جامعة طرابلس كلية تقنية المعلومات

مذكرة يقين PDF

Design and Analysis Algorithms

تحليل وتصميم الخوارزميات

ITGS301

إعداد الطالب :

محمد احنيش

المحاضرة 1:

Intro

Algorithms

تصميم وتحليل الخوارزميات (ITGS301)

محاضرة #1

ما المقصود بالخوارزمية Algorithm؟

- هي سلسلة قائمة بذاتها من الإجراءات التي يتعين القيام بها.
- أو هي مجموعة من الخطوات لإنجاز أو إكمال مهمة موصوفة بدقة كافية بحيث يمكن لجهاز كمبيوتر تشغيلها.

★- هي عبارة عن مجموعة من الخطوات المنطقية المحددة لهدف أو الغرض تعطي حل أو إجابة مسألة في فترة زمنية معينة.

خصائص الخوارزمية: المعايير التي يجب أن تتوفر في الخوارزمية:

1- يجب أن يأخذ إدخالات (مدخلات) (على الأقل صفر أو أكثر)
2- يجب أن يعطي بعض المخرجات (نعم/لا، قيمة) مخرج واحد أو أكثر
3- الوضوح: كل تعليمات واضحة ولا لبس فيها
4- أن تكون خوارزمية محدودة تنتهي بعد العديد من الخطوات بمعنى تعطي نتيجة معينة أو محدودة.
5- الفعالية: يجب أن تكون كل تعليمات أساسية أي تعليمات بسيطة.

★ ما الفرق بين الخوارزمية Algorithm والبرنامج Program؟

الخوارزمية هي أسلوب أو فن لكتابة كود برمجي

Programs = Algorithms + Data structures

كيف يتم تمثيل الخوارزمية؟

1- اللغة الطبيعية Natural Language: اللغة الإنجليزية هي حاليا أفضل طريقة.

ولغة البرمجة

2- المخططات الانسيابية.

3- كود مزيف (Pseudocode)

★ ما الهدف من الخوارزميات؟

بناء برنامج أكثر كفاءة وبالتالي عند كتابة برنامج من الضروري كتابة خوارزمية أولاً.

- أي مشكلة / مسألة يوجد بها عدة حلول وبالتالي نحتاج لتحليل الخوارزمية حتى يتم اختيار أفضل حل مسألة معينة وهي التي تعمل بشكل أسرع (لديه تقصير وقت أقل)

★ ما المقصود بتحليل الخوارزمية؟

تقييم اداء الخوارزميات من حيث الكفاءة.

★ كيف يتم حساب كفاءة الخوارزمية؟ (العوامل المستخدمة لتحديد كفاءة الخوارزمية)

1- سرعة الاداء (سرعة التنفيذ الخوارزمية)

2- المساحة space حجم مساحة الذاكرة التي تحتاجها لتنفيذ الخوارزمية (تعتبر عامل ثانوي)

Running time :- هو عدد العمليات الأولية (الخطوات) المنفذة قبل الإنهاء.

★ تقاس كفاءة الخوارزمية memory space ، Running time.

(يزيد زمن التشغيل بزيادة حجم المدخلات)

كيف يتم حساب زمن التنفيذ؟

باستخدام أحد المعايير الاساسية الثلاثة

1. Big-O O

2. Omega Ω

3. Theta Θ

س/ من الادوات والمعايير المستخدمة في حساب زمن التنفيذ الخوارزمية؟

- أول عامل من عوامل هو Big-O

ما معنى Big-O ← مثلاً يوجد لدينا مصفوفة {5,4,15,50,10}

كود مزيف :- هو مزيج من اللغه الطبيعيه ولغه البرمجه تقوم احيانا بتضمينا عبارات اللغه الانجليزية في الشفرة الزائفة لذلك، على عكس لغه البرمجه الحقيقه لا يمكننا إنشاء مترجم يترجم (Pseudocode) إلى كود الآله.

المطلوب كتابة خوارزمية لإيجاد مجموعة من العناصر هذه المطلوبة؟
كم عملية تم تنفيذها ← n من العمليات

زمن التنفيذ هو عبارة عن عدد العمليات (الخطوات) الذي يتم تنفيذها داخل الخوارزمية.

في هذا المثال يمكن استخدام أحد المعايير ولكن المعيار الأنسب في حساب زمن التنفيذ هو معيار Big O

Big O ← هو أفضل معيار من المعايير الثلاثة لحساب زمن التنفيذ الخوارزمية

مثال/ إيجاد أكبر عنصر في مصفوفة {4, 50, 13, 10, 5}؟

الحل

أو مقارنة كل عنصر بالعنصر الذي يليه ← زمن التنفيذ n من العمليات
مقارنة كل عنصر بجميع العناصر ← زمن تنفيذ n^2 من العمليات

ما هي الحالات لتحديد الوقت الخوارزمية؟

1. worst case (الحالة الأسوأ)
هو أطول زمن تنفيذ ممكن أن تستغرقه الخوارزمية يوفر حداً أعلى لوقت التشغيل (الحد الأعلى للتكلفة)

2. Best case (أفضل حالة)
هو أقصر زمن تنفيذ ممكن أن تستغرقه الخوارزمية يوفر حداً أدنى لوقت التشغيل (الحد الأدنى للتكلفة)

3. Average case (متوسط الحالة)
هو الوقت المتوقع لحساب زمن التنفيذ الممكن أن تستغرقه الخوارزمية يوفر تنبؤ حول وقت التشغيل (تكلفة متوقعة)

note: ← يحسب زمن التنفيذ بـ unit of Time ويتم تمثيلها بـ second عادة.

المحاضرة 2:

Asymptotic Notation

Big Oh , Omega , Theta

Date:
No:

- عند حساب Running time عادة Best case يكون بدون فائدة و Average case مفيد للغاية ولكن غالباً ما يصعب تحديده

* نحن نركز على تشغيل worst case الزمن لأنها أسوأ حل في التحليل ومهمة بالنسبة لتطبيقات مثل الألعاب والتمويل والروبوتات

- معدل النمو وقت التشغيل ← هو المعدل الذي تكلفه الخوارزمية تنمو مع نمو حجم المدخلات.

$f(n) = \underline{6n^2} + 100n + 50$

Upper Bound → الحد الأعلى

$f(n) = \underbrace{2n^3}_{\text{Upper Bound}} + 3n^2 + 100n$

$f(n) = \underbrace{4\log_2(n)}_{} + 40$

Examples: Leading Terms

$a(n) = \frac{1}{2}n + 4$

Leading Terms: $\frac{1}{2}$

: Upper Bound: The Big-O Notation

- : O-notation (Big-Oh)

يمثل Big-O الحد الأعلى لوقت تشغيل الخوارزمية. وبالتالي فإنه يعطي worst case تعقيد الخوارزمية.

التعريف الرياضي Big-O ←

دالة: تمثل تحليل زمن تعقيد الخوارزمية $f(n) = O(g(n))$

إذا تحققت العلاقة ← شرط

if $f(n) <= C.g(n)$ ، $C > 0$ ، $n_0 > 0$

ثابت ومن شروطه أن يكون موجب

نقول أن $g(n)$ هي Big-O للدالة $f(n)$ عندما ← $0 \leq f(n) \leq C\,g(n)$

حيث أن $n > n_0$

- Omega (Ω): هو معيار يحسب أقل مدة زمنية ونعبر عنه Best Case أفضل زمن تنفيذ.

- Theta (Θ): هو المعيار الذي يساوي قيمة worst case و Best case وهنا يعني الحالة المثالية أو Avg Case.

Date:
No:

Example #1

نظهر أن

$f(n)=n+5$ $\quad g(n)=n$ $\quad$ Show that $f(n)=O(g(n))$

$C=6$

الجواب

قوانين { $f(n)=O(g(n))$ $\quad c,n_0>0$

$f(n) <= C.g(n)$ $\quad n>n_0$

تعويض مباشر { $n+5 <= C.n$

$n+5 <= 6n$

$f(n) = O(n)$

حل بطريقة

$f(n) <= c\, g(n)$

$n+5 <= n+5$

$n+5 <= n+n$

$n+5 <= 2n$

n_0

بالتعويض ←

$5+5 <= 2(5)$

$10 <= 10$ ✓

من هنا عرفنا أن $C=2$

$n_0=5$

مثال آخر لهذه الطريقة

$f(n)=3n+2$ $\quad g(n)=n$

$f(n)<=C.g(n)$

$3n+2 <= 3n+2$

$3n+2 <= 3n+n$

$3n+2 <= 4n$ $\quad (C=4)$

$(n_0=2)$

$3(2)+2 <= 4(2)$ بالتعويض بقيم n_0, c

$8 <= 8$ ✓

* طريقة أخرى من وضع قيمة ابتدائية فرضي

$f(n)=n+5$ ، $g(n)=n$

$f(n) <= C.g(n)$

$n+5 <= C.n$

فرضي ان $(n_0=1)$

$1+5 <= C.1$

$6 <= C$ → $n_0=1$ في حالة $(C=6)$

بالتعويض في الدالة

$n_0+5 <= 6.n_0$

$1+5 <= 6.1$

$6 <= 6$ ✓ (تتحقق العلاقة)

Example 2 #

أثبت ان وقت التشغيل $f(n)=3n^2+10n$ هو $O(n^2)$

إثبات: من خلال التعريف

$f(n)=O(n^2)$

$f(n) <= C.g(n)$ حيث ان $C,n_0>0$

$3n^2+10n <= C.n^2$ → $\frac{3n^2}{n^2}+\frac{10n}{n^2} <= \frac{Cn^2}{n^2}$ بالقسمة على n^2

$3+10/n <= C$

بفرض ان $n_0=1$

$3+10/1 <= C$

$13 <= C$

تم الإثبات اذ وجدنا $C=13$ و $n_0=1$ يحقق الشروط

★ اذا كانت $f(n) = \Theta(g(n))$ فإن $O(g(n)) = \Omega(g(n))$

★ $N^2 + N\log N = O(N^2)$

★ اثبت ان زمن التنفيذ للدالة $f(n) = n^3 + 4n + 2$ يساوي $O(n^3)$

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

$f(n) <= C.g(n)$ $\quad C, n_0 > 0$

$n^3 + 4n + 2 <= C.n^3$

$\frac{n^3}{n^3} + \frac{4n}{n^3} + \frac{2}{n^3} <= C.\frac{n^3}{n^3}$

$1 + 4/n^2 + 2/n^3 <= C$ $\quad$ نفرض ان $n_0 = 1$

$1 + 4/1^2 + 2/1^3 <= C$ $\quad \leftarrow$ بالتعويض

$7 <= C$

ثم اثبات ان عندما $C=7$ و $n_0=1$ يتحقق الشروط وهو المطلوب $f(n) = O(n^3)$

$1^3 + 4(1) + 2 <= 7.(1)^3$

$7 <= 7$ ✓ تتحقق العلاقة

★ اثبت ان زمن التنفيذ للدالة $f(n) = 2n^2 + 3n + 1$ يساوي $O(n^2)$

$f(n) = O(g(n))$ $\quad C, n_0 > 0$

$f(n) <= C.g(n)$

$2n^2 + 3n + 1 <= C.n^2$

Date:
No:

$2 + 3/n + 1/n^2 <= C$

$2 + 3 + 1 <= C$ ← $n_0 = 1$ نفرض أن

$6 <= C$

تم اثبات انه عندما $C=6$ و $n_0=1$ يتحقق الشرط
$f(n) = O(n^2)$

Ex: → $f(n) = 3n+2$ $\quad g(n) = n$

نفرض ← $(n_0 = 1)$

$f(n) <= C.g(n)$

$3n+2 <= C.n$

$(5 <= C)$

حل آخر

$3n+2 \leq C.g(n)$

$\leq 3n+2$

$\leq 3n+3$

$\vdots$

$\leq \underbrace{3n+n}_{4n}$

$C=4$

n_0	$3n+2$	$4n$
2	8	8

تحقق علاقة $8 \leq 8$

متى تتحقق العلاقة؟
النقطة التي تتحول فيها النقطة الأكبر من $f(n)$ إلى $C.g(n)$ تسمى n_0 وتتحقق العلاقة

$f(n) = 3n^3 + 5$

	Big-O	Theta	Big-Omega	
	n^4			
$3n^3+5$	n^3	n^3	n^3	يوجد أقل منا
			n^2	
			n	

Big_O ممكن أن يكون يوجد قيم أعلى منا

note:- في الامتحان لما السؤال يقول حل المطلوب f(n) يعني الخطوات المطلوبة، عندما يقول أوجد زمن التنفيذ المطلوب هو O( ).

