

## البرمجة الديناميكية (Dynamic Programming)

**البرمجة الديناميكية (Dynamic Programming):** هي تقنية من تقنيات البرمجة التي تستخدم لحل المشاكل وذلك بجعل حل المشكلة الأساسية يؤول إلى حل مشاكل فرعية (Subproblem)، ثم نحصل على حل المشكلة الأساسية عن طريق مراكمة حلول هذه المشاكل الفرعية. خلافا لتقنية (فرق تسد Divide & Conquer) التي تعمل بكفاءة عندما تكون المسائل الفرعية مستقلة عن بعضها (independent)، فإننا نحتاج إلى البرمجة الديناميكية عندما تكون المشاكل الفرعية متداخلة (Overlapping Subproblem) ولا نستطيع فصلها عن بعضها.

### ملاحظة :

1. إن كل المشاكل التي تستخدم البرمجة الديناميكية من أجل حلها هي مشاكل تحتوي على استدعاء ذاتي بطبيعتها (Recursive Definition). ومعروف أن الاستدعاء الذاتي يستخدم عادة لحل مشكلة اصغر من المشكلة الأساسية.
2. إن التداخل (Overlap) بين المشاكل الفرعية يتسبب في تنفيذ عدد من الحسابات أكثر من مرة، وهذا يرفع تعقيد الخوارزمية بشكل أسّي (Exponential Growth). ولتجنب هذا نلجأ إلى حفظ هذه الحلول (Memoization) داخل مصفوفة (كي لا نقوم بحسابها مرة أخرى) أي نضحي بالقليل من الذاكرة مقابل الكثير الوقت (Space Complexity vs Time Complexity).
3. تستخدم البرمجة الديناميكية لحل كثير من مشاكل الأمثلية (Optimization problems). ولكنها يمكن ان تستخدم لحل مشكلات أخرى .
4. يبدأ العمل بإيجاد حلول المشاكل البدائية ثم الصعود حتى نصل إلى المشكلة الأساسية.

عزيزي القارئ حاول أن تستشعر الأمور السابقة و أنت تقرأ حلول المسائل المطروحة كأمثلة للبرمجة الديناميكية. حاول أن تعرف الاستدعاءات الذاتية التي تحل المشكلة و انظر كيف تحفظ حلول المشكلات الفرعية، وإذا كانت المشكلة تحتوي على امثلية (افضل ، اسوأ، اكبر، أصغر) فانتبه لذلك. وانظر ما هي الحالات الابتدائية.

**تعليق تاريخي :** ان كلمة برمجة (Programming) الموجودة في العبارة (Dynamic Programming) لا تعني كتابة شفرة بل تعني تخطيط (Planning) وقد ظهر مصطلح (Dynamic Programming) في وقت كان فيه الكلمة Programming شائعة بين الاقتصاديين أكثر من أي تخصص آخر وكان الحاسوب حديث العهد آنذاك (1950) وكانت تستخدم بمعنى Planning.

كل الشفرات هنا بلغة السي شارب C#

**مثال: 1 أعداد فيبوناتشي (Fibonacci numbers):** كيف نستخدم البرمجة الديناميكية لتوليد العدد  $n$  من سلسلة فيبوناتشي ؟

معلوم ان السلسلة معرفة كمايلي:  $0, 1, 1, 2, 3, 5, 8, \dots$

$$F_0 = 0, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad n > 1$$

لاحظ ان المشكلة يمكن صياغتها باستخدام الاستدعاء الذاتي، كذلك إن حساب  $F_n$  أمكن تقسيمه إلى مشكلتين اصغر في الحجم هما  $F_{n-1}$  وكذلك  $F_{n-2}$ . لاحظ كذلك أن المشكلتين متداخلتان (overlapping) إذ لا يمكن حساب  $F_{n-1}$  دون حساب  $F_{n-2}$ . ألا ترى أن :

$$F_{n-1} = F_{n-2} + F_{n-3}$$

أي مجموع العددين الذين يسبقانه مباشرة. وهكذا ترى ان  $F_{n-2}$  متداخل مع  $F_{n-1}$  اذا يلزم لايجاد  $F_{n-1}$  كما يلزم لحل المشكلة الاساسية ، ثم الا ترى ان الحالات الابتدائية هي الحد الأول والحد الثاني من سلسلة فيبوناتشي وقيمتها هي 1.

