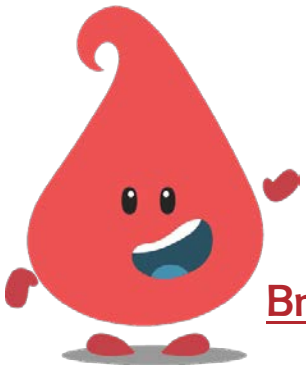
Brute Force

د. أبي صندوق

الخوارزميات

RB Informatics;

15/12/2021



تحدثنا في المحاضرات السابقة عن ثلاثة مفاهيم أساسية وهي:

بنى المعطيات Data Structures: وقلنا أنها كل ما يحجز مكانا في الذاكرة.

الخوارزمية: وهي مجموعة خطوات منتهية تصل إلى حل مشكلة.

طرق التفكير: التي نتبعها للوصول إلى خوارزمية ما، وتحدثنا عن ثلاث طرق تفكير :

Brute Force, Divide & Conquer, Decrease & Conquer

وقلنا أن لكل مسألة يمكن توصيفها معلوماتياً حل Brute Force، لكن ما هي الـ Brute Force؟

▪ Brute Force

هي طريقة يتم فيها توليد جميع التراكيب (الاحتمالات) الممكنة للمسألة والبحث عن التراكيب التي تمثل الحل الصحيح.
إذاً جميع المسائل التي لها عدد منتهي من التراكيب (مهما كان كبير) يمكن لنا توليدها والبحث عن الحل ضمنها، أي يمكننا حلها باستخدام هذه الطريقة.

▪ مثال:

بفرض انه لدينا ثلاثة أعداد a, b, c نريد ترتيبها تصاعدياً فإن الحل من النوع Brute Force هو أن نولد جميع التراكيب الممكنة لهذه الثلاثة أعداد أي:
ثم نبحث عن التركيب الذي تكون الأعداد فيه مرتبة تصاعدياً.

abc	acb
bca	bac
cba	cab

فلو كان لدي الأعداد 7, 5, 9 فإن التراكيب الممكنة هي:

7, 5, 9	7, 9, 5
5, 7, 9	5, 9, 7
9, 5, 7	9, 7, 5

نلاحظ أن التركيب الذي يرتب الأرقام تصاعدياً هو 5, 7, 9

لكن قد يتبادر إلى أذهاننا سؤال: بما أننا نستطيع حل أي مسألة عن طريق الـ Brute Force لماذا تعلمنا طرق التفكير الأخرى؟!

بعض المسائل (أو أغلبها) يستهلك موارد هائلة وقد تكون غير بشرية عندما نريد حلها بطريقة Brute Force.

فمثلاً قد تستغرق ملايين السنوات ومهما حاولنا لن نستطيع تقليل هذا الزمن بشكل كبير فلو استخدمنا معالجين عندئذ بدلاً من ان نحتاج لمئة مليون سنة سنحتاج إلى خمسين مليون سنة ومازال الزمن كبير جداً وغير بشري وليس منطقياً.

وحتى لو حاولنا القيام بتحسينات ستكون من رتبة خطية بينما التعقيد من مرتبة أكبر من خطية (أسية مثلا) لذلك فإن هذه التحسينات لا فائدة ملموسة منها.

لذلك لابد لنا من تعلم طرق أخرى للحل مثل divide, decrease،إلخ...

وفي أسوأ الأحوال إن لم نجد حل من طرق أخرى فلدينا طريقة مضمونة وهي الـ Brute Force.

تمرين:

ليكن لدينا مجموعة من الإعلانات لكل منها مدة معينة، نريد اختيار مجموعة من هذه الإعلانات بحيث تملأ أكبر مدة ممكنة من فاصل إعلاني مدته 90 ثانية فقط.

فلتكن لدينا الإعلانات والمدة التي تستغرقها بالترتيب:

(A, 40), (B, 35), (C, 30), (D, 25), (E, 22), (F, 19)

القواعد:- لا يمكن أن تتجاوز مدة مجموع الإعلانات المختارة 90 ثانية.
- لا يسمح بالتكرار (الإعلان الواحد يعرض مرة واحدة).

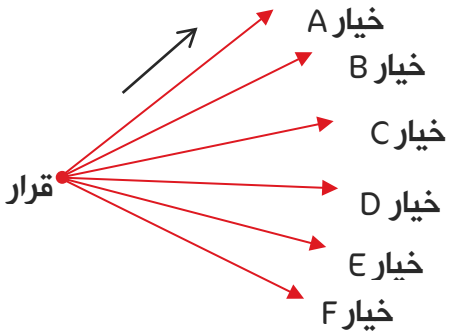
قد نفكر بالعديد من الطرق للحل:

- أخذ الإعلانات من الأقل إلى الأكثر فنكون بذلك قد أخذنا أكبر قدر ممكن من الإعلانات فمثلاً سنأخذ F, E, D وعندها سيكون المجموع 66 فلن نستطيع أخذ أي إعلان أخر لأنه سيتجاوز الـ 90 ثانية المسموحة.
لاحظ أنه حل سريع لكنه لا يعطيني الجواب الأمثل لأن ما أريده هو أكبر قدر ممكن من الثواني على ألا أتجاوز الـ 90.
لذلك فهذا الحل مرفوض ✖