Ex: #4

```
for (i=1 ; i<=n ; i++)        // 1, n+1, n
    sum+=i  -> n
for ( j=1  j<=n ; j++)        // 1, n+2, n+1
{
        x+=j  -> n+1
}
```

حل وأوجد زمن التنفيذ

$2n+2$ — $O(n)$

$1+n+2+n+1 = O(n)$

$2n+4$ — $O(n)$

$O(n)$

قانون ← زمن التنفيذ مقدار الزيادة دائماً أقل من زمن تنفيذ الشرط بواحد

$f(n) = 2n+2+n+2n+4+n+1$

$f(n) = 6n+7 \rightarrow O(n)$

Ex:-

```
for (i=1 ; i<n ; i++)
    for (j=0 ; j<i ; j++)
    {
        statement
    }
```

تعني متسلسلة ويستخدم القانون

* في حالة بداية الشرط بـ 0 الداخلي مع for يكونوا

* في حالة بداية الشرط بـ 1 الداخلي for يكون n

زمن التنفيذ ← $T(n) = O(n^2)$

$\sum_{j=1} = 1+2+3+ \dots n$

عندما أكل وهي تعني متسلسلة ونستخدم القانون ← $n(n+1)/2$

و statement سنكون نضعها بـ 1-

$n(n+1)/2 \rightarrow n^2+n/2 \Rightarrow \frac{1}{2}(n^2+n)$

عند وجود = يكون $\log_2 n+2$

عند عدم وجود = يكون $\log_2 n+1$

Date:

note:

لما الـ i يكون مضروب في 2 أو مقسوم يكون تحليل الفقرة دائماً $\log n$

Ex: $\log_2 n+2$ $\log_2 n+1$

```
for (i=1; i<=n; i=i*2)
    sum=sum+i;
```

$\log_2 n+1$

نفرض $n=8$

i=1	i≤n	i*2
1	1≤8	$2 \to 2^1$
2	2≤8	$4 \to 2^2$
4	4≤8	$8 \to 2^3$
8	8≤8	$16 \to 2^4$
16	16≤8	x
⋮		
n^k		

$2^{k} * 2 \quad k=i$

$2^x \le n \to 2^x = n$

$k = \log_2 n$

$f(n) = O(\log_2 n)$

* يعني 2 أس شي لحد ما يكون الناتج n

Ex:

```
for (i=n; i>=1; i=i/2)
    sum=sum+i;
```

نفرض $i=10$

i=10	i≥1	i/2
10	✓ 10≥1	5
5	✓ 5≥1	2.5
2.5	✓ 2.5≥1	1.25
1.25	✓ 1.25≥1	0.7
0.7	x 0.7≥1	x

$i<n \to \log_2 n+1+1$

$\log_2 n+2$

$i<n \to \log_2 n+1$

$O(\log_2 n)$

?

Ex:-   Best case

```
if(x<n)                       → 1      ⇒ O(1)
    Printf(%d;n);               1      ⇒ O(1)
else          1       n+2   n+1
  { for(i=0;i<n;i++)           2n+4    ⇒ O(n)
      Printf(i)          n     ⇒ O(n)
  }
```

$T(n) = O(n)$

note ← في if, else
ناخذ أعلى فيهم لأن بتنفذ واحدة فيهم فقط

2. Ω-notation (Omega)

$$f(n) = \Omega(g(n))$$

if $f(n) \Rightarrow C.g(n)$

بشرط $C, n_0 > 0 \quad n \geq n_0$

$g(n)$ lower bound for $f(n)$

الحد الأدنى

Example #1

$f(n) = 5n^2$ $\qquad\qquad \Omega(n^2)$

$f(n) = \Omega(g(n))$

$f(n) \Rightarrow C.g(n)$

$\dfrac{5\cancel{n^2}}{\cancel{n^2}} \Rightarrow \dfrac{C.\cancel{n^2}}{\cancel{n^2}} \rightarrow C = 5$

بفرض $c=5$ $\qquad$ أو $5n^2 \Rightarrow C.n^2$

و $n_0=1$ $\qquad$ $5(1) \Rightarrow 5(1)$

$5 \Rightarrow 5$ ✓

Example #2

$f(n) = 2n+3$

$f(n) = \Omega(g(n))$

$f(n) \Rightarrow C.g(n)$

$2n+3 \Rightarrow C.n$ $\qquad$ بفرض أن $n_0=1$

$5 \Rightarrow C \Leftarrow 2+3 \Rightarrow C$

3- Θ notation (Theta)

$$f(n) = \Theta(g(n))$$

$$C_1 \cdot g(n) <= f(n) <= C_2 \cdot g(n)$$

بشرط $c_1, c_2, n_0 > 0$ ، $n >= n_0$

g(n) tight bound of f(n)

$f(n) = \Theta(g(n))$ فإن

$O(g(n)) = \Omega(g(n))$

Example #1

$f(n) = 3n+2$ ، $g(n) = n$ $C_1 = 3$ $C_2 = 4$

$$f(n) = \Theta(g(n))$$

$$C_1 \cdot g(n) <= f(n) <= C_2 \cdot g(n)$$

نفرض أن $n_0 = 2$

$$3n <= 3n+2 <= 4n$$

$$6 <= 8 <= 8$$

$$f(n) = \Theta(n)$$

thita $\Leftarrow \Theta(n)$

$$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < 2^n < n!$$

* نفرض أن $O(n)$ ، هذا كله يعتبر Upper bound وبالتالي يحقق العلاقة.
* نفرض أن $\Omega(n)$ ، هذا كله يعتبر lower bound وبالتالي يحقق العلاقة.

Date:
No:

Example # $f(n)=10n^4+3n^2+5n$

اثبت ما اذا كانت العلاقات صحيحة

T 1- $T(n)=O(n^5) \to n^4 \le n^5$ ✓

F 2- $T(n)=O(n\log n) \to n^4 \le n\log n$ ✗

T 3- $T(n)=O(n^7) \to n^4 \le n^7$ ✓

F 4- $T(n)=\Omega(n^5) \to n^4 \ge n^5$ ✗

T 5- $T(n)=\Omega(n^3) \to n^4 \ge n^3$ ✓

F 6- $T(n)=\Theta(n^2) \to n^4 = n^2$ ✗

T 7- $T(n)=\Theta(n^4) \to n^4 = n^4$ ✓

Note:

- $f(n) \le C(g(n)) \to O$
- $f(n) \ge C(g(n)) \to \Omega$
- $C_1(g(n)) \le f(n) \le C_2(g(n)) \to \Theta$

- ايجاد العلاقات باستخدام الغايات

$\lim_{n\to\infty} \frac{f(n)}{g(n)} =$

1. $0 \Rightarrow f(n)=O(g(n))$
2. $\infty \Rightarrow f(n)=\Omega(g(n))$
3. $c \Rightarrow f(n)=\Theta(g(n))$

Example #1

$f(n)=5n^2-4n-10$ ، $g(n)=n^2$

اثبت ما اذا كانت $f(n)=O, \Omega$ or Θ باستخدام الغايات.

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} \to \lim_{n\to\infty}\frac{5n^2-4n-10}{n^2} \to \lim_{n\to\infty}\frac{5\cancel{n^2}}{\cancel{n^2}}-\frac{4\cancel{n}}{n^{\cancel{2}}}-\frac{10}{n^2}$$

$$\lim_{n\to\infty} 5 - \frac{4}{n} - \frac{10}{n} \rightarrow 5 - \cancel{\frac{4}{\infty}}^{\,0} - \cancel{\frac{10}{\infty}}^{\,0} = 5$$

$f(n) = \Theta(g(n))$

Example #2

$f(n) = n^2$ ، $g(n) = n\log n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \rightarrow \frac{n^2}{n\log n} \rightarrow \lim_{n\to\infty} \frac{n}{\log n} \rightarrow \frac{\infty}{\log\infty} = \infty$$

$f(n) = \Omega(g(n))$

Example #3

$f(n) = \sqrt[3]{n}$ ، $g(n) = \sqrt{n}$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \rightarrow \lim_{n\to\infty} \frac{\sqrt[3]{n}}{\sqrt{n}} \rightarrow \lim_{n\to\infty} \frac{n^{\frac{1}{3}}}{n^{\frac{1}{2}}}$$

$$\lim_{n\to\infty} \frac{n^{\frac{1}{3}}}{n^{\frac{1}{2}}} \rightarrow \lim_{n\to\infty} n^{\frac{1}{3}-\frac{1}{2}} \rightarrow \lim_{n\to\infty} n^{-\frac{1}{6}} \rightarrow \lim_{n\to\infty} \frac{1}{n^{\frac{1}{6}}}$$

$$\rightarrow \cancel{\frac{1}{\infty}}^{\,0} = 0$$

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

المحاضرة 3:

Insertion Sort

Date:
No:

محاضرة 3

من أشهر الخوارزميات 1. خوارزمية الترتيب
2. خوارزمية البحث

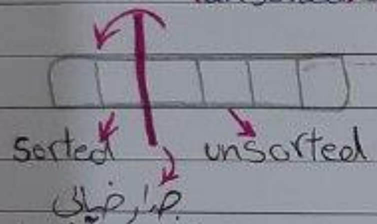
Sorting

يعد الفرز من أكثر تطبيقات معالجة Data شيوعاً حيث يتم ترتيب Data وفقاً لقيمها.

خوارزمية الترتيب بالإدخال وبالإدراج

عادة عندما نرتب أي قائمة تكون ترتيب تصاعدي أو تنازلي وإذا لم يتم تحديد ترتيب الفرز، فمن المفترض أن يكون تصاعدياً.

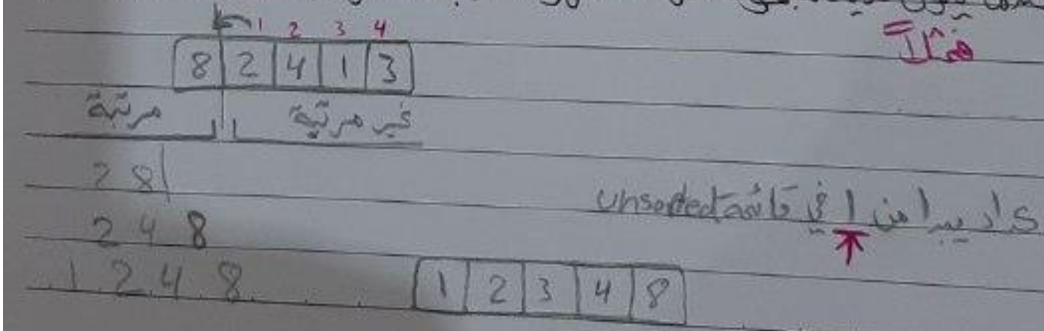
فكرة الخوارزمية إن نتخيل أو نفترض وجود جدار خيالي (افتراضي) وظيفته يقسم القائمة List إلى جزئين: 1. مرتبة (sorted)
2. غير مرتبة (unsorted)



تقوم بترتيب جميع عناصر تبدأ بعنصر عنصر في قائمة الغير مرتبة تبدأ بإدراجه (إدخاله) في قائمة المرتبة في مكانه الصحيح

عندما يكون لدينا مجموعة من عناصر والمطلوب منا ترتيب أولاً نبدأ بأول عنصر

مثلاً



Date:
No.

```
InsertionSort (A,n){
  for i=2 to n {          أقل من أو يساوي
     Key = A[i]
      j = i-1;
     while(j>0) and (A[j]>key){
           A[j+1] = A[j]
             j = j-1
     }
     A[j+1] = Key
  }
}
```

1	2	3	4	5
8	2	4	1	3

i=2 , Key = 2 , j=1

while(1>0) & (8>2) T

A[2] = 8 ← A[j+1] = A[j]

j=0

8	~~8~~	4	1	3

while (0>0) & ______ F

لا يتحقق الشرط

A[j+1] = Key

A[1] = 2

2	8	4	1	3

نزيد عداد ال i بواحد و j أقل من ال i بواحد

i=3 , Key=4 , j=2

while(2>0 & 8>4) T

A[3]=8

j=1

2	8	8	1	3

while(1>0 & 2>4) F

لا يحقق الشرط

A[2]=4

1	2	3	4	5
2	4	8	1	3

i=4 , Key=1 , j=3

while(3>0 & 8>1) T

A[4]=8

2	4	8	8	3

j=2

while(2>0 & 4>1) T

A[3]=4

j=1

2	4	4	8	3

while(1>0 & 2>1) T

A[2]=2

j=0

2	2	4	8	3

while(0>0 & ____) F

A[1]=1

1	2	4	8	3

Date:
No:

$i=5$ Key $=3$ $j=4$
while(4>0 & 8>3) T
$A[5]=8$
$j=3$

1	2	4	8	8

while(3>0 & 4>3) T
$A[4]=4$
$j=2$

1	2	4	4	8

while(2>0 & 2>3) F
لا يحقق الشرط
$A[3]=3$

1	2	3	4	8

Sorted

Time Complexity ← تعقيد الوقت

```
n=5
For i=2 to n{                      n
    Key = A[i]        ← 4 مرات      n-1
    j=i-1             ← 4 مرات      n-1
    while(j>0 & A[j]>key){          Σ_{i=2}^{n} ti
        A[j+1] = A[j]               Σ_{i=2}^{n} (ti-1)
        j=j-1                       Σ_{i=2}^{n} (ti-1)
    }
A[j+1]=Key            ← 4 مرات      n-1
}
```

قانون ← النهاية - البداية + 1 + 1 = $n-2+1+1=n$

Date:
No:

الان عدد مرات المقارنة يختلف من حالة لأخرى ونلاحظ انه حتى في نفس الـ list تختلف فعبرنا عنها بحرف t

t: تمثل عدد مرات إختبار الشرط الـ while

ومعادلات الموجودة داخل الـ while هتكون أقل كزمن تنفيذ الان تم إختبارها في حالة تحقق الشرط فقط

