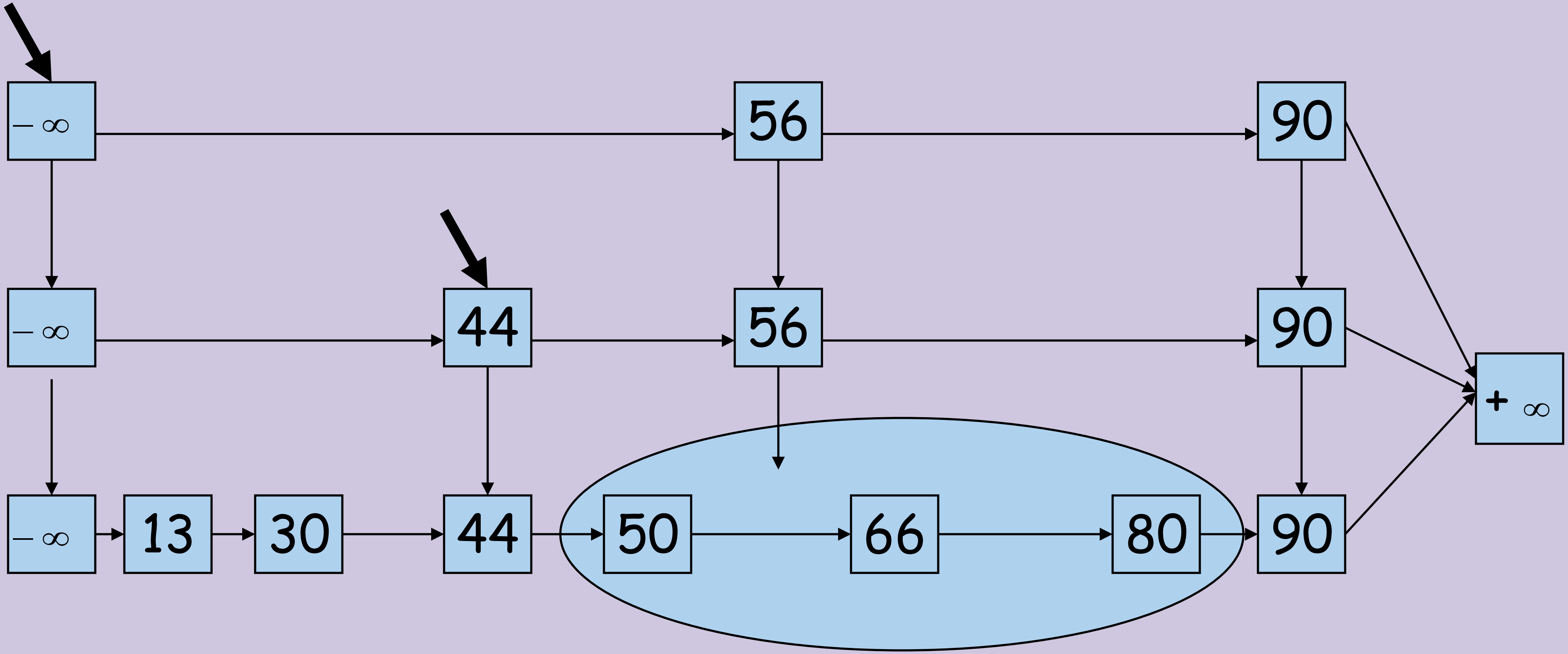
Delete(56)



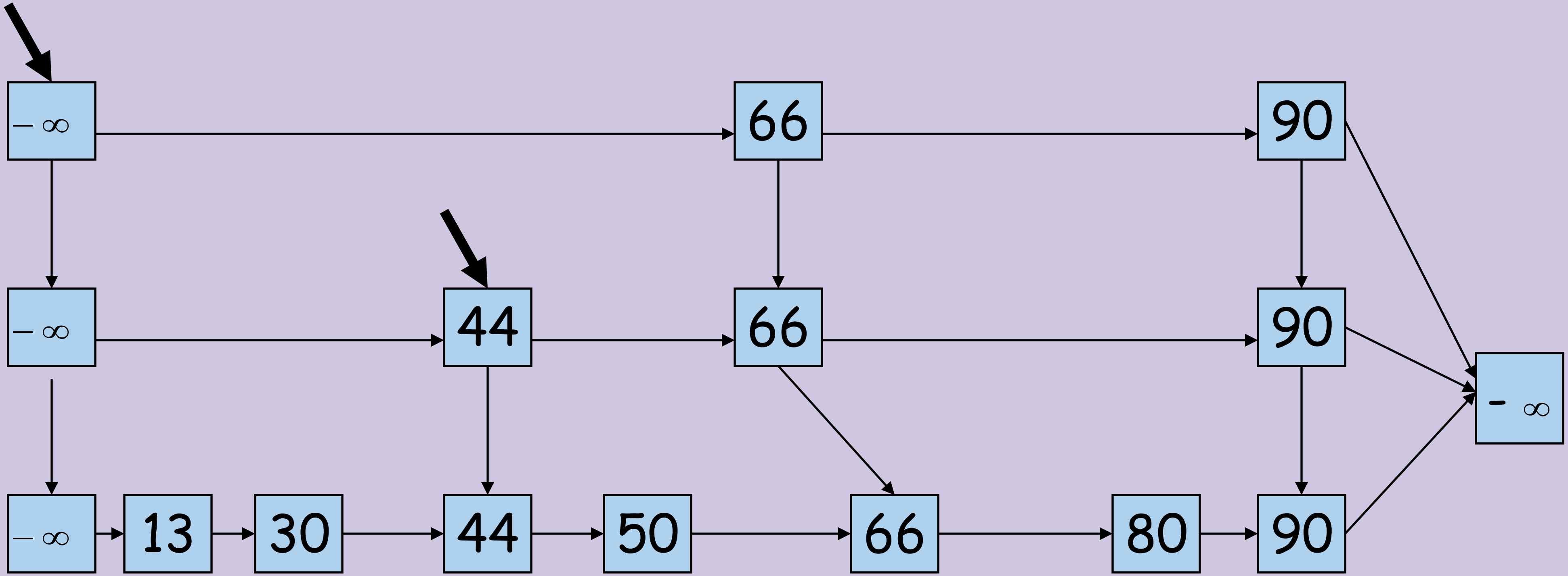
Delete(56)