- قد نفكر بأخذ إعلان كبير وآخر صغير أو غير ذلك من الطرق لكن لاحظ أنها جميعاً لن توصلنا للحل الأمثل.
- في الواقع إن الطريقة التي ستعطينا الحل الأمثل قطعاً هي تجريب جميع الحالات الممكنة، لكن كيف نقوم بذلك؟
بما أنه لدينا 6 إعلانات فيجب علينا أن نتبع منهجية معينة للمرور عليها واحداً تلو الآخر.
لدينا طريقتان:

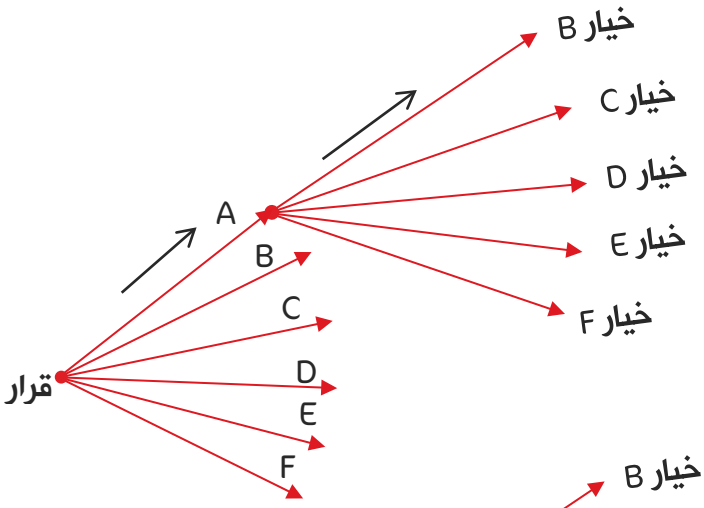
الطريقة الأولى:

نحن سنعرض عدد ما من الإعلانات، إذاً سيكون لدينا ما يسمى بالإعلان الأول ثم الإعلان الثاني وهكذا...
حتى أقرر أن إعلان ما هو الإعلان الأول لدينا **ستة خيارات** يجب أن نتخذ **قراراً واحداً** منهم، لتقرير أي خيار هو الأفضل يجب علينا تجريب جميع هذه الخيارات، إذاً يجب أن نجرب أحد هذه الخيارات كقرار واحداً تلو الآخر، ونرى ما النتائج المرتبة.
فمن أجل الإعلان الأول لدينا **ستة خيارات** نريد أخذ **قرار** منهم:

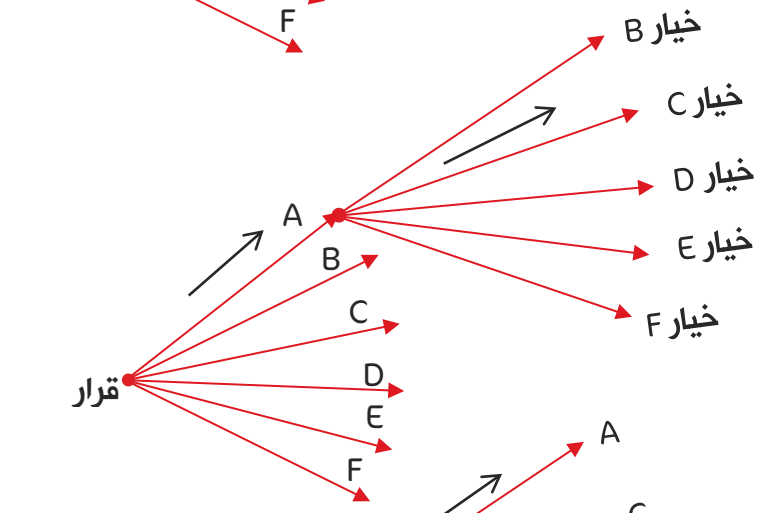


بفرض أننا اتخذنا A **كقرار** مثلاً:

عندئذ سننتقل إلى الإعلان الثاني لدينا **خمسة خيارات** نريد آخذ **قرار واحد** منهم



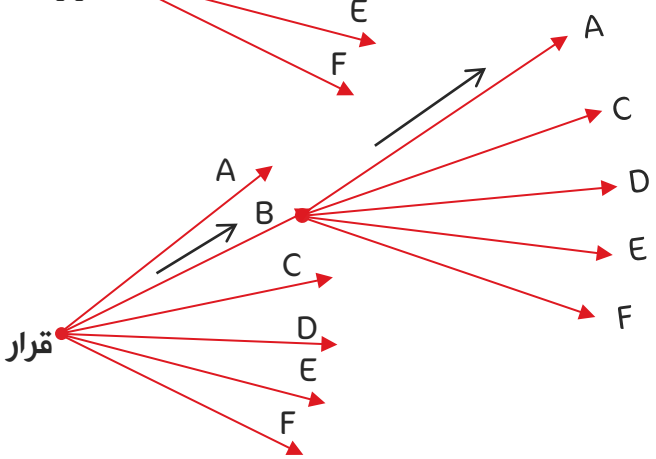
بفرض أننا أخذنا B **كقرار**:
ونكون قد اخترنا حتى الآن **قراراين**: $\{A, B\}$ مجموعهما 75 أي لم نعد نستطيع آخذ أي إعلان آخر إذاً فقد وصلنا إلى أحدى الحالات الممكنة وهي
$(\{A, B\}, 75)$
فلنتراجع عن آخر **قرار** آخذناه ونجرب أخذ **خيار** آخر.



