תשס"ח סמסטר ב'

תרגול 1

ADT, O() notation, Recursions

מתרגל:

דן פלדמן

מבני נתונים

פרטים טכניים

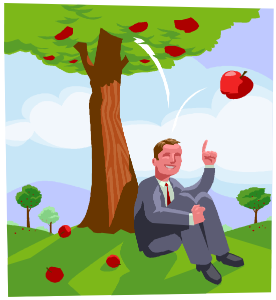
- מתרגל: דן פלדמן
  - שעת קבלה: מתי שרוצים, נא לתאם באי-מייל
  - לאתר הקורס ניתן להגיע דרך:
- http://www.cs.tau.ac.il/~dannyf/
- או לרשום Danny Feldman ב-Google

ADT is an interface

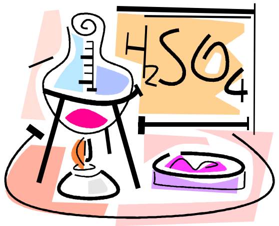
- It defines
  - the type of the data stored
  - operations, what each operation does (not how)
  - parameters of each operation

שיעורי בית

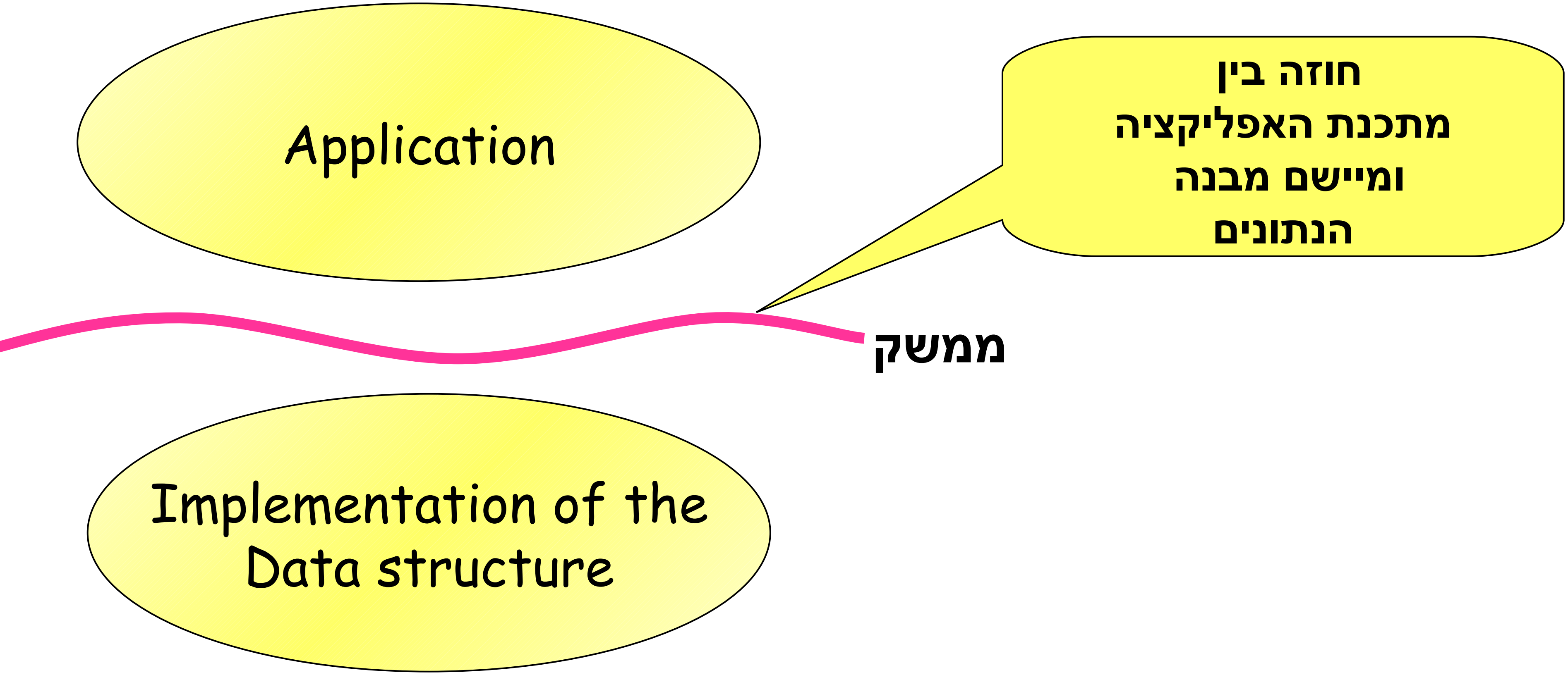
- תרגילים תיאורטיים
  - יינתנו כל שבוע/שבועיים
  - חובת הגשה: 100%
  - הגשה: ישירות לתא הבודק שיפורסם באתר.
  - החזרה: חדר צילומים (קומה 1, שרייבר)
  - איחורים: להגיש בהקדם ולצרף פתק עם סיפור ואישורים אם יש.
  - **לא** לשלוח מייל לגבי הגשה באיחור
  - עדכונים ברשימת התפוצה של הקורס. בידקו שאתם מקבלים דואר!



- פרוייקטים
  - יינתנו 1-2 פרוייקטים במהלך הסמסטר
  - חובת הגשה: 100%
  - הפרוייקטים ייעשו ביחידים



ADT


Application
חוזה בין
מתכנת האפליקציה
ומיישם מבנה
הנתונים
ממשק
Implementation of the
Data structure

פסאודו-קוד

- בקורס זה נכתוב אלגוריתמים ב-psuedo-code
- זהו תיאור קומפקטי ולא רשמי של אלגוריתם במדעי המחשב
- נשמיט פרטים טכניים ולא חשובים ונשמור על העיקר

i ← 5	Assign the value 5 to i
while i>0 do	Begin a loop, condition is i>0
i ← i-1	Decrease value of i by 1
end while	End the loop

Example: Stacks

- Push(x,S) : Insert element x into S
- Pop(S) : Delete the last element inserted into S
- Empty?(S): Return yes if S is empty
- Top(S): Return the last element inserted into S
- Size(S)
- Make-stack()

Implementation

- We will be interested in algorithms to implement the ADT..
- And their efficiency..

Big-O

$T(n) = O(f(n))$ - קיים c ו-n_0 כך ש:

$$T(n) \leq c \cdot f(n) \quad \forall n > n_0$$

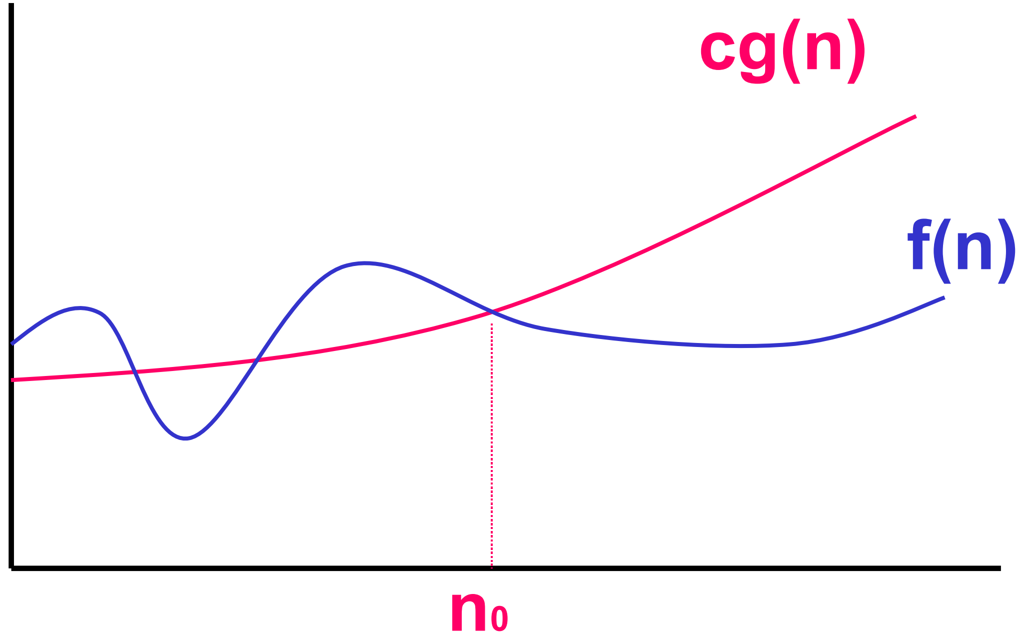
דוגמא:

$$T(n) = (n+1)^2$$

$$(n+1)^2 \leq 4n^2 \Rightarrow O(n^2)$$

Big-O

$$f(n) = O(g(n))$$



More examples

- $4n$ □ $O(n^2)$
- $4n^2$ □ $O(n^2)$
- 2^n □ $O(n^{10})$
- 10 □ $O(1)$
- $100n^3+10n$ □ $O(n^3)$
- $\log_2(n)$ □ $O(\log_{10}(n))$

תרגיל

- בהינתן n מספרים שלמים בטווח $1..n^c$ כיצד ניתן למיין בזמן $O(n\cdot c)$
- פתרון
  - נתחיל ממספרים בטווח $1..n^2$ האם ניתן למיין אותם ב$O(n)$?
  - אילו אלגוריתמים אנו מכירים שממיינים בזמן לינארי?
    - Counting sort – בהינתן n מספרים בטווח 0..k. $\theta(k+n)$

$$k = O(n)$$

  - נכתוב כל מספר בבסיס n, כמה ספרות?
  - נמיין כמו radix sort ,$O(n)$ לכל ספרה ולכן $2O(n)=O(n)$
  - עבור הבעיה המקורית כמה ספרות יהיו לנו?

רקורסיה

- נקבל ביטוי מהצורה $T(n) = aT(g(n)) + h(n)$
- נרצה למצוא f(n) כך ש- $T(n) = \Theta(f(n))$
- n מספר שלם
- תמיד נחשב עד תנאי עצירה מסוים
- בקורס הזה נתעניין בהתנהגות האסימפטוטית של T ולא בחישוב מדויק

תרגיל 1

- פתרו את הרקורסיה הבאה: $T(n) = 9T(n/7) + n$
- פתרון Brute Force 1: $T(n) = \Theta(f(n))$

$$T(n) = \quad 9T(n/7) + n =$$

$$9(9T(n/49) + n/7) + n = 81T(n/49) + \frac{9n}{7} + n =$$

$$81(9T(n/7^3) + n/49) + \frac{9n}{7} + n = 9^3T(n/7^3) + 9^2\, n/7^2 + 9n/7 + n =$$

$$\ldots$$

$$9^{k+1}T(n/7^{k+1}) + n\underbrace{(1 + \frac{9}{7} + \left(\frac{9}{7}\right)^2 + \ldots\left(9/7\right)^k)}_{\to\ \infty}$$

תרגיל 1

- מה עשינו לא נכון?
  - אי אפשר להמשיך עד אינסוף, צריך לעצור ב-T(1) או קבוע אחר
  - באיזה k נעצור? $\frac{n}{7^{k_0+1}} = 1 \Rightarrow 7^{k_0+1} = n \Rightarrow k_0 = \log_7 n$
  - זהו **עומק הרקורסיה** (ההתפצלות מדרגה 7)
  - ונחזור לביטוי...

$$T(n) = 9^{\log_7 n+1} \cdot T\left(n/7^{\log_7 n+1}\right) + n\left(1 + \frac{9}{7} + \left(\frac{9}{7}\right)^2 + \ldots \left(9/7\right)^{\log_7 n}\right)$$

$$= \Theta\left(9^{\log_7 n} + n\left(9/7\right)^{\log_7 n}\right) = \Theta\left(n^{\log_7 9}\right)$$

Master Theorem – רקורסיה

- "מתכון" לפתור רקורסיות מסוג מסוים
- עבור נוסחת רקורסיה כך ש $T(n) = aT(n/b) + f(n)$
- קיימים שלושה מקרים

$$1.\ \exists \varepsilon > 0: \quad f(n) = O(n^{\log_b a - \varepsilon}) \quad \P \quad T(n) = \Theta(n^{\log_b a})$$

$$2. \quad f(n) = \Theta(n^{\log_b a}) \quad \square \quad T(n) = \Theta(n^{\log_b a} \log n)$$

$$3.\ \exists \varepsilon > 0, c < 1: \quad f(n) = O(n^{\log_b a + \varepsilon}), \quad af(n/b) \ \square\ cf(n) \quad \square \quad T(n) = \Theta(f(n))$$

חזרה לתרגיל 1

$$T(n) = 9T(n/7) + n$$

- a=9;b=7;f(n)=n
- האם אנחנו במקרה הראשון?

$$\exists \varepsilon : n = O(n^{\log_7 9 - \varepsilon})$$

$\log_7 9$ □1.2

ε �?0.1 □ $n \quad n^{1.1}$

$$T(n) = \Theta(n^{\log_7 9})$$

דוגמה נוספת

$$T(n) = 2T(\frac{n}{2}) + 10n$$

$$a = 2, b = 2, f(n) = 10n, \log_b a = \log_2 2 = 1$$

$n^{\log_b a - \varepsilon} = n^{1-\varepsilon}$ ⇨ $10n = O(n^{1-\varepsilon})?$

$n^{\log_b a} = n$ ⇨ $10n = \Theta(n)?$

מסקנה: זהו מקרה 2

תרגיל 3

- פתרו את הרקורסיה הבאה: $T(n) = 2T(\frac{n}{2}) + n\log^2 n$
- פתרון
  - בואו ננסה לפתוח את זה קצת...

$$T(n) = n\log^2 n + n\log^2(\frac{n}{2}) + n\log^2(\frac{n}{4}) + n\log^2(\frac{n}{8}) + \ldots$$

  - נמצא חסם משני הצדדים

תרגיל 3

- חסם תחתון
  - בואו נשמור רק את חצי האברים הגדולים ביותר
  - מיהו האבר הקטן ביותר ששמרנו?

$$\frac{n}{2^{\log n/2}} = n \cdot \sqrt{\frac{1}{2^{\log n}}} = n \cdot \sqrt{\frac{1}{n}} = \frac{n}{\sqrt{n}} = \sqrt{n}$$

  - לכן אם נחליף את n/k בשורש n נקבל...

$$T(n) \geq \frac{\log n}{2} n \log^2 \sqrt{n} = \frac{\log n}{2} n \left(\frac{1}{2}\log n\right)^2 = \frac{n\log^3 n}{8}$$

$$T(n) = \Omega(n\log^3 n)$$

תרגיל 3

- חסם עליון
  - נחליף כל n/k ב-n ונקבל

$$T(n) \le n\log^2 n + n\log^2(n) + n\log^2(n) + n\log^2(n) + \ldots$$

$$= \log n \cdot n\log^2(n) = n\log^3 n$$

$$T(n) = O(n\log^3 n)$$

הסוף

שבוע נעים

תשס"ח סמסטר ב'

תרגול 2

Sorting, Compatison Model, and Lower Bounds

מתרגל:

דן פלדמן

מבני נתונים

פרוט תרגיל 1 משיעור קודם

$$9^{\log_7 n} + n\left(\frac{9}{7}\right)^{\log_7 n} = 9^{\log_7 n} + n\left(\frac{9^{\log_7 n}}{7^{\log_7 n}}\right)$$

$$= 2\cdot 9^{\log_7 n} = 2\cdot 9^{\log_9 n \cdot \log_7 9} = 2\cdot n^{\log_7 9}$$

מיונים

- Counting Sort
  - ממיין n מפתחות של שלמים בין 0 ל-n בזמן לינארי.
  - זאת בעזרת שיבוץ המפתחות למערך בגודל n+1.
- Radix Sort
  - מיון יציב לפי ספרת האחדות, עשרות וכו' עד הספרה האחרונה (MSB - Most Significant Bit)
  - אפשר גם רקורסיבית מה-MSB ל-LSB
  - זמן: O(nk) כש-k זה האורך המקסימלי של מספר
- טוב גם למיון לקסיקורפי וכדומה

תרגיל 1 להיום

- כיצד ניתן למיין בזמן $O(n)$, n מספרים שלמים בטווח $1..n^2$?
- פתרון:
  - נהפוך כל מספר לבסיס n בזמן $O(n)$. נקבל n מספרים בין 2 ספרות.
  - נפעיל Radix Sort בזמן $O(n)$
- עבור מספרים בתחום $1..n^c$ נמיין בזמן $O(nc)$

מודל ההשוואות

- הגישה היחידה לקלט היא בעזרת פונקציה Comapre(index i, index j)
- מחזירה בזמן O(1) תשובה בוליאנית
- רוב האלגוריתמים שנלמד עובדים תחת הנחות מודל זה
- יעיל גם למיונים וחיפושים תחת אילוצים שונים

תרגיל

- נתונים n מטבעות זהב שאחת מהן מזוייפת (שוקלת פחות).
  - כתבו אלגוריתם שמוצא אותה בכמה שפחות שקילות מאזניים (אסימפטוטית) במקרה הגרוע (WC).
  - מצאו חסם עליון ותחתון לזמן הריצה של האלגוריתם.
  - מצאו חסם עליון ותחתון לכמות השקילות ההכרחיות WC לפיתרון הבעיה.
  - הוכיחו שאין אלגוריתם יעיל יותר (אסימפטוטית) WC

תשובה 1

- עבור n זוגי: נשים חצי מהמטבעות בכל מאזן, ונמשיך רקורסיבית עם החצי הקל.

- עבור n אי זוגי: נשים מטבע בצד ואת השאר נחלק לשניים. נמשיך רקורסיבית עם החצי הקל. במקרה של שיוויון - המטבע המזוייף הוא המטבע שהוצאנו.

חסמים לזמן ריצה של האלגוריתם

- חסם עליון: $T(n)=1+T(n/2)=O(\log n)$
- חסם תחתון: $O(1)$ – למשל, אם המזוייפת היתה המטבע שהוצאנו, באחת מ-$O(1)$ השקילות הראשונות
- חסם הדוק : אין.
- אם n הוא חזקה של 2: נבצע $\Omega(\log n)$ שקילות.
- עבור n כזה יש חסם הדוק לכמות ההשוואות של האלגוריתם: $\Theta(\log n)$

הוכחת חסמים לזמן ריצה של אלגוריתם

- חסם עליון: הוכחה לזמן ריצה מקסימלי עבור כל קלט. למשל: נוסחאות נסיגה, פונקציית פוטנציאל
- חסם תחתון: קלט לדוגמא עבורו האלגוריתם לוקח הרבה זמן.

- נניח שהוכחנו שאלגוריתם מסויים פותר את הבעיה לכל קלט בזמן $O(t)$
- אזי $O(t)$ הוא חסם עליון לזמן הנדרש לפתור את הבעיה WC

מצאנו חסם עליון לבעיה

- הוכחנו ש:

קיים אלגוריתם שפותר את הבעיה לכל קלט בגודל n,

בזמן O(logn).

כדי למצוא חסם תחתון לבעיה יש להוכיח ש:

לכל אלגוריתם שפותר את הבעיה קיים קלט שעבורו זמן הריצה הוא Ω(logn).

אבחנות

- לעץ עם n עלים ומס' בנים קבוע לכל צומת, יש גובה $\Omega(\log n)$
  - כי: בסידרה הנדסית עם מנה קבועה, שבה האיבר הראשון הוא 1 והסכום הוא n, יש $\Omega(\log n)$ איברים.
- כל אלגוריתם שפותר את הבעיה ניתן להצגה כעץ החלטות
- בעץ יש לפחות n עלים, שמתאימים ל-n פלטים אפשריים
- לכל צומת בעץ ההחלטות יש מס' קבוע של בנים

מצאנו חסם תחתון לבעיה

- לכל אלגוריתם שפותר את הבעיה קיים קלט שעבורו זמן הריצה הוא $\Omega(\log n)$
- מצאנו חסם עליון לפתרון הבעיה, וכיוון שהוא שווה לחסם התחתון – זהו פיתרון הדוק.
- זה לא אומר שאין אלגוריתם שפותר את הבעיה בזמן $O(1)$ עבור קלטים מסויימים

חסמי זמן ריצה לבעיה

- חסם עליון $O(t)$: אלגוריתם לדוגמא שפותר את הבעיה
לכל קלט בזמן $O(t)$
- חסם תחתון $\Omega(t)$:
  - מיפוי כל אלגוריתם לעץ החלטות בגובה $\Omega(t)$
  - רדוקציה לבעיה שהוכחנו בכיתה עם חסם תחתון על זמן הריצה
(מיון או חיפוש בינארי במודל ההשוואות).
  - זמן קריאת הקלט. למשל: כל אלגוריתם לבעיית מציאת
המינימום מ-n מספרים חייב לעבור על כל הקלט ולכן לוקח זמן
$\Omega(n)$

חסם תחתון על חיפוש במודל ההשוואות

- מה החסם התחתון על חיפוש במערך ממוין?
  - במקרה הגרוע ביותר נשווה לכל איבר
  - ז"א בעץ יש n עלים
  - מה העומק המינימלי של עץ בינארי בעל n עלים?
    - זמן לוגריתמי O(logn)

פתק למבחן

- נניח שקיים אלגוריתם שפותר את הבעיה בזמן $O(f(n))$.
- נציג אלגוריתם A שממיין n איברים כלשהם בזמן WC $o(nlogn)$ במודל ההשוואות.
- זוהי סתירה למה שהוכחנו בכיתה. לכן לא קיים אלגוריתם שפותר את הבעיה בזמן $O(f(n))$ WC. מ.ש.ל.
- האלגוריתם A:
  - קלט: מערך של n מספרים כלשהם.
  - פלט: מערך ממויין של המספרים
  - מימוש: ...

- נתונים n אלמנטים מתחום סדור במערך A. ידוע כי קיים אינדקס j כך שלכל $i \le j$ מתקיים $A[i]<A[i+1]$ ולכל $i>j$ מתקיים $A[i+1]<A[i]$.

דוגמה:

j=6

1	2	4	5	6	9	**26**	15	14	3

תארו אלגוריתם יעיל למציאת j.

- נתאר אלגוריתם הפועל בסיבוכיות O(logn).
- האלגוריתם מאוד דומה לחיפוש בינארי במערך ממוין.
- נסתכל על האבר ה-n/2. אם זהו האבר המתאים אזי סיימנו (גדול משני שכניו), אחרת אם הוא גדול משכנו הימני וקטן משכנו השמאלי, נחזור על הפעולה בחצי השמאלי של המערך רקורסיבית (על n/2 אברים).
- אחרת נחזור על הפעולה בחצי הימני של המערך רקורסיבית.

- בכל שלב אנו מקטינים את מס' האברים הנבדקים פי 2, ולכן כפי שראינו פעמים רבות הסיבוכיות לוגריתמית.
- טעויות נפוצות: לחפש בכל שלב את החציון ולא את התא האמצעי. חציון ניתן למצוא רק בזמן לינארי ולכן זה לא מתאים!!

- הוכיחו כי $\Omega(\log n)$ הוא חסם תחתון על מספר ההשוואות הנדרש למציאת j.

- אופציה א': נבנה עץ השוואות, סה"כ יש n אפשרויות עבור המיקום של j, ולכן בעץ צריכים להיות לפחות n עלים. בכל צומת בעץ ההשוואות אנו יכולים להשוות רק בין שני אברים ולכן אם העץ מאוזן עומקו הוא לוגריתמי

- אופציה ב': רדוקציה לחיפוש בינארי. נניח כי ניתן למצוא את j בזמן o(logn) (ממש קטן מ-logn). אזי בהינתן מערך ממוין A ואבר x אותו אנו מחפשים נפעיל אלגוריתם למציאת j , כאשר נתייחס לכל אבר גדול מ-x כאל מספר שלילי. כך לדוגמה נתייחס למערך 1,3,5,7,9,15 כאשר x=7 כאל המערך 1,3,5,7,-9,-15. כעת נפעיל את האלג' ונקבל את המיקום של x. אך אנו יודעים שהחסם התחתון על חיפוש בינארי הינו לוגריתמי, וזאת בסתירה להנחה.

- הערות בדיקה: היו כאן כמה טעויות חוזרות
- הוכיחו שהאלגוריתם בסעיף א' עובד בזמן לוגריתמי במקרה הגרוע ביותר (נוסחת רקורסיה). זה היה צריך להיכתב בסעיף א' ולא כאן.
- ניסו לעשות רדוקציה למיון של מערך. זה התחלק לשתי טעויות עיקריות
  - הניחו שA מערך כמו שמתואר והראו שניתן למיינו בזמן ממש קטן מ- (nlogn), אבל מערך כמו שמתואר ניתן למיין בקלות בזמן לינארי (למצוא את j ואז למזג את שני תתי המערכים)
  - הניחו שA מערך כללי, אבל אז לא ניתן להפעיל את הפונקציה, היא מסתמכת על כך שלA מבנה מאוד מסוים!

חסם תחתון על מיון על ידי השוואות

- אם אלגוריתם המיון שלנו מתבצע רק ע"י השוואות, כמה סידורים אפשריים יש ל-n אברים?
  - יש n! סידורים אפשריים

- For <a,b,c>
  - a,b,c
  - a,c,b
  - b,a,c
  - b,c,a
  - c,a,b
  - c,b,a

n!

חסם תחתון על מיון על ידי השוואות

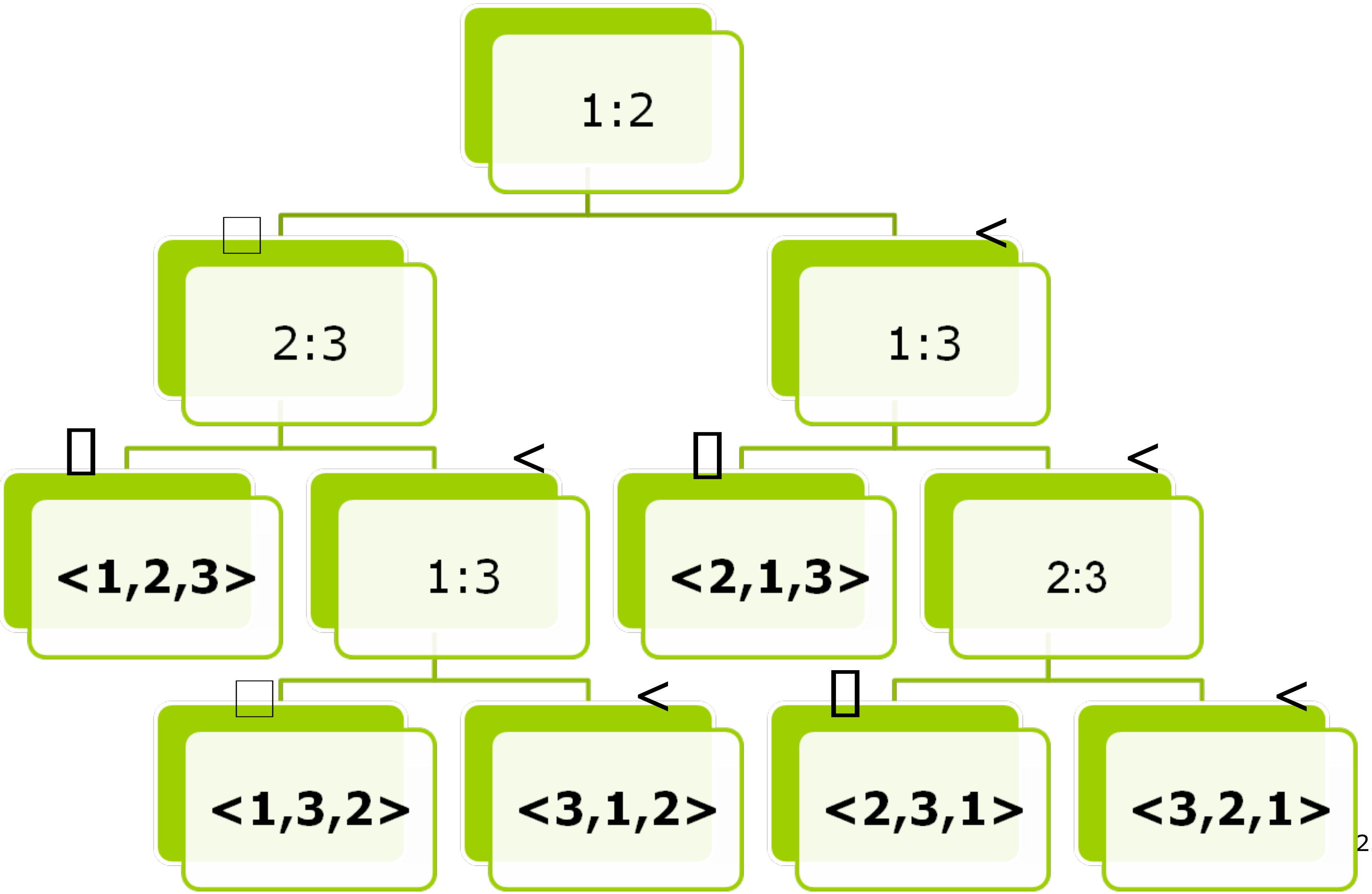
- מס' ההשוואות המקסימאלי = עומק העלה העמוק ביותר בעץ
  - מס' ההשוואות הממוצע = עומק עלה ממוצע
- עץ החלטה למיין n אברים הוא בעל n! עלים בהכרח
  - עץ בינארי מעומק d הוא בעל 2^d עלים לכל היותר
    - עץ בינארי בעל 2^d עלים חייב להיות בעל עומק d
    - עץ בעל n! עלים חייב להיות מעומק של לפחות $|\log(n!)|$
  - ולכן כל אלגוריתם מיון מבוסס השוואות דורש לפחות $|\log(n!)|$ השוואות במקרה הגרוע ביותר.

חסם תחתון על מיון על ידי השוואות

$$n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\log(n!) \geq \frac{n}{2}\log\frac{n}{2} = \Omega(n \log n)$$

חסם תחתון על מיון על ידי השוואות



תרגיל

- הוכיחו כי לא יכול להיות אלגוריתם במודל ההשוואות בו עבור קלט באורך n, לפחות חצי מהפרמוטציות האפשריות של המספרים מ-1 עד n ניתנות למיון בזמן ליניארי.
(עבור המחצית השניה של הפרמוטציות, לאלגוריתם מותר להחזיר כל דבר, אפילו סדר שגוי).

חסם תחתון על מיון על ידי השוואות

- ניתן לייצג כל אלגוריתם מיון (השוואתי) על ידי עץ החלטות בינארי
- בעץ עצמו
  - כל צומת מייצגת את הסידור החלקי כפי שידוע לנו עד נקודה זו
  - קשתות העץ הן תוצאות ההשוואות
  - מסלול משורש לעלה הוא ריצה עבור קלט מסויים

הסוף

שבוע נעים

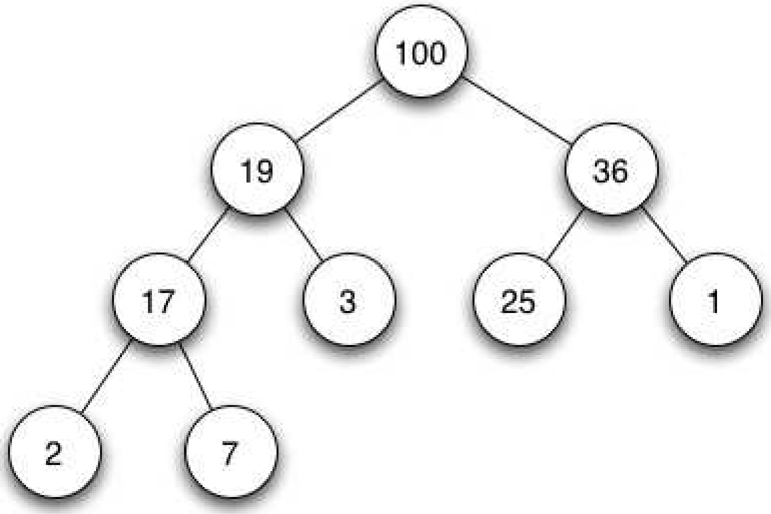
מבני נתונים 08b

ערמות;

מבוסס על מצגות של
יאור שפירא, חיים קפלן וחברים

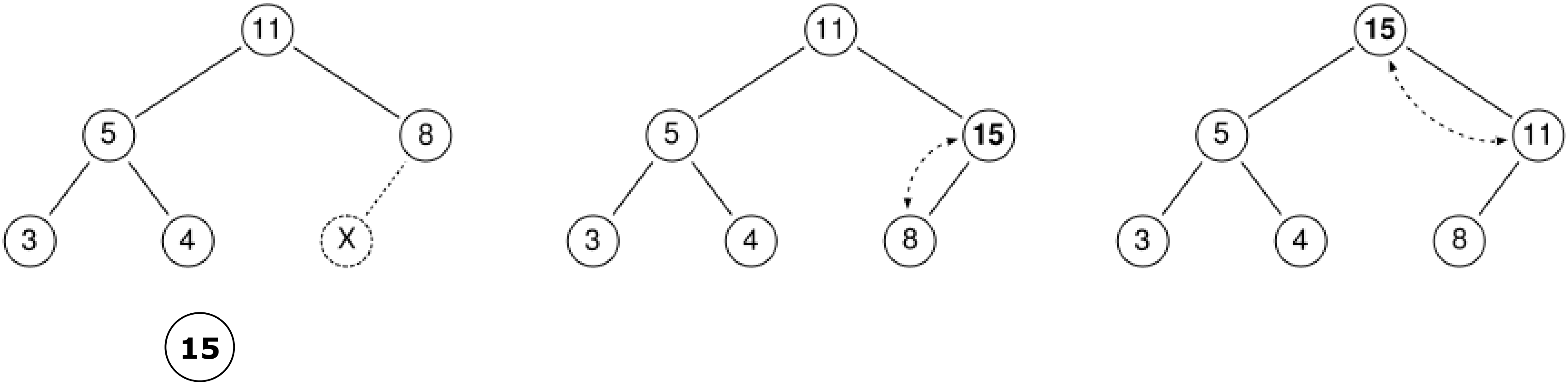
תזכורת: Heaps

- עץ בינארי מלא
- החוק הבסיסי
  - אם צומת B צאצא של צומת A אזי (Key(A)≤Key(B
- הפעולות הנתמכות
  - Find-min
  - Delete-min
  - Decrease-key
  - Insert
  - Merge

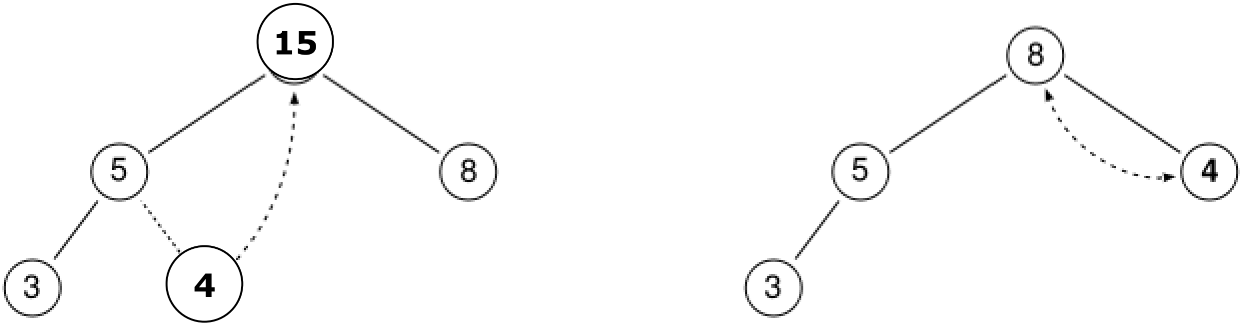


תזכורת: Heaps

הוספת צומת (עבור max-heap)



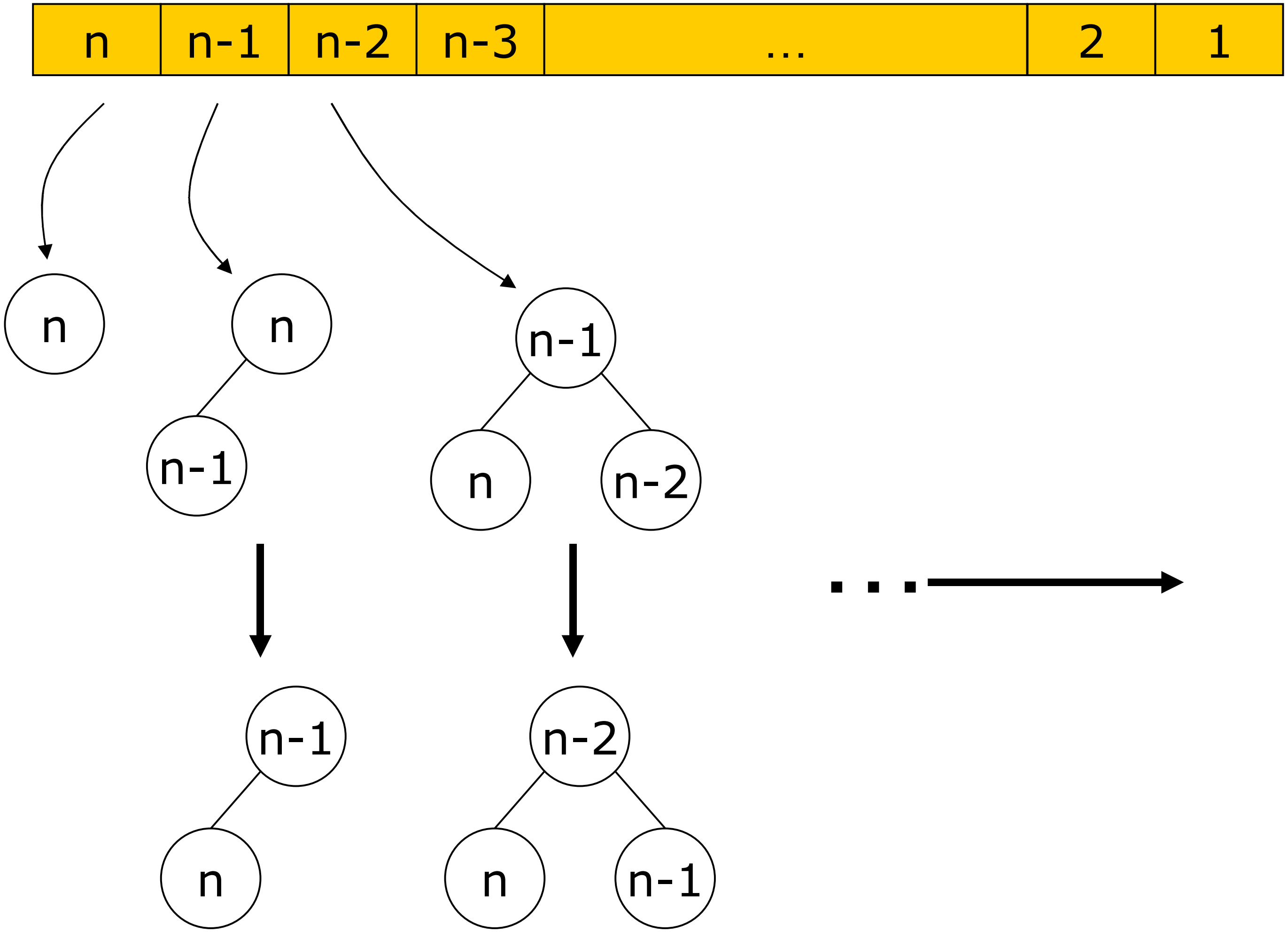
מחיקת השורש



תרגיל 1

- בהינתן מערך באורך n, נרצה ליצור min heap ע"י הכנסה סדרתית של ערכי המערך. הראו סדרת הפעולות לוקחת Ω((nlogn במקרה הכי גרוע (worst case)
- פתרון
  - נצטרך להראות דוגמה של סדרת ההכנסות שלוקחת Ω((nlogn פעולות
  - נחפש סדרה ש"תקשה" כמה שיותר על ה-heap

תרגיל 1



תרגיל 1

- כל ערך שנוסיף צריך לבעבע לראש העץ
- n/2 ההכנסות האחרונות לוקחות לפחות (log(n/2 כל אחת

$$\frac{n}{2}\log(\frac{n}{2}) = \frac{1}{2}n(\log n - \log 2) =$$

$$\frac{1}{2}n\log n - \frac{1}{2}n\log 2 = \theta(n\log n)$$

- מסקנה: W.C =Ω((nlogn

תרגיל 2

- בהינתן heap שתומך בפעולות extract-min ו-insert בזמן amortized $f(n)$, הראו שניתן למיין מערך מגודל n בזמן $O(n \cdot f(n))$
- פתרון
  - נבצע n פעולות הכנסה בזמן $O(n \cdot f(n))$
  - מבצע n פעולות הוצאת מינימום בזמן $O(n \cdot f(n))$
  - סה"כ $O(n \cdot f(n))$
- אלגוריתם מיון זה נקרא heap-sort
- בשיעור תלמדו כי מיון n אברים הוא $\Omega(n \cdot \log n)$

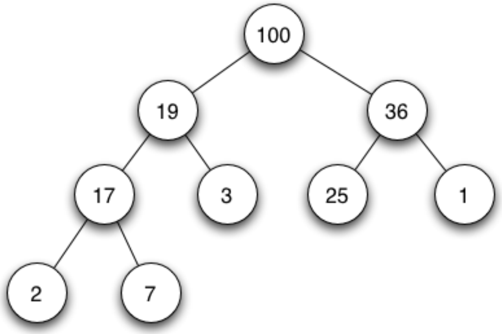
תרגיל 3 – Median Heap

- ממשו מבנה נתונים התומך בפעולות
  - insert בזמן O(logn)
  - extract-median בזמן O(logn)
  - find-median בזמן O(1)

2	4	5	7	8	12	14	15	20

תרגיל 3 - פתרון

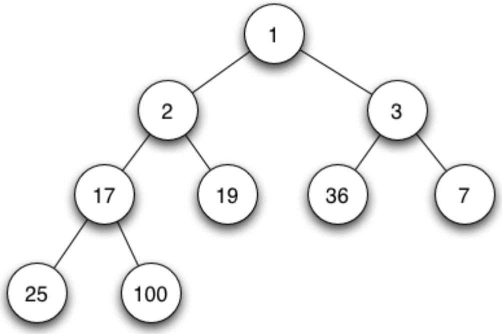
Max-heap



האברים הקטנים
(עד החציון)

+

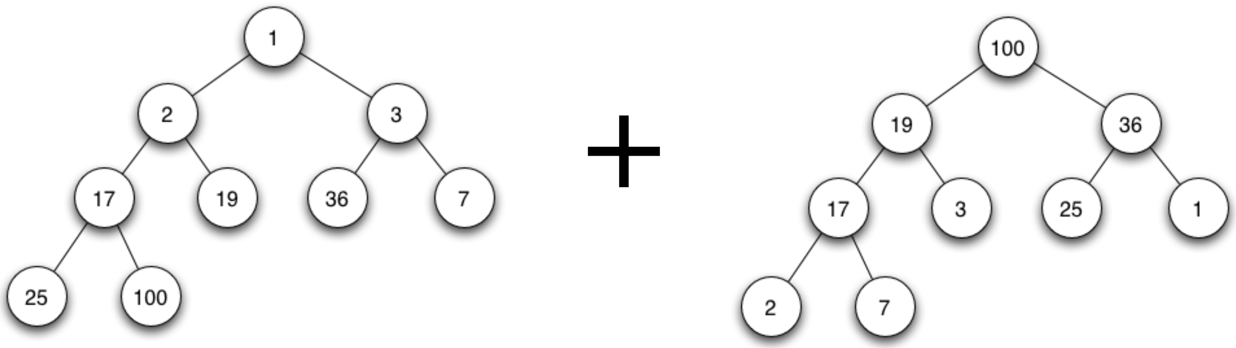
Min-heap



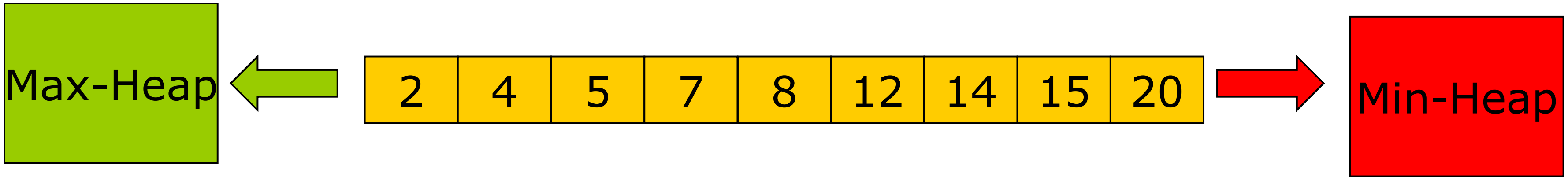
האברים הגדולים
(מהחציון)

תרגיל 3 - פתרון

Min-heap　　　　　　　　Max-heap



- נשתמש ב-max-heap ו-min-heap
- n/2 הערכים הגדולים ביותר יישמרו ב-max-heap
- השאר יישמרו ב-min-heap
- החציון תמיד נמצא בשורש של אחד מהם



תרגיל 3 - פתרון

- Find-median
  - If (size(minheap)>size(maxheap))
    - return getmin(minheap)
  - Else
    - return getmax(maxheap)

O(1)

- Insert(x)
  - If (x<getmin(minheap))
    - Insert(maxheap,x)
  - Else
    - Insert(minheap,x)
  - If (abs(size(minheap)-size(maxheap))>1)
    - Balance heaps (move root from bigger heap to smaller heap)

O(logn)

- Extract-Median
  - Extract median from the max-heap or min-heap…

O(logn)

Heaps

Chapter 6 in CLRS

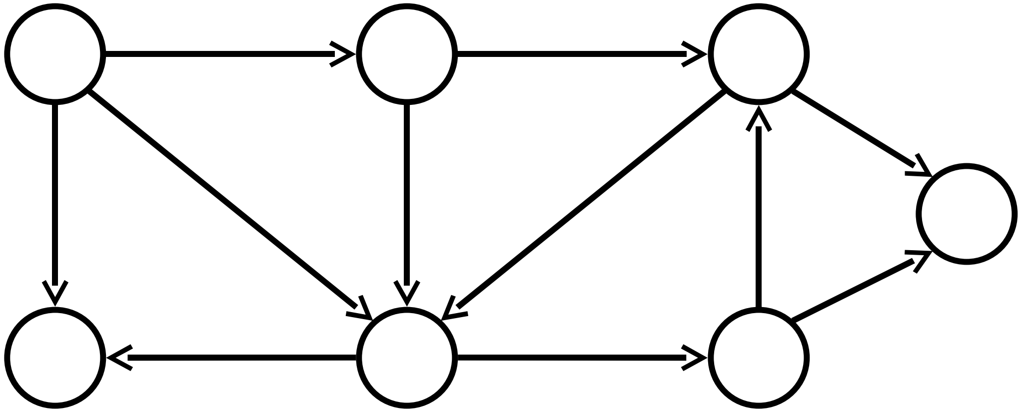
Motivation

- Dijkstra's algorithm for single source shortest path
- Prim's algorithm for minimum spanning trees

Motivation

- Want to find the shortest route from New York to San Francisco
- Model the road-map with a graph