Skip lists



Skip lists



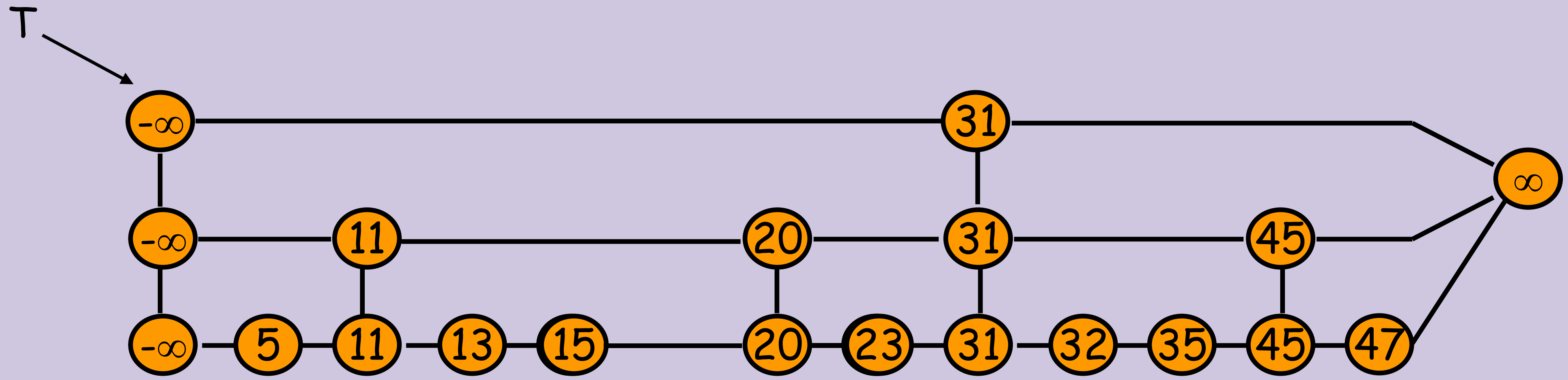
רשימת דילוגים דטרמיניסטית
Deterministic Skip List

נראה כעת כיצד לממש skip list דטרמיניסטי.

מבנה נתונים זה מאפשר פעולות חיפוש/הכנסה/הוצאה כמו בעצים מאוזנים (ובסיבוכיות זמן זהה).

נתאר כעת בפרוטרוט את פעולת ההכנסה.

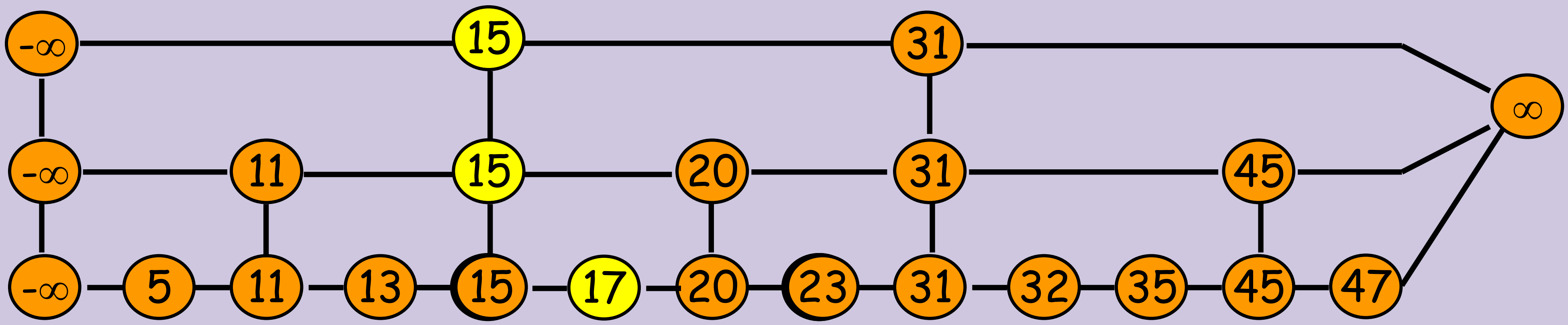
אח"כ נציג את המבנה בהקשר של עצי 2-3.



הרעיון העיקרי

יש לשמור שמספר הצמתים של רמה i בין כל שני צמתים של רמה i-1 יהיה 1 או 2. אם מספר הצמתים הוא 3, מוסיפים צומת ברמה ה- (i-1).

דוגמא: הכנס 17



<u>הערה</u>: בניגוד לרשימת דילוגים (רנדומלית) המפתח שמוכנס אינו בהכרח המפתח שמשוכפל ברמה מעל. בדוגמא הוכנס 17 אך שוכפל המפתח 15.

©cs,Technion

תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

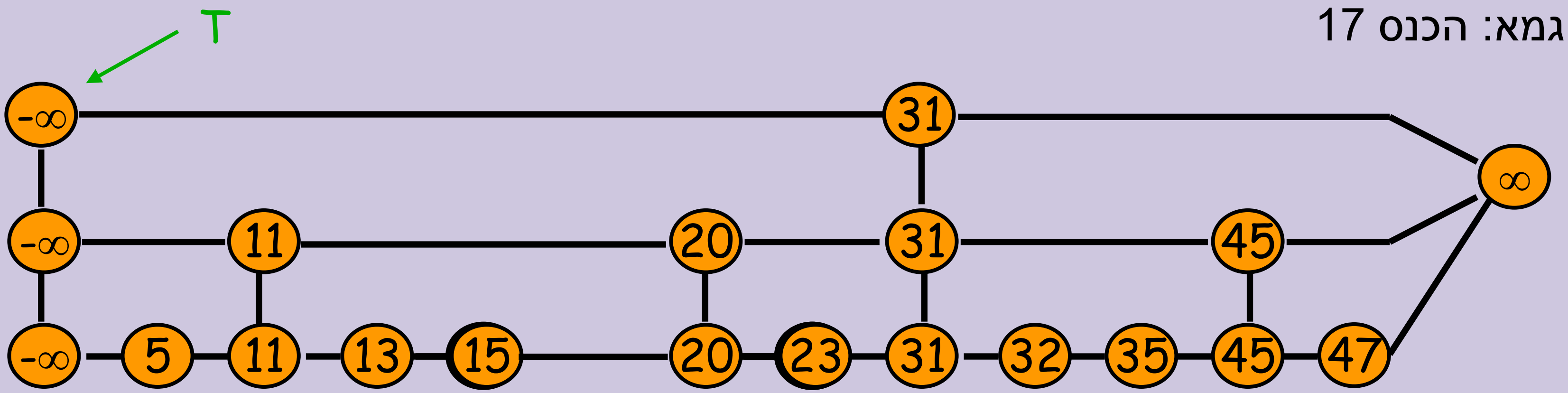
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

