

מבנה נתונים

עפ"י חלומה

10 בספטמבר 2009

תוכן עניינים

5	1. הרצאה מס.1	1
7	2. הרצאה מס.2	2
7	2.1 מיון בועות	2.1
8	2.1.1 השיטה האיטרטיבית	2.1.1
8	2.2 שיטת הניחוש וההוכחה באינדוקציה	2.2
9	2.3 מיון מיזוג - Merge Sort	2.3
10	2.4 סיבוכיות של אלגוריתם אלפרדו משהוא	2.4
13	3. תרגול מס.2	3
13	3.1 שיטות לפתור ריקורסיה	3.1
13	3.1.1 שיטת עץ הריקורסיה	3.1.1
15	4. הרצאה מס.3	4
15	4.1 אלגוריתם Quicksort	4.1
16	4.1.1 סיבוכיות ממוצעת	4.1.1
17	5. הרצאה מס.4	5
19	6. הרצאה מס.?	6
19	6.1 תור קדימיות	6.1
19	6.1.1 מימוש בעזרת ערימה	6.1.1
21	6.1.2 איך מייצרים ערימה?	6.1.2
23	7. הרצאה מס.?	7
24	7.1 זמן ריצה של Max_Heapify	7.1
24	7.2 עץ חיפוש בינארי	7.2
27	8. הרצאה מס.?	8
28	8.1 רכיבי קשירות בגרף לא מכוון	8.1
29	8.2 גרף מכוון	8.2
29	8.2.1 גרף מכוון ללא מעגלים DAG: Directed Acyclic Graph	8.2.1
29	8.2.2 רכיבי קשירות בגרף מכוון	8.2.2

פרק 1

הרצאה מס.1

אתר: www.cs.huji.ac.il/~dast

פרק 2

הרצאה מס.2

סיקום של הרצאה קודמת: o, Θ, Ω, O

1. קבועים בפנים לא משנים

2. $f(n) = \Omega(g(n) + h(n)) = \Omega(g(n))$ כלומר $h(n)$ זניח יחסית ל $g(n)$ אז אפשר להשמיט את $h(n)$

3. $f(n) = n^d, g(n) = c^n$ אזי $g(n)$ מנצח את $f(n)$. $f(n) = O(g(n))$

4. $f(n) = \log^c(n), g(n) = n^d$ אזי $g(n)$ מנצח את $f(n)$.

2.1 מיון בועות

Bubble Sort	לחץ עיפוטם 1.2
<pre>Bubble Sort A[1...n] { if n>1 { bubble(A[1...n]) bubblesort(A[1,...,n-1]) } } Bubble(A[1...n]) { for j=1 to n-1 { if A(j)>A(j+1) then swap } }</pre>	

סיבוכיות של Bubble היא .

$$\begin{aligned}T(n) &= T(n-1) + \Theta(n) \\T(1) &= \Theta(1)\end{aligned}$$

¹כאשר אומרים \log מתכוונים ל \log_2 שהוא שווה ל \log_{10} עד כדי קבוע

נרצה לפתור את הבעיה הזו ולנצל את זה לראות כל מיני שיטות לפתור בעיות ריקורסיה.

2.1.1 השיטה האיטרטיבית

זה פשוט "לא יודעים אז בואו ננסה"

$$\begin{aligned}
 T(n) &\leq T(n-1) + c \cdot n \\
 &\leq T(n-2) + c(n-1) + cn \\
 &\leq T(n-3) + c(n-2) + c(n-1) + cn \\
 &\leq T(1) + 2c + 3c + \dots + c(n-1) + cn \\
 &\leq T(1) + \sum_{i=2}^n c \cdot i \\
 &= T(1) + c \left(\frac{n(n-1)}{2} - 1 \right) \\
 &= O(n^2)
 \end{aligned}$$

מאותו שיקול

$$\begin{aligned}
 T(n) &\geq T(n-1) + c'n \\
 &\vdots \\
 &\geq \Theta(n^2)
 \end{aligned}$$

מסיקים כי $T(n) = \Omega(n^2)$ ולכן $T(n) = \Theta(n^2)$ וגם $O(n^2)$

2.2 שיטת הניחוש וההוכחה באינדוקציה

ננסה להוכיח באינדוקציה $T(n) \leq cn^2$

$$\begin{aligned}
 T(n) &= T(n-1) + \Theta(n), \Theta(n) \leq cn \\
 T(1) &= \Theta(1) \leq c
 \end{aligned}$$

עבור $n=1$: $T(1) \leq c$ בסדר.