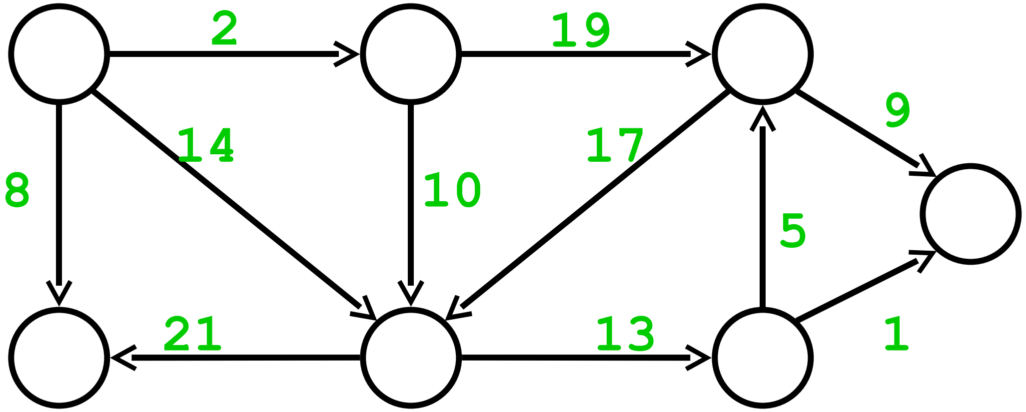
A Graph G=(V,E)



V is a set of vertices

E is a set of edges (pairs of vertices)

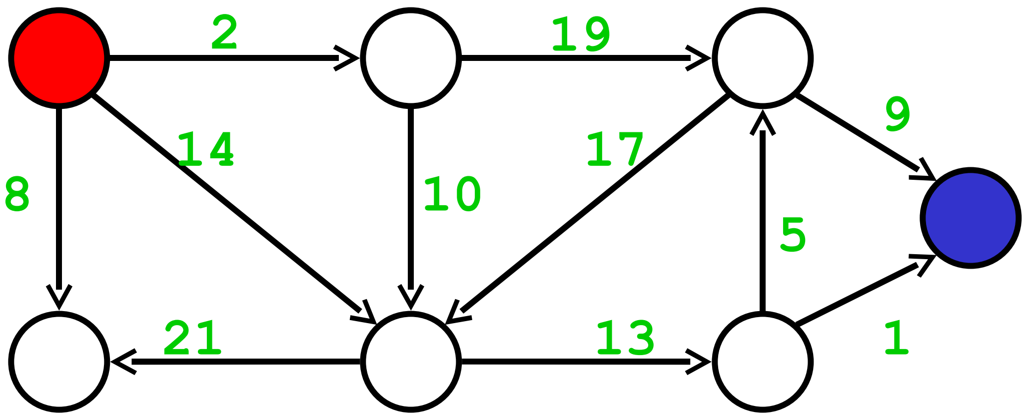
Model driving distances by weights on the edges



V is a set of vertices

E is a set of edges (pairs of vertices)

Source and destination



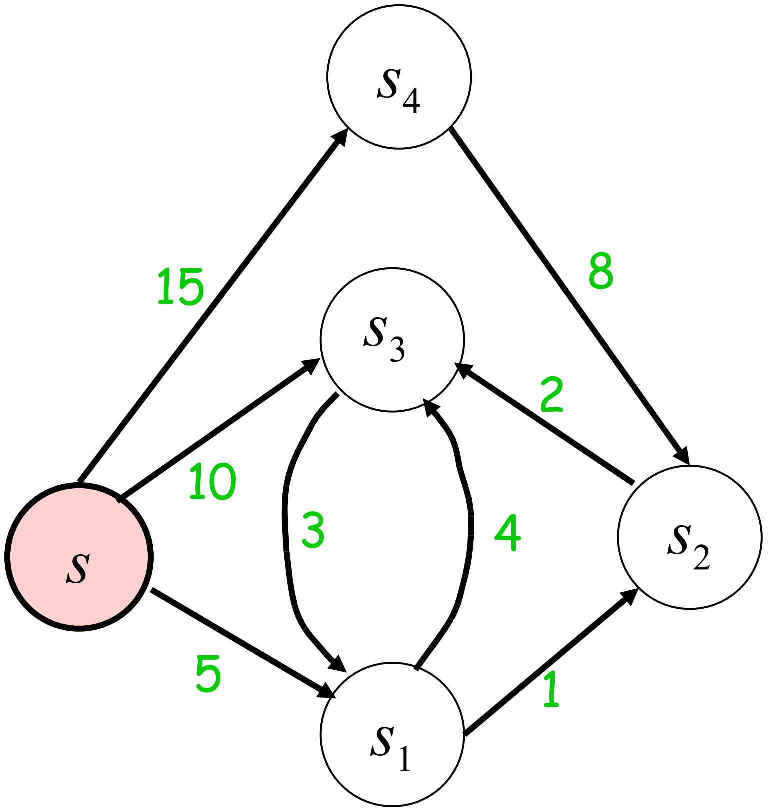
V is a set of vertices

E is a set of edges (pairs of vertices)

Dijkstra's algorithm

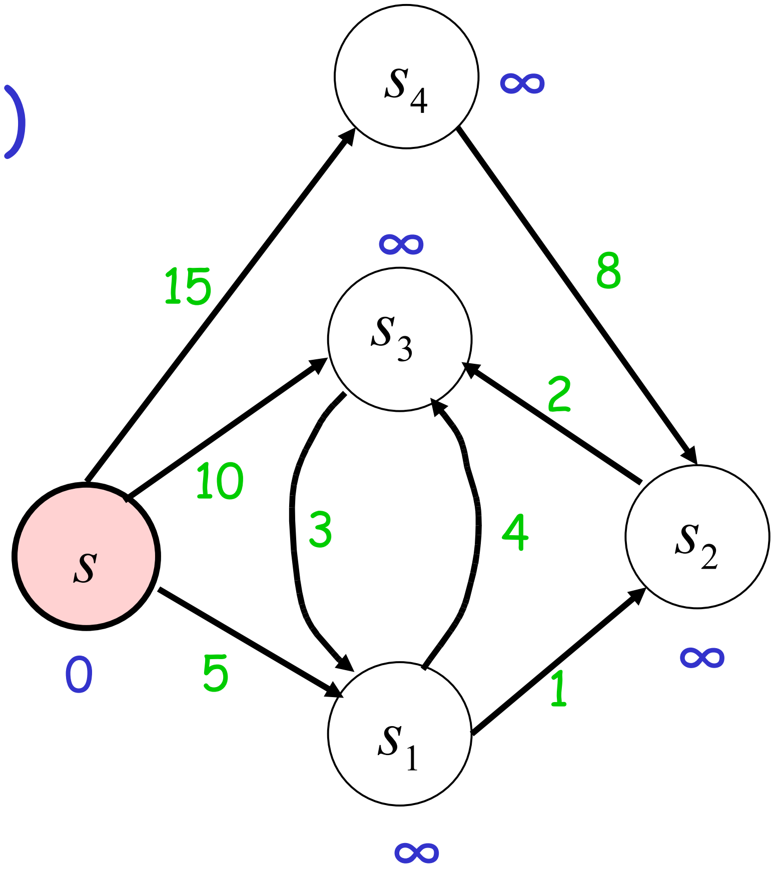
- Assume all weights are non-negative
- Finds the shortest path from some fixed vertex s to every other vertex

Example



Example

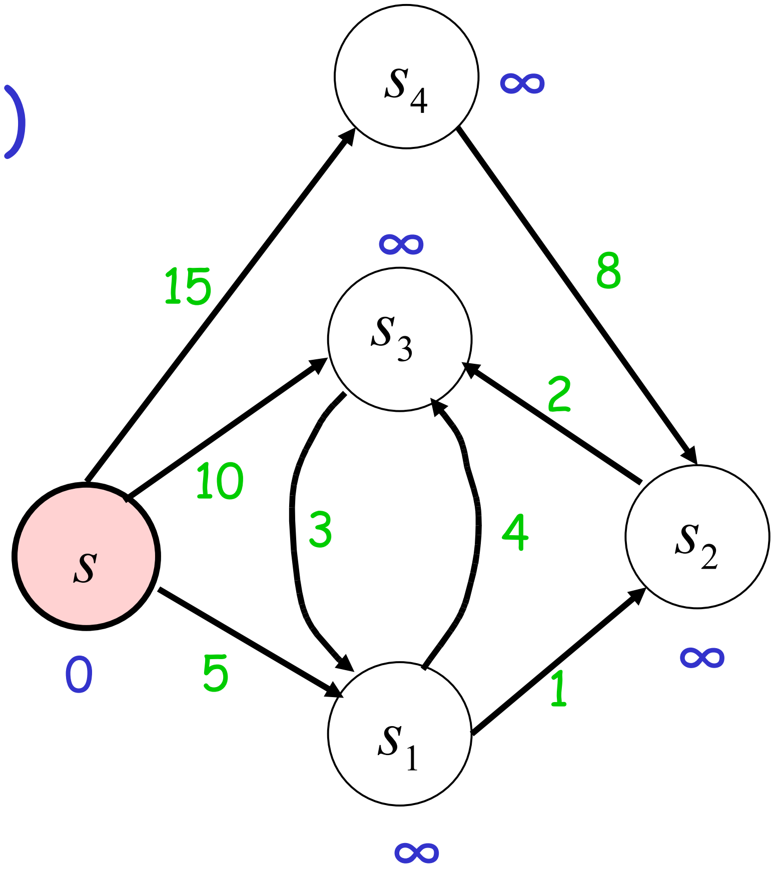
Maintain an upper bound d(v) on the shortest path to v



Maintain an upper bound d(v) on the shortest path to v

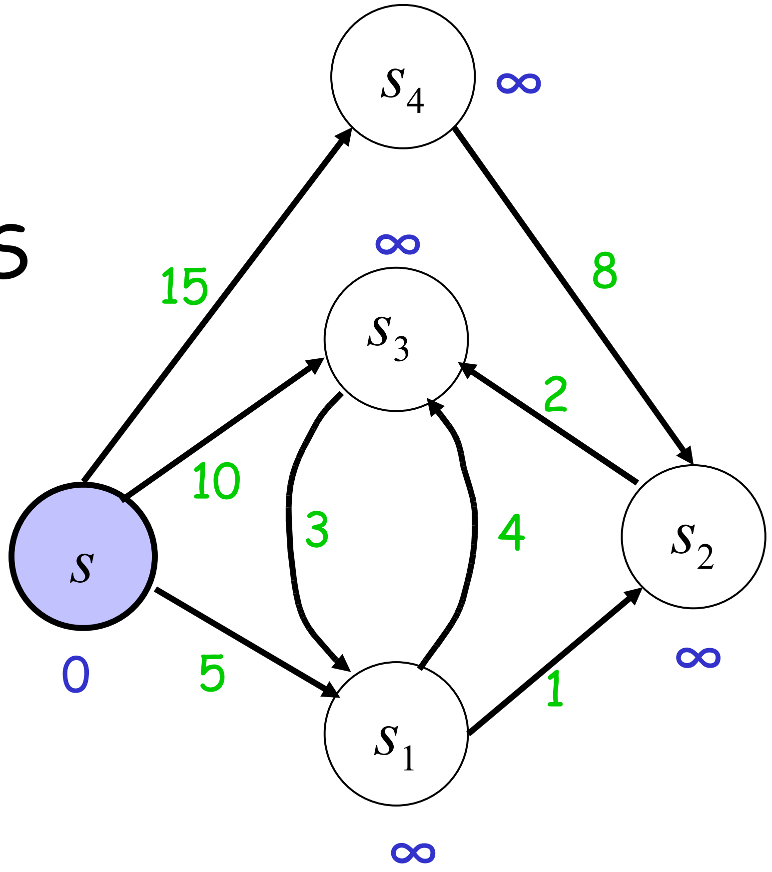
A node is either scanned (in S) or labeled (in Q)

Initially S = ∅ and Q = V



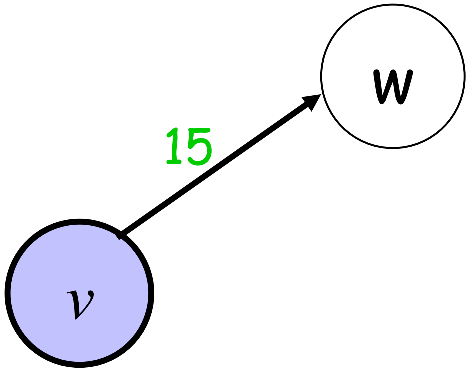
Initially S = ∅ and Q = V

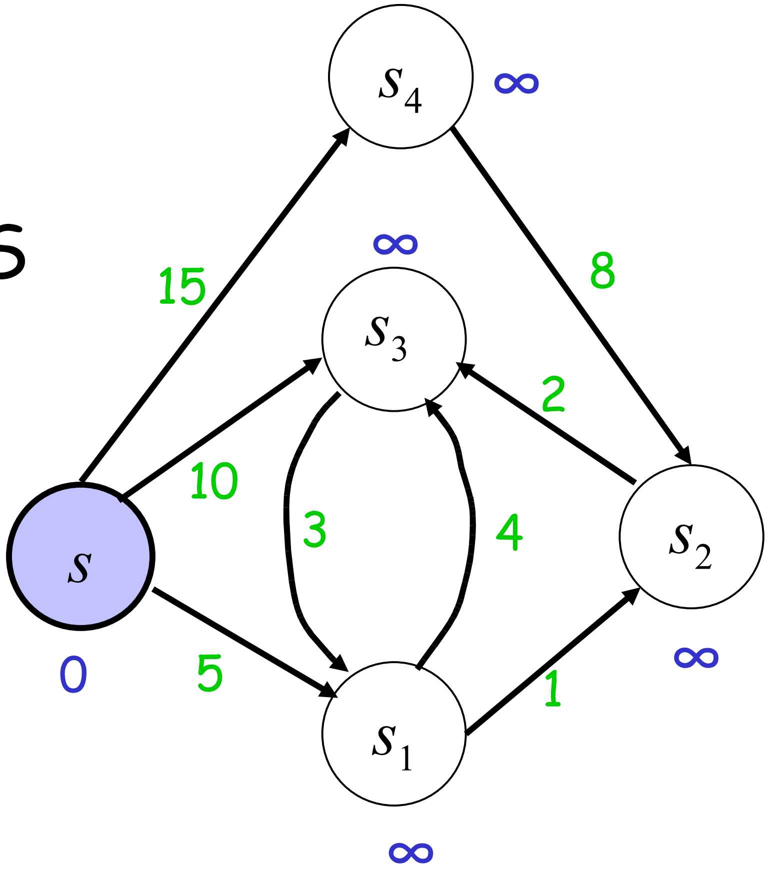
Pick a vertex v in Q with minimum d(v) and add it to S



Initially S = ∅ and Q = V

Pick a vertex v in Q with minimum d(v) and add it to S





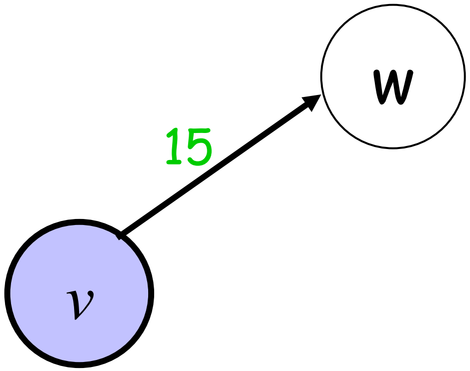
For every edge (v,w) where w in Q relax(v,w)

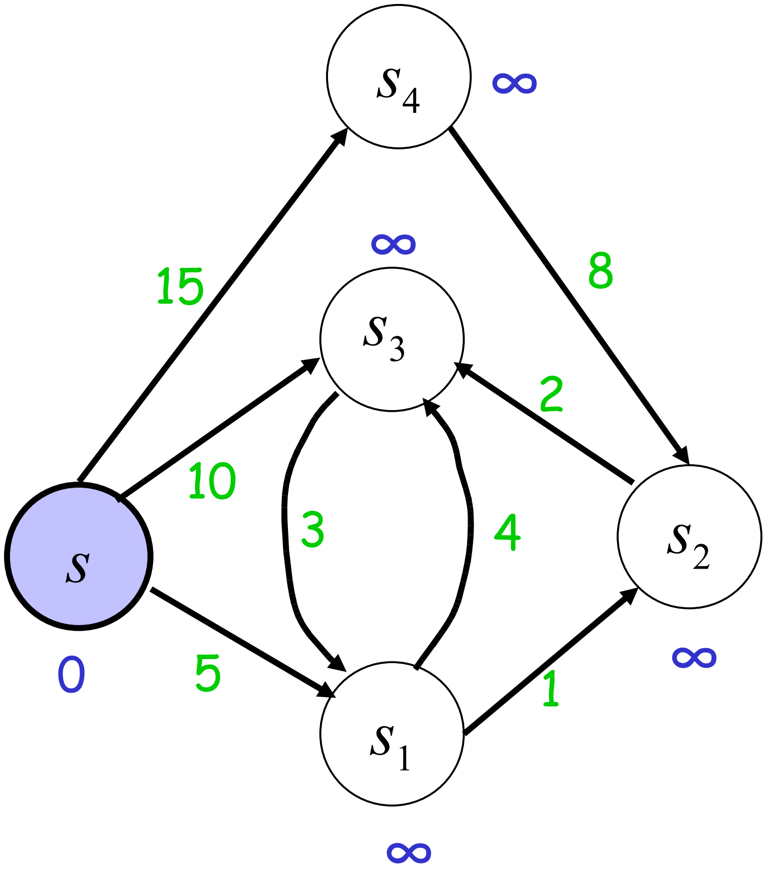
Relax(v,w)

If $d(v) + w(v,w) < d(w)$ then

$d(w) \leftarrow d(v) + w(v,w)$

$\pi(v) \leftarrow w$

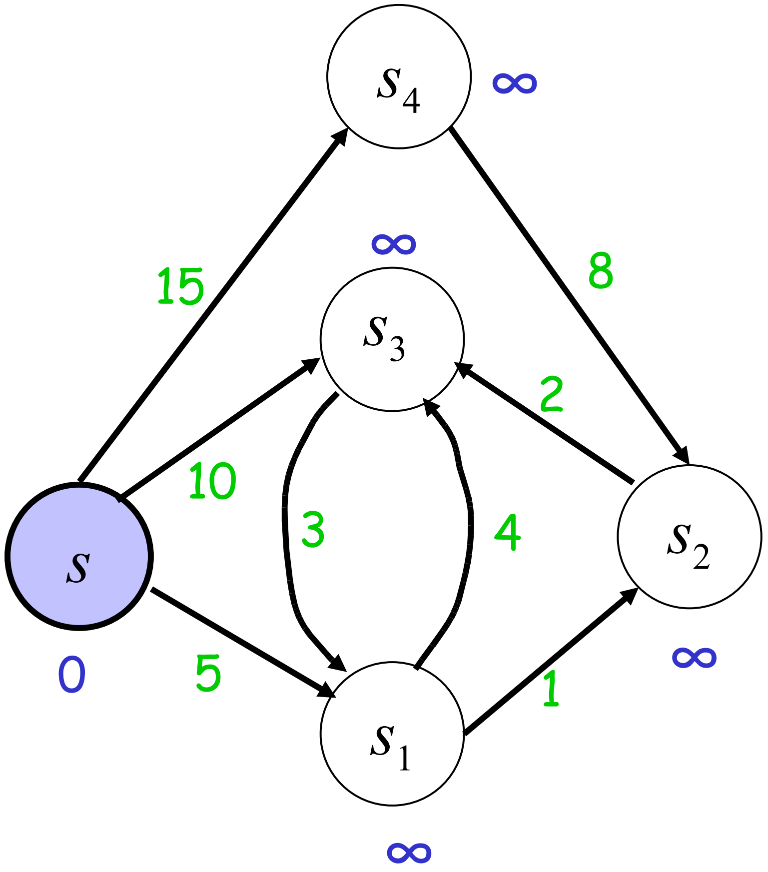




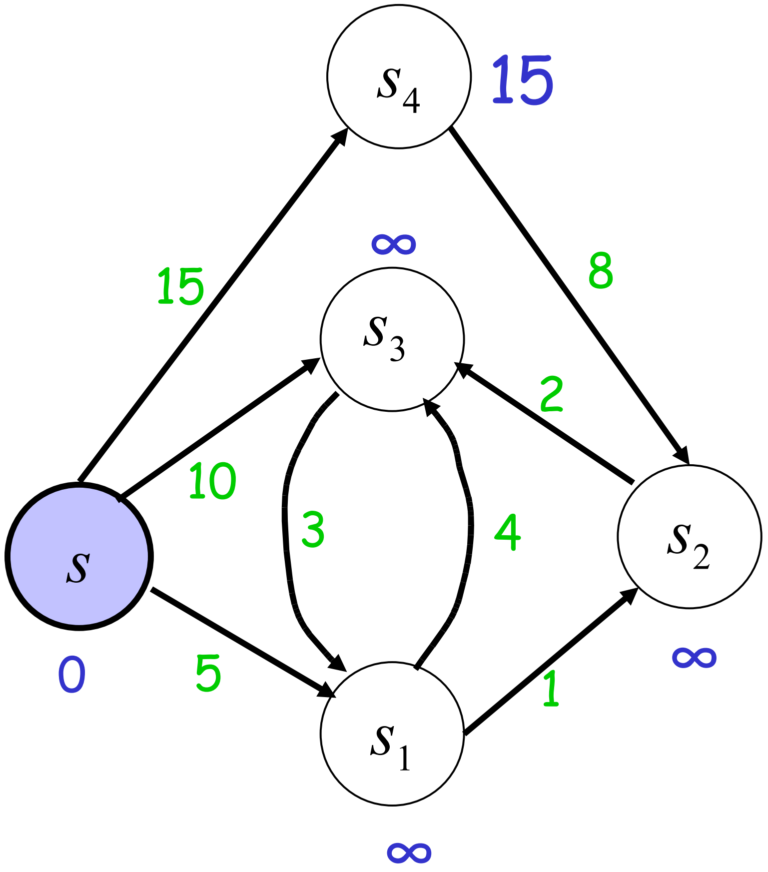
For every edge (v,w) where w in Q relax(v,w)

$S = \{s\}$

$\text{Relax}(s, s_4)$

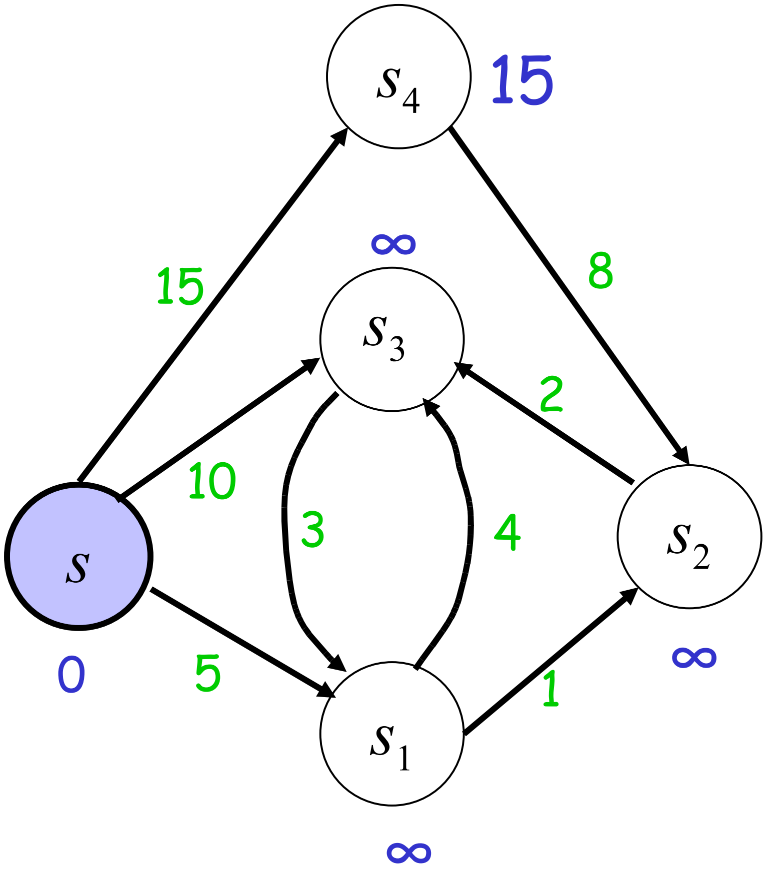


$S = \{s\}$



Relax(s,s_4)

$S = \{s\}$

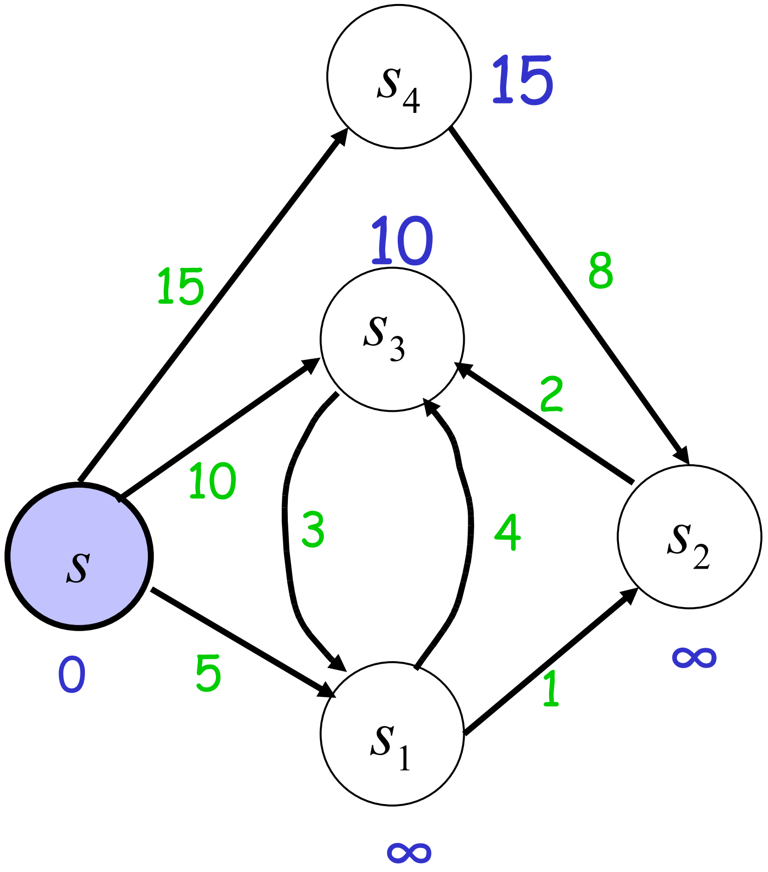


Relax(s,s_4)

Relax(s,s_3)

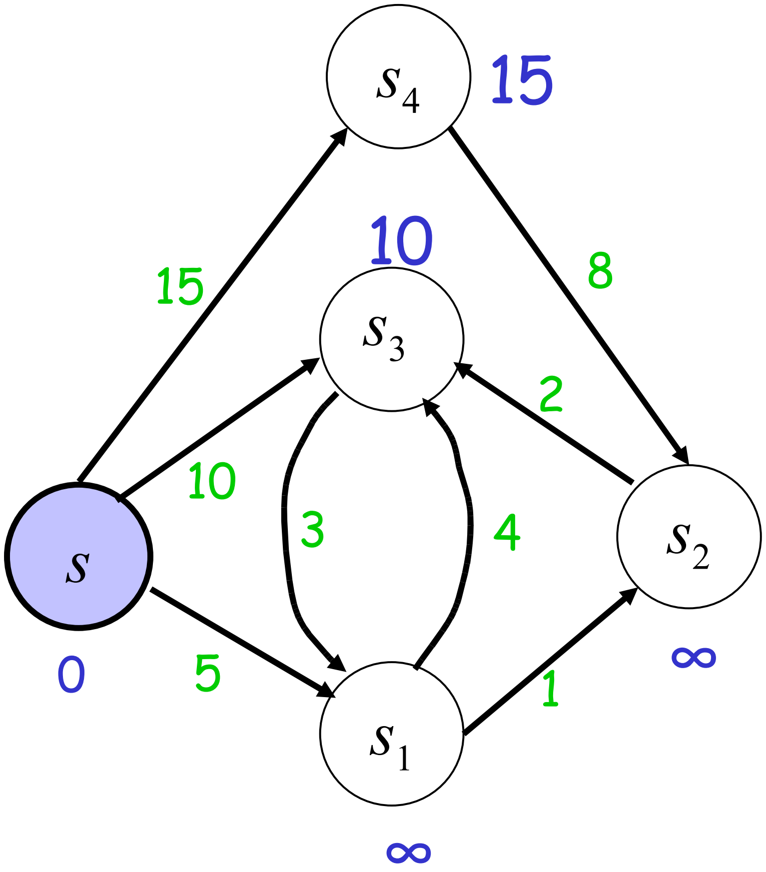
$S = \{s\}$

Relax(s, s_4)
Relax(s, s_3)



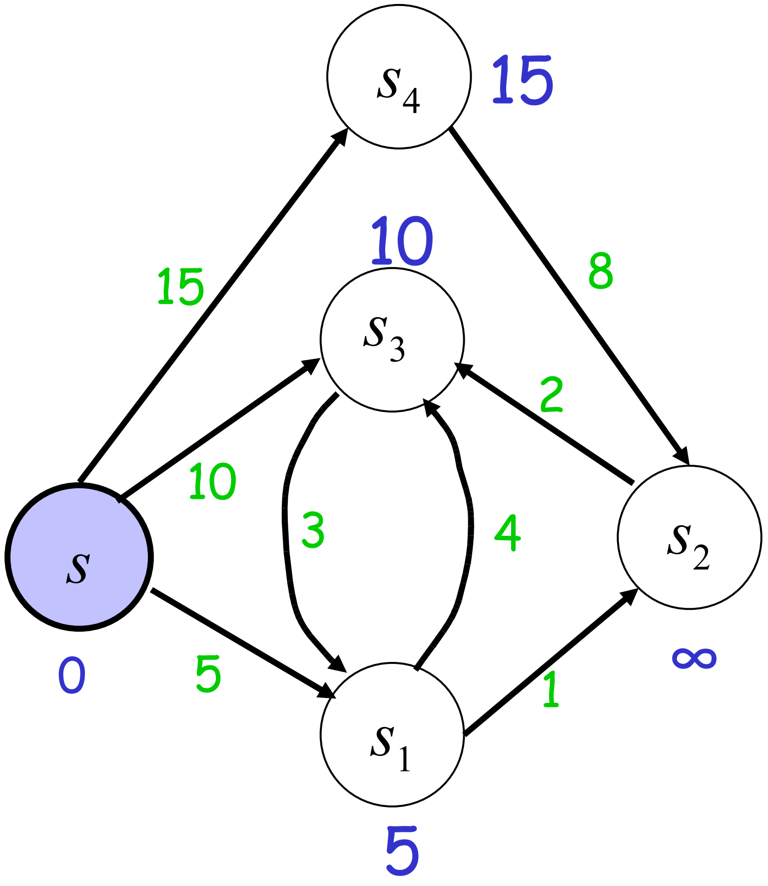
$S = \{s\}$

$Relax(s,s_4)$
$Relax(s,s_3)$
$Relax(s,s_1)$



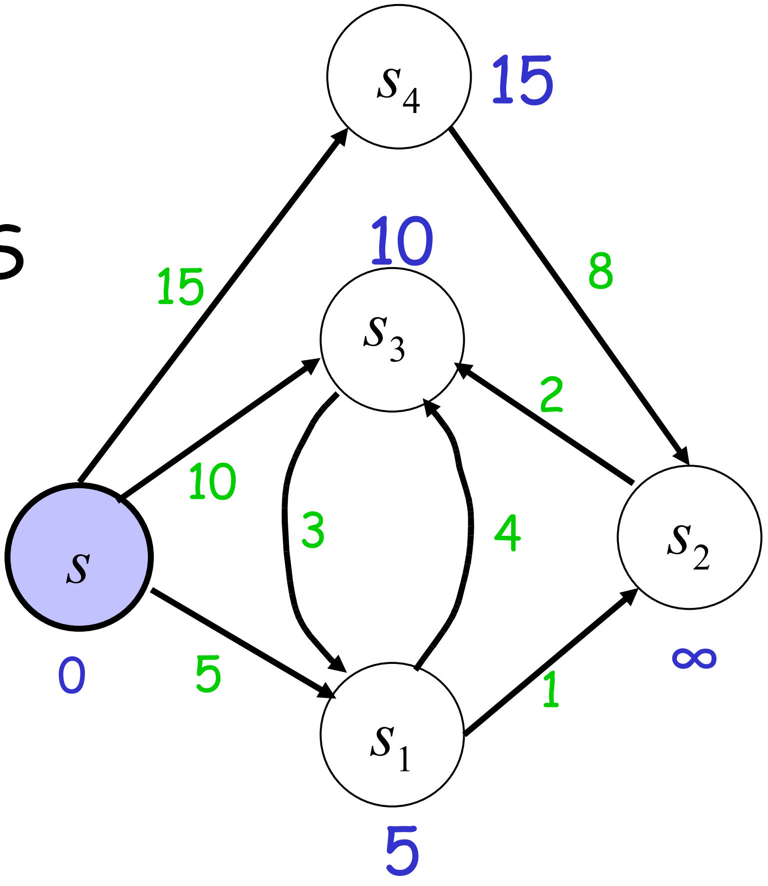
S = {s}

Relax(s,s_4)
Relax(s,s_3)
Relax(s,s_1)



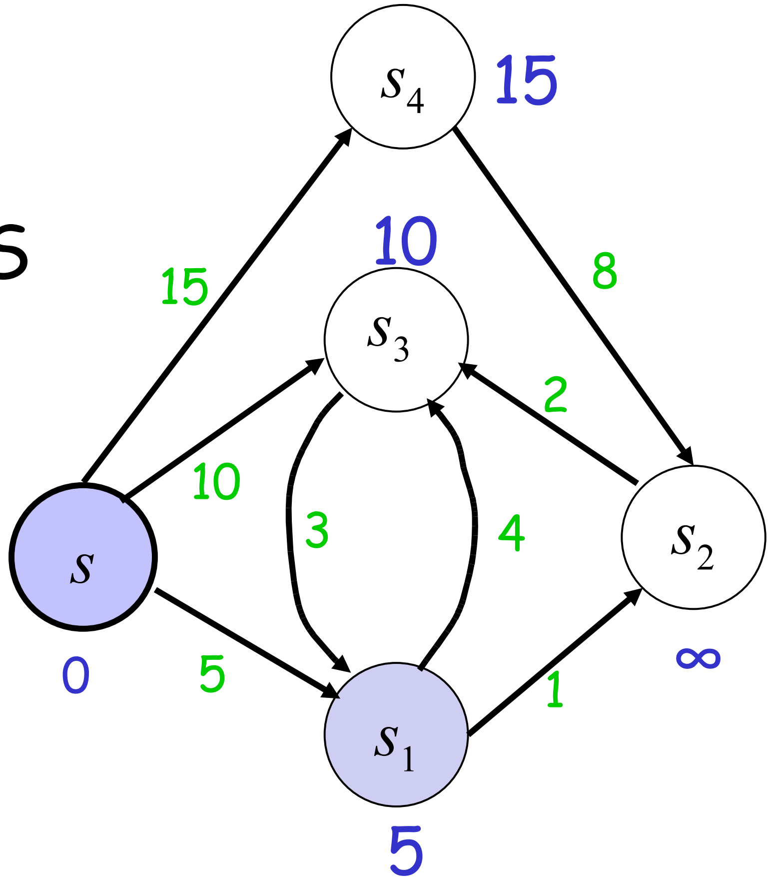
S = {s}

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1\}$

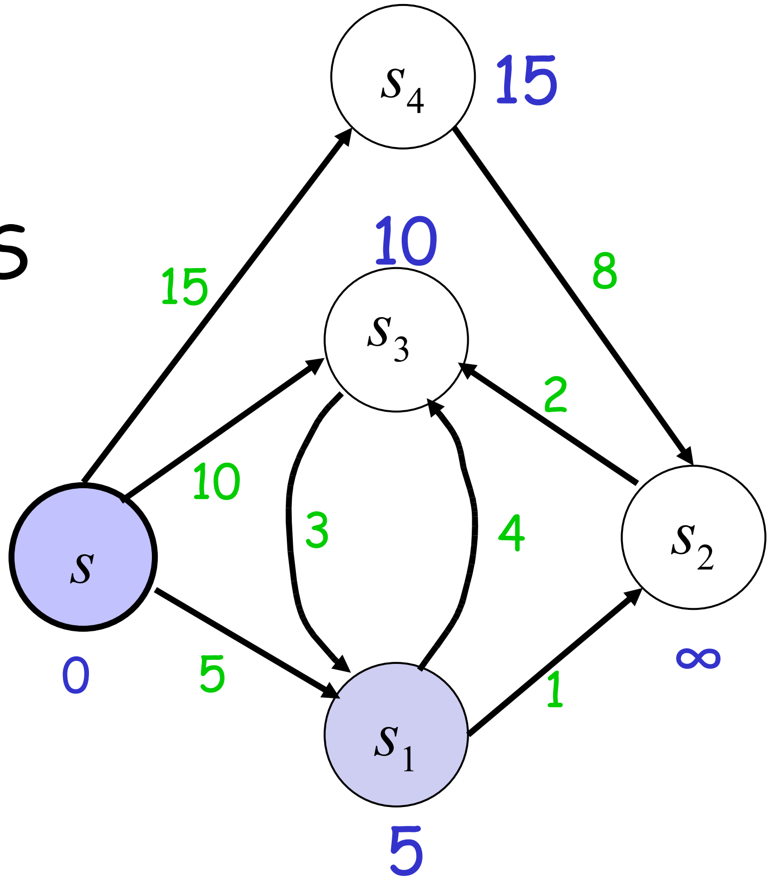
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

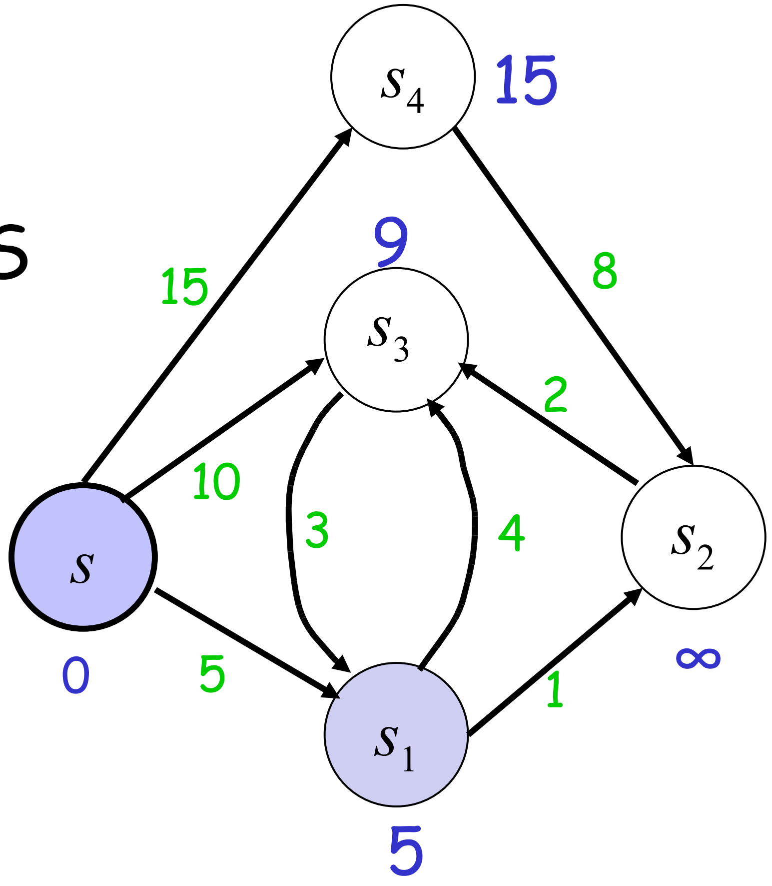
Relax(s_1,s_3)



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

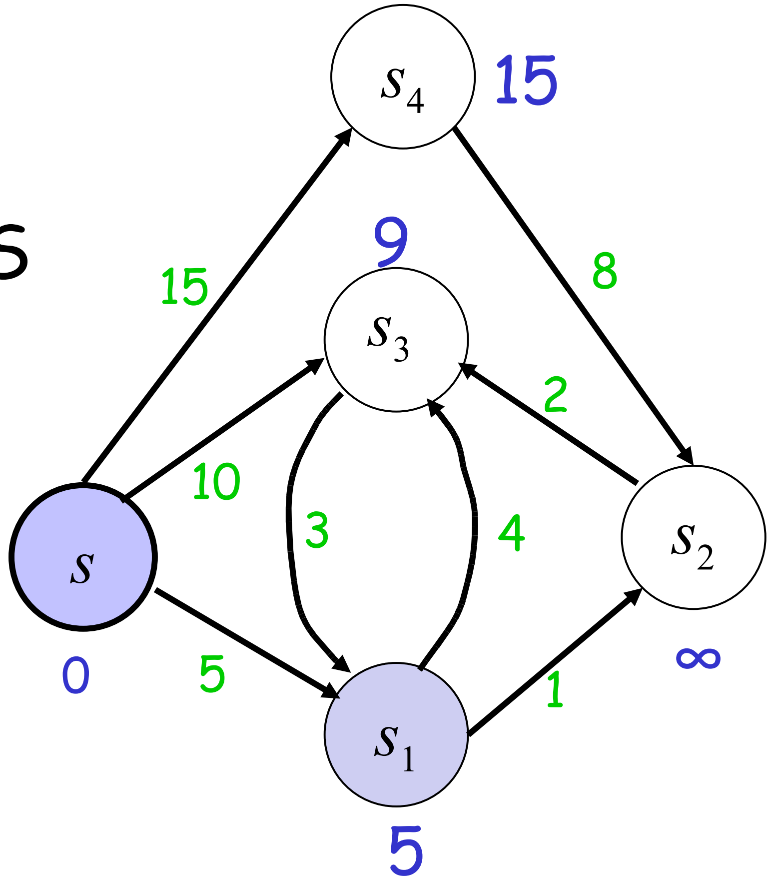
Relax(s_1,s_3)



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

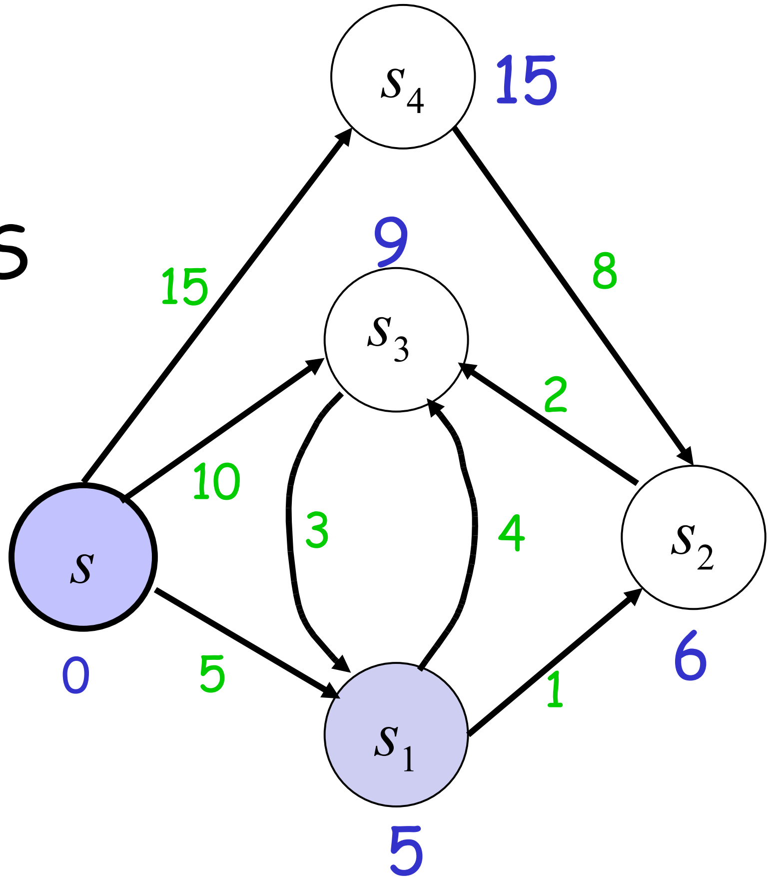
Relax(s_1,s_3)
Relax(s_1,s_2)



$S = \{s, s_1\}$

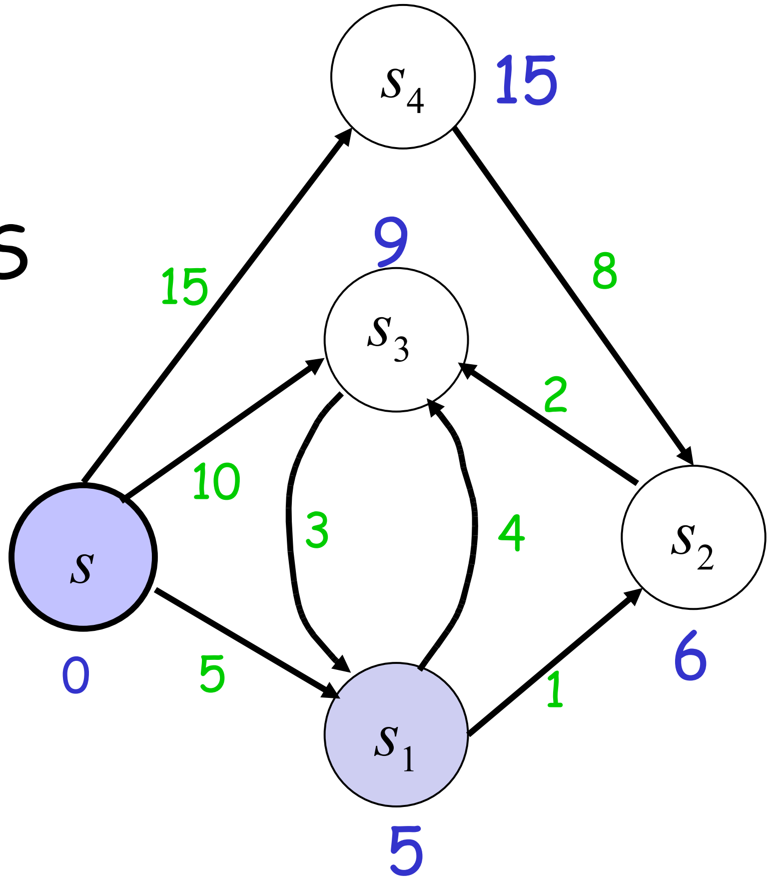
Pick a vertex v in Q with minimum d(v) and add it to S