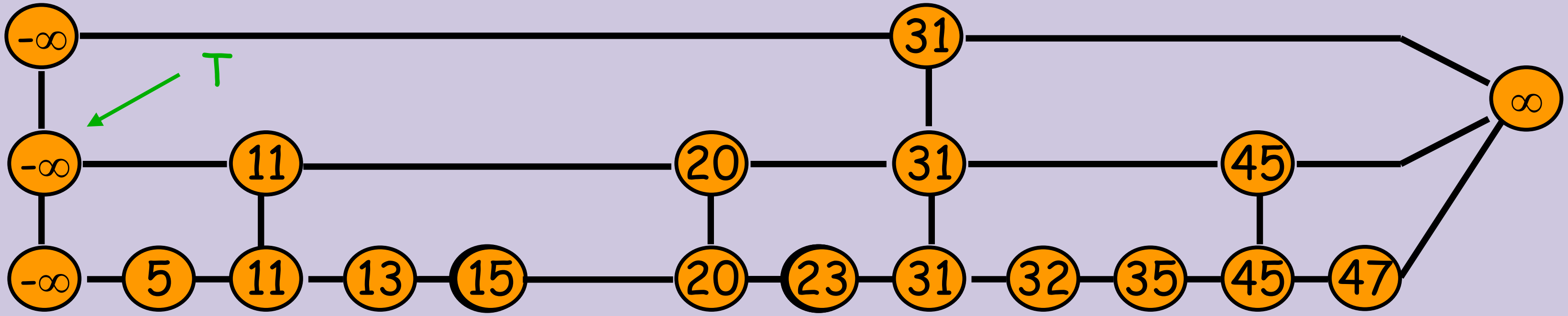
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



©cs,Technion

תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

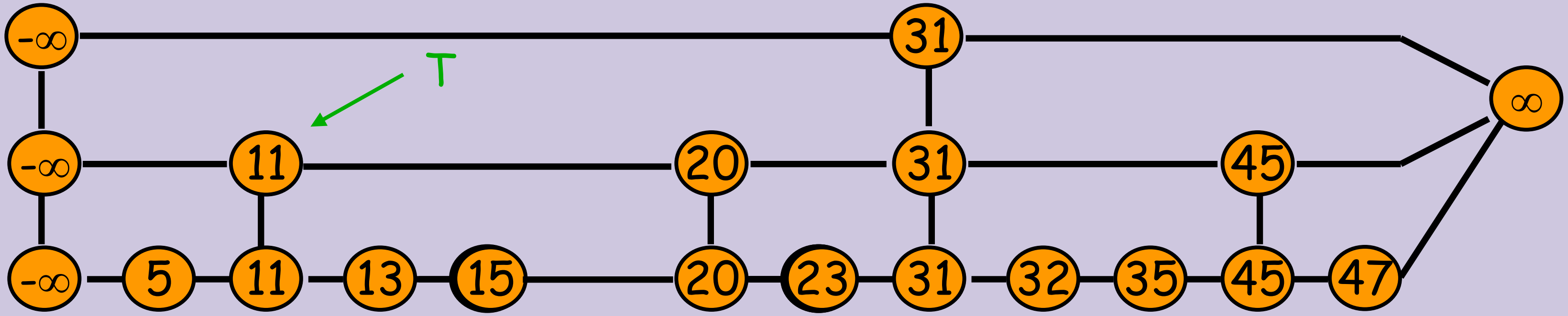
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

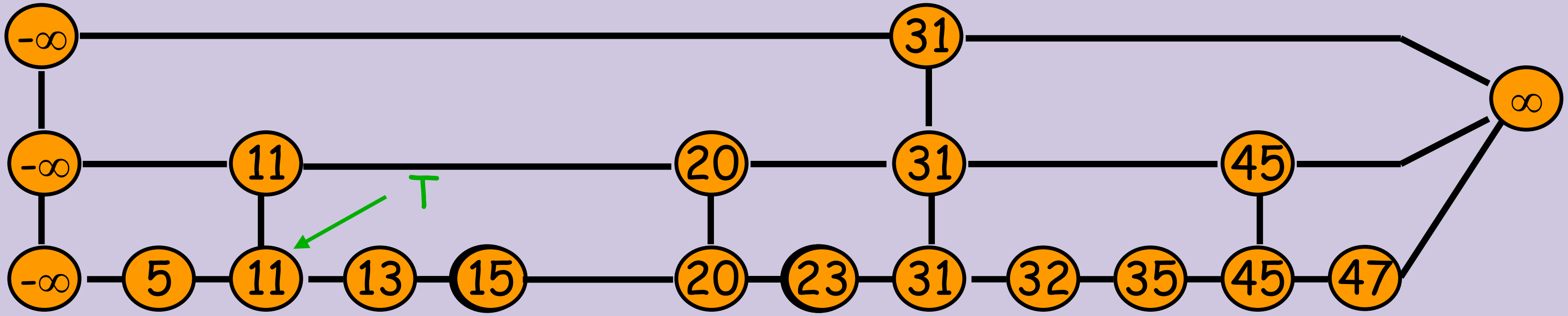
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

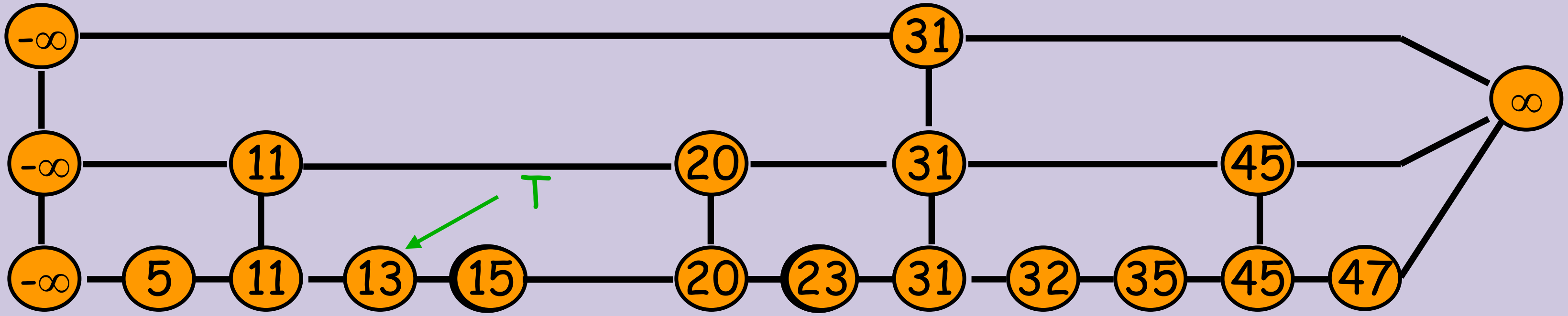
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

