בעיית קבוצות זרות – Union/Find

חומר קריאה לשיעור זה:

Chapter 22- Data Structures for Disjoint Sets (440 - 462)

©Geiger & Itai, 2001

קבוצות זרות – בעיית Union/Find

נתון עולם של איברים U, ‏|U|=n. לצורך הנוחות נניח U = {1,2,…,n}.

מבנה נתונים לשמירת קבוצות זרות תומך בפעולות הבאות:

1. Makeset(i) – מחזיר קבוצה חדשה בעלת איבר בודד i.
2. Find(i) – מחזיר את הקבוצה לה שייך האיבר i.
3. Union(p,q) – מאחד את הקבוצות p ו- q, כלומר מחזיר קבוצה חדשה המכילה את איחוד האיברים בקבוצות p,q. (הקבוצות p,q המקוריות חדלות מלהתקיים.)

©cs,Technion

בעיית Union/Find – קבוצות זרות

נתון עולם של איברים U, $|U|=n$. לצורך הנוחות נניח $U = \{1,2,\ldots,n\}$.

מבנה נתונים לשמירת קבוצות זרות תומך בפעולות הבאות:

1. Makeset(i) – מחזיר קבוצה חדשה בעלת איבר בודד i.
2. Find(i) – מחזיר את הקבוצה לה שייך האיבר i.
3. Union(p,q) – מאחד את הקבוצות p ו- q, כלומר מחזיר קבוצה חדשה המכילה את איחוד האיברים בקבוצות p,q. (הקבוצות p,q המקוריות חדלות מלהתקיים.)

לדוגמא, נניח ברגע נתון הגענו למצב: {1,3} {5} {2,4,6,7}

p = Find (6)
q = Find (5)

r = Union (p,q)

הקבוצה שהמשתנה *r* שומר היא {5,2,4,6,7}

s = Find (6)

המשתנים p ו-*q* אינם מחזיקים יותר קבוצות.
והמשתנה *s* מחזיק את אותה קבוצה כמו *r*.

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,...,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,...,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},...,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

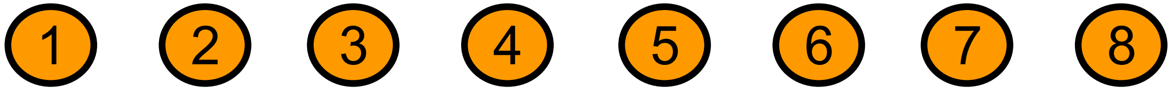
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i $\le$ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

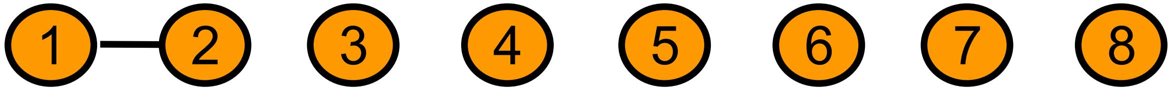
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

<u>פתרון</u>: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

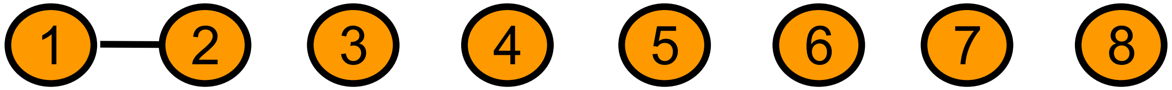
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

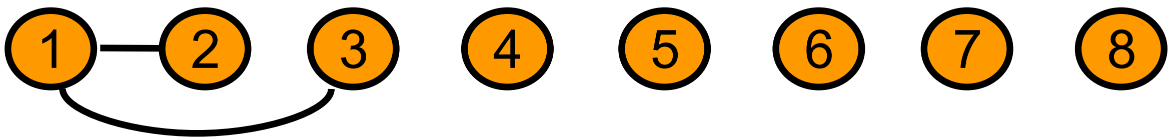
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

<u>פתרון</u>: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

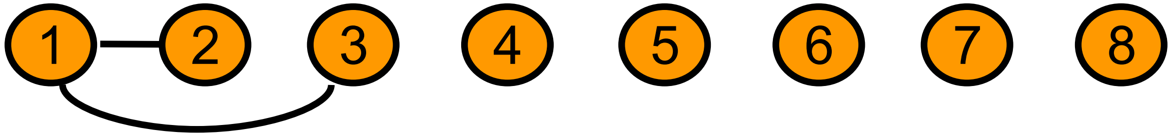
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

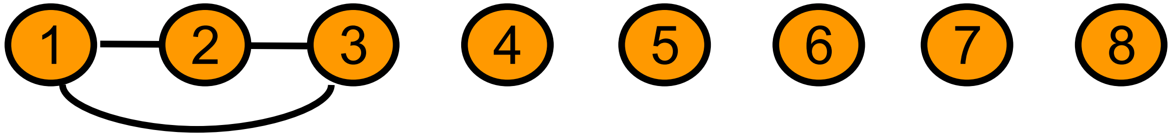
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

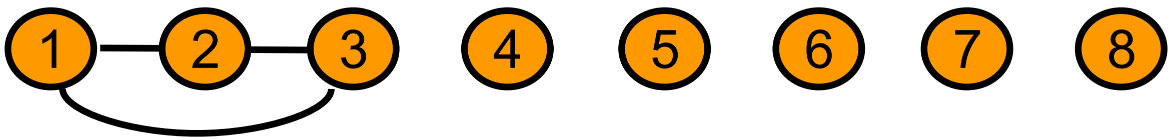
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

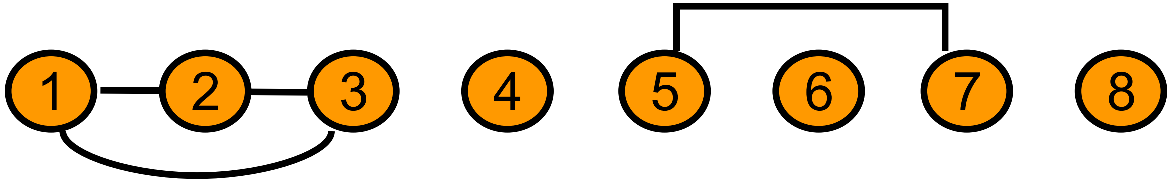
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

<u>פתרון</u>: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i $\le$ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

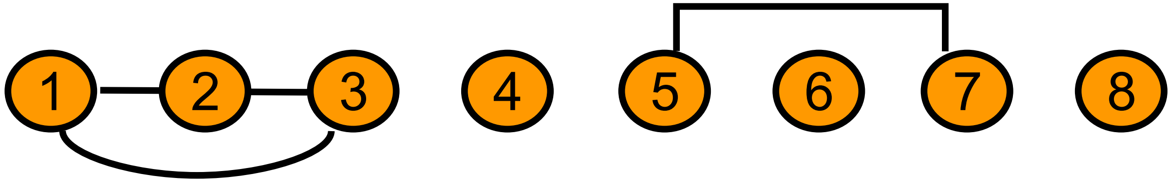
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(5,7)	{1,2,3}			{4}	{5,7}	{6}		{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

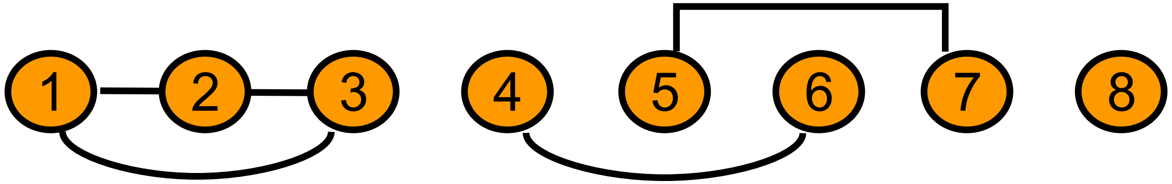
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i $\le$ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(5,7)	{1,2,3}			{4}	{5,7}	{6}		{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

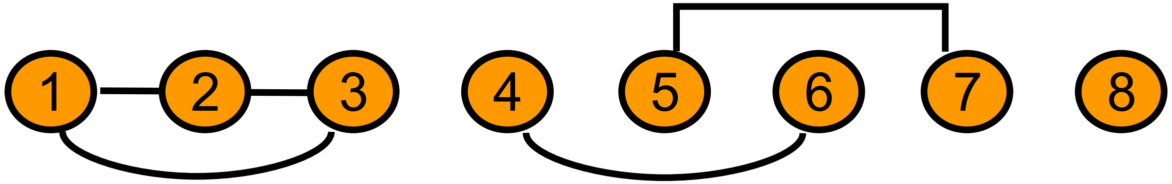
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(5,7)	{1,2,3}			{4}	{5,7}	{6}		{8}
(4,6)	{1,2,3}			{4,6}	{5,7}			{8}

©cs,Technion

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

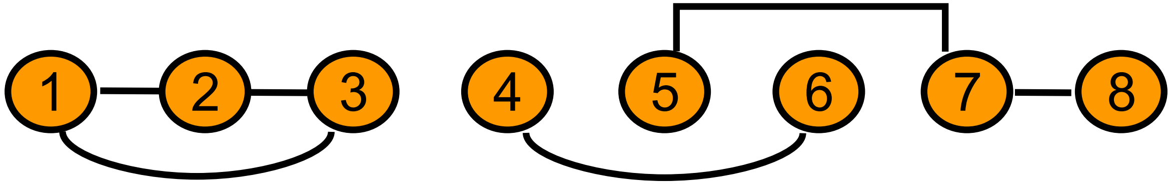
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(5,7)	{1,2,3}			{4}	{5,7}	{6}		{8}
(4,6)	{1,2,3}			{4,6}	{5,7}			{8}

דוגמא היפותטית לשימוש

נתונות n ערים {1,…,n}. בתחילה כל הערים מנותקות. כל זמן מה בונים כביש ישיר בין שתי ערים. יש לאפשר את הפעולות הבאות:

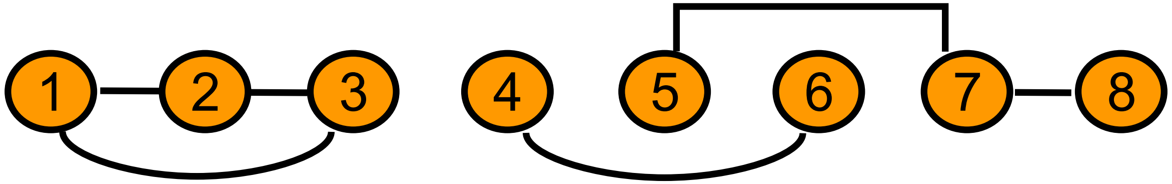
Add-Road(x,y) -- הוסף כביש ישיר בין שתי ערים x ו- y.

Check-Connectivity(x,y) -- קבע האם שתי הערים x ו- y מחוברות במסלול כלשהו.

פתרון: ניצור n קבוצות: {1}, {2},…,{n} ע"י: for (i=1; i ≤ n; i++) Makeset(i) ;

Add-Road(x,y) ממומשת ע"י Union(Find(x),Find(y)).

Check-Connectivity(x,y) מחזירה "אמת" אם ורק אם Find(x)=Find(y)



כביש שהוסף	אוסף הקבוצות הזרות							
	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}
(1,2)	{1,2}		{3}	{4}	{5}	{6}	{7}	{8}
(1,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(2,3)	{1,2,3}			{4}	{5}	{6}	{7}	{8}
(5,7)	{1,2,3}			{4}	{5,7}	{6}		{8}
(4,6)	{1,2,3}			{4,6}	{5,7}			{8}
(7,8)	{1,2,3}			{4,6}	{5,7,8}			

פתרון נאיבי ראשון

נשתמש במערך A מסוג SET בגודל n ובמונה counter המאותחל ל- 0. בתא A[i] נשמור את שם הקבוצה אליה שייך האיבר i. לדוגמא: {2,4,6,7} {5} {1,3} מיוצגות ע"י המערך הבא (במימוש זה SET הוא פשוט int):

	1	2	3	4	5	6	7
A	4	12	4	12	5	12	12

Makeset(i) ממומשת ע"י A[i]=++counter.

Find(i) ממומשת ע"י A[i].

