

كنتمة لموضوع البرمجة الديناميكية سأقدم حل مشكلة تقليدية:

مشكلة لص الذهب (Knapsack problem)

دخل لص الى قصر كي يسرق كل ما خف وزنه وغلا ثمنه ولديه حقيبة ظهر سيضع المسروقات داخلها. فإذا كان هناك n أحجامها $s_1, s_2, s_3, \dots, s_n$ قيمة كل منها هي: $v_1, v_2, v_3, \dots, v_n$. ما هي اكبر قيمة (مجموع قيم الأشياء المسروقة) يمكن أن يحصل عليه اللص بحقيبة ظهر حجمها V .

الحل : لاحظ أنه عند حقيبة حجمها 0 مهما كانت الأشياء المسروقة فإن اللص سيحصل على قيمة عظمى مقدارها 0 ، وعندما يكون عدد الأشياء المسروقة 0 فإنه مهما كان حجم الحقيبة فإن اكبر يحصل عليها اللص هي صفر (البند الأول من العلاقة (1)). سنرتب القطع تصاعديا حسب الحجم.

لتكن $A[i, j]$ هي افضل قيمة ممكن الحصول بواسطة أول i قطعة عند حقيبة حجمها j . الآن لاحظ أننا عند كل خطوة سنجيب على السؤال التالي: عند حقيبة حجمها j و عدد من القطع مقداره i أيهما اكبر؟ القيمة التي حققها السارق من التجربة السابقة اي عند حقيبة حجمها j وعدد قطع مقداره $i-1$ أم أن القيمة الناتجة إضافة القطعة i (البند الثالث من العلاقة (1)). *الا تشم رائحة الاستدعاء الذاتي هنا (Recursion).*

واضح ان كون الحجم الحالي هو j معناه إضافة قيمة القطعة i أي v_i إلى العبارة التي تعبر عن ما قبل إضافة هذه القطعة اي الى $A[i-1, j-s_i]$ سينتج عن هذه الإضافة العبارة التالية: $A[i-1, j-s_i] + v_i$ عند نفس الحجم j . لا حظ انه إذا كان الحجم الحالي j اصغر من حجم القطعة i أي اصغر من s_i فانك لن تحصل على أفضل مما حصلت عليه سابقا لان القطعة i لم تدخل ضمن أي حساب وستبقى النتيجة كما هي عليه (البند الثاني من العلاقة (1)). *الا تشم رائحة الاستدعاء الذاتي هنا (Recursion)*

سنضع ما سبق بالعلاقة الرياضية التالية:

$$A[i, j] = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ A[i-1, j] & j < s_i \\ \max(A[i-1, j], A[i-1, j-s_i] + v_i) & j \geq s_i \end{cases} \quad (1)$$

سنحل المثال التالي ثم نكتب شفرة له بعد ذلك:

Size	1	2	5	6	7
Value	1	6	18	22	28

ما هي اكبر قيمة اذا علمت ان حجم الحقيبة هو $V=11$ الحل:

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

واضح انه بإجراء أول قفزة نحصل على 22 وثاني قفزة 18 و من ثم نتوقف وهذه هي الشفرة باستخدام السي شارب C#

```

static void Main(string[] args)
{
    int i, j;
    int V = 11; // size of knapsack
    int[] s = {1,2,5,6,7 }; //sizes
    int[] v = { 1, 6, 18, 22, 28 }; //values
    int[,] A=new int[s.Length+1,V+1];
    //=====
    for (j = 0; j <= V; j++)
        A[0, j] = 0;
    //=====
    for (i = 0; i <=s.Length ; i++)
        A[i, 0] = 0;
    //=====
    for (i = 1; i <= s.Length; i++)
        for (j = 1; j <= V; j++)
            if (j < s[i - 1])
                A[i, j] = A[i - 1, j];
            else
                A[i, j] = max(A[i - 1, j], A[i-1,j - s[i - 1]]+v[i-1]);
    //=====
    for (i = 0; i <= s.Length; i++)
    {
        for (j = 0; j <= V; j++)
            Console.Write("{0} ",A[i, j]);
        Console.WriteLine();
    }
    Console.ReadKey();
}

static int max(int a, int b)
{
    if (a > b)
        return a;
    return b;
}

```

والآن هل تستطيع أيها القارئ الكريم أن تقسم مجموعة من الأعداد الموجبة $v_1, v_2, v_3, \dots, v_n$ الى مجموعتين بحيث يكون مجموعا عناصر المجموعتين متساويين باستخدام الخوارزمية السابقة؟ في الحقيقة يمكن ذلك بحساب مجموع كل الأعداد ($sum = \sum v_i$) ثم قسمة المجموع على 2 ونبحث عن الأعداد تحقق نصف المجموع باستخدام الخوارزمية السابقة (نجعل $V = sum/2$) ولا ننسى ان نجعل مصفوفة القيم تساوي مصفوفة الأحجام كما يلي $s = v = v_1, v_2, \dots, v_n$ اي ان العبارة $A[i - 1, j - s_i] + v_i$ ستحول إلى $A[i - 1, j - v_i] + v_i$.