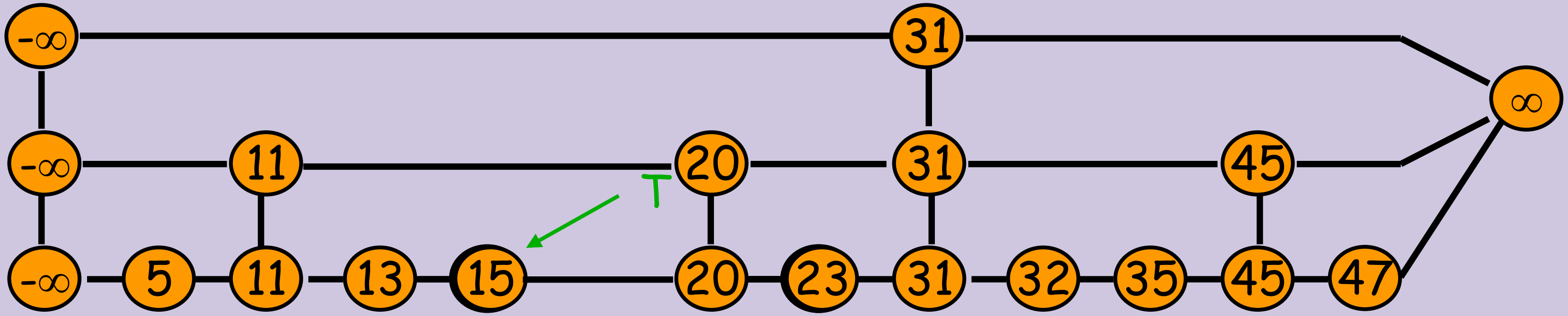
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

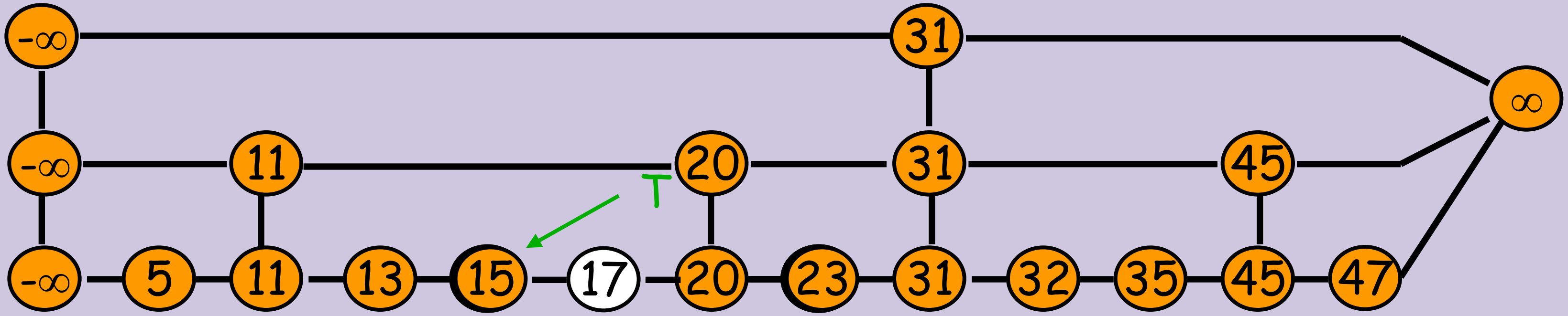
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



©cs,Technion

תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

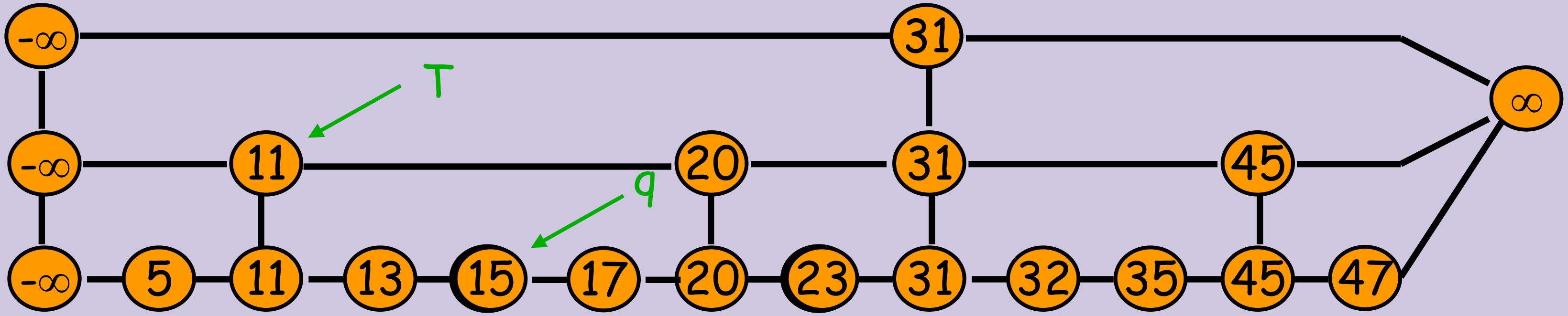
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