פתרון נאיבי ראשון

נשתמש במערך A מסוג SET בגודל n ובמונה counter המאותחל ל- 0. בתא A[i] נשמור את שם הקבוצה אליה שייך האיבר i. לדוגמא: {2,4,6,7} {5} {1,3} מיוצגות ע"י המערך הבא (במימוש זה SET הוא פשוט int):

	1	2	3	4	5	6	7
A	4	12	4	12	5	12	12

Makeset(i) ממומשת ע"י A[i]=++counter.

Find(i) ממומשת ע"י A[i].

Union(p,q) ממומשת ע"י הגדלת ה-counter באחד, מעבר על המערך A וכתיבת ערך ה-counter בכל מקום שבו כתוב p או q.הפונקציה מחזירה את הערך של counter.

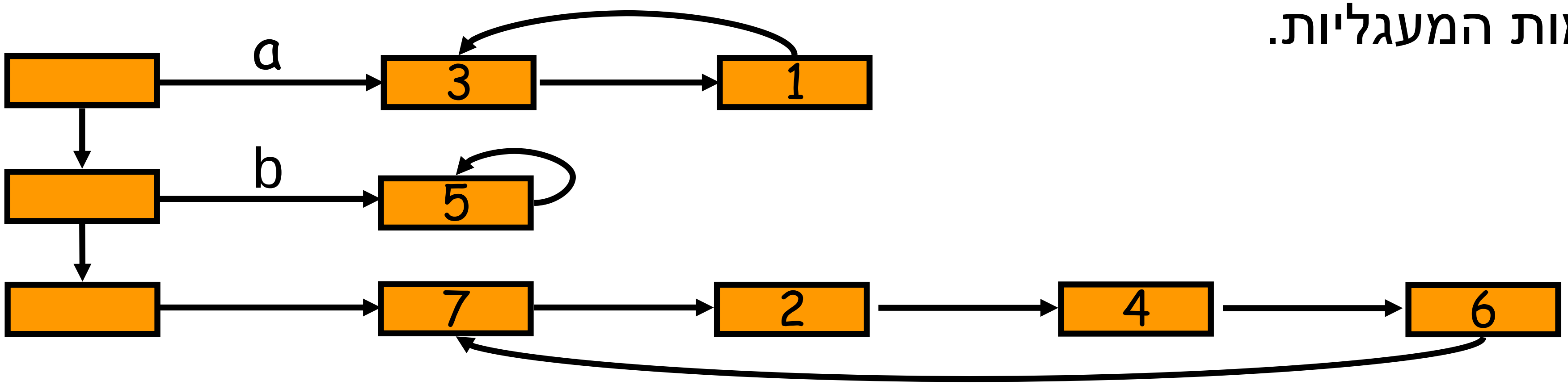
דוגמא: לאחר Union(p,q) כאשר p=4,q=5, מאוחדות הקבוצות p ו-q ומקבלים:

	1	2	3	4	5	6	7
A	13	12	13	12	13	12	12

סיבוכיות הזמן של Find ו-Makeset היא O(1) ושל Union היא O(n).

פתרון נאיבי שני

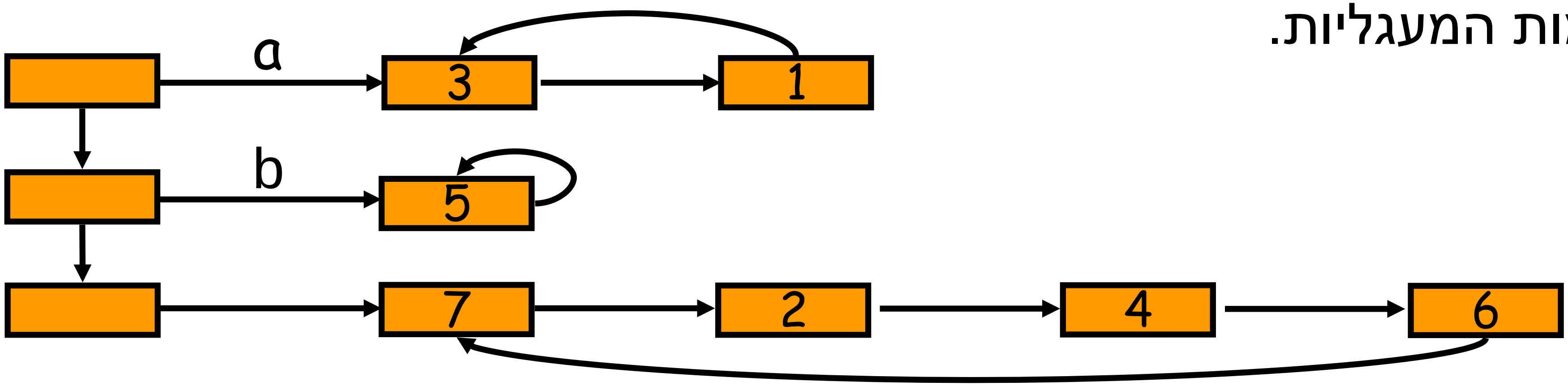
נייצג כל קבוצה כרשימה מעגלית. נחזיק רשימה מקושרת של מצביעים אל הרשימות המעגליות.



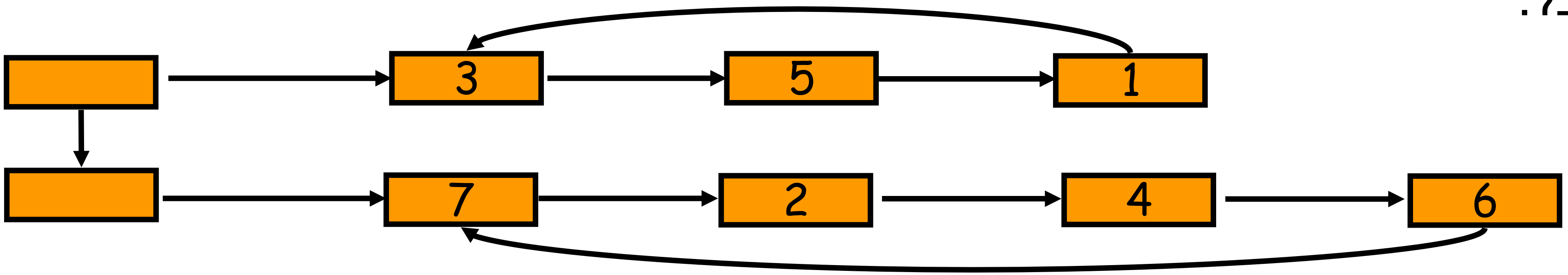
©cs,Technion

פתרון נאיבי שני

נייצג כל קבוצה כרשימה מעגלית. נחזיק רשימה מקושרת של מצביעים אל הרשימות המעגליות.



Union(p,q) -- ממומשת ע"י אחוד הרשימות המוצבעות ע"י p ו- q. זמן O(1).
במימוש זה SET הוא מצביע לצומת מסוג NODE. דוגמא: לאחר Union(a,b) נקבל:



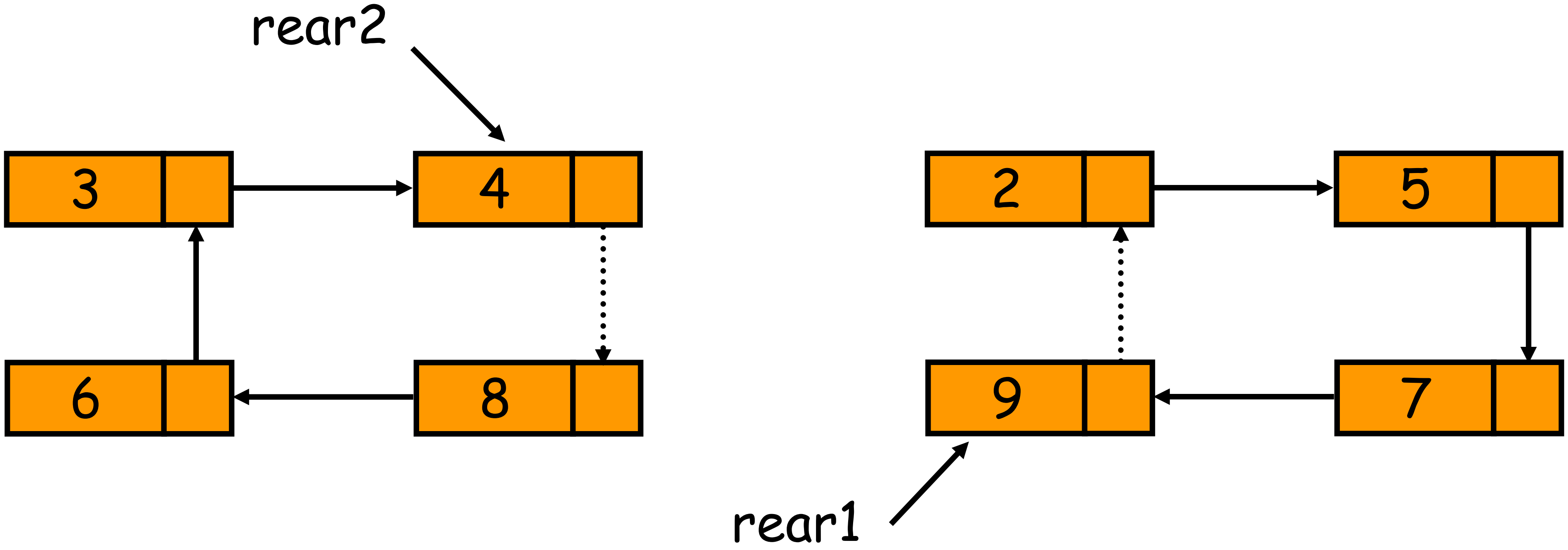
Find(i) ממומשת ע"י מעבר על כל הרשימות עד שנמצא האיבר i. זמן O(n).

Makeset(i) ממומשת ע"י הוספת רשימה עם איבר בודד. זמן O(1).

