

Fuerza Bruta

Algorítmicas

de Técnicas

Análisis

- ▶ La técnica de FUERZA BRUTA es la que se utiliza para resolver un problema siguiendo la estrategia más obvia de solución.
- ▶ Generalmente suele comportar más operaciones de las necesarias para resolver el problema, por lo suelen tener un coste bastante alto.
- ▶ Por el contrario, suelen ser bastante sencillos de implementar.
- ▶ Suelen utilizarse para problemas de tamaños pequeños.
- ▶ Algunos ejemplos de algoritmos por fuerza bruta son:
 - El algoritmo de búsqueda lineal.
 - El algoritmo de ordenación por el método de inserción.

Ejemplo Búsqueda Secuencial

Algorítmicas

de Técnicas

Análisis

- ▶ buscar un número en un vector de enteros de tamaño n :

```
int buscar(int a[n], int x){  
    for (int i=0; i<n; i++){  
        if (a[i]==x) return i;  
    }  
    return -1;  
}
```

- ▶ El algoritmo utiliza la fuerza bruta para buscar el número mirando uno por uno todos los elementos del vector, incluso si el vector estuviese ordenado.
- ▶ El coste de este algoritmo es $O(n)$.
- ▶ Aunque no es la forma óptima de buscar un elemento en un vector ordenado, cuando el tamaño del vector es pequeño resulta bastante rápido.

Ejemplo Ordenación por Inserción

Algorítmicas

de Técnicas

Análisis

algoritmo de ordenación por Inserción

```
void ordenacion_insercion(int a[],int n){
    int aux;
    for(int i=1;i<n;i++){
        aux=a[i];
        for(j=i-1;j>=0;j--){
            if(aux>array[j]){
                array[j+1]=aux;
                break;
            }
            else array[j+1]=array[j];
        }
        if(j==-1) array[0]=aux;
    }
}
```

Ejemplo Ordenación por Inserción

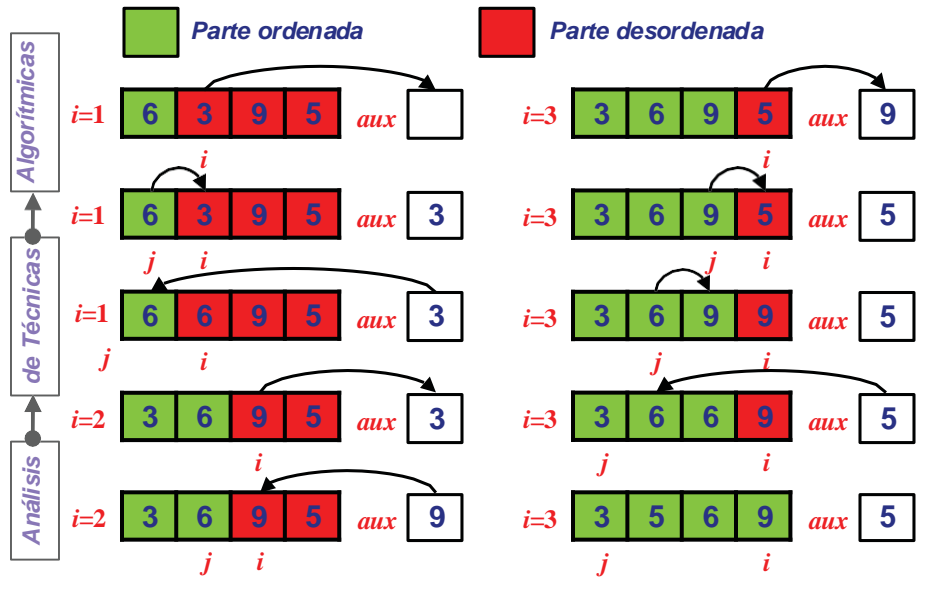
Algorítmicas

de Técnicas

Análisis

- ▶ Este algoritmo divide el vector en dos partes, una con elementos ya ordenados y la otra con elementos todavía sin ordenar.
- ▶ Inicialmente la parte ordenada está vacía y la que no está ordenada contiene todos los números.
- ▶ El algoritmo utiliza la fuerza bruta para ir tomando uno por uno los elementos de la parte ordenada y los va insertado en la parte ordenada en el lugar que les corresponde.
- ▶ Resulta sencillo comprobar que este algoritmo tiene un coste temporal $O(n^2)$.

Ejemplo de Ordenación por Inserción



Divide y Vencerás

- ▶ La técnica de DIVIDE Y VENCERÁS consiste básicamente en descomponer el problema inicial en varios subproblemas del mismo tipo pero más sencillos, resolver los subproblemas y combinar sus soluciones para obtener la solución del problema inicial.
- ▶ A su vez, para resolver cada uno de los subproblemas puede volver a aplicarse la técnica de divide y vencerás recursivamente para descomponerlo en otros subproblemas de menor complejidad, y así hasta que los subproblemas obtenidos tengan una solución trivial.