وهذا تقليل زمن تنفيذ الخوارزمية

توضيح

Statement	Time	Best case	worst case
For i=2 to n	n	n	n
Key = A[i]	n-1	n-1	n-1
j=i-1	n-1	n-1	n-1
while(j>0)&(A[j]>key)	$\sum_{i=2}^{n}(t_i)$	n-1	n(n-1)/2
A[j+1]=A[j]	$\sum_{i=2}^{n}(t_i-1)$	$\sum_{i=2}^{n}(1-1)=0$	n(n-1)/2
j=j-1	$\sum_{i=2}^{n}(t_i-1)$	$\sum_{i=2}^{n}(1-1)=0$	n(n-1)/2
A[j+1]=key	n-1	n-1	n-1

Best case running time:

$$T(n) = n+(n-1)+(n-1)+(n-1)+0+0+(n-1)$$

$$=5n-4$$

$$T(n)=\Omega(n)$$

worst case running time:

$$T(n)=n+(n-1)+(n-1)+n(n-1)/2+n(n-1)/2+n(n-1)/2+(n-1)$$

$$=n+n-1+n-1+n^2-n/2+n^2-n/2+n^2-n/2+n-1$$

$$T_n=O(n^2)$$

تحليل شرح كيف تم الخوارزمية

for(i=2 ; i<=n ; i++)

when (n=5)

for i=2 to n

معناها إلى n اي يساوي

i<=n

i<=5	i++
1 2<=5 ✓	2+1
2 3<=5 ✓	3+1
3 4<=5 ✓	4+1
4 5<=5 ✓	5+1
5 6<=5 X	

عدد مرات المقارنة = 5 يعني ← n

عدد مرات الزيادة = n-1 ← i++

$T(n) = O(n)$

- زمن تنفيذ ال while سيكون $\sum_{n}^{i=2} t_i$

والموجود داخل ال while Loop يتنفذ: $\sum_{n}^{i=2} t_i - 1$

* Best case لهذه الخوارزمية هي عندما تكون القائمة مرتبة

[1|2|3|4|8]

جميع القيم ال while loop ستنفذ مرة

في هذا المثال ← $\sum t_i = 4$ ويكون الناتج عليه ب n-1

وما داخل ال while Loop زمن التنفيذ لها ← Zero

* Worst case تكون عندما القائمة تكون مرتبة عكسياً

[7|5|4|3|1]

$\sum_{n}^{i=2} t_i = 1+2+3+4+ \ldots +n$

رياضياً ← $n(n+1)/2$

القانون الرياضي يبدا من ال 1 ومسألة الموجودة من 2 فالتالي

زمن التنفيذ While ← $n(n-1)/2 - 1$

ما داخل ال while loop

$n(n-1)/2$

المحاضرة 4:

RECURRENCE RELATIONS

Date:
No:

محاضرة 4

* تصنف الخوارزمية إلى
1. Non Recursion Algorithm
2. Recursion Algrothm

كمصطلح عام Recursion ← هي عملية تكرار مستمر أو عملية تعيد نفسها

كمصطلح في البرمجة هو دالة تعيد نفسها أو استدعاء ذاتي.

* في خوارزمية السابقة كنا نحسبوا في زمن تنفيذ كل طريقة نجمعوهم لنحصلوا على زمن التنفيذ بدلالة Big O وتسمى الخوارزميات Non Recursion

* النوع الثاني Recursion Algr وهي الخوارزمية التي تحتوي على دالة تستدعي نفسها أو ذاتية الاستدعاء ← مثلاً Function Z()

المفروض يكون في جملة شرطية Function Z() عادة لكي تقف استدعاء الدالة أو توقف التكرار

* Recursion Function شرطين
1. general Case oR Recursion base
الاستدعاء الذاتي للدالة.
base Case
2. نقطة التوقف (كود ايقاف الاستدعاء)

- الخوارزمية التي تحتوي على حل تكرار تسمى
Iteration Algorithm oR Non-recursion Algorthm

- الدالة recrursive ممكن تكون دالة ← Iteration

مثال:

خوارزمية لحساب مضروب العدد N!

[n=5]　　$5! = 5*4*3*2*1$

120 Factorial(5) = 5 * Factorial(4)
24 Factorial(4) = 4 * Factorial(3)
6 Factorial(3) = 3 * Factorial(2)
2 Factorial(2) = 2 * Factorial(1)
1 Factorial(1) = 1

Recursion base _1_
Factorial(X) = X * Factorial(X-1)
base case _2_
Factorial(1) = 1

Note: ← نتوقف عندما نصل لمضروب الواحد

الخوارزمية: ←

```
int Factorial(n)
  if(n=1)
     return(1)
  else
     return(n*Factorial(n-1))
```

في حالة ال 3 = معلوم 1 → base case

recrusive case ← return(n*Factorial(n-1))

تتابع استدعاء

قانون $T(n) = \boxed{\text{Recursion}} + \boxed{\text{base case}}$ → شكل المعادلة

recurse Equation

تسمى معادلة التكرار ← $T(n) = T(n-1) + 1$

يمثل الشرط (base case) ويمكن كتابة C بدل كتابة 1 والتي C تمثل ثابت

$T(5) = T(4) + 1$

$T(4) = T(3) + 1$

n=5

ومعادلة التكرار تصف زمن التنفيذ للخوارزمية

* حل الخوارزمية السابقة بالطريقة القديمة (الأولى)

```
{ int fact=1, n;                      1        O(1)
  for (i=0; i<n; i++)                 2n+4     O(n)
      fast = fast *(i+1)              n+1      O(n)
```

Date:
No:

i	Fact	Fact(i+1)
0	1	1(0+1)
1	1	1(1+1)
2	2	2(2+1)
3	6	6(3+1)
4	24	24(4+1)
(5	120)	

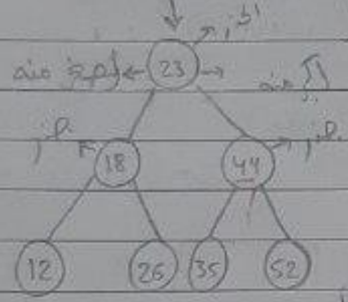
$T(n) = O(1) + O(n) + O(n)$

$T(n) = O(n)$

* خوارزمية البحث binary tree

في هذه الخوارزمية تقوم بتقسيم العناصر الموجودة في القائمة إلى جزئين بتقسيم عدد العناصر على 2، وتقوم بوضع العنصر الأوسط في الجذر والعناصر في اليمين الجذر أكبر منه والعناصر يسار الجذر أصغر منه ثم تقوم برؤية العنصر المراد البحث عنه اذا كان يساوي الجذر ترجع قيمة من أول موضوع وإذا لا تقوم برؤيته هل هو أصغر أم أكبر من الجذر، إذا كان أكبر تذهب للعناصر يمين الجذر، وإذا كان أصغر تذهب إلى العناصر يسار الجذر كل Node تحقق الخصائص الخاصة بهذه الخوارزمية.

12	18	20	23	35	44	52



Lo ← مؤشر يؤشر إلى أول عنصر في list

hi ← مؤشر يؤشر إلى آخر عنصر في List

Mid ← يمثل عنصر الموجود في منتصف القائمة

Lo			Mid			hi
12	18	20	23	35	44	52
a_0	a_1	a_2	a_3	a_4	a_5	a_6

mid → (Lo + hi)/2 → (6/2) → 3

مثال: X يمثل الرقم الذي نبحث عنه في القائمة

if (X = Mid)

تعني X تساوي الجذر

if (X < Mid)

هيستدعي الدالة وهيبحث في العناصر من الجهة اليسار

if (X > Mid)

هيستدعي الدالة وهيبحث في العناصر من الجهة اليمين

* فكرة الخوارزمية

من الممكن كتابة هذه الخوارزمية بالحلقات التكرارية سيتم استدعاء الدالة على نصف حجم input اي لان عملية البحث تتم $\frac{n}{2}$ من العناصر

① Recursion $= \frac{n}{2}$ ② base Case $= C$

∴ زمن التنفيذ

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

باقي المحاضرة 4

recrusion Equation المضروب

$$T(n) = T(n-1) + 1$$

نحتاج إلى تحليل هذه المعادلة للحصول على زمن التنفيذ بدلالة ال Bio_o

كيف نحللها؟

بأحد طرق حل معادلات التكرار

هناك عدة طرق حل [حوالي 6-7 طرق] سناخذ 3 طرق منهم

أول طريقة سنستخدمها لتحليل معادلة التكرار

Iteration Method

فكرتها ان نحصل على متسلسلة في الاخر تسمى الصيغة النهائية ClosedEnd

الصيغة النهائية معادلة مبسطة ينتهي فيها التكرار

Example:-

$T(n-1)$

$T(n-2)$

$T(n-3)$

$\vdots$ لحد Base Case

$T(n-K)$

$T(1) = 1$ شرط التوقف

Date:
No:

سنجد زمن التنفيذ للتكرار الثاني والثالث إلى ان نتحصل على الصيغة النهائية (العامة) عند تكرار K وبعدها نعوض بنقطة التوقف بعدين بدلالة n حتى نتحصل على زمن تنفيذ بدلالة المعادلة نهائية نتحصل منها)
Big_O

① خوارزمية مضروب n!

$T(n) = T(n-1) + 1$

المعادلة الاصلية

$T(n-1) = T(n-1-1) + 1$

زمن تنفيذ التكرار الاول
الذي سنعوض في التكرار الثاني

(نتوقف عندما نصل n-1 ← T(n)=1 ← T(n)=1)

② $= T(n-2) + 1$

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

$T(n) = T(n-2) + 2$ ← زمن تنفيذ التكرار الثاني

$T(n) = T(n-1) + 1$

(هذا حالياً مجهول ولكن نظلعه من التكرار السابق الثاني)

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

③ $T(n) = T(n-3) + 2 + 1$

$T(n) = T(n-3) + 3$

زمن تنفيذ التكرار الثالث

متى نتوقف؟

عند الوصول الشرط التوقف [في المضروب =1] ولكننا لا نعرف قيمة n ولهذا نستنتج:

الصيغة العامة General form

* كذا نقدر نستنتجها من ثالث تكرار

$T(n) = T(n-1) + 1$ التكرار الأول

$T(n) = T(n-2) + 2$ التكرار الثاني

$T(n) = T(n-3) + 3$ التكرار الثالث

من هنه التكرارات نستنتج الصيغة العامة بدلالة K

$T(n) = T(n-K) + K$ تكرار K (الصيغة العامة)

حتى نتحصل على شرط التوقف $n-k=1$

note:

$T(n) = T(1) + K$ $\quad$ $n-K=1$

$T(n) = 1 + K$ $\quad$ $n=1+K$

$T(n) = \cancel{1} + n \cancel{-1}$ $\quad$ $K=n-1$

$T(n) = n$

الصيغة النهائية Closed End

$\therefore T(n) = O(n)$

Example #2

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$n=1$ $\quad$ $T(n)=1$ $\quad$ $T(1)=1$

① $T(n) = T\left(\frac{n}{2}\right) + 1$

$T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$

② $T(n) = T\left(\frac{n}{4}\right) + 1 + 1$

$$= T\left(\frac{n}{4}\right) + 2$$

$$T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$$

③ $T(n) = T\left(\frac{n}{8}\right) + 1 + 2$

$T(n) = T\left(\frac{n}{8}\right) + 3$

⋮

Ⓚ $T(n) = T\left(\frac{n}{2^k}\right) + k \rightarrow$ الصيغة العامة

نقطة التوقف $T(n) = 1 \rightarrow T(1) = 1$ ونعوضوا بهاذي الصيغة العامة.

$$T(n) = T\left(\frac{n}{2^k}\right) + k \rightarrow \frac{n}{2^k} = 1 \rightarrow n = 2^k$$

ملاحظة $x = a^y$ ، $y = \log_a x$

$$k = \log_2 n$$

$$T(n) = T(1) + \log_2 n \rightarrow T(n) = \lg n + 1$$

$$\therefore T(n) = O(\lg n)$$

Example #3

$T(n) = 2T(n-1) + 1$ ، $n = 0 \rightarrow T(n) = 0$

① $T(n) = 2T(n-1) + 1$

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

② $T(n) = 2\left[2T(n-2) + 1\right] + 1$

$$= 4T(n-2) + 2 + 1$$
$$= 4T(n-2) + 3$$

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

③ $T(n) = 4(2T(n-3)+1)+3$

$= 8T(n-3)+4+3$

$= 8T(n-3)+7$

$\vdots$

(k) $T(n) = 2^k T(n-k) + 2^k - 1$ → الصيغة العامة

note

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

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

3 $T(n) = 2^3 T(n-3) + 2^3$

$\vdots$

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

$n-k=0$

نعوضوا ب $n=0$ ← $T(n)=0$

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

$T(n) = 2^k \cancel{T(0)} + 2^k - 1$

$T(n) = 2^k - 1$

بما ان $n-k=0$ ← $k=n$ بالتعويض في

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

$\therefore T(n) = O(2^n)$

المحاضرة 5:

Master Method

Date:
No:

محاضرة 5

Recurrence relations: Master Method

ثاني طريقة لإيجاد الزمن التنفيذ لمعادلات التكرار هو Master Method

الفكرة من هذه الطريقة حتى يتم إيجاد الزمن التنفيذ بدلالة الـ Big-O لذا من الضروري يكون على هذه الصيغة

$$T(n) = a\,T(n/b) + f(n)$$

بشرط: $a => 1$ ، $b > 1$

لازم يتحققوا حتى نستعمل Master Method

أول نظرية في Master Method:

يجب انه نجد قيمة $n^{\log_b^n}$ من المعادلة وهذا المقدار حيتم مقارنته مع $f(n)$

$$n^{\log_b^a} \approx f(n)$$

* 3 حالات

Case 1: $n^{\log_b^a} > f(n)$

في هذه الحالة زمن التنفيذ حيساوي: $T(n) = O(n^{\log_b^a})$

(احيانا نكتب بـ Θ بدل O عادي نكتب وحدة منهم)

Case 2: $n^{\log_b^a} = f(n)$

$T(n) = O(n^{\log_b^a} \cdot \lg n)$ نكتب Θ عادي

Case 3: $n^{\log_b^a} < f(n)$

$$T(n) = \Theta(f(n))$$

حيث أنه: n هو حجم المشكلة

a هو عدد المشاكل الفرعية في التكرار

n/b هو حجم كل مشكلة فرعية. (من المفترض هنا أن جميع المشكلات الفرعية هي في الأساس من نفس الحجم)

$f(n)$ هو مجموع العمل المنجز خارج الاستدعاءات التكرارية والذي يتضمن مجموع قسمة المشكلة ومجموع دمج الحلول للمشكلات الفرعية.