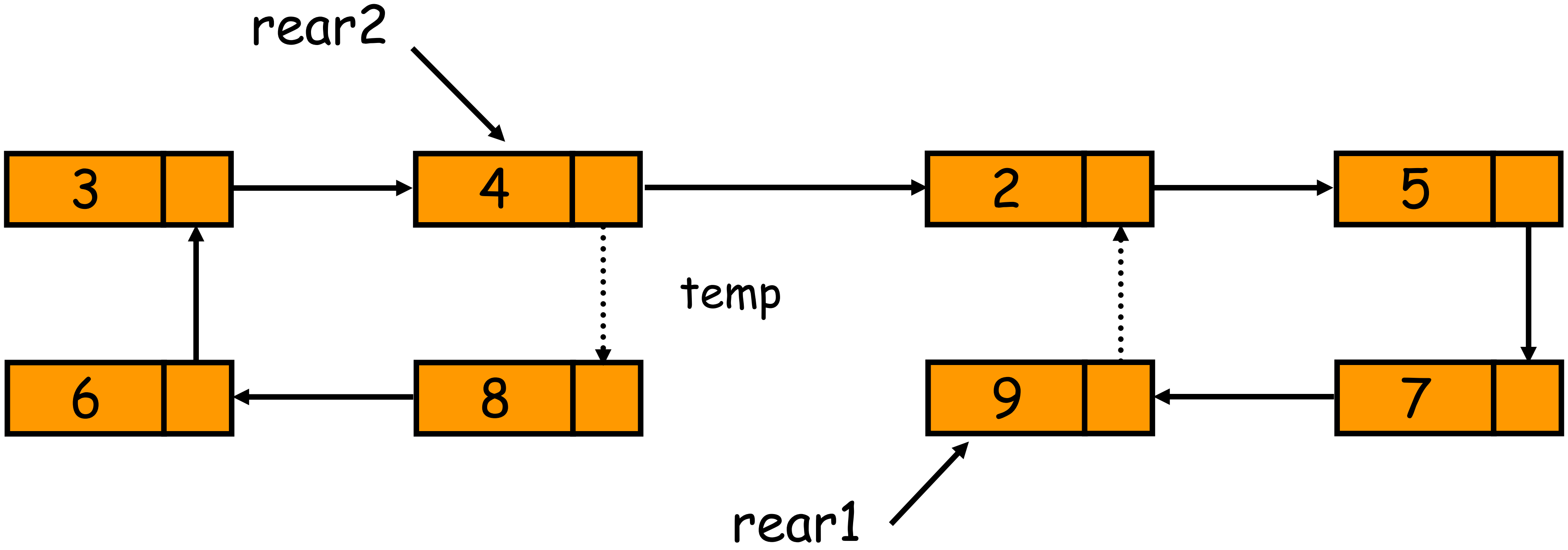
תזכורת: איחוד רשימות מעגליות

©cs,Technion

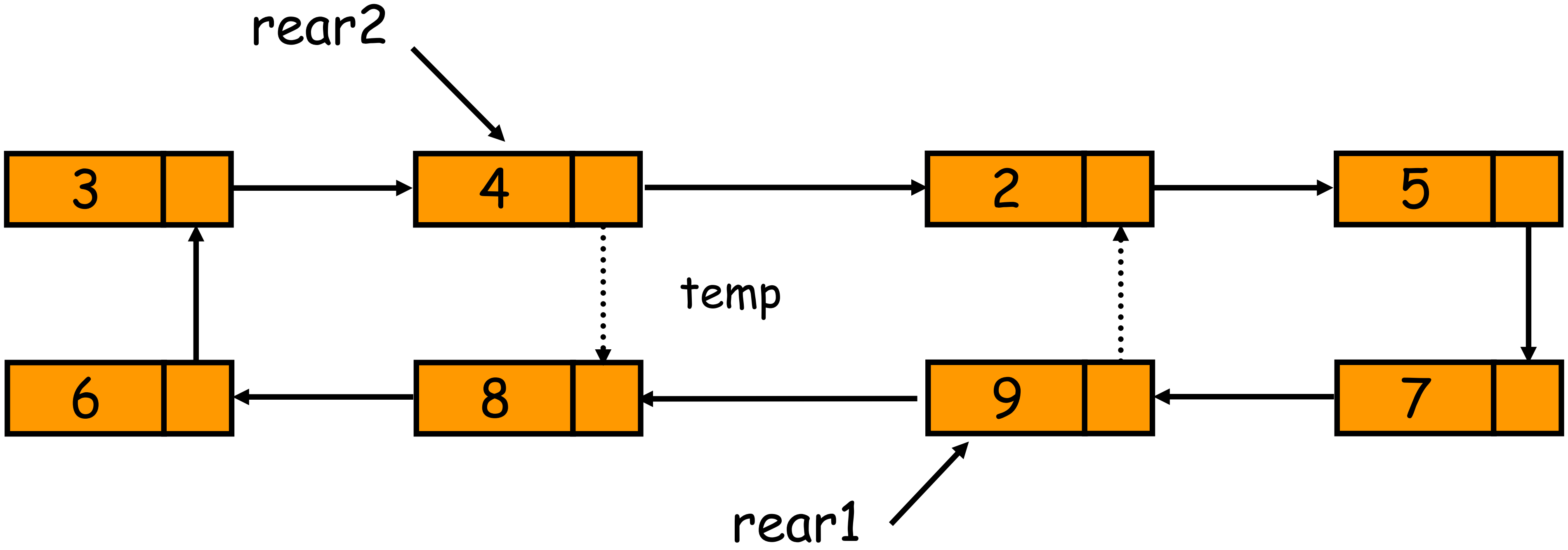
תזכורת: איחוד רשימות מעגליות



תזכורת: איחוד רשימות מעגליות

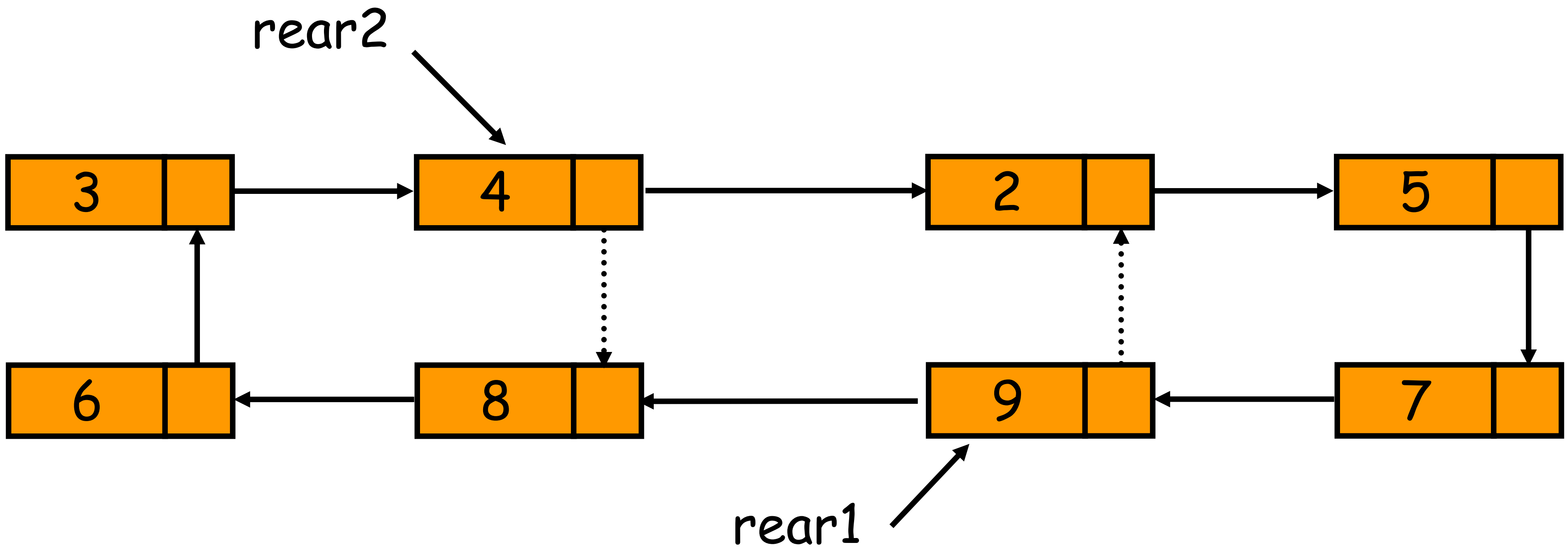


תזכורת: איחוד רשימות מעגליות



©cs,Technion

תזכורת: איחוד רשימות מעגליות



הערות

בפתרון הנאיבי הראשון סיבוכיות הזמן של פעולות Makeset, ו-Find היא O(1) ושל Union היא O(n).

בפתרון הנאיבי השני סיבוכיות הזמן של Find היא O(n) ושל פעולות Makeset, ו- Union היא O(1).

<u>הערה נוספת</u>: בהגדרת הבעיה שמות האיברים נבחרו להיות מספרים שלמים. ניתן להשתמש בשמות כלליים (כלומר מחרוזות תווים). לשם כך ניתן להשתמש במבנה נתונים הממיר ביעילות בין שמות כלליים ומספרים שלמים, למשל Hash table.

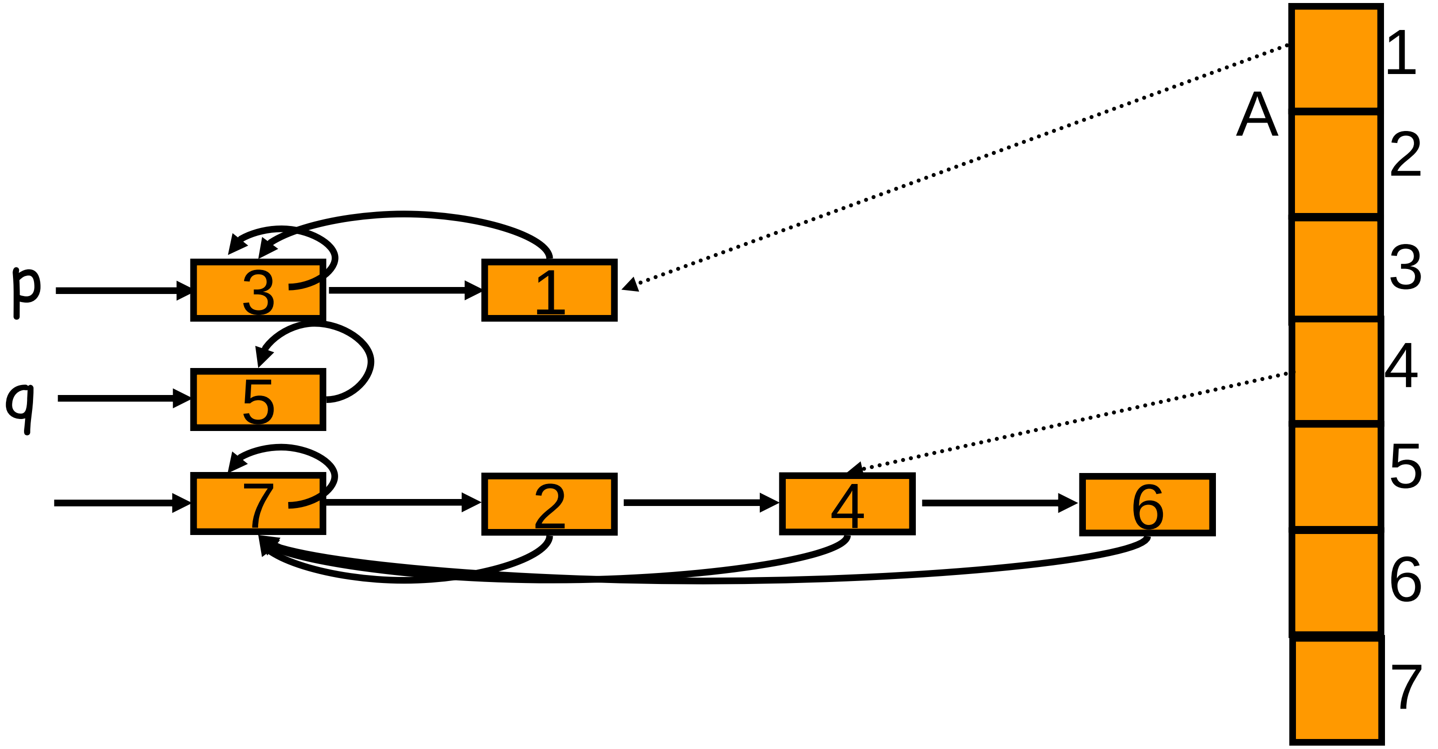
נפתח כעת פתרון ובו הזמן של פעולת Find הוא O(1) והזמן <u>המשוערך</u> (יוגדר בהמשך) של פעולת Union הוא O(log n) .

ולסיום נפתח מבנה נתונים בו הזמן המשוערך לפעולת Find הוא "כמעט קבוע" וזמן פעולת Union הוא O(1).

©cs,Technion

פתרון שלישי

נייצג כל קבוצה כרשימה. כל איבר ברשימה מצביע גם לראש הרשימה וגם לאיבר הבא ברשימה. נחזיק מערך מיפוי איברים A[i] ובו מצביע לאיבר i. קבוצה מוחזרת כמצביע לראש הרשימה המתאימה.