Algorítmicas

de Técnicas

Análisis

Algoritmo Genérico de Divide y Vencerás

Algorítmicas

de Técnicas

Análisis

```
funcion divide_venceras(n)
  si  $n < n_0$ 
    s=resolver(n)
  si_no
     $[n_1, \dots, n_k] = \text{descomponer}(n)$ 
    desde i=1 hasta k hacer  $s_i = \text{divide\_venceras}(n_i)$ 
    s=combinar( $s_1, \dots, s_k$ )
  fin_si
  devolver s
fin_funcion
```

n es el tamaño del problema,

n_0 es el umbral de tamaño por debajo del cual el problema es trivial
 n_i son los tamaños de los subproblemas en que se descompone el problema inicial.

s_i son las soluciones de los subproblemas de tamaño n_i

s es la solución del problema inicial.

Condiciones Necesarias para Aplicar Divide y Vencerás

Algorítmicas

de Técnicas

Análisis

- Los problemas resolubles mediante la técnica de divide y vencerás tienen que cumplir una serie de condiciones:
 - Es necesario que el problema admita una formulación recursiva.
 - El problema inicial debe poder resolverse mediante la combinación de las soluciones de los subproblemas.
 - Los subproblemas deben ser del mismo tipo que el problema inicial pero con un tamaño estrictamente menor.

Condiciones Deseables para Aplicar Divide y Vencerás

Algorítmicas
de Técnicas
Análisis

- ▶ Además de las condiciones anteriores, si queremos que el algoritmo sea eficiente, debemos exigir las condiciones:
 - El tamaño de los subproblemas debe ser lo más parecido posible y debe decrecer en progresión geométrica, es decir, debe ser n/c con $c > 0$ una constante.
 - Nunca se debe resolver un mismo subproblema más de una vez
 - Las operaciones de descomponer y de combinar deben costar poco.
 - Hay que evitar generar nuevas llamadas cuando el tamaño de los subproblemas es suficientemente pequeño. Por tanto, hay que elegir bien el umbral n_0 .

Coste del Algoritmo Divide y Vencerás

Algorítmicas
de Técnicas
Análisis

- ▶ Suponiendo que la función de descomposición origina k subproblemas de tamaño n/c , el coste del algoritmo divide y vencerás es la suma de
 - El tiempo que tarda en descomponer el problema inicial
$$t_d(n)$$
 - El tiempo que tarda en resolver los k subproblemas en que se divide el problema original
$$kt(n/c)$$
 - El tiempo que tarda en combinar las soluciones de los subproblemas
$$t_c(n)$$

es decir

$$t(n) = kt(n/c) + t_d(n) + t_c(n)$$

Coste del Algoritmo Divide y Venceras

Algorítmicas

de Técnicas

Análisis

- ▶ Si la descomposición y la combinación no son excesivamente costosas, tendrán a lo sumo un coste polinómico, es decir

$$t(n) = kt(n/c) + O(n^i)$$

- ▶ Por tanto la solución a esta ecuación recursiva dependerá de k , c e i :
 - Si $k < c^i$ entonces $t(n) \in O(n^i)$.
 - Si $k = c^i$ entonces $t(n) \in O(n^i \log n)$.
 - Si $k > c^i$ entonces $t(n) \in O(n^{\log_c k})$.
- ▶ Así pues, la utilización de divide y vencerás no garantiza nada acerca de la eficiencia del algoritmo obtenido. Puede suceder que el coste empeore, se mantenga o mejore respecto al de un algoritmo iterativo para el mismo problema.

Ejemplos de Divide y Vencerás

Algorítmicas

de Técnicas

Análisis

- ▶ Algunos algoritmos conocidos que utilizan la técnica de divide y vencerás son:
 - El algoritmo de Karatsuba y Ofman para la multiplicación de enteros grande.
 - El algoritmo de Strassen para el producto de matrices.
 - El algoritmo de búsqueda binaria.
 - El algoritmo de ordenación rápida Quicksort.

Ejemplo Búsqueda Binaria

Algorítmicas

de Técnicas

Análisis

búsqueda binaria recursiva de un elemento en un vector ordenado

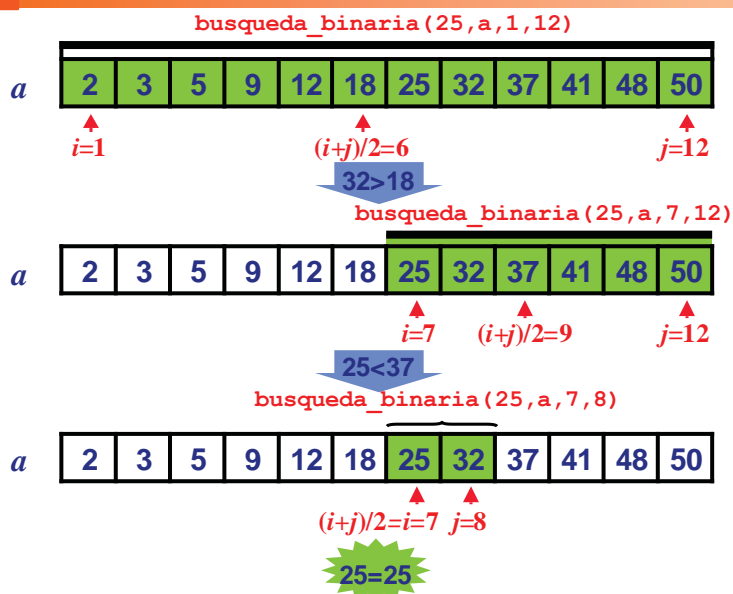
```
int busqueda_binaria(int x,int a[],int i,int j){  
    int k;  
    if (i>j) return -1;  
    else{  
        k=(i+j)/2;  
        if (a[k]==x) return k;  
        else  
            if(a[k]>x) return busqueda_binaria(x,a,i,k-1);  
            else return busqueda_binaria(x,a,k+1,j);  
    }  
}
```