الحل بدون استخدام برمجة ديناميكية هو:

```
static int Fib(int n)
{
    if (n == 0 || n == 1)
        return 1;
    else
        return Fib(n - 1) + Fib(n - 2);
}
```

الحل باستخدام البرمجة الديناميكية:

لاحظ أن تعقيد الخوارزمية السابقة هو  $O(\varphi^n)$  حيث  $\varphi = \frac{1+\sqrt{5}}{2}$  والسبب في ذلك هو أن الكثير من الحدود يعاد حسابها مرات ومرات بينما في الحقيقة، يمكن الوصول إلى الحل بتعقيد خطي. انظر إلى الشفرة التالية:

```
static void Main(string[] args)
{
    const int N = 15;
    int[] a = new int[N];
    int i;
    a[0]=1;
    a[1]=1;
    for (i = 2; i < N; i++)
        a[i] = a[i - 1] + a[i - 2];
    Console.WriteLine(a[N-1]);
}
```

ولا يخفي على أحد أن تعقيد هذه الخوارزمية هو  $O(n)$

**مثال 2: المسار الرتيب (monotonic path) :** لتكن لدينا صالة  $M \times N$ ، ولدينا لاعب يقف في الزاوية العليا اليسرى ويريد الوصول إلى الزاوية اليمنى السفلى، كم عدد المسارات التي يمكن للاعب أن يسير فيها كي يصل إلى هدفه إذا علمت أنه يحق للاعب التحرك إما إلى اليمين أو إلى الأسفل فقط. (إن المسار الذي يمكن السير فيه فقط إلى اليمين وإلى أسفل يطلق عليه مسار رتيب).

|   |   |    |    |     |
|---|---|----|----|-----|
| 0 | 1 | 1  | 1  | 1   |
| 1 | 2 | 3  | 4  | 5   |
| 1 | 3 | 6  | 10 | 15  |
| 1 | 4 | 10 | 20 | 35  |
| 1 | 5 | 15 | 35 | 70  |
| 1 | 6 | 21 | 56 | 126 |

**الحل :** لاحظ الطابع التعاودي (Recursive Character) لهذه المسألة. ليكن  $f_{i,j}$  هو عدد المسارات كي نصل إلى المربع  $a_{i,j}$  وكي يصل اللاعب إلى المربع  $a_{i,j}$  يجب أن يمر إما بالمربع الذي يعلوه أو المربع الذي يقع على يساره أي أن:  $f_{i,j} = f_{i-1,j} + f_{i,j-1}$ . في الشكل إلى اليسار عدد المسارات من اجل  $6 \times 5$ .

كما نر

يمكن تحويل التوضيح السابق إلى شفرة كما يلي:

```
static void Main(string[] args)
{
    int i, j;
    const int N = 3, M = 5;
    int[,] a = new int[N, M];
    a[0, 0]=1;
    //*****
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            if (i == 0 || j == 0)
                a[i, j] = 1;
            else
                a[i, j] = a[i - 1, j] + a[i, j - 1];
            Console.Write("{0} ", a[i, j].ToString());
        }
        Console.WriteLine();
    }
}
```

```

    }
    Console.ReadKey();
}

```

ان اختبار تحقق الشرط  $z=0$  معناه ان المربع أقصى اليسار، ولذا لا يوجد إلا طريق واحد من الأعلى وبالمثل اذا كان  $z=0$  فان المربع واقع في أعلى صف في المصفوفة وهذا يعني وجود طريق وحيد قادم من اليسار. لاحظ أن الحالات الابتدائية هي المربعات الواقعة في أقصى اليسار وفي الصف العلوي.