Example 1#

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

$a=9$, $\frac{n}{b} = \frac{n}{3}$, $b=3$

$f(n)=n$ $\quad a \geq 1 \rightarrow 9 \geq 1 \quad b>1 \rightarrow 3>1$

$n^{\log_b a} \rightarrow \log_b a = \log_3 9 = 2$

$n^{\log_b a} \approx f(n)$

$n^2 \approx n$ $\quad$ (Case 1)

$n^2 > n$ $\quad \therefore T(n) = O(n^{\log_b a})$

$T(n) = O(n^2)$ ← الإجابة

Example 2#

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

$a=8$, $b=2$, $\frac{n}{b} = \frac{n}{2}$, $f(n)=n^2$

$n^{\log_b a} \rightarrow \log_b a \rightarrow \log_2 8 = 3$ $\quad 2^{\square} = 8 \leftarrow \log_2 8 = 3$ الحل هاي

n^3

$n^{\log_b a} \approx f(n)$

$n^3 \approx n^2$ $\quad$ (Case 1)

$n^3 > n^2$ $\quad \therefore T(n) = O(n^{\log_b a})$

$T(n) = O(n^3)$ ← الإجابة

Example 3#

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

$a=1$, $b=\frac{3}{2}$ $\quad f(n)=1$

$\therefore \frac{2n}{3} = \frac{n}{\frac{3}{2}} \rightarrow b = \frac{3}{2}$

$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = \text{zero}$

$n^0 \approx 1$

$1 = 1$

Case 2: $T(n) = O(n^{\log_b a} . \lg n)$

$T(n) = O(n^0 . \lg n)$

$= O(1 . \lg n)$

$\therefore T(n) = O(\lg n)$

* Inadmissible equations المعادلات غير مقبولة

لا يمكن حل المعادلات التالية باستخدام النظرية الرئيسية:

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

a ليس ثابتاً يجب إصلاح هذه المشكلات العرضية

. $T(n) = 0.5 T(\frac{n}{2}) + n$

لا يمكن أن تحتوي الحد على أقل من مشكلة فرعية واحدة

. $T(n) = 64 T(\frac{n}{8}) - n^2 \log n$ →

$f(n)$ ← ليس موجباً

. $T(n) = T(\frac{n}{2}) + n(2 - \cos n)$

لكن انتهاك الانتظام Case3:

نسخة موسعة من نظرية الـMaster

* Extended Version of Master Theorem

$T(n) = a T(\frac{n}{b}) + \Theta(n^k \log^p n)$

$T(n) = 2 T(\frac{n}{2}) + n \log n$ ← مثال حالة خاصة تعتبر $\Theta(n^k \log^p n)$

عندما يكون عندنا log هل تكون حالة خاصة

(K بمقارنة مع $\log_b a$) (K تطلع من المعادلة)

($a \geq 1$ ، $b > 1$ ، $k \geq 0$ ، p هو رقم حقيقي)

→ $\log_b a$, K

Case1 → if $a > b^k$ أو $\log_b a > K$

then $T(n) = \Theta(n^{\log_b a})$

Case 2 → if $a = b^k$ أو $\log_b a = K$

then (a) if $p > -1$ → $T(n) = \Theta(n^k . \log^{p+1} . n)$ [illegible]

(b) if $p = -1$ → $T(n) = \Theta(n^k \log\log n)$

(c) if $p < -1$ → $T(n) = \Theta(n^{\log_b a})$ أو n^k

Case 3 → if $a < b^k$ أو $\log_b a < K$

then (a) if $p \geq 0$ → $T(n) = \Theta(n^k \lg^p n)$

(b) if $p < 0$ → $T(n) = \Theta(n^k)$

Example #

$$T(n) = 2T(\tfrac{n}{2}) + n \log n$$

$T(n) = a\,T(\frac{n}{b}) = \Theta(n^k \log^p n)$ ← نقارن المثال بمعادلة الحالة الخاصة لكون القيم

$a=2$ $b=2$ $k=1$ $p=1$ ← من مقارنة المعادلتين نستنتج أن:

$\log_2 2 = 1$ ← نوجد قيمة $\log_b a$

* Case → $\log_b a = k$ → $1=1$

$\therefore$ Case2 when $P > -1$ ← $P=1$ ← ننظر قيمة P

$T(n) = \Theta(n^k . \log^{p+1} n)$

$T(n) = \Theta(n . \log^2 n)$

($T(n) = a\,T(\frac{n}{b}) + Cn^k$) ← الشكل البسيط لنظرية الـ Master *

Example #

$$T(n) = 4T(\tfrac{n}{2}) + n$$

$a=4$ $b=2$ $k=1$

$\log_b a = \log_2 4 = 2$

$\therefore \log_b a > K$

$2 > 1$

$\therefore T(n) = O(n^{\log_b a})$

$T(n) = O(n^2)$

$a \geq 1$ $b > 1$

Case 1: $T(n) = O(n^{\log_b a})$ if $a > b^k$

Case 2: $T(n) = O(n^k \log n)$ if $a = b^k$

Case 3: $T(n) = O(n^k)$ if $a < b^k$

$f(n) = \Theta(n)$ → $f(n) = O(n)$

يعني $f(n) = \Theta(g(n))$ نستطيع أن نقول $f(n) = O(g(n))$

Date:
No.

Example

$$T(n) = 4T(\frac{n}{2}) + n^3$$

$a=4 \quad b=2 \quad k=3 \quad b^k = 2^3$

$\therefore a < b^k$

$\rightarrow$ case 3 $\rightarrow$ $T(n) = O(n^3)$

Example

$$T(n) = 2T(\frac{n}{2}) + 1$$

$a=2 \quad b=2 \quad k=0 \quad b^k = 2^0$

$\therefore a > b^k$

$\rightarrow$ Case 1 $\rightarrow$ $T(n) = O(n^{\log_b a})$

$= O(n^2)$.

Example

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

$T(n) = 2T(\frac{n}{2}) + n.\log^{-1} n$

$a=2, \; b=2, \; k=1 \quad p=-1$

$\log_b a = \log_2 2 = 1 \quad \rightarrow 1=1 \rightarrow \log_b a = k$

$p=-1$

$-1=-1 \quad T(n) = O(n^k . \log\log n)$

$= O(n . \lg\lg n)$

$\log_a(bc) = \log_a(b) + \log_a(c)$

$\log_a(b^c) = c\log_a(b)$

$\log_a(\frac{1}{b}) = -\log_a(b)$

$\log_a(1) = 0$

$\log_a(a) = 1$

$\log_a(a^r) = r$ ← قوانين

$\log_{1/a}(b) = -\log_a(b)$

$\log_a(b)\log_b(c) = \log_a(c)$

$\log_b(a) = 1/\log_a(b)$

$\log_a m(a^n) = \frac{n}{m} \quad m \neq 0$

المحاضرة 5:

الجزء (2)

Tree Method

Date:
No:

محاضرة 5 جزء 2

Recursion Tree Method

ثالث طريقه لإيجاد زمن تنفيذ لمعادلات التكرار هو RecursionTree التي تكون معادلتها بالصورة $T(n) = a\,T(\frac{n}{b}) + f(n)$

• من اسمها Tree شجرة يعني حيكون عندنا Root و Nodes اي لدينا Tree ، SubTree

الفكرة العامة (Recursion Tree) سنقوم بتحويل معادلة التكرار إلى شجرة هذه الشجرة تكون على شكل مستويات Levels لغاية النقطه التي يتوقف فيها تكرار.... سنقوم بإيجاد cost كل مستوى وبعد ما نجمعهم لنتحصل على متسلسلة نحلها للحصول على زمن التنفيذ بدلالة Big O

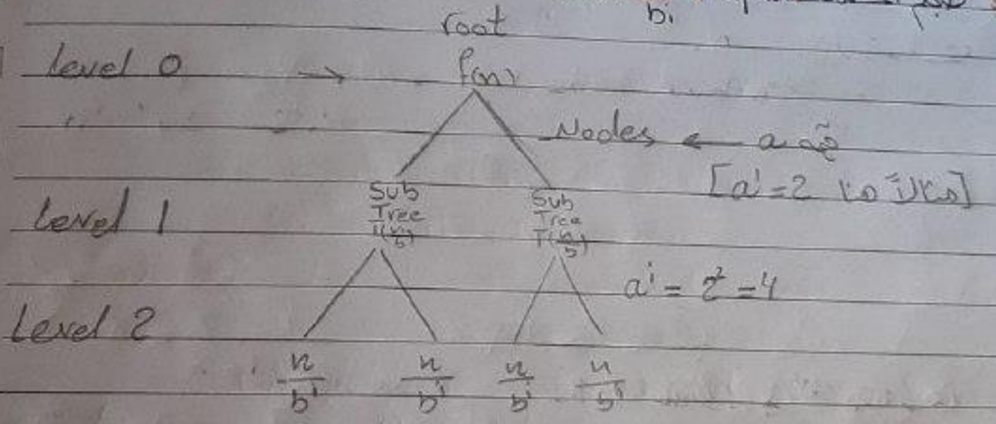
صيغه المعادلة $T(n) = a\,T(\frac{n}{b}) + f(n)$

حيث ان ← جذر الشجرة قيمتها = $f(n) \rightarrow$ Tree root

• عدد Nodes سيحدد بقيمة a

• SubTree قيمتها سيكون $T(\frac{n}{b})$

• حجم المشكلة (input) $\frac{n}{b_1}$ يمثل الاستدعاء الاول الثاني ....



س كم سيكون عندي مستوى ؟

في المثال نطلع 3 مستويات ونجمعهم

زمن التنفيذ عند اخر مستوى = 1

Date:
No:

تكلفه كل مستوى هي عبارة على تكلفه عند مستوى i : قانون *

$a^i \cdot T\left(\frac{n}{b^i}\right)$

حيث تطبق هذا القانون على كل مستوى

$\sum a^i \cdot \frac{f(n)}{b^i}$

قيمة i عند آخر مستوى = $\log_b n$

* كيف عرفنا قيمة i ؟

نحن نعرف ان زمن تنفيذ اخر مستوى = 1 وكذلك نعرف ان زمن تنفيذ (Cost)

المستويات نستنتج منهم انه $\frac{f(n)}{b^i} = 1 \rightarrow \frac{n}{b^i} = 1$

باستخدام قانون اللوغاريتمات $i = \log_b n$

$\therefore$ ارتفاع الشجرة = $\log_b n$

التكلفه عند اخر مستوى $\leftarrow T(n) = a^{\log_b n}$

باستخدام قانون اللوغاريتمات $T(n) = n^{\log_b a}$

$$T(n) = \Theta\Big(\underbrace{n^{\log_b a}}_{\text{زمن تنفيذ عند آخر مستوى}} + \sum_{i=0}^{\log_b n - 1} a^i \cdot \frac{f(n)}{b^i}\Big)$$

من أول مستوى i=0 إلى مستوى ماقبل الأخير

من اول لاخر مستوى $T(n) = \Theta\left(\sum_{i=0}^{\log_b n} a^i \cdot \frac{f(n)}{b^i}\right)$

Example

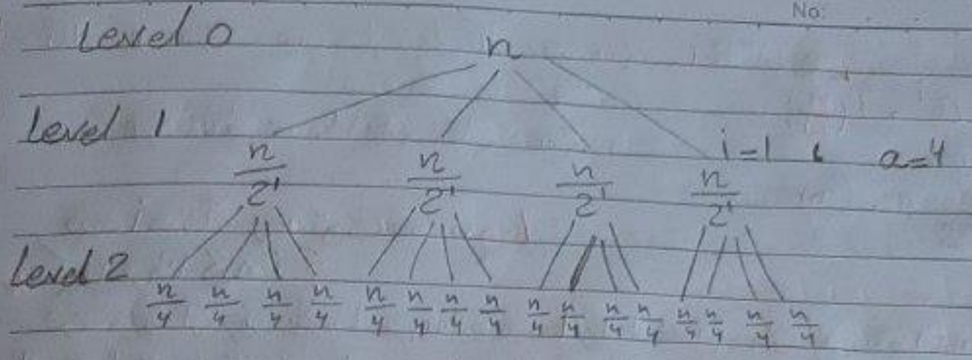
$$T(n) = 4\left(\frac{n}{2}\right) + n$$