Ejemplo de Búsqueda Binaria

Algorítmicas

de Técnicas

Análisis



Coste de la Búsqueda Binaria

Algorítmicas

de Técnicas

Análisis

- ▶ En la búsqueda binaria, el problema principal se descompone en 1 subproblema de tamaño $n/2$. Por otro lado, el coste de la descomposición y la combinación de soluciones es constante, así que el coste total es

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

Resolviendo la ecuación recursiva tenemos

$$\begin{aligned} t(n) &= t(n/2) + O(1) = t(n/4) + O(1) + O(1) = \dots \\ &= t(n/2^{\log_2 n}) + \log_2 n O(1) = t(1) + \log_2 n O(1) = \\ &= 1 + \log_2 n O(1) \in O(\log n) \end{aligned}$$

- ▶ Por tanto, el algoritmo de búsqueda binaria es más eficiente que el de búsqueda lineal.

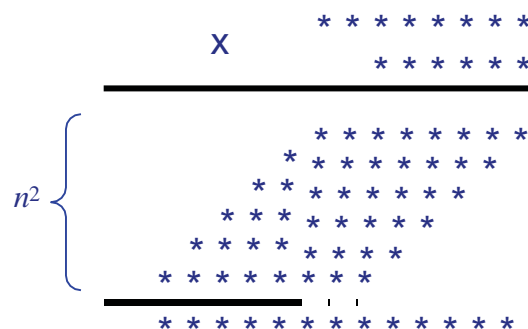
Ejemplo Multiplicación de Enteros Grandes

Algorítmicas

de Técnicas

Análisis

- ▶ Todos conocemos el algoritmo para multiplicar enteros que nos enseñaron en el colegio



- ▶ El coste de este algoritmo es $O(n^2)$.
- ▶ ¿Se pueden multiplicar dos enteros con un coste menor?

Multiplicación de Enteros Grandes mediante Divide y Vencerás

Algorítmicas

de Técnicas

Análisis

- Supongamos que queremos multiplicar dos enteros x e y de tamaño n expresados en una base i .

$$\begin{array}{c}
 \text{\textit{n} dígitos} \\
 \begin{array}{cc}
 x & \begin{array}{|c|c|} \hline a & b \\ \hline \end{array} & x = ai^{n/2} + b \\
 y & \begin{array}{|c|c|} \hline c & d \\ \hline \end{array} & y = ci^{n/2} + d \\
 & \begin{array}{cc} n/2 & n/2 \end{array}
 \end{array}
 \end{array}$$

- Se comprueba fácilmente que

$$xy = ac i^n + (ad+bc) i^{n/2} + bd$$

y de esta forma, se puede calcular xy mediante 4 productos entre enteros de tamaño $n/2$.

- El coste que se obtiene es $t(n) = 4t(n/2) + O(n)$

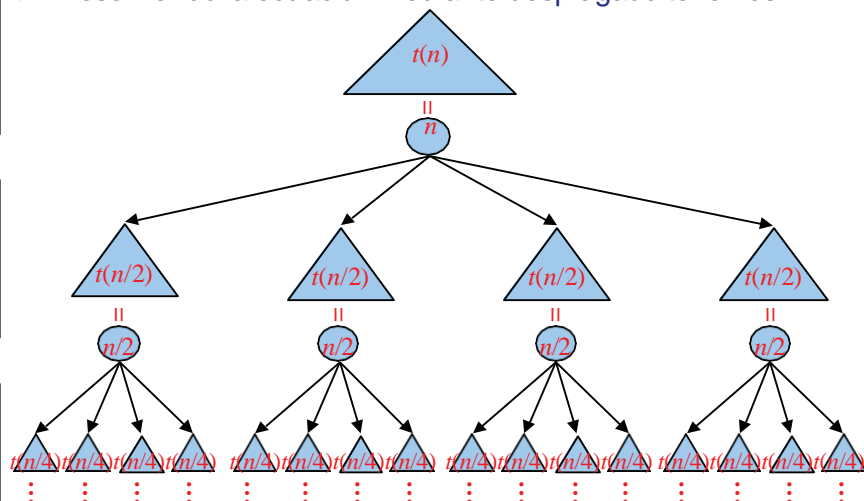
Calculo del Coste de la Multiplicación de Enteros Grandes

Algorítmicas

de Técnicas

Análisis

- Resolviendo la ecuación mediante desplegado tenemos



Mejora del Algoritmo de Multiplicación de Enteros Grandes

Algorítmicas

de Técnicas

Análisis

- ▶ Así pues, tenemos un coste total:

$$O(n^2)$$

de modo que este algoritmo, a pesar de aplicar la técnica de divide y vencerás no es más eficiente que el anterior.