נניח עבור n ש $T(n) \leq cn^2$

נוכיח עבור $n+1$ כי $T(n+1) = T(n) + c(n+1)$

$$\begin{aligned}
 T(n+1) &= T(n) + c(n+1) \\
 &\stackrel{\text{מהנחת האינדוקציה}}{\leq} cn^2 + c(n+1) \\
 &= c(n^2 + n + 1) \\
 &\leq c(n+1)^2
 \end{aligned}$$

זה מראה כי $T(n) = O(n^2)$
 באותה צורה אפשר להוכיח הצד השני של האינדוקציה

2.3 מיון מיזוג - Merge Sort

Merge Sort

לחזעיפטם 2.2

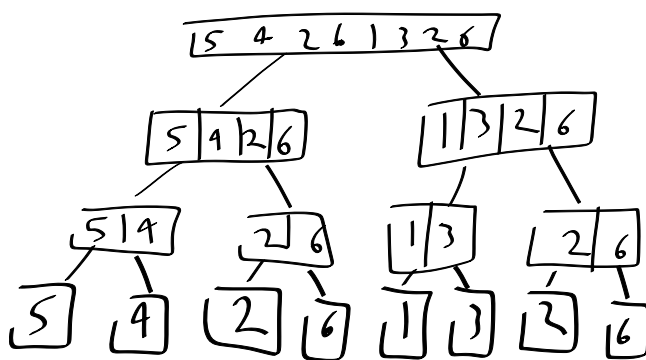
```

Merge Sort(A[1...n]) {
  if n > 1 {
    mergesort(A[1...n/2])
    mergesort(A[n/2 + 1...n])
    merge(A[1...n/2], A[n/2 + 1...n], A[1...n])
  }
}

Merge(A, B, C) {
  l = 1, i = 1, j = 1
  do while i < n/2 or j < n/2 {
    c[l] = min(A[i], B[j])
    if min was A[i] increment i, else increment j
    increment l
  }
}

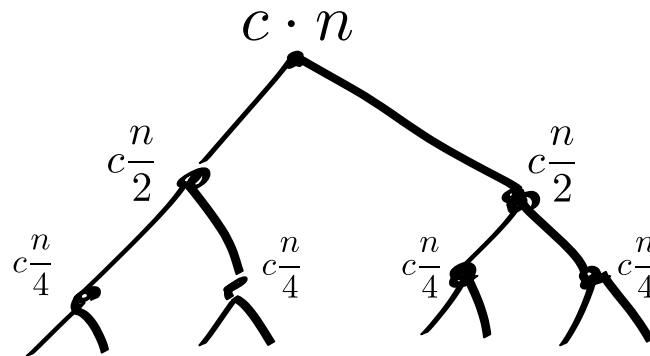
```

עץ הרקורסיה:



איור 2.1: עץ הרקורסיה של מיון מיזוג

נבנה עכשיו את עץ הרקורסיה ונכתוב כמה פעולות קוראות בכל קודקוד.

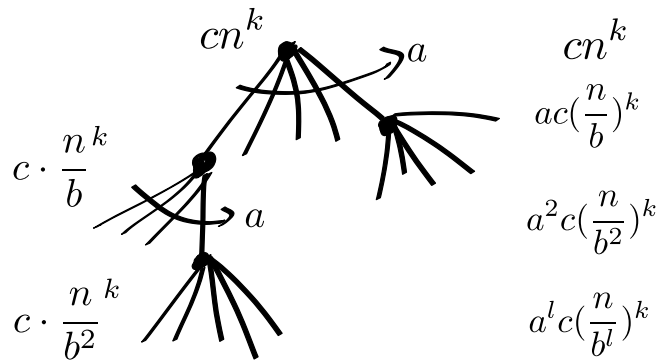


איור 2.2: משקל כל קודקוד במיזוג מיון

ניתן לראות כי בכל שורה בעץ יש cn פעולות. אזי האלגוריתם הזה עולה $cn \cdot \log_2(n)$ (גובה העץ)