قانون زمن تنفيذ المستويات كلها

$T(n) = \Theta\left(n^{\log_b a} + \sum_{i=0}^{\log_b n - 1} a^i \cdot \frac{f(n)}{b^i}\right)$

$a = 4$, $b = 2$, $f(n) = n$

اول خطوة نحول المعادلة إلى شجرة



Cost on level 0:-

$a^i \frac{f(n)}{b^i} \rightarrow 4^0 \cdot \frac{n}{2^0} = n$

Cost on Level 1:-

$4^1 \cdot \frac{n}{2^1} = 2n$

Cost on Level 2:-

$4^2 \cdot \frac{n}{2^2} = 16 \frac{n}{4} = 4n$

Cost on level 3:-

$4^3 \cdot \frac{n}{2^3} = 8n$

تكلفه المستويات $2^i . n$

$T(n) = n + 2n + 4n + 8n + \ldots$ + last level $n^{\log_b a}$

$n^{\log_b a} + \sum_{i=0}^{\log_2 n - 1} 2^i . n$

$= n^{\log_2 4} + \sum_{i=0}^{\log_2 n - 1} 2^i . n$

$= n^2 + n \sum_{i=0}^{\log_2 n - 1} 2^i$

$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1}$ $\quad x \neq 1 \leftarrow$ بشرط $\quad 2^i \approx x^i$ ، $\sum_{i=0 \approx i=0}^{n = \log_b n}$

نعوض قانون المتسلسلة :-

$T(n) = O(n^2 + n\left[\frac{2^{\log_2 n} - 1}{2 - 1}\right])$ $\quad \log_2 n - 1 + 1$

$T(n) = O(n^2 + n(2^{\log_2 n} - 1))$

$$2^{\log_2^n} \cong n^{\log_2^2}$$

$$T(n) = O(n^2 + n(n^{\log_2^2} - 1))$$

$$T(n) = O(n^2 + n^2 - n) \rightarrow T(n) = O(2n^2 - n)$$

$$\therefore T(n) = O(n^2)$$

المحاضرة 6:

Merge Sort

محاضرة 6

Algorithm Design

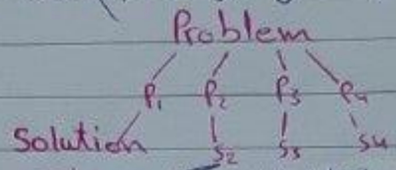
The divide and conquer approach

مفهوم فرق التسوية

3 خطوات أساسيه لتقيم الخوارزمية

1- Divide التقسيم

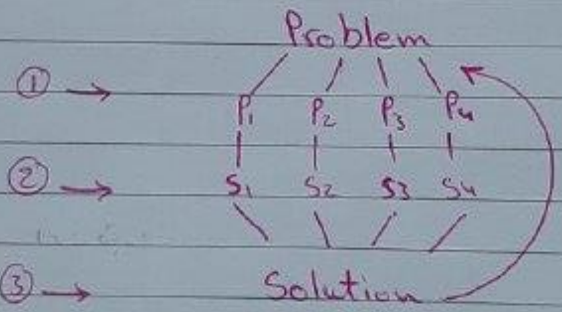
تقسيم المشكله إلى مشاكل أقل في الحجم (أول خطوة تتبع المفهوم).



2- Conquer ايجاد الحلول المشاكل الصغيرة.

هذه الخطوة تتم بتكرار الخطوة الاولى (المشاكل الصغيرة تكون نفس النوع يعني خوارزمية ترتيب مشاكلها مشاكل الترتيب).

3- Combine الدمج للحلول حتى نتحصل على الحل النهائي لهذه المشكلة.



دمج حلول هذه المشكله

Binary Search Tree تتبع هذا المفهوم

∴ خوارزمية الترتيب بالدمج (Merge Sort algorithm)

* تعتمد هذه الخوارزمية على مفهوم فرق التسوية

وتمر بـ 3 مراحل (تقسيم - Conquer - الدمج)

Date:
No:

شرح

Example #

لدينا Array تحتوي على n من العناصر
أول خطوة نقسمها إلى مشاكل صغيرة بمعنى نقسم القائمة بالنص بعد الحصول على قائمتين مستقلين بمعنى يحتاجو حل ويتم تكرار أول خطوة Divide حتى نتحصل على عنصر واحد غير قابل للتقسيم وأخيراً يتم دمج القوائم الفرعية.

يؤشر على أول عنصر — p

يؤشر على آخر عنصر — r

p			q				r
2	4	5	7	1	2	3	6
0	1	2	3	4	5	6	7

إيجاد منتصف القائمة $q = \frac{p+r}{2}$

يؤشر على المنتصف

$q = \frac{0+7}{2} = 3.5 \rightarrow 3$

يتم استدعاء Merge Sorts لتقسيم القائمة

N1 حجم القائمة الفرعية الأولى ← L

N2 حجم القائمة الفرعية الثانية ← R

p q r

2 4 5 7 1 2 3 6

L: 2 4 5 7 R: 1 2 3 6

L: 2 4 R: 5 7 L: 1 2 R: 3 6

2 | 4 | 5 | 7 | 1 | 2 | 3 | 6

إلى هذه الخطوة تكون عملية التقسيم حتى تم الحصول على عنصر واحد غير قابل للتقسيم

عملية دمج

2 | 4 | 5 | 7 | 1 | 2 | 3 | 6

أول ترتيب

2 4 | 5 7 | 1 2 | 3 6

الدمج

L: 2 4 5 7 R: 1 2 3 6

نقارن كل عنصر بعنصر مثلاً
2<1
2≤2

1 2 2 3 4 5 6 7

بهذه الكود يتم ترتيب

```
Do if L[i] ≤ R[j]
   Then A[K] ← L[i]
        i ← i+1
Else
   A[K] ← R[j]
   j ← j+1
```

Date:
No:

$n1 = q - p + 1$ ← تفسير الكود

$n1 = 3 - 0 + 1 \rightarrow 4$ ← طول N_1

$n2 = R - q$

$7 - 3 = 4$ ← طول N_2 (حجم)

سؤال الامتحان ميكون لرسم وليس كتابه الكود

هذه الخوارزمية من امثلة Divide Algorithm ومن ضروري تكون من نفس النوع

$$n \to \frac{n}{2} \quad \frac{n}{2}$$

$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + n$ ← معادلة زمن التنفيذ

$T(n) = 2T(\frac{n}{2}) + n$ المعادلة التي تصف زمن التنفيذ لل Merge Sort لهذه الخوارزمية

$T(n) = 1$ عندما $n = 1$

ممكن ان يتم حلها باستخدام الطرق الاخرى مثال

* طريقه Recursion Tree

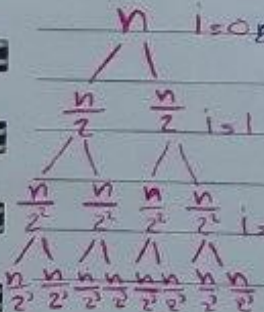
$T(n) = 2T(\frac{n}{2}) + n$

$T(n) = aT(\frac{n}{b}) + n$

الجذر $= f(n)$

عدد Nodes $= a^i$ في المستوى i

قيمه ال Nodes $\frac{n}{b^i}$ (حجم مشكله مستوى i)



$i=0$ $2^0(\frac{n}{2^0}) = n$

$i=1$ $2^1(\frac{n}{2^1}) = n$ $i = a^i f(\frac{n}{b^i})$ تكلفه

$i=2$ $2^2(\frac{n}{2^2}) = n$

$i=3$ مستوى الثالث $2^3(\frac{n}{2^3}) = n$

نحتاج ان نجد تكلفه كل مستوى ونجمع

$$T(n) = \sum_{i=0}^{i=\log_b n} a^i \cdot f\left(\frac{n}{b^i}\right)$$

المستوى الآخير $\Theta\left(n^{\log_b a} + \sum_{i=0}^{i=\log_b n} a^i f\left(\frac{n}{b^i}\right)\right)$

$$T(n) = \sum_{i=0}^{i=\log_2 n} a^i f\left(\frac{n}{b^i}\right) \xleftarrow{\text{تعويض}} T(n) = \sum_{i=0}^{i=\log_2 n} 2^i \left(\frac{n}{2^i}\right)$$

$$T(n) = \sum_{i=0}^{i=\log_2 n} n$$

$$T(n) = \underset{i=0}{n} + n + n + n + \cdots + \underset{i=\log_2 n}{n}$$

تبسيط ←

$$T(n) = n \cdot \sum_{i=0}^{i=\log_2 n} 1$$

قانون ← $T(n) = n \cdot \log_2 n$

$\therefore T(n) = n\log n$

زمن التنفيذ $T(n) = O(n\log n)$

المحاضرة 7:

Quick Sort

Date:
No:

محاضرة 7

خوارزمية Quick Sort

* تتبع مفهوم التقسيم ولكن تختلف في آلية التقسيم.
* الفكرة هي وجود "المحور"
* تقسم القائمة إلى قائمتين غير متساويتين حيث ان القسمة تتم من خلال المحور P.
* بعد تساوي القوائم تكون يمين المحور تكون عناصر الاكبر.
* " " " يسار " " الاصغر.

P

اصغر — اكبر

* ممكن ان نعتبر P اكبر أو أصغر عنصر في قائمة ولكن يفضل اختياره أول عنصر
* عند كل مقارنة يأخذ المحور P موقعه الصحيح
* ليست أفضل خوارزمية من حيث زمن التنفيذ

* Quick Sort Running Time is worst than Merge Sort Running Time.

Example 1 *

المحور ← P

L ←

6	10	13	5	8	3	2	11
1 (i)	2 (J)	3	4	5	6	7	8

R →

(Left, Right نقارن دائماً بالمحور P)

- نقارن جميع العناصر المصفوفه بالمحور
العناصر الأكبر تبقى اخر القائمة Right
" الاصغر " أول القائمة left

شرح

Note: اذا لم يتحقق الشرط سيزيد قيمة J
" تحقق الشرط سيزيد قيمة i

```
if(A [J] < P)
  { i++;
    Swap A[J], A[i] }
```
← كود المقارنة

else

if (10 ≤ 6) → False →

j++

J=3

P i, J

6	10	13	5	8	3	2	11
1	2	3	4	5	6	7	8

if(13<6) → False →

J=4

P i, j

6	10	13	5	8	3	2	11
1	2	3	4	5	6	7	8

Date:
No:

• if (5 ≤ 6) → True

{ i=2

Swap A[j], A[i] }

→ 6 | 5 | 13 | 10 | 8 | 3 | 2 | 11

J=5 → 6 | 5 | 13 | 10 | 8 | 3 | 2 | 11

• if (8 ≤ 6) → False

J=6 → 6 | 5 | 13 | 10 | 8 | 3 | 2 | 11

• if (3 ≤ 6) → True

i++; i=3 → 6 | 5 | 13 | 10 | 8 | 3 | 2 | 11

Swap A[j], A[i]; Swap [13, 3]

J=7 → 6 | 5 | 3 | 10 | 8 | 13 | 2 | 11

• if (2 ≤ 6) → True

i++; i=4 → 6 | 5 | 3 | 10 | 8 | 13 | 2 | 11

Swap {10, 2}

j++; j=8 → 6 | 5 | 3 | 2 | 8 | 13 | 10 | 11

• if (11 ≤ 6) → False

j=9

Swap {P, A[i]}

والمصفوفة أكملت 9 حيث يتم تبديل بين (P و A[i])

1	2	3	4	5	6	7	8
2	5	3	6	8	13	10	11

L P R

2 | 5 | 3   6   8 | 13 | 10 | 11

2   5 | 3   6   8   13 | 10 | 11

2   3 | 5   6   8   10 | 11   13

2   3   5   6   8   10   11   13

ضم القوائم الفرعية يعتمد على القيم نفسها والمحور

2	3	5	6	8	10	11	13

Date:
No:

Note :-

in merge Sort → Best case = worst case

يعني اصلاً في الاساس نطلعوا في تيتا Θ

in Quick Sort → Best Case ≠ worst case

كل Case عندها زمن تنفيذها

Best case

يتم تقسيم القائمة الى $\frac{n}{2}$ تقريبا في البداية ويتم تقسيم الى $n-1$ في حالة قائمة فرعية واحدة

worst case

ايجاد زمن التنفيذ: اولا نجد معادلة التكرار

$T(n) = 2T(\frac{n}{2}) + n$ — Best Case

$T(n) = T(n-1) + n$ — worst Case

(recursion to input size)