- ▶ ¿Se puede construir un algoritmo mediante divide y vencerás más eficiente que el anterior?
- ▶ Karatsuba y Ofman desarrollaron un algoritmo más eficiente a partir de la misma descomposición anterior pero basándose en la propiedad

$$xy = ac \, i^n + ((a-b)(d-c) + ac + bd) \, i^{n/2} + bd$$

- ▶ De este modo, sólo se realizan 3 productos de enteros de tamaño $n/2$.
- ▶ El coste de este algoritmo es $t(n) = 3t(n/2) + O(n) \in O(n^{1.57})$

Multiplicación de Enteros Grandes (Karatsuba y Ofman)

Algorítmicas

de Técnicas

Análisis

- Suponiendo que x e y son enteros de n dígitos, el algoritmo de Karatsuba y Ofman puede escribirse así

```
funcion productoK&O(enterogrande x,enterogrande y)
si n=1 devolver xy
si_no
  (a,b)=descomponer(x)
  (c,d)=descomponer(y)
  ac=productoK&O(a,c)
  bd=productoK&O(b,d)
  abdc=productoK&O(a-b,d-c)
  devolver ac*i^n + (abdc+ac+bd)*i^n/2 + bd
fin_si
fin_funcion
```

- Para que el algoritmo sea más eficiente debe fijarse el umbral en $n_0=500$ dígitos.

Algoritmos Voraces

Algorítmicas

de Técnicas

Análisis

- La técnica VORAZ consiste en resolver un problema de manera gradual tomando decisiones o realizando acciones que permiten ir construyendo progresivamente la solución del problema.
- Con cada decisión tomada se obtiene una solución parcial del problema y se reduce la dimensionalidad del mismo.
- Suele utilizarse en problemas de optimización.

Condiciones Necesarias para Aplicar la Técnica Voraz

Algorítmicas

de Técnicas

Análisis

- ▶ Debe existir una función que hay que maximizar o minimizar conocida como FUNCIÓN OBJETIVO.
- ▶ La función objetivo depende de varias variables que toman valores en un conjunto conocido como DOMINIO.
- ▶ Existe una función, conocida como FUNCIÓN SOLUCIÓN, que permite saber si unos determinados valores de las variables son o no solución del problema. La asignación de un valor a una variable se denomina DECISIÓN.
- ▶ Existe un conjunto de restricciones que deben satisfacer los valores de las variables de la función objetivo y existe una función para saber si las decisiones tomadas hasta el momento satisfacen o no las restricciones. Esta función se suele llamar FACTIBLE, y el conjunto de decisiones factibles tomadas hasta el momento se llama SOLUCIÓN EN CURSO.

Funcionamiento de la Técnica Voraz

Algorítmicas

de Técnicas

Análisis

- ▶ La técnica voraz intenta resolver el problema fijando progresivamente los valores de las variables de la función objetivo, sin violar las restricciones, hasta llegar al máximo o mínimo de función.
- ▶ En cada paso se determina el valor de una de las variables aún no fijada mediante una FUNCIÓN DE SELECCIÓN. En cada instante, la función de selección siempre toma la decisión factible que es localmente óptima, es decir, la que hace que la función objetivo tome el mejor valor posible hasta el momento, sin intentar averiguar si forma parte de la solución óptima global.
- ▶ Una de las características que diferencian a la técnica voraz de otras que estudiaremos después es que ***¡nunca reconsidera una decisión tomada!***

Algoritmo Genérico Voraz

Algorítmicas

de Técnicas

Análisis

- Suponiendo que el dominio de las variables de la función objetivo fuese C , podemos expresar el algoritmo voraz así:

```
funcion voraz(conjunto_candidatos c)
    s=∅
    mientras (no solucion(s) y no vacio(c))
        x=seleccion(c)
        C=C-{x}
        si factible(s U {x}) entonces s=s U {x}
    fin_mientras
    si solucion(s) entonces devolver s
    si_no devolver ∅
fin_funcion
```

Coste Temporal del Algoritmo Voraz

Algorítmicas

de Técnicas

Análisis

- El coste de un algoritmo voraz depende de dos factores:
- El número de iteraciones del bucle, que depende del tamaño de la solución construida y del tamaño del conjunto de candidatos.
 - El coste de las funciones **seleccion** y **factible**.
La función **factible** suele tener un coste constante, pero la función **seleccion** ha de explorar el conjunto de candidatos y depende de su tamaño.

Ventajas e Inconvenientes de los Algoritmos Voraces

Algorítmicas

de Técnicas

Análisis

- ▶ La principal ventaja de los algoritmos voraces es que suelen ser bastante eficientes.
- ▶ Sin embargo, puesto que siempre toman decisiones localmente óptimas y no reconsideran nunca las decisiones tomadas, no garantizan que se obtenga la solución óptima del problema, ni siquiera que se obtenga una solución, aún cuando dicha solución exista.
- ▶ Esto es debido a que las decisiones localmente óptimas no garantizan la obtención de la solución óptima global.
- ▶ Los problemas resolubles mediante la técnica voraz, deben cumplir el **principio de optimalidad**:

“Una secuencia óptima de decisiones, toda subsecuencia ha de ser también óptima.”

Ejemplos de Algoritmos Voraces

Algorítmicas

de Técnicas

Análisis

- ▶ Algunos problemas conocidos que pueden resolverse mediante la técnica voraz son:
 - El problema del cambio en monedas.
 - El problema de los códigos de Huffman.
 - El problema de los tiempos de espera mínimos.
 - El problema de la mochila.
 - El problema de los caminos mínimos en grafos.