### ملاحظة :

انتبه للحالات الابتدائية في المفتاح الى حل المسائل دائما وكذلك الاستدعاء الذاتي (Recursion) الموجود بين ثنايا المسألة!! كما سيتضح في المثال التالي:

**مثال 3 : (سقوط الكرة)** لتكن لدينا كرة واقعة فوق سلم مبنى ، يتكون هذا السلم من  $n$  درجة ، عندما تسقط الكرة فانها تسقط الى الدرجة التالية مباشرة او تتجاوز درجة او تتجاوز درجتين و هكذا حتى تصل الأرض. مثلا إذا كانت فوق الدرجة الثامنة ( $n=8$ ) فإنها تسقط فوق السابعة (التالية مباشرة) او السادسة (تتجاوز درجة) او الخامسة (تتجاوز درجتين) ثم توصل طريقها إلى الأرض. السؤال بكم طريقة ممكن ان تصل الكرة الارض؟

الحل: واضح في المسألة السابقة وجود استدعاء ذاتي (Recursion) بين ثنايا للمسألة فما ان تصل الكرة الى الدرجة السابعة أو السادسة أو حتى الخامسة حتى تبدأ نفس العمل فهي قد تسقط فوق الدرجة التالية مباشرة او تتجاوز واحدة او اثنتين. لتكن  $f_n$  عدد الطرق اللازم لسقوط الكرة من الدرجة  $n$  حينها وحسب مبدأ العد (counting principle) يمكن وضع العلاقة التالية:

$$f_n = 1 \cdot f_{n-1} + 1 \cdot f_{n-2} + 1 \cdot f_{n-3}$$

واضح أن الكرة ستقفز الى الدرجة التالية بطريقة واحدة (لهذا ضربنا في واحدة حسب مبدأ العد) ثم تحاول النزول من الدرجة  $n-1$  اي تحل المسألة  $f_{n-1}$  أو (وهذا هو سبب الجمع) ستتجاوز الكرة درجة (و هذا يتم بطريقة واحدة) ثم تحاول النزول من الدرجة  $n-2$  أي تحل المسألة  $f_{n-2}$  أو تتجاوز درجتين وتحاول النزول من الدرجة  $n-3$  اي تحل المسألة  $f_{n-3}$ .

بقي أن نعرف ما هي الحالات الابتدائية لهذه المسألة: عندما  $n=1$  واضح ان الكرة ستسقط بطريقة واحدة اي أن  $f_1 = 1$  واذا كانت  $n=2$  فلما ان تسقط الكرة فوق الدرجة الأولى او تتجاوز درجة لتسقط فوق الأرض مباشرة  $f_2 = 2$  و اذا كانت  $n=3$  فلما تتجاوز درجتين (تهبط مباشرة) او تتجاوز درجة أو تسقط فوق التالية مباشرة  $f_3 = 3$

```

static void Main(string[] args)
{
    int i;
    const int N = 8;
    int[] a = new int[N];
    a[0]=1;a[1]=2;a[2]=3;
    //*****
    for (i = 3; i < N; i++)
        a[i] = a[i - 1] + a[i - 2] + a[i - 3];
    //*****
    for (i = 0; i < N; i++)
        Console.Write("{0} ",a[i]);
    Console.ReadKey();
}

```

**مثال 4 : العداء والمضمار :** يحاول عداء تجاوز مضمار طوله  $N$  من الأمتار و في كل خطوة يقطع من متر واحد إلى  $K$  من الأمتار حيث  $K \leq N$ . السؤال بكم طريقة يمكن للعداء أن يقطع هذا المضمار. هذه المسألة هي تعميم للمسائل السابقة. لاحظ انه من أجل أول  $K$  من الأمتار فان العداء يستطيع قطع المسافة بخطوة واحدة مباشرة أو يتجاوز متر واحد ثم يحل المسألة من أجل  $K-1$  من الأمتار أو يتجاوز مترين ثم يحل المسألة من أجل  $K-2$  الخ اي انه من اجل أول  $i$  ( $i \leq k$ ) من الأمتار لدينا العلاقة التالية (مجموع كل العناصر التي تسبقه والواحد يشير الانتقال المباشرة):

$$a_i = 1 + \sum_{j=1}^{i-1} a_j$$

$$a_1 = 1; a_2 = 1 + a_1; a_3 = 1 + a_1 + a_2; a_k = 1 + \sum_{j=1}^{k-1} a_j$$

ثم من أجل الخطوة  $i$  مابعد ( $i > k$ ) متر لدينا  $a_i$  هو مجموع الـ  $k$  عناصر التي تسبق  $a_i$