مثلا لنأخذ C **كقرار**
الآن وصلنا إلى نقطة أخذنا **قرارين** قبلها حتى وصلنا إليها {A, C} من ثم لدينا **أربعة خيارات** نريد اتخاذ **قرار** منها

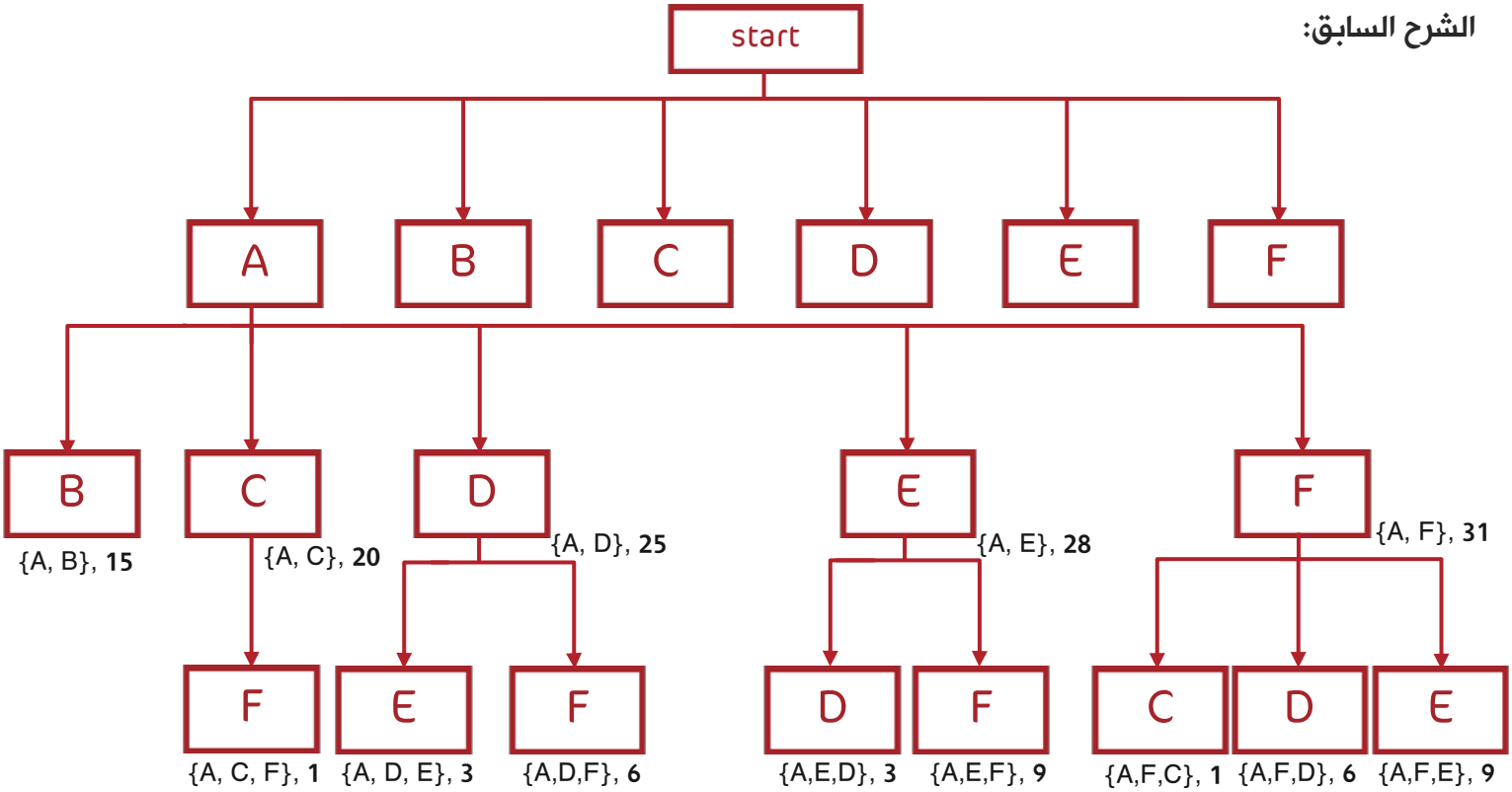
⋮

وهكذا



حتى نستعرض جميع النتائج الممكنة إذا أخذنا **القرار** A كأول إعلان ثم **سنتراجع** عن **القرار A** ولنأخذ **القرار B**، فنصبح أمام **خمسة خيارات** نريد الاختيار منها إذا أخذنا A **كقرار** عندها سنكون أخذنا القرارين {B, A} بمجموع 75 ولم يعد بإمكاننا أخذ المزيد من الإعلانات فنكون وصلنا إلى أحد الحالات الممكنة وهي $(\{B, A\}, 75)$ والآن **سنتراجع** عن **القرار** A ونجرب **خيار** أخر ⋯ وهكذا حتى نولد جميع الحالات الممكنة.

لاحظ أننا في كل نقطة يهمنا النقاط التي أخذناها من قبل في هذا المسار وكم الوقت المتبقي، والآن سنرسم شجرة تمثل الشرح السابق:



لاحظ أننا لاختيار الإعلان الأول لدينا 6 خيارات لتجربتها ومن أجل الإعلان الثاني لدينا 5 خيارات وهكذا...

إذا فسيتولد لدينا على أقصى حد $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 6! = 720$

لكن هناك تكرار في الحالات مثلاً: $(\{B, A\}, 75), (\{A, B\}, 75)$ وغيرها الكثير.