$Relax(s_1, s_3)$
$Relax(s_1, s_2)$



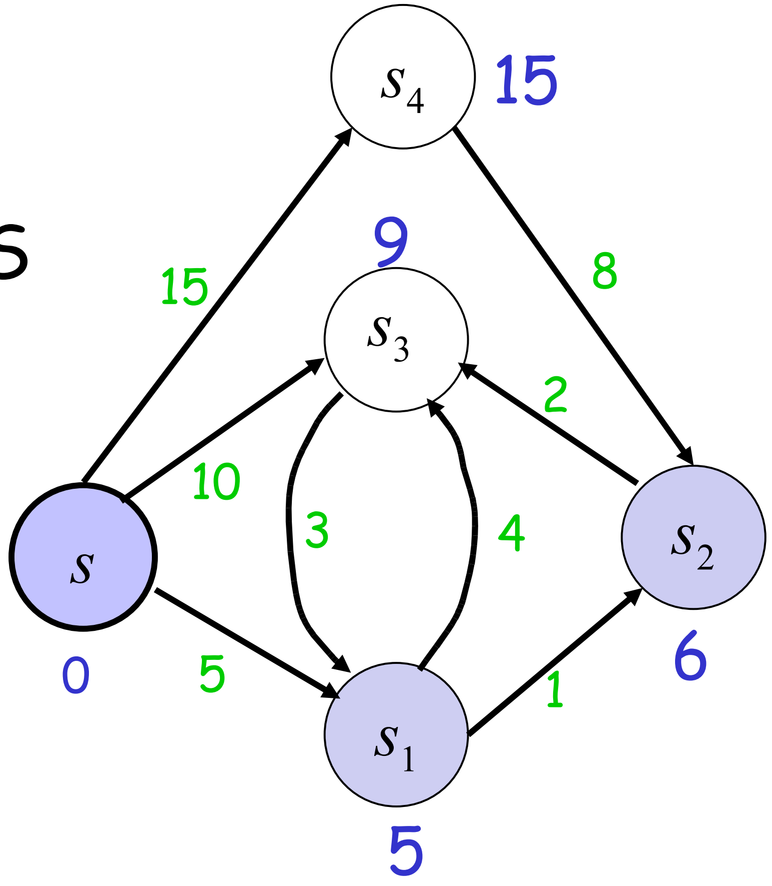
$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2\}$

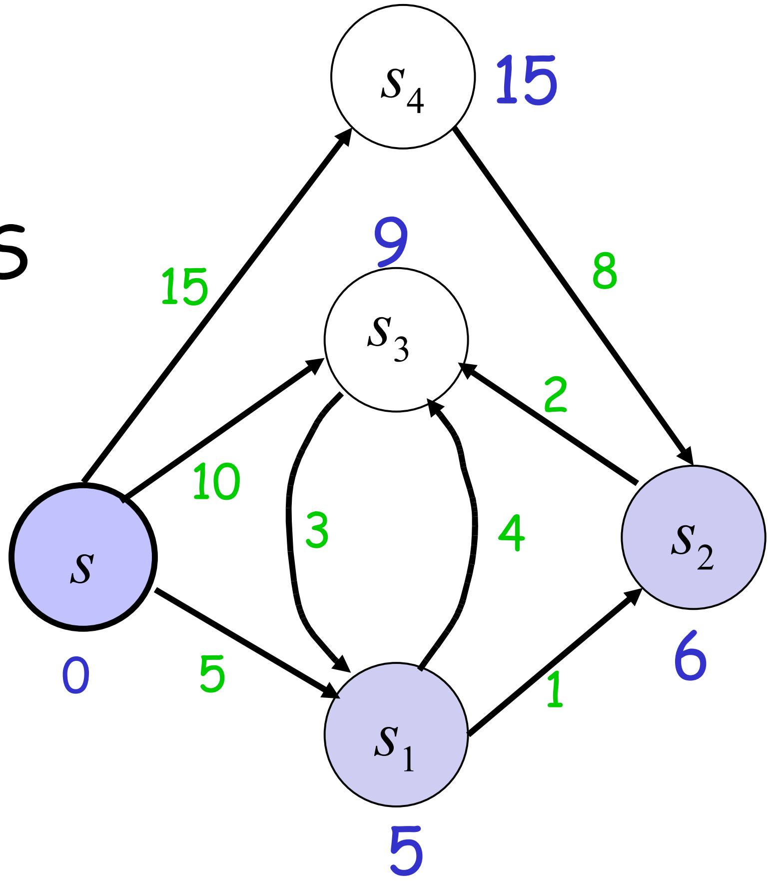
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2\}$

Pick a vertex v in Q with minimum d(v) and add it to S

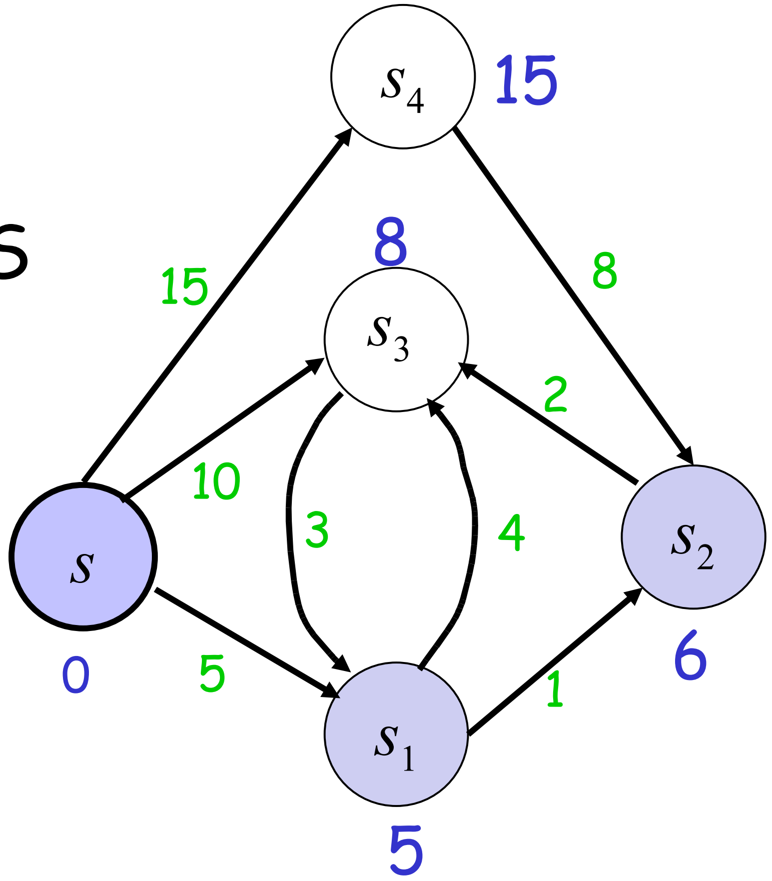
Relax(s_2,s_3)



$S = \{s, s_1, s_2\}$

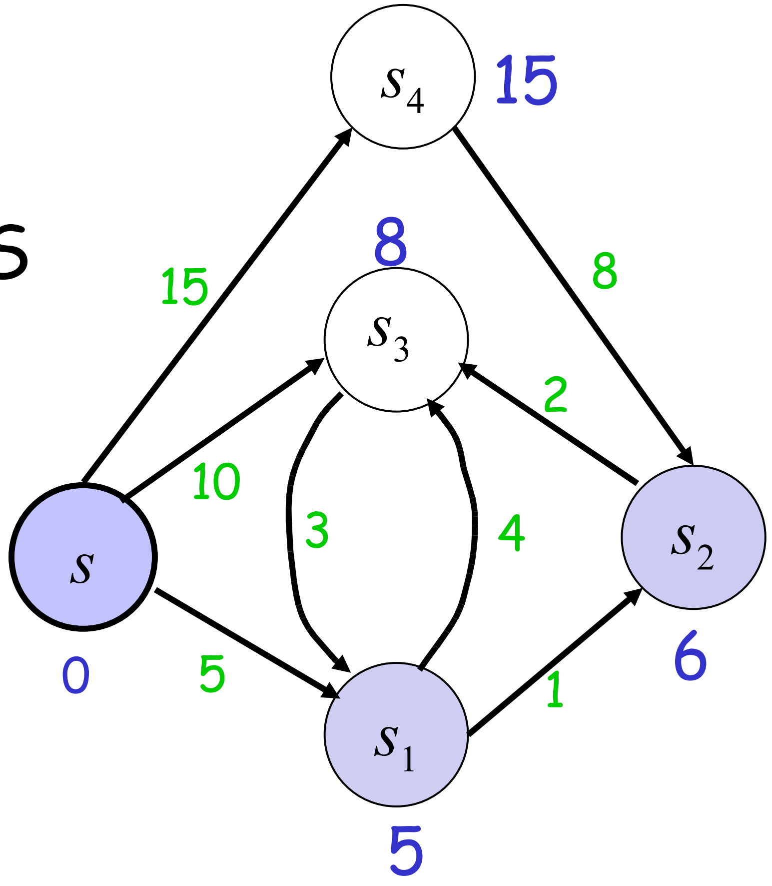
Pick a vertex v in Q with minimum d(v) and add it to S

Relax(s_2,s_3)



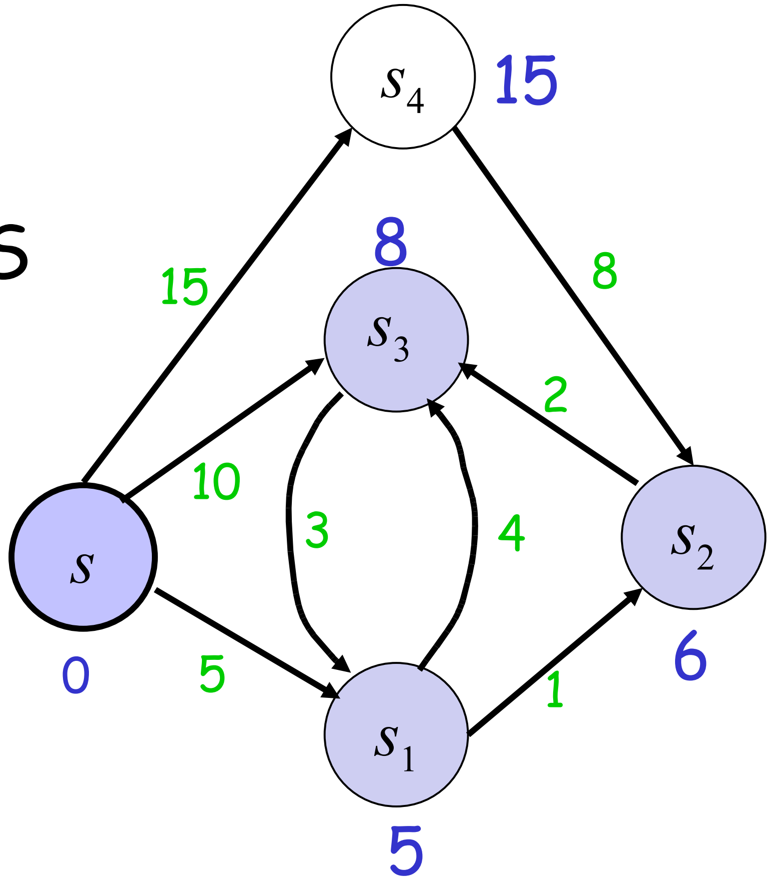
$S = \{s, s_1, s_2\}$

Pick a vertex v in Q with minimum d(v) and add it to S



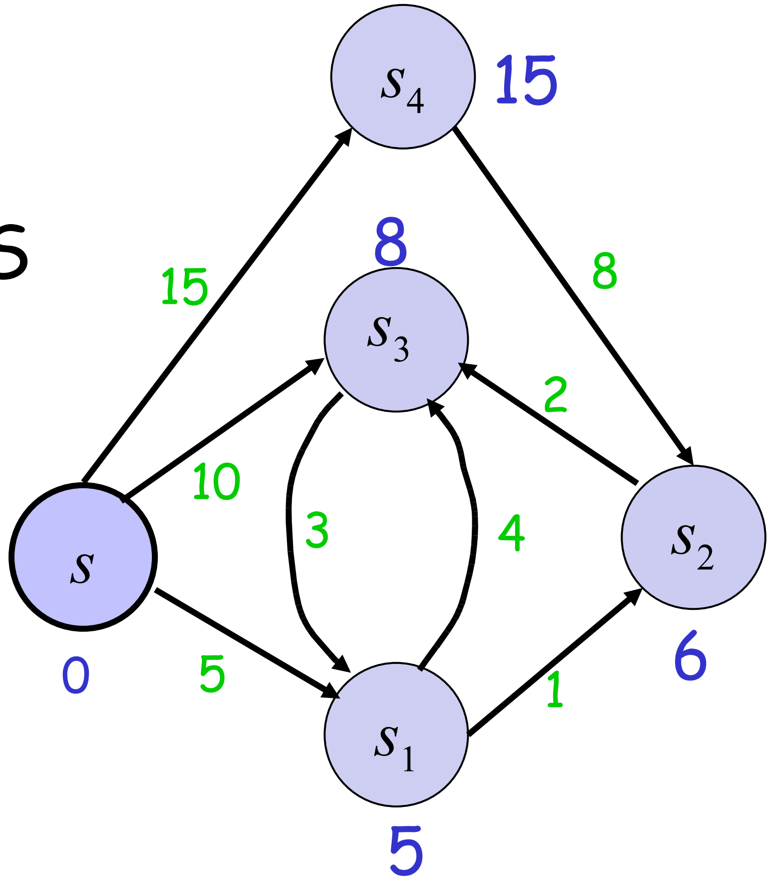
$S = \{s, s_1, s_2, s_3\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2, s_3, s_4\}$

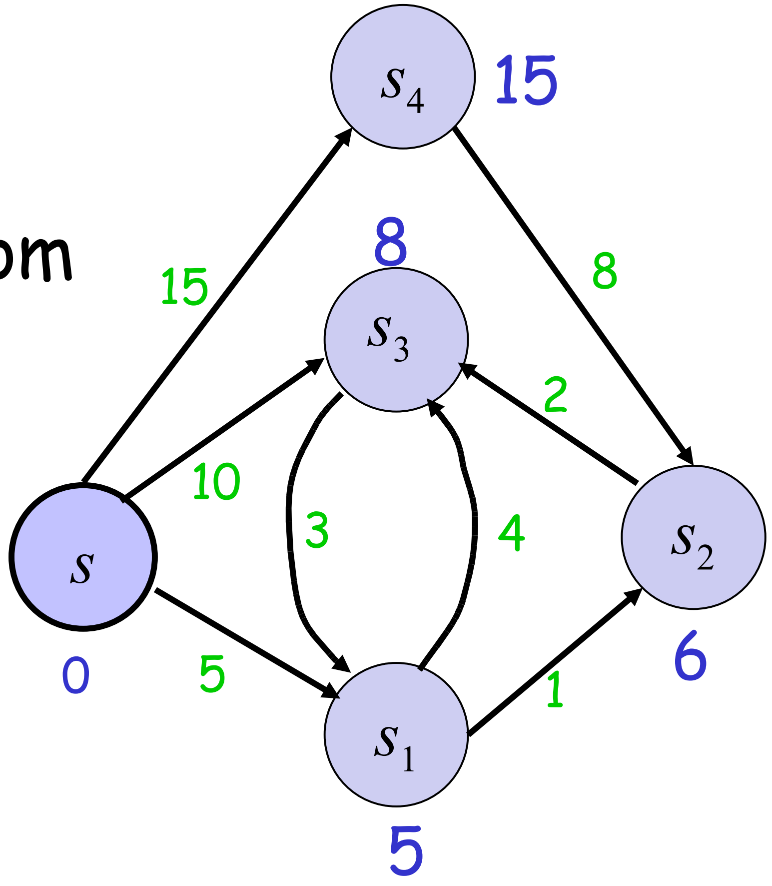
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2, s_3, s_4\}$

When $Q = \emptyset$ then the d() values are the distances from s

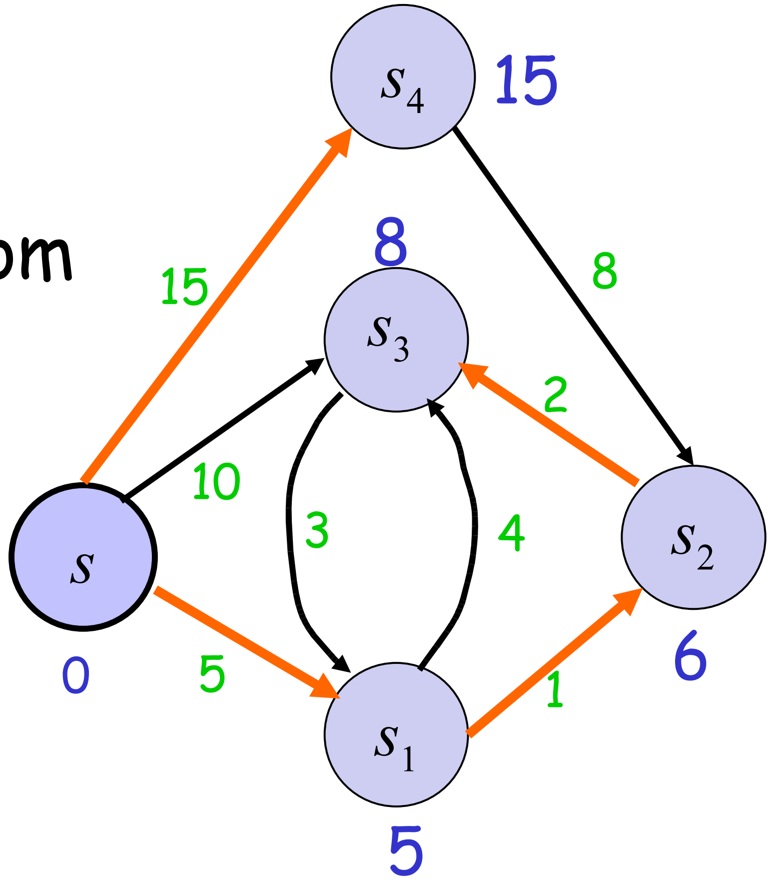
The π function gives the shortest path tree



$S = \{s, s_1, s_2, s_3, s_4\}$

When $Q = \emptyset$ then the d() values are the distances from s

The π function gives the shortest path tree



Implementation of Dijkstra's algorithm

- We need to find efficiently the vertex with minimum d() in Q
- We need to update d() values of vertices in Q

Required ADT

- Maintain items with keys subject to

- Insert(x,Q)
- min(Q)
- Deletemin(Q)
- Decrease-key(x,Q,Δ)

Required ADT

- Insert(x,Q)
- min(Q)
- Deletemin(Q)
- Decrease-key(x,Q,Δ): Can simulate by
  Delete(x,Q), insert(x-Δ,Q)

How many times we do these operations ?

- Insert(x,Q) $n = |V|$
- min(Q) n
- Deletemin(Q) n
- Decrease-key(x,Q,Δ): Can simulate by Delete(x,Q), insert(x-Δ,Q) $m = |E|$

Do we know an algorithm for this ADT ?

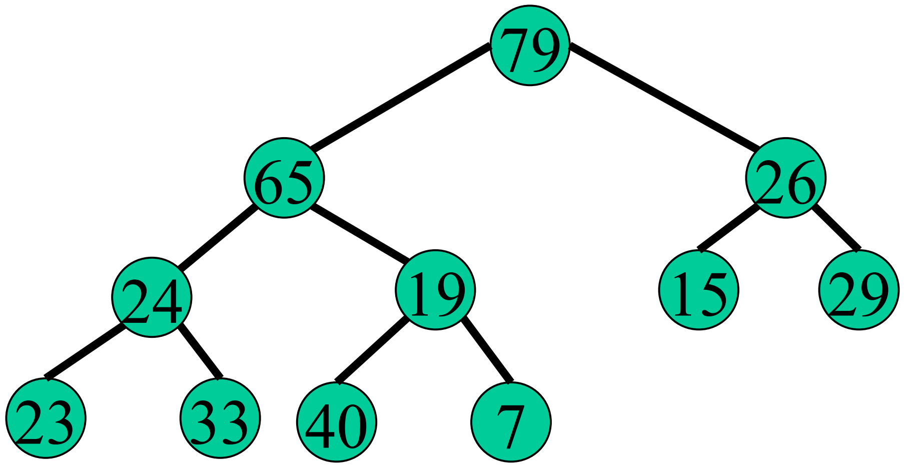
- Insert(x,Q)
- min(Q)
- Deletemin(Q)
- Decrease-key(x,Q,Δ): Can simulate by Delete(x,Q), insert(x-Δ,Q)

Yes! Use binary search trees, we had insert and delete and also min in the dictionary data type

Heapsort (Williams, Floyd, 1964)

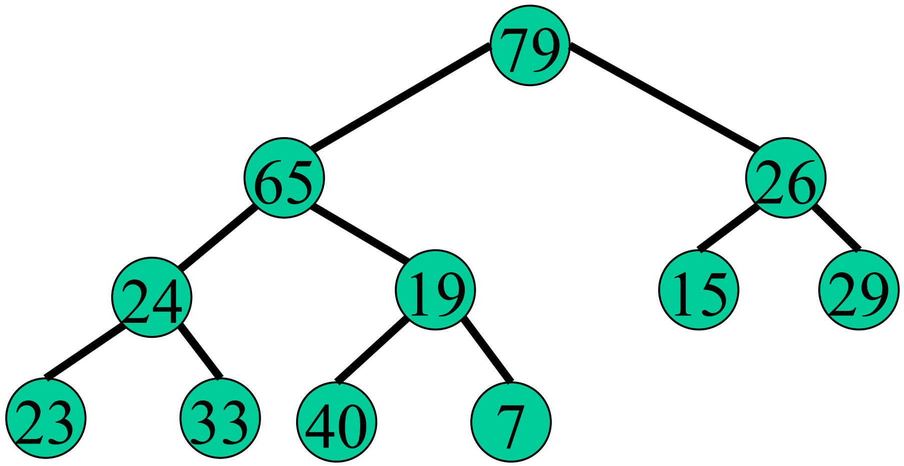
- Put the elements in an array
- Make the array into a heap
- Do a deletemin and put the deleted element at the last position of the array

Put the elements in the heap



Q	79	65	26	24	19	15	29	23	33	40	7

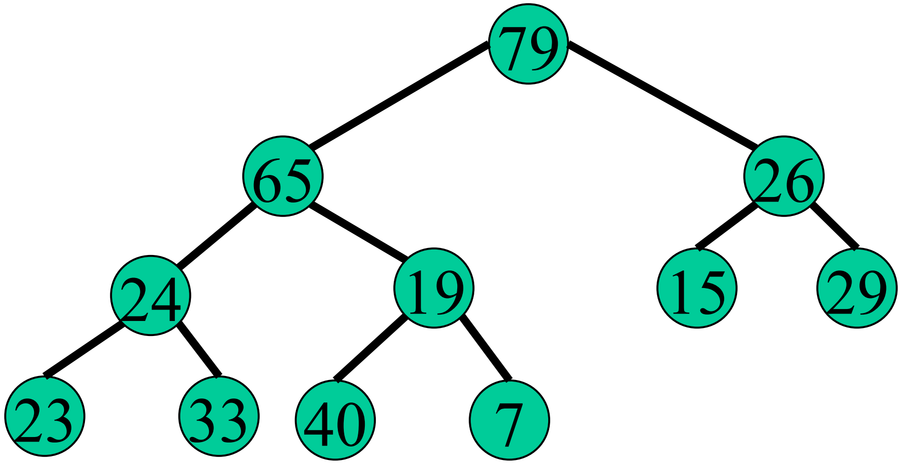
Make the elements into a heap



Q										
79	65	26	24	19	15	29	23	33	40	7

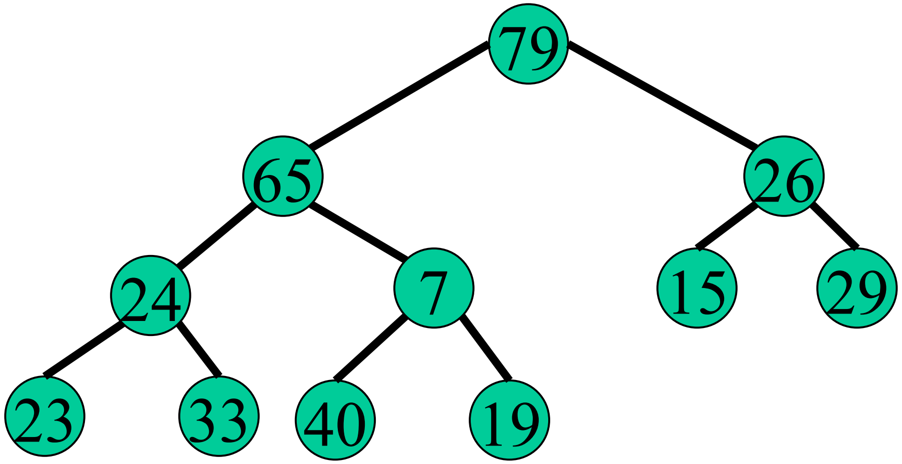
Make the elements into a heap

Heapify-down(Q,4)



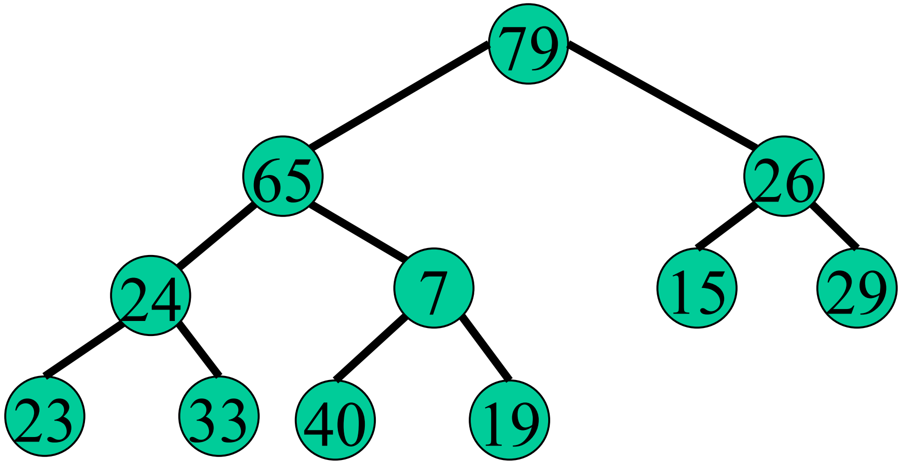
Q										
79	65	26	24	19	15	29	23	33	40	7

Heapify-down(Q,4)



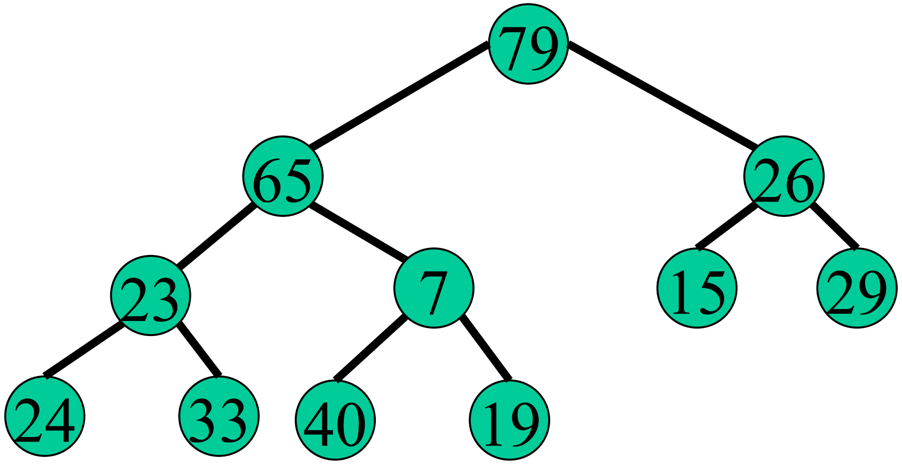
Q											
	79	65	26	24	7	15	29	23	33	40	19

Heapify-down(Q,3)



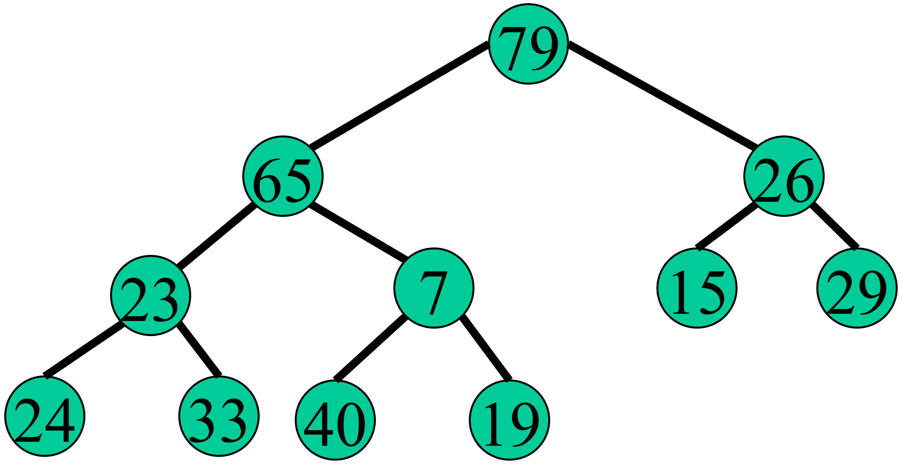
Q										
79	65	26	24	7	15	29	23	33	40	19

Heapify-down(Q,3)



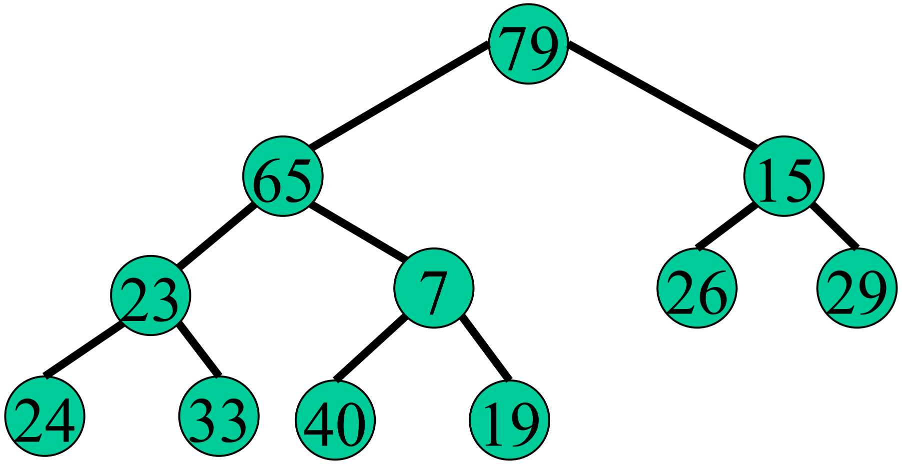
Q	79	65	26	23	7	15	29	24	33	40	19

Heapify-down(Q,2)



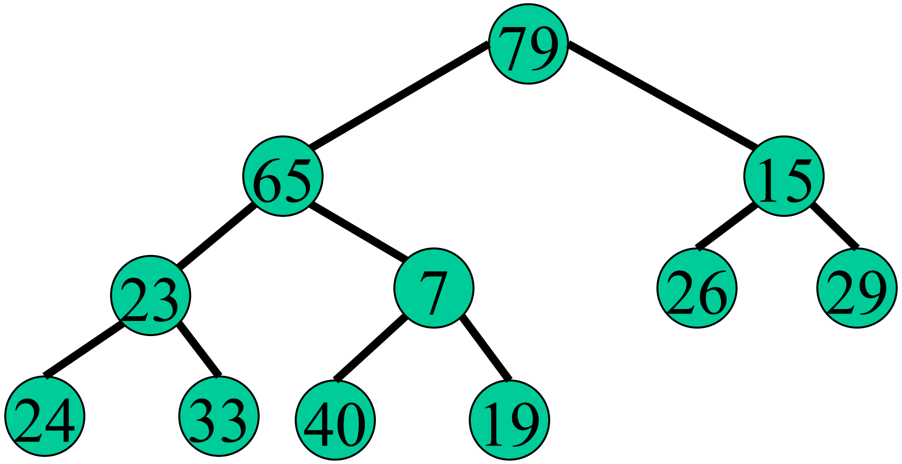
Q											
	79	65	26	23	7	15	29	24	33	40	19

Heapify-down(Q,2)



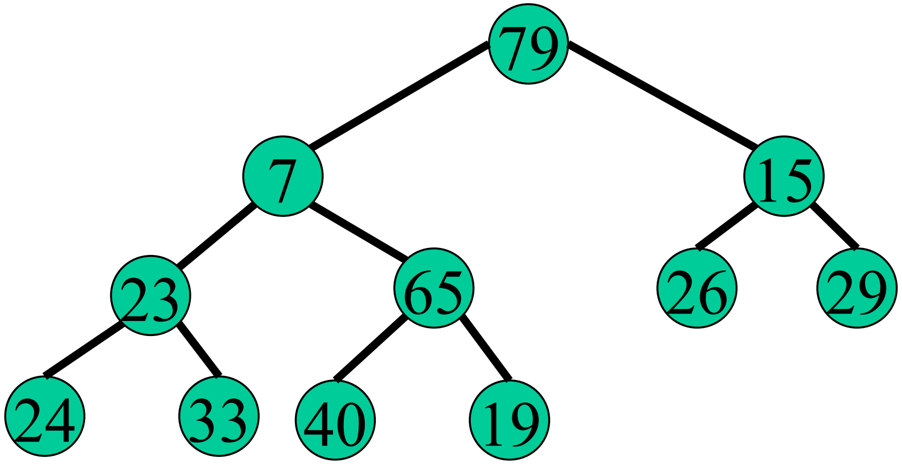
Q	79	65	15	23	7	26	29	24	33	40	19

Heapify-down(Q,1)



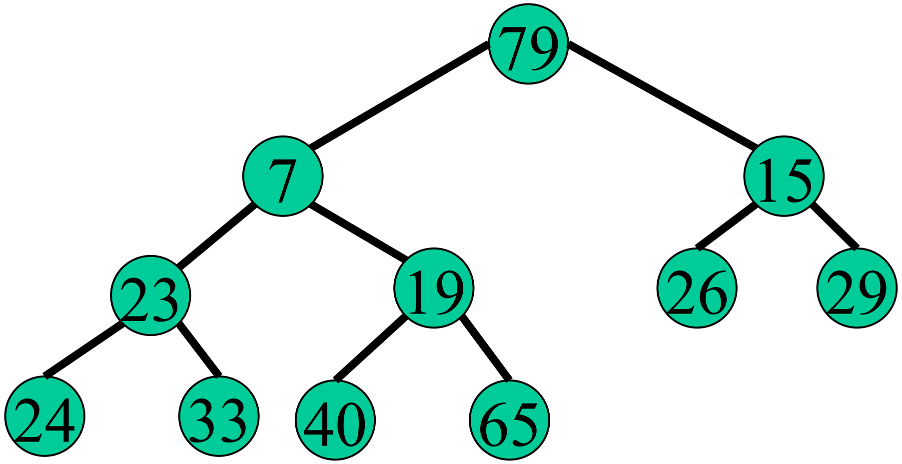
Q										
79	65	15	23	7	26	29	24	33	40	19

Heapify-down(Q,1)



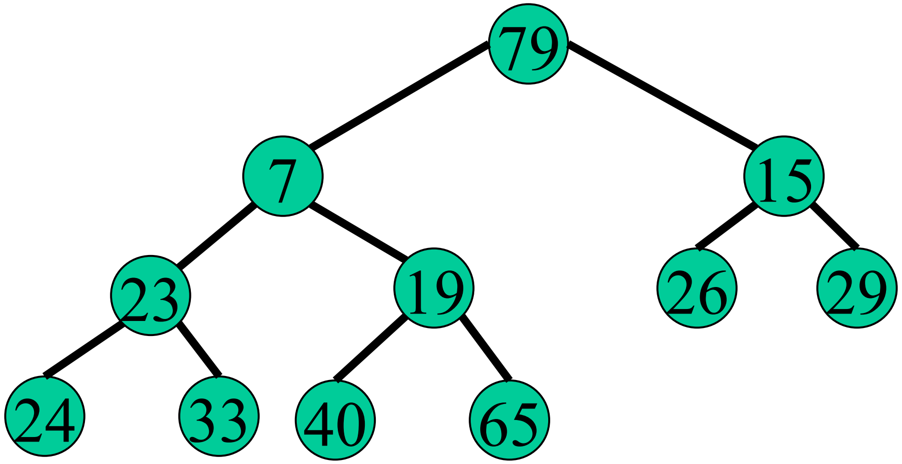
Q	79	7	15	23	65	26	29	24	33	40	19

Heapify-down(Q,1)



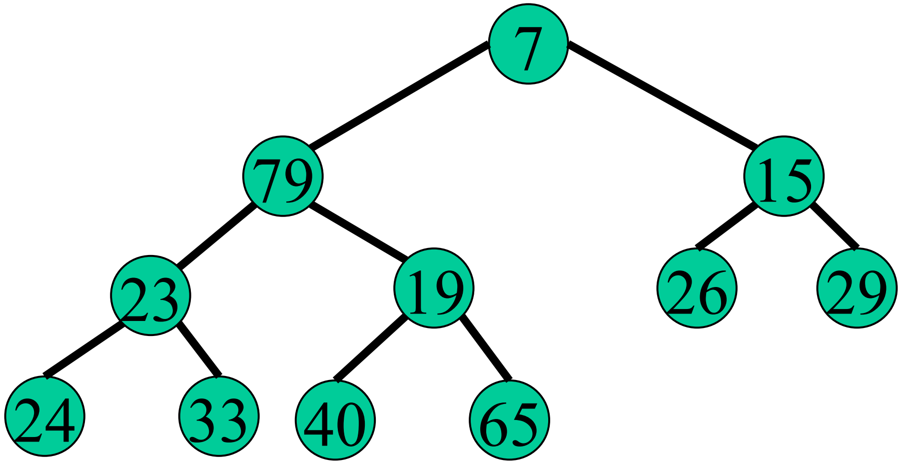
Q											
	79	7	15	23	19	26	29	24	33	40	65

Heapify-down(Q,0)



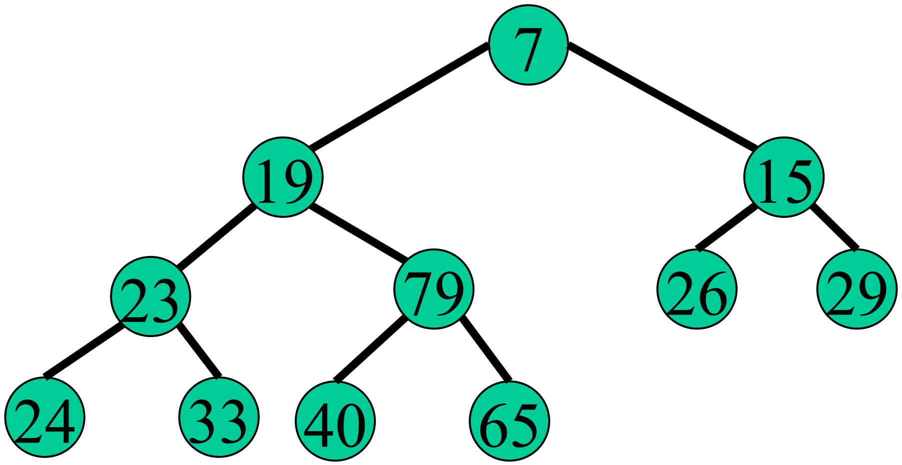
Q											
	79	7	15	23	19	26	29	24	33	40	65

Heapify-down(Q,0)



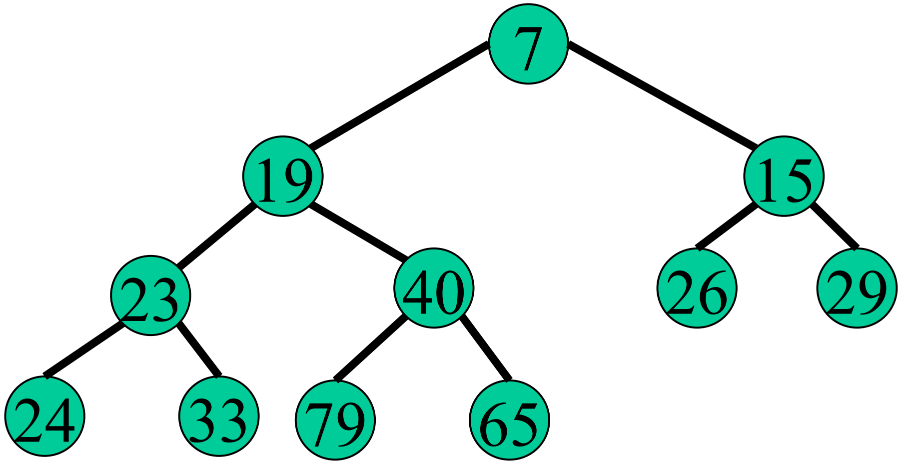
Q	7	79	15	23	19	26	29	24	33	40	65

Heapify-down(Q,0)



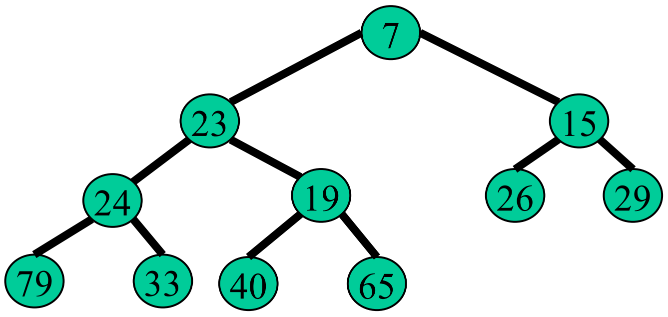
Q										
7	19	15	23	79	26	29	24	33	40	65

Heapify-down(Q,0)



Q	7	19	15	23	40	26	29	24	33	79	65

How much time it takes to build the heap this way ?



We have at most n/2 nodes heapified at height 1

We have at most n/4 nodes heapified at height 2

We have at most n/8 nodes heapified at height 3

$$O\left(\frac{n}{2}\times 1+\frac{n}{4}\times 2+\frac{n}{8}\times 3+\ldots\ldots\ldots\right)=O\left(n\sum_{1}\frac{h}{2^h}\right)=O(n)$$

Summary

- We can build the heap in linear time (we already did this analysis)
- We still have to deletemin the elements one by one in order to sort that will take $O(n\log(n))$

שאלה 1

בערימת מקסימום, החציון נמצא בהכרח:

- א. בשורש.
- ב. בעומק לכל היותר .
- ג. בעומק לכל היותר
- ד. בשתי השכבות הנמוכות ביותר.
- ה. אף אחד מהנ"ל.

תשובה 1

ה. החציון יכול להיות בן ישיר של השורש (למשל אם כל תת עץ ימין גדולים מכל תת עץ שמאל) ויכול גם להיות באחד העלים (אם חצי הערכים הקטנים ביותר נמצאים בעלים). המקסימום חייב להימצא בשכבה התחתונה.

שאלה+תשובה 2

- האם מערך הממוין בסדר הפוך הוא ערימה?

תשובה: כן.

האיבר במקום ה-i גדול מהאיבר ה-$2i+1$ ו-$2i$ (שהם בניו) לכל i.

שאלה 3

ערימה מכילה $N = 2^i - 1$ צמתים, עבור i שלם חיובי כלשהו.

העומק של צומת הינו מספר הקשתות בין השורש ובין הצומת (עומק השורש הינו 0).

הגובה של צומת הינו מספר הקשתות בין הצומת ובין העלה הרחוק ביותר (גובה עלה הינו 0).

יהי D **סכום העומקים של כל הצמתים (כולל עלים)** אזי

א. $D = \Theta(N)$

ב. $D = \Theta(N \log N)$

ג. $D = \Theta(N^2)$

ד. $D = \Theta(N^2 \log N)$

ה. אף אחת מהתשובות הנ"ל אינה נכונה.

*(במקור הופיעה השאלה עם עץ טרינארי מאוזן).

תשובה 3

ב. <u>חסם עליון:</u>

$O(n \lg n)$.

$\Omega(n)$ עלים לכן הסכום הוא גם $\Omega(n \lg n)$

שאלה 4

אותם נתונים כמו בשאלה הקודמת.

יהי H **סכום הגבהים של כל הצמתים (כולל עלים)** אזי

א. $H = \Theta(N)$

ב. $H = \Theta(N \log N)$

ג. $H = \Theta(N^2)$

ד. $H = \Theta(N^2 \log N)$

ה. אף אחת מהתשובות הנ"ל אינה נכונה.