2.4 סיבוכיות של אלגוריתם אלפרדו משהוא

$$T(n) = aT(n/b) + \Theta(n^k)$$



איור 2.3: עץ קריאה של אלגוריתם אלפרדו

$m = \log_b n$ אזי מספר השורות הוא $m + 1$

העבודה בשורה העליונה היא $\Theta(n^k)$

העבודה השורה התחתונה היא $\Theta(a^{\log_b n})$

אזי

$$\begin{aligned}
T(n) &\leq cn^k \sum_{l=0}^m \left(\underbrace{\frac{a}{b^k}}_k \right)^l \\
&\leq cn^k \left(\frac{q^{m+1} - 1}{q - 1} \right) = cn^k \left(\frac{1 - q^{m+1}}{1 - q} \right) = cn^k \left(\frac{1 - q^{m+1}}{1 - q} \right) \\
&\underbrace{\leq}_{q < 1} cn^k \frac{1}{q - 1} = O(n^k) \\
T &= O(n^k)
\end{aligned}$$

מה אם $q = 1$ כמו ב merge sort?

$$\begin{aligned}
T(n) &\leq cn^k (m + 1) \\
&= cn^k (\log_b (n + 1)) \\
&= \Theta(n^k \log n)
\end{aligned}$$

מה אם $q > 1$?

$$\begin{aligned}
T(n) &\leq cn^k \frac{q^{m+1} - 1}{q - 1} \\
&= \Theta(n^k q^m) \\
&= \Theta\left(n^k \left(\frac{a}{b^k}\right)^m\right) \\
&= \Theta\left(n^k \frac{a^m}{a^{mk}}\right) \\
&\underbrace{=}_{b^m = n} \Theta(a^m) \\
&= \Theta(a^{\log_b n}) \\
&\underbrace{=}_{\text{תרגיל}} \Theta(n^{\log_b a})
\end{aligned}$$

פרק 3

תרגול מס.2

מעריכים אלגוריתם לפי זמן ריצה וכמה מקום הוא לוקח בזכרון, מתארים את זה דרך פונקציות $\mathcal{O}, \theta, \Omega$

3.1 שיטות לפתור ריקורסיה

1. שיטה איטרטיבית

2. לנחש ולהוכיח באינדוקציה

3. עץ ריקורסיה

4. משפט המאסטר - Master Theorem

5. לחסום מלמעלה ומלמטה על ידי פומקציה אחרת

3.1.1 שיטת עץ הריקורסיה

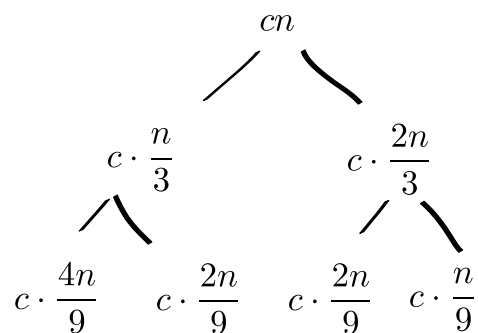
משתמשים בזה לנחש את הפתרון

אחרי שמנחשים צריך להוכיח

למשל, תמצא חסם עליון על $T(1) = 6, T(2) = 7, T(3) = 10$

$$\begin{aligned}T(1) &= 6, T(2) = 7, T(3) = 10 \\T(n) &= T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n)\end{aligned}$$

את $\Theta(n)$ נכתוב בצורת cn



איור 3.1: דוגמה לעץ ריצה

אפשר לראות כי בכל שורה יש cn פעולות. אבל זה לא בדיוק נכון כי הענף הימני ארוך יותר מהענף השמאלי.

הגובה המקסימאלי של עץ זה הוא $h = \log_{3/2}(n)$

אזי נרצה להוכיח כי $cn \log_{3/2} n = \mathcal{O}(n \log n)$

פרק 4

הרצאה מס.3

4.1 אלגוריתם Quicksort

Quicksort	לחזעיפטם 1.4
<hr/>	
Quicksort[<i>Left</i> , <i>Right</i>]	
-if(<i>right</i> > <i>left</i>)	
— $m \leftarrow \text{partition}[A, \text{Left}, \text{Right}]$	
—Quicksort $A[\text{Left}, m - 1]$	
—Quicksort $A[m + 1, \text{Right}]$	
Partition $A[\text{Left}, \text{Right}]$	
-If $\text{Left} < \text{Right}$	
— $L = \text{Left}, R = \text{Right}, \text{pivot} = A(\text{Right})$	
—for $j = \text{Left}$ to Right	
—if $A(j) < \text{pivot}$	
— $B(L) = A(j), L++$	
—else	
— $B(R) = A(j), R--$	
— $B(L) = A[\text{Right}]$	
— $A[\text{Left}, \text{Right}] = B[\text{Left}, \text{Right}]$	
—return L	

יש גרסה יותר יעילה לפונקציה *Partition*:

Partition In Place	לחזעיפטם 2.4
<hr/>	
Partition In Place $A[\text{Left}, \text{Right}]$	
— $\text{pivot} = A[\text{Right}]$	
—scan A from Left to find $A(j) > \text{pivot}$	
—scan B from Right to find $A(j) < \text{pivot}$	
—Swap	
—continue until pointers cross	
—swap pivot with that location(where pointers cross)	
—return pointer	

אז רוצים לנתח את *Quicksort*:

$$T(n) = T(m-1) + T(n-m) + \Theta(n)$$

אבל את כל זה תלוי ב m שאי אפשר לבטא אותו באופן חד חד ערכי. אז נסתכל על כמה מקרי קיצון:

1. אם $m = n - 1$ תמיד מקבלים $T(n) = T(n-2) + T(1) + \theta(n)$ ועומק הריקורסיה הוא n אזי $T(n) = \Theta(n^2)$

2. אם $m = \frac{n}{2}$ תמיד מקבלים $T(n) = T(\frac{n}{2} - 1) + T(\frac{n}{2}) + \Theta(n)$ כמו Mergesort $\Theta(n \log n)$

3. נניח $m = \frac{9}{10}n$ אזי מקבלים $T(n) = T(\frac{9}{10}n) + T(\frac{n}{10}) + \Theta(n)$ העומק המקסימאלי הוא $\log_{10/9} n = \frac{\log_2 n}{\log_2 \frac{10}{9}}$ אבל הרבה ענפים נפסקים באמצע אזי מקבלים כי זה $\mathcal{O}(n \log n)$ ¹

טענה: לכל קלט $T(n) = \mathcal{O}(n^2)$ וכן $T(n) = \Omega(n \log n)$ הקלט האכי גרוע הוא מערך ממיון וסיבוכיות של Worst Case של Quicksort הוא $\mathcal{O}(n^2)$

4.1.1 סיבוכיות ממוצעת

$$\langle T(n) \rangle = \sum_{\delta} \frac{(T(n) \text{ for input } \delta)}{n!}$$

δ הוא מיון מסויים של המספרים $\{k_1, \dots, k_n\}$ מקבלים כי $\langle T(n) \rangle = \Theta(n \log n)$ אבל את זה לא נוכיח דרך אחרת לחשב התוחלת היא להגריל את m שזה די נכון לעשות ואז נחשב את התוחלת $E(T(n))$

$$E(T(n)) = \sum_{m=1}^n \frac{1}{n} \left(E(T(m-1)) + E(T(n-m)) \right) + \Theta(n)$$

¹שים לב, זה \mathcal{O} ולא Θ

פרק 5

הרצאה מס. 4

התוכלת של Randomized Quicksort היא

$$\begin{aligned} E(T(n)) &= \sum_{m=1}^n \frac{1}{n} \left(E(T(m-1)) + E(T(n-m)) \right) + \Theta(n) \\ &= \sum_{m=0}^{n-1} \frac{1}{n} \cdot 2 \cdot E(T(m)) + \Theta(n) \end{aligned}$$

בקורס זה תמיד $T(1) = \times(1)$ אזי

$$\sum_{m=0}^{n-1} \frac{2}{n} E(T(m)) + c_2 n \leq E(T(n)) \leq \sum_{m=0}^{n-1} \frac{2}{n} E(T(m)) + c_1 n$$
$$c_2 \leq E(T(1)) \leq c_1$$

אז נגדיר

$$\begin{aligned} U_C(n) &= \sum_{m=0}^{n-1} \frac{2}{n} U_c(m) + cn \\ U_c(1) &= U_c(0) = c \end{aligned}$$

טענה:

$$U_{c_2} \leq E(T(n)) \leq U_{c_1}(n)$$

ניתן להוכיח באינדוקציה (ההוכחה בתרגיל)

=

=

$$- \begin{cases} nU_c(n) &= 2 \sum_{m=0}^{n-1} U_c(m) + cn^2 \\ (n+1)U_c(n+1) &= 2 \sum_{m=0}^n U_c(m) + c(n+1)^2 \end{cases}$$

$$\begin{aligned} (n+1)U_c(n+1) - nU_c(n) &= 2U_c(n) + c(n+1)^2 - cn^2 \\ (n+1)U_c(n+1) &= (n+2)U_c(n) + 2cn + c \\ U_c(n+1) &= \frac{n+2}{n+1}U_c(n) + \frac{2cn+c}{n+1} \end{aligned}$$