نوجد زمن التنفيذ بطريقة Iteration بدلالة Big O

التكرار الأول $T(n) = T(n-1) + n$

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

التكرار الثاني $T(n) = T(n-2) + (n-1) + n$

$T(n-2) = T(n-3) + n-2$

تكرار الثالث $T(n) = T(n-3) + (n-2) + (n-1) + n$

الصيغة العامة: (عند التكرار K)

$T(n) = T(n-K) + (n-(K-1)) + (n-(n-K-2)) + (n-(K-2)) + (n-(K-3)) +$

$+ (n-1) + n$

نوجد الصيغة النهائية

$T(n) = 1 \quad , \quad n = 0 \qquad n-K = 0 \rightarrow n = K$

$T(n) = T(0) + (n-n+1) + (n-n+2) + \dots + (n-1) + n$

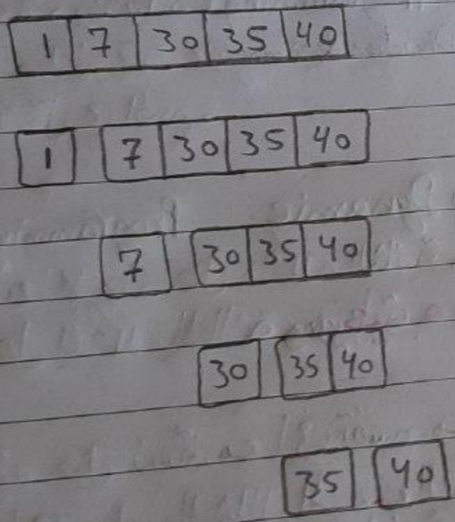
$T(n) = 1 + \underbrace{1 + 2 + \dots + (n-1) + n}_{\sum_{i=0}^{i=n} i}$

56

لأنها سلسلة نستخدم هذا القانون

$$T(n) = 1 + \frac{n(n+1)}{2} \longrightarrow T(n) = O(n^2)$$

Example ثاني بالرسم



المحاضرة 8:

Dynamic Programming

The Fibonacci Sequence Problem

No:

محاضرة 8

(Dynamic Programming)

- مفهوم تقسيم الخوارزمية:-

① Divide & conquer

② (برمجة ديناميكية) Dynamic Programming

تشبه طريقة Divide & Conquer بأنها تقسم المشكلة إلى مجموعة مشاكل الأقل في الحجم.

- مفهوم Dynamic Programming يبدأ بحل أول مشكلة أو المشكلة المصغرة وثم يستخدمها لإيجاد حل مشكلة الثانية وحل المشكلة الاولى والثانية يستعمل لإيجاد حل المشكلة الثالثة وميزاته ← التخزين ومشاكل التكرار.

حل

- هي طريقة حل نستخدمها في تقسيم مشكلة البرمجة وكل مشكلة تحل مرة واحدة ونستخدمها في حلول المشاكل السبب الآن المفهوم الاول Divide& conquer ممكن ان تتكرر فيه المشاكل الفرعية أما Dynamic Programming طالما هو يستخدم الحل الأول في الثاني في هذه الحالة المشكلة الفرعية يتم حلها مرة واحدة فقط اي سيتم تخزين الحلول لحل المشاكل الفرعية التالية فبتالي يمكن استخدام الحل اكثر من مرة بمعنى لما تتم عملية التكرار لحل المشاكل الفرعية وبالتالي هذا المفهوم يحسن من زمن التنفيذ الخوارزمية وأيضاً Dynamic Programming يعطينا الحل الأمثل (الأفضل).

- ماهي المشاكل التي تعالجها الDynamic وتكون موجودة في Divide & conquer؟

ج/ مشكلة عملية التكرار المشاكل الفرعية)

- ما الفرق بين Divide & conquer و Dynamic Programming؟

تعمل على حل المشاكل الفرعية بشكل متكرر

تحل المشكلة الفرعية مرة واحدة (تتجنب إعادة حساب حل المشاكل الفرعية في حل التكرار.

Date:
No:

Example

اكي عالم

سلسلة Fibonacci Sequence هي عبارة على سلسلة تبدأ من Zero و One ومن بعدها كل رقم عبارة عن جمع العنصرين السابقين

$f(0)=0$ ، $f(1)=1$ ، $f(n)=f(n-1)+f(n-2)$

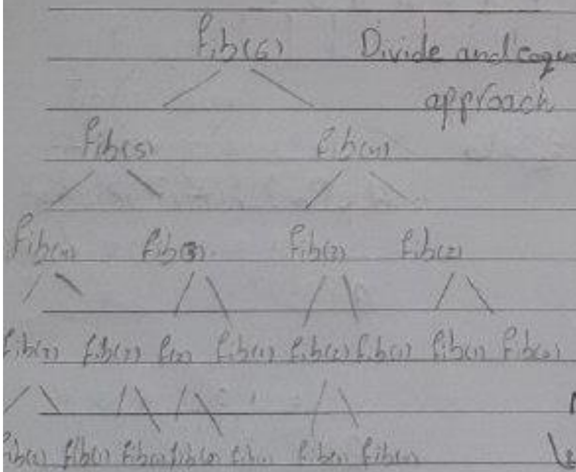
f(n)	0	1	2	3	4	5	6	7	8	9	10 —
القيم	0	1	1	2	3	5	8	13	21	34	55 —

Algorithm ① ... لحساب Fibonacci

n=6 مثلاً

```
fib(n) {
  if(n==0)
    return 0;
  else if (n==1)
    return 1;
  else
    return fib(n-1)+fib(n-2);
```

تتبع الخوارزمية السابقة



Divide and conquer approach

⑧ $f(6)=f(5)+f(4)$

⑤ $f(5)=f(4)+f(3)$

③ $f(4)=f(3)+f(2)$

② $f(3)=f(2)+f(1)$

① $f(2)=f(1)+f(0)$

1 0

Note:

الدالة تحسب ما سبق حسابه مرة ثانية لأنها ليست مخزنة.

No:

- زمن التنفيذ

$T(n) = T(n-1) + T(n-2) + 1$ ← معادلة التكرار

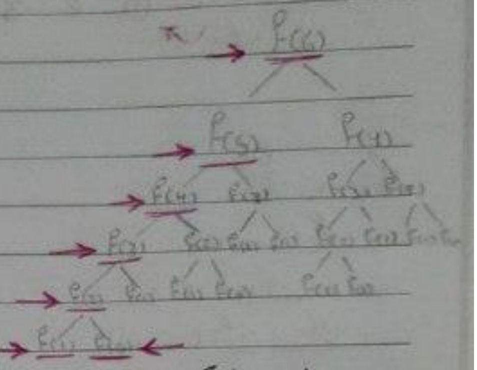
(وعند حلها باستخدام طرق معادلات التكرار فرضوا ان $T(n-2) \equiv T(n-1)$ فأصبحت)

ومن محاضرة 4 كان هذا الناتج ←

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

$T(n) = O(2^n)$ ← زمن تنفيذه يساوي

لحل هذه الخوارزمية بمفهوم الثاني سيتم تخزين كل مشكله فرعية ويتم حلها.



```
fib(n){
   if(n==0)
       return 0
   else if(n==1)
       return 1
   else
     { if A[n] is Null
       A[N]=fib(n-1)+fib(n-2)
       return A[N]
```

عدد مرات التكرار 7 ← $n+1$

$T(n) = n+1 \rightarrow O(n)$

يتم التخزين في Array

شرح هذا الكود

	0	1	2	3	4	5	6
A							

حاليا فارغة ←

شرح الحل تحت

بفرض n=6

-1 if A[6] is Null → ✓ فارغة

A[6] = F[5] + F[4]   (5 + 3 = 8)

	0	1	2	3	4	5	6
A	0	1	1	2	3	5	8

-2 if A[5] is Null ✓

A[5] = fib[4] + F[3]   (3 + 2 = 5)

	0	1	2	3	4	5	6
A	0	1	1	2	3	5	

-3 if A[4] is Null ✓

A[4] = fib[3] + fib[2]

2 + 1 = 3

	0	1	2	3	4	5	6
A	0	1	1	2	3		

4- if A[3] is Null ✓

نقوم بتخزينها في A[3] = fib[2] + fib[1] = 2

5- if A[2] is Null ✓

A[2] = fib[1] + fib[0]

A[2] = 1 + 0 = ① ✓

تم تخزينها في A[2]

Date: No

0	1	2	3	4	5	6
0	1	1	2			

A

مخزنات

نقوم بتخزين الأرقام في المصفوفة

0	1	2	3	4	5	6
0	1	1				

A

Fib(6)

Fib(5)

Fib(4)

Fib(3)

Fib(2)

Fib(1) Fib(0)

(تم إلغاء عمليات التكرار)

Dynamic Programming يستخدم في طريقتين الحل :-

1- Memazition (Top-down method)

2- Tabulation (Buttom-up method)

والمثال السابق هو (Top-down method) و يستخدم في Recursive Algorthim

أما النوع الثاني يستخدم في ITeration Algorthim

مثال لنوع الثاني

```
int fib (int n){
    A[0] = 0;
    A[1] = 1;
    for(i=2; i<=n; i++){
        A[i] = A[i-1] + A[i-2]
    return A[n]
```

نتتبع هذا الكود

بفرض أن n=6

A[2] = A[1] + A[0] = 1

A[3] = A[2] + A[1] = 2

0	1	2	3	4	5	6
0	1	1	2	3	5	8

زمن التنفيذ $T(n) = O(n)$

المحاضرة 8:

Dynamic Programming

الجزء (2)

The Knapsack Problem

محاضرة 8 الباقي

The Knapsack Problem

مشكلة الحقيبة

معطى العناصر X_a ، ...... X_i حيث يكون لكل عنصر X_i وزناً w
والقيمة V_i X_i إذا كانت موضوعة في الحقيبة أوجد المجموعة الفرعية من العناصر
التي يجب وضعها في الحقيبة من أجل زيادة القيمة إلى أقصى حد بافتراض
أن الحقيبة بها سعة w.

حيث حيتم اختيار العناصر التي تكون القيمة منها هي أعلى Value
(Max) بشرط ان مجموع القيم لا يتجاوز وزن الحقيبة

هذه المشكلة منها نوعين :- أو طريقتين

1. 0.1 Knapsack ← فكرته تاخذ العنصر أو تتركه
2. Fractional Knapsack ← فكرته تاخذ العنصر أو تاخذ جزء منه

(0.1 Knapsack)

فكرة هادي الطريقة :-

takes item ← 1 أخذ العنصر

$V(i,w) = V_i + V(i-1, w-w_i)$

doesn't takes item ← 0 لم أخذ العنصر

$V(i,w) = V(i-1,w)$

0.1 Knapsack

item [10] → 0   w=30
item [30] → take
item [40] → 0   30

(Fractional Knapsack)

take it or take part of it

item [20] → take   w=30
item [40] take part
   30

بعنا منه

No:

0-1 Knapsack dynamic Programming

$$V(i,w) = \max\{\underbrace{v_i + V(i-1, w-w_i)}_{\text{Item } i \text{ was taken}}, \underbrace{V(i-1, w)}_{\text{Item } i \text{ was not taken}}\}$$

	0			$w-w_i$				w
0	0	0	0	0	0	0	0	0
	0							
$i-1$	0							
i	0							
	0							
n	0							

$V(i,w) =$ الحد الأقصى للقيمه التي يمكن الحصول عليها من العناصر 1 إلى i إذا كان الحقيبه بحجم w.

Example:

item	weight	value
1	2	12
2	1	10
3	3	20
4	2	15

Capacity $W=5$

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	$\boxed{37}$ → Max

* شرح رسم المصفوفه وتعبئة المصفوفه

ملاحظه ← كل عنصر عنده وزن w_i وعنده قيمه Value (v_i)

عدد الأعمدة تعتمد على وزن (w) الحقيبه

عدد الصفوف يعتمد على عدد الـ item

∴ دائماً آخر عنصر أو آخر قيمة في المصفوفه هي أعلى قيمه (Max)

والعمود والصف الأصفار لا يعدو

في المصفوفه دائماً العمود والصف الأول صفر السبب تكون لا توجد عناصر

بينما لما item=1 وزنه 2 لا يمكن ان نضعه في الحقيبه وزنها 1

وعندما تكون وزن الحقيبه 2 نقدر نضع قيمه الـ item الان وزنه 2

Date:
No:

شرح توضيحي

وكذلك في وزن الحقيبة عندما 3 و4 و5 نضع القيمة السابقة item واحد الان وزن الحقيبة اعلى من وزن الـ item

• عندما item=2 نقوم بحفظ قيمتها لما تكون وزن الحقيبة الان وزن الـ item=1 ولما وزن الحقيبة 2 وكما نعلم ان في الحقيبة موجود فيها العنصر السابق

2
12
1 → 12
2 → 10
12 = Max

3
22
1 → 12
2 → 10
الـ Max = مجموعهم (قيمتهم 22)

وكذلك 4 و 5

• عندما item=3 ووزنه 3

3
22
2 → 12
1 → 10
3 → 20
تم ضم الاثنين الان حجمهم 3 وقيمتهما Max

4
30
2 → 12
1 → 10
3 → 20
$4 = 3 + 1$ وقيمتها 30

5
32
2 → 12
1 → 10
3 → 20
$5 = 3 + 2$
وقيمتهم 32

• عندما item=4 ووزنه 2 = 15

3
25
2 → 12
1 → 10
3 → 20
2 → 15

4
30
2 → 12
1 → 10
3 → 20
2 → 15

5
37
2 → 12
1 → 10
3 → 20
2 → 15

Max القيمة الأعلى = 37 (لمجموع عناصر وزنهم = 5) وهذه أول خطوة

بعد تعبئة الصفوف سيتم تحديد ايهما العناصر التي سوف يتم وضعهم في الحقيبة التي تحقق هذه القيمة أولاً حتى يتم تحديد العناصر سنبدأ بآخر عنصر هو 37

خطوة 2: ← حتى يتم تحديد العناصر نبدأ بآخر عنصر (37) و37 تم أخذ موجود في عنصر رقم 4 تمت تسميته بـ I_4 ← قيمة 15

فأولاً سنقارن القيمة الأخيرة بالعنصر الموجود قبلها (اذا كانت القيمتين متساويات في هذه الحالة سيتم تجاهل العنصر) اما في حالة (اذا كانت

Date:
No:

القيمتين مختلفتين فمناخذ العنصر)

فمثلاً 37 و 32 ← قيم مختلفه فبتالي نأخذ العنصر I_4 وسيكون $=1$ من ضمن الحقيبه.