ونلاحظ أنه بما أننا لا نهتم بالترتيب فكل ما يهمني هو هل سيتم عرض الإعلان أم لا، فلا يهمني مكانه، لذلك بدلاً من التفكير ما الإعلانات التي يمكننا عرضها أولاً وماذا سيحصل إن عرضناها؟

يمكننا التفكير: هل سنعرض هذا الإعلان وماذا سيحصل إن عرضته؟

الطريقة الثانية:

1) الآن سنتبع أسلوباً جديداً: من أجل الإعلان A لدينا خياران:
   a) إما أن نعرض الإعلان A ويتبقى لدي 50 ثانية إذا أخذته كقرار عندئذ:
      i) إما أعرض الإعلان B ويتبقى لدي 15 ثانية إذا أخذته كقرار عندئذ:

نكون قد وصلنا إلى حالة لا نستطيع عرض المزيد من الإعلانات فسنخزن هذا المسار ({A, B}, 15)

      ii) أو سأتراجع عن القرار وأجرب الخيار الأخر وهو عدم عرض B ويتبقى لدي 50 ثانية إذا أخذته كقرار:
   (a) إما أن أعرض الإعلان C ويتبقى لدي 20 ثانية إذا أخذته كقرار:
      1. لاحظ لا أستطيع عرض D أو E لذلك إما أن أعرض الإعلان F ويتبقى لدي ثانية واحدة عندئذ:

نكون قد وصلنا إلى حالة لا نستطيع عرض المزيد من الإعلانات فسنخزن النتيجة 1,{A, C, F}.

      2. أو سأتراجع عن القرار وأجرب الخيار الأخر وهو ألا أعرض F ويتبقى لدي 20 ثانية عندئذ:

نكون قد وصلنا إلى حالة مسدودة وهي 20 ,{A,C} ومن ثم أتراجع عن القرار وآخذ الخيار الاخر

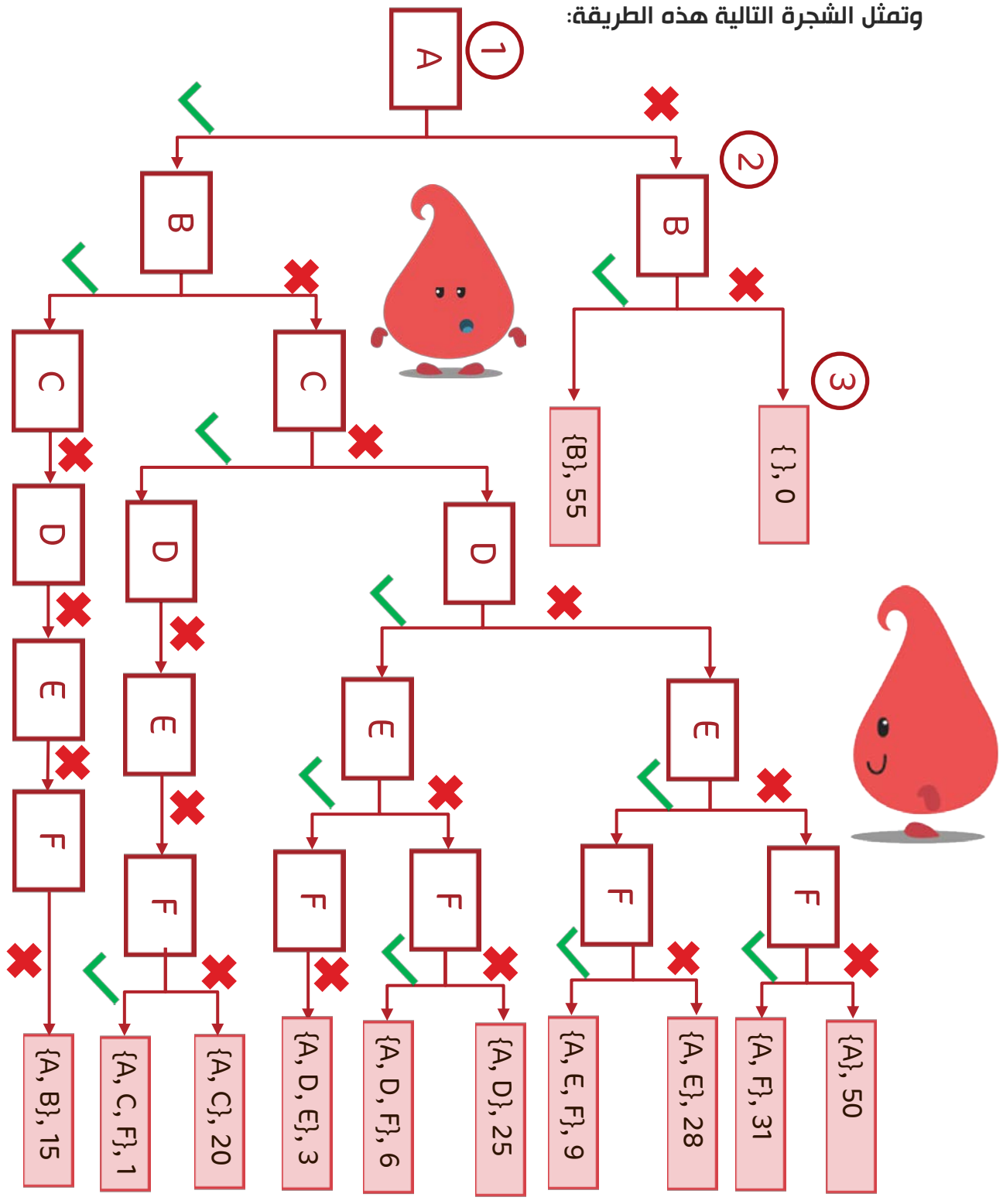
(b) أو سأتراجع عن القرار وأجرب الخيار الأخر وهو ألا أعرض الإعلان C ويتبقى 50 ثانية عندئذ:

i. إما أن أعرض الإعلان Dويتبقى 25 ثانية ومن ثم أقوم بالاختيار ومن ثم التراجع وهكذا...

b) أو سأتراجع عن القرار وأجرب الخيار الأخر وهو ألا أعرض A ويتبقى لدينا 90 ثانية عندئذ:

i) إما أن أعرض الإعلان B ويتبقى 55 ثانية عندئذ: سنحتاج للقيام بالخيارات من جديد والتراجع عند الوصول لحالة مسدودة....