טענה קלה:

$$c \leq \frac{2cn+c}{n+1} \leq 2c$$

אי

$$U_c(n+1) \leq \frac{n+2}{n+1}U_c(n) + 2c$$

נפתור את זה בשיטת האיטרציה:

$$\begin{aligned} U_c(n+1) &\leq \frac{n+2}{n+1} \left(\frac{n+1}{n} U_c(n-1) + 2c \right) + 2c \\ &= \frac{n+2}{n+1} \cdot \frac{n+1}{n} U_c(n-1) + \frac{n+2}{n+1} \cdot 2c + \frac{n+2}{n+1} \cdot 2c \\ &\leq \frac{n+2}{n+1} \cdot \frac{n+1}{n} \left(\frac{n}{n-1} U_c(n-2) + 2c \right) + \frac{n+2}{n+1} + \frac{n+2}{n+1} 2c \\ &\leq \frac{n+2}{n+1} \cdot 2c + \frac{n+2}{n+1} \cdot 2c + \frac{n+2}{n} + 2c + \frac{n+2}{n+1} \cdot \frac{n+1}{n} \left(\frac{n}{n-1} U_c(n-2) + 2c \right) \end{aligned}$$

רואים כי כל כופלי $U_c(\dots)$ מצתמצמים ונשאר המונה הראשון והמכנה האחרון
אי

$$U_c(n+1) \leq (n+2) 2c \sum_{j=1}^{n+2} \frac{1}{j} + (n+2) \cdot 2c$$

פרק 6

הרצאה מס.?

השיעורים הקודמים היו סתם הכנה למתמטיקה. היום נכנס למבנה הנתונים. סוג נתונים אבסטרקטי¹ הוא אוסף נתונים + פעולות. דוגמעות

- מחסנית: push, pop LIFO

- תור: enqueue, dequeue FIFO

6.1 תור קדימיות²

קיימים שתי סוגים של תור זה:

- Min Priority Queue

- Max Priority Queue

פעולות:

- $max(S)$ מחזירה רשומה עם ערך קדימות הכי גבוה.

- $extract - max(S)$ אותו דבר אבל עם הוצאת הרשומה מהתור.

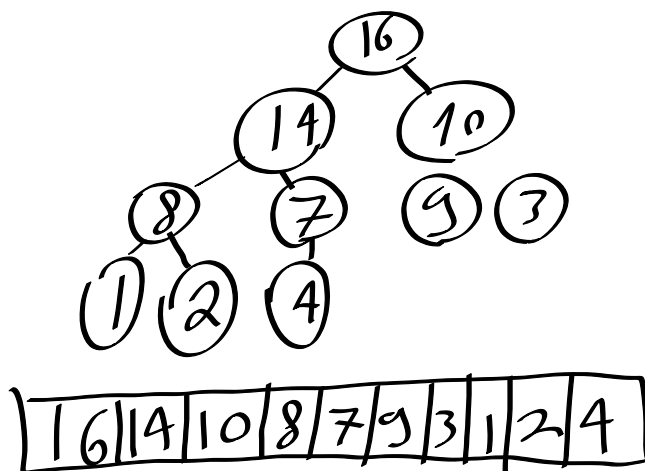
- $insert(S, x)$ מכניסה לתור רשומה עם קדימות x

- $increase_key(S, i, key)$ מעלה את קדימות של האיבר i בתור ל key

6.1.1 מימוש בעזרת ערימה

ערימה זו עץ קמעט שלם שמקיים תכונת הערימה כלומר: $\forall i : A[parent(i)] \geq A(i)$

Abstract Data Type¹
Priority Queue²



איור 6.1: עץ קמעט שלם עם הצגתו במערך

במערך של העץ מתקיים:

$$\begin{aligned}
 left(i) &= 2i \\
 right(i) &= 2i + 4 \\
 parent(i) &= \left\lfloor \frac{i}{2} \right\rfloor
 \end{aligned}$$

רוצים לממש את הפעולות של ה ADT של תור הקדימות כך שמבנה הערימה ישמר.