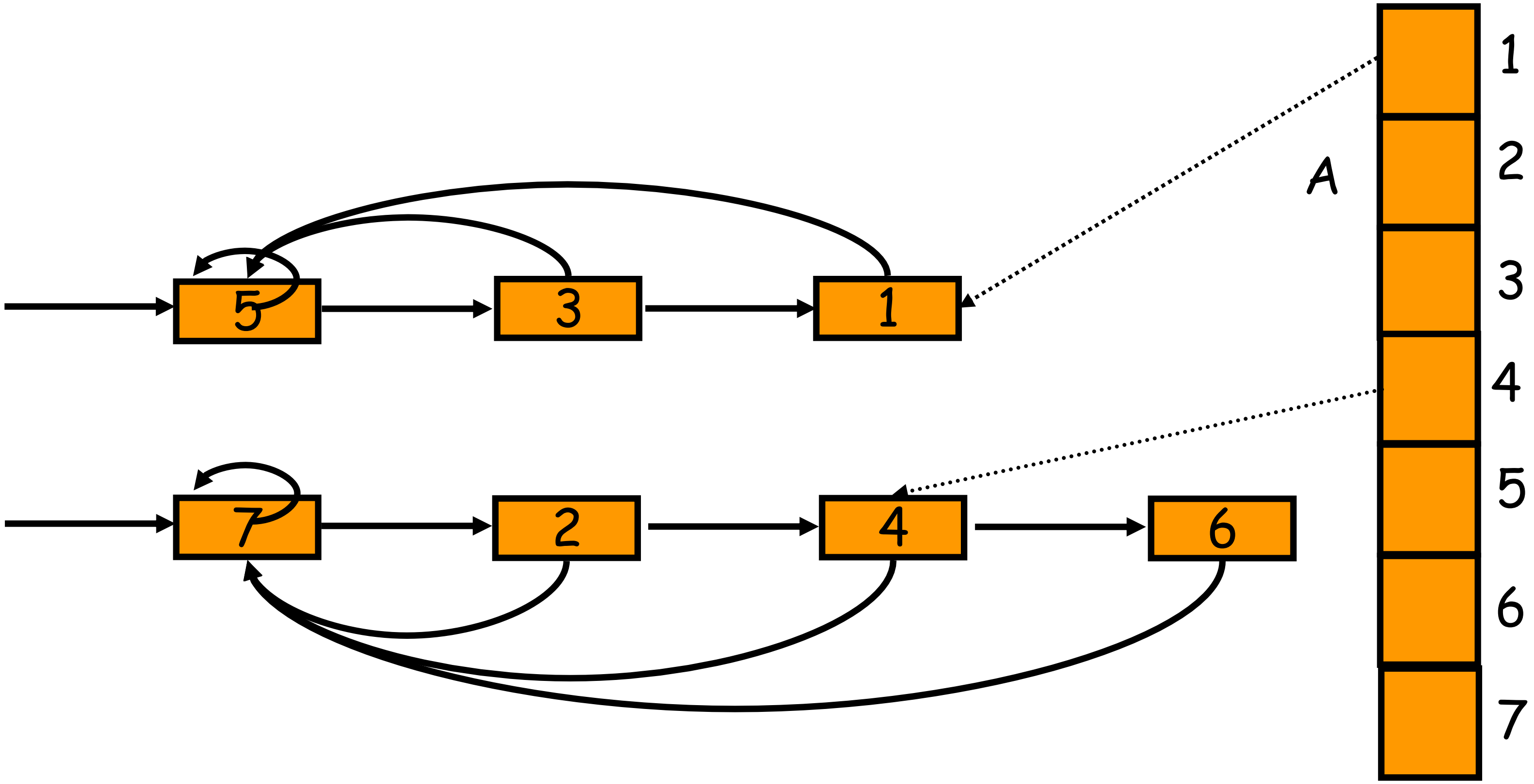
Find(i) -- ממומשת ע"י מעבר על המצביע A[i] והחזרת המצביע לראש הרשימה. זמן O(1).

Union(p,q) -- ממומשת ע"י אחוד הרשימות המוצבעות ע"י p,q לרשימה אחת וכן עדכון מצביעי האיברים לראש הרשימה החדשה. הפונקציה מחזירה את ראש הרשימה החדשה. זמן O(n).

©cs,Technion

דוגמא

דוגמא: לאחר Union(p,q) נקבל:



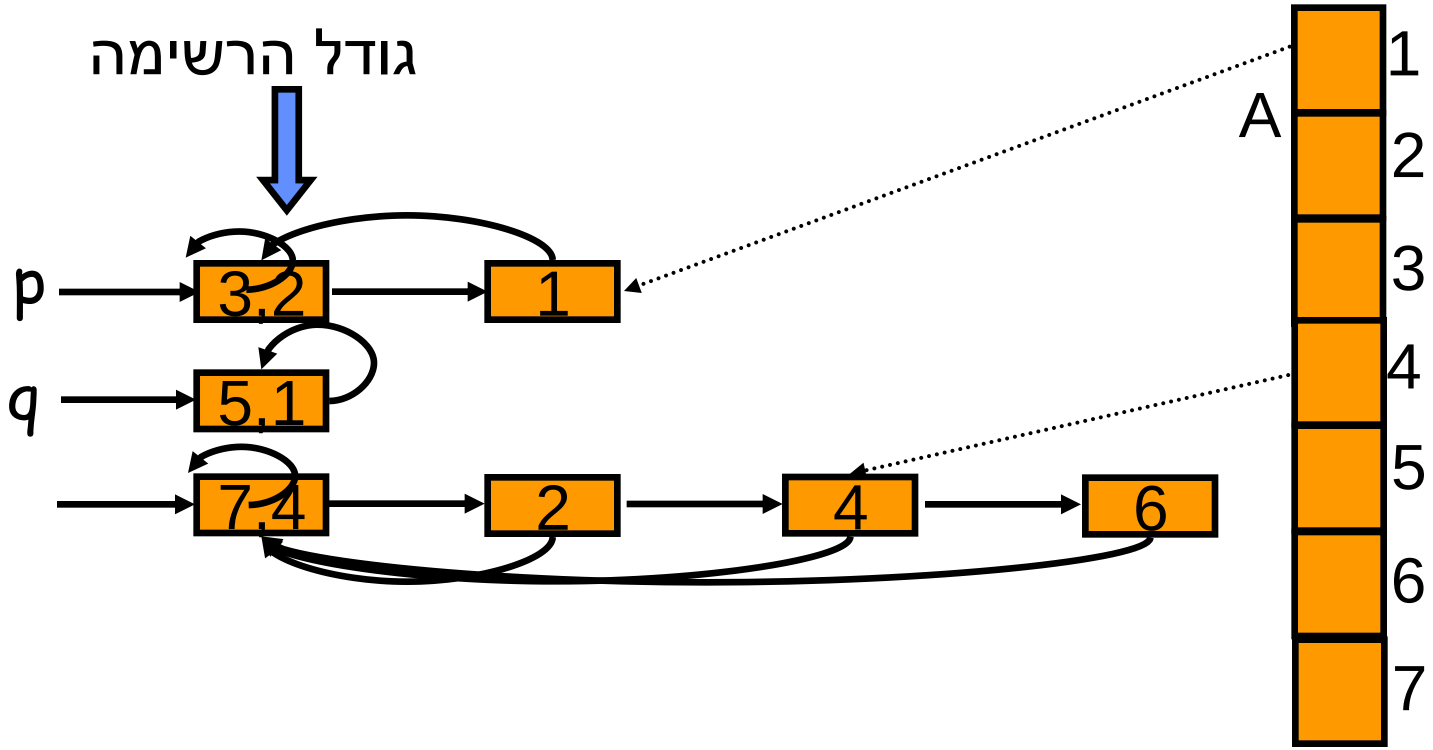
כאשר מאחדים שתי קבוצות יש לשנות באחת הקבוצות את כל המצביעים לכותרת החדשה. ברור שאת שינוי המצביעים עדיף לעשות בקבוצה הקטנה מבין השתיים (בניגוד לדרך שפעלנו בדוגמא זו).

©cs,Technion

פתרון שלישי משופר

שוב נייצג כל קבוצה כרשימה. בנוסף, בראש הרשימה נשמור את גודלה.

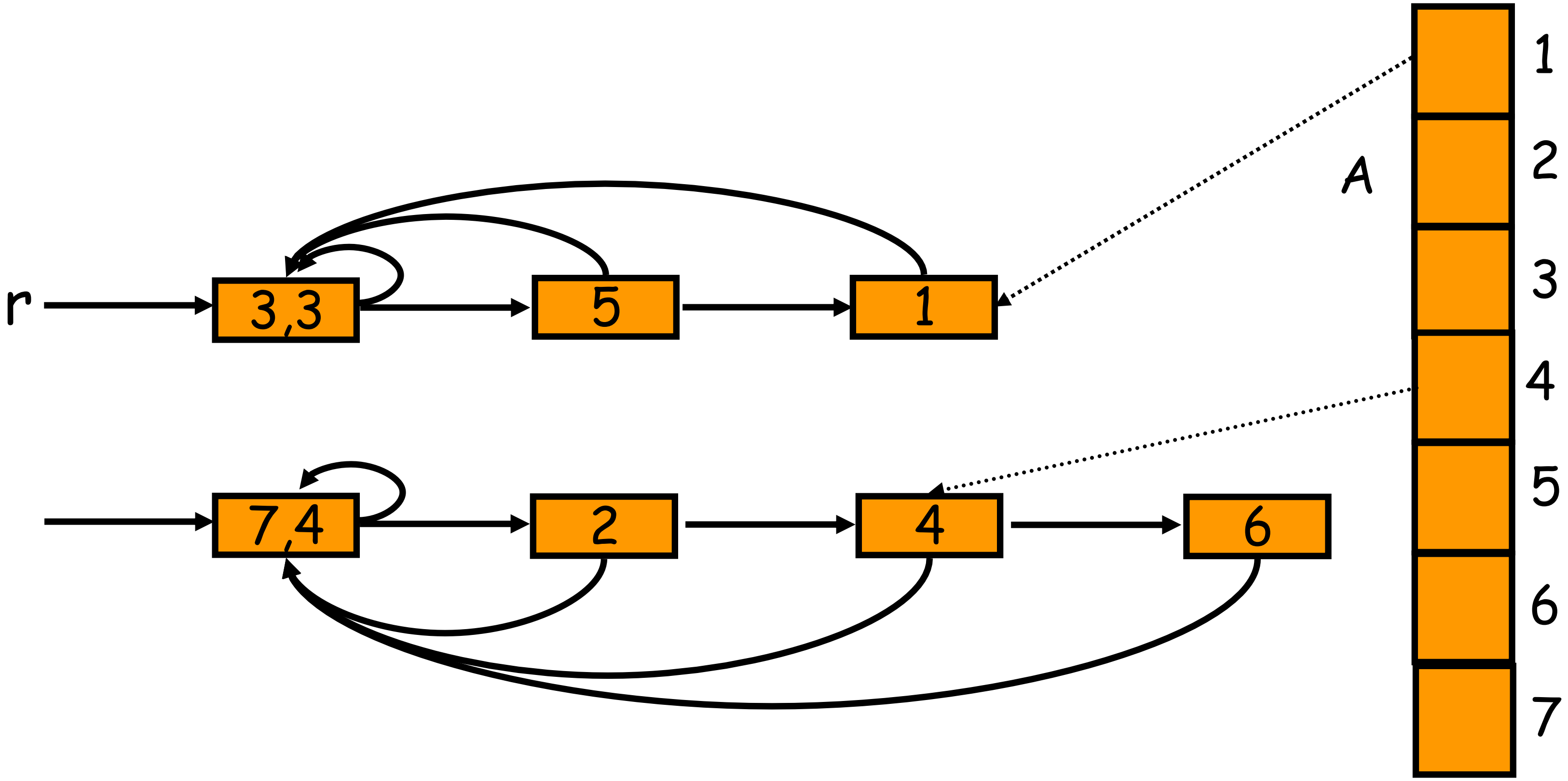
Find יתבצע כמו קודם ו- Union יתבצע ע"י הוספת הקבוצה הקטנה לתוך הקבוצה הגדולה. כלומר ראש הרשימה החדשה יהיה ראש הרשימה הגדולה יותר, וכל המצביעים יצביעו עליו.



כיצד נבצע r = Union(p,q) ?

דוגמא

לאחר r = Union(p,q) נקבל:



שימו לב שהקבוצה {5} שהייתה הקטנה מבין שתי הקבוצות הוספה לתוך הקבוצה הגדולה יותר {3,1} והקבוצה החדשה שנוצרה מוצבעת ע"י r.

©cs,Technion

ניתוח השיפור

<u>טענה א</u>: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי כל איבר x משנה את הקבוצה אליה הוא שייך לכל היותר log_2n פעמים כאשר n הוא מספר האיברים במבנה.

©cs,Technion

ניתוח השיפור

טענה א: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי כל איבר x משנה את הקבוצה אליה הוא שייך לכל היותר log_2n פעמים כאשר n הוא מספר האיברים במבנה.

הוכחה: בכל פעם שאיבר x משנה את הקבוצה אליה הוא שייך הוא עובר לקבוצה חדשה הגדולה לפחות פי 2 מקבוצתו העכשווית. לאחר לכל היותר log_2n מעברים הקבוצה הנוצרת מכילה את כל n האיברים ולפיכך שיוכו של איבר x לא ישתנה יותר.

ניתוח השיפור

טענה א: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי כל איבר x משנה את הקבוצה אליה הוא שייך לכל היותר log_2n פעמים כאשר n הוא מספר האיברים במבנה.

הוכחה: בכל פעם שאיבר x משנה את הקבוצה אליה הוא שייך הוא עובר לקבוצה חדשה הגדולה לפחות פי 2 מקבוצתו העכשווית. לאחר לכל היותר log_2n מעברים הקבוצה הנוצרת מכילה את כל n האיברים ולפיכך שיוכו של איבר x לא ישתנה יותר.