תשובה 4

- **א.** החישוב בדומה לחישוב זמן הריצה של Build-Heap.

$$\sum_{h=0}^{O(\log n)} h \cdot O\left(\frac{n}{2^h}\right) = \Theta(n) \sum_{h=0}^{O(\log n)} \frac{h}{2^h} = \Theta(n) \sum_{h=0}^{O(\log n)} \frac{1.5^h}{2^h}$$

$$= \Theta(n) \sum_{h=0}^{\infty} \left(\frac{3}{4}\right)^h = \Theta(n)$$

הסוף

1	3	5	7
9	11	13	15
17	19	21	23
25	27	29	31

2	3	6	7
10	11	14	15
18	19	22	23
26	27	30	31

4	5	6	7
12	13	14	15
20	21	22	23
28	29	30	31

8	9	10	11
12	13	14	15
24	25	26	27
28	29	30	31

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

ערמות

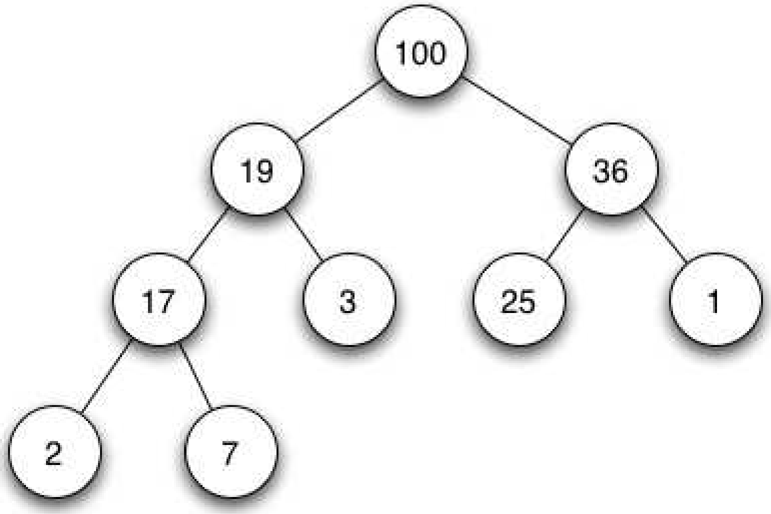
Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci † Heap	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log n	log n	1	1
delete-min	n	log n	log n	log n	log n
decrease-key	n	log n	log n	1	1
delete	n	log n	log n	log n	log n
union	1	n	log n	1	1
find-min	n	1	log n	1	1

n = number of elements in priority queue

† amortized

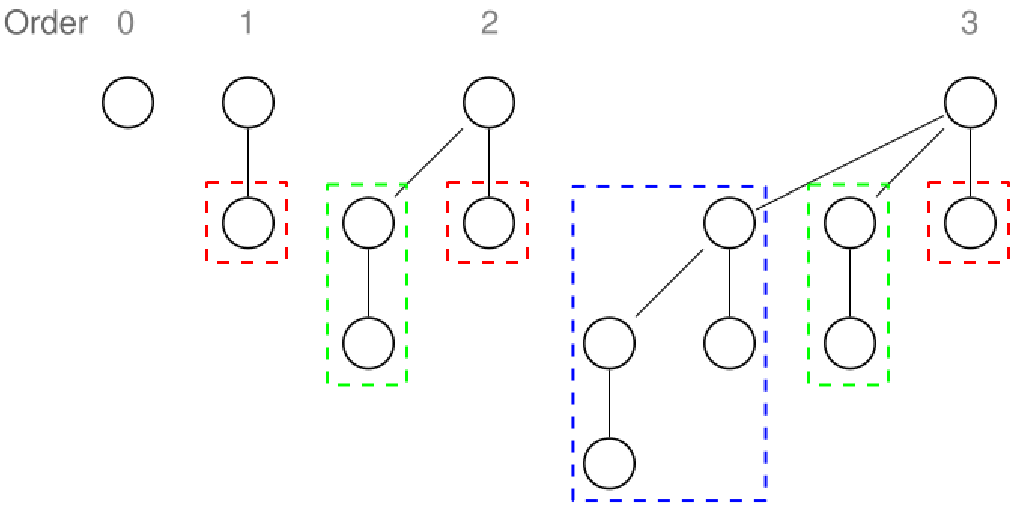
תזכורת: Heaps

- עץ בינארי מלא
- החוק הבסיסי
  - אם צומת B צאצא של צומת A אזי (Key(A)≤Key(B
- הפעולות הנתמכות
  - Find-min
  - Delete-min
  - Decrease-key
  - Insert
  - Merge



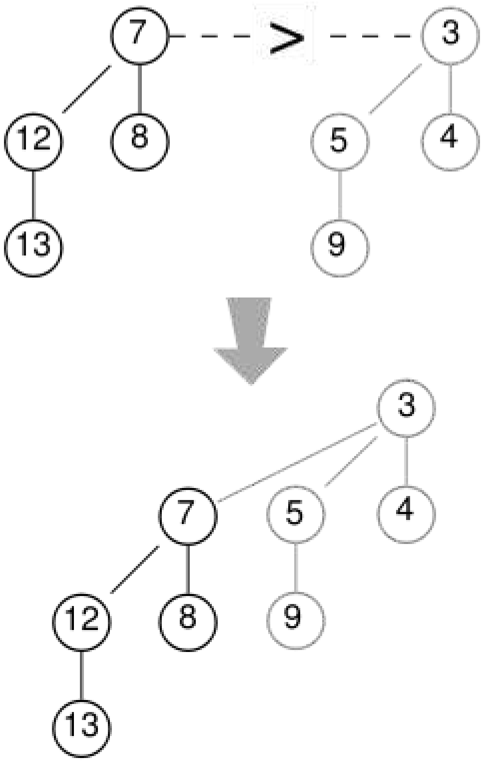
תזכורת: Binomial Heaps

- תחילה נגדיר Binomial Tree:
  - עץ בינומי מדרגה 0 מכיל צומת יחידה
  - עץ בינומי מדרגה k הוא
    - בעל שורש מדרגה k
    - ילדיו הינם עצים בינומיים מדרגות k-1,k-2,…,0 (בסדר זה)
    - עץ בינומי מדרגה k מכיל 2^k צמתים והינו בגובה k



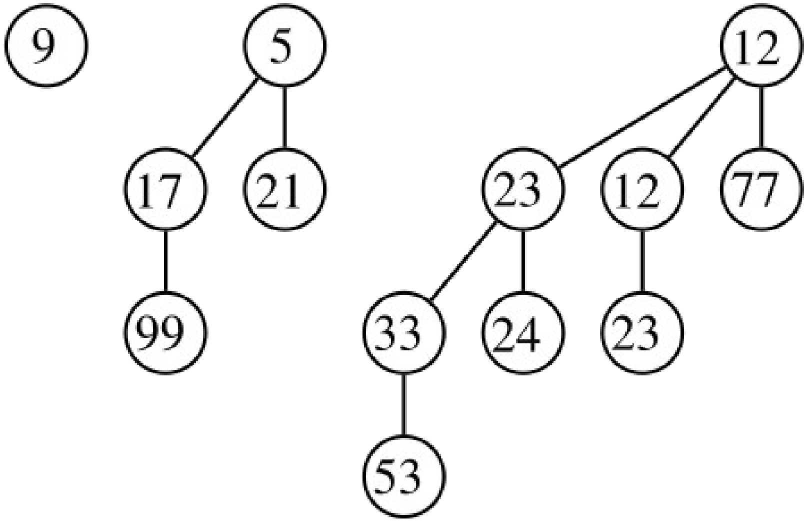
תזכורת: Binomial Heaps

- איחוד שני עצים בינומיים מסדר k-1
  - ניצור עץ בינומי מסדר k ע"י תליית אחד העצים כבן השמאלי ביותר של העץ השני



תזכורת: Binomial Heaps

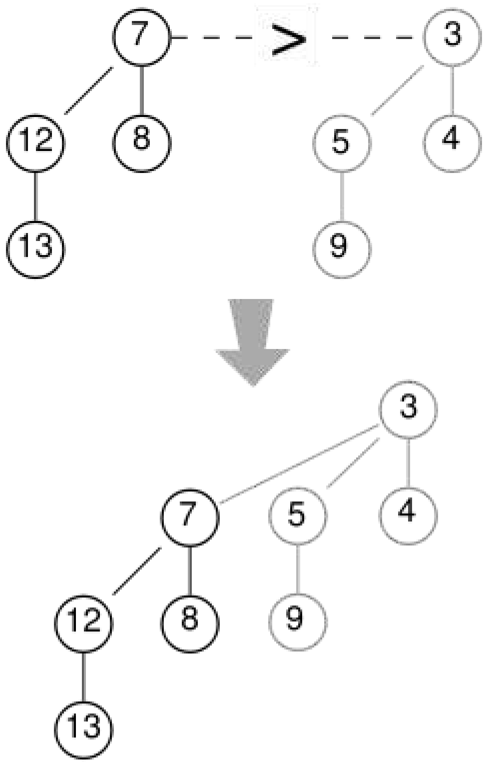
- הגדרת Binomial Heap
  - סט עצים בינומיים המקיימים את התכונות הבאות
    - כל עץ מקיים את תכונת minimum-heap (כל צאצא גדול מההורה שלו)
    - עבור כל סדר k של עץ בינומי, יש 0 או 1 עצים כאלו ב-heap



Binomial Heaps :תזכורת

- פעולת Merge של שני עצים מדרגה k

```
function mergeTree(p, q)
    if p.root <= q.root
        return p.addSubTree(q)
    else
        return q.addSubTree(p)
end
```



תזכורת: Binomial Heaps

- פעולת Insert
  - ניצור heap חדש המכיל את האבר החדש, ונבצע merge בין שני ה-heaps
- פעולת minimum
  - עלינו לחפש את הערך המינימלי מבין שורשי העצים בheap
- פעולת delete-min
  - מצא את האבר ומחק אותו
  - הפוך את בניו ל-binomial heap ומזג את שני ה-heaps
- פעולת Decrease-min
  - בדומה לפעולות ב-heap רגיל
- פעולת Delete
  - שנה את הערך ל-∞ ובצע delete-min

הדגמה באנימציה של Binomial Heaps: http://www.cse.yorku.ca/~aaw/Sotirios/BinomialHeap.html

Properties of binomial trees

1) $|B_k| = 2^k$

2) $degree(root(B_k)) = k$

3) $depth(B_k) = k$

==> The degree and depth of a binomial tree with at most n nodes is at most log(n).

Define the rank of B_k to be k

Binomial heaps (ops cont.)

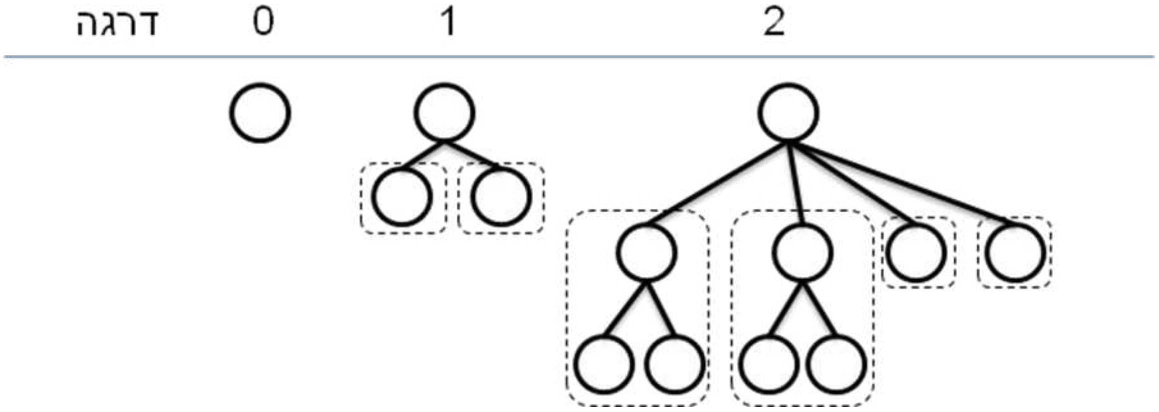
Basic operation is meld(h1,h2):

Like addition of binary numbers.

	B_5	B_4		B_2	B_1		
h1:		B_4	B_3		B_1	B_0	
h2:		B_4	B_3			B_0	+
	B_5	B_4		B_2			

נגדיר עצים בינומיים "שמנים" בצורה הבאה:

- עץ בינומי "שמן" מדרגה 0 מכיל צומת אחד בלבד.
- עץ בינומי "שמן" מדרגה k ניתן לבנות משלושה עצים בינומיים "שמנים" מדרגה k-1 כאשר נתלה שניים מהם על העץ השלישי.



תארו מבנה נתונים אנלוגי לערמה בינומית (binomial heap) המשתמש בעצים "שמנים".

ערמה בינומית "שמנה" מוגדרת בצורה הבאה:
בערמה יש n עצים בינומיים "שמנים" כאשר יש לכל היותר 2 עצים מדרגה k, לכל k. שורשי העצים מחוברים ביניהם ברשימה מקושרת. הפעולות מוגדרות בדומה לערמה בינומית רגילה וכאשר יש צורך לעשות merge בין העצים (יש יותר משני עצים מדרגה k) בונים משלושה עצים מדרגה k עץ יחיד מדרגה k+1.

תארו פעולת meld של שתי ערמות בינומיות "שמנות" שהגדרתם בסעיף הקודם.

פעולת meld הינה מיזוג בין שתי ערמות בינומיות "שמנות". הדבר מקביל לחיבור שני מספרים בבסיס 3 (כמו שבערמות בינומיות הדבר היה מקביל לחיבור מס' בבסיס 2). נתחיל מדרגה 0 ונוסיף את העצים בערמה אחת לשנייה. אם יש סה"כ 3 או יותר עצים מדרגה 0 בערמה החדשה נמזג שלושה ליצירת עץ יחיד מדרגה 1. כעת נעבור לדרגה הבאה (1) ונמשיך כך...

- הראו שאין מימוש heap שתומך בפעולות extract-min ו-insert בזמן $o(\log n)$ תחת הנחות מודל ההשוואות.
- פתרון
  - נבצע n פעולות הכנסה בזמן $n \cdot o(\log n)$
  - נבצע n פעולות הוצאת מינימום בזמן $n \cdot o(\log n)$
  - סה"כ $o(n \cdot \log n)$
- למדנו כי מיון n אברים הוא $\Omega(n \cdot \log n)$, סתירה

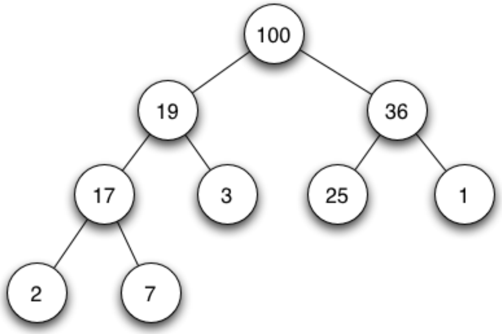
תרגיל 3 – Median Heap

- ממשו מבנה נתונים התומך בפעולות
  - insert בזמן O(logn)
  - extract-median בזמן O(logn)
  - find-median בזמן O(1)

2	4	5	7	8	12	14	15	20

תרגיל 3 - פתרון

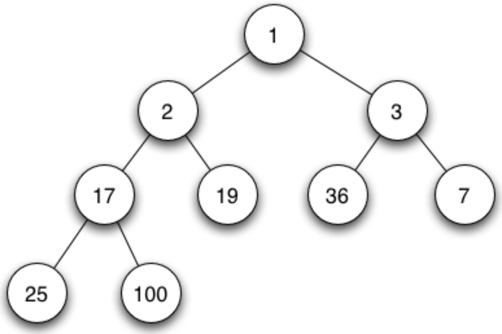
Max-heap



האברים הקטנים
(עד החציון)

+

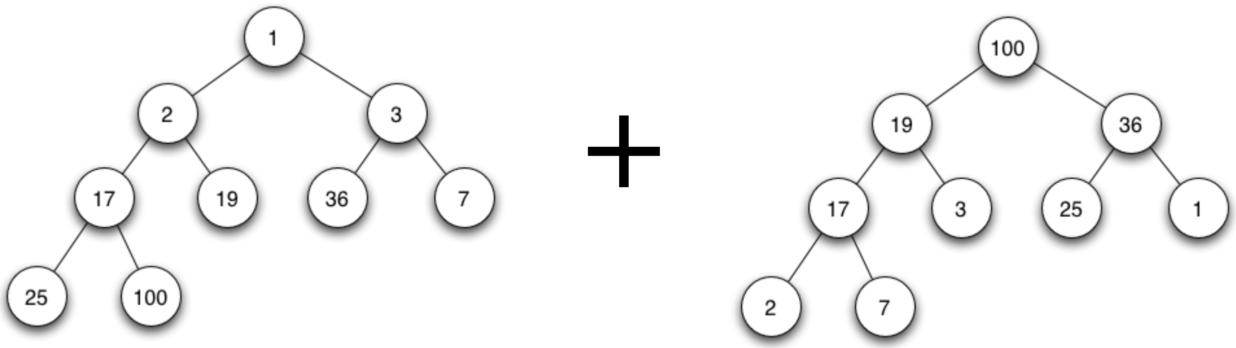
Min-heap



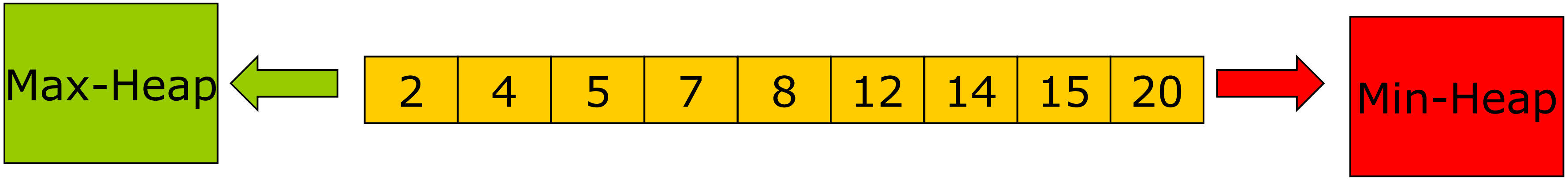
האברים הגדולים
(מהחציון)

תרגיל 3 - פתרון

Min-heap Max-heap



- נשתמש ב-max-heap ו-min-heap
- n/2 הערכים הגדולים ביותר יישמרו ב-max-heap
- השאר יישמרו ב-min-heap
- החציון תמיד נמצא בשורש של אחד מהם



תרגיל 3 - פתרון

- Find-median
  - If (size(minheap)>size(maxheap))
    - return getmin(minheap)
  - Else
    - return getmax(maxheap)

O(1)

- Insert(x)
  - If (x<getmin(minheap))
    - Insert(maxheap,x)
  - Else
    - Insert(minheap,x)
  - If (abs(size(minheap)-size(maxheap))>1)
    - Balance heaps (move root from bigger heap to smaller heap)

O(logn)

- Extract-Median
  - Extract median from the max-heap or min-heap…

O(logn)

תארו מבנה נתונים התומך בפעולות הבאות על קבוצה S מתחום סדור מלא

- Median(S): מחזיר את החציון ב-S (אם מס' אברים זוגי – חציון תחתון)
- Min(S) : מחזיר את האבר הקטן ביותר ב-S
- Max(S) : מחזיר את האבר הגדול ביותר ב-S
- Insert(x,S) : מוסיף אבר x ל-S
- Delete(x,S) מניחים ש-x נמצא לפני הפעולה ב-S. הפעולה מוציאה את x מ-S

על הפעולות median,min,max לקחת O(1) זמן במקרה הגרוע, ועל הפעולות insert ו-delete לקחת O(logn) זמן במקרה הגרוע.

שומרים median-heap כמו שלמדנו בכתה. כלומר, שתי ערימות: אחת ערימת-מקסימום ואחת ערימת מינימום, כאשר כל איבר בערימת המינימום גדול מכל איבר בערימת המקסימום, ובנוסף הגודל של ערימת המקסימום שווה או קטן בבדיוק 1 מהגודל של ערימת המינימום. בפעולת insert או delete מכניסים/מוציאים את האיבר מ-/ל- ערימה המתאימה, ואם צריך מעבירים איבר מערימה אחת לאחרת כדי לאזן את הגדלים. ה-median תמיד יהיה האיבר המינימלי בערימת המינימום. בנוסף, מחזיקים ערימת מינימום וערימת מקסימום שמחזיקות את כל האיברים, כדי לענות על שאילתות ה-min וה-max

Motivation

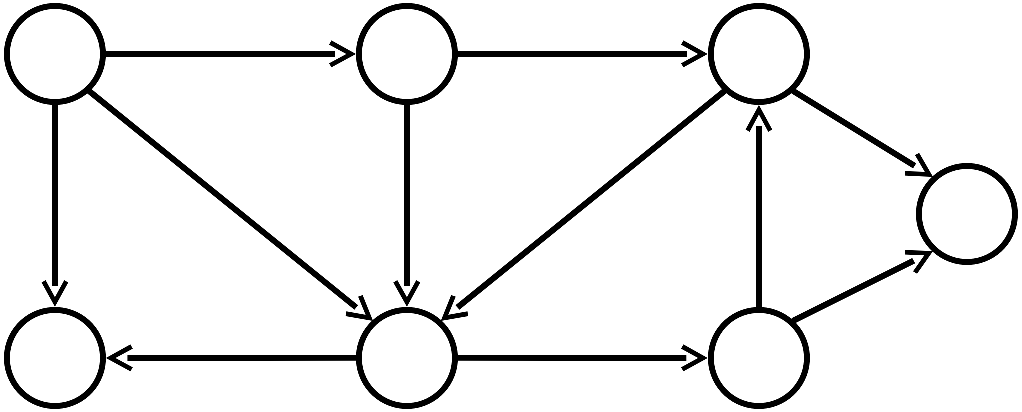
- Dijkstra’s algorithm for single source shortest path

- Prim’s algorithm for minimum spanning trees

Motivation

- Want to find the shortest route from New York to San Francisco
- Model the road-map with a graph

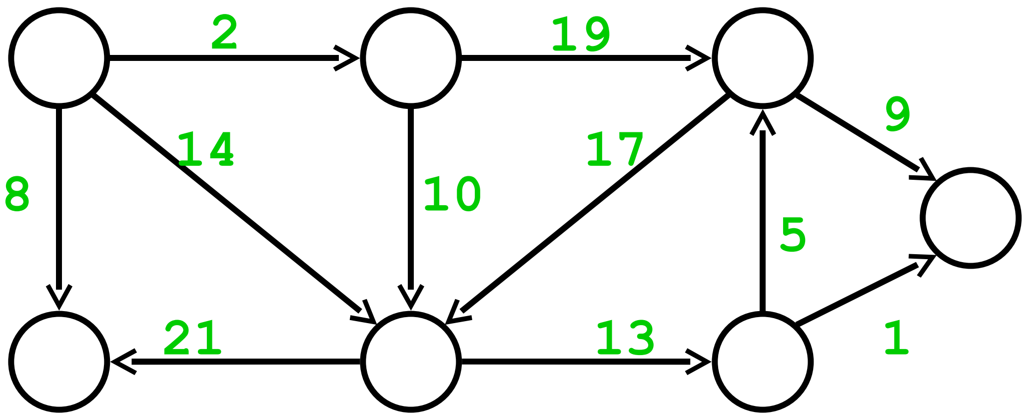
A Graph G=(V,E)



V is a set of vertices

E is a set of edges (pairs of vertices)

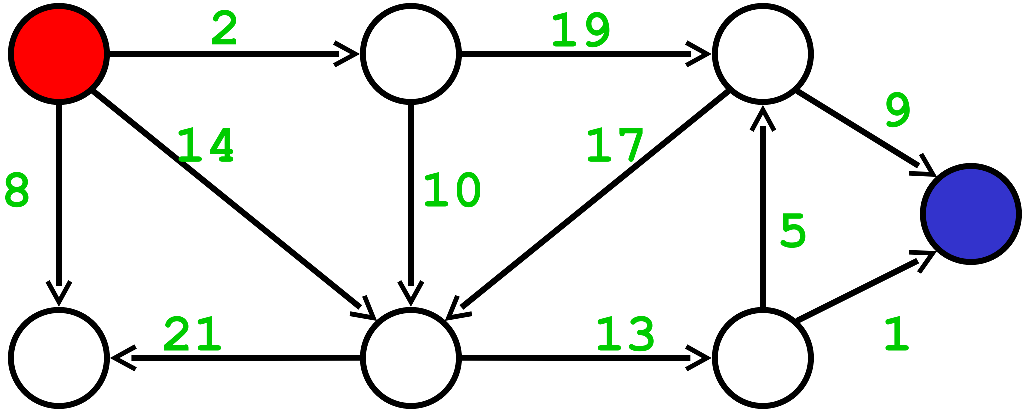
Model driving distances by weights on the edges



V is a set of vertices

E is a set of edges (pairs of vertices)

Source and destination



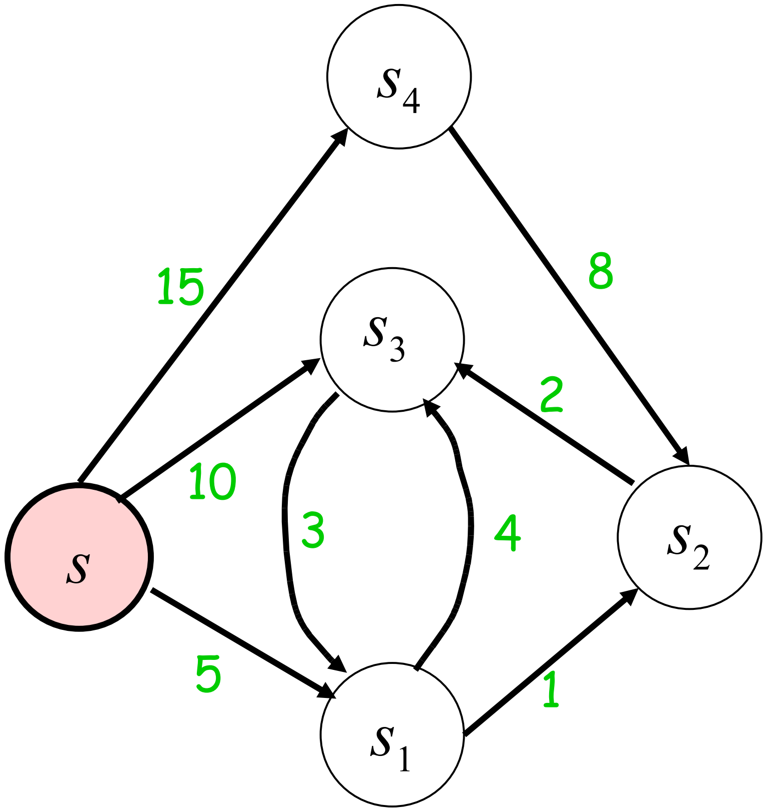
V is a set of vertices

E is a set of edges (pairs of vertices)

Dijkstra's algorithm

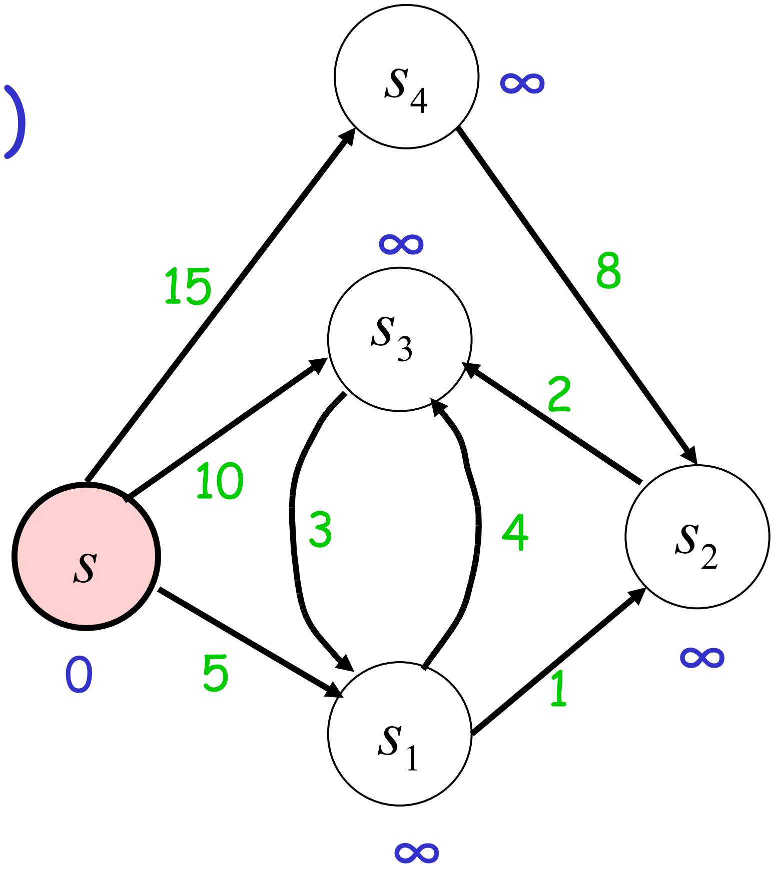
- Assume all weights are non-negative
- Finds the shortest path from some fixed vertex s to every other vertex

Example



Example

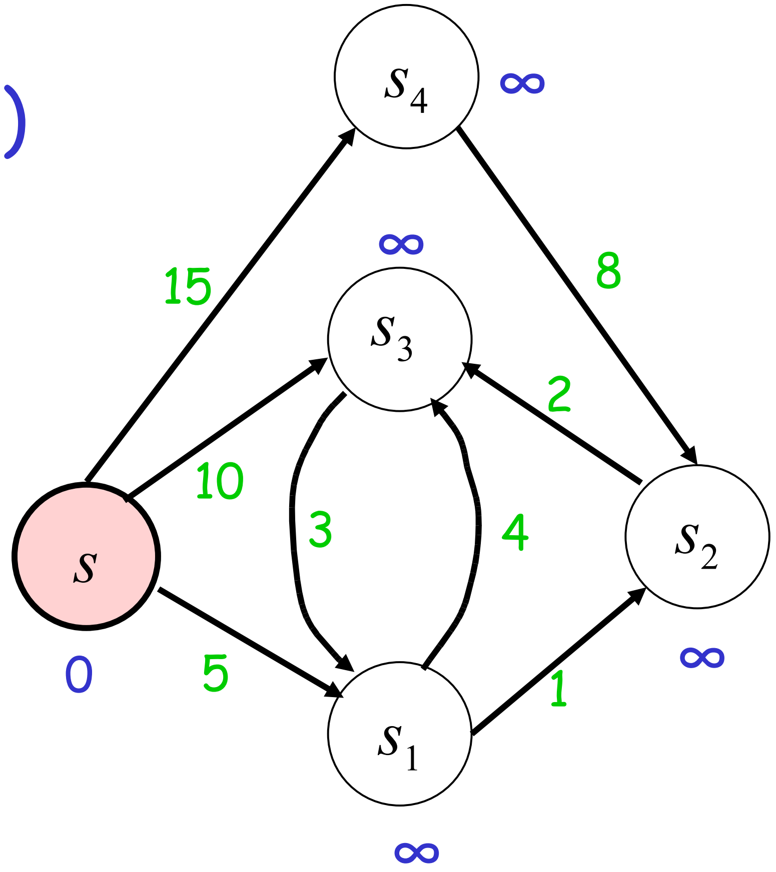
Maintain an upper bound d(v) on the shortest path to v



Maintain an upper bound d(v) on the shortest path to v

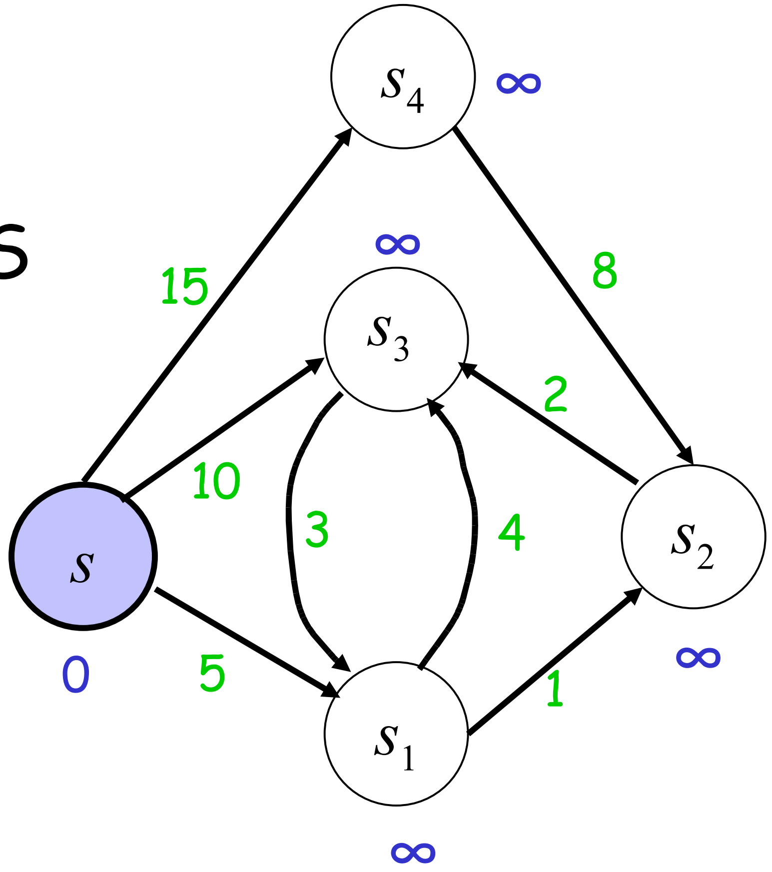
A node is either scanned (in S) or labeled (in Q)

Initially S = ∅ and Q = V



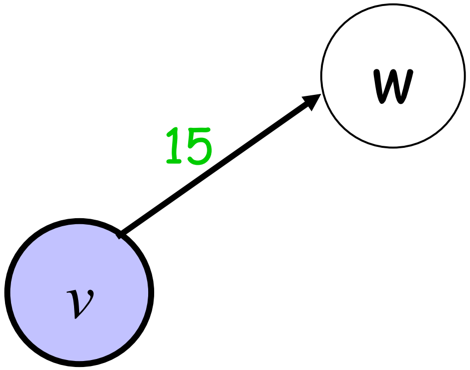
Initially S = ∅ and Q = V

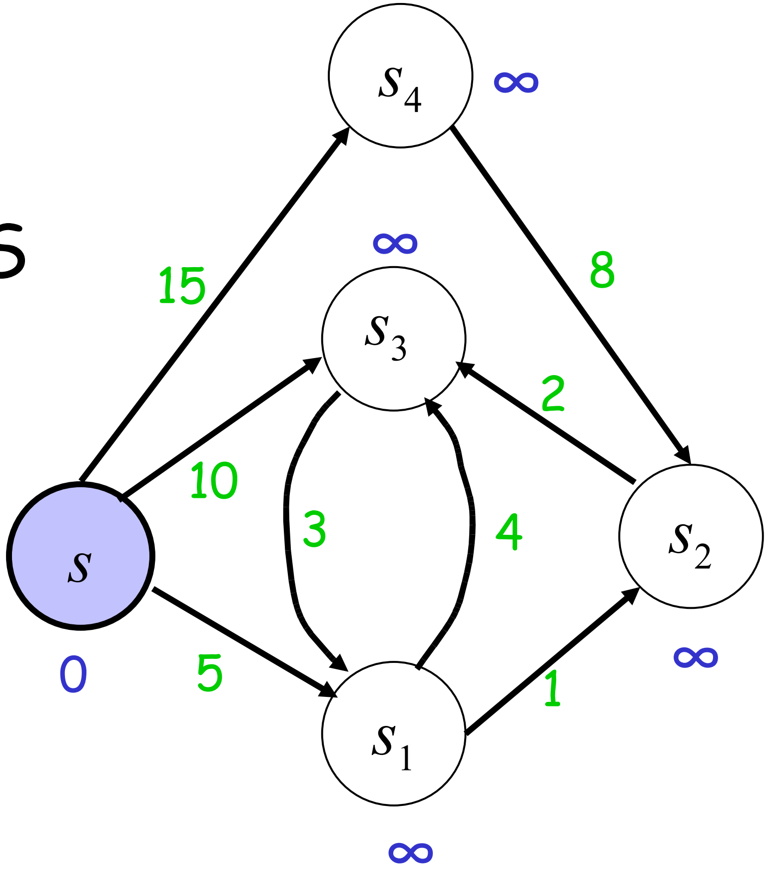
Pick a vertex v in Q with minimum d(v) and add it to S



Initially S = ∅ and Q = V

Pick a vertex v in Q with minimum d(v) and add it to S





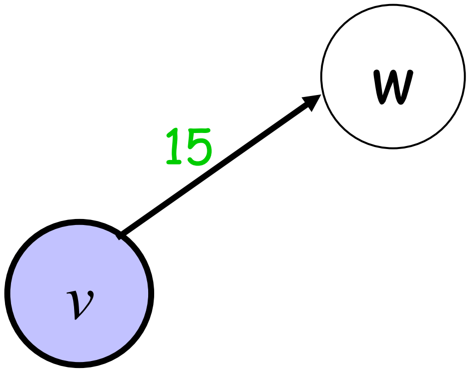
For every edge (v,w) where w in Q relax(v,w)

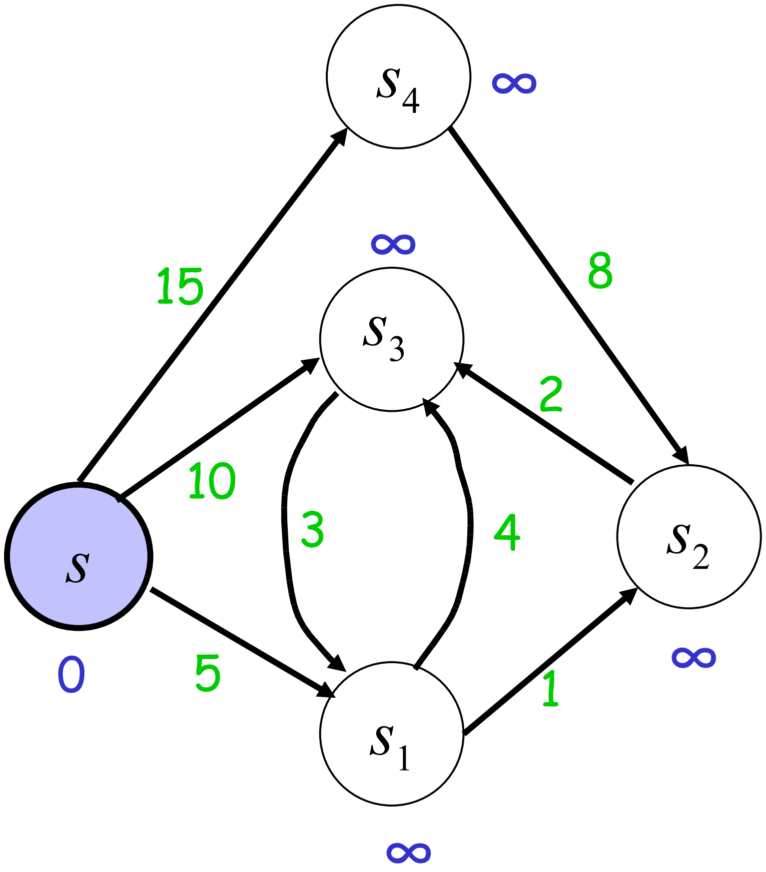
Relax(v,w)

If d(v) + w(v,w) < d(w) then

d(w) ← d(v) + w(v,w)

π(v) ← w

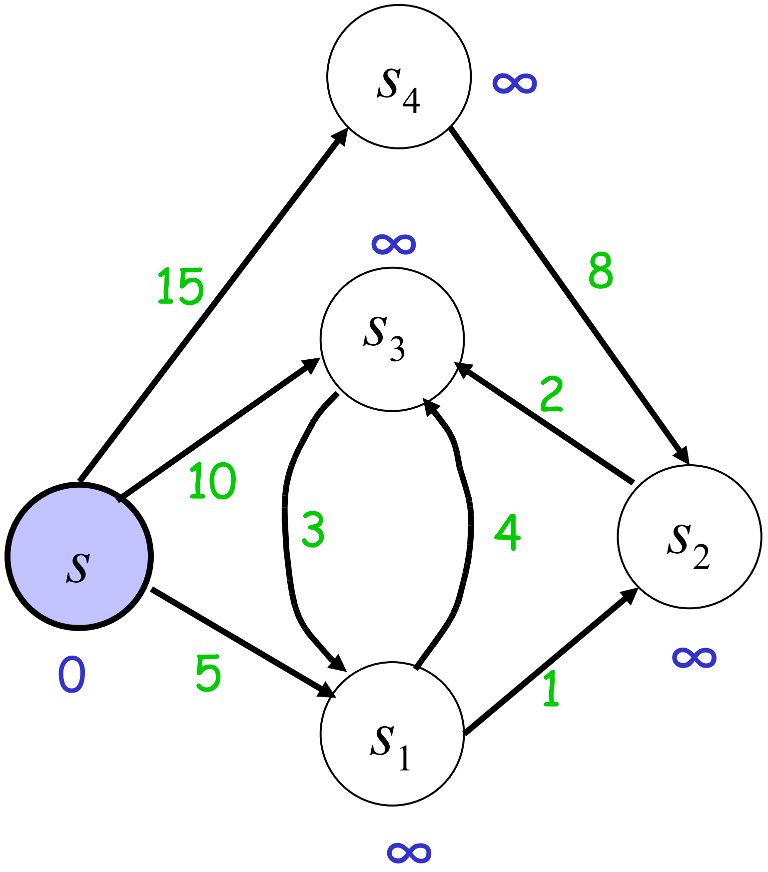




For every edge (v,w) where w in Q relax(v,w)

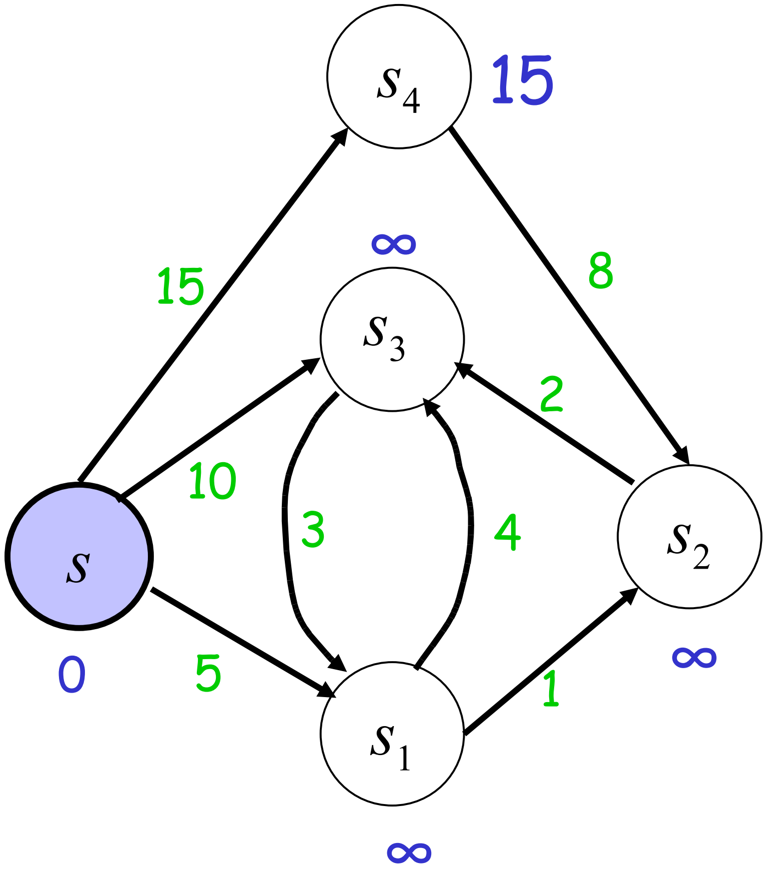
S = {s}

Relax(s,s_4)



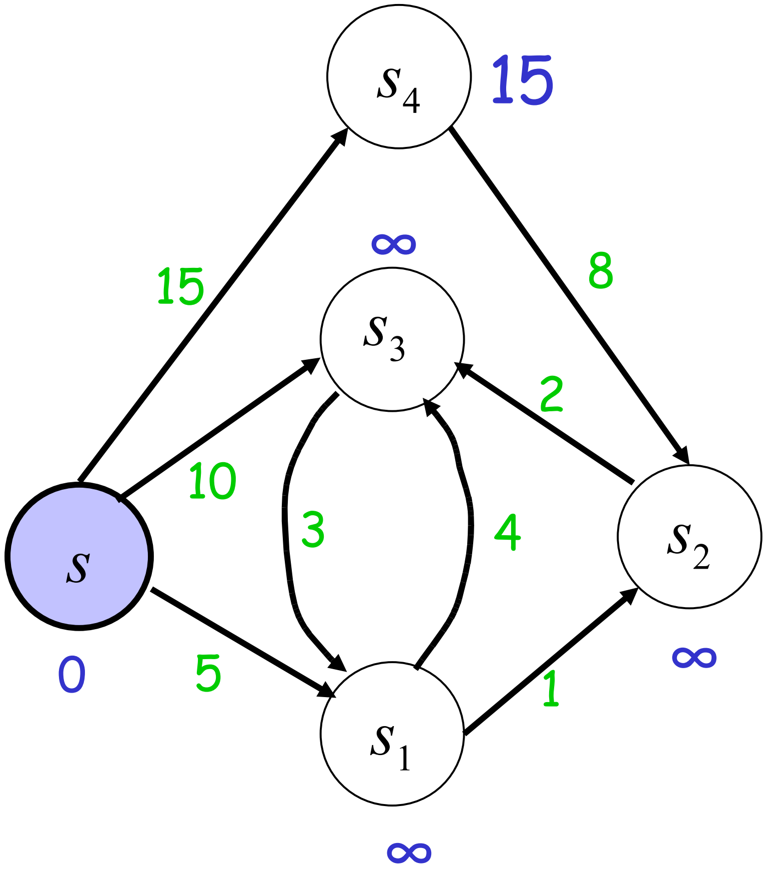
$S = \{s\}$

Relax(s,s_4)



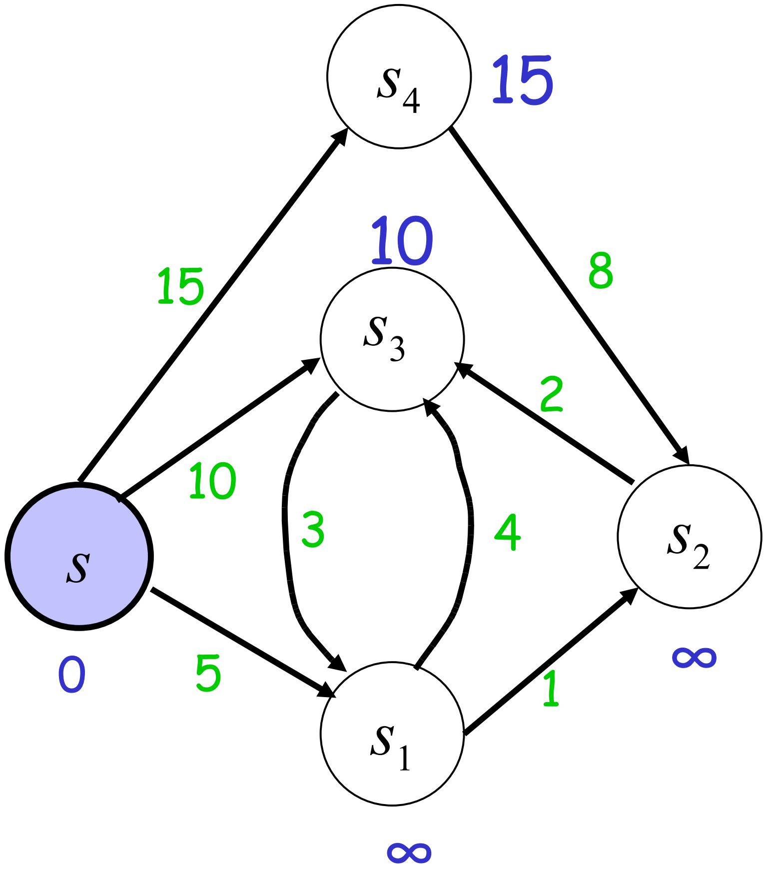
S = {s}

Relax(s,s_4)
Relax(s,s_3)