אז נממש את הפעולות שהגדרנו:

 לחזעיפטם 1.6 פעולת $extract_max(S)$ בערימת קדימות

```

max(S){
return A(1);
}
extract-max(S){
max=S(1)
swap(min(S), S(1))
heapsize--;
max_heapify(A,1);
}
min(S) {
return S[heapsize(S)]
}
max_heapify(A,i) {
l=left(i);
r=right(i)
if(l ≤ heapsize[A]) then {
if(r > heapsize[A]) then r=reapsize(A)
if(A(l) > A(i)) largest=l, else largest=i
if(A(r) > A(largest)) largest=r
if(largest ≠ i) then swap (A[largest], A[i]), max_heapify(A,largest)
}
}

```

הסיבוכיות של maxheapify היא $\mathcal{O}(\log n)$

 לחזעיפטם 2.6 פעולת $increase_key(A, i, key)$ בערימת קדימות

```

increase.key(A, i, key) {
if(key < A[i]) error "new key smaller than old"
A(i)=key
while i>1 and A[parent(i)] < A[i] {
swap(A[i], A[parent(i)])
i=parent(i)
}
}

```

סיבוכיות של פעולה זו היא גם $\mathcal{O}(\log n)$

 לחזעיפטם 3.6 פעולת insert

```

heapsize(a)++
A(heapsize(A)) = -∞
increase_key[A, heapsize(A), key]

```

6.1.2 איך מייצרים ערימה?

יש כל מיני פתרונות שעובדות ב $\mathcal{O}(n \log n)$ אבל יש גם דרך לעשות את זה ב $\mathcal{O}(n)$:

לחזעיפטם 4.6 בנית ערימה

```

build_heap(A) {
  heapsize[A]=length[A]
  for(i=n to 1) max_heapify(A,i)
}

```

זה נראה כמו $\mathcal{O}(n \log n)$ אבל באמת זה יוצא $\mathcal{O}(n)$
האלגוריתם הזה עובד כי:

1. כל שורש בקודקוד בערימת מקסימום זה שורש של ערימת מקסימום.
2. כל קודקוד בעץ כמעט שלם הוא שורש של עץ קמעט שלם
3. אם מפעילים max_heapify על קודקוד i ששני בניו שורשי ערימות מקסימום אזי i יהיה שורש של ערימת מקסימום

פרק 7

הרצאה מס.?

ערמה זה עץ כמעט שלם
כל קודקוד גדול אושווה לבניו.
נרצה לכתוב את האלגוריתם לבנות ערמה חדשה ממערך

לחצעיפטם 1.7 בנית ערימה $\text{BuildHeap}(A)$

```
buildHeap(A) {  
  n=length(A)  
  for i=n to 1 do{  
    max_heapify(A, i)  
  }  
}
```

נרצה עכשיו לראות למה זה עובד ומה הזמן ריצה של זה.¹
הדרך להוכיח את זה הוא דרך אינדוקציה.
אינטואיציה: האלגוריתם כל פעם מתקן הערימה הקטנה שיש לו ומתחיל מהסוף
עד להתחלה. אז כל פעם שאנחנו מפעילים max_heapify יש לנו כבר שתי תתי ערימות
תקינות שמחברים לקודקוד שאנחנו מפעילים אותה עליו

הוכחה פורמלית: נשתמש במשהוא שנקרא Loop Invariant שיש לו שלוש תכונות:

1. קודקוד בעץ כמעט שלם הוא שורש של עץ כמעט שלם.
2. קודקוד בערימת מקסימום הוא שורש של ערימת מקסימום.
3. אם מריצים max_heapify על קודקוד שבניו הם ערימת מקסימום אז מתקבלת ערימת מקסימום.

בשלו ה- i הקודקודים $n, n-1, \dots, n-i+1$ הם שורשים של ערימת מקסימום

הוכחה:

בסיס אינדוקציה: עבור בסיס $k=1$ זה טריוויאלי כי כל עלה הוא כבר ערימת מקסימום.

הנחת האינדוקציה: הנחת האינדוקציה: $n, n-1, \dots, n-k+1$ הם שורשים לערימת מקסימום.

¹זאת הבעיה הראשונה שאנחנו נוכיח כי היא עובדת.

היכחה עבור $k+1$: לפי הנחת האנדוקציה $left(n-k), right(n-k)$ כי האינ-קסים האילו יותר גדולים מ- $(n-k)$ הם ערימות מקסימום. לכן מותר לבצע `max_heapify`, לכן אחרי הפעלת `max_heapify` על $n-k$ הוא נהפך שורש של ערימת מקסימום. ולפי ההנחה כל תת-עץ הוא גם ערימת מקסימום. זה מה שרוצים להוכיח.