מסקנה: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי הזמן הכולל לביצוע סדרה כלשהי של m פעולות Union/Find/Makeset הוא לכל היותר $O(m \log n)$.

ניתוח השיפור

טענה א: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי כל איבר x משנה את הקבוצה אליה הוא שייך לכל היותר $log_2 n$ פעמים כאשר n הוא מספר האיברים במבנה.

הוכחה: בכל פעם שאיבר x משנה את הקבוצה אליה הוא שייך הוא עובר לקבוצה חדשה הגדולה לפחות פי 2 מקבוצתו העכשווית. לאחר לכל היותר $log_2 n$ מעברים הקבוצה הנוצרת מכילה את כל n האיברים ולפיכך שיוכו של איבר x לא ישתנה יותר.

מסקנה: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי הזמן הכולל לביצוע סדרה כלשהי של m פעולות Union/Find/Makeset הוא לכל היותר $O(m \log n)$.

הוכחה: n הוא מספר האיברים ב- U עבורם בוצע Makeset. לאור טענה א, סה"כ הזמן הנדרש לכל פעולות ה- Union הוא $O(n \log n)$ לכל היותר. מתקיים $n \le m$. שאר הפעולות הן פעולות Find/Makeset שזמן כל אחת מהן הוא $O(1)$. לפיכך סיבוכיות הזמן היא $O(m + n \log n)$. סיבוכיות זמן זו חסומה ע" $O(m \log n)$.

©cs,Technion

ניתוח השיפור

טענה א: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי כל איבר x משנה את הקבוצה אליה הוא שייך לכל היותר log_2n פעמים כאשר n הוא מספר האיברים במבנה.

הוכחה: בכל פעם שאיבר x משנה את הקבוצה אליה הוא שייך הוא עובר לקבוצה חדשה הגדולה לפחות פי 2 מקבוצתו העכשווית. לאחר לכל היותר log_2n מעברים הקבוצה הנוצרת מכילה את כל n האיברים ולפיכך שיוכו של איבר x לא ישתנה יותר.

מסקנה: אם בכל איחוד מוסיפים את הקבוצה הקטנה לגדולה אזי הזמן הכולל לביצוע סדרה כלשהי של m פעולות Union/Find/Makeset הוא לכל היותר $O(m \log n)$.

הוכחה: n הוא מספר האיברים ב- U עבורם בוצע Makeset. לאור טענה א, סה"כ הזמן הנדרש לכל פעולות ה- Union הוא $O(n \log n)$ לכל היותר. מתקיים $n \leq m$. שאר הפעולות הן פעולות Find/Makeset שזמן כל אחת מהן הוא $O(1)$. לפיכך סיבוכיות הזמן היא $O(m + n \log n)$. סיבוכיות זמן זו חסומה ע" $O(m \log n)$.

לפיכך כל פעולה לוקחת זמן משוערך (Amortized time) של $O(\log n)$.

זמן משוערך

הגדרה: אם m פעולות מתבצעות בתוך זמן M, אזי הזמן המשוערך לכל פעולה מוגדר להיות M/m.

נימוקים להגדרה: כאשר ישנה סדרה ארוכה של פעולות שרובן קלות לביצוע (כגון find במימוש האחרון) וחלקן קשה לביצוע (כגון union באותו מימוש), אז ההשפעה של הפעולות הקשות מתחלקת על סדרת הפעולות כולה. יתכן גם מצב שזמן בצוע הפעולה משתנה כתוצאה מ"עזרה" הניתנת בזמן פעולה קודמת ואז הזמן המשוערך הוא מדד המתחשב בתרומה של פעולה אחת לרעותה.

©cs,Technion

זמן משוערך

הגדרה: אם m פעולות מתבצעות בתוך זמן M, אזי הזמן המשוערך לכל פעולה מוגדר להיות M/m.

נימוקים להגדרה: כאשר ישנה סדרה ארוכה של פעולות שרובן קלות לביצוע (כגון find במימוש האחרון) וחלקן קשה לביצוע (כגון union באותו מימוש), אז ההשפעה של הפעולות הקשות מתחלקת על סדרת הפעולות כולה. יתכן גם מצב שזמן בצוע הפעולה משתנה כתוצאה מ"עזרה" הניתנת בזמן פעולה קודמת ואז הזמן המשוערך הוא מדד המתחשב בתרומה של פעולה אחת לרעותה.

להלן סוגי ממוצעים שראינו בקורס:

1. בשיעור זה הממוצע נלקח עלפני סדרת הפעולות שמשתמש מייצר.
2. באלגוריתם רנדומלי (למשל עבור רשימת דילוגים), הממוצע נלקח עלפני תוצאות הגרלה שהאלגוריתם מבצע.
3. בניתוח בניית עץ חיפוש בינרי אקראי, הממוצע נלקח עלפני פילוג הקלט.

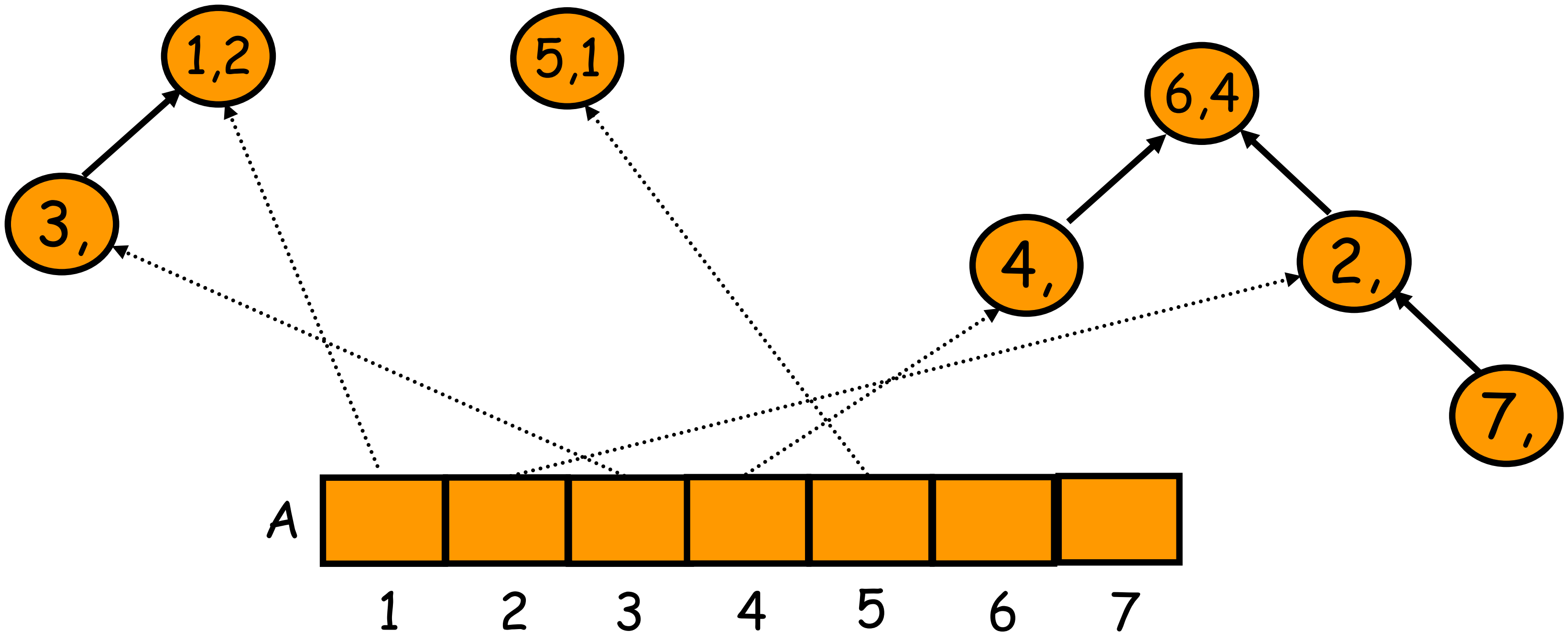
©cs,Technion

מימוש נוסף: עצים הפוכים (Up trees)

לכל קבוצה ניצור עץ הפוך (בנים מצביעים להורה) ובו צומת לכל איבר בקבוצה. שורש העץ יכיל גם את מספר איברי הקבוצה. בנוסף נחזיק מערך גישה לאיברים כמו במימוש השלישי.

בזמן פעולת Union תולים את שורש העץ הקטן יותר מתחת לשורש העץ הגדול יותר ומעדכנים את גודל הקבוצה המאוחדת.

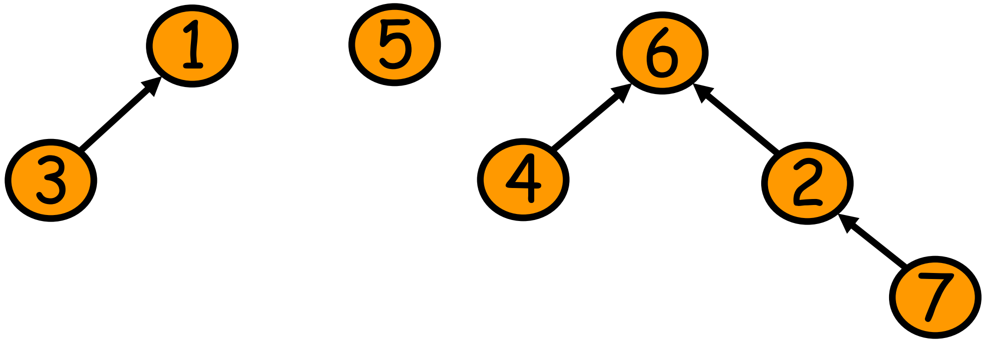
לדוגמא הקבוצות {2,4,6,7} {5} {1,3} מיוצגות ע"י:



מימוש עצים הפוכים בעזרת מערכים

בהנחה שמספר האיברים ידוע מראש, ניתן לייצג את כל הרשימות במערכים ללא שימוש במצביעים. ניתן לעשות זאת גם בכל המימושים הקודמים.

תאור גרפי:



מימוש:

size	2				1	4	
parent	--	6	1	6	--	--	2
elements	1	2	3	4	5	6	7

האיבר בשורש משמש מציין לקבוצה

Find(i) -- ממומש ע"י סיור במעלה העץ. זמן O(h) כאשר h הוא גובה העץ.

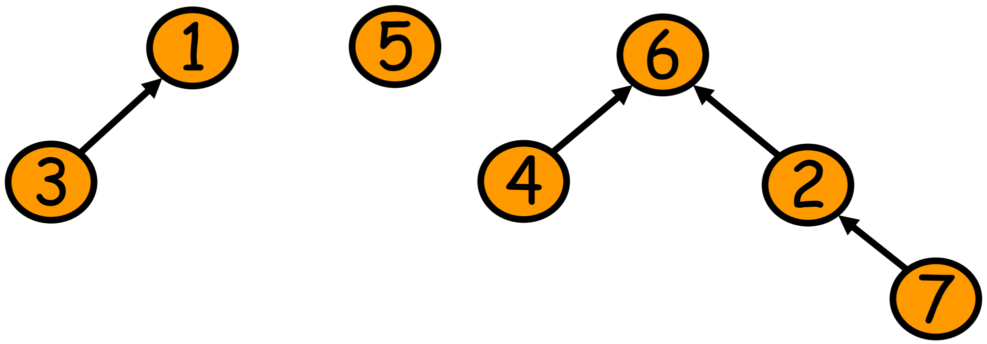
Union(p,q) -- ממומש ע"י אחוד העצים ששורשיהן p ו- q כך ששורש העץ הקטן יופנה לשורש העץ הגדול. זמן O(1).