$$a_i = \sum_{j=1}^k a_{k-j}$$

فمثلا كي يصل العداء الى المتر رقم  $k+1$  او مثلا  $k+2$  فانه يجب حل المسألة من أجل  $k$  مترا التي تسبق المتر ذو الرقم  $k+1$  ثم حلها من أجل المتر الأخير وهكذا دواليك حتى نصل إلى  $N$  وهناك يكمن الاستدعاء الذاتي ثم لاحظ أن أول متر يمكن قطعه بطريقة واحدة، فقط كما أن أول مترين يمكن اجتيازهما بطريقتين مترا مترا أو مترين مباشرة (وهذه هي الحالات الابتدائية) لأن العداء يستطيع التجاوز مباشرة حتى  $k$  متر.

```
static void Main(string[] args)
{
    int i, j;
    const int N = 15;
    const int K = 4;
    int[] a = new int[N];
    //*****
    a[1]=1;
    for (i = 2; i <= K; i++)
    {
        for (j = 1; j < i; j++)
            a[i] = a[i]+a[j];
        a[i] = a[i] + 1;
    }
    for (i = K+1; i < N; i++)
    {
        a[i] = 0;
        for (j = 1; j <= K; j++)
            a[i] =a[i]+ a[i - j];
    }
    for (i = 1; i < N; i++)
        Console.Write("{0} ", a[i]);
    Console.ReadKey();
}
```

**مثال 5 (محصل الضرائب):** يريد محصل ضرائب ان يقوم بعمله في سوق يتكون  $n \times m$  من المحلات (ضريبة كل محل محددة سلفا) ويقوم بعمله في تحصيل الضرائب، فإذا كان المحل الاول يقع عند أقصى الشمال الغربي ويحق للمحصل ان يسير إما إلى الشرق و إما الى الجنوب. المطلوب مساعدة المحصل كي يصل الى المحل الأخير (أقصى جنوب الغرب) بأكبر قدر من الضرائب المحصلة .

الحل:

ألا تشبه هذه المسألة مسألة اللاعب؟ ألا ترى أننا نبحث عن مسار ترتيب؟ في الحقيقة لن نصل إلى المحل  $a_{ij}$  إلا عن طريق احد المحلين أما عبر  $a_{i-1,j}$  أو عبر  $a_{i,j-1}$  (هل تلاحظ Recursion ) فإذا كان المحصل يريد اكبر كمية من التحصيل فهو سيختار من الطريقين ما يضمن له اكبر تحصيل، إي سيختار  $\max$  من الطريقين. و الآن ألا ترى إن المحلات التي تقع أقصى الغرب (العمود الأول) وأقصى الشمال (الصف الأول فقط) تملك فقط طريقا واحد(حالاتنا الابتدائية).

يوضح المربع الأول إلى اليسار مقدار الضرائب التي يمكن تحصيلها من كل محل بينما المربع الى اليمين يوضح الحل بعد التحصيل ويعطى الحل بالعلاقة التالية:

$$b_{ij} = \max\{b_{i-1,j}, b_{i,j-1}\} + a_{ij}$$

|    |   |   |   |     |
|----|---|---|---|-----|
| 1  | 1 | 1 | 1 | 100 |
| 2  | 1 | 3 | 1 | 1   |
| 4  | 8 | 7 | 3 | 5   |
| 10 | 8 | 5 | 2 | 7   |

|    |    |    |    |     |
|----|----|----|----|-----|
| 1  | 2  | 3  | 4  | 104 |
| 3  | 4  | 7  | 8  | 105 |
| 7  | 15 | 22 | 25 | 110 |
| 17 | 25 | 32 | 34 | 117 |

|    |   |   |     |   |
|----|---|---|-----|---|
| 1  | 1 | 1 | 1   | 3 |
| 2  | 1 | 3 | 1   | 1 |
| 4  | 8 | 7 | 100 | 5 |
| 10 | 8 | 5 | 2   | 7 |