7.1 זמן ריצה של Max_Heapify

יודעים כבר כי `max_heapify` הוא רץ בסיבוכיות זמן $\mathcal{O}(\log n)$. אבל זה זמן ריצה מקסימאלי $\mathcal{O}(\log n)$ ולא $\Theta(\log n)$ אז נרשום ביטוי יותר מדויק: מסמנים עימק העץ ב- D אז ברמה הראשונה (השורש) יש D פעולות. ברמה השניה יש לכל קודקוד $D-1$ פעולות אזי $2^1(D-1)$. הביטוי הכללי עבור כל רמה d הוא $2^d(D-d)$. אזי

$$\sum_{n=1}^D h \cdot 2^{D-h} = 2^D \sum_{h=1}^d h 2^{-h} = 2^D \sum_{h=1}^d h \left(\frac{1}{2}\right)^{-h}$$

יודעים כי אם $x \in (0, 1)$ אזי $\lim_{h \rightarrow \infty} \sqrt{x}^h = 0$ אז קיים k' כך שבו מתקיים $\sqrt{x}^{k'} = 1$

$$\begin{aligned} \sum_{k=1}^{\infty} kx^k &= \sum_{k=1}^{k'} kx^k + \sum_{k=k'+1}^{\infty} kx^k \\ &= C + \sum_{k=k'+1}^{\infty} \underbrace{k\sqrt{x}^k}_{<1} \sqrt{x}^k \\ &\leq C + \sum_{k=k'+1}^{\infty} 1 \cdot \sqrt{x}^k \\ &\leq C + \frac{1}{1-\sqrt{x}} + c \end{aligned}$$

שהביטוי הזה תלוי רק ב- x שזה במקרה שלנו $(\frac{1}{2})$

7.2 עץ חיפוש בינארי

עץ בינארי בו עבור כל קודקוד x מתקיים

• עבור y בתת עץ שמאלי של $key(y) \leq key(x)$

• עבור y בתת עץ ימיני של $key(y) > key(x)$

רוצים שעץ הזה יתמוך בפעולות הבאות:

• `Insert(T,x)`

• `Remove(T,x)`

• `Search(T,x)`

• `min(T)`

- $\max(T, x)$
- $\text{Successor}(T, x)$
- $\text{Pre}....(T, x)$

אז הקל ביותר זה Max, Min ומחזירים האיבר הימיני ביותר או השמאלי ביותר.

לחזעיפטם 2.7	$\text{Search}(\text{node}, \text{value})$
	<pre> search(n,x) { if(key(n)==x n==null) return n if(key(n)<x) return search(right(n),x); else return search(left(n),x); }</pre>

למצא successor:

- אם ל x יש בן ימני y אזי $\min(T, y)$ עוקב של x
- אין בן ימיני אזי נעלה בעץ עד שנמצא עץ ש x הוא בן ימיני שלו.
- אם x הוא ה maximum

פרק 8

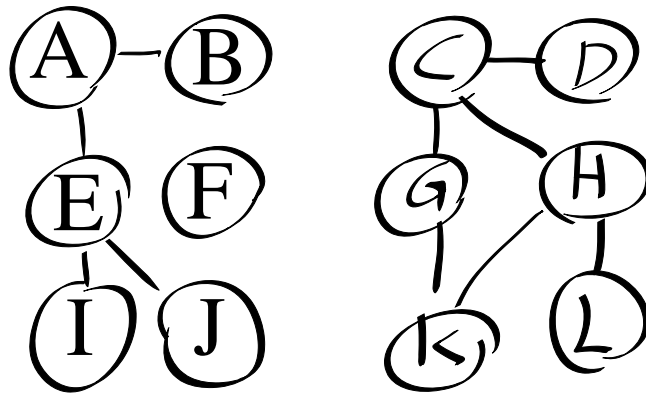
הרצאה מס.?

לחזעיפטם 1.8	Explore(v)
	$\begin{array}{l} \text{Explore}(v) \{ \\ \text{visited}(v)=1; \\ \text{previsit}(v) \\ \text{for all } u, (v,u) \in E \text{ if } (\text{visited}(u)=0; \\ \text{eplore}(u); \\ \text{postvisit}(v) \\ \} \end{array}$

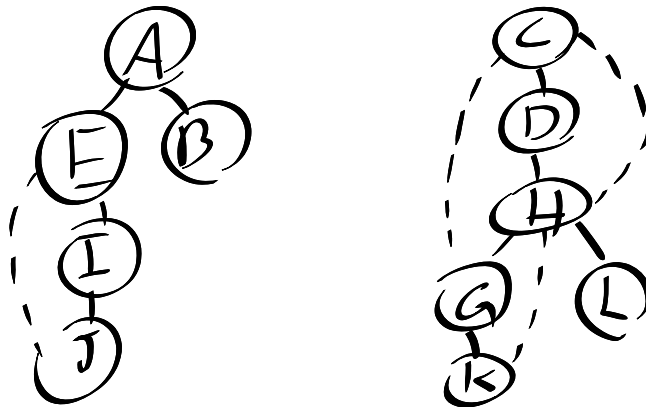
טענה: הפרוצדורה explore מגיע לכל הקודקודים בגרף G שניתן להגיע אליהם מ v