وتمثل الشجرة التالية هذه الطريقة:



لو أكملنا الشجرة فسنجد أن الحل الأمثل هو ({B,C,D},0)

الفرق بين 1 و 2 و 3: لاحظ أنها جميعاً تحوي نفس قيم الزمن المتبقي لكن

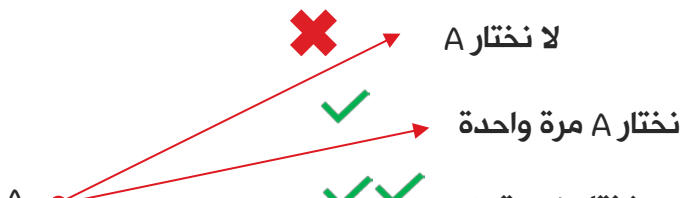
في 1: لم نتخذ أي قرار بعد ← يمكننا لاحقا أخذ A أو B أو كليهما

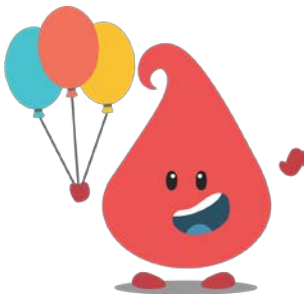
في 2: فنحن اتخذنا قرار بعدم أخذ A ← يمكننا لاحقاً أخذ B لكن لن يمكنني أخذ A أبداً

في 3: اتخذنا قرار بعدم أخذ B ← لا يمكنني أبداً أخذ A أو B

هذه الشجرة تمثل طريقة ممنهجة لتوليد جميع الحالات الممكنة، إن تعقيد هذه الطريقة هو عدد التفريعات الممكنة في عقدة مرفوعاً لقوة هي العمق أي هنا: $2^{depth} = 2^6$

- لنفرض الآن أنه بإمكاننا تكرار الإعلان مرتين فقط(أي يمكن عرضه مرة أو مرتين أو لا نعرضه أبداً) عندئذ سيصبح شكل العقدة هو:





فيصبح التعقيد 3^n

إن كان التكرار مسموحاً عندئذ سيكون بعض الإعلانات لا يتسع الوقت إلا لعرضها مرة واحدة فسيكون عدد الأفرع عندها فرعين فقط (مثلاً لو كان لدينا إعلان مدته 50 ثانية فإننا إما سنعرضه أو لا نعرضه فلا نستطيع عرضه مرتين لأنه سيستغرق 100 ثانية >90 فيتجاوز الحد).

وبعض الإعلانات قد نتمكن من عرضها عدد كبير من المرات (مثلاً لو كان لدينا إعلان مدته 10 ثواني فأمامنا عشرة خيارات)

فسيكون لدينا حد أدنى للتعقيد (أقل عدد تفريعات) وحد أعلى للتعقيد (أكبر عدد تفريعات) يفيدنا هذان الحدان بالمقارنة مع خوارزميات أخرى.

ففي مثالنا: لا يمكننا تكرار الـ A أكثر من مرتين فالحد الأدنى للتعقيد $3^n = 3^6$، ولا يمكننا تكرار الـ F أكثر من 4 مرات فالحد الأعلى للتعقيد هو $5^n = 5^6$ فالتعقيد هنا محصور بين $3^n, 5^n$.

▪ القص:

تتمثل هذه الفكرة في انه إذا ضمنا أن هذا الطريق لن يعيد لنا حلاً أفضل من الحل الذي لدينا إذا لماذا نسلكه؟؟!

انتبه إذا ضمنت أي بنسبة 100%

في مثالنا السابق بعد أن تركنا A, B, C وصلنا إلى حالة هي لو أخذنا D, E, F فإن أعلى مجموع سنحصل عليه هو 66.

وإن جربنا أن نترك أحدها فسنحصل على مجموع أقل إذاً فأفضل ناتج قد يأتي من هذا الفرع هو 66 ونحن أساساً لدينا ناتج أفضل منه إذا لم أجرب هذه الفرع وأفحصه.

تسمى هذه العملية بالقص.

لاحظ أن هذه العملية لن تقلل التعقيد بشكل عام (إلا إذا كانت عملية القص متقنة جداً وذات كفاءة عالية جداً) لكنها ستوفر علينا عدد لا بأس منه من العمليات.

تمثيل الحلول بالـ Brute Force

دائماً سنتمكن من تمثيل الحل الـ Brute Force بشجرة مثل الشجرة السابقة.