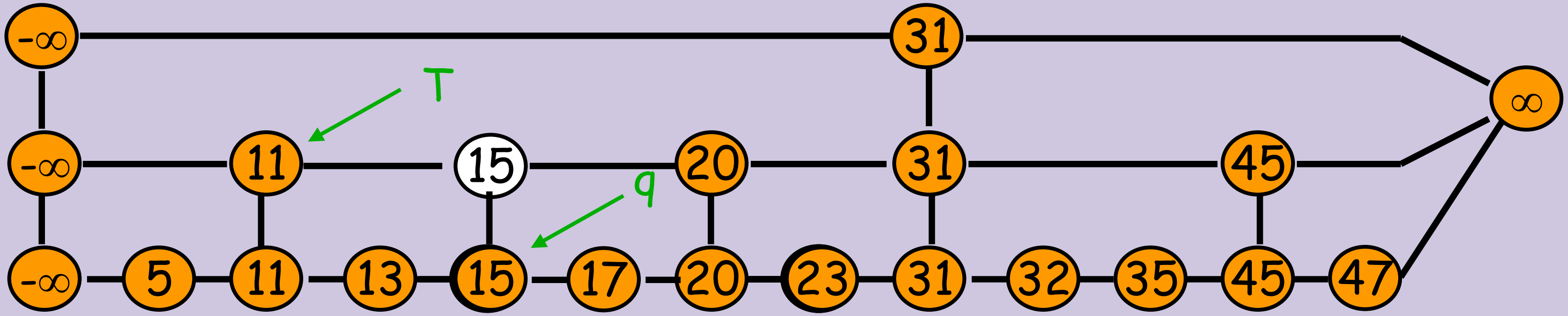
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



©cs,Technion

תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

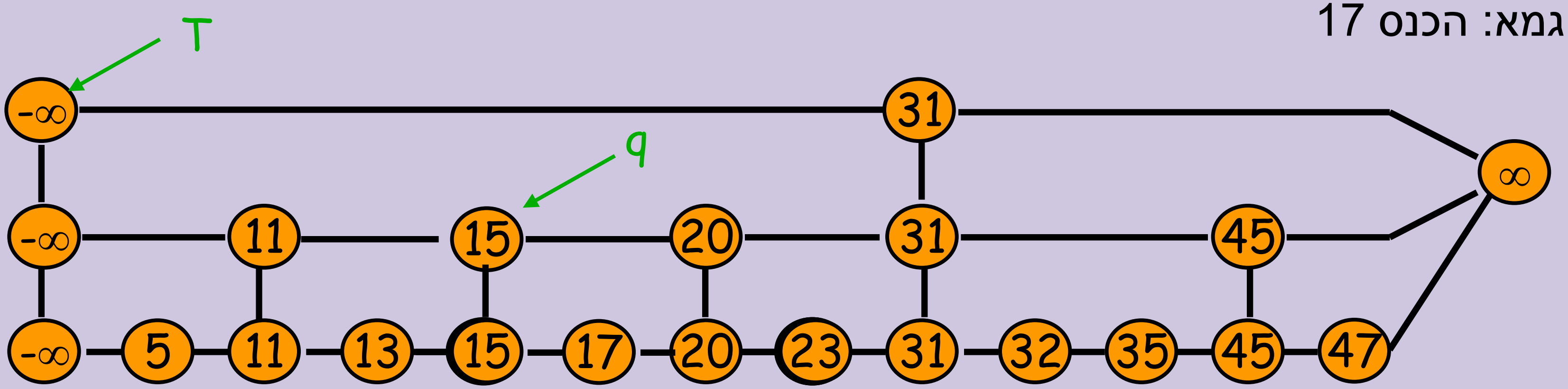
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



©cs,Technion

תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

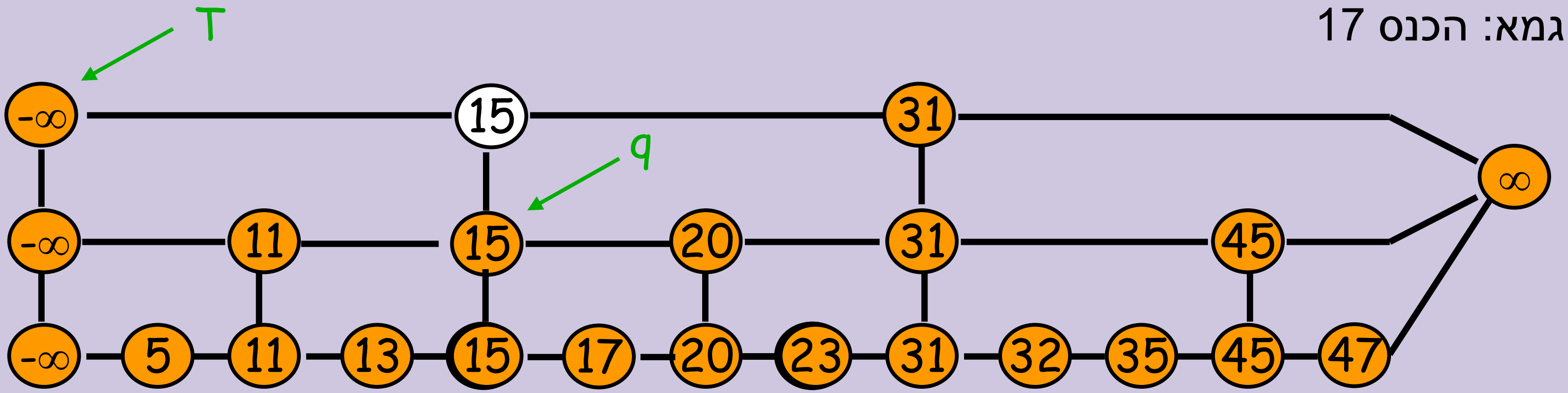
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17



תוכנית להכנסה ברשימת דילוגים דטרמיניסטית

```
int   insert1(int x, NODE  *T)
{ /* returns 1 iff new node allocated on this level */
NODE * q ;
for ( ; T → next →  value  <= x ; T =  T → next) ;
if (T → down = = NULL){     /* a leaf */
            T → next = get_node(x, T → next, NULL);
            return 1;
}
if (!insert1(x, T → down) )  return 0;
/* check if as result of insertion, there are three nodes
 between T → down and T → next → down  */
q = T → down → next → next ;
if ( q → next → value < T → next → value ) {
      T → next = get_node(q → value, T → next, q);
      return 1 ;
}
return 0 ;
}
```