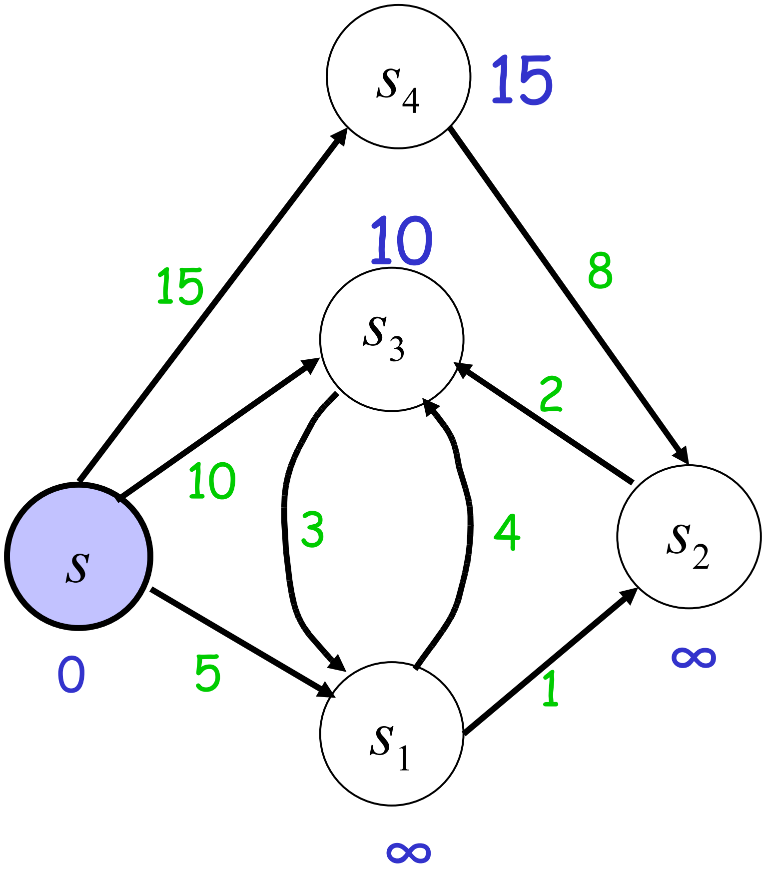
$S = \{s\}$

$\text{Relax}(s, s_4)$
$\text{Relax}(s, s_3)$



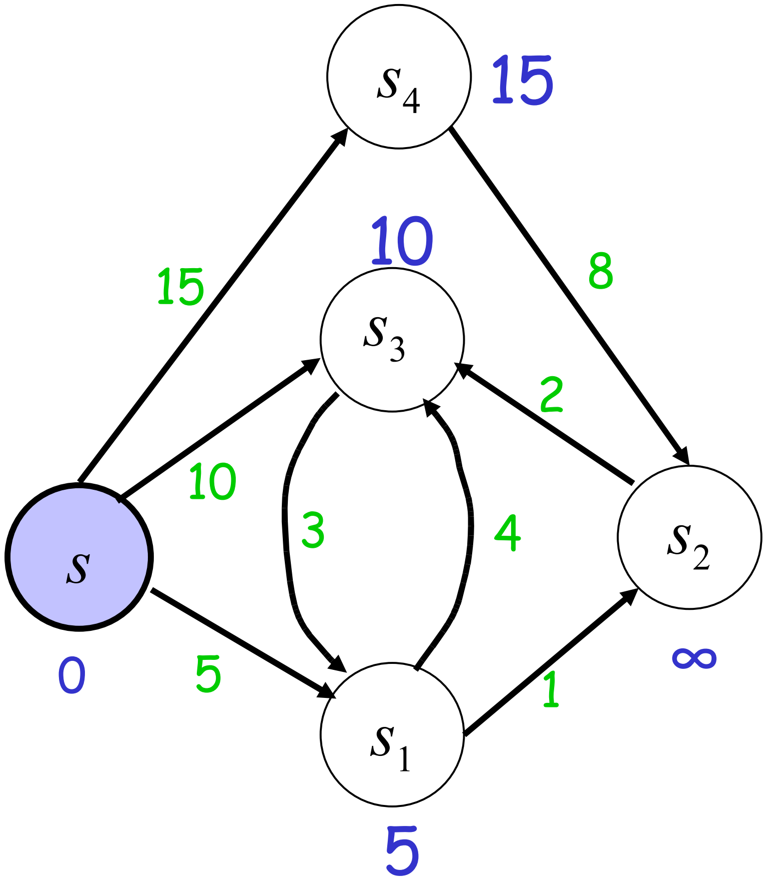
$S = \{s\}$

$Relax(s, s_4)$
$Relax(s, s_3)$
$Relax(s, s_1)$



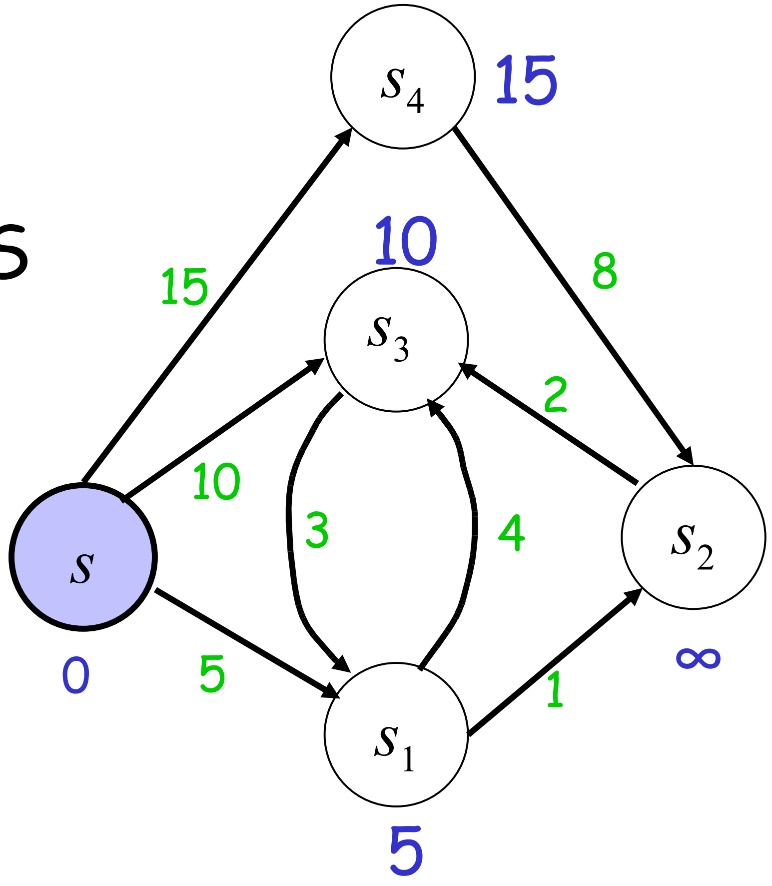
$S = \{s\}$

Relax(s,s_4)
Relax(s,s_3)
Relax(s,s_1)



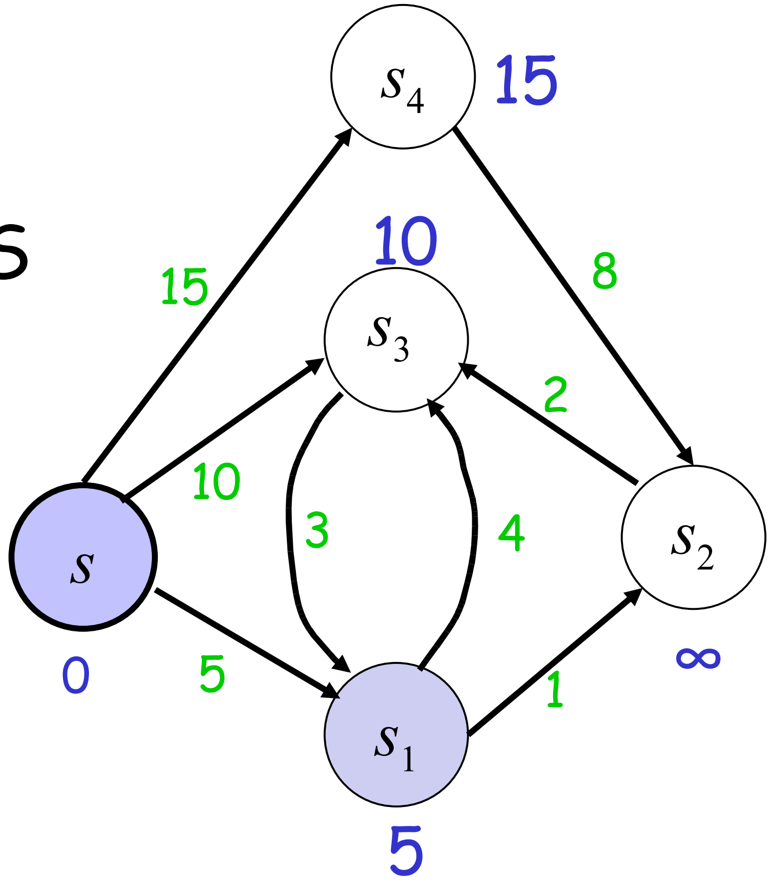
S = {s}

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1\}$

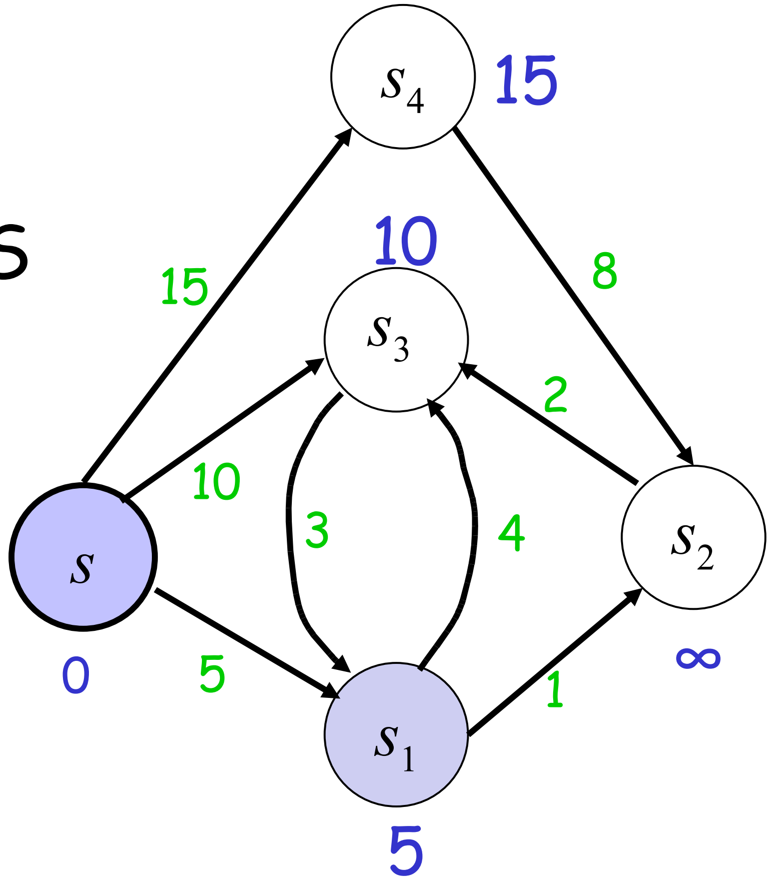
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

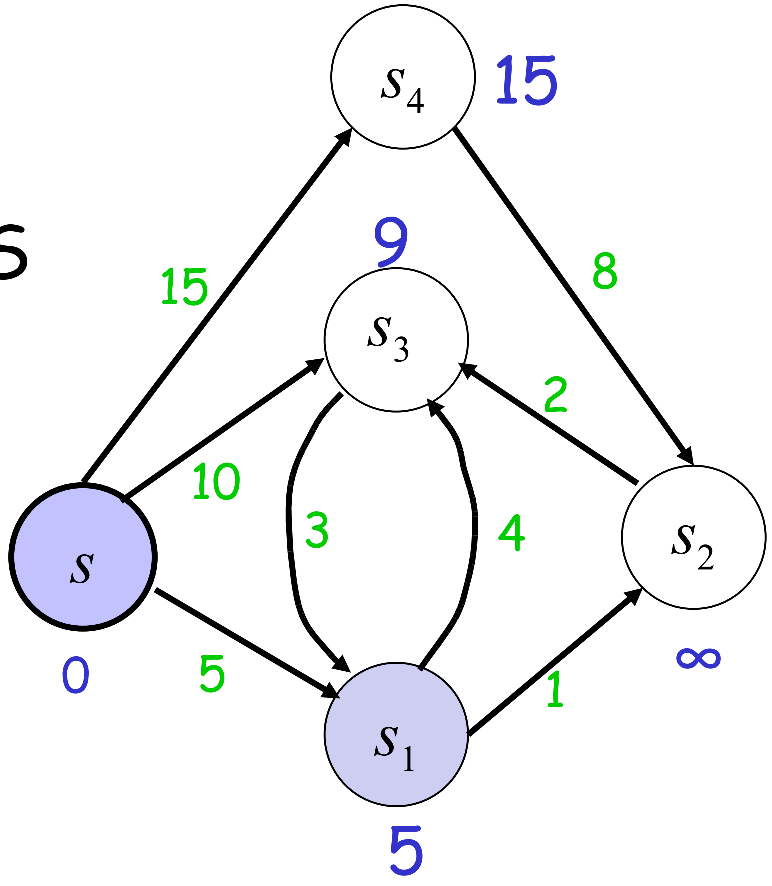
Relax(s_1,s_3)



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

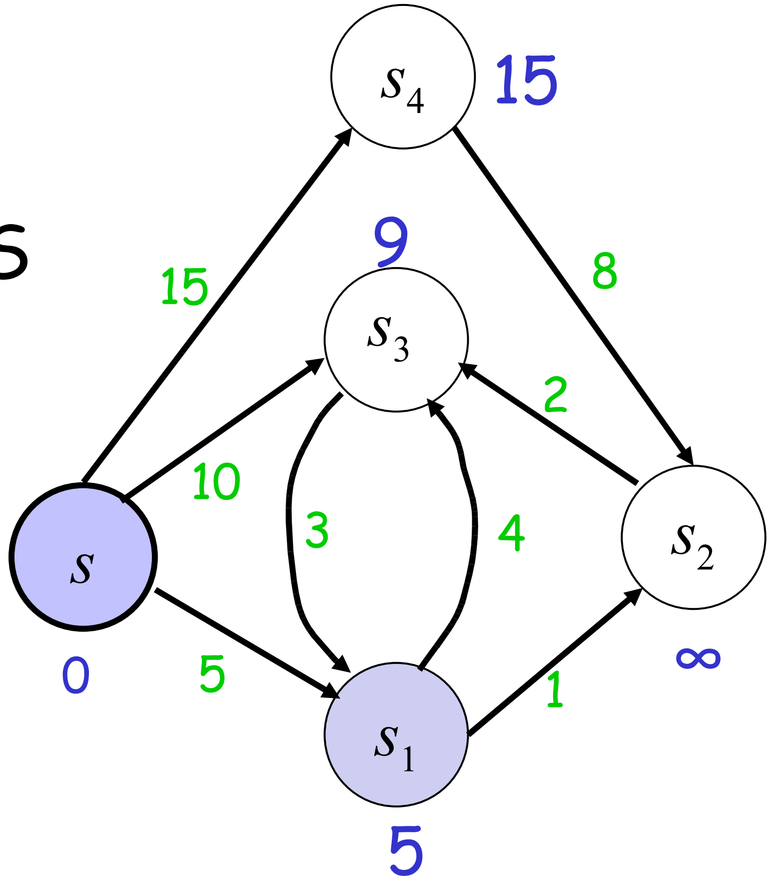
Relax(s_1,s_3)



$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S

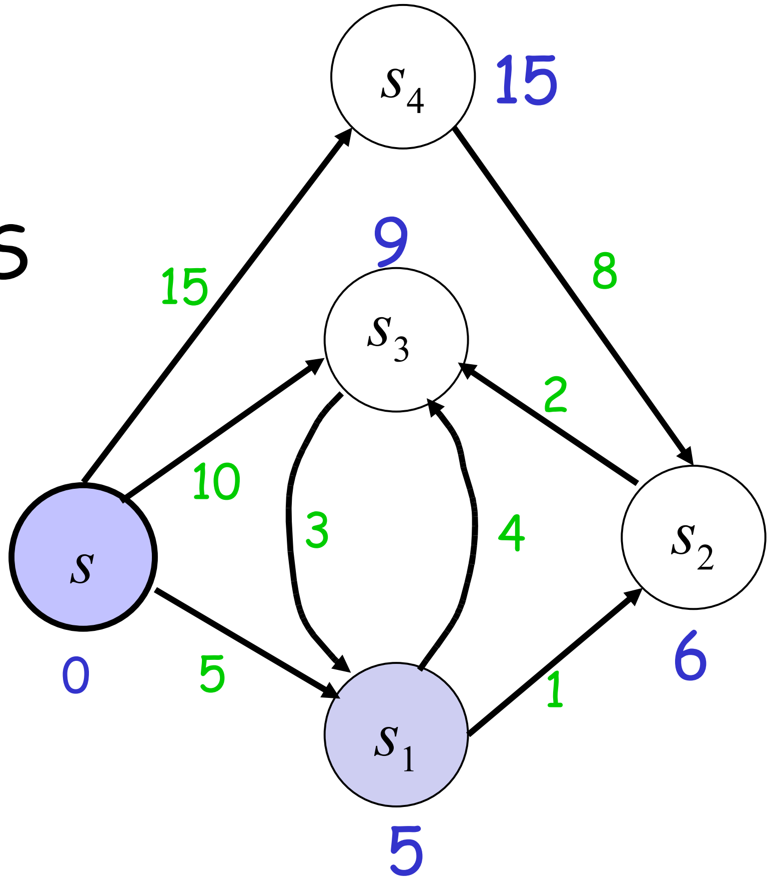
Relax(s_1,s_3)
Relax(s_1,s_2)



$S = \{s, s_1\}$

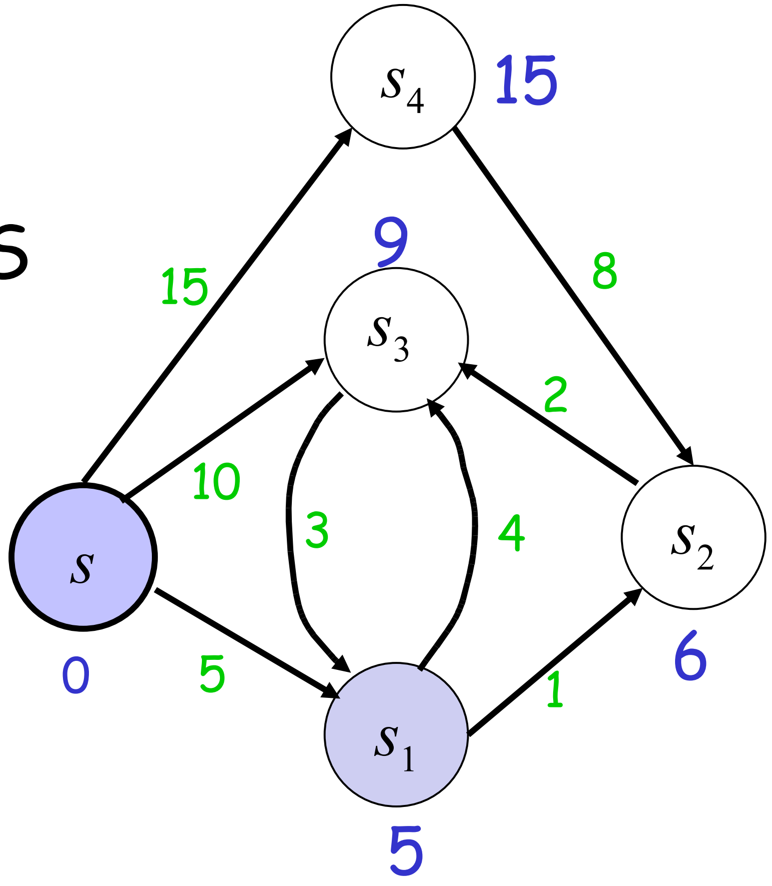
Pick a vertex v in Q with minimum d(v) and add it to S

Relax(s_1,s_3)
Relax(s_1,s_2)



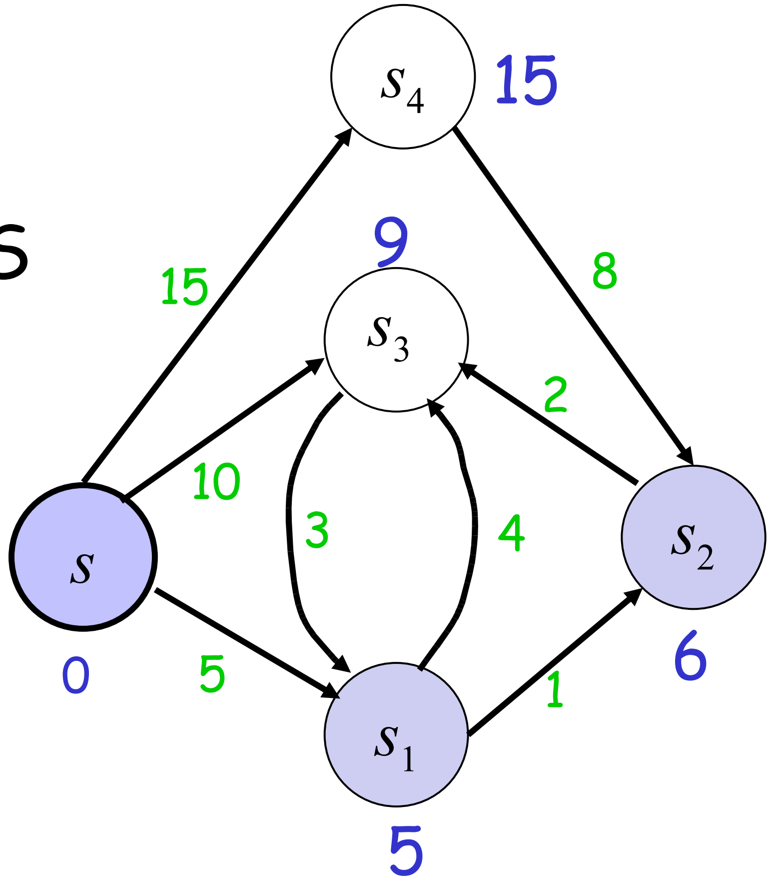
$S = \{s, s_1\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2\}$

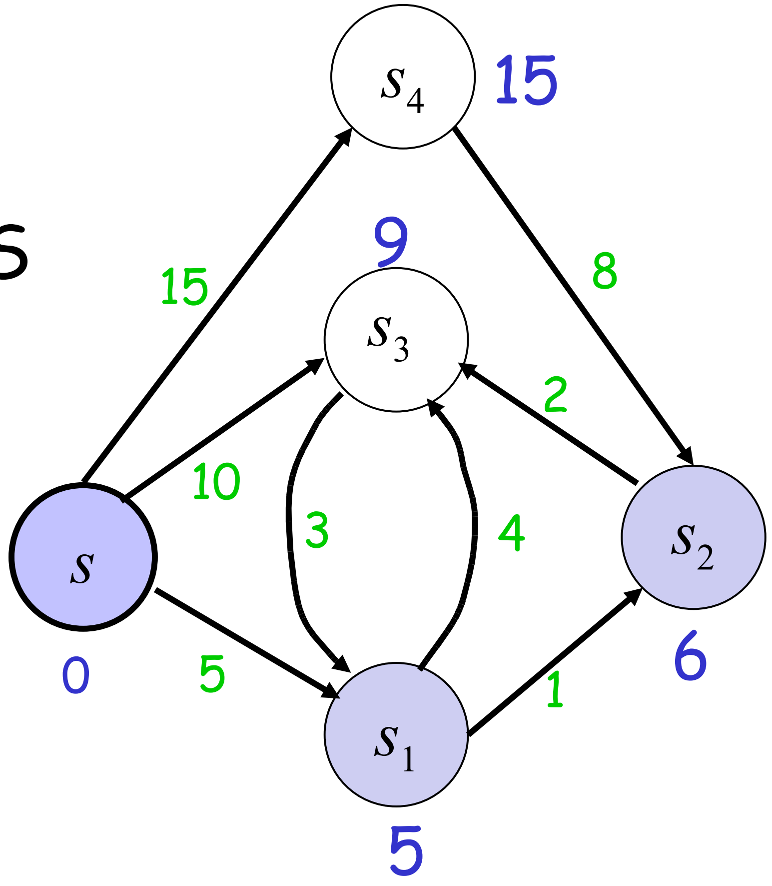
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2\}$

Pick a vertex v in Q with minimum d(v) and add it to S

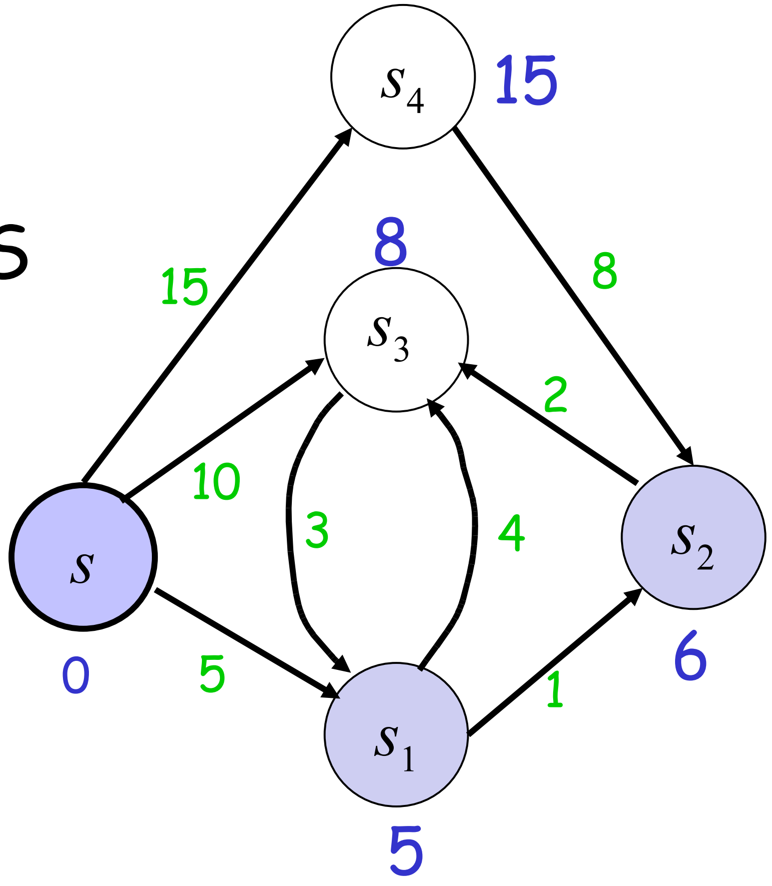
Relax(s_2,s_3)



$S = \{s, s_1, s_2\}$

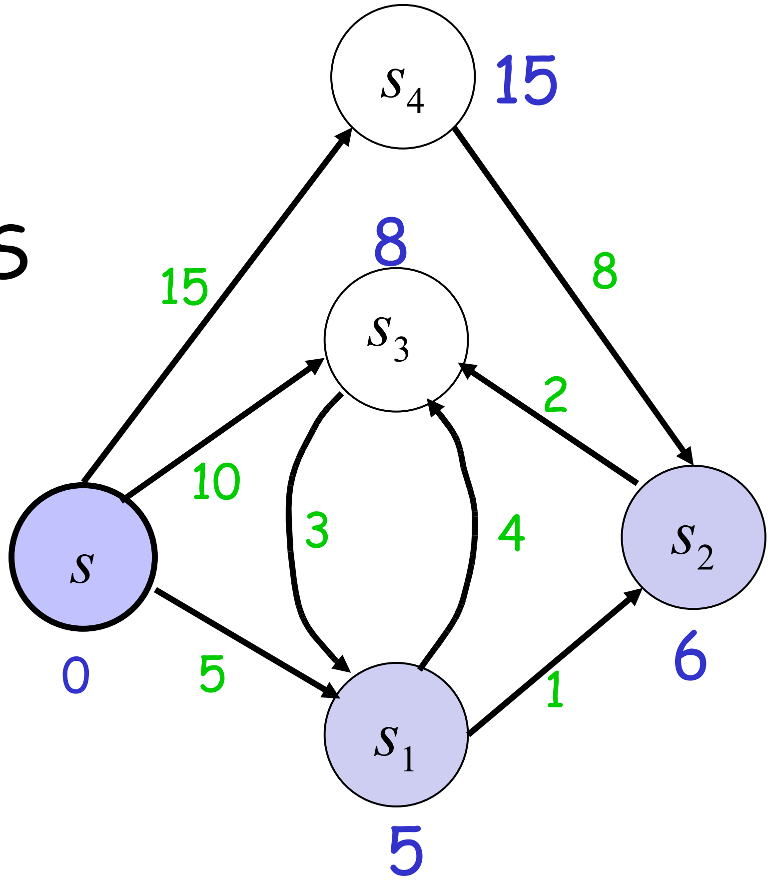
Pick a vertex v in Q with minimum d(v) and add it to S

Relax(s_2,s_3)



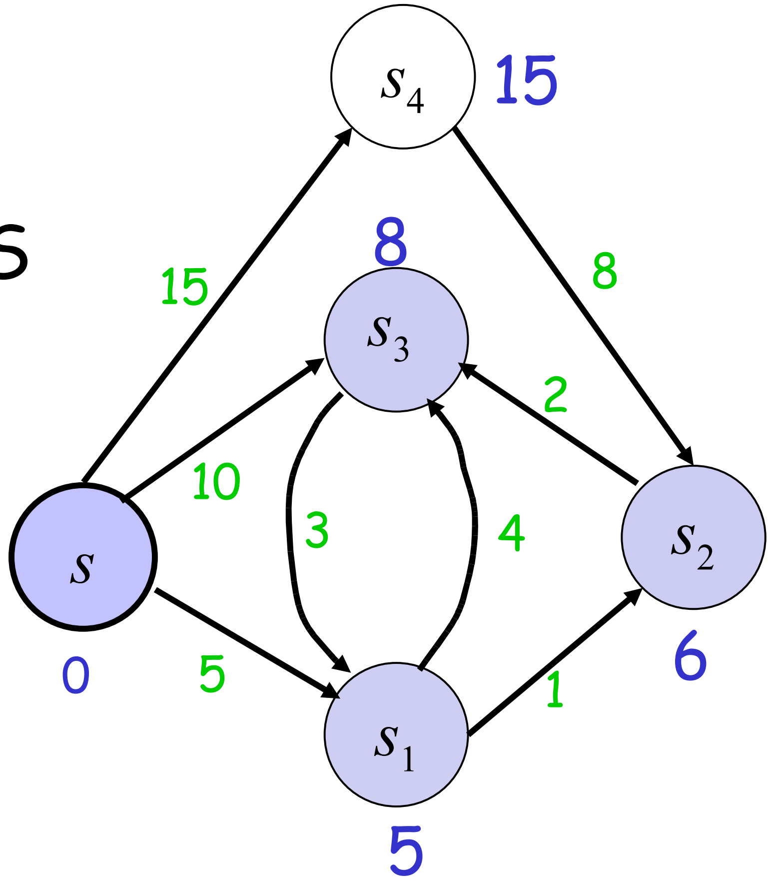
$S = \{s, s_1, s_2\}$

Pick a vertex v in Q with minimum d(v) and add it to S



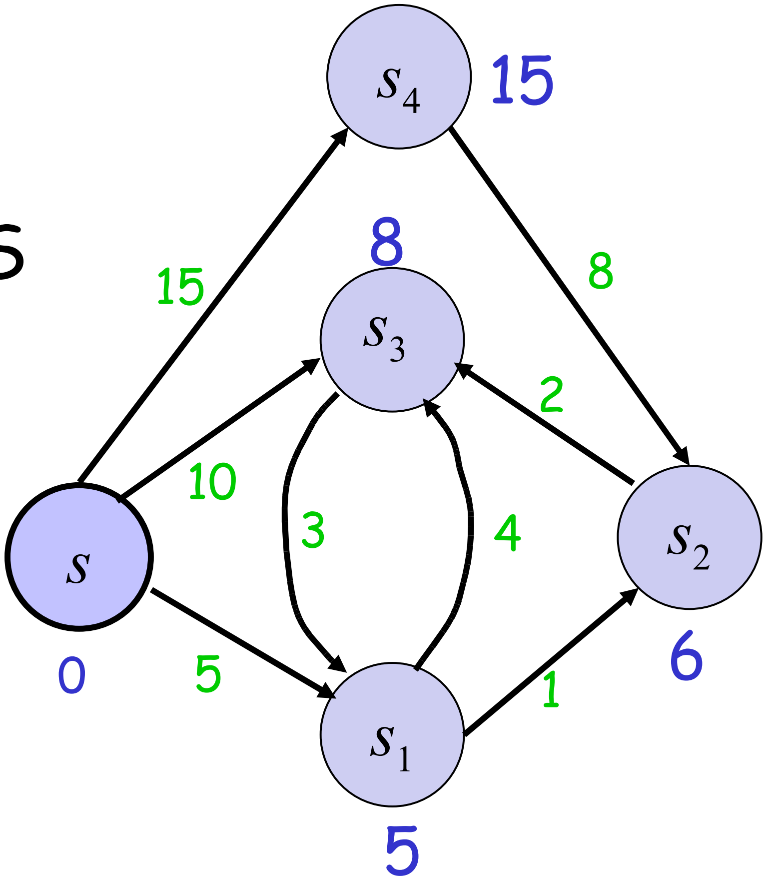
$S = \{s, s_1, s_2, s_3\}$

Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2, s_3, s_4\}$

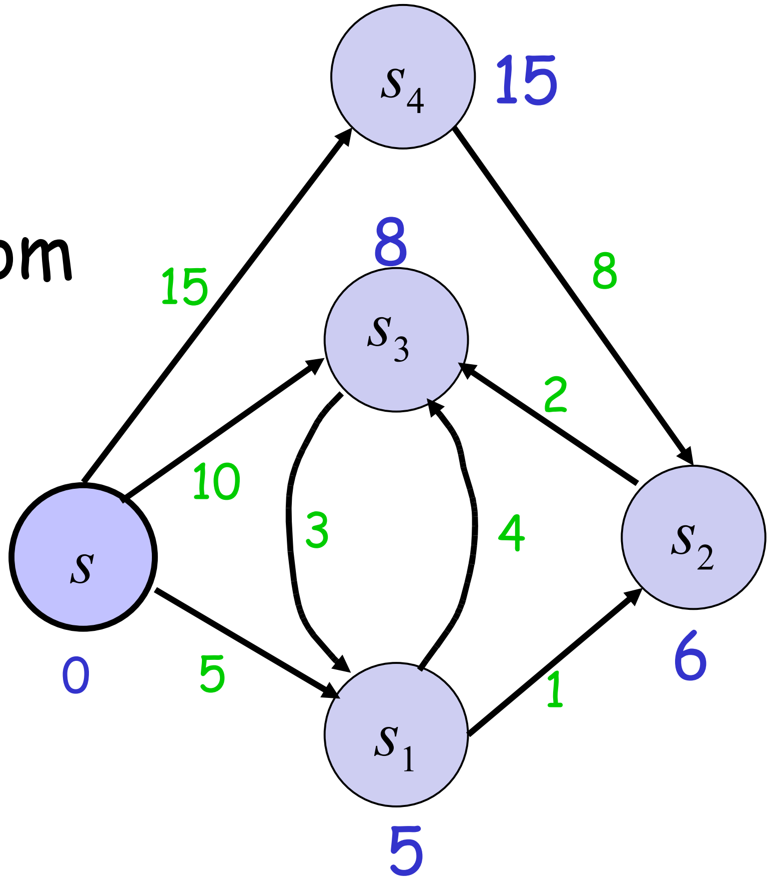
Pick a vertex v in Q with minimum d(v) and add it to S



$S = \{s, s_1, s_2, s_3, s_4\}$

When $Q = \varnothing$ then the d() values are the distances from s

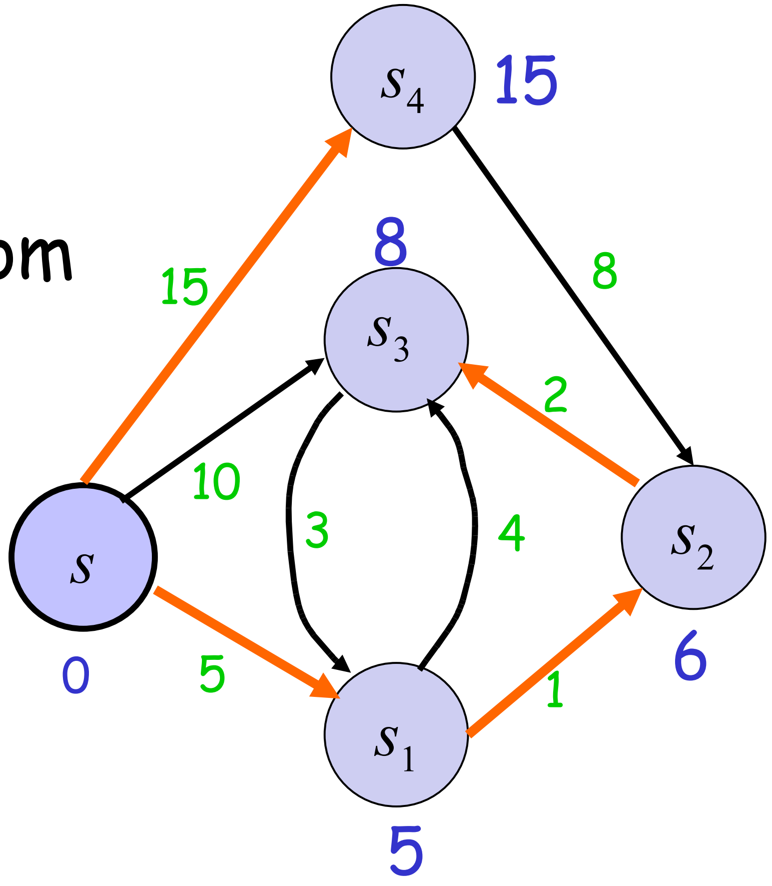
The π function gives the shortest path tree



$S = \{s, s_1, s_2, s_3, s_4\}$

When $Q = \varnothing$ then the d() values are the distances from s

The π function gives the shortest path tree



Implementation of Dijkstra's algorithm

- We need to find efficiently the vertex with minimum d() in Q
- We need to update d() values of vertices in Q

Required ADT

- Maintain items with keys subject to

- Insert(x,Q)
- min(Q)
- Deletemin(Q)
- Decrease-key(x,Q,Δ)

Required ADT

- Insert(x,Q)
- min(Q)
- Deletemin(Q)
- Decrease-key(x,Q,Δ): Can simulate by
  Delete(x,Q), insert(x-Δ,Q)

How many times we do these operations ?

- Insert(x,Q) — $n = |V|$
- min(Q) — n
- Deletemin(Q) — n
- Decrease-key(x,Q,Δ): Can simulate by
  Delete(x,Q), insert($x-\Delta$,Q) — $m = |E|$

Do we know an algorithm for this ADT ?

- Insert(x,Q)
- min(Q)
- Deletemin(Q)
- Decrease-key(x,Q,Δ): Can simulate by
  Delete(x,Q), insert(x-Δ,Q)

Yes! Use binary search trees, we had insert and delete and also min in the dictionary data type. However Heap is better W.C.

מבני נתונים 08a

תרגול 8

14/2/2008

המשך ערמות

ליאור שפירא

ערמות פיבונאצ'י

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log n	log n	1	1
delete-min	n	log n	log n	log n	log n
decrease-key	n	log n	log n	1	1
delete	n	log n	log n	log n	log n
union	1	n	log n	1	1
find-min	n	1	log n	1	1

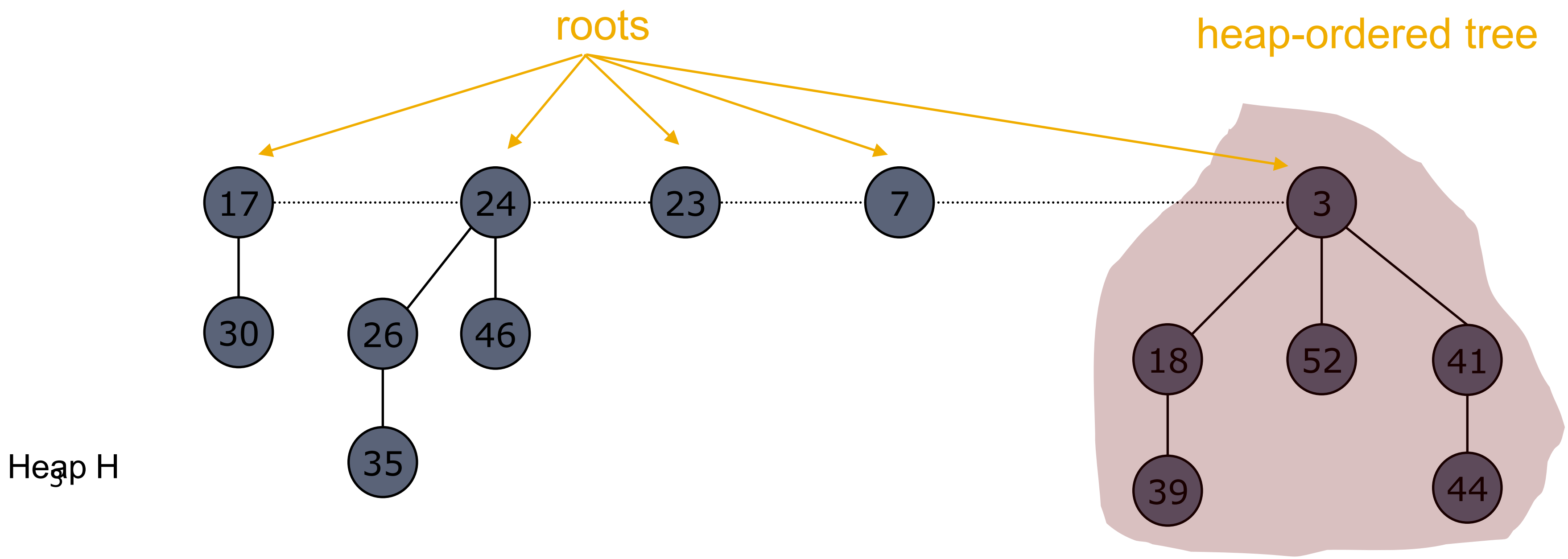
n = number of elements in priority queue

† amortized

ערימות פיבונאצ'י - מבנה

- Fibonacci heap.
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.

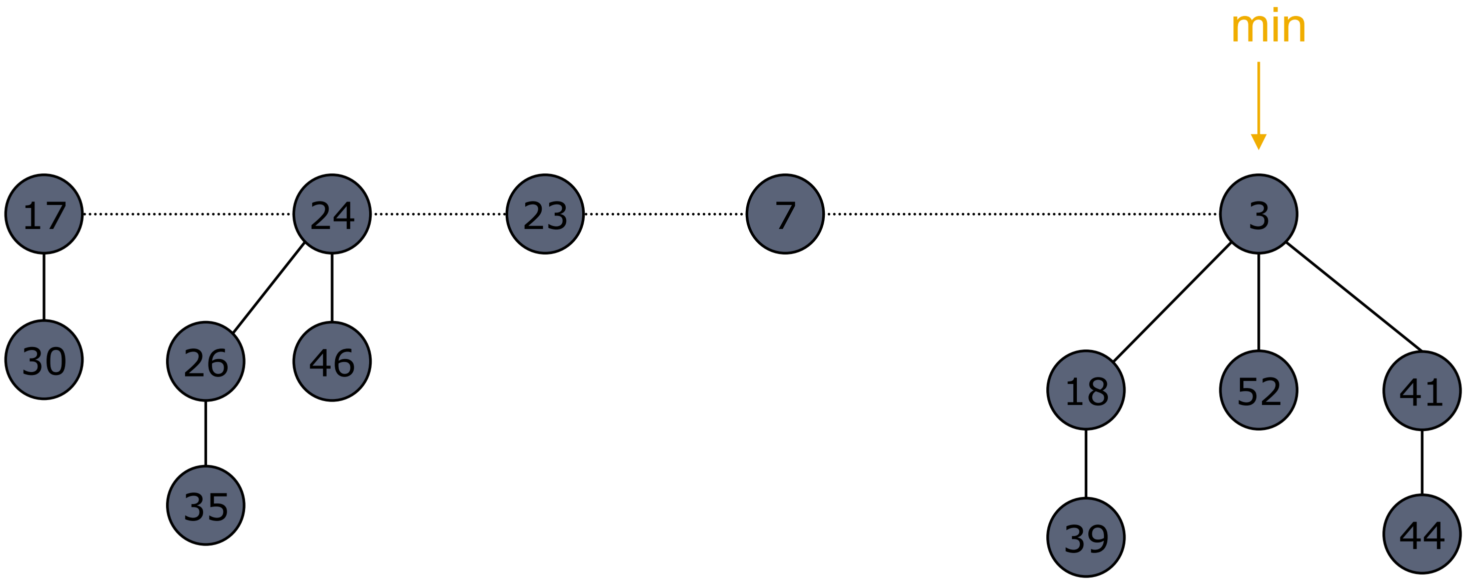
each parent larger than its children



ערימות פיבונאצ'י - מבנה

- Fibonacci heap.
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.

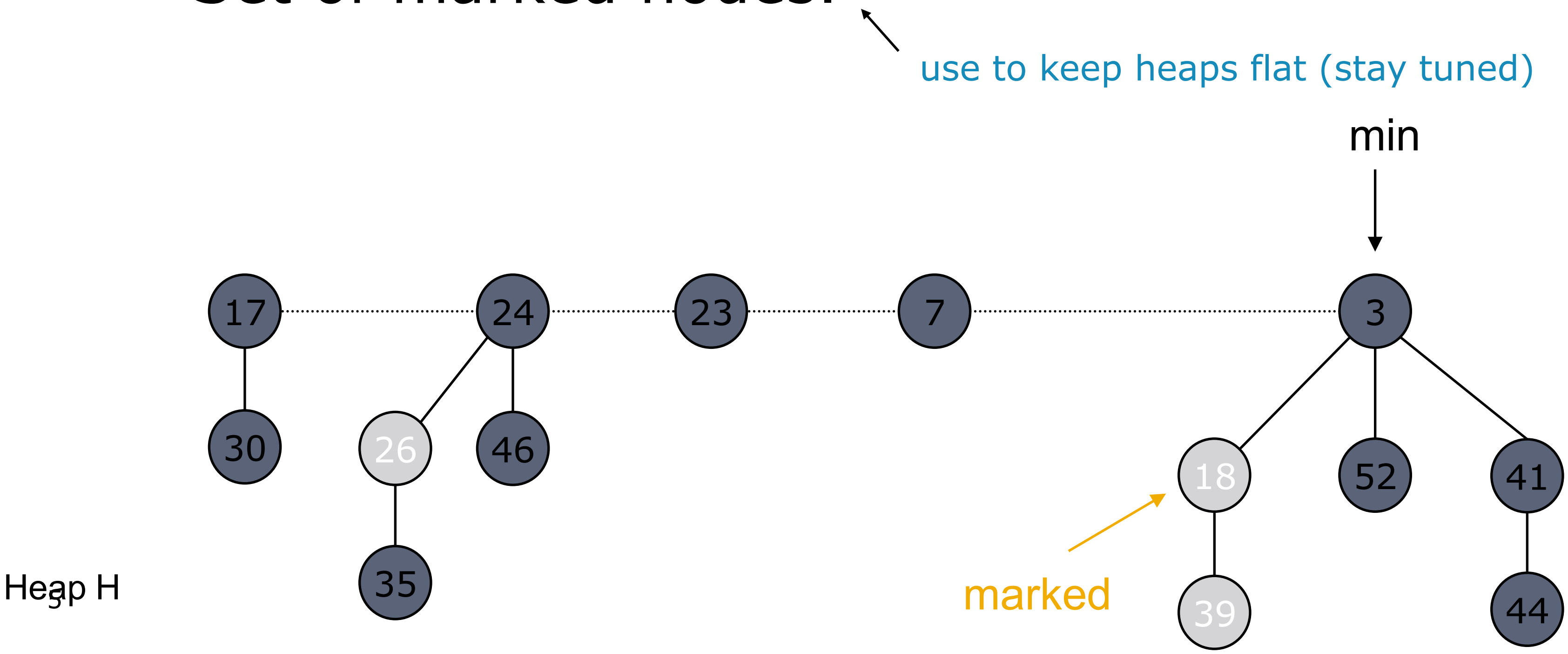
find-min takes O(1) time



Heap H

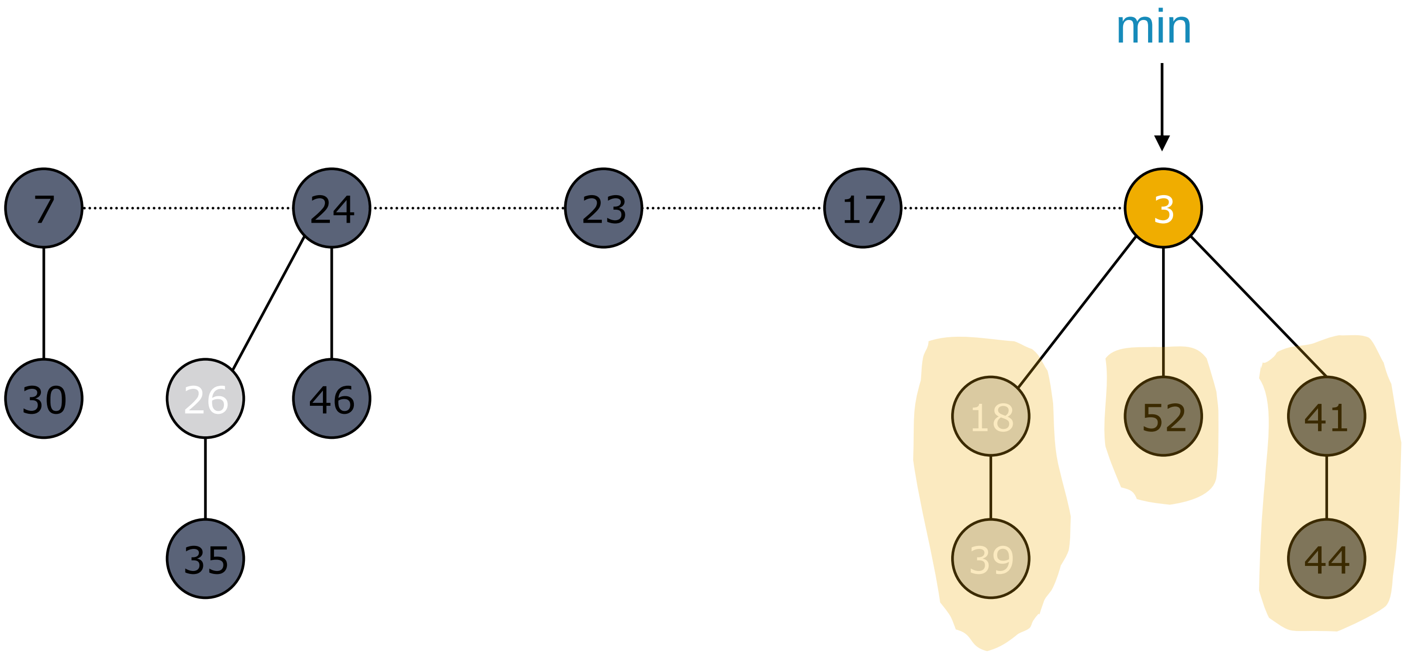
ערימות פיבונאצ'י - מבנה

- Fibonacci heap.
  - Set of heap-ordered trees.
  - Maintain pointer to minimum element.
  - Set of marked nodes.