لكن يتبقى السؤال ماهي الخوارزمية التي سأتمكن من التجول على هذه الشجرة من خلالها؟ أي ما هي الخوارزمية التي ستمكننا من اتخاذ القرار ثم التراجع عنه؟

إنها العودية ☺

لأن معنى التراجع هو أن اتخذ قرار (سيتم تخزينه في الـ stack)، ثم التراجع عن هذا القرار والعودة إلى الحالة قبل اتخاذ هذا القرار (حذف الحالة من الـ stack)

الشكل العام لتابع التراجعية العودي

الآن سنفكر بطريقة لكتابة الشكل العام لتابع التراجعية العودي:

- يمكنني تسمية التابع أي اسم نريد بفرض أنه Try ويمكنني تمرير أي باراميترات نحتاجها.
- عند أي عقدة كنت أقف فأنا أريد أن أجرب أخذ جميع الخيارات المتاحة لكن قبل أخذ هذا الخيار يجب علي فحص هل يمكنني أخذ هذا الخيار أم لا أستطيع في حال كنت أستطيع اخذ هذا الخيار سآخذه وأحوله إلى قرار أي سأستدعي هذا التابع مع تغيير على البارامترات بحيث تدل على أنني أخذت هذا الخيار وحولته إلى قرار، ثم بعد أخذ القرار يجب أن أتراجع عنه لأجرب غيره من الخيارات، فستصبح لدي مجموعة الخطوات التالية من أجل كل خيار.
- المقصود بالتراجع هو أننا قد نكون غيرنا قيم بعض المتغيرات من أجل الاستدعاء، لذلك يجب علينا إعادة تلك القيم كما كانت قبل الاستدعاء، أو غير ذلك من التغييرات التي يجب علينا التراجع عنها، وفي بعض الأحيان قد لا نحتاج إلى هذه الخطوة.

```
if (is_acceptable)
{
   Try( );
   Backtrack( );
}
```

```
إذا (أستطيع أخذ القرار)
}
استدعي التابع من أجل هذا الخيار؛
تراجع عن هذا الخيار؛
{
```

سنأخذ كل نتيجة يعيدها لنا الخيار الذي استدعينا من أجله ونقارنه مع أفضل نتيجة حتى الآن ونحتفظ بالنتيجة الأفضل بينهما.

لا تنسى أن أهم شيء في التابع العودي هو شرط التوقف وهنا هو عندما لا يتبقى لدي خيارات.

```
Try (parameters)
{
    if(noMoreChoices)
    {
        //some statement
        return;
    }
    if(choice1 is acceptable)
    {
        x = Try(choice1);
        if(x is optimal)
        {
            save(x);
        }
        backtrack();
    }
    if(choice2 is acceptable)
    {
        x = Try(choice2)
        if(x is optimal)
        {
            save(x);
          }
        backtrack();
    }
}
```

فيصبح القالب العام من الشكل:

ملاحظة:

في حال كان لدينا عدد غير محدد من الخيارات نضع هذه الـ block في حلقة for.

الآن نطبق هذا القالب على مسألتنا:

لدينا خيارين إما أن نأخذ أو نترك أي سيكون لدين حالتين:

1. Pick: شرط أخذ الإعلان الحالي هو أن يكون (عدد ثواني الإعلانات التي أخذتها مضافاً إليها عدد ثواني الإعلان الحالي) أصغر من عدد الثواني الكلي عندئذ يمكنني أخذ هذا الإعلان.

ماذا يعني أن نأخذ الإعلان الحالي؟

يعني أن نضيف مدة هذا الإعلان إلى مدة الإعلانات التي أخذناها وأن ننتقل للإعلان التالي.

2. Leave: لاحظ أنه يمكننا أن لا نضع أي شرط لأننا لم نختر الإعلان، وبالتالي عدد الثواني الحالي يبقى نفسه وهو أصغر من عدد الثواني الكلي.
لكن كما ذكرنا يمكننا أن نقص بعض الفروع وذلك من خلال فحص لو لم نأخذ الخيار الحالي واخذنا كل ما يليه هل سيكون الناتج أكبر من أفضل حل لدينا؟
إذا كان ذلك ممكن فسنجرب ألا نأخذ الخيار الحالي.
أما إن كان مجموع كل ما يلي مع ما أخذناه أصغر من أفضل حل لدينا فإن هذا الطريق لن يعود علينا بحل أفضل لذلك لن نأخذه.

- شرط التوقف:

عندما لا يتبقى لدينا خيارات أي عندما يصبح رقم الإعلان الذي أقف عنده الآن i أكبر من عدد الإعلانات الكلي n عندئذ سأقارن القيمة التي لدي مع أفضل ناتج وأقرر هل أخزنه أم لا.

- تحديد البارامترات التي سنمررها:

لاحظ أننا احتجنا إلى:

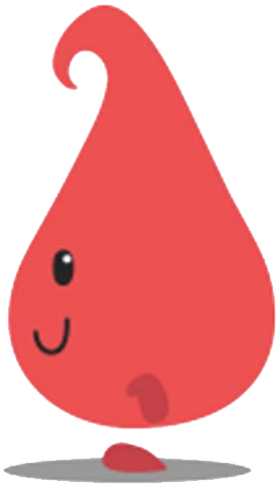
- العنصر الذي أقف عنده الآن i ← قيمته الأولية عند أول استدعاء (0)
- عدد العناصر الكلي n
- مجموع أزمنة العناصر الذي أخذتها حتى الآن $curr$ ← قيمته الابتدائية (0)
- الحد الزمني الذي لا يجب تجاوزه limit
- أزمنة الإعلانات المتبقية $possible$ ← قيمته الابتدائية مجموع عناصر $x[n]$
- وسنعرف كـ global:
  - أزمنة الإعلانات (لاحظ أقصى عدد للإعلانات 90 إعلان) $x[90]$
  - أفضل نتيجة optimal

```
int x[90];
int optimal;
void Try(int n, int Limit, int i, int curr, int possible)
{
   if(i == n)
   {
      if(curr>optimal)
         optimal = curr;
      return;
   }
   //pick:
   if(curr + x[i] <= Limit)
   {
      Try(n, Limit, i+1, curr+x[i], possible-x[i]);
   }
   //Leave:
   if(curr + possible - x[i] > optimal)
   {
      Try(n, limit, i+1, curr, possible - x[i]);
   }
}
```