הוכחה: נניח כי יש מסלול מקודקוד v לקודקוד z : $v \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow z$.
אם Explore לא הגיע ל z יש קודקוד אחרון v_j שכן הגיע אליו ולא הגיע ל v_{j+1} וזה לא יתכן כי explore סורקת את כל הכנים של v_i

לחזעיפטם 2.8	DFS(G)
	$\begin{array}{l} \forall v \in V : \text{visited}(v) = 0 \\ \forall v \in V : \text{if } \text{visited}(v) = 0, \text{ explore}(v) \end{array}$

איור 8.1: דוגמה של $DFS(G)$

דוגמה: אזי הפונרציה תעבור על הקודקודים בצירה הבאה וכו מקווקו זה צלע שלא עברה עליהם הריקורסיה):



איור 8.2: גרף ריקורסיה

סיבוכיות של DFS זה $\mathcal{O}(|E|) + \mathcal{O}(|V|) = \mathcal{O}(|E| + |V|)$

8.1 רכיבי קשירות בגרף לא מכוון

נגדיר¹:

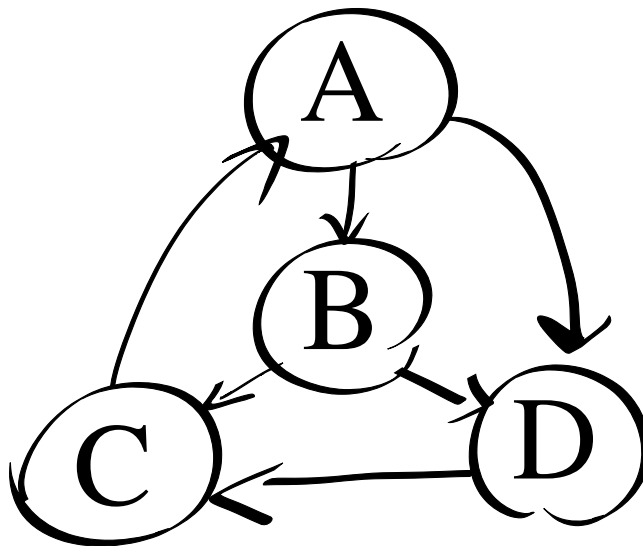
$CC(v)$ מחזיר רכיב הקשירות שבו נמצא הקודקוד v .

$CCNum$ יספור רכיבי קשירות. $CCNum=0$ בתחילת DFS .

בכל פעם ששקוראים ל- $explore$ בתוך DFS נעסה $CCNum \leftarrow CCNum + 1$ ונגדיר הפונקציה $previsit$ להיות $CC(v) \leftarrow CCNum$. זה יגדיר לנו את רכיב הקשירות של כל קודקוד.

¹Connected Component - רכיב קשירות

8.2 גרף מכוון



איור 8.3: דוגמה לגרף מכוון

יש כמה סוגים של צלעות בגרף זה:

1. Tree Edges

2. Forward Edges: צלע לצאצא לא ישיר בגרף

3. Backedges: צלע שהולכת לאב קדמון בעץ

4. Cross Edges: כל דרך אחר

8.2.1 גרף מכוון ללא מעגלים DAG: Directed Acyclic Graph

נרצה למצא מיון טופולוגי הגרף כזה: שזה סידור של קודקודים כך שלכל צלע $v \rightarrow w$, w יופיע אחרי v

למשל באיור 8.3 A, B, D, C זה מיון טופולוגי אם מוחקים את הצלע CA

יש מיון טופולוגי אם "ם אין צלע מסוג Backedge

8.2.2 רכיבי קשירות בגרף מכוון

רכיב קשירות חזק זה אוסף מקסימאלי של קודקודים כך שלכל זוג קודקודים v, w יש מסלול מ v ל w ומ w ל v .

לחזעיפטם 3.8 שעון

```
clock=0
previsit(v)
pre(v)=clock
clock=clock+1
postvisit(v)
post(v)=clock
clock=clock+1
```

טענה: לכל צלע בגרף DAG $v \rightarrow w$ מתקיים $post(v) > post(w)$ אם נמין לפי ערכי $post$ בסדר יורד נקבל מיון טופולוגי של DAG .