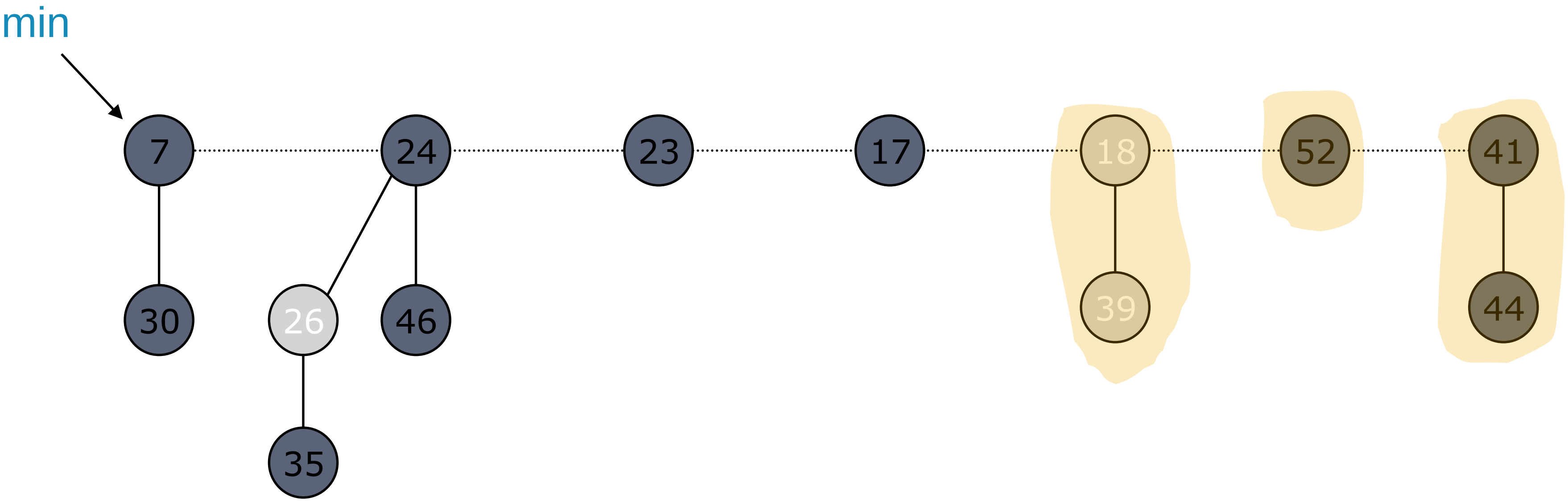
Cascading cuts & Successive linking

- פעולת delete-min
  - יודעים מי המינימום בערימה
  - נתלה את הבנים שלו כעצים בערימה
  - נבצע successive linking – מכל דרגה עץ יחיד!



Cascading cuts & Successive linking

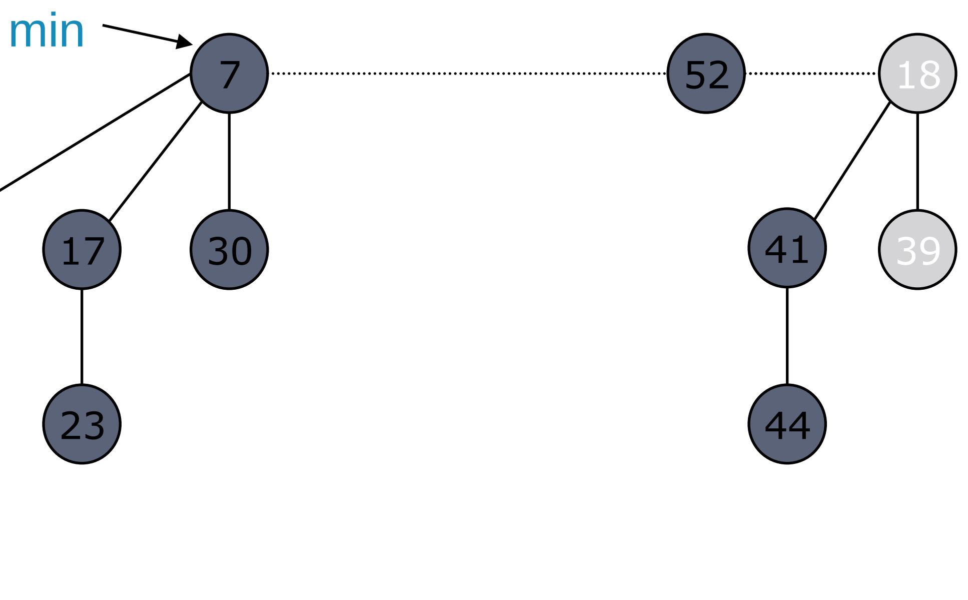
- פעולת delete-min
  - יודעים מי המינימום בערימה
  - נתלה את הבנים שלו כעצים בערימה
  - נבצע successive linking – מכל דרגה עץ יחיד!



Cascading cuts & Successive linking

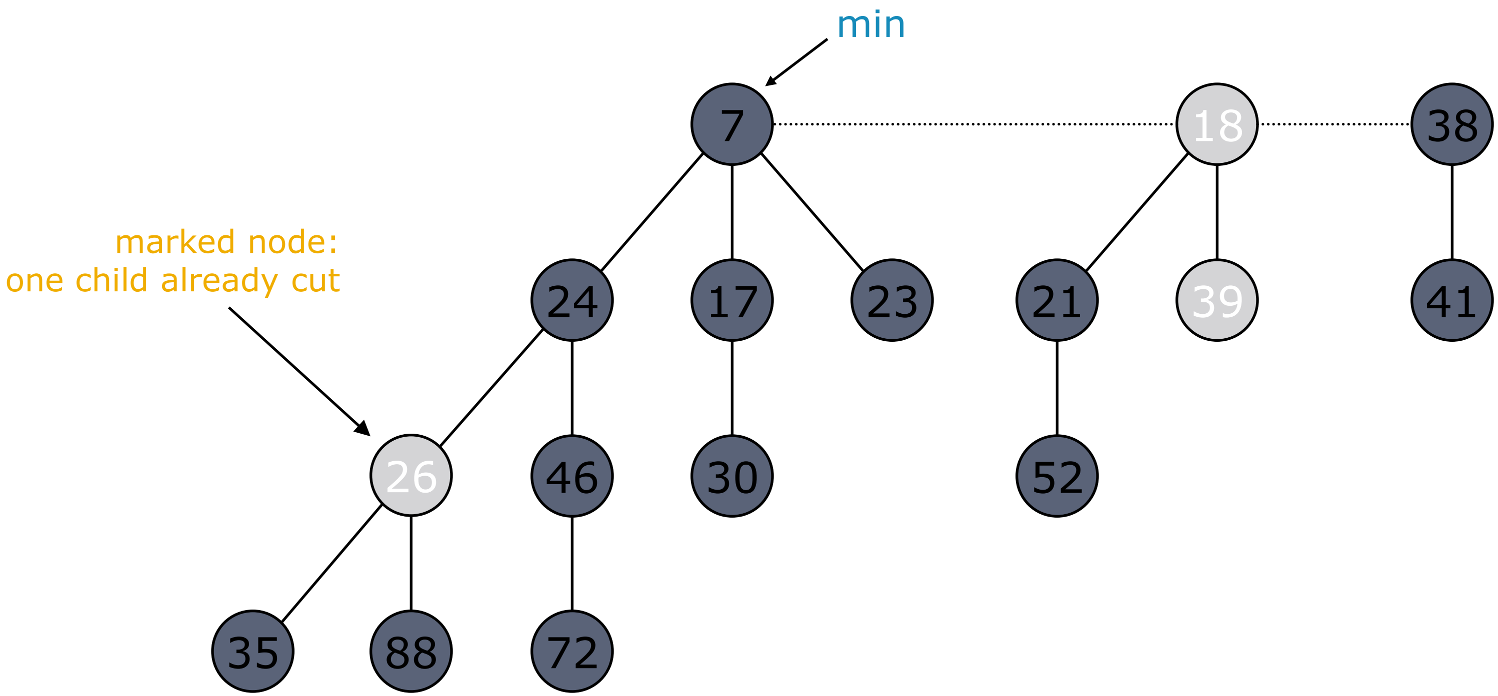
- פעולת delete-min
  - יודעים מי המינימום בערימה
  - נתלה את הבנים שלו כעצים בערימה
  - נבצע successive linking – מכל דרגה עץ יחיד!

עלות הפעולה:
Amortized O(rank(H))



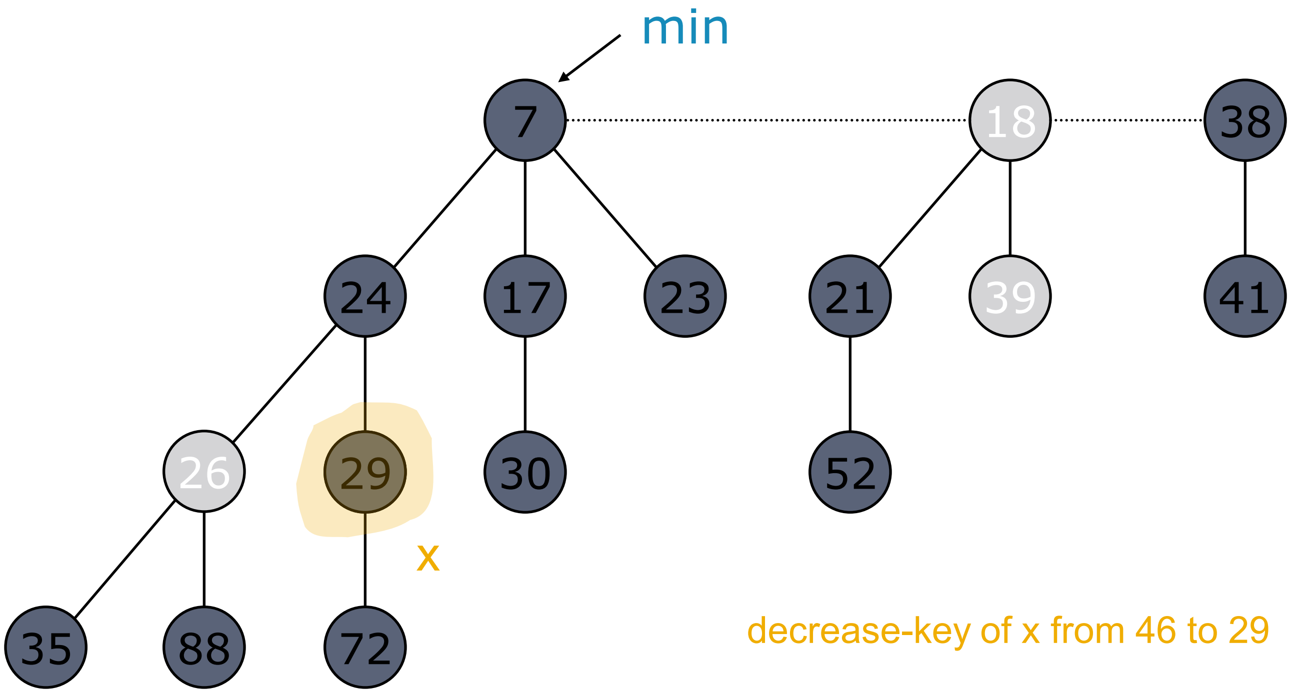
Cascading cuts & Successive linking

- פעולת decrease-key
  - אינטואיציה:
    - אם חוק הערמה לא מופר, פשוט נשנה את ערך הצומת
    - אחרת נחתוך את תת העץ של הצומת ונתלה כעץ חדש
    - בכדי לשמור על עצים שטוחים יחסית, ברגע שחותכים בן שני לצומת מסוים, גם הוא נתלה כעץ חדש



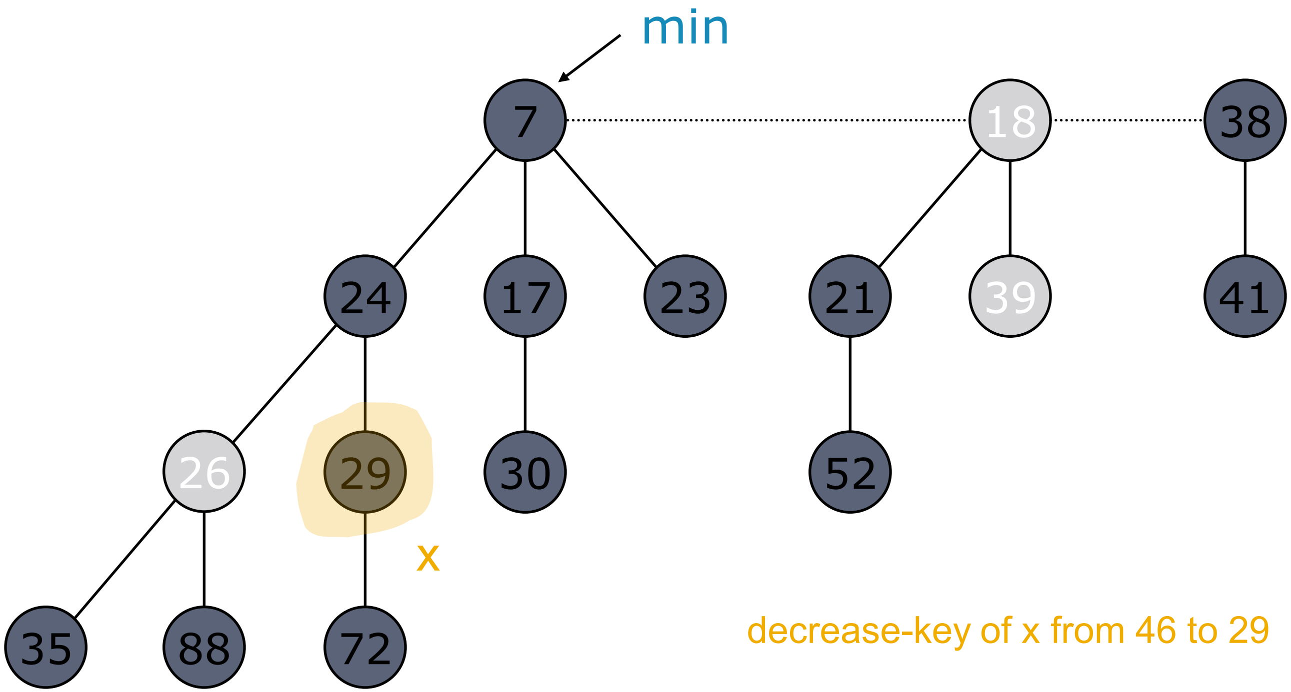
Fibonacci Heaps: Decrease Key

- Case 1. [heap order not violated]
  - Decrease key of x.
  - Change heap min pointer (if necessary).



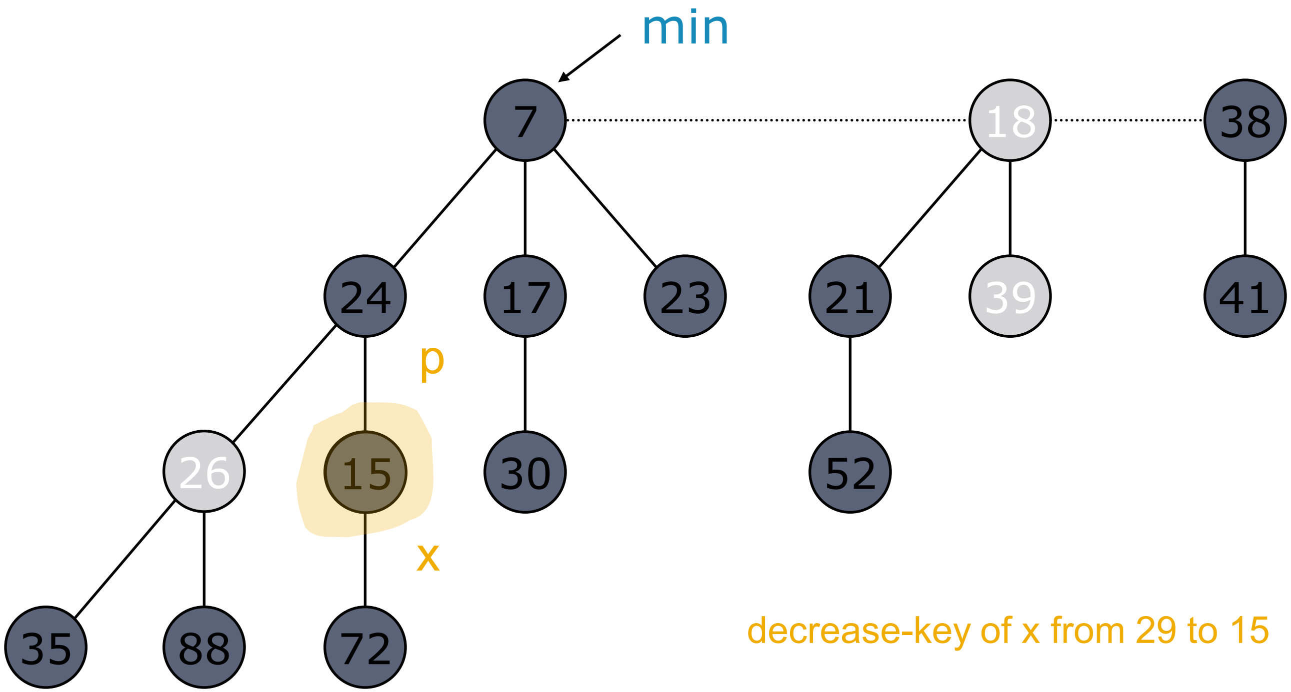
Fibonacci Heaps: Decrease Key

- Case 1. [heap order not violated]
  - Decrease key of x.
  - Change heap min pointer (if necessary).



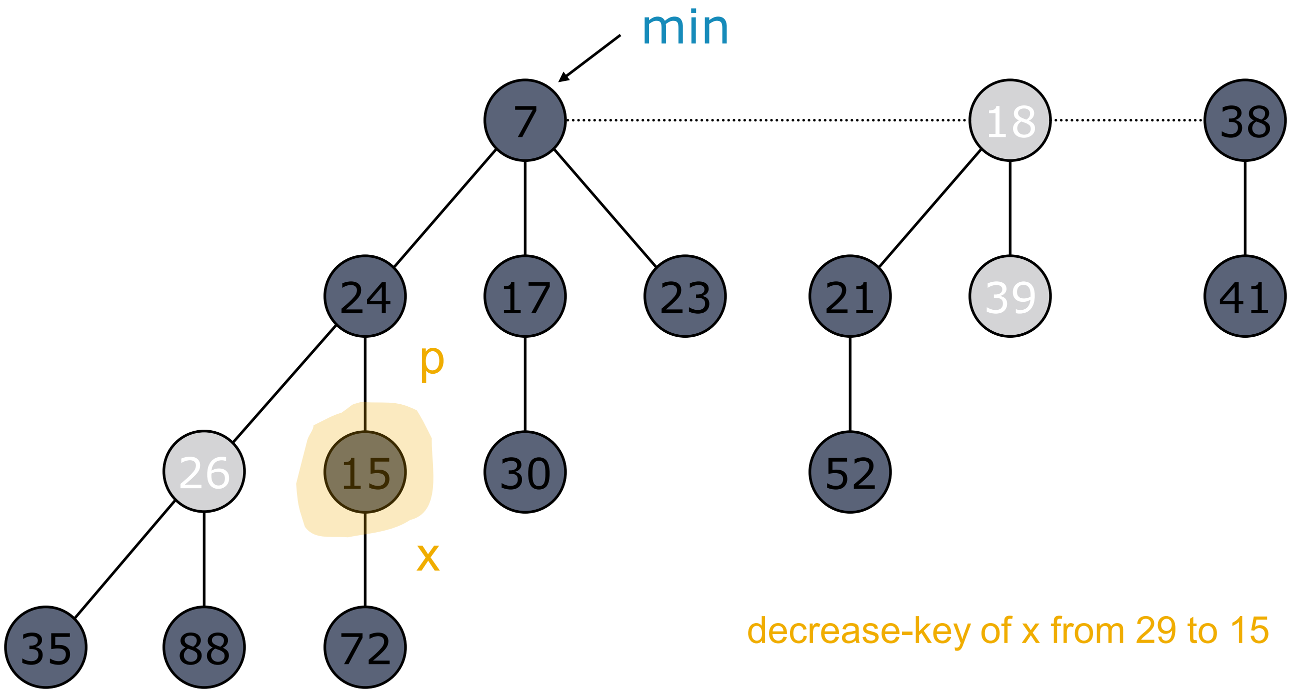
Fibonacci Heaps: Decrease Key

- Case 2a. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



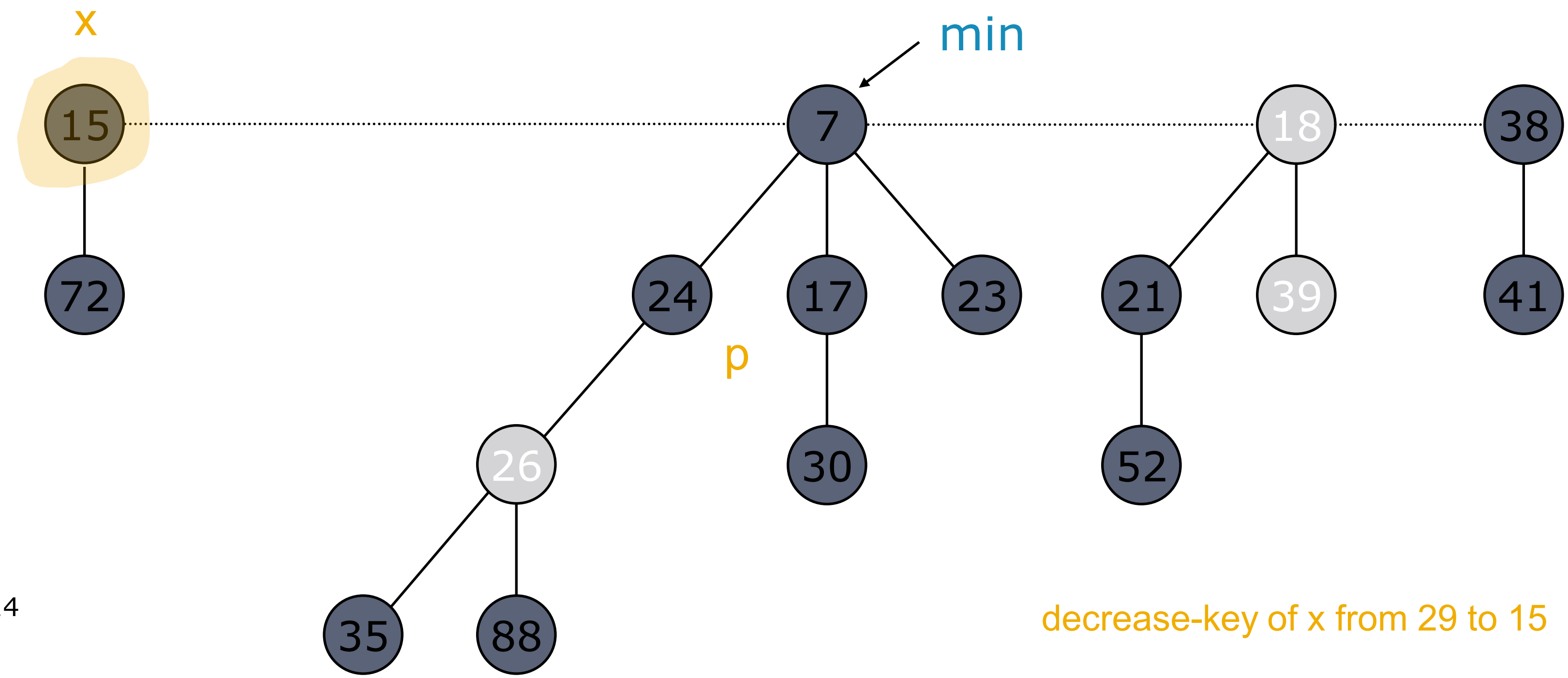
Fibonacci Heaps: Decrease Key

- Case 2a. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



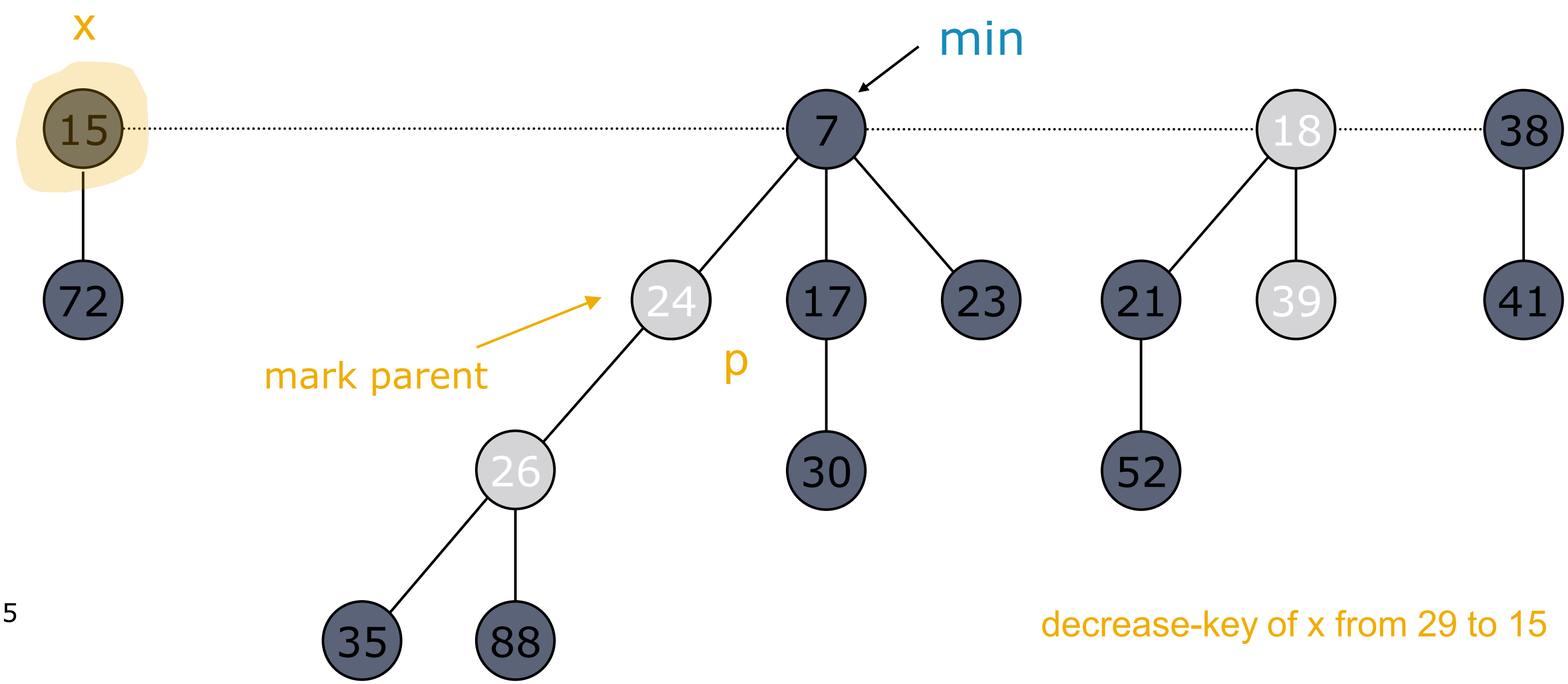
Fibonacci Heaps: Decrease Key

- Case 2a. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



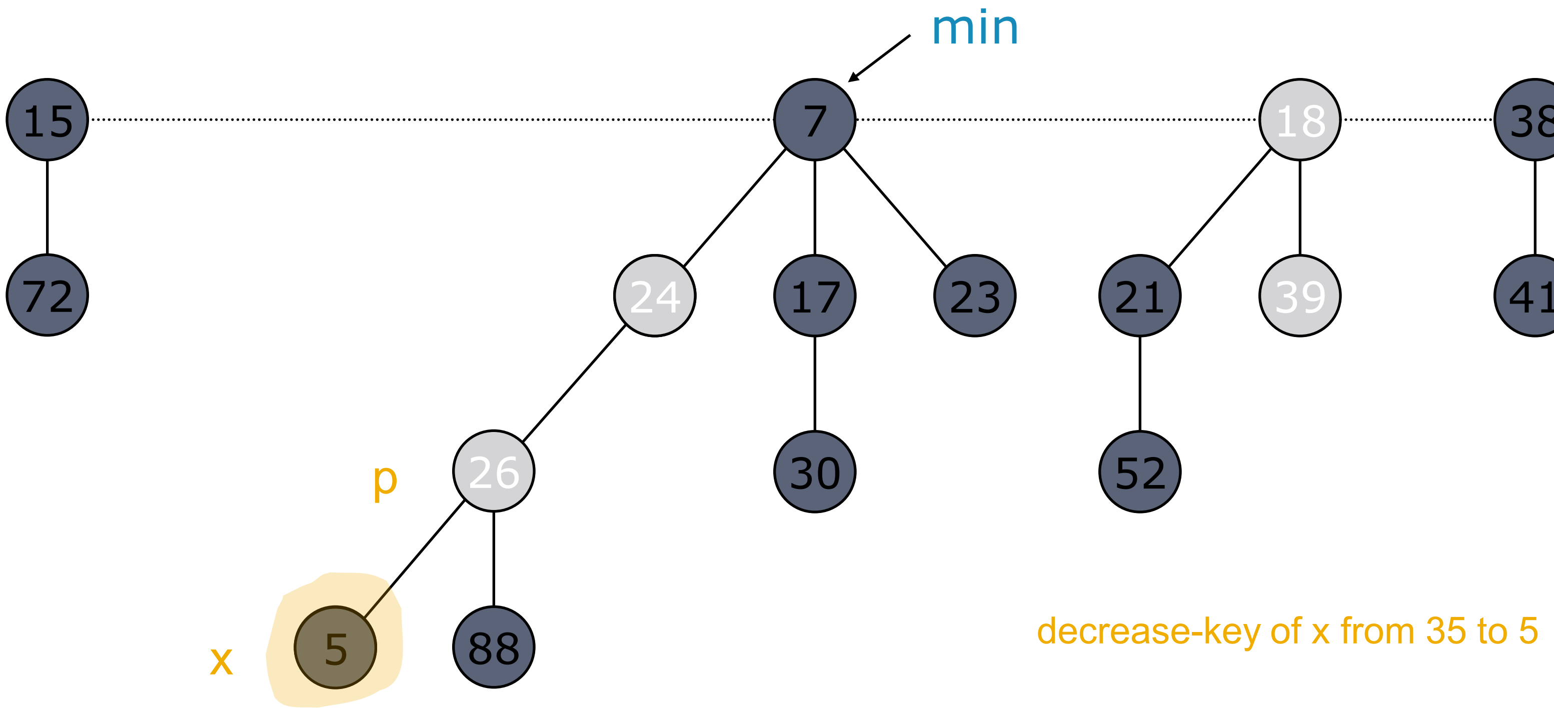
Fibonacci Heaps: Decrease Key

- Case 2a. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



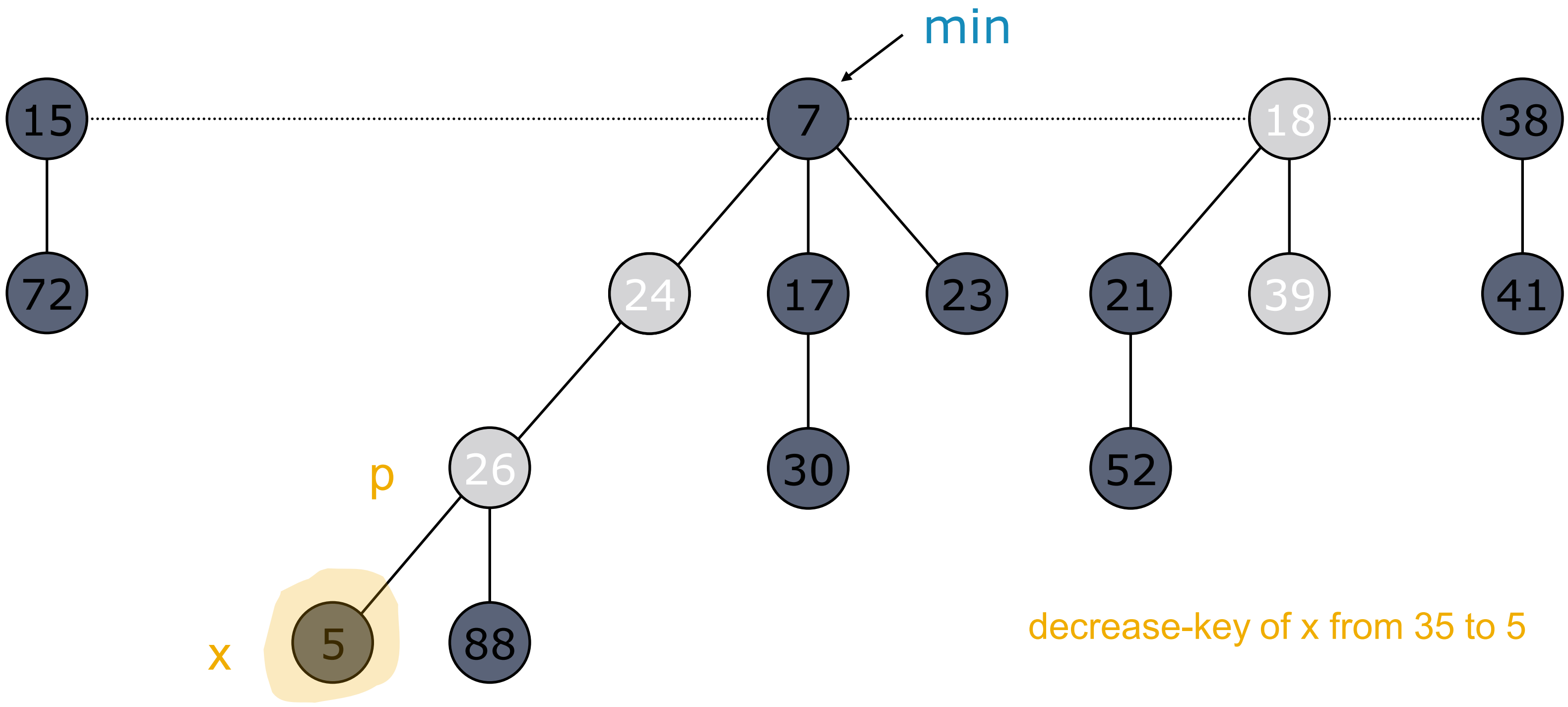
Fibonacci Heaps: Decrease Key

- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



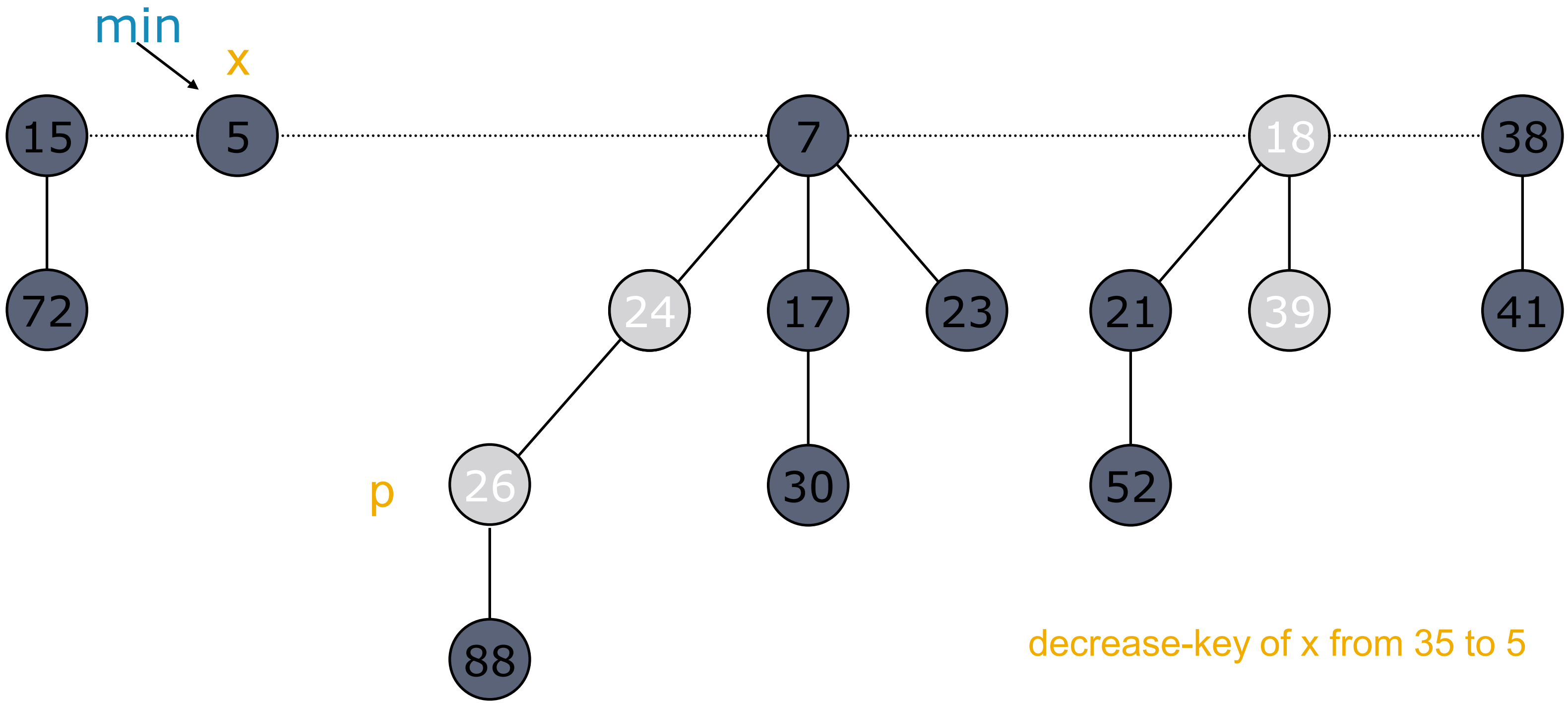
Fibonacci Heaps: Decrease Key

- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

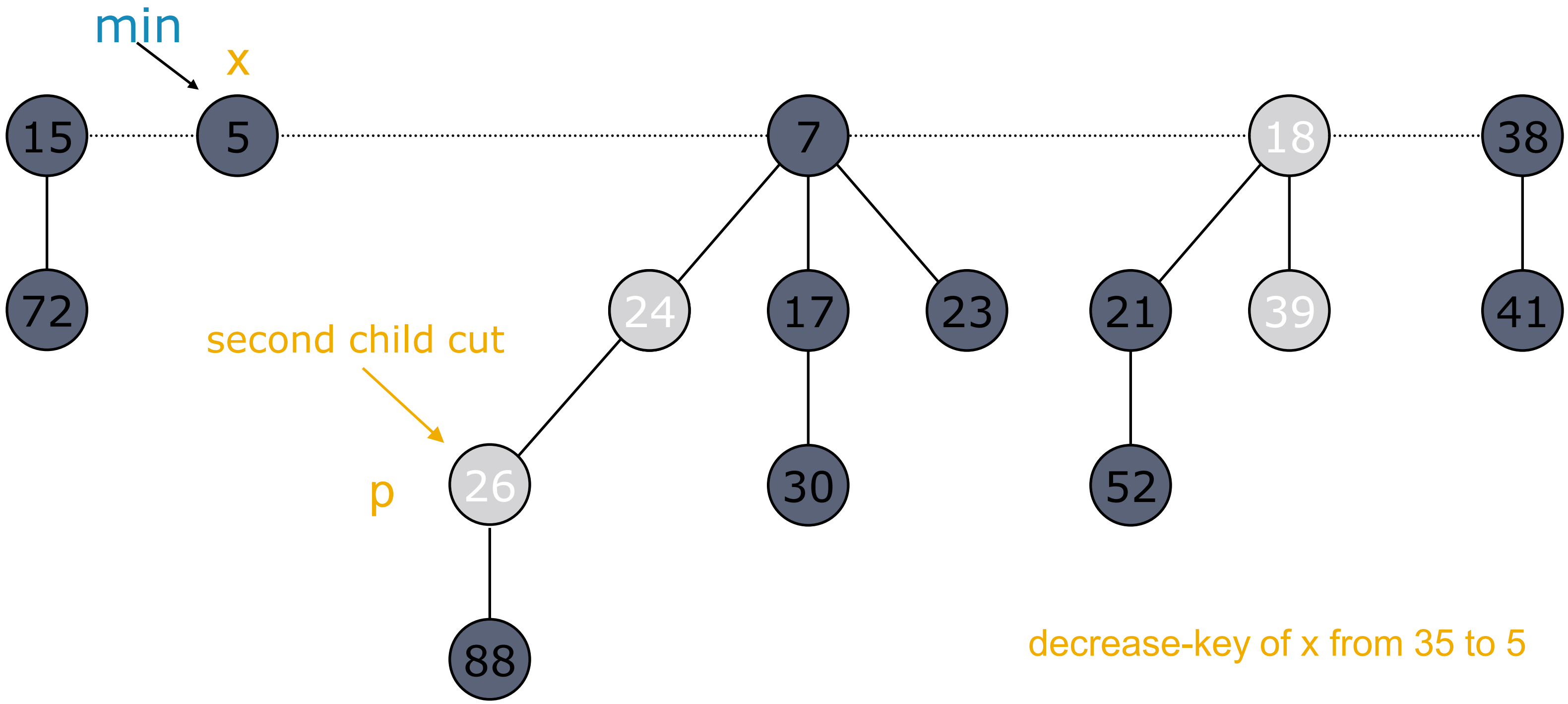
- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



decrease-key of x from 35 to 5

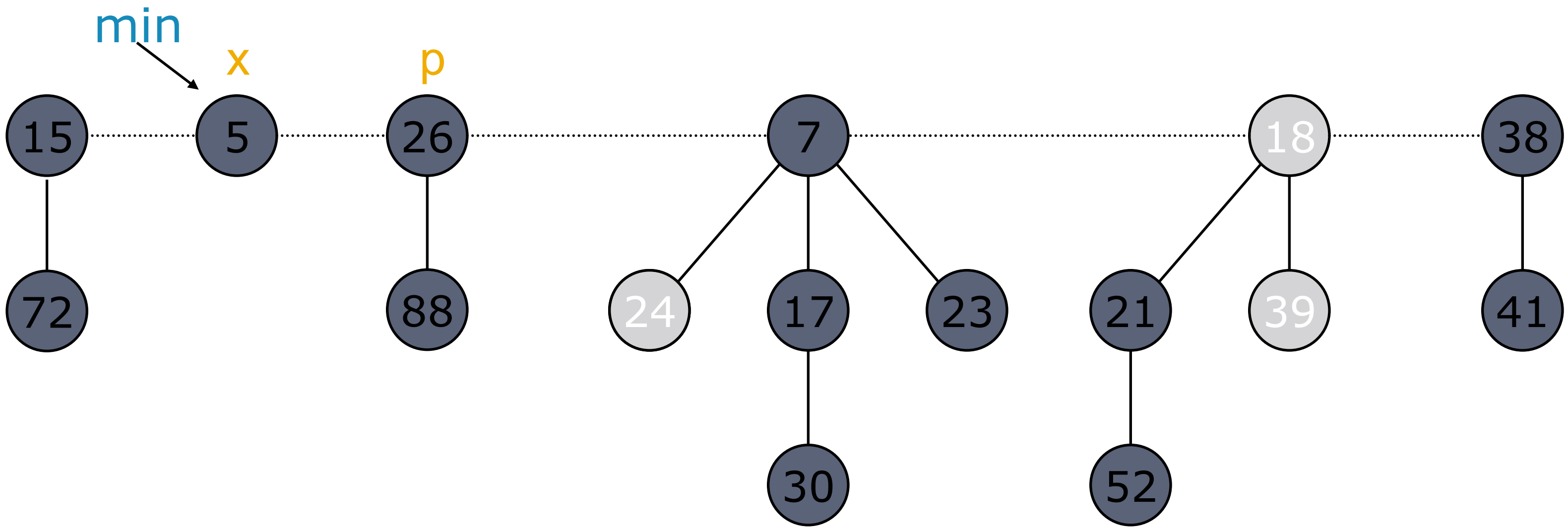
Fibonacci Heaps: Decrease Key

- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

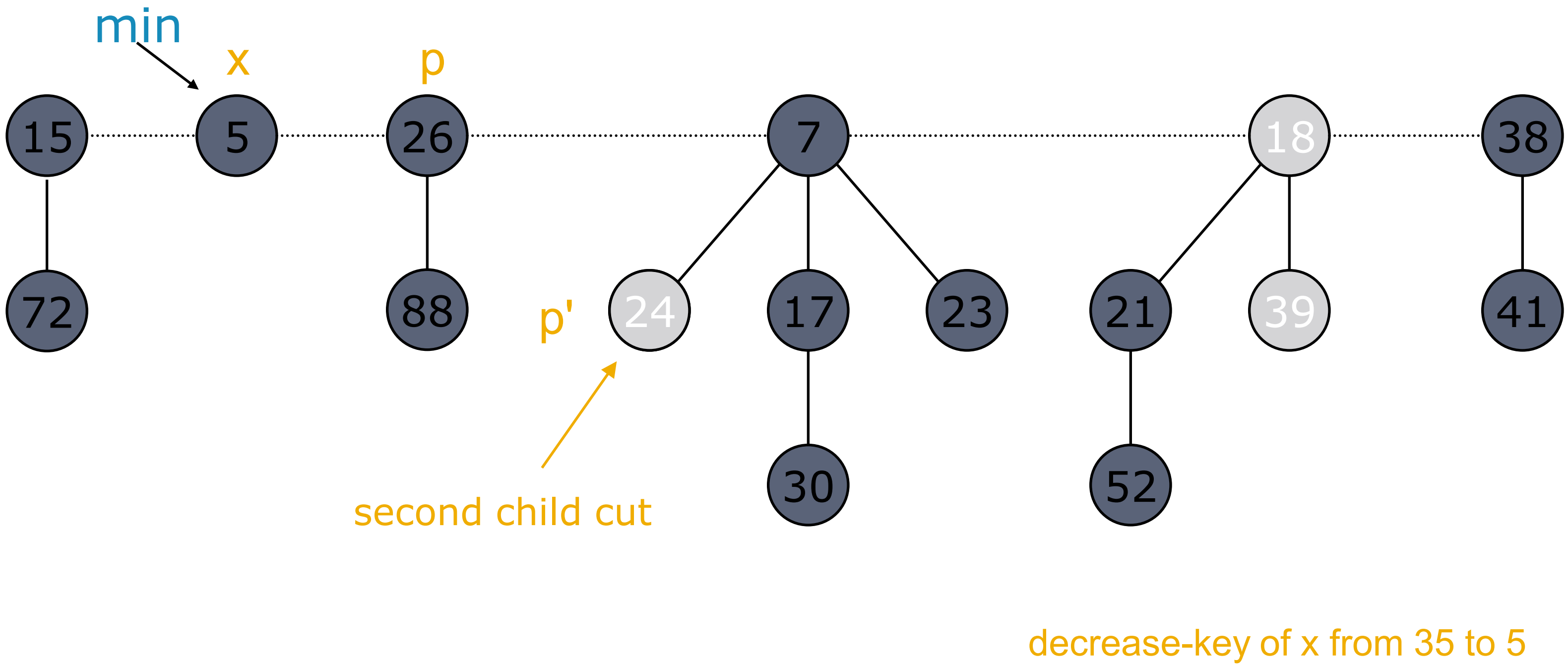
- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



decrease-key of x from 35 to 5

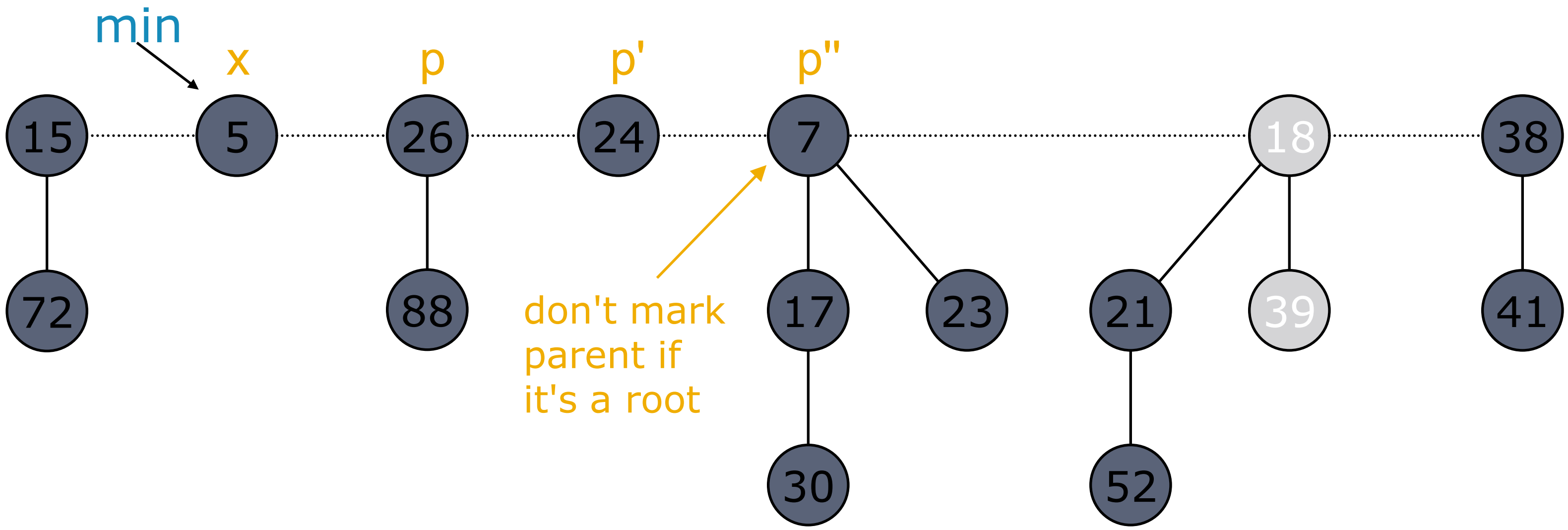
Fibonacci Heaps: Decrease Key

- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

- Case 2b. [heap order violated]
  - Decrease key of x.
  - Cut tree rooted at x, meld into root list, and unmark.
  - If parent p of x is unmarked (hasn't yet lost a child), mark it;
    Otherwise, cut p, meld into root list, and unmark
    (and do so recursively for all ancestors that lose a second child).