يمكن إضافة قواعد أخرى في أمثلة أخرى مثلاً:

- لا يمكن عرض الإعلان A مع الإعلان B
- يسمح بتكرار الإعلان مرتين على الأكثر

كما أنه لو كان الترتيب مهم لوجب علينا اتباع الطريقة الأولى



نسمي هذه الطريقة بالتراجعية Back Track

وجدنا في التابع السابق أنَّه في حال تأكدت من أن مسار ما لن يأتي بنتيجة أفضل من النتيجة الحالية فلن آخذه، صحيح أنَّ التعقيد سيبقى أسيًّا لكن تقليل عدد من العمليات أفضل من تنفيذها دون فائدة، لكن انتبه أنَّ هذا القرار سيتخذ في حال تأكدت 100٪ أن المسار لن يأتي بنتيجة أفضل من النتيجة الحالية لذلك تبقى الخوارزمية لدي صحيحة وأستطيع أن أبرهن أن الحل الذي ستأتي به عند انتهاء التنفيذ هو أفضل حل ممكن (الحل الأمثل).

تخيل أنك في نقطة ما وصلت إلى حل قريب من الحل الأمثل بنسبة تصل إلى 99٪ بعد مسح (المرور على) نصف الشجرة (أنت تعرف أن أمثل حل ممكن هو 90 وهو الحل الذي تبحث عنه) وأنت هنا أمام أحد خيارين إما أن تقوم بمسح نصف الشجرة الأخر مقابل أن تتأكد 100٪ أنك أتيت بالحل الأمثل أو أن تقف هنا وتقبل بحل قريب من الحل الأمثل مقابل توفير نصف الزمن،

فماذا ستختار؟

في الواقع هنا يجب أن ننظر إلى أمرين قبل أن نختار:

- ما مدى قرب الحل الذي لدينا الآن من الحل الأمثل؟
- ما هو الذي سأصرفه في حال قررت البحث عن الحل الأمثل؟

إذا كان الزمن الذي سأوفره كبير نسبياً (ستوفر 90٪ من الزمن الكلي مثلاً) مقابل الحصول على حل قريب بنسبة 99٪ بالمئة من الحل الأمثل فلا بأس، فكما وجدنا في المثال السابق استطعنا بمسح النصف الأول من الشجر أن نحصل على حل 89 ثانية ولكن بعد مسح النصف الثاني وصلنا إلى 90 ثانية وهو الحل الأمثل، كان من الممكن أن نقبل بهذا الحل (89 ثانية) مقابل توفير نصف الزمن.

بطريقة أخرى لنفرض أنك في نقطة ما استطعت معرفة أن الحل الأمثل سيظهر من اتباع الطريق الأول بنسبة 99٪ ومن الطريق الأخر بنسبة 1٪ في هذه الحالة إما أن تجرب الطريقتين وتضمن حصولك على حل أمثل أو أن تتخذ الطريق صاحب الاحتمال الأعلى لكن في هذه الحالة لن تضمن حصولك على حل أمثل إنما ستضمن حصولك على حل قريب منه.

فماذا لو كنت عند كل عقدة استطعت تحديد الفرع صاحب الاحتمال العالي جداً والفرع صاحب الاحتمال المنخفض جداً؟ في حالة كهذه تستطيع ان تصل إلى حل قريب من الحل الأمثل (وقد تصل إلى الحل الأمثل عن طريق الصدفة) لكن سنناقش عدة حالات هنا:

- في حال كنا نستطيع عند كل عقدة أن نأتي بقرار مضمون 100٪ أي طريق يجب أن نسلك فهذه خوارزمية تستطيع أن نبرهن أنها ستأتي بالحل الأمثل فهي لا تتعامل مع احتمالات بل مع قرارات مضمونة تسمى هذه الطريقة greedy.
- في حال كنت عند كل عقدة تحسب احتمال الأفرع فتجد أن أحد الطرق احتماله مرتفع جداً والفرع الأخر منخفض جداً فهنا أنت تضحي بالحل الأمثل مقابل توفير الوقت لكن ستحصل على حل قريب منه (يفي بالغرض). طريقة تسريع الخوارزمية وتثقيل الأفرع لمعرفة الفرع صاحب الاحتمال الأعلى وما إلى ذلك سنتعلمه لاحقاً بتفصيل أكبر في مقرر خوارزميات البحث الذكية.
- إذا كان احتمال أحد الأفرع ليس مرتفعاً بالدرجة الكافية هنا يجب أن تجرب الأفرع كلها لأنك لم تحصل على احتمال يجعلك تختار طريق وتهمل الأخر.

يمكننا توليد شجرة فضاء الحلول (فضاء البحث) Search Space أو المرور عليها بطريقتين:

1. بالعمق أولاً Depth First (DFS):

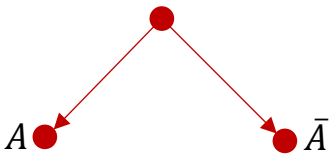
وهي أن نأخذ الفرع ونولد جميع الاحتمالات الممكنة منه وأن نسير على هذا الفرع حتى نهايته ثم الانتقال إلى الفرع الآخر، وهذا ما اعتمدناه في مسألتنا السابقة، وهو يعتمد على الـ stack في عمله.