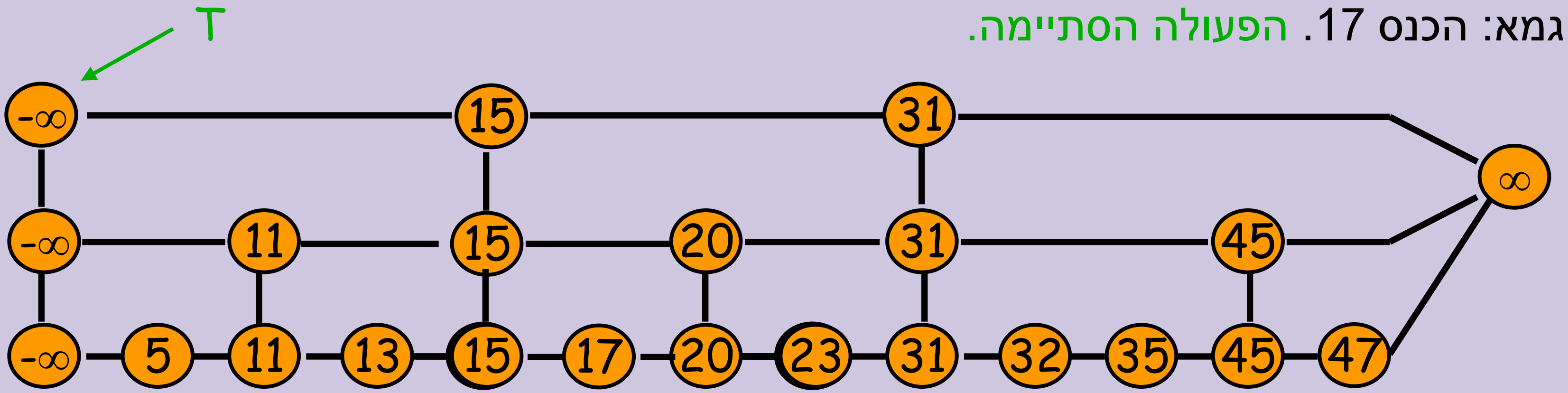
חיפוש ברמה נוכחית.

הוספה ברמה תחתונה.

קריאה רקורסיבית.

חזרה מקריאה רקורסיבית.

דוגמא: הכנס 17. הפעולה הסתיימה.



<u>הערה</u>: במידה והרמה העליונה הכילה שני מפתחות אז יש להוסיף רמה חדשה.

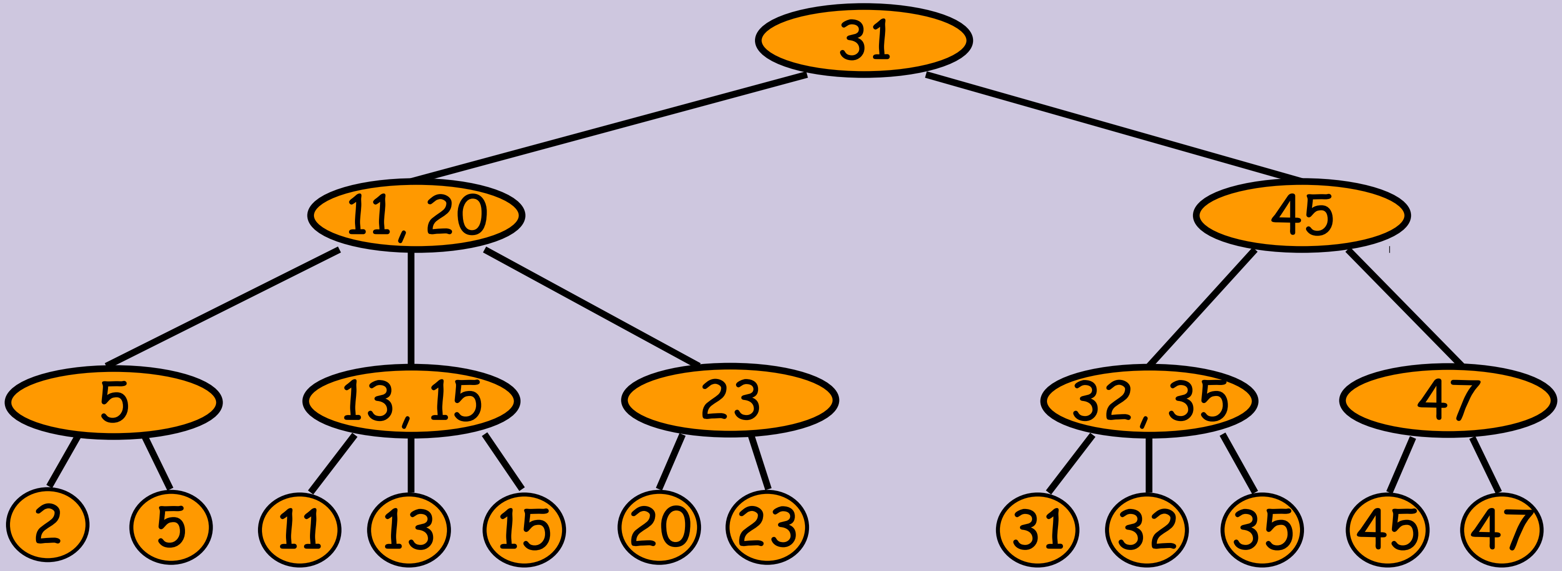
הקשר לעצי 3-2

נראה כעת כיצד עץ 3-2 הופך להיות רשימת דילוגים דטרמיניסטית. הטרנספורמציה שנראה נועדה להדגים את הקשר בין שני מבני הנתונים שלמדנו.