El Problema del Cambio de Monedas

Algorítmicas

de Técnicas

Análisis

- ▶ Dado un conjunto C de monedas de n tipos, se trata de encontrar el número mínimo de monedas que sumen una cantidad x .
- ▶ Este problema satisface el principio de optimalidad, ya que puede demostrarse que si se tiene una solución óptima $S=\{e_1, \dots, e_k\}$ para el problema de sumar una cantidad x , entonces, $S-\{e_i\}$ también es la solución óptima para sumar la cantidad $x-e_i$.



El Problema del Cambio de Monedas Elementos del Algoritmo Voraz

Algorítmicas

de Técnicas

Análisis

- ▶ Función objetivo: El número de monedas, que hay que minimizar.
- ▶ Dominio o conjunto de candidatos: El conjunto C formado por todas las monedas de que disponemos.
- ▶ Función solución: $S=\{e_1, \dots, e_k\}$ es solución si $\sum e_i = x$
- ▶ Función factible: $S=\{e_1, \dots, e_k\}$ es factible si $\sum e_i \leq x$
- ▶ Función selección: Elegir si es posible la moneda de mayor valor de C .

El Problema del Cambio de Monedas Algoritmo Voraz

Algorítmicas

de Técnicas

Análisis

```
int cambio(int x,int valor[n],int solucion[n]){
for (int i=0;i<n;i++) solucion[i]=0;
int i=0, suma=0;
while (suma<x && i<n)
    if (suma+valor[i]<=x){
        solucion[i]++;
        suma+=valor[i];
    }
    else
        i++;
if (suma==x) return 1;
else{
    for (int i=0;i<n;i++) solucion[i]=0;
    return 0;}
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

El Problema del Cambio de Monedas Optimalidad

Algorítmicas

de Técnicas

Análisis

- ▶ El algoritmo anterior no siempre proporciona la solución óptima.
- ▶ Ejemplo. Si tenemos 5 monedas de valores $\text{valor}[5]=\{50,25,20,5,1\}$ y queremos sumar $x=42$, el algoritmo nos daría la solución $s[5]=\{0,1,0,3,2\}$, es decir, 6 monedas, mientras que la solución óptima es $s[5]=\{0,0,2,0,2\}$, que son 4 monedas.
- ▶ Además, en caso de que C no contenga la moneda unidad, ni siquiera se garantiza que el problema tenga la solución.
- ▶ Para que el algoritmo alcance la solución óptima es necesario que C esté formado por tipos de monedas que sean potencia de un tipo básico t (por ejemplo, $C=\{125,25,5,1\}$). En este caso, el problema se reduce a encontrar la descomposición de x en base t , que es única y mínima

El Problema del Cambio de Monedas Coste Temporal

Algorítmicas

de Técnicas

Análisis

- ▶ Para que la función de selección de la función **cambio** funcione adecuadamente, el vector de valores de las monedas debe estar ordenado en orden decreciente.
- ▶ El coste de este algoritmo es de $O(\max(n \log n, m))$ donde n es el número de tipos de monedas de C , y m es el número de iteraciones que realiza el bucle.

El Problema de la Mochila

Algorítmicas

de Técnicas

Análisis

- ▶ Se dispone de una colección de n objetos o_1, \dots, o_n , cada uno de ellos con un peso p_i y un valor v_i asociados.
- ▶ Por otro lado se tiene una mochila capaz de soportar un peso máximo p_{max} .
- ▶ El problema consiste en maximizar el valor de los objetos que se introduzcan en la mochila sin sobrepasar p_{max} .
- ▶ Dependiendo de si los objetos se pueden fraccionar o no, existen dos variantes del problema:
 - **Mochila fraccionada:** Los objetos se pueden fraccionar, es decir se puede colocar en la mochila trozos de objetos.
 - **Mochila entera:** Los objetos no se pueden fraccionar y por tanto deben ir enteros en la mochila.
- ▶ Ambas versiones satisfacen el principio de optimalidad pero sólo el primero puede resolverse con la técnica voraz.

Problema de la Mochila Fraccionada Elementos del Algoritmo Voraz

Algorítmicas

de Técnicas

Análisis

- ▶ Puesto que los objetos se pueden fraccionar, llamaremos x_i a la fracción del objeto o_i que colocamos en la mochila.
- ▶ Función objetivo: El valor de los objetos introducidos en la mochila $\sum_i x_i v_i$, que hay que maximizar.
- ▶ Dominio o conjunto de candidatos: La colección de objetos C que queremos introducir en la mochila.
- ▶ Función solución: $S=\{x_1, \dots, x_k\}$ es solución si $\sum_i x_i p_i = p_{max}$
- ▶ Función factible: $S=\{x_1, \dots, x_k\}$ es factible si $\sum_i x_i p_i \leq p_{max}$
- ▶ Función selección: Existen varias estrategias posibles
 - Seleccionar los objetos en orden decreciente de valor.
 - Seleccionar los objetos en orden creciente de peso.
 - Seleccionar los objetos por orden decreciente de relación valor/peso. (Sólo esta conduce a la solución óptima).

Problema de la Mochila Fraccionada Algoritmo Voraz

Algorítmicas

de Técnicas

Análisis