©cs,Technion

דוגמא

מצב התחלתי (כמו בשקף הקודם):

size	2				1	4	
parent	--	6	1	6	--	--	2
elements	1	2	3	4	5	6	7

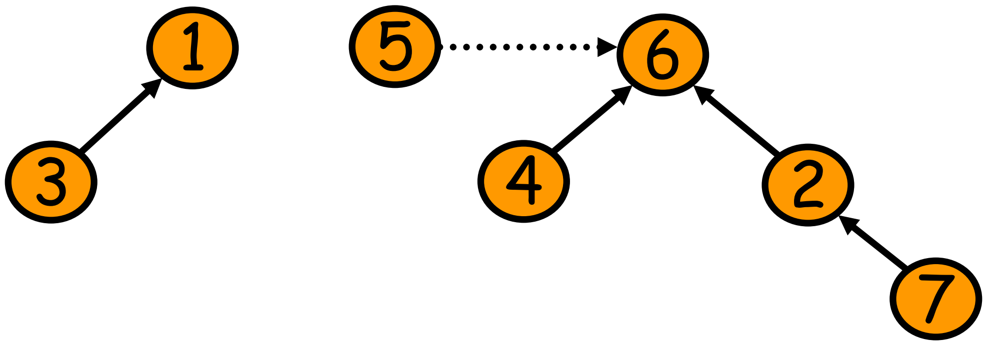


דוגמא

מצב התחלתי (כמו בשקף הקודם):

elements	1	2	3	4	5	6	7
size	2				1	4	
parent	--	6	1	6	--	--	2

לאחר Union(5,6):



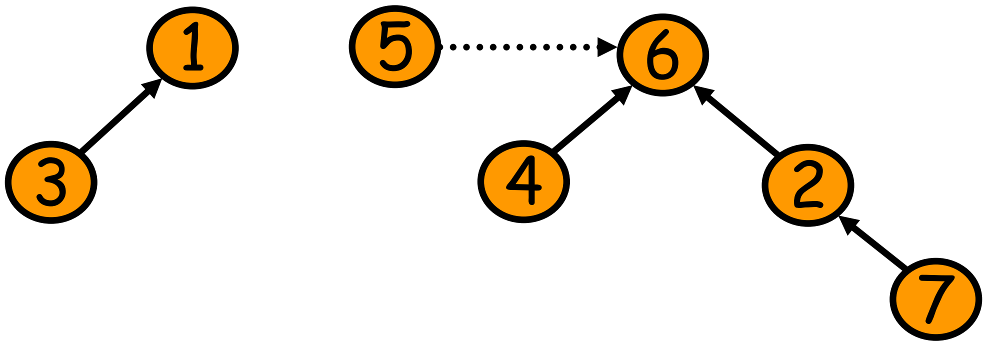
©cs,Technion

דוגמא

מצב התחלתי (כמו בשקף הקודם):

elements	1	2	3	4	5	6	7
size	2				1	4	
parent	--	6	1	6	--	--	2

לאחר Union(5,6):



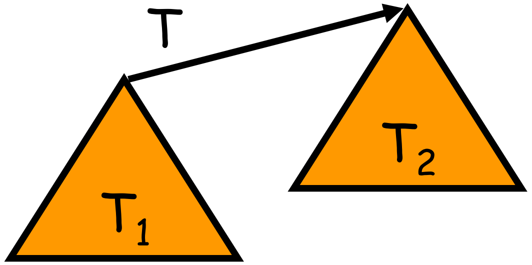
elements	1	2	3	4	5	6	7
size	2					5	
parent	--	6	1	6	6	--	2

זמן לפעולת Union הוא O(1).

©cs,Technion

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.

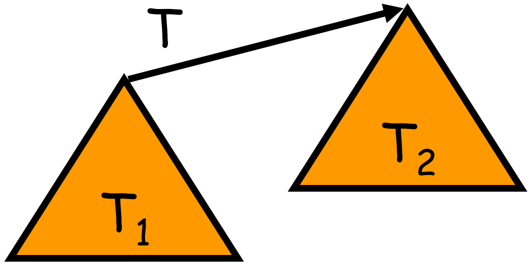


הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

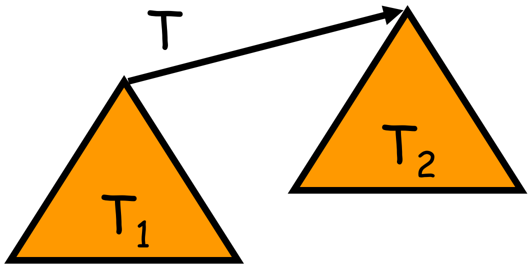
יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

- בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_1| \leq |T_2|$.

©cs,Technion

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

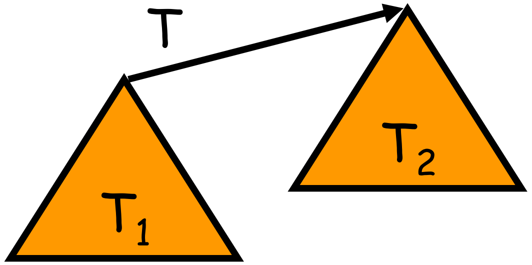
יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

- בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_1| \geq |T_2|$.
- נסמן ב- h_1 וב - h_2 את גובה העצים T_1 ו- T_2 בהתאמה.

©cs,Technion

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

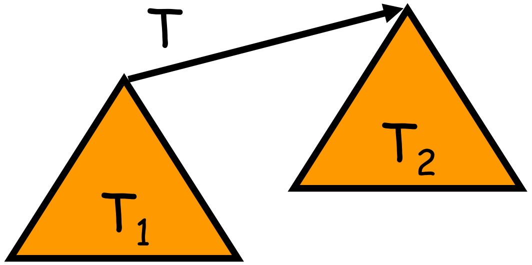
יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

- בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_1| \leq |T_2|$.
- נסמן ב- h_1 וב - h_2 את גובה העצים T_1 ו- T_2 בהתאמה.
- מקרה א: $h_1 < h_2$. סתירה שכן אז T הוא עץ בעל גובה זהה ל-T_2 ואינו בעל מספר צמתים מינימלי.

©cs,Technion

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

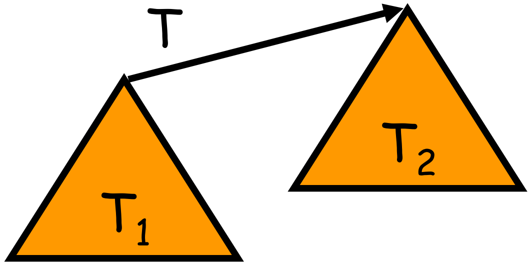
יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

- בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_2| \geq |T_1|$.
- נסמן ב- h_1 וב - h_2 את גובה העצים T_1 ו- T_2 בהתאמה.
- מקרה א: $h_1 < h_2$. סתירה שכן אז T הוא עץ בעל גובה זהה ל-T_2 ואינו בעל מספר צמתים מינימלי.
- מקרה ב: $h_1 \geq h_2$. במקרה זה מתקיים $h = h_1 + 1$ ולפיכך

©cs,Technion

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

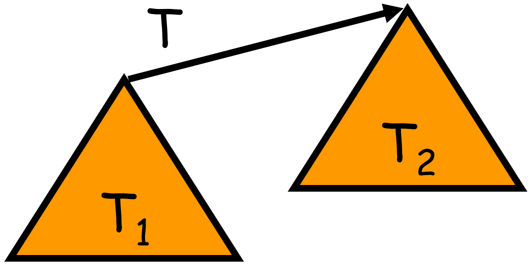
- בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_1| \geq |T_2|$.
- נסמן ב- h_1 וב - h_2 את גובה העצים T_1 ו- T_2 בהתאמה.
- מקרה א: $h_2 < h_1$. סתירה שכן אז T הוא עץ בעל גובה זהה ל-T_2 ואינו בעל מספר צמתים מינימלי.
- מקרה ב: $h_2 \geq h_1$. במקרה זה מתקיים $h = h_1 + 1$ ולפיכך

$|T| = |_T1| + |_T2| \geq 2 \cdot |_T1| \geq 2 \cdot {}^{2h}1 = {}^{2h}$ כנדרש.

©cs,Technion

ניתוח גובה העץ

טענה: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



הוכחה: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

- בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_1| \geq |T_2|$.
- נסמן ב- h_1 וב - h_2 את גובה העצים T_1 ו- T_2 בהתאמה.
- מקרה א: $h_2 < h_1$. סתירה שכן אז T הוא עץ בעל גובה זהה ל-T_2 ואינו בעל מספר צמתים מינימלי.
- מקרה ב: $h_2 \geq h_1$. במקרה זה מתקיים $h = h_1 + 1$ ולפיכך

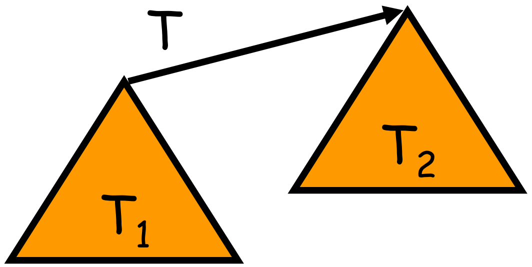
$$|T| = |_T1| + |_T2| \geq 2 \cdot |_T1| \geq 2 \cdot {}^{2h}1 = {}^{2h}$$ כנדרש.

מסקנה 1: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $h \leq \log_2|T|$.

©cs,Technion

ניתוח גובה העץ

<u>טענה</u>: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $|T| \geq 2^h$.



<u>הוכחה</u>: באינדוקציה על h. עבור $h = 0$ ישנם 2^0 צמתים.

יהי T עץ בגובה h בעל מספר צמתים מינימלי. נבחן את פעולת האיחוד האחרונה:

• בפעולת האיחוד האחרונה חובר עץ T_1 לתוך עץ T_2 ולפיכך התקיים $|T_1| \geq |T_2|$.

• נסמן ב- h_1 וב - h_2 את גובה העצים T_1 ו- T_2 בהתאמה.

• מקרה א: $h_2 > h_1$. סתירה שכן אז T הוא עץ בעל גובה זהה ל-T_2 ואינו בעל מספר צמתים מינימלי.

• מקרה ב: $h_2 \leq h_1$. במקרה זה מתקיים $h = h_1 + 1$ ולפיכך

$|T| = |_T1| + |_T2| \geq 2 \cdot |_T1| \geq 2 \cdot {}^{2h}1 = {}^{2h}$ כנדרש.

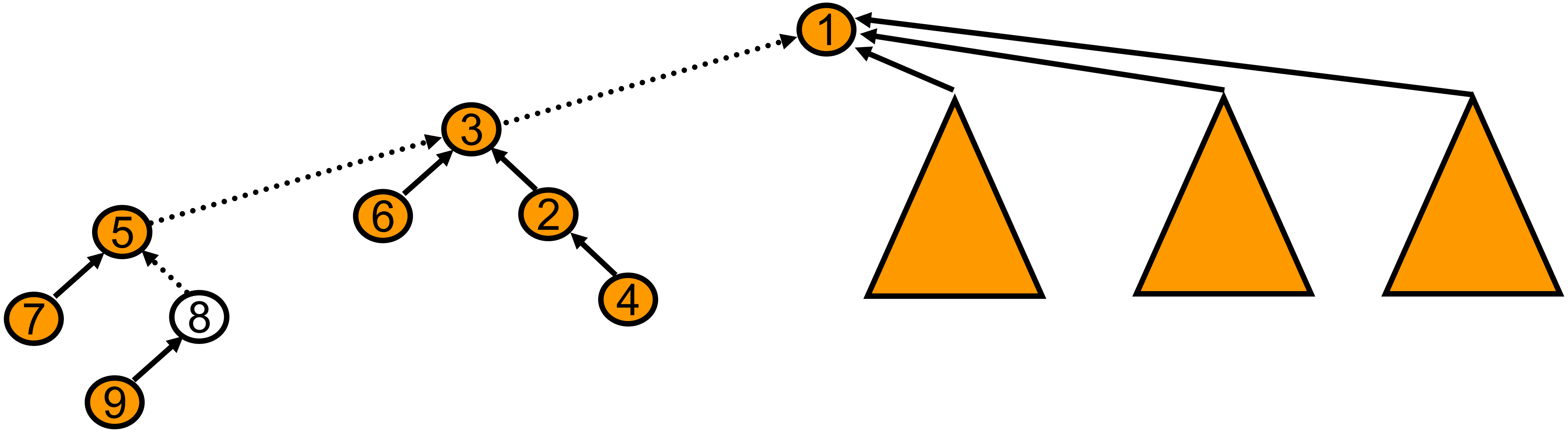
<u>מסקנה 1</u>: עבור עץ T בעל גובה h שנבנה מאיחודים של קבוצות קטנות לתוך גדולות, מתקיים $h \leq \log_2|T|$.

<u>מסקנה 2</u>: זמן פעולת Find הוא O(log n) במקרה הגרוע ביותר כאשר n הוא מספר הצמתים הכללי בכל העצים.