وزن ← $5-2=3$ & $37-15=22$ ← قيمه

سنشوف قيمه 22 الموجوده في المصفوفه الموجود في العنصر I_3 ونقارنه ونشوفه من ضمن الحقيبه أولا

فمثلاً 22 و 22 ← قيم متشابهات نتعامله لا نأخذ العنصر $I_3 = 0$

والان العنصر I_2 نقارنه مع الموجود قبله

فمثلاً 22 و 12 ← قيم مختلفه فبتالي نأخذ العنصر I_2 وسيكون ضمن الحقيبه $I_2=1$

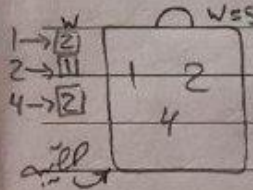
$22-10=12$ & $3-1=2$

سنشوف قيمه 12 الموجوده في المصفوفه الموجوده في العنصر I_1 ونقارنه ونشوفه من ضمن الحقيبه أولا

فمثلاً 12 و 0 ← قيم مختلفه نأخذ العنصر I_1 وسيكون ضمن الحقيبه

$12-12=0$ & $2-2=0$

والعنصر تعني لا توجد سعة



I_4	I_3	I_2	I_1
1	0	1	1

Knapsack-item(4, 2, 1)

$W=2+1+2 \rightarrow W=5$ & $V=37$

وأيضاً ممكن أن نوجد قيم المصفوفه من القانون هذا

$$V(i,w)=\max\{V_i+V(i-1,w-w_i),V(i-1,w)\}$$

فمثلاً $V(2,4)$

$$V(2,4)=\max\{10+V(2-1,4-1),V(2-1,4)\}$$
$$=\max\{10+V(1,3),V(1,4)\}$$
$$=\max\{10+12,12\}$$
$$=\max\{22,12\}$$
$$=22$$

إذا لم نتمكن إيجاد القيم بطريقه الاوله يمكن ايجادهم بهذا القانون

المحاضرة 9:

Graph

Date:
No:

محاضرة 9 ‏Graph Algorithms.

خوارزمية الرسم البياني

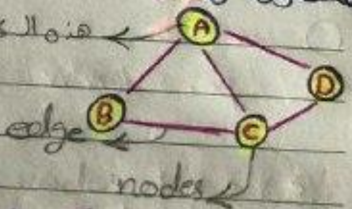
Graph ← abstract data structure

هو عبارة عن بنية بيانات مجردة تمثل مجموعة من العناصر وهذه العناصر تسمى vertices أو nodes ذات علاقة ثنائية بين هذه العناصر

ومثال يوجد مجموعة من items أحياناً هذه العناصر في بعض المسائل محتاج علاقة تربط بينهم حيث تمثل هذه العلاقة أو ربطهم بما يسمى edge(link)

أنها تتكون من ← set of vertices(nodes) مرتبطة بواسطة set of edges(links)

هذه الـ nodes تمثل كحرف أو رقم موضوعة ضمن شكل



ويرمز إليه بـ G(V, E) حيث V ← عبارة عن مجموعة من Vertices.

E ← مجموعة من edges.

V(G) ← يقصد بها nodes لـ Graph

$V(G) = \{A, B, C, D\}$

V → number of n (n=4)

يقصد بها كم node في رسمة

E(G) ← عدد edge في رسمة

$E(G) = \{(AB), (AC), (AD), (BC), (CD)\}$

$E = 5$

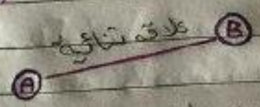
E → number of edges in he Graph

graph
Directed يايجي كده
UnDirected أو كله
مايجيش بعض بعض

$|E| = m$ $|V| = n$

* أنواع Graph

1) Undirected Graph
هذا النوع غير موجه
(AB)=(BA)



هنا يعني مافش سهم يقولي من وين نبدا
ترفع وتجيب بعض دوا اتجاهين أو أكثر من مرة

2) Direct Graph or Digraph
هذا النوع موجه
$AB \neq BA$

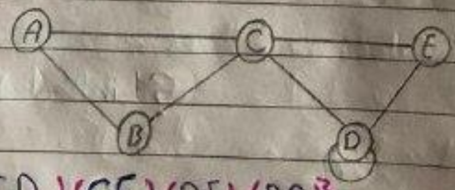
هذا النوع يوجد به سهم يوضح ان نبدأ من A وينتهي في B
يعطي علاقة وحدة فقط تعني لاتجاه واحد

Weighed Graph ← لكل edge وزن مرتبط يمكن أن يشير إلى التكلفه والمسافة والوقت وما إلى ذلك بين 2 nodes متجاورين

ملاحظه ← يمكن ان تكون هذه القيم بسالب في حالات معينه (التكلفه) وتكون مسافه ووقت ← موجبه

Another Example

$V=5$ & $E=7$
$V(G)=\{A,B,C,D,E\}$
$E(G)=\{(AC),(AB),(BC),(CD),(CE),(DE),(DD)\}$



Date:
No:

- كيف يتم تمثيله أو تخزينه في الذاكرة (يوجد طريقتين): :
بما انه abstract لا يوجد تخزين بطريقة مباشرة في الذاكرة لهذا اصبحنا نحتاج آلية لتخزينهم.

① Adjacency Matrix

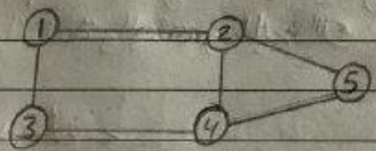
② Adjacency list

- ما المقصود بـ Adjacency ؟
معنى ترجمة هذا المصطلح تعني جار الـ edge إلى هيكون علاقة بين 2 nodes Ⓑ———Ⓐ ← الجار ليه A، الجار لها هي B بمعنى تتصل بهما بشكل مباشر عن طريق الـ edge

1. Adjacency Matrix

تمثيل graph كـ n*n للمصفوفة A
و n تعني عدد الـ nodes ← 5*5 حجم المصفوفة

ex:



(1,2)(1,3)
(2,1)(2,4)(2,5)
(3,1)(3,4)
(4,2)(4,3)(4,5)
(5,2)(5,4)

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	1
3	1	0	0	1	0
4	0	1	1	0	1
5	0	1	0	1	0

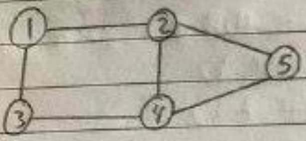
1 ← اذا كانت edge تنتمي إلى الـ E عدد الـ edge
0 ← اذا كانت edge لا ينتمي (بمعنى لا يوجد edge يربط بين الـ nodes)

- ما الفرق بين المصفوفه و Linked list ؟
المصفوفه تخزن Data في الذاكرة بشكل متتالي بمعنى تحجز في مساحة مواقع في الذاكرة للقيم بشكل متتالي (خلف بعض).
بينما Linked list
ليست محتاجه ان تحجز مواقع خلف بعض (متتالية) في الذاكرة.

وهنا ما يميز الـ linked list عن المصفوفة ان ليست بحاجه لمواقع في الذاكرة متتالية

2 Adjacent List

كل رأس $v \in V$ هم بقترن قائمة من Vertices المجاورة لـ v



1 → 2 → 3
2 → 1 → 4 → 5
3 → 1 → 4
4 → 2 → 3 → 5
5 → 2 → 4

المعنى 1 يشير إلى 2 ، و1 يشير إلى 3
2 يشير إلى 1 ، و2 يشير إلى 4 ، و2 يشير إلى 5 وهكذا...

ومن خلال الـ graph نشوف كل قيمه تشير إلى قيمه وبما انه unDirected يأخذ حتى بالعكس لأن لا يوجد اسهم توشر إلى قيم

linked list ← هي الأفضل لأن تعمر مسافه أقل (والاقل تكون الأفضل)

* Graph Terminology

- Degree

يقصد بها عدد الـ edge المتصلة بـ node

ex:

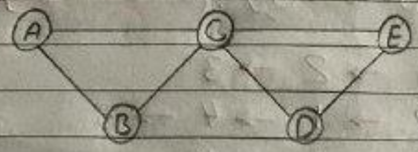
edge(A)=1
edge(B)=2   لأن تشير لـ C و لـ A
edge(C)=1

- Path مسار

المسار عبارة عن سلسلة من Vertices يكون فيها كل رأس مجاوراً للرأس التالي

أو مثلاً هو مسار من نقطة A إلى B وهو جزء من Graph

ex:-



Path A to D

P① ACD

P② ABCD

P③ ACED

P④ ABCED

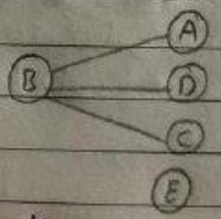
والان لا يمكن تحديد اما اكبر أو اصغر مسار لسه تتواجد القيم.

- Connected

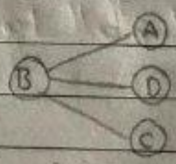
- UnConnected

كل الnodes الموجودة في Graph متصلة ببعضها. هناك node غير متصلة

UnConnected



Connected

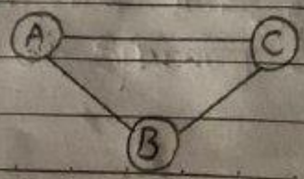


- Cyclic

هو عبارة عن SubGraph بمعنى ممكن تكون جزء من الـGraph أو ممكن تكون Graph نفسه الفكرة ممكن من نقطة البداية تكون مسار ان نقطة البداية فيه هيا النهاية نفسها.

ABC is cycle

الحلقة



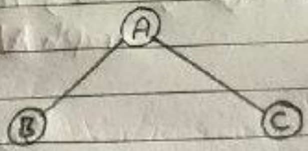
Cycles:

A_B_C_A

C_B_A_C

- Uncycle

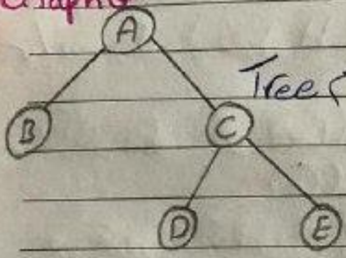
لا توجد حلقة



- Tree

من شروطها: من ضروري تكون UnDirected, Connected
ولا تكون Cyclic

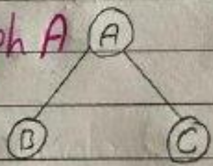
Graph G



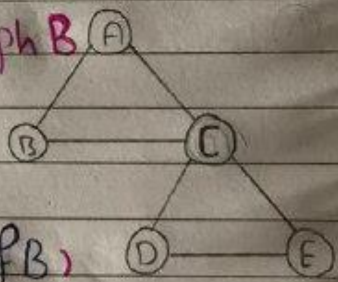
من ضروري وجود هذه الشروط حتى نطلق عليها (Tree)

- Subgraph

ex: ① Graph A

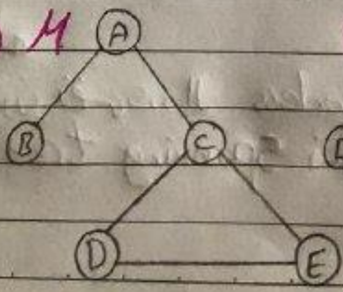


Graph B

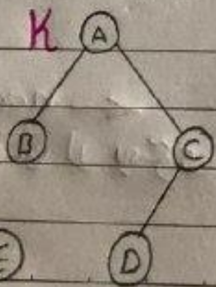


(A is a subgraph of B)