```
float mochila(float v[n],float p[n],float x[n],float
pmax){
    struct objeto{
        float coste;
        int posicion;
    };
    objeto vp[n];
    for (int i=0;i<n;i++){
        vp[i].coste=v[i]/p[i];
        vp[i].posicion=i;
        x[i]=0;}
    ordenar_decreciente_coste(vp)
    float peso=0, valor=0;
    int i=0, j;
```

Problema de la Mochila Fraccionada Algoritmo Voraz (continuación)

Algorítmicas

de Técnicas

Análisis

```
while (peso<=pmax && i<n){
    j=vp[i].posicion;
    si (peso+p[j]<=pmax) {
        x[j]=1;
        valor+=v[j];
        peso+=p[j];
    } else {
        x[j]=(pmax-peso)/p[j];
        valor+=v[j]*x[j];
        peso=pmax;
    }
    i++;
}
return valor;
}
```

Problema de la Mochila Fraccionada Coste Temporal

Algorítmicas

de Técnicas

Análisis

- ▶ El tamaño del problema es el número de objetos n .
- ▶ El primer bucle realiza n iteraciones y el coste de su cuerpo es constante, de modo que el coste total del bucle es $O(n)$.
- ▶ El coste de la función de ordenación decreciente de los objetos por valor/peso es el coste del algoritmo de ordenación utilizado (en el mejor caso $O(n \log n)$).
- ▶ El segundo bucle realiza, a lo sumo n iteraciones, y el coste de su cuerpo es constante, de manera que el coste total del bucle también es $O(n)$.
- ▶ Por tanto, el coste total del algoritmo es
$$O(n) + O(n \log n) + O(n) = O(n \log n).$$

Programación Dinámica

Algorítmicas

de Técnicas

Análisis

- ▶ Con la técnica de divide y vencerás, la descomposición natural de un problema puede originar subproblemas que se solapen, de manera que al resolverlos independientemente se acabe resolviendo el mismo problema múltiples veces.
- ▶ La técnica de PROGRAMACIÓN DINÁMICA es similar a la de divide y vencerás, sólo que los subproblemas obtenidos sólo se resuelven una vez. Esto exige guardar la solución de cada subproblema para, si se presenta de nuevo, no tener que volver a resolverlo.
¡Se sacrifica espacio para ganar tiempo de ejecución!
- ▶ Es una técnica ascendente, que se inicia con la resolución de subproblemas más pequeños, y utiliza sus soluciones en la resolución de los subproblemas de niveles superiores.

Ejemplo Los Números de Fibonacci

Algorítmicas

de Técnicas

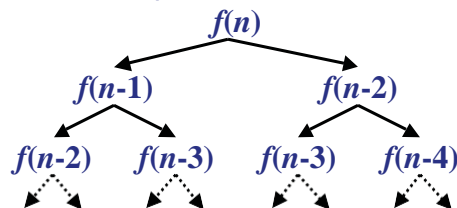
Análisis

- ▶ Sucesión de Fibonacci:
 $F_0 = 1$
 $F_1 = 1$
 $F_n = F_{n-1} + F_{n-2} \quad \forall n \geq 2$

- ▶ Función C utilizando divide y vencerás es:

```
long int fibonacci(int n){  
    if (n<2) return 1;  
    else return fibonacci(n-2)+fibonacci(n-1);  
}
```

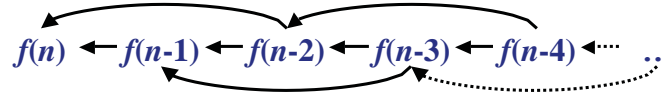
- ▶ En la resolución se repiten llamadas recursivas:



Los Números de Fibonacci con Programación Dinámica

Algorítmicas

- ▶ Resulta sencillo evitar la repetición de llamadas recursivas utilizando el siguiente esquema:



Fibonacci con programación dinámica:

```
int * fibonacci(int f[n]){    // Se supone n>2
    f[0]=0;
    f[1]=1;
    for (int i=2;i<n;i++)
        f[i]=f[i-1]+f[i-2];
    return f;
}
```

Análisis

- ▶ El coste de este algoritmo es $O(n)$ frente a $O(2^n)$ del anterior.

Características de la Programación Dinámica

Algorítmicas

- ▶ La descomposición en subproblemas suele hacerse de acuerdo al principio de optimalidad.
- ▶ La estructura de datos donde se van guardando las soluciones de los subproblemas resueltos suele ser una tabla.
- ▶ La mayor dificultad de esta técnica consiste en determinar el orden en que se deberá rellenar la tabla. Para resolver un subproblema, previamente deben estar almacenadas todas las soluciones necesarias.
- ▶ La programación dinámica supone un compromiso entre el coste espacial y el coste temporal. Si para reducir un poco el coste temporal se requiere un excesivo coste espacial, puede no ser una buena técnica.

de Técnicas

Análisis

Ejemplos de Programación Dinámica

Algorítmicas

de Técnicas

Análisis

- ▶ Algunos problemas conocidos que pueden resolverse mediante programación dinámica son:
 - El cálculo de los números de Fibonacci.
 - El cálculo de números combinatorios.
 - El problema de la mochila entera.
 - El problema de los caminos mínimos en grafos.
 - La multiplicación de una secuencia de matrices.
 - El problema del viajante.

El Problema de la Mochila Entera

Algorítmicas

de Técnicas

Análisis

- ▶ Ya vimos un algoritmo voraz para resolver el problema de la mochila fraccionaria, sin embargo, la técnica voraz no garantiza la obtención de la solución óptima cuando los objetos no se pueden fraccionar.
- ▶ Ejemplo. Si tenemos 3 objetos con valores $v_1=18$, $v_2=20$ y $v_3=12$, y pesos $p_1=6$, $p_2=4$ y $p_3=2$, y el peso máximo de la mochila es $p_{\max}=10$, entonces el algoritmo voraz calcula la solución $(x_1=0, x_2=1, x_3=1)$ que tiene un valor 32, mientras que la solución óptima es $(x_1=1, x_2=1, x_3=0)$ que tiene un valor 38.
- ▶ Veremos cómo la programación dinámica si permite llegar a la solución óptima.

El Problema de la Mochila Entera

Algorítmicas

- ▶ Si llamamos $f(n, p_{\max})$ al valor de la mochila en la solución óptima, aplicando el principio de optimalidad, se tiene

$$f(n, p_{\max}) = \begin{cases} 0 & \text{si } n = 1 \text{ y } p_{\max} < p_1 \\ v_1 & \text{si } n = 1 \text{ y } p_{\max} \geq p_1 \\ f(n-1, p_{\max}) & \text{si } n > 1 \text{ y } p_{\max} < p_n \\ \max(f(n-1, p_{\max}), f(n-1, p_{\max} - p_n) + v_n) & \text{si } n > 1 \text{ y } p_{\max} \geq p_n \end{cases}$$

de Técnicas

- ▶ El algoritmo recursivo que implementa directamente la función anterior tiene un coste exponencial $O(2^n)$.
- ▶ Sin embargo, el número total de subproblemas a resolver no es tan grande, ya que el primer parámetro de f puede oscilar entre 1 y n , y el segundo entre 0 y p_{\max} , en total $n(p_{\max}+1)$.

Análisis

- ▶ Así pues, *¡hay subproblemas que se repiten!* Podemos utilizar una tabla para almacenar los subproblemas resueltos, en la que en la posición (i, j) se almacene el valor de $f(i, j)$.

El Problema de la Mochila Entera Ejemplo

Algorítmicas

de Técnicas

Análisis

- Supongamos que tenemos 5 objetos con pesos $p[0]=5$, $p[1]=4$, $p[2]=2$, $p[3]=3$, $p[4]=4$, y valores $v[0]=20$, $v[1]=15$, $v[2]=10$, $v[3]=12$, $v[4]=14$, entonces, si el peso máximo de la mochila es 10, la función anterior generaría la siguiente tabla:

Peso	0	1	2	3	4	5	6	7	8	9	10
Objeto 0	0	0	0	0	0	20	20	20	20	20	20
Objeto 1	0	0	0	0	15	20	20	20	20	35	35
Objeto 2	0	0	10	10	15	20	25	30	30	35	35
Objeto 3	0	0	10	12	15	22	25	30	32	37	42
Objeto 4	0	0	10	12	15	22	25	30	32	37	42

Valor máximo de la mochila



El Problema de la Mochila Entera Coste Temporal

Algorítmicas

de Técnicas

Análisis

- ▶ Cada componente de la tabla se calcula en un tiempo constante, luego la construcción de la tabla es $O(np_{\max})$.
- ▶ Por otro lado, la función que determina los objetos introducidos en la mochila a partir de la tabla, sólo tiene un bucle que recorre todos los objetos para ver si están o no dentro de la mochila. Como el cuerpo del bucle tiene un coste constante, su coste es $O(n)$.