|    |    |    |     |     |
|----|----|----|-----|-----|
| 1  | 2  | 3  | 4   | 7   |
| 3  | 4  | 7  | 8   | 9   |
| 7  | 15 | 22 | 122 | 127 |
| 17 | 25 | 32 | 124 | 134 |

```

static void Main(string[] args)
{
    int i,j;
    const int N = 5;
    const int M = 4;
    int[,] a = new int[N,M];
    int[,] b = new int[N,M];
    Random r = new Random();
    //*****
    for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
            a[i, j] = r.Next (10,99);    // random numbers in matrix a
    b[0,0]=a[0,0]; //first shop
    //*****
    for (i = 1; i < N; i++) //first column
        b[i, 0] = a[i-1,0] + a[i, 0];
    for (j = 1; j < M; j++) // first Row
        b[0, j] = a[0, j - 1] + a[0, j];
    //*****
    for (i = 1; i < N; i++)
        for (j = 1; j < M; j++)
            b[i, j] = max(b[i - 1, j],b[i,j-1]) + a[i, j];
    output(a, N, M);
    output(b, N, M);
    Console.ReadKey();
    //*****
}

static int max(int x, int y)
{
    if (x > y)
        return x;
    return y;
}

static void output(int[,] a, int n, int m)
{
    int i, j;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            Console.Write("{0}\t",a[i,j]);
        Console.WriteLine ();
    }
    Console.WriteLine();
}

```

**مثال 6 الصراف :** دخل زبون الى محل صرافة ليقوم بتبديل عملة من الفئة n (مثلا 100) بفئات اصغر. فاذا كان الصراف يملك الفئات التالية  $(a_1, a_2, a_3, a_5, \dots, a_m)$  المرتبة تصاعديا ( $a_1 = 1$ ). ساعد الصراف في معرفة اقل عدد من الأوراق النقدية اللازمة.

الحل : ليكن لدينا  $c_j$  هو عدد الأوراق المستخدمة لصرف القيمة  $j$  واضح ان تبديل العملة بفئات اصغر يعطي عدد من الأوراق أكثر بمقدار واحد على الأقل (لذلك نزيد بمقدار 1) كما ان حل المسألة باختيار الفئة  $i$  يجعل الحل يؤول الى حل صرف  $-c[j - d_i]$  لذا سيكون الحل بالشكل التالي:

$$c[j] = \begin{cases} \infty & j < 0 \\ 0 & j = 0 \\ 1 + \min\{c[j - d_i]\} & j \geq 1 \end{cases}$$

حيث  $d_i$  قيمة الفئة  $i$  المتوفرة عند الصراف. إذا كان المطلوب صرف ورقة ذات قيمة سالبة فهذا الأمر مستحيل  $\infty$ ، إذا لم يعطنا الزبون أي ورقة فلن يحصل على شيء  $c[0]=0$ ، أما إذا أعطانا ورقة موجبة فسنعطيه أصغر قطع ممكن أن تحقق الغرض ونحل المسألة من أجل بقية المبلغ. انظر هذا الحالة التوضيحي.

لدينا عملات من الفئة (1-5-10-25) عندما  $i=2$  فإن  $d_i = 5$

$$c[0] = 0$$

$$c[1] = \min \begin{cases} 1 + c[1 - 25] = \infty \\ 1 + c[1 - 10] = \infty \\ 1 + c[1 - 5] = \infty \\ 1 + c[1 - 1] = 1 * \end{cases}$$

$$c[2] = \min \begin{cases} 1 + c[2 - 25] = \infty \\ 1 + c[2 - 10] = \infty \\ 1 + c[2 - 5] = \infty \\ 1 + c[2 - 1] = 2 * \end{cases}$$

$$c[3] = \min \begin{cases} 1 + c[3 - 25] = \infty \\ 1 + c[3 - 10] = \infty \\ 1 + c[3 - 5] = \infty \\ 1 + c[3 - 1] = 1 + 2 = 3 * \end{cases}$$

$$c[4] = \min \begin{cases} 1 + c[4 - 25] = \infty \\ 1 + c[4 - 10] = \infty \\ 1 + c[4 - 5] = \infty \\ 1 + c[4 - 1] = 1 + 3 = 4 * \end{cases}$$

$$c[5] = \min \begin{cases} 1 + c[5 - 25] = \infty \\ 1 + c[5 - 10] = \infty \\ 1 + c[5 - 5] = 1 + 0 = 1 * \\ 1 + c[5 - 1] = 1 + 4 \end{cases}$$

سوف نحفظ داخل مصفوفة  $d$  بالفئات المتوفرة لدى الصراف، وسنحتفظ داخل  $c[i]$  بالقيمة التي حسبناها سلفاً. وسنحفظ داخل  $c[i]$  عدد الطرق التي يمكن تمثيل القيمة  $i$  بأقل عدد من القطع المتوفرة. إذا كانت قيمة  $j$  سالبة فإن  $c[j]=\infty$  حسب العلاقة السابقة.