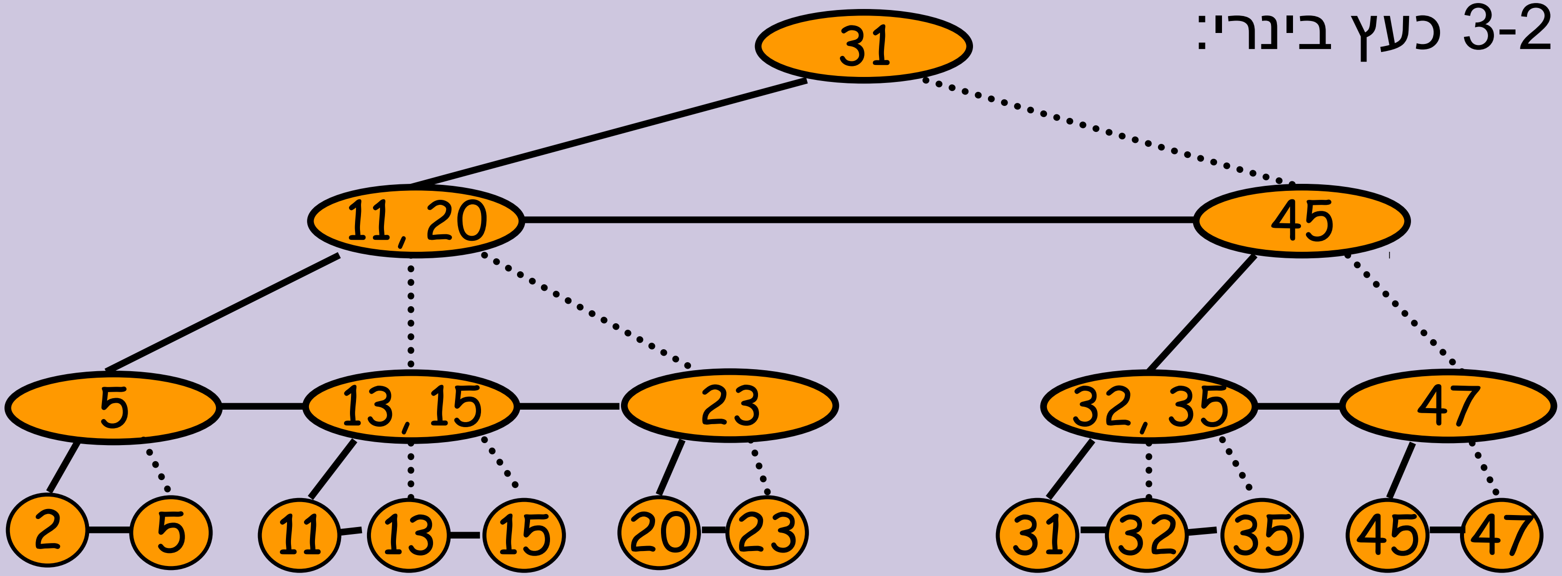
בשום מצב אין צורך לממש טרנספורמציה זו בתוכנית מחשב.

רשימת דילוגים דטרמיניסטית בהקשר לעצי 3-2

עץ 3-2



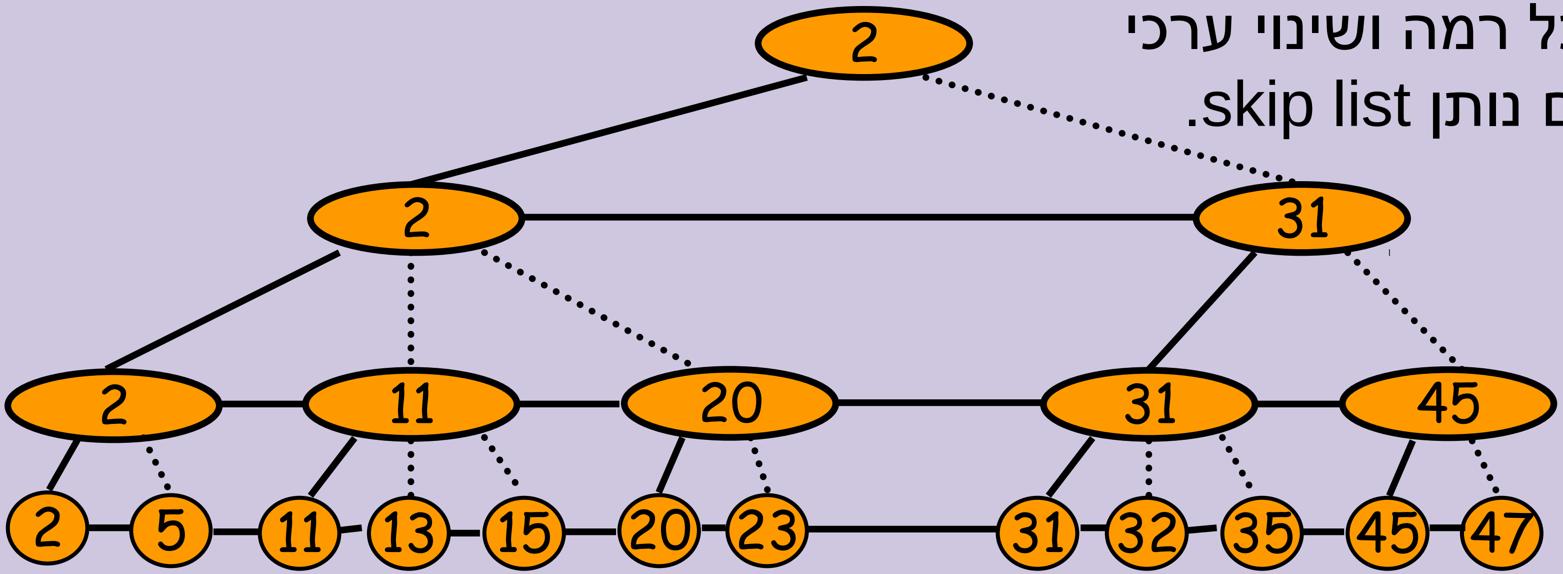
מימוש עץ 3-2 כעץ בינרי:



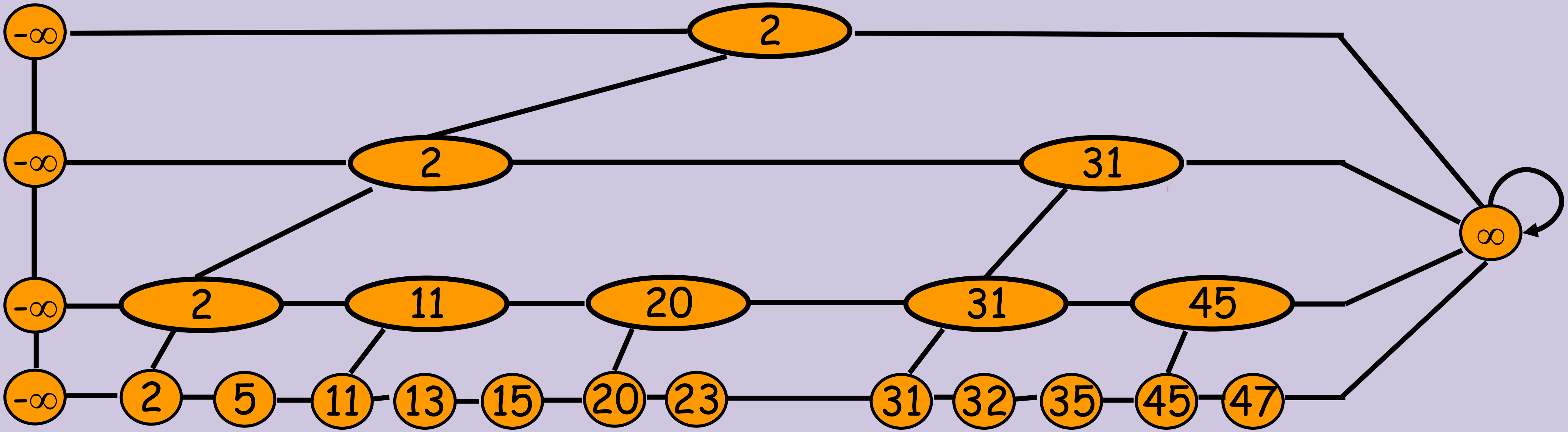
©cs,Technion

רשימת דילוגים דטרמיניסטית (המשך)