©cs,Technion

שיפור נוסף: כווץ מסלולים

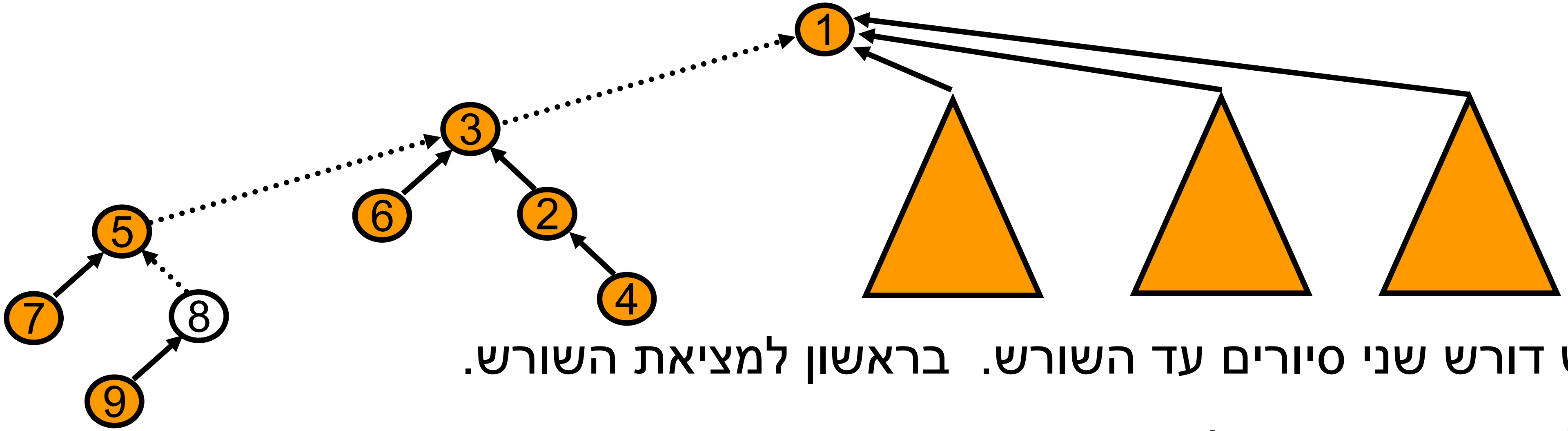
בזמן ביצוע Find(i), עדכן את שדה ה-parent של כל הצמתים מ-i ועד השורש כך שיצביעו ישירות לשורש. דוגמא: Find(8).



©cs,Technion

שיפור נוסף: כווץ מסלולים

בזמן ביצוע Find(i), עדכן את שדה ה-parent של כל הצמתים מ-i ועד השורש כך שיצביעו ישירות לשורש. דוגמא: Find(8).

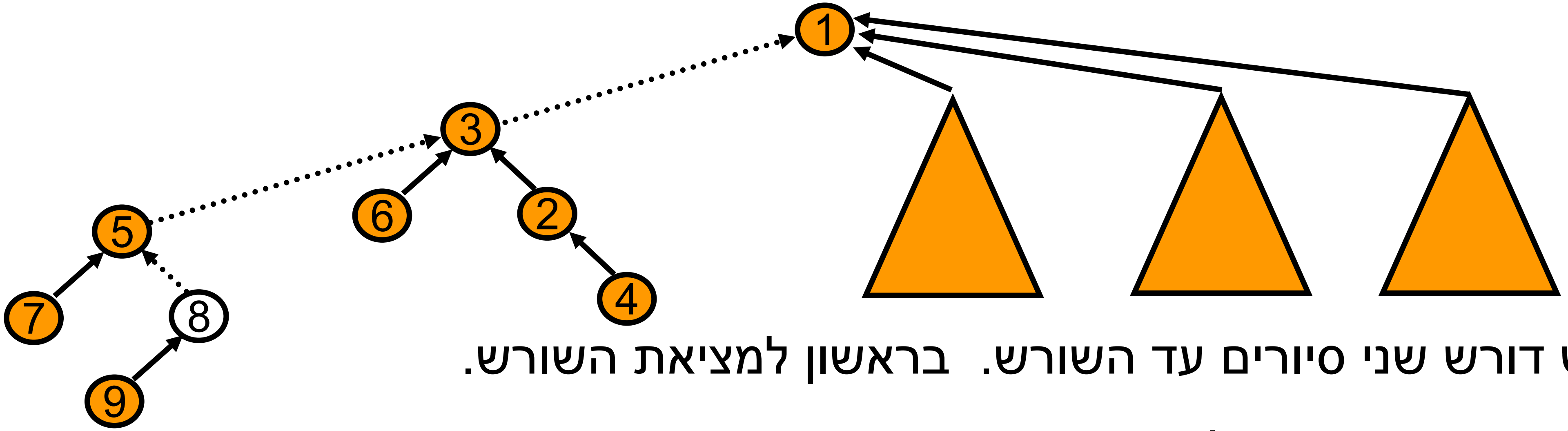


המימוש דורש שני סיורים עד השורש. בראשון למציאת השורש.

בשני לעדכון המצביעים לכוון השורש.

שיפור נוסף: כווץ מסלולים

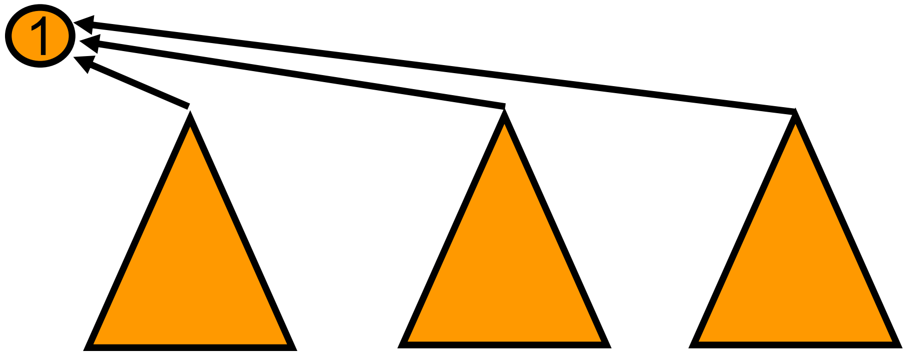
בזמן ביצוע Find(i), עדכן את שדה ה-parent של כל הצמתים מ-i ועד השורש כך שיצביעו ישירות לשורש. דוגמא: Find(8).



המימוש דורש שני סיורים עד השורש. בראשון למציאת השורש.

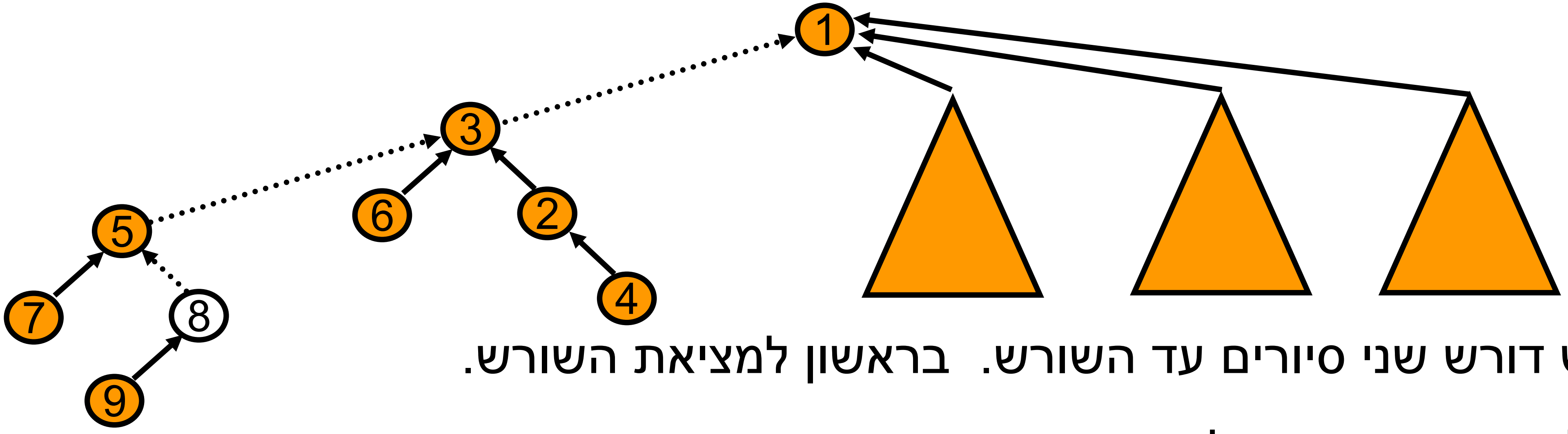
בשני לעדכון המצביעים לכוון השורש.

<u>תהליך העדכון</u>:



שיפור נוסף: כווץ מסלולים

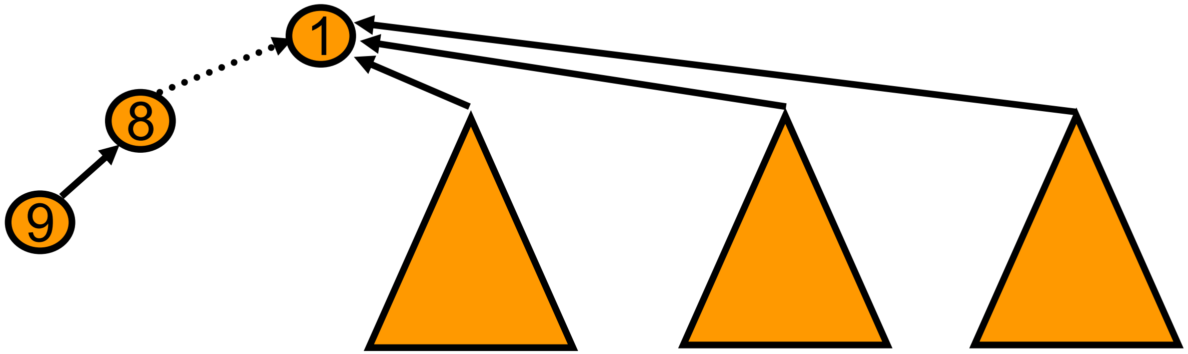
בזמן ביצוע Find(i), עדכן את שדה ה-parent של כל הצמתים מ-i ועד השורש כך שיצביעו ישירות לשורש. דוגמא: Find(8).



המימוש דורש שני סיורים עד השורש. בראשון למציאת השורש.

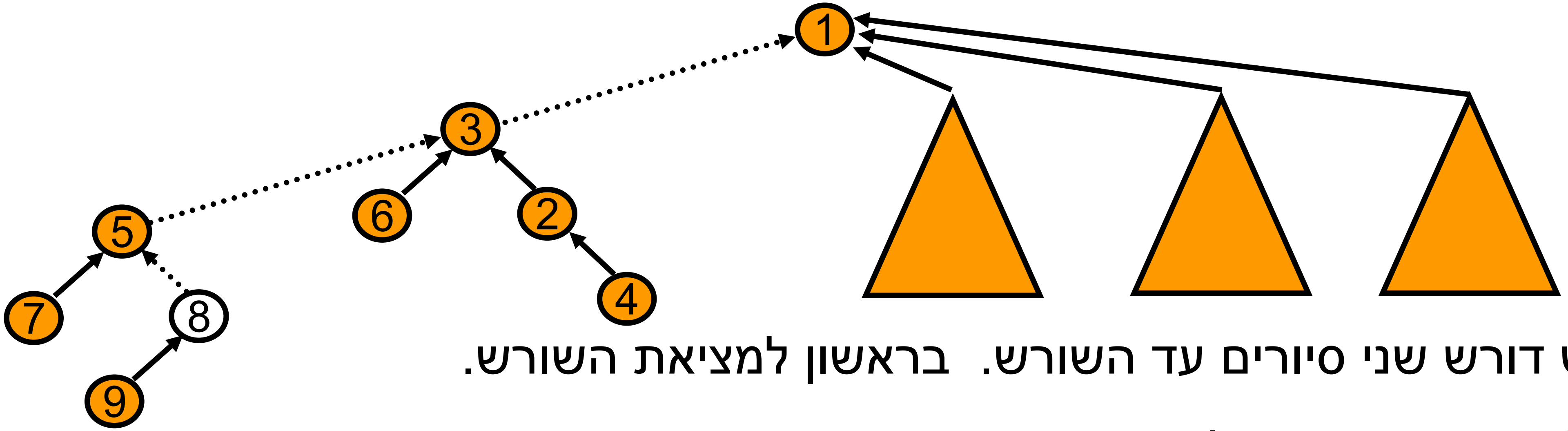
בשני לעדכון המצביעים לכוון השורש.