فيما يلي شفرة لإيجاد صرف أي مبلغ من 1 وحتى 40 باستخدام أقل عدد من الأوراق من الفئات (1 - 5 - 10 - 50). تعيد الدالة MinimumsCount مصفوفة تحتوي على القيم التي تقع يمين الحاصرة (انظر إلى الأعلى)، بينما تحسب Minimum أقل قيمة كي تمنح هذه القيمة إلى  $c[j]$  كما أننا وضعنا  $c[0]=0$ .

```
//static int
const int N=40;
static int[] d = new int[] { 1, 5, 10, 50 }; // coins
static int[] c = new int[N]; // save the counted sums
static void Main(string[] args)
{
    c[0] = 0;
    //=====
    int m = 37;
    int j;
    int[] MinimumArray;
    c[0]=0;
    for (j = 1; j < m; j++)
```

```

    {
        MinimumArray = valuesCount(j);
        c[j] = Minimum(MinimumArray);
    }
    for (j = 1; j < m; j++)
        Console.Write("{0}:{1} ", j, c[j]);
    Console.ReadKey();
    //*****
}
static int[] valuesCount(int j)
{
    int i;
    int[] mins = new int[d.Length];
    for (i = 0; i < mins.Length; i++)
        if (j - d[i] < 0)
            mins[i] = 1+1000000; //infinity
        else
            mins[i] = 1 + c[j - d[i]]; //1+0 Or 1+somevalue
    return mins;
}
static int Minimum(int[] a )
{
    int i, min;
    min = a[0];
    for (i = 1; i < a.Length; i++)
        if ( a[i]<min)
            min = a[i];
    return min;
}

```

## تمارين

1. كيف يمكن حل مسألة الصراف بالاستفادة من مسألة حرامي الذهب (Knapsack problem)؟
  2. لديك مصفوفة مربعة  $n \times m$  مكونة من أصفار و وحدات، هات اكبر مصفوفة داخل المصفوفة المعطاة والتي تتكون فقط من وحدات.
  3. ماذا لو عكسنا الوضع في مسألة محصل الضرائب بحيث تصبح دفع غرامات بدل تحصيل ضرائب. كيف تساعد المخالف على دفع اقل غرامة ممكن بخروجه من آخر مكتب؟
  4. مصفوفة تحتوي على مجموعة من الأعداد الموجبة، كيف يمكن ان تقسمها إلى مجموعتين متساويتين بحيث يكون مجموع عناصر الاولى يساوي مجموع عناصر الثاني. (استقد من Knapsack problem )
  5. \* ليكن لدينا مجموعة من المصفوفات  $(M_1, M_2, \dots, M_n)$  . فإذا علمت ان حاصل ضرب مصفوفتين قياساتهما  $i \times r$  and  $r \times j$  يكلف  $i \times r \times j$  فأوجد أفضل ترتيب لعملية الضرب بحيث تكون مجموع التكاليف النهائية اقل ما يمكن. طبعاً مقاييس كل مصفوفة من المصفوفات معطى بالصورة  $(r_1 \times r_2, r_2 \times r_3, \dots, r_n \times r_{n+1})$  من اجل أربعة مصفوفات A,B,C,D لدينا مثلاً الحالات التالية:
- $(A((BC)D)), (AB)(CD), (((AB)C)D), (A(B(CD)))$
- يجب أن يكون الجواب احد الحالات السابقة ، لاحظ أن عمليات الضرب الموجود داخل الأقواس ستنفذ قبل البقية.  $(A(B(CD)))$  مثلاً.
6. في روضة أطفال يقف التلاميذ في طابور الصباح، و تكون هناك احتمال حدوث فوضى اذا وقفت ثلاث فتيات أو أكثر على التوالي ( لا يوجد أي قيود على الغلمان ). فإذا أعطيت طابورا يتكون من  $n$  تلميذا وتلميذة بين احتمال حدوث فوضى في هذا الطابور.
  7. ليكن لديك عدد مكون من  $n$  ( $n \leq 9$ ) خانة (واضح ان الخانة الاول لا تساوي صفر). احسب كمية الأعداد التي لا تحتوي على صفيرين على التوالي