חבור הצמתים בכל רמה ושינוי ערכי הצמתים הפנימיים נותן skip list.

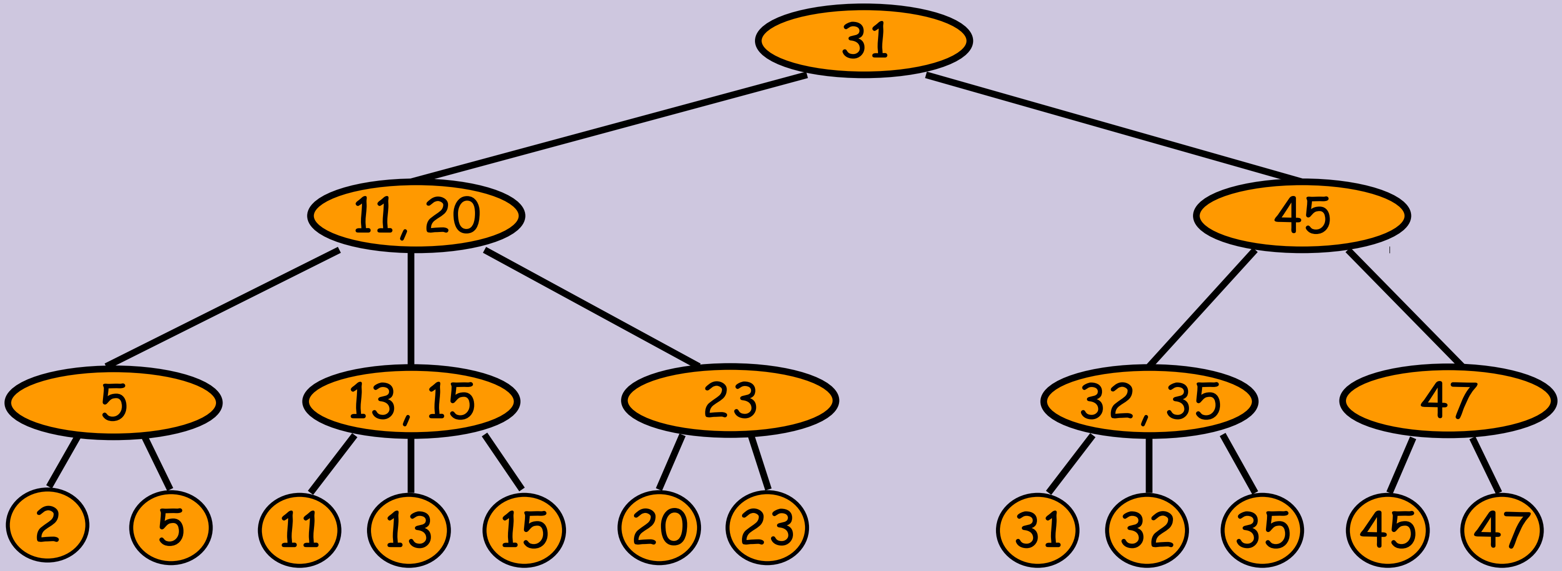


הוספת איבר ראשון ואיבר סופי.

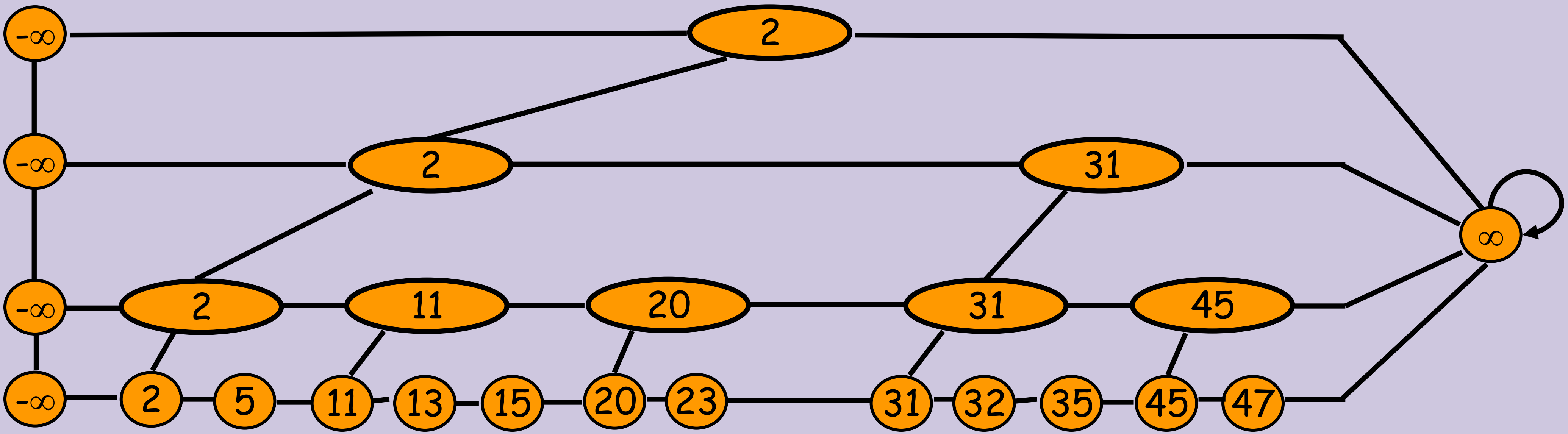


רשימת דילוגים דטרמיניסטית (המשך)

העץ המקורי.



רשימת דילוגים אקוויוולנטית.



Skip lists