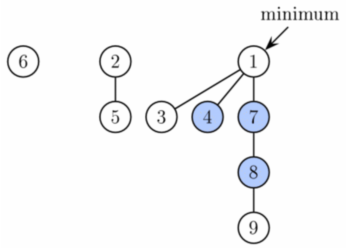


decrease-key of x from 35 to 5

ערמות פיבונצ'י

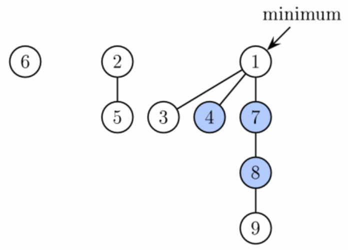
- כמו שראיתם בכיתה:

$$Rank(H) < \log_{\Phi}(n)$$



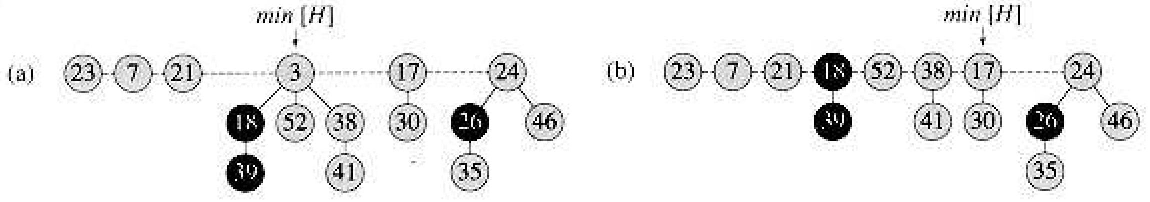
שאלה משבוע שעבר

- בהינתן מימוש של Fibonacci heaps בו לא מתבצע cascading cuts, הראו שעבור סדרת m פעולות על מקס' n אברים, עלות פעולה ממוצעת גבוהה ככל האפשר
- קצת תזכורת
  - פעולות decrease-key מאד מהירות (זמן קבוע)
  - בעת פעולת extract-min נבצע consolidate
  - דרגת כל צומת הינה D(n) והיא חסומה ע"י O(logn)

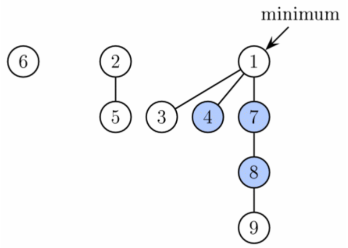


שאלה משבוע שעבר

- בהינתן שאין cascading cuts, מה משתנה ב"מה שמובטח לנו"? ז.א איפה השתמשנו בידע שלא ניתן להוריד 2 בנים מצומת בלי שהוא ייחתך?
  - בהוכחה שהדרגה המקסימלית O(logn) (חסם על D(n))
- איפה משתמשים בעובדה זו?
  - ב-extract min נעבור על כל השורשים (מקס' D(n))

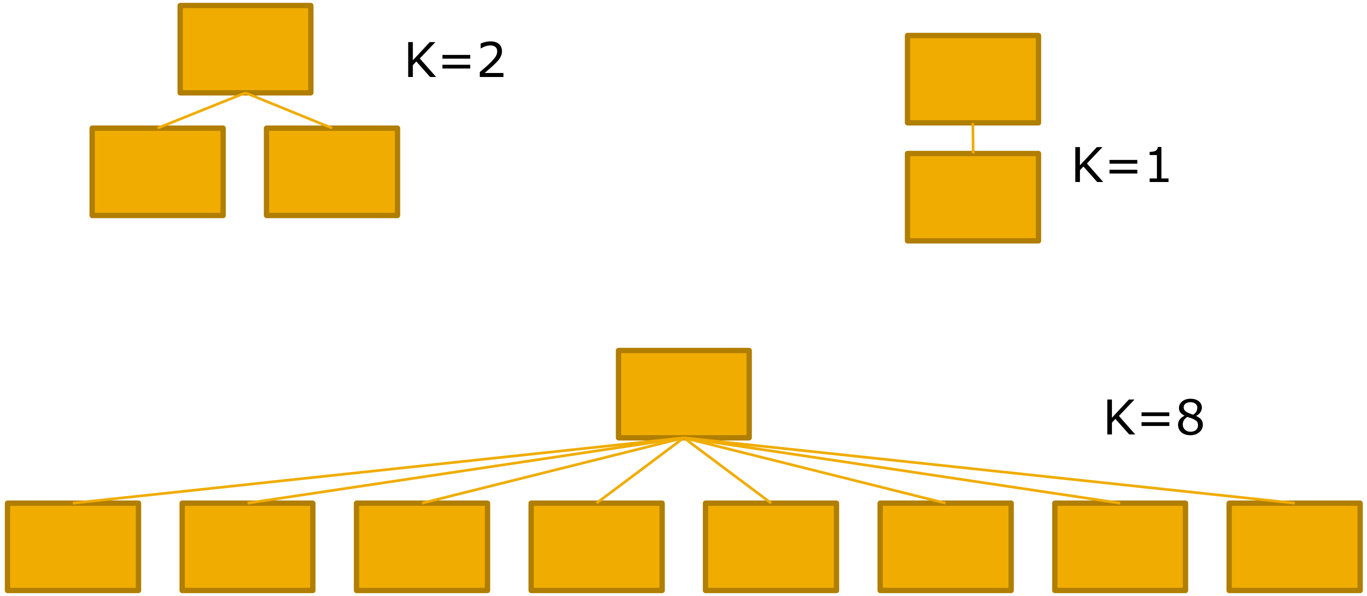


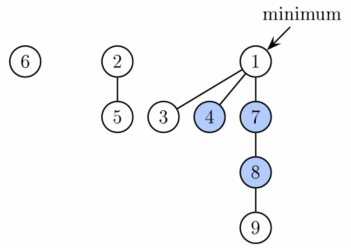
  - ז"א אם נגיע למצב שבו יש הרבה מאד עצים בערמה, כל פעולה תהיה יקרה
  - מעבר לכך, אם כל עץ מדרגה שונה, פעולת ה-consolidate לא תחבר עצים ביחד



שאלה משבוע שעבר

- כיצד נגדיר עץ מדרגה k שהוא "ממש גרוע"?
  - נגדיר עץ מיוחד מדרגה k ← k-rank star. זהו שורש עם k בנים, כולם עלים.

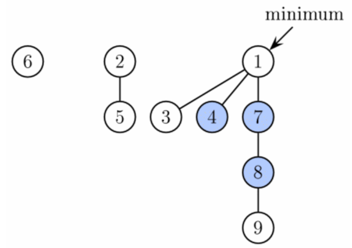




שאלה משבוע שעבר

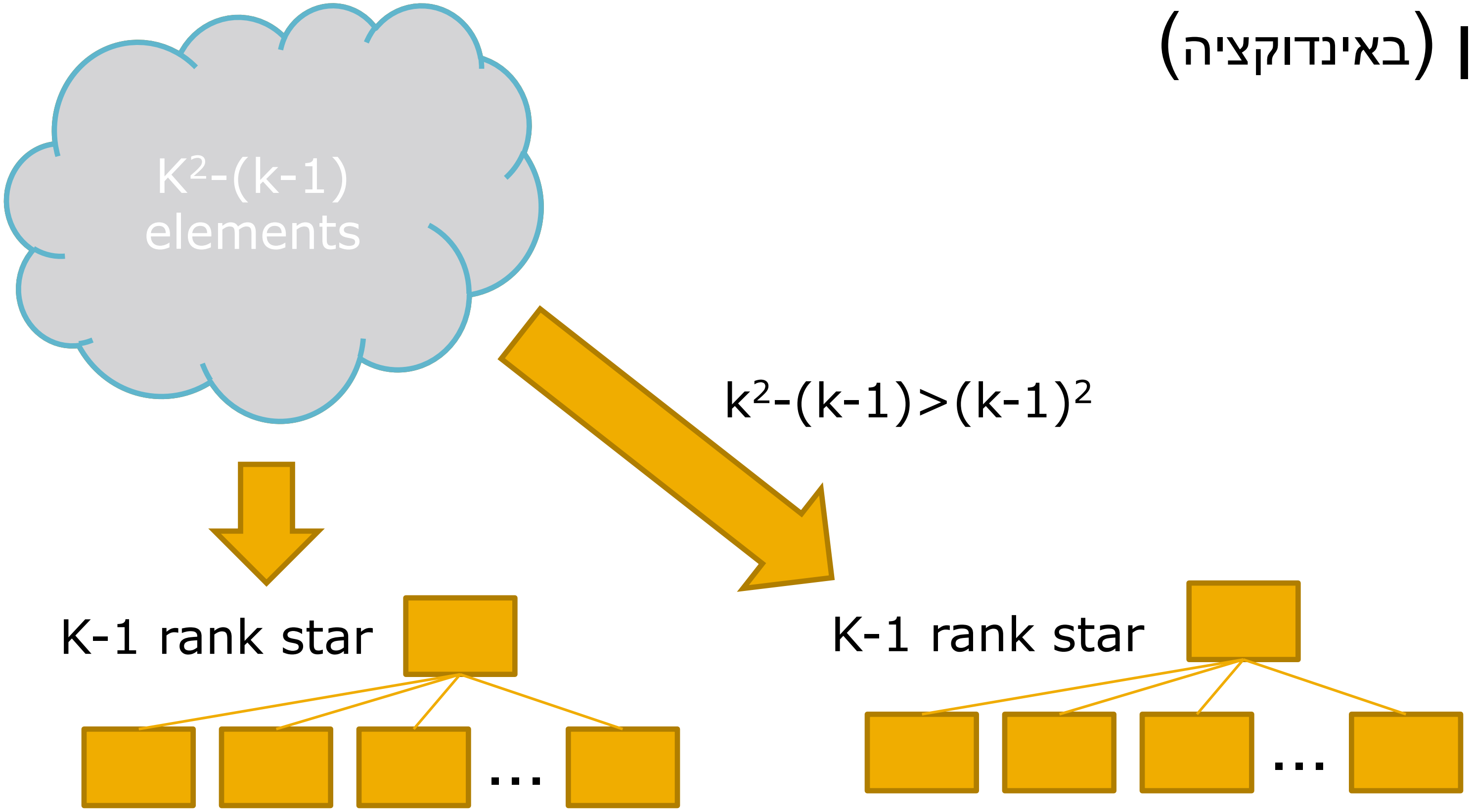
- הפתרון יורכב משני שלבים

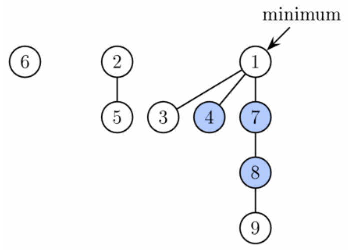
1. בסדרה של f(n) שלבים, בנו ערמת פיבונצ'י כך שיש
   - 1 rank-0 star
   - 1 rank-1 star
   - ...
   - 1 rank-sqrt(n) star
2. חזרו על הפעולות הבאות מספר רב של פעמים (O(m))
   - הכנס ערך X ממש קטן
   - בצע extract-min (מוחק את X)
   - **כל אחת מהפעולות לוקחת $\Omega(\sqrt{n})$ זמן, כך שעבור m>>n פעולה תיקח בממוצע $\sqrt{n}$**



שאלה משבוע שעבר

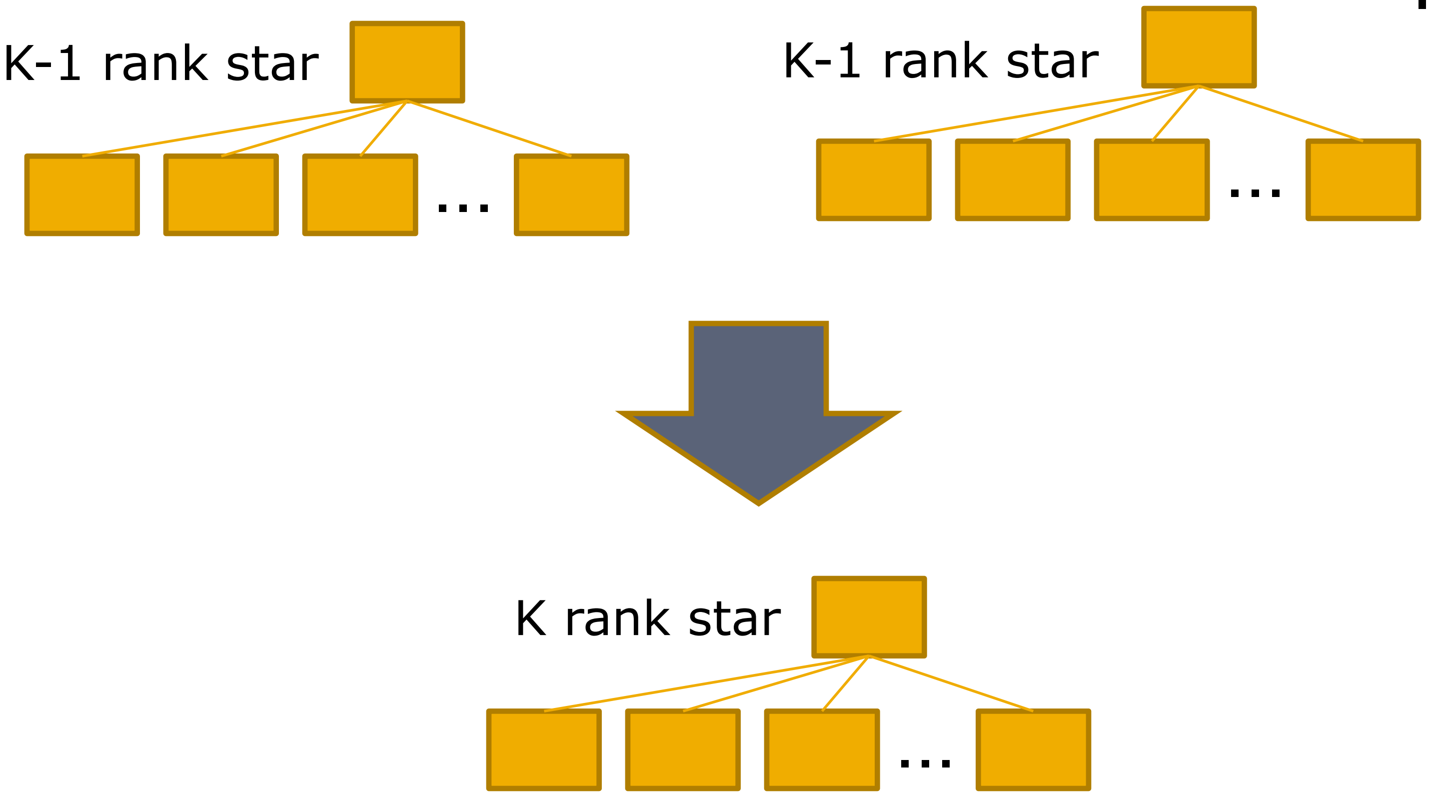
- אך כיצד בונים k-rank star? כיצד נוודא שבשום שלב לא יהיו יותר מ-n אברים בערמה?
- פתרון (באינדוקציה)





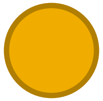
שאלה משבוע שעבר

- אך כיצד בונים k-rank star? כיצד נוודא שבשום שלב לא יהיו יותר מ-n אברים בערמה?
- פתרון

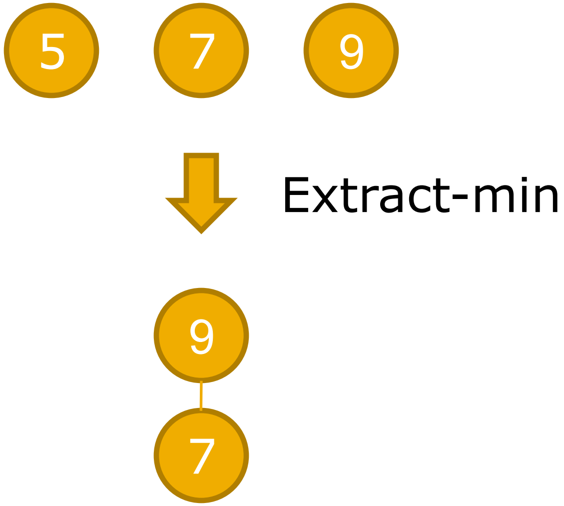


תרגיל 2 – ערמות פיבונאצ'י

- האם ניתן לבנות ערמת פיבונאצ'י ובה עץ אחד מעומק n?
- פתרון
  - עבור n=1

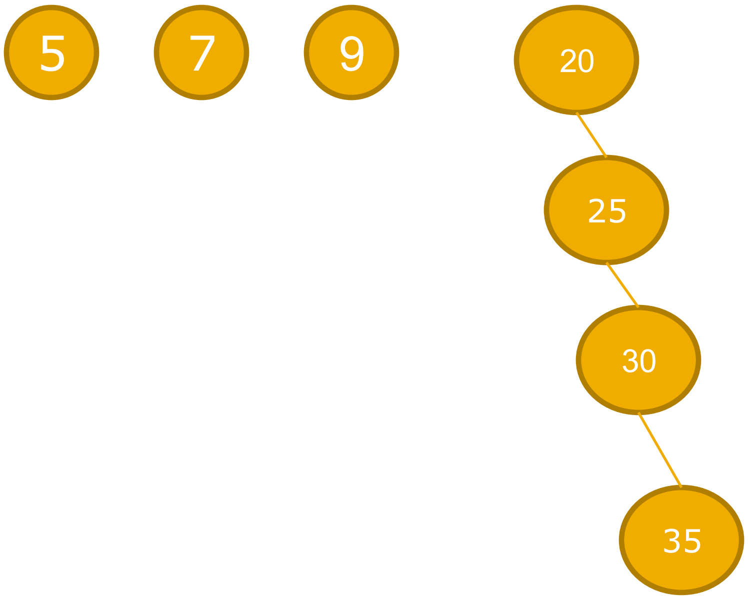


  - עבור n=2



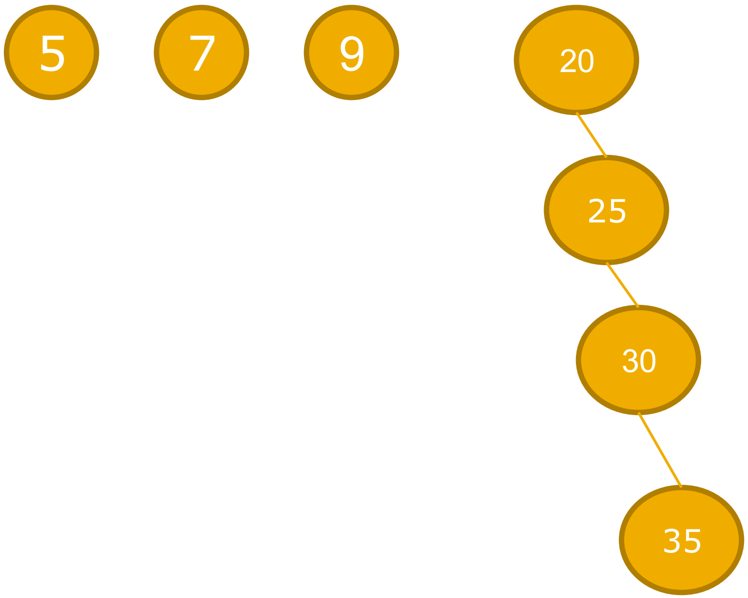
תרגיל 2 – ערמות פיבונאצ'י

- האם ניתן לבנות ערמת פיבונאצ'י ובה עץ אחד מעומק n?
- פתרון
  - נניח שאנו יודעים לפתור עבור n=k
  - נפתור עבור n=k+1
    - נגדיר x′,y′,z′ כך ש- $key[z'] < key[y'] < key[x'] < min[H]$
    - נכניס אותם לתוך הערמה



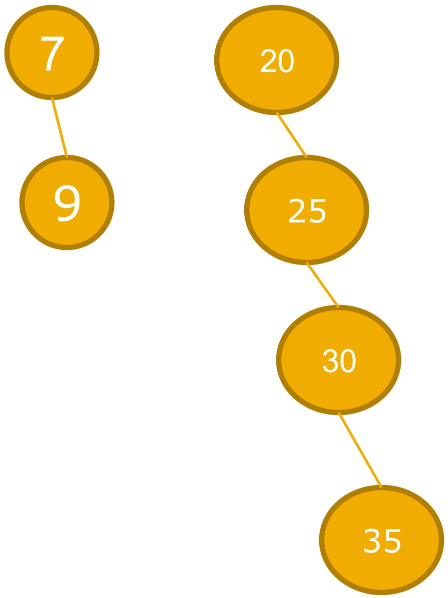
תרגיל 2 – ערמות פיבונאצ'י

- האם ניתן לבנות ערמת פיבונאצ'י ובה עץ אחד מעומק n?
- פתרון
  - נניח שאנו יודעים לפתור עבור n=k
  - נפתור עבור n=k+1
    - נגדיר x', y', z' כך ש- $key[z'] < key[y'] < key[x'] < \min[H]$
    - נכניס אותם לתוך הערמה
    - נפעיל extract-min



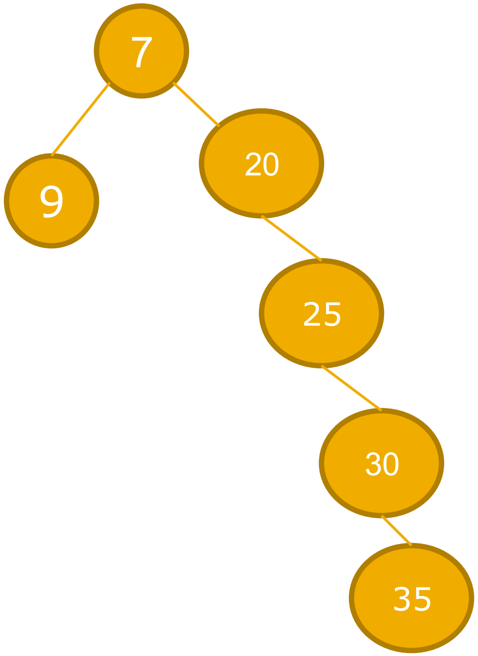
תרגיל 2 – ערמות פיבונאצ'י

- האם ניתן לבנות ערמת פיבונאצ'י ובה עץ אחד מעומק n?
- פתרון
  - נניח שאנו יודעים לפתור עבור n=k
  - נפתור עבור n=k+1
    - נגדיר x′,y′,z′ כך ש- $key[z'] < key[y'] < key[x'] < min[H]$
    - נכניס אותם לתוך הערמה
    - נפעיל extract-min



תרגיל 2 – ערמות פיבונאצ'י

- האם ניתן לבנות ערמת פיבונאצ'י ובה עץ אחד מעומק n?
- פתרון
  - נניח שאנו יודעים לפתור עבור n=k
  - נפתור עבור n=k+1
    - נגדיר x′,y′,z′ כך ש- $key[z'] < key[y'] < key[x'] < min[H]$
    - נכניס אותם לתוך הערמה
    - נפעיל extract-min
    - נמחק את x′ (9)



תרגיל 3

- בערמות פיבונצ'י, אנחנו מבצעים cascading cuts בצומת v אם הוא איבד צומת בן מאז הפעם האחרונה שהוא נתלה על צומת אחרת. נניח שנבצע CC רק אם v איבד שני בנים מאז, כיצד משתנה הלמה:
  - x צומת בערמת פיב', $y_1,...,y_n$ בנים של x, מסודרים לפי הסדר בו נתלו על x (הכי ישן ועד הכי חדש). אזי $rank(y_i) \geq i-2$ לכל i.

תרגיל 3

- Answer
  - rank(y_i)≥i-3
  - Since y_i had the same rank as x when it became a child of x
  - x must have had at least i-1 children at that time, so y_i had at least i-1 rank.
  - It could have lost at most two children since then, therefore rank at least i-3

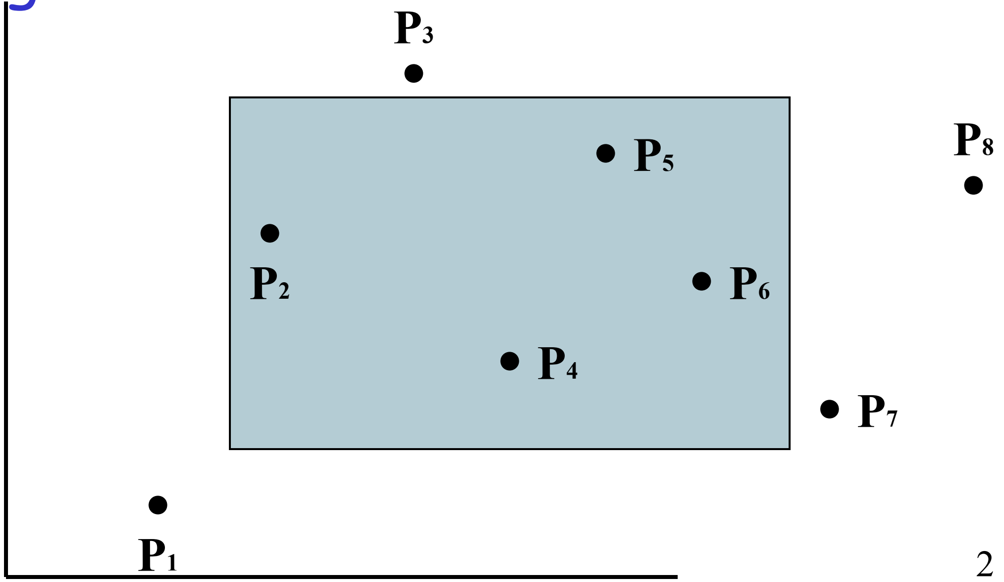
הסוף

Balanced Trees

Motivation Example - Reporting (2-D)

Given a set of points S on the plane, preprocess them to build structure that allows efficient queries of the from:

Given an rectangle $R=[x_1,x_2][y_1,y_2]$ find all points in S that are in the rectangle.



Summary

	Arrays	Linked List	Bal. Trees
	Simple, fast Inflexible	Simple Flexible	Not-So-simple Flexible
Add	O(1) O(n) *inc sort*	O(1) *O(n) - inc sort*	O(logn)
Delete	O(n)	O(1) - *any* O(n) - *specific*	O(log n)
Find	O(n) O(logn) *binary search*	O(n) *(no binary search)*	O(log n)

Order statistics Trees

rank and select

The dictionary ADT

- Insert(x,D)
- Delete(x,D)
- Find(x,D):

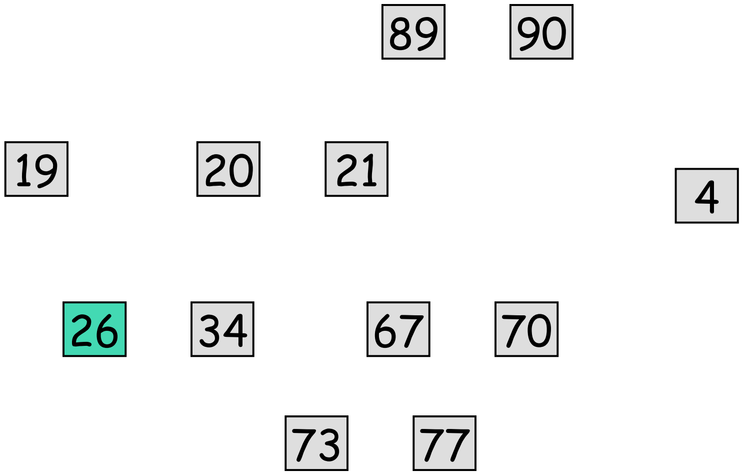
  Returns a pointer to x if $x \in D$, and a pointer to the successor or predecessor of x if x is not in D

Suppose we want to add to the dictionary ADT

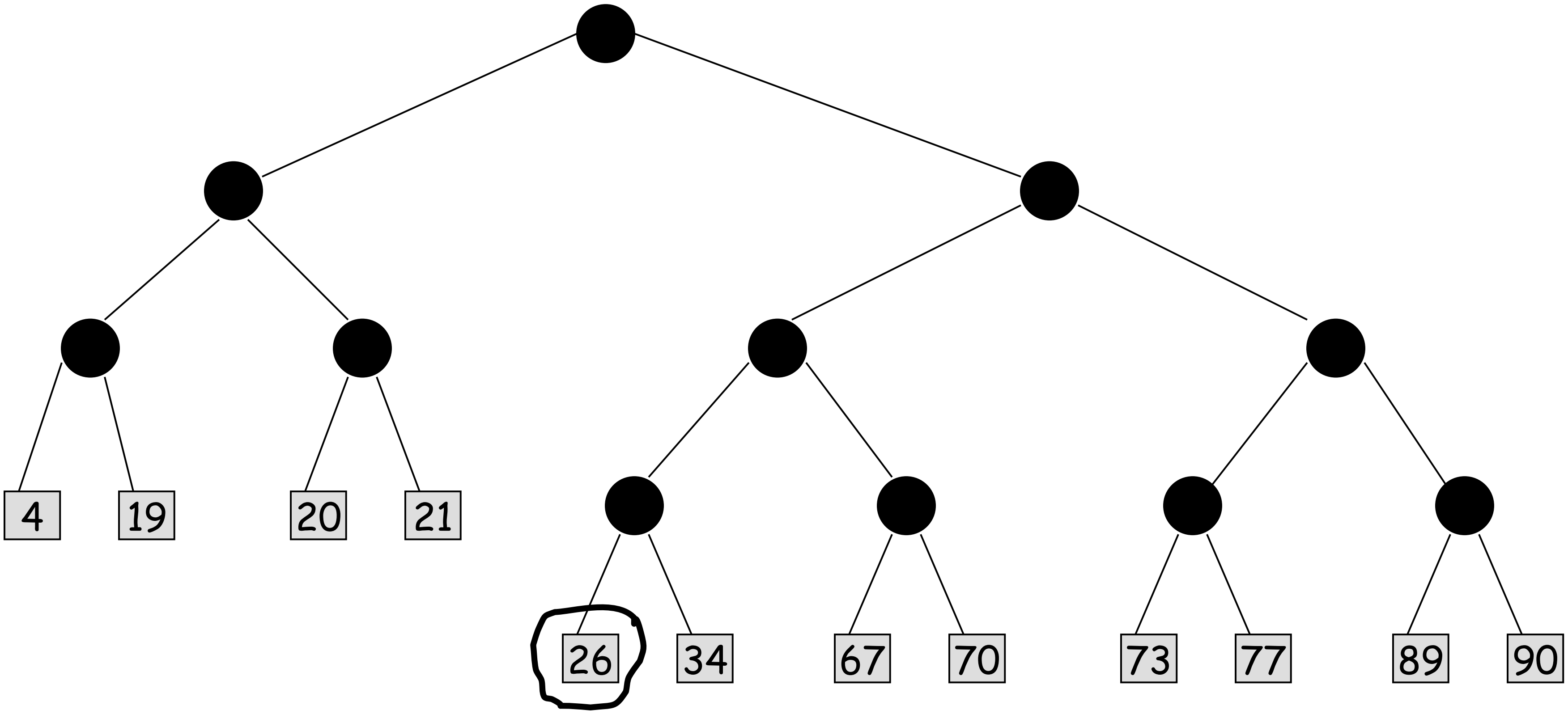
- Select(i,D): Returns the ith element in the dictionary:

An element x such that i-1 elements are smaller than x

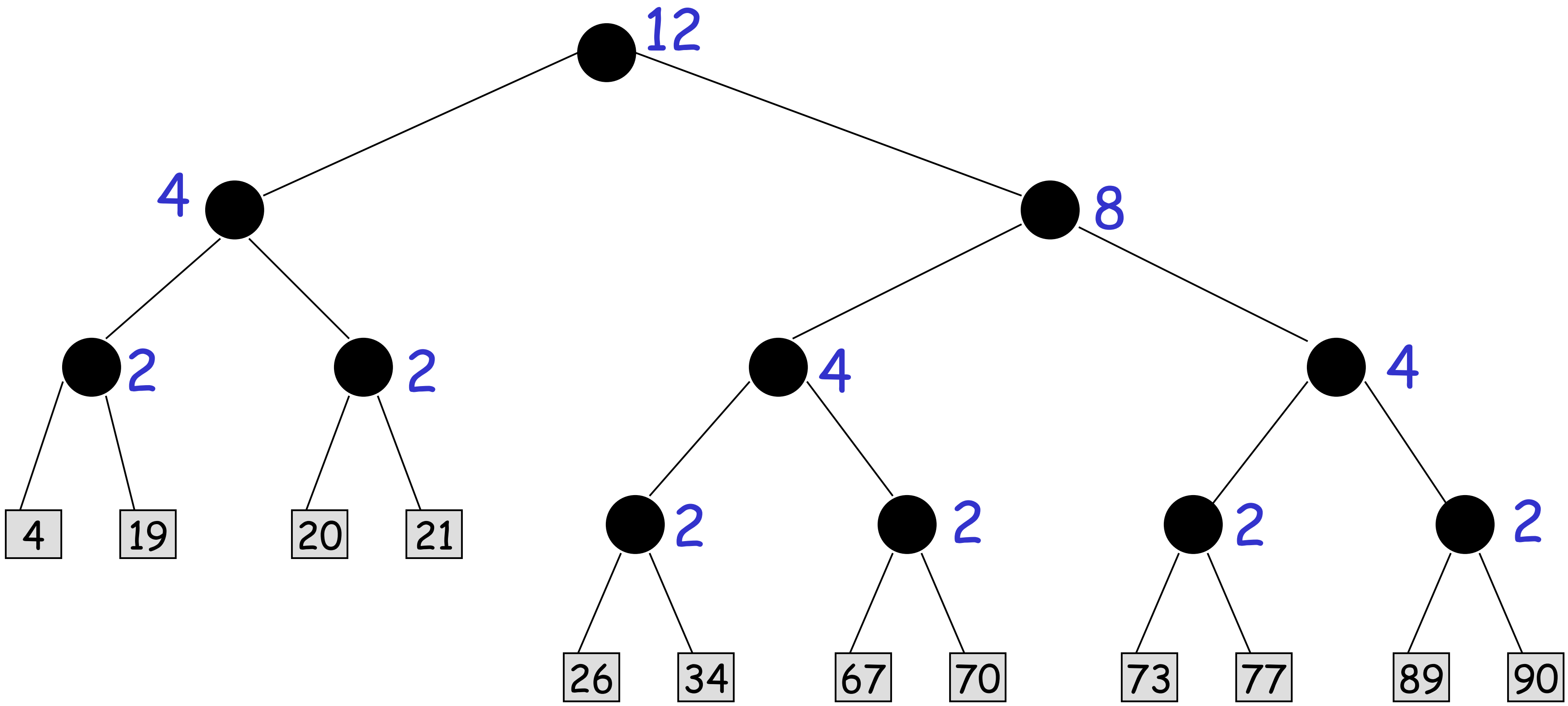
Select(5,D)



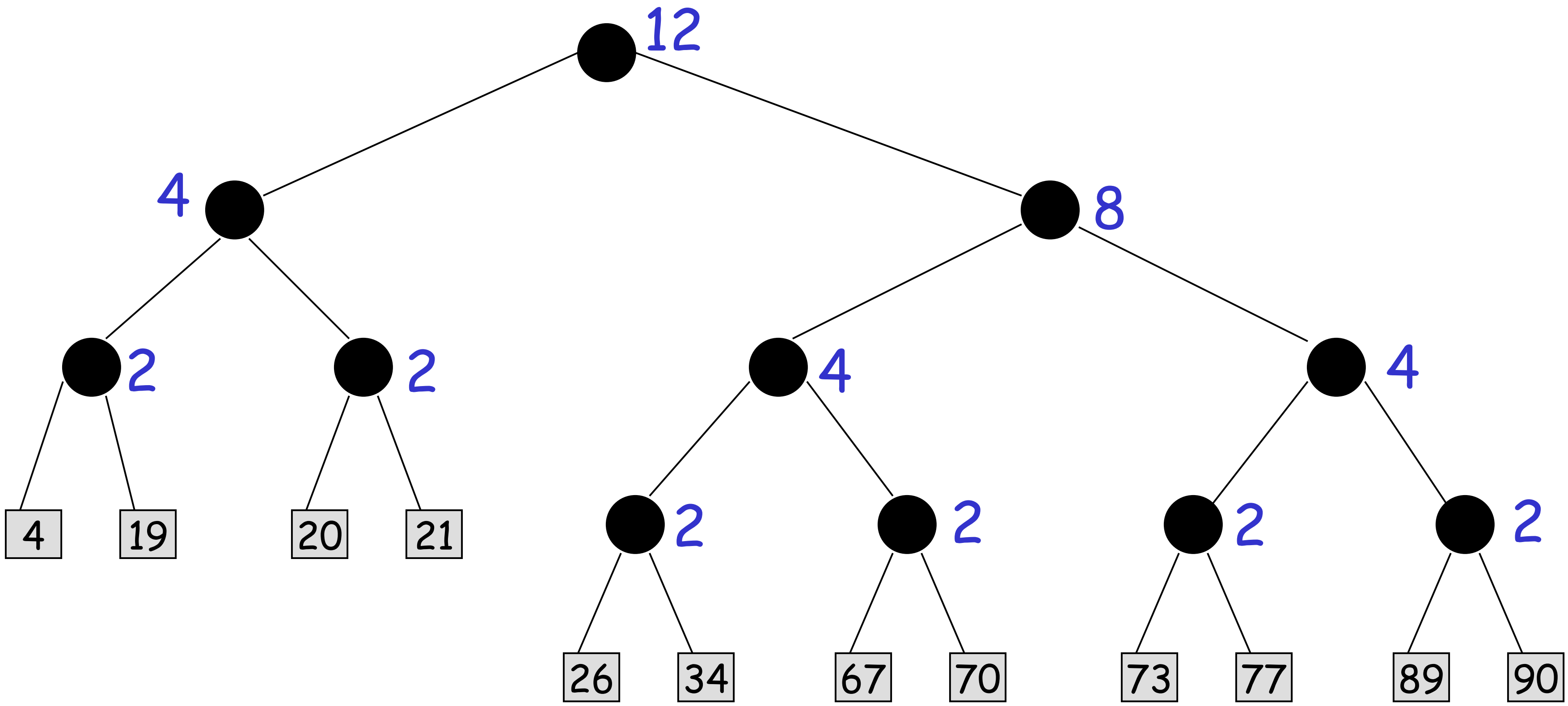
Can we still use trees ?