<u>תהליך העדכון</u>:



שיפור נוסף: כווץ מסלולים

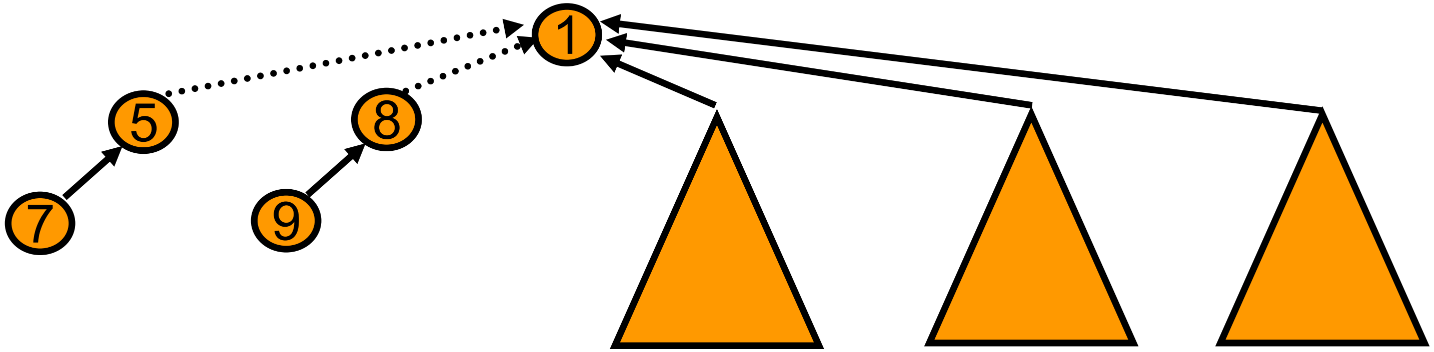
בזמן ביצוע Find(i), עדכן את שדה ה-parent של כל הצמתים מ-i ועד השורש כך שיצביעו ישירות לשורש. דוגמא: Find(8).



המימוש דורש שני סיורים עד השורש. בראשון למציאת השורש.

בשני לעדכון המצביעים לכוון השורש.

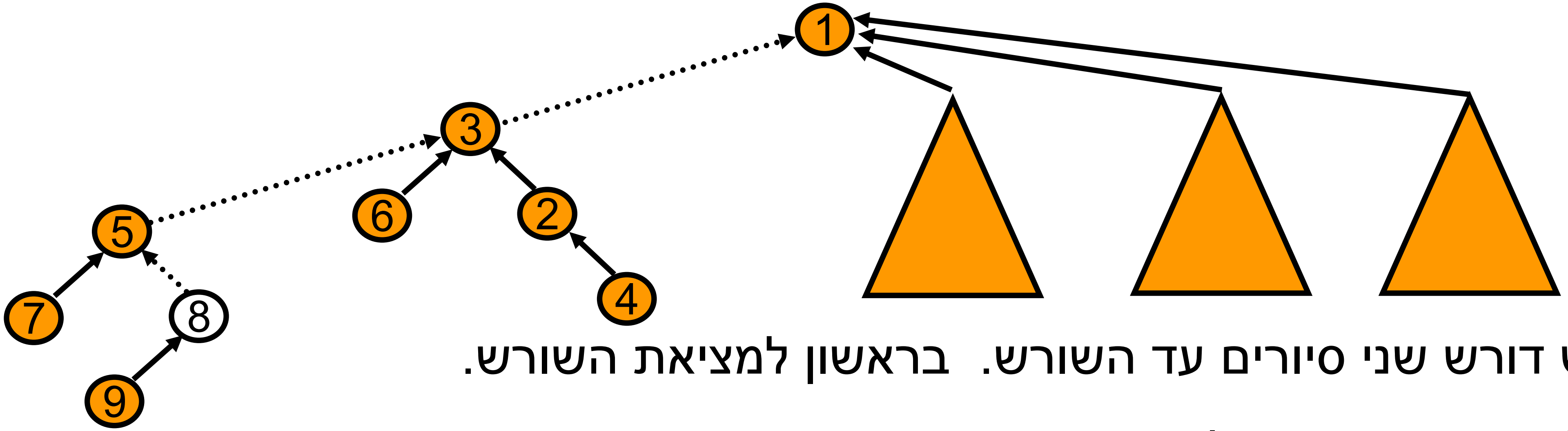
<u>תהליך העדכון</u>:



©cs,Technion

שיפור נוסף: כווץ מסלולים

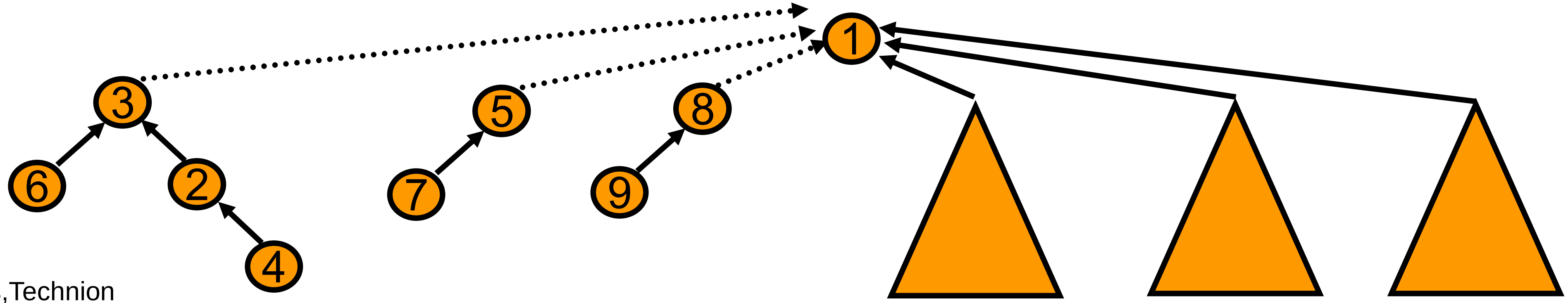
בזמן ביצוע Find(i), עדכן את שדה ה-parent של כל הצמתים מ-i ועד השורש כך שיצביעו ישירות לשורש. דוגמא: Find(8).



המימוש דורש שני סיורים עד השורש. בראשון למציאת השורש.

בשני לעדכון המצביעים לכוון השורש.

<u>תהליך העדכון</u>:



©cs,Technion

ניתוח זמנים

משפט: במימוש באמצעות אוסף עצים של n איברים, סיבוכיות זמן בצוע m פעולות union/find/makeset חסומה ע"י $O(m \log^* n)$.

לפיכך הזמן המשוערך (amortized time) לכל פעולה חסום ע"י $O(\log^* n)$,

כאשר הפונקציה $\log^* n$ מוגדרת כדלהלן:

$$\log^{(i)}(n) = \overbrace{\log_2 \log_2 \ldots \log_2}^{i} n$$

$$\log^*(n) = \min\{i \geq 0 \mid \log^{(i)}(n) \leq 1\}$$

©cs,Technion

ניתוח זמנים

משפט: במימוש באמצעות אוסף עצים של n איברים, סיבוכיות זמן בצוע m פעולות union/find/makeset חסומה ע"י $O(m \log^* n)$.

לפיכך הזמן המשוערך (amortized time) לכל פעולה חסום ע"י $O(\log^* n)$, כאשר הפונקציה $\log^* n$ מוגדרת כדלהלן:

$$\log^{(i)}(n) = \underbrace{\log_2 \log_2 \ldots \log_2}_{i} n$$

$$\log^*(n) = \min\{i \geq 0 \mid \log^{(i)}(n) \leq 1\}$$

ובמילים, $\log^* n$ הוא מספר הפעמים שצריך לקחת log כדי לקבל מספר קטן או שווה לאחד. זוהי פונקציה מונוטונית העולה בקצב מאד איטי. לכל מספר n פרקטי $\log^* n \leq 5$. כלומר פעולת find לוקחת זמן משוערך שהוא קבוע מבחינה מעשית.

ניתוח זמנים

משפט: במימוש באמצעות אוסף עצים של n איברים, סיבוכיות זמן בצוע m פעולות union/find/makeset חסומה ע"י $O(m \log^* n)$.

לפיכך הזמן המשוערך (amortized time) לכל פעולה חסום ע"י $O(\log^* n)$, כאשר הפונקציה $\log^* n$ מוגדרת כדלהלן:

$$\log^{(i)}(n) = \overbrace{\log_2 \log_2 \ldots \log_2}^{i} n$$

$$\log^*(n) = \min\{i \ge 0 \mid \log^{(i)}(n) \le 1\}$$

ובמילים, $\log^* n$ הוא מספר הפעמים שצריך לקחת log כדי לקבל מספר קטן או שווה לאחד. זוהי פונקציה מונוטונית העולה בקצב מאד איטי. לכל מספר n פרקטי $\log^* n \le 5$. כלומר פעולת find לוקחת זמן משוערך שהוא קבוע מבחינה מעשית.

$$\log^*(1) = 0 \quad \log^*(2) = 1 \quad \log^*(2^2) = 2 \quad \log^*(2^{2^2}) = 3 \quad \log^*(2^{2^{2^2}}) = 4$$

©cs,Technion

ניתוח זמנים

משפט: במימוש באמצעות אוסף עצים של n איברים, סיבוכיות זמן בצוע m פעולות union/find/makeset חסומה ע"י $O(m \log^* n)$.

לפיכך הזמן המשוערך (amortized time) לכל פעולה חסום ע"י $O(\log^* n)$, כאשר הפונקציה $\log^* n$ מוגדרת כדלהלן:

$$\log^{(i)}(n) = \overbrace{\log_2 \log_2 \ldots \log_2}^{i} n$$

$$\log^*(n) = \min\{i \geq 0 \mid \log^{(i)}(n) \leq 1\}$$

ובמילים, $\log^* n$ הוא מספר הפעמים שצריך לקחת log כדי לקבל מספר קטן או שווה לאחד. זוהי פונקציה מונוטונית העולה בקצב מאד איטי. לכל מספר n פרקטי $\log^* n \leq 5$. כלומר פעולת find לוקחת זמן משוערך שהוא קבוע מבחינה מעשית.

$$\log^*(1) = 0 \quad \log^*(2) = 1 \quad \log^*(2^2) = 2 \quad \log^*(2^{2^2}) = 3 \quad \log^*(2^{2^{2^2}}) = 4$$

$$\log^*(2^{2^{2^{2^2}}}) = \log^*(2^{65{,}536}) = 5$$

©cs,Technion

ניתוח זמנים

משפט: במימוש באמצעות אוסף עצים של n איברים, סיבוכיות זמן בצוע m פעולות union/find/makeset חסומה ע"י $O(m \log^* n)$.

לפיכך הזמן המשוערך (amortized time) לכל פעולה חסום ע"י $O(\log^* n)$, כאשר הפונקציה log*n מוגדרת כדלהלן:

$$\log^{(i)}(n) = \overbrace{\log_2 \log_2 \ldots \log_2}^{i} n$$

$$\log^*(n) = \min\{i \ge 0 \mid \log^{(i)}(n) \le 1\}$$

ובמילים, log*n הוא מספר הפעמים שצריך לקחת log כדי לקבל מספר קטן או שווה לאחד. זוהי פונקציה מונוטונית העולה בקצב מאד איטי. לכל מספר n פרקטי $\log^* n \le 5$. כלומר פעולת find לוקחת זמן משוערך שהוא קבוע מבחינה מעשית.

$$\log^*(1) = 0 \quad \log^*(2) = 1 \quad \log^*(2^2) = 2 \quad \log^*(2^{2^2}) = 3 \quad \log^*(2^{2^{2^2}}) = 4$$

$$\log^*(2^{2^{2^{2^2}}}) = \log^*(2^{65{,}536}) = 5$$

הוכחת המשפט ותאור חסם הדוק עוד יותר באמצעות פונקציות אקרמן ניתן למצוא בספר הלימוד בעמודים 450-453.

©cs,Technion

©cs,Technion