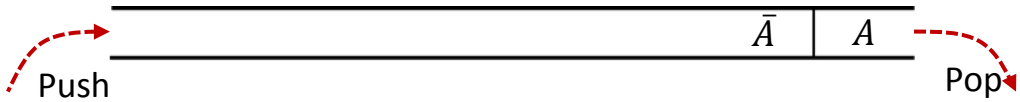
2. بالعرض أولاً Breadth First (BFS):

وهي تعتمد أن نسير في الأفرع معاً على التوازي، وتعتمد على الـ queue في عملها.

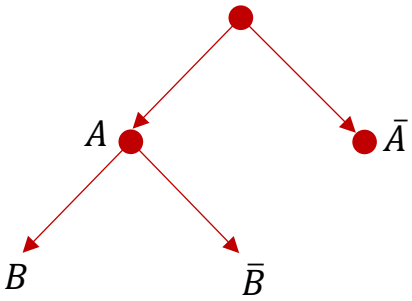
فمثلاً إن أردنا حل مثالنا السابق اعتماداً عليها:

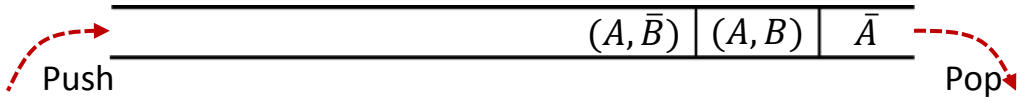
سنولد جميع الخيارات المتعلقة بـ A ونضيفها إلى الـ queue وهي $\{A, \bar{A}\}$:



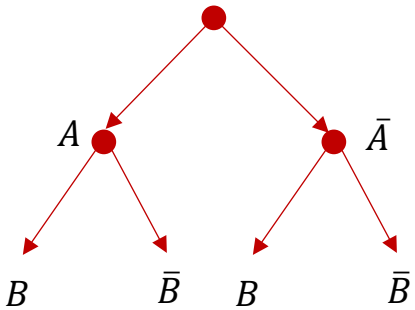


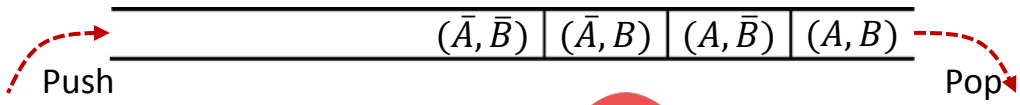
نحذف أول عنصر من الـ queue ونولد كل ما يتعلق به، أي نولد $\{(A, B), (A, \bar{B})\}$





نحذف أول عنصر من الـ queue ونولد كل ما يتعلق به، أي نولد $\{(\bar{A}, B), (\bar{A}, \bar{B})\}$

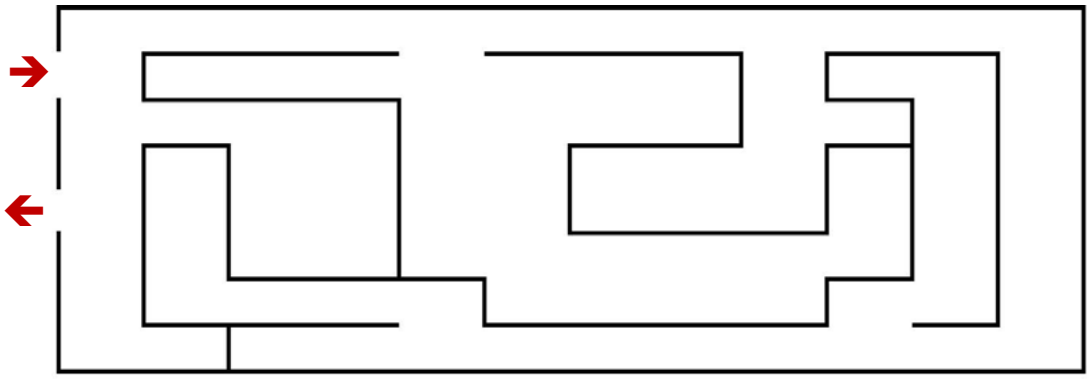




ونتابع على هذا المنوال.

نستخدم هذه الطريقة في مسائل مثل: حل متاهة، التجول في بيئة، أقصر طريق ...

فمثلاً لو أنني أريد حل المتاهة التالية:



تخيل أن نسلك الطريق العلوي أولاً ونجرب جميع الخيارات الممكنة في هذا المسار حتى نهايته DFS فكم سنضيع من الوقت عندئذ دون أن نصل إلى نتيجة.

بينما لو اتبعنا طريقة BFS فجربنا أن نسير في كل طريق خطوة ونرى ماذا سيحدث، ثم نجرب خطوة أخرى في كل فرع لدينا، عندئذ بعد بضعة تفريعات سأكون وصلت إلى المخرج، وفي أسوأ الأحوال سنضطر إلى أن نكمل الشجرة كاملة فيما لو كان الطريقين متساويين.

رغم ذلك يبقى للطريقتين DFS وBFS نفس التعقيد.

سنتوسع في هاتين الطريقتين وغيرها من الأمور في المقرر القادم الخوارزميات وبنى المعطيات 2.

والى هنا نصل معاً إلى نهاية مقرر الخوارزميات، كان مليئاً بالمعلومات المفيدة والمعززة لتفكيرنا البرمجي، والذي تعلمنا فيه بعض أفضل الحلول للمشاكل البرمجية التي قد تواجهنا مستقبلاً

نتمنى لكم امتحانات موفقة

... See you next Semester ...