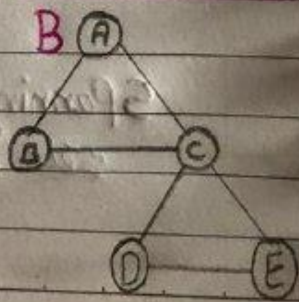
ex: ② M



K

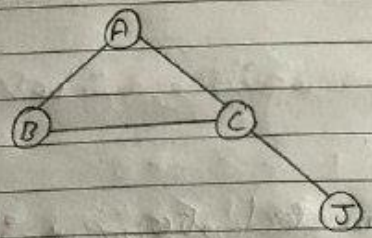


B



K و M جزء من B،

من نفس المثال الـ Graph N



N ليس جزء من B

لأن فيه اسماء nodes الـ J

والموجودة في B اسمها E

لو عندنا

Graph G

س/ متى يمكن أن نقول H جزء من G ؟

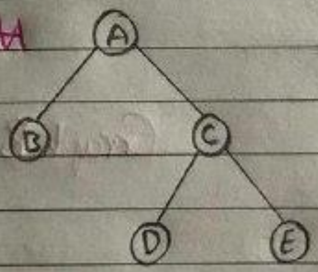
Graph H

اذا كان ← ① $V(H) \subseteq V(G)$

② $E(H) \subseteq E(G)$

من ضروري تحقق الشرطين ↑ حتى نقول H is Subgraph of G

ex:- Graph H



مثال تابع السؤال السابق

$V(H) = V(G)$

$E(H) \subseteq V(G)$

متساويين في nodes وليس متساوين في edge

ملاحظة

عندما يكون نفس عدد الـ nodes ← Spanning

و Spanning tree تعني لا يحتوي على حلقة

المحاضرة 10:

Greedy Method

Kruskal Algorithm

Date:
No:

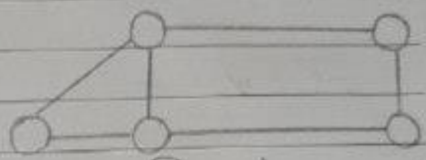
محاضرة ١٠

بداية هذه المحاضرة هي تكملة لمحاضرة ٩ + محاضرة ١٠

1 graph

Set of vertex (node) V

set of edges (links) E



Graph

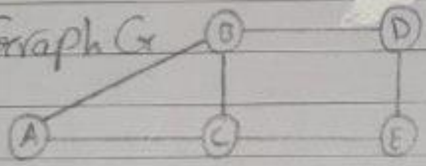
2 Subgraph

H is subgraph of G ← متى تكون

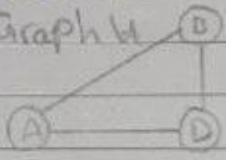
عندما يكون 1. $V(H) \subseteq V(G)$

$E(H) \subseteq E(G)$

Graph G



Graph H

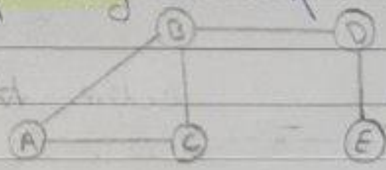


2 $V(H) = V(G)$

$E(H) \subseteq E(G)$

H is Spanning SubGraph G

Graph H



س/ ما الفرق بين Subgraph و Spanning subgraph؟

ج ← جزء من ال Graph (Subgraph)

أما Subgraph Spanning عندما يكون عدد nodes متساوي مع Graph

Spanning Trees

هو عبارة عن شجرة

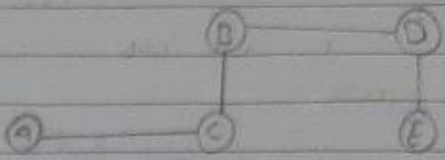
قد يحتوي الـ Graph على العديد من الاشجار الممتدة

شروط: ① E = n - 1 ② لا يحتوي على Cycle ③ كل الـ nodes تكون مرتبطة

Connected Graph

نقول ان

(K is a spanning tree of G)



Graph K

E = 5 - 1 = 4

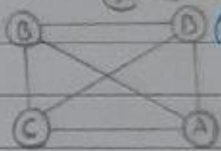
no Cycle

- ماذا يعني عدم وجود Cycle؟ يعني انه عدد الـ edges تكون V-1 يعني اقل من عدد الـ node بـ 1
- Complete Graph: كل الـ nodes المتصلة بشكل مباشر) Connected و (undirect

توضيح

Complete Graph Connected ← G

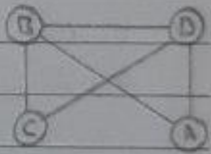
Direct



connected ← الان كل الـ nodes متصلة

ولكن ليس Complete لان الـ C ليست متصلة بـ A node

not Direct

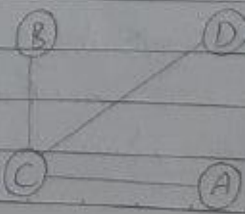


س/ ايهما تحقق شروط Spanning Tree SubGraph؟

V=4 ، E=3 no Cycle

H تحقق شروط

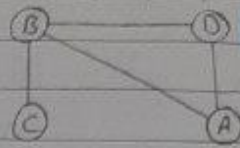
H is Spanning Tree SubGraph of G



H

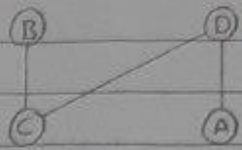
No:

V=4 , E=4 , Cycle
L is Spanning sub graph of G



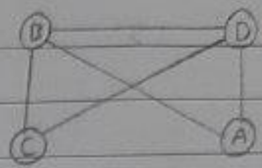
L

V=4 , E=3 , no cycle
M تتحقق شروط
M is Spanning Tree SubGraph of G

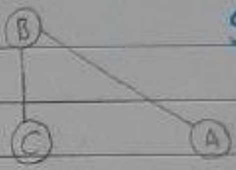


M

V=4 , E=5 , Cycle
K spanning sub graph of G



K

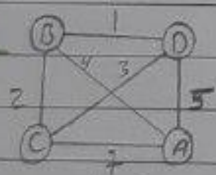


S

S is supgraph of G
هو جزء من ال graph الاصلي

المفروض (الهدف) في اي Graph نريد عليه نريد ان نصل ل Spanning Tree
ليس فقط Spanning Tree نريد الوصول إلى minimum Spanning Tree

Ex:



هنا Graph :
Connected , undirected , weighted

وهذه الأوزان تمثل مسافة زمن... على حسب المثال weighted Graph

ملاحظة: كل Graph من الممكن ان يكون لديه اكثر من Spanning Tree
عندما يكون لدينا weighted Graph المفروض ان نتحصل باستغلال الخوارزميات
اللي صاضوها على minimum Spanning Tree

Date:
No:

مثال: في المواصلات نحن نتوصل لنقطة معينة ونبني شبكة من الطرق المفروض ان نختار أقصر مسار (مسافة أقل) (زمن أقل)

من هنا جاءت minimum Spanning Tree

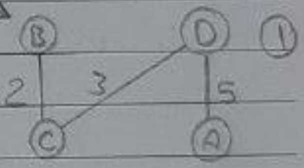
minimum Spanning Tree قد نعرفها لما يكون لدينا الـ Graph به أوزان أو weighted

مثال: لحساب minimum Spanning Tree Graph

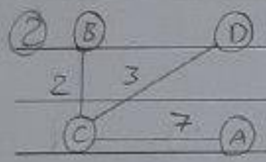
نحسب تكلفة هذا الـ Graph نجمع الأوزان (القيم) :-

Graph G الأصلي

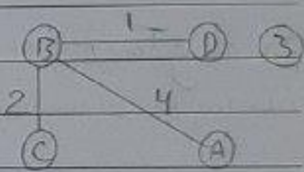
"M" = 2+3+5=10



"H" = 2+3+7=17



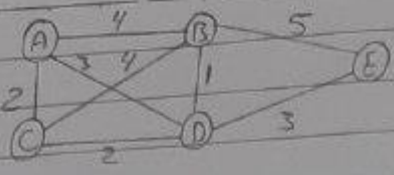
"K" = 2+4+1=7



في الـ Graph الأصلي G هناك أكثر من Spanning Tree وتكلفتها تختلف نحن نبحث على الأقل تكلفة وسنستخدم الخوارزميات التي تحسب أقل تكلفة Minimum Spanning Tree

* محاضرة ١٠

Kruskal algorithm :- يستعمل لحساب Minimum Spanning Tree



Graph G
weighted

يعطيني الخوارزمية الحل "الأفضل"

- إيجاد Minimum Spanning Tree عن طريق هذه الخوارزمية هو مثال لل Greedy Algorithm (Method)

لإيجاد Minimum Spanning Tree نحتاج نستخدم إحدى خوارزميات ال Greedy

★ أول خوارزمية عندنا هي Kruskal algorithm

في المثال (Graph G) كم عندي مسار من A إلى E ؟

A, B, E → 4+5=9
A, D, E → 3+3=6
A, C, D, E → 2+2+3=7
A, B, D, E → 4+1+3=8

الثاني هو أقل مسار

* هذه فكرة حساب المسار الأقل تكلفة

- فكرة عمل Kruskal Algorithm ؟

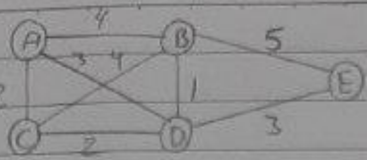
لديه 3 خطوات رئيسية

① أول خطوة هو ترتيب edges ترتيب تصاعدي بناء على تكلفة
② أضف أول edge (الذي هو أقل تكلفة) ليصبح جزء من Tree (T)
T ترمز Minimum Spanning Tree

نقوم بعدها بتكرار هذه الخطوة (إضافة أصغر edge) لل T مع مراعاة شروط Spanning Tree بعدم تكوين حلقة

يتم تجاهل edge الذي يكون حلقة

Graph G



① نبدأ بترتيب edges ترتيب تصاعدي

BD=1 ✓

AC=2 ✓

CD=2 ✓

AD=3 skip → لأنه يصنع حلقة

DE=3 ✓

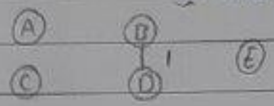
AB=4 skip

BC=4 skip

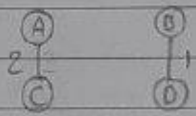
BE=5 skip

نبدأ بأول edge BD ونضعه في T

T={BD

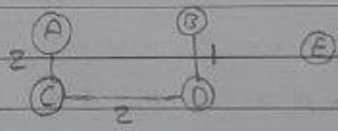


T={BD,AC



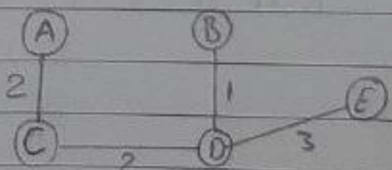
ضروري نضيف edge مع مراعاة شروط Spanning Tree

T={BD,AC,CD



E=V-1 ، No Cycle ، Graph Connected

T={BD,AC,CD,DE}

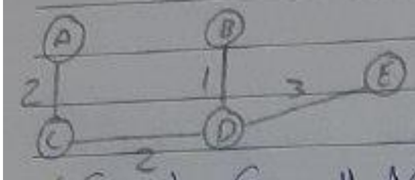


في هذه الخطوة نلاحظ ان كل nodes أصبحت Connected

Date:
No:

بمعنى اي edge يأتي بعدهم يكون حلقه
كما نصل لمرحلة التي بها جميع Nodes متصله نتوقف الان اي edge
بعدها سيكون Cycle

هذه هي فكرة عمل الخوارزمية
في المثال السابق ماكم تساوي تكلفه ؟



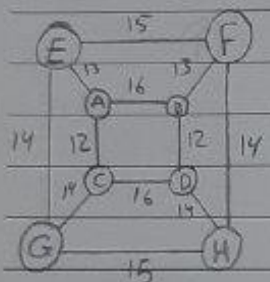
$MST = 1 + 2 + 2 + 3 = 8$

optimal Solution

يمثل (Minimum Spanning Tree للـ Graph G)

اذا كان سؤال امتحان المطلوب الرسمة والتكلفه

مثال آخر



- أولاً ترتيب تصاعدي ←

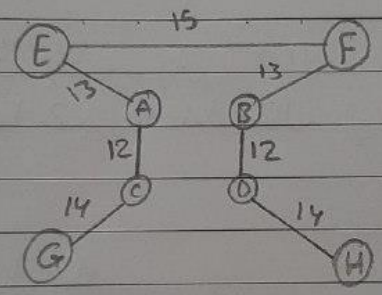
AC = 12 ✓
BD = 12 ✓
AE = 13 ✓
BF = 13 ✓
CG = 14 ✓
DH = 14 ✓
EG = 14 تجاهل
EF = 15 ✓
HG = 15 تجاهل
AB = 16 تجاهل
CD = 16 تجاهل

عند تساوي edge مش مهم أيهما نضعه الأول

- ثانياً نبدأ بأول edge ←

1) T = { AC , BD , AE , BF , CG , DH , EF }

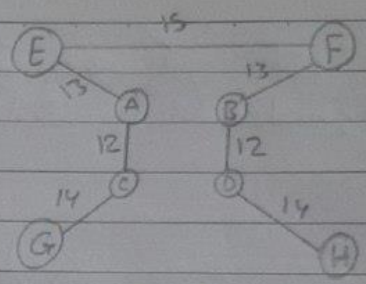
Date:
No:



$MST = 12 + 12 + 13 + 13 + 14 + 14 + 15 = 93$

* لو بنجي نقدر نختار FH بدل DH عادي
وكذلك نقدر نختار GH بدل EF عادي

Date:
No:



$MST = 12 + 12 + 13 + 13 + 14 + 14 + 15 = 93$

* لو بنجي نقدر نختار FH بدل DH عادي
وكذلك نقدر نختار GH بدل EF عادي