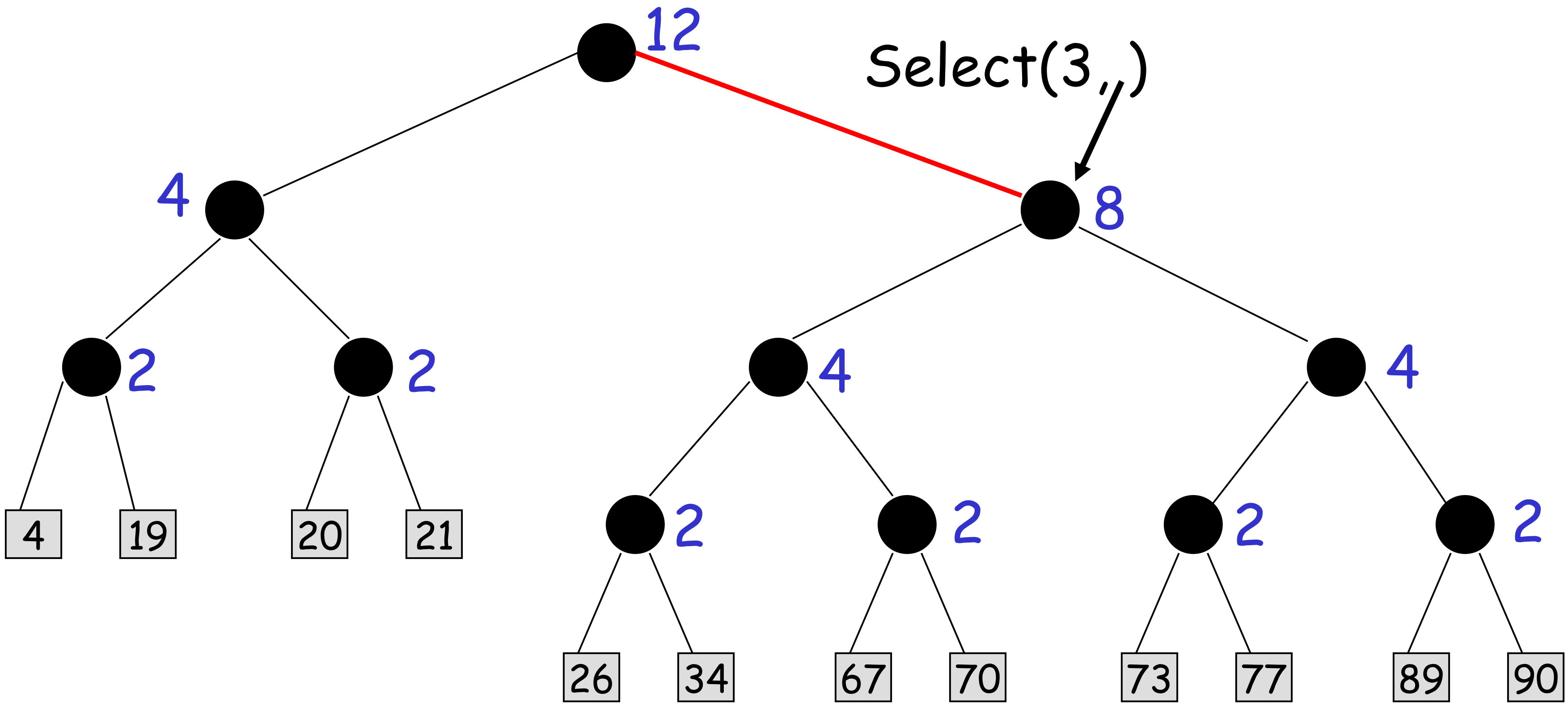
For each node v store # of leaves in the subtree of v



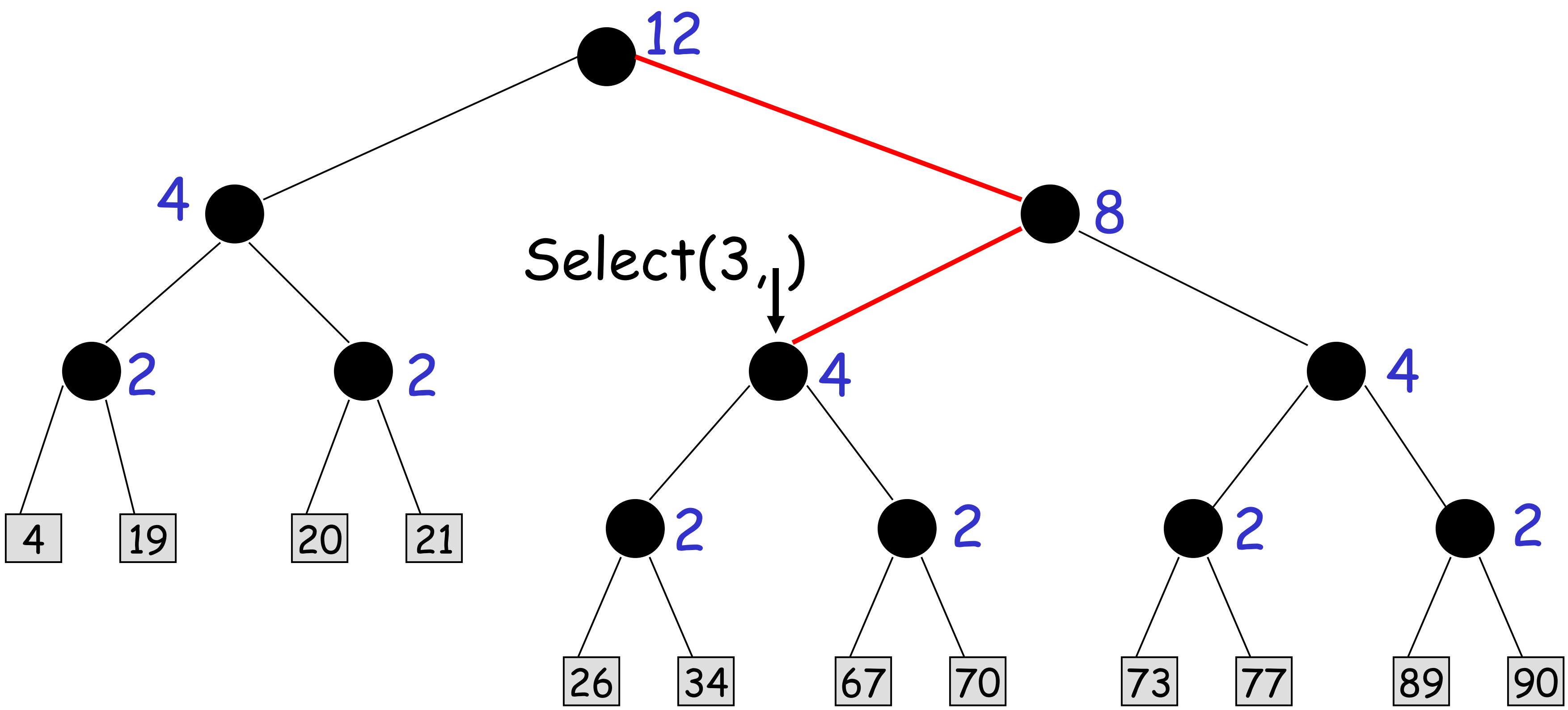
Select(7,T)



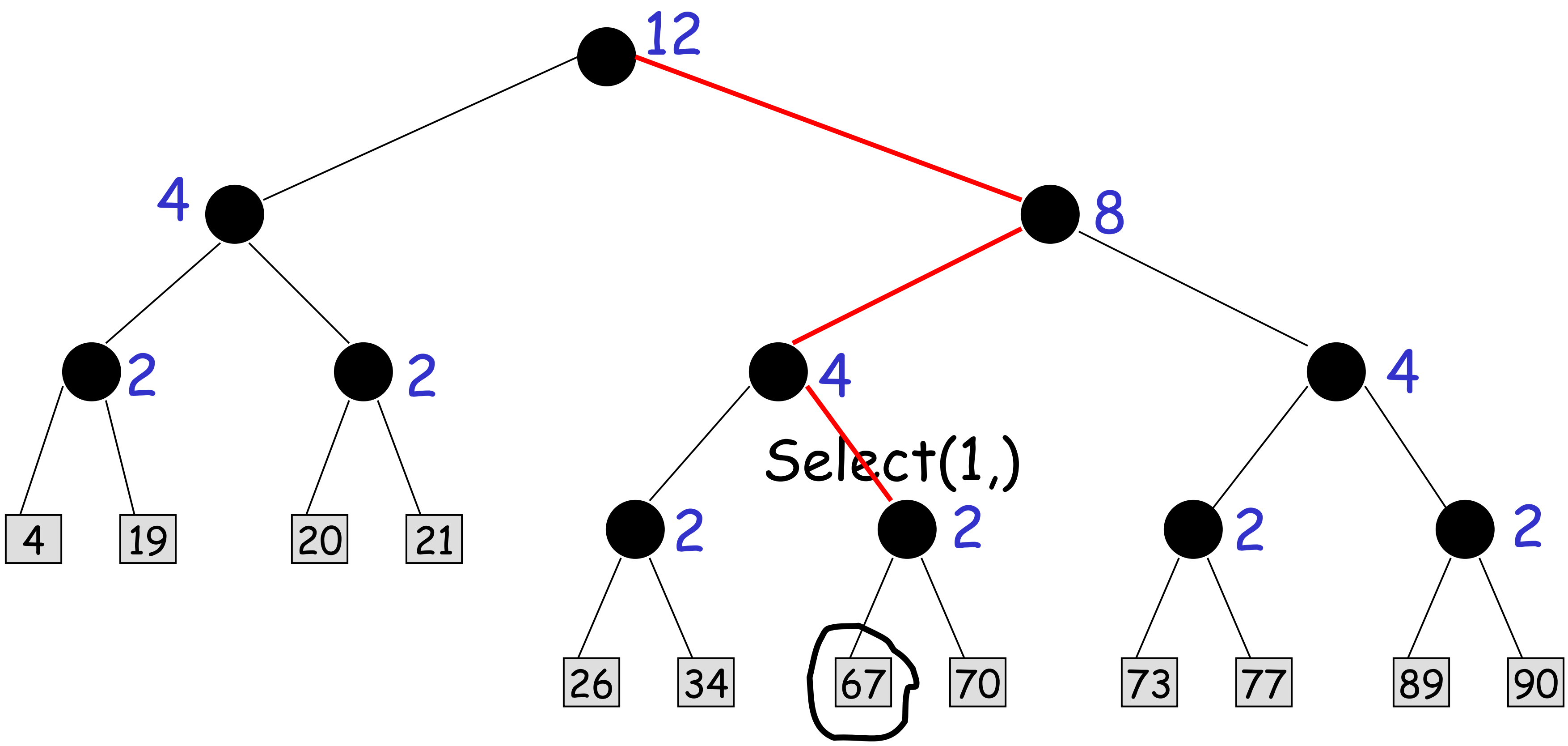
Select(7,T)



Select(7,T)



Select(7,T)

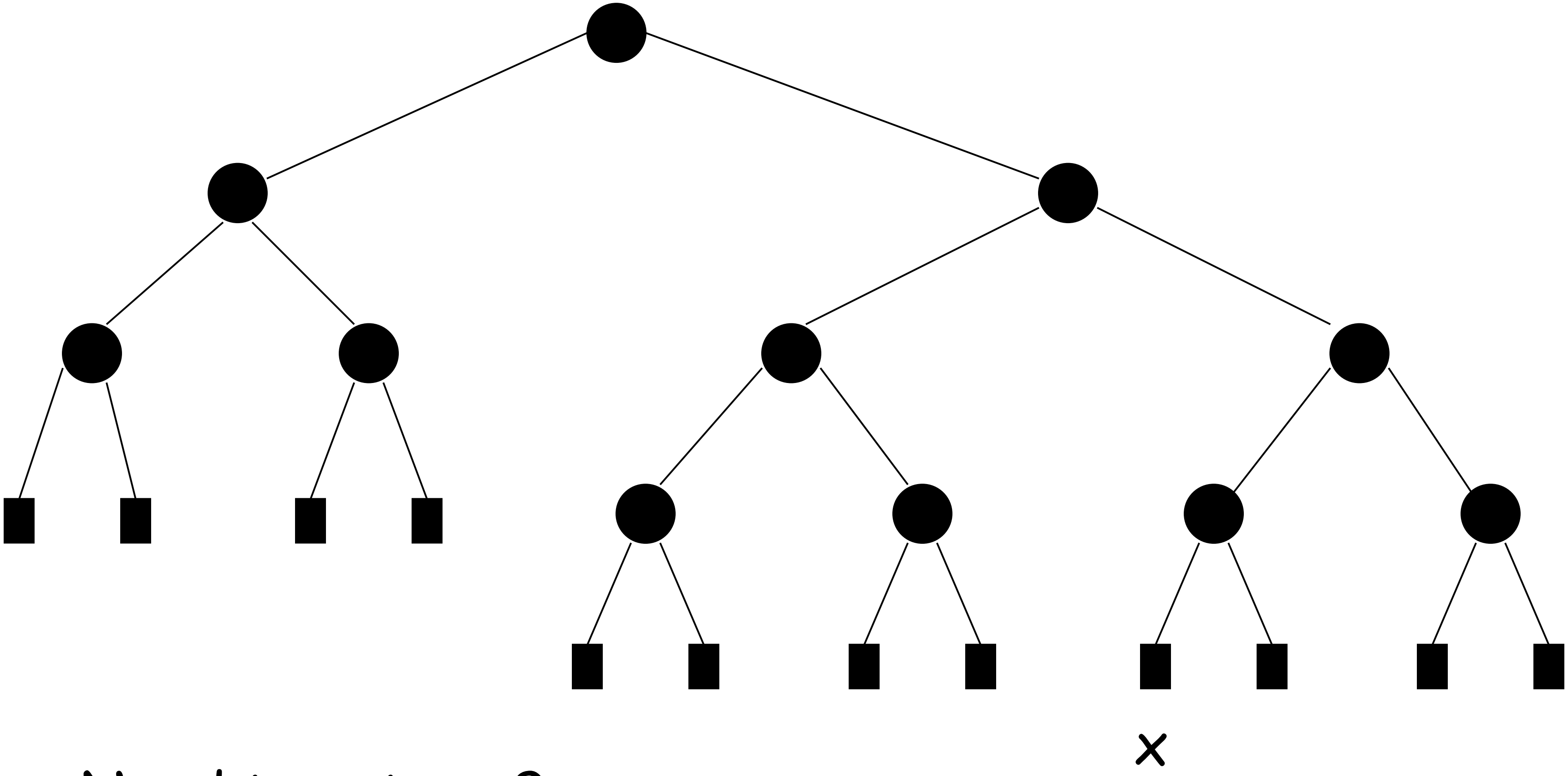


O(logn) worst case time, in case of balanced tree

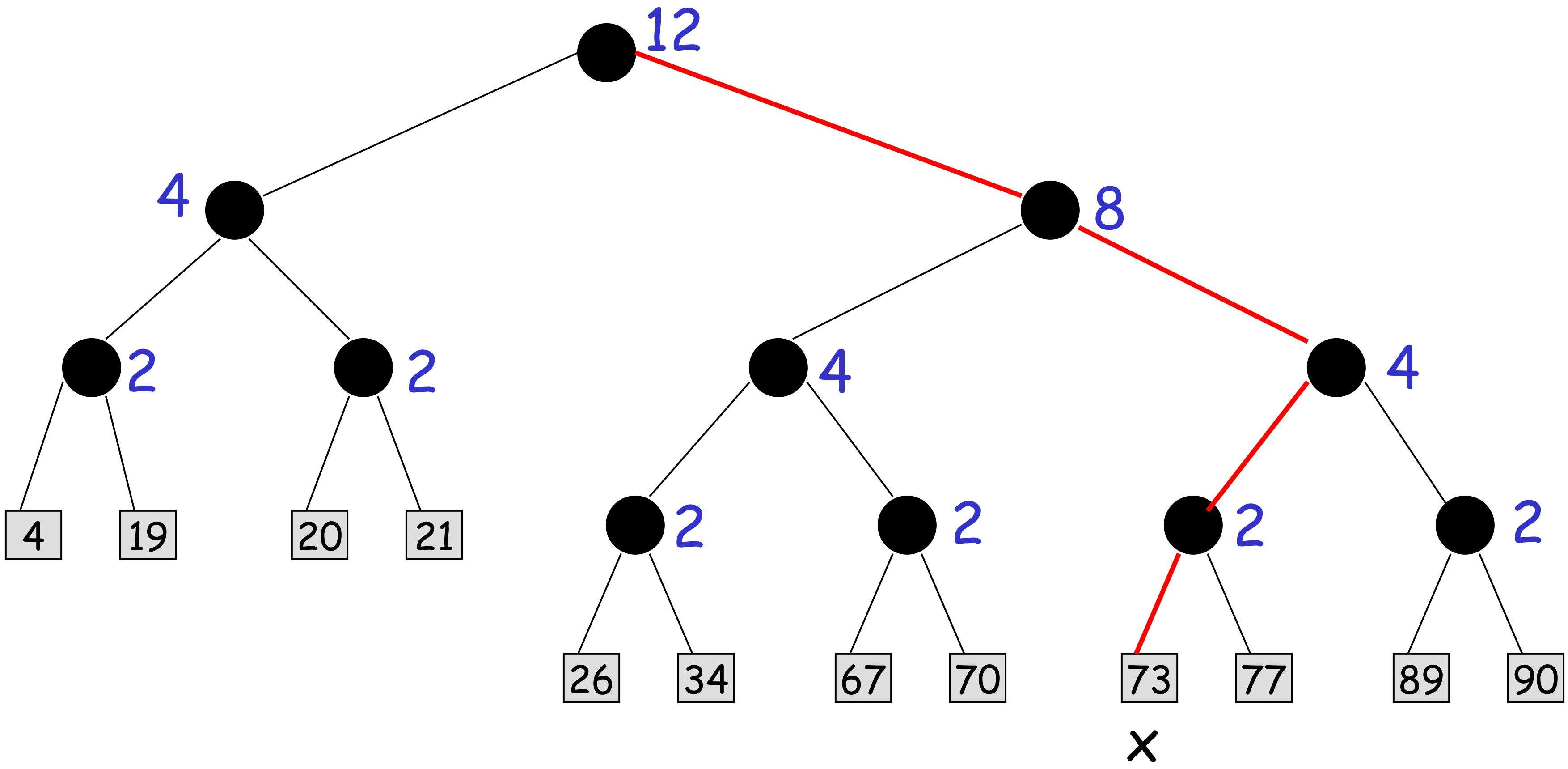
Rank(x,T)

- Return the index of x in T

Rank(x,T)

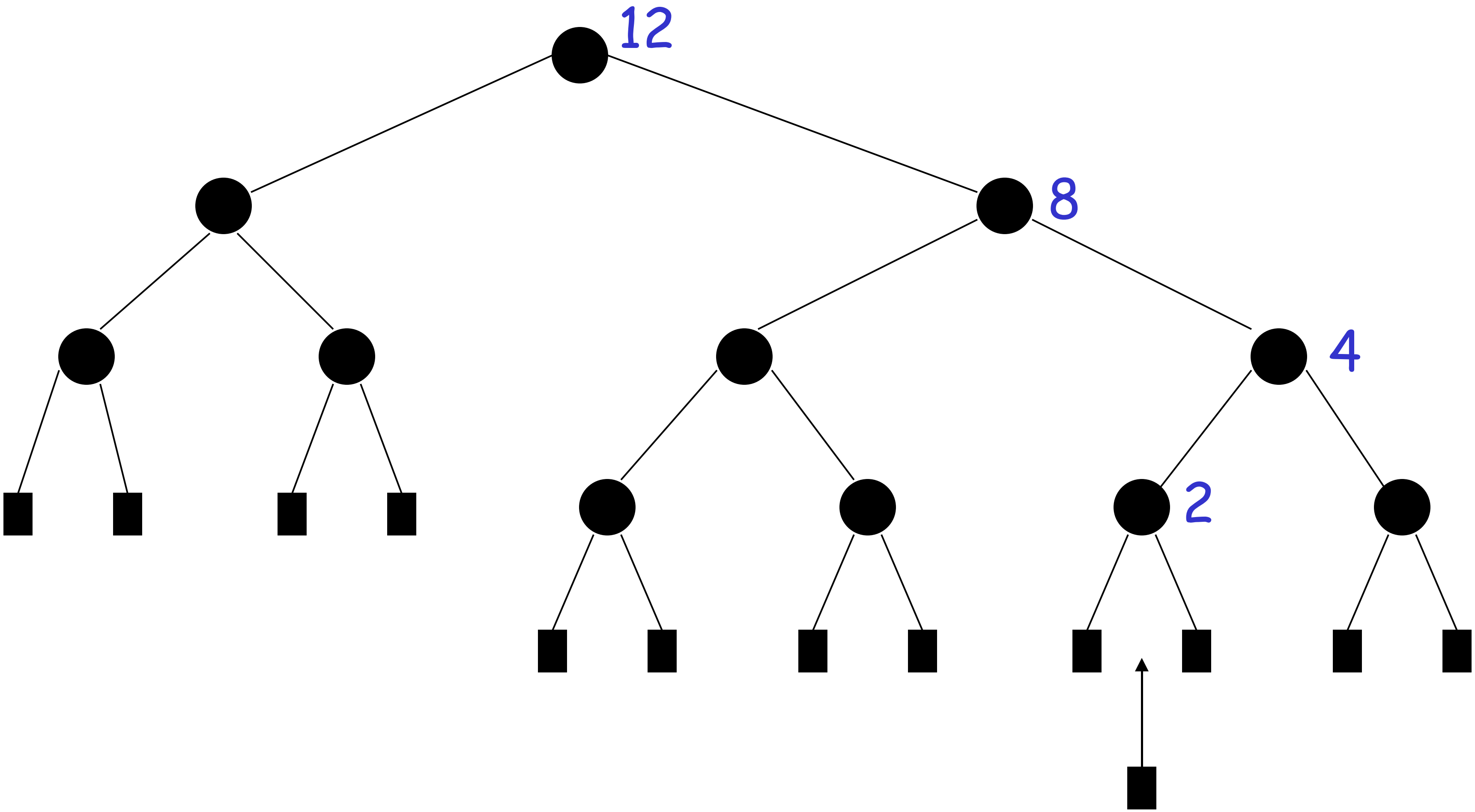


Need to return 9

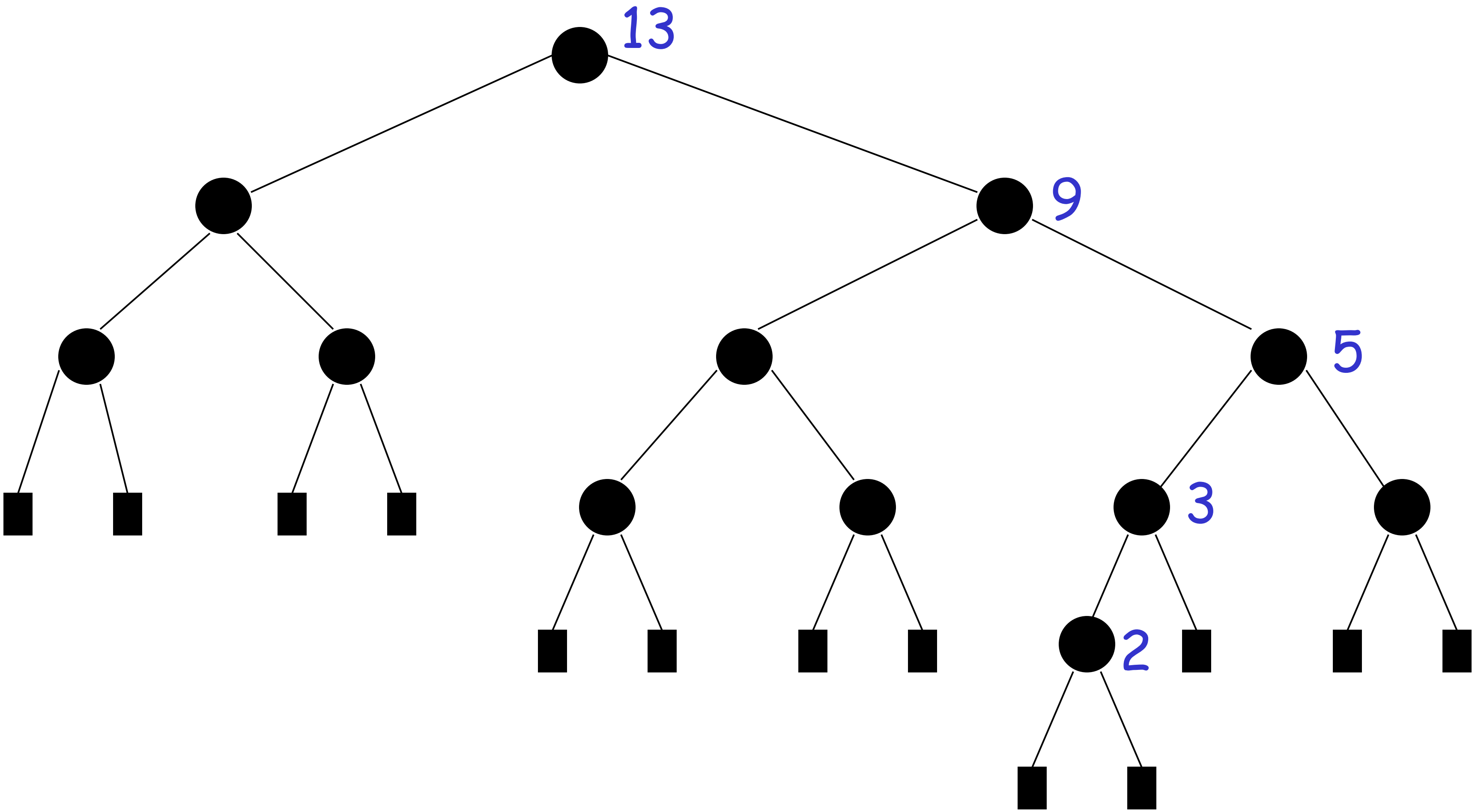


Sum up the sizes of the subtrees to the left of the path

Insert (non-balanced tree)



Insert (cont)



update only O(h) sizes

Summary

- Insertion and deletion and other dictionary operations still take O(log n) time for balanced trees

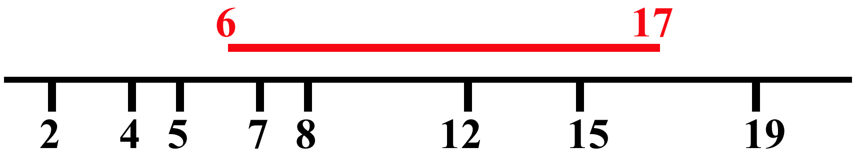
Orthogonal range searching

Range trees

Reporting (1-D)

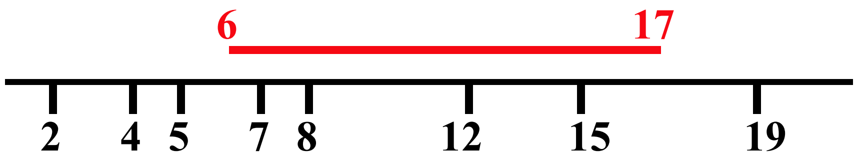
Given a set of points S on the line, preprocess them to build structure that allows efficient queries of the from:

Given an interval $I=[x_1,x_2]$ find all points in S that are in the interval.

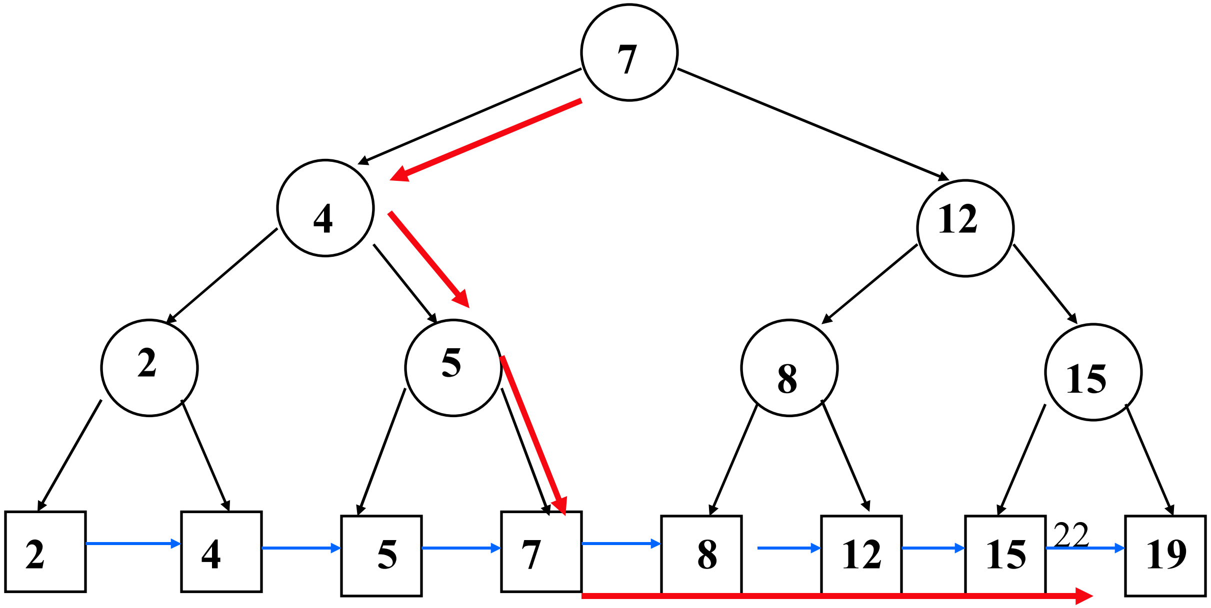


Reporting (1-D)

Build a balanced tree over the points
Concatenate them in a list



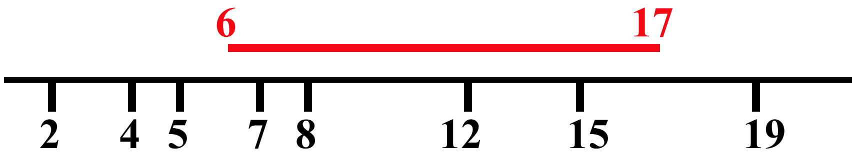
query: O(log n+k)
space: O(n)



Counting (1-D)

Given a set of points S on the line, preprocess them to build structure that allows efficient queries of the from:

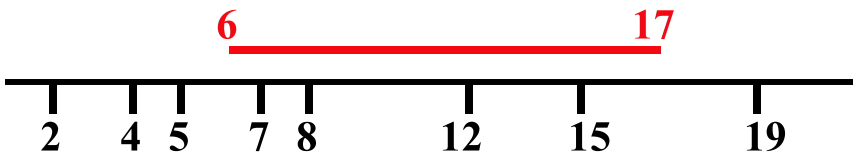
Given an interval $I=[x_1,x_2]$ return the # of points in the interval



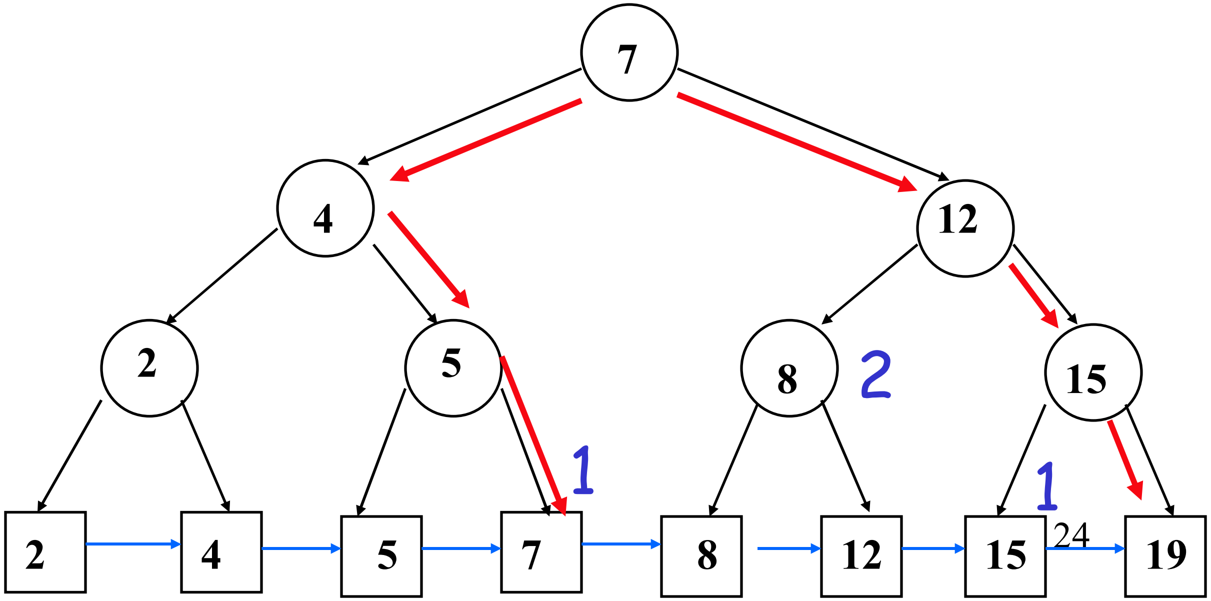
Counting (1-D)

Build a balanced tree over the points, with subtree sizes.

Return: $Rank(x_2,T) - Rank(x_1,T)+1$



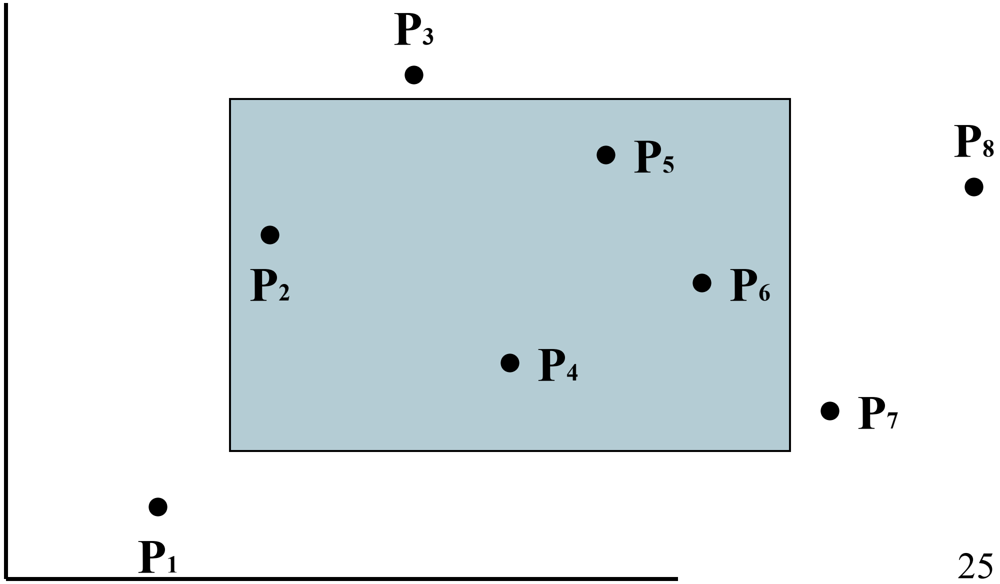
query: O(log n)
space: O(n)



Reporting (2-D)

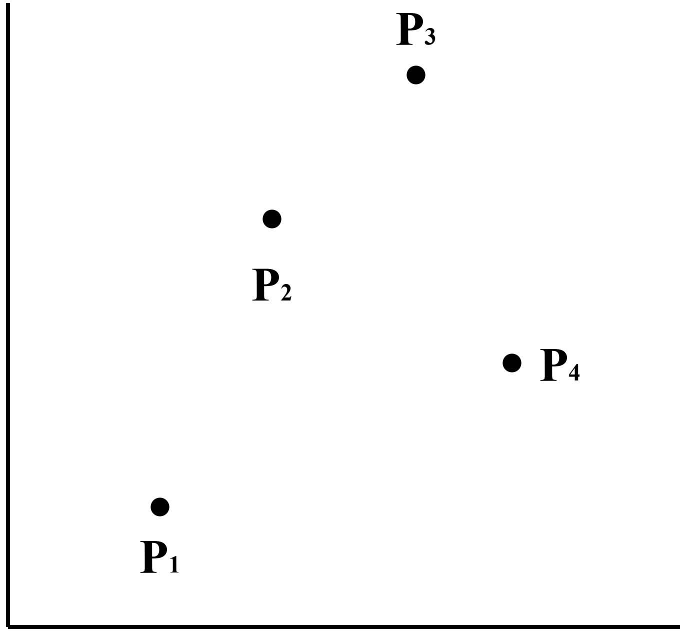
Given a set of points S on the plane, preprocess them to build structure that allows efficient queries of the from:

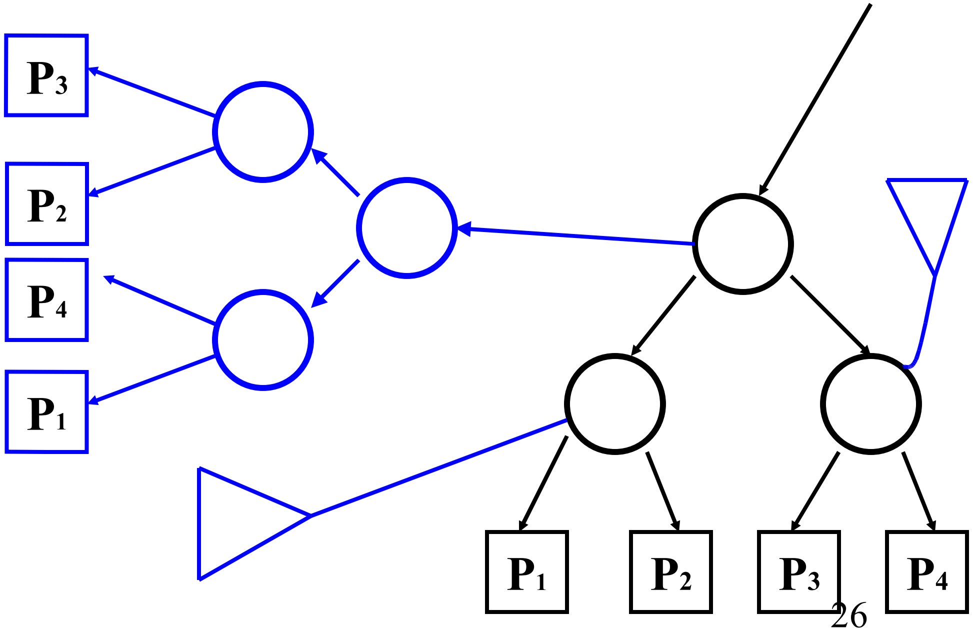
Given an rectangle $R=[x_1,x_2][y_1,y_2]$ find all points in S that are in the rectangle.



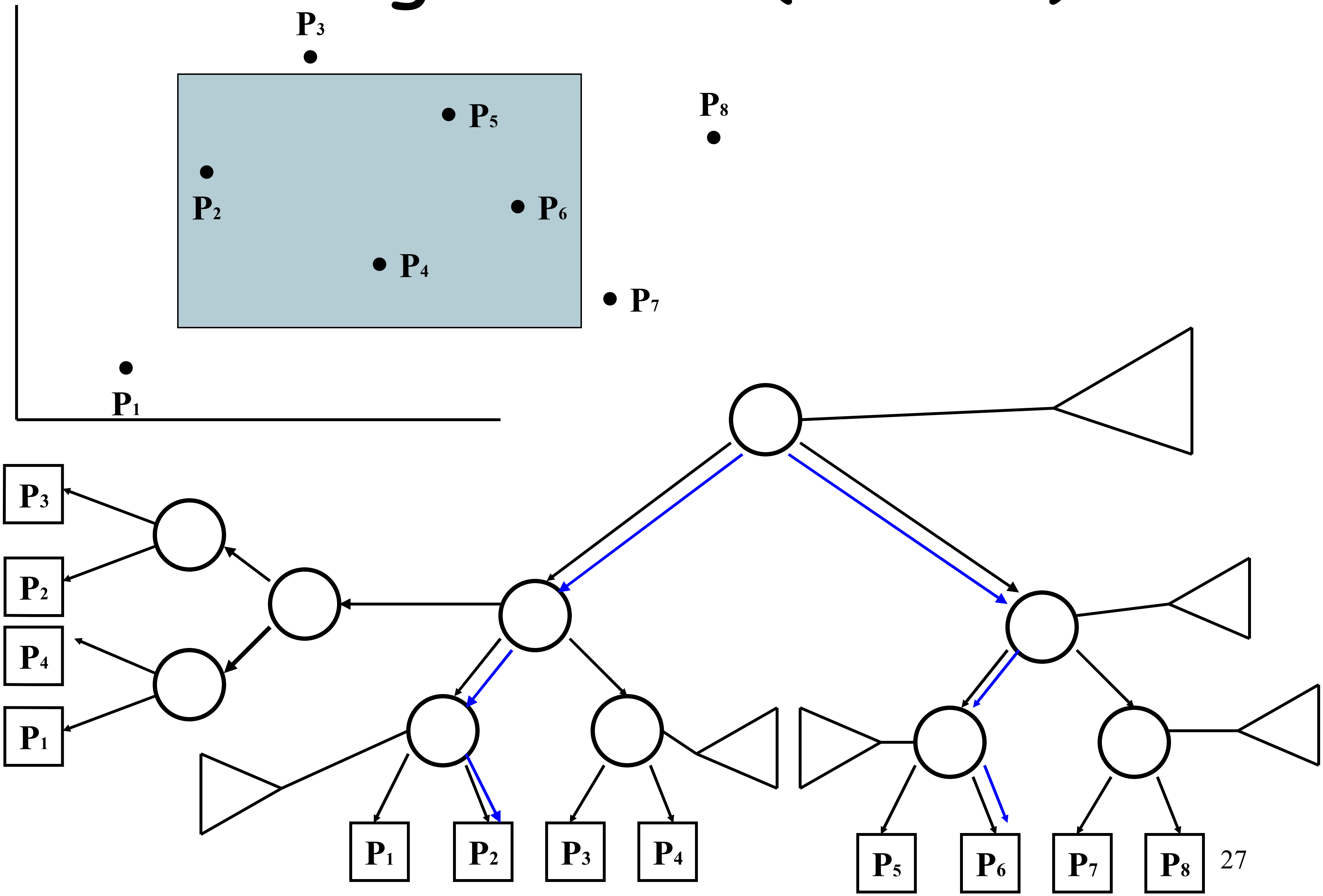
Range Trees (2-D)

Maintain the points in a balanced search tree ordered by x-coor. In each internal node maintain the points in its subtree in a balanced search tree ordered by y

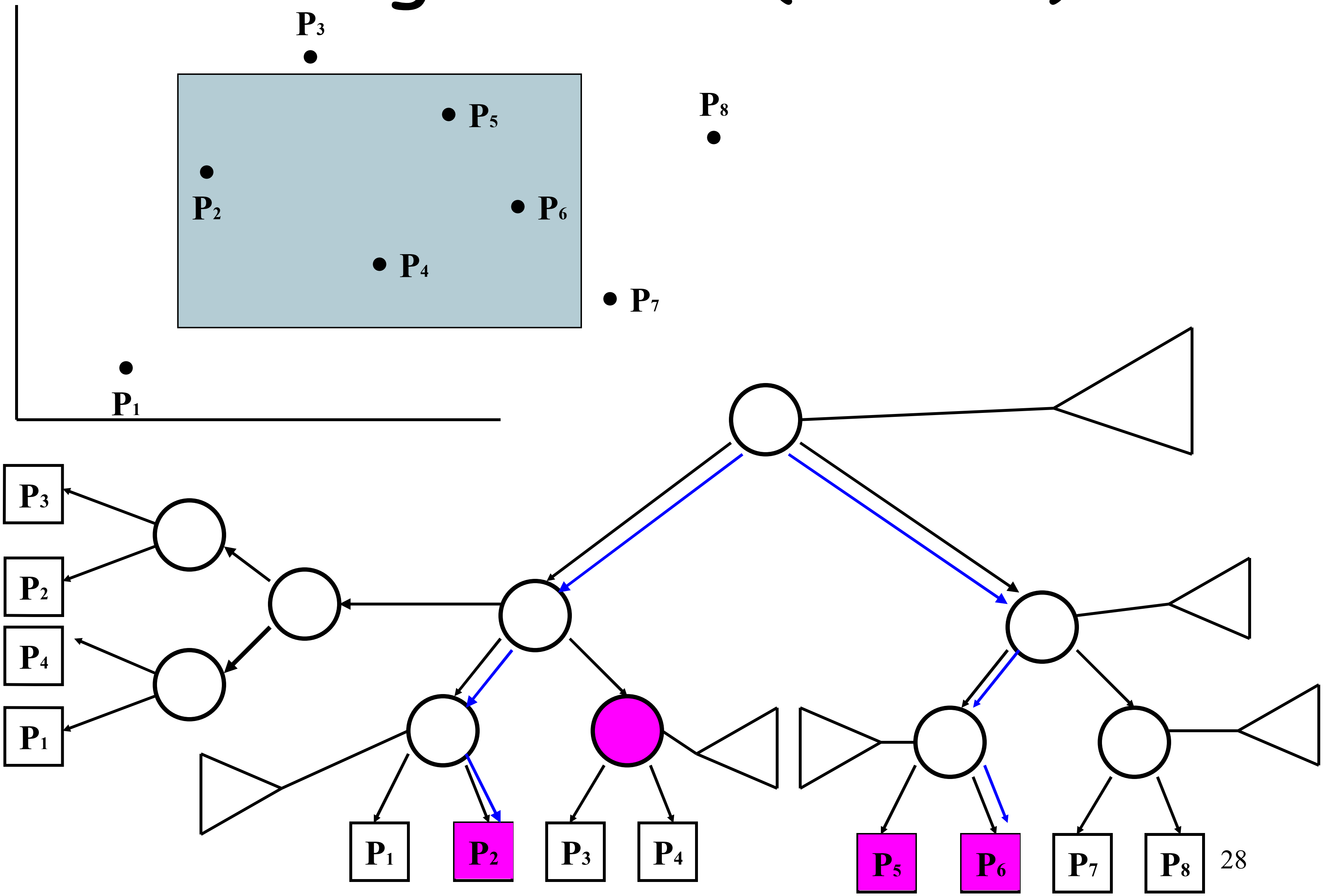




Range Trees (Contd.)



Range Trees (Contd.)



Query processing

- Search by the first dimension gives us O(logn) trees which together contain the output.
- We search each of these trees to get the answer

Analysis (2-D)

- Space $O(n\log n)$
- Query $O(\log^2 n+k)$

Further facts

- Generalizes to d-dimensions

שאלה 1

- תארו מבנה נתונים המתחזק קבוצה של נקודות במישור, כאשר כל נקודה piמיוצגת כזוג מספרים (xi,yi).
- מבנה הנתונים צריך לתמוך בפעולות ,insert(x,y) (delete(x,y), find(x,y בזמן גרוע ביותר O(log n), וכמו כן תומך בפעולה CountPointsAtDistance(d1,d2), המבצעת את הפעולה הבאה:

במבנה שמרחקן מהראשית גדול מ d1 וקטן מ d2 (זיכרו כי מרחק נקודה (x,y) מהראשית הוא).

- (CountPointsAtDistance(d1,d2 צריכה להתבצע בסיבוכיות הנמוכה ביותר האפשרית.

תשובה 1

מתחת לכל עלה מוחזק עץ מאוזן שבו המפתחות המתאימים (x,y) בסדר לקסיקוגרפי. העץ הראשי בנוי כעץ ORDER STAT בו בכל צומת שומרים את מספר המפתחות שנמצאים בתת העץ.

האלגוריתם

מוצאים את הנקודה הרחוקה ביותר מהראשית שמרחקה קטן מ-d2.

- מחשבית את הסדר הסטטיסטי שלה: a.
- מוצאים את הנקודה הקרובה ביותר לראשית שמרחקה גדול מ-d1.
- מחשבית את הסדר הסטטיסטי שלה: b.
- מחזירים את a-b+1.
- כל אחת מהפעולות הנ"ל מבוצעת בזמן

שאלה 2

רוצים לבנות מבנה נתונים לתחזוקת מספרים טבעיים התומך בפעולות insert(x), delete(x), find(x), עבור מספר טבעי x, ובפעולת multiple_of_5() המחזירה שני ערכים שונים במבנה x, y המקיימים y=5x, אם קיימים כאלו (אחרת מחזירה false). הציעו מבנה התומך בפעולות עם סיבוכיות WC הטובה ביותר.

תשובה 2

- מבנה הנתונים יהיה עץ חיפוש מאוזן שמכיל את כל האיברים. רשימה דו כיוונית שמכילה את הזוגות שהמנה ביניהם היא 5. כמו כן יש הצבעות הדדיות בין האיברים בשני המבנים.
- פעולות הוספה, מחיקה וחיפוש מבוצעות על העץ. במקרה של הוספה ומחיקה יש לעדכן גם את הרשימה (כלומר להוסיף או להוריד זוג מהרשימה, לפי הצורך).
- פעולת Multiple_of_5() תבוצע על ידי החזרת ראש הרשימה המקושרת.

שאלה 3

בונה בזמן <u>O(nW.C)</u> מבנה נתונים התומך בפעולה find בלבד, באופן ש: זמן ה W.C של find הוא **O(n)**, אולם על **לפחות n/log n** מהאיברים, זמן הריצה של find הוא **O(log n)**.

שימו לב כי המבנה אינו דינאמי, כלומר אינו צריך לתמוך ב insert ו delete.

תשובה 3

- <u>תיאור האלגוריתם והוכחת נכונות</u>
- לוקחים **$n/\log n$ איברים** ומכניסים **לעץ חפוש מאוזן**. עלות כוללת היא $O(n)$. שאר האיברים מוכנסים לרשימה.
- בעת בצוע $find$ מחפשים קודם בעץ ואחר כך ברשימה. עלות חפוש על כל אבר בעץ היא $O(\log n)$. שאר האיברים – $O(n)$.

שאלה 4

- נגדיר פרמוטציה מסדר (n,m) באופן הבא:
- מסדרים את המספרים 1,2,3,...,n במעגל, ומצביעים על המספר 1.
- מבצעים m צעדים לאורך המספרים **שנותרו** על המעגל.
- מדפיסים את המספר אליו הגענו, ומוחקים אותו מהמעגל. אם נותרו מספרים חוזרים לב', ולא עוצרים. מספר שנמחק, **אינו** נספר במניין הצעדים בשלב ב'.

שאלה 4 (המשך)

לדוגמא: הפרמוטציה מסדר (7,3) היא 4,7,3,1,6,2,5 (ודאו כי אתם מבינים מדוע!)

- תארו אלגוריתם <u>**יעיל**</u> המקבל מספר n, ומחזיר את הפרמוטציה מסדר $(n,n/2)$.

- תשובה 4: זמן ריצת האלגוריתם $O(n\log n)$

תשובה - 4 הסבר

- נחזיק את המספרים 1 עד n בעץ חיפוש בינארי עם סדר סטטיסטי. ז"א כל קודקוד בעץ ישמור בנוסף את מספר הקודקודים שמתחתיו (1+) בשדה size. נתחיל כשפוינטר מצביע על האיבר 1.
- כשאנחנו באיבר שאינדקסו I, נקפוץ לאיבר שאינדקס שלו הוא (I+n/2)mod k בעזרת OS_Select כאשר k הוא מספר האיברים שנותרו בעץ (כלומר size של השורש).
- נדפיס את האיבר שהגענו אליו, ונמחוק אותו.
- נבצע את 1 ו- 2 עד שלא יישארו איברים בעץ.