- ▶ Así pues, el coste total es $O(np_{\max}) + O(n) = O(np_{\max})$.
- ▶ Se observa que si p_{\max} es muy grande ($p_{\max} \geq n$) sería un coste cuadrático o superior, por lo que no sería muy eficiente.
- ▶ Por último, hay que notar que si los pesos no fuesen enteros, el algoritmo no valdría.

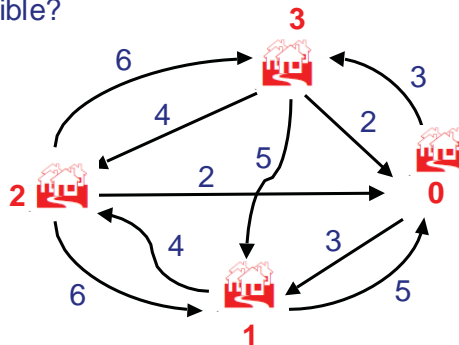
El Problema del Viajante

Algorítmicas

de Técnicas

Análisis

- ▶ Un viajante tiene que recorrer n ciudades y regresar a la ciudad de partida, sin pasar dos veces por la misma ciudad.
- ▶ Se supone que los caminos son unidireccionales y se conoce la distancia de cada camino.
- ▶ ¿Cuál es recorrido que debe seguir para que sea lo más corto posible?



El Problema del Viajante Modelización Mediante Grafos

Algorítmicas

de Técnicas

Análisis

- Si representamos cada ciudad mediante un nodo de un grafo y cada camino como una arista dirigida, tendremos un grafo dirigido $G=(V,A)$ donde

- $V=\{1,\dots,n\}$ es el conjunto de vértices (ciudades).
- A es el conjunto de aristas (i,j) con $i,j \in V$ (caminos).
- $D(i,j)$ es la longitud de (i,j) si $(i,j) \in A$, o ∞ si $(i,j) \notin A$.

De\A	0	1	2	3
0	∞	3	∞	3
1	5	∞	4	∞
2	2	6	∞	6
3	2	5	4	∞

- Se trata de un problema de búsqueda en un grafo del ciclo Hamiltoniano de longitud mínima.

El Problema del Viajante

Algorítmicas

de Técnicas

Análisis

- Suponiendo que la ciudad de partida es la 0, si llamamos $C(i,W)$ a la longitud del camino mínimo de i a 0 que pasa por todos los vértices de $W \subset V$, siendo $0 \notin W$, entonces

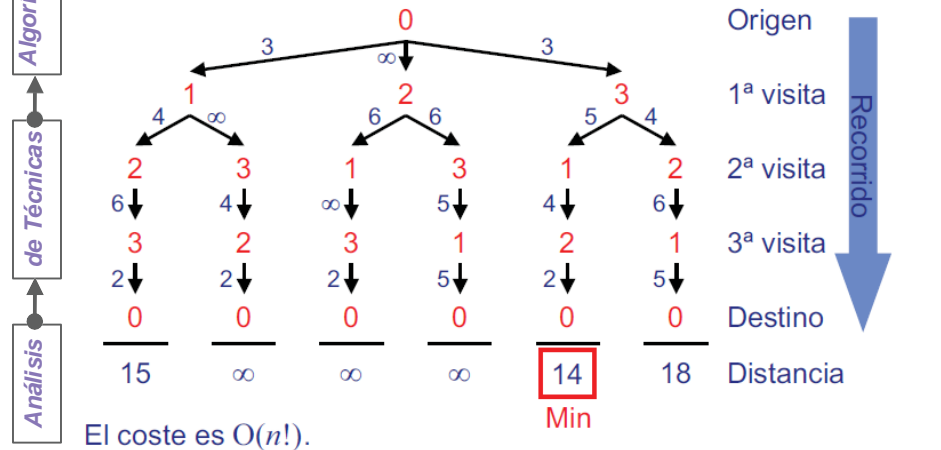
$$C(i,W) = \begin{cases} D(i,0) & \text{si } W = \emptyset \\ \min_{j \in W} (D(i,j) + C(j,W - \{j\})) & \text{si } W \neq \emptyset \end{cases}$$

- Como la ciudad de partida es la 0, la solución al problema será $C(0,V-\{0\})$.
- Se cumple el principio de optimalidad.


El Problema del Viajante

Coste del Algoritmo de Fuerza Bruta

- Esta función estudia todos los posibles caminos por fuerza bruta



El Problema del Viajante Mejora de la eficiencia

- 
 - ▶ La función anterior repite llamadas para calcular $C(i, W)$, y por tanto es ineficiente.
 - ▶ Utilizando la técnica de programación dinámica, podemos guardar los valores de $C(i, W)$ en una tabla para evitar repetir su cálculo.
 - ▶ Para indexar la tabla en la segunda dimensión, asignaremos un código único a cada subconjunto W para identificarlo.
 - ▶ Puesto que el número de ciudades es n , el número de posibles subconjuntos es 2^n . Si representamos cada subconjunto mediante un vector

$$W = [x_0, \dots, x_{n-1}] \text{ con } x_i = \begin{cases} 0 & \text{si } i \notin w \\ 1 & \text{si } i \in w \end{cases}$$

entonces, $\text{id}(W) = x_0 2^0 + x_1 2^1 + \dots + x_{n-1} 2^{n-1}$ asigna un código único a cada subconjunto.

El Problema del Viajante Construcción de la Tabla

► Para el ejemplo anterior, con $D(i,j)$

DeA	0	1	2	3
0	∞	3	∞	3
1	5	∞	4	∞
2	2	6	∞	6
3	2	5	4	∞

la tabla de distancias mínimas resultante $C(i,W)$ es

$i \setminus W$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	14	-1
1	-1	-1	-1	-1	6	-1	-1	-1	∞	-1	-1	-1	12	-1	-1	-1
2	-1	-1	11	-1	-1	-1	-1	-1	8	-1	16	-1	-1	-1	-1	-1
3	-1	-1	10	-1	6	-1	11	-1	-1	-1	-1	-1	-1	-1	-1	-1

Algorítmicas

de Técnicas

Análisis

Vuelta Atrás (Backtracking)

Algorítmicas
de Técnicas
Análisis

- ▶ La técnica de VUELTA ATRÁS es parecida a la técnica devoradora en que resuelve un problema de manera gradual tomando decisiones o realizando acciones que permiten ir construyendo progresivamente la solución del problema. Sin embargo, a diferencia de la técnica devoradora, las decisiones tomadas pueden volverse a reconsiderar si no se llega una solución aceptable u óptima.
- ▶ Básicamente, esta técnica comienza a resolver el problema buscando todas las soluciones posibles, como lo hace la técnica de fuerza bruta, pero cuando detecta que una solución parcial no puede llegar a ser óptima, la rechaza y no construye más soluciones a partir de ella.
- ▶ Se suele aplicar también a problemas de decisión u optimización con o sin restricciones, cuando no existe un criterio óptimo de toma de decisiones locales.

Recordemos los Elementos de un Problema de Decisión

Algorítmicas

de Técnicas

Análisis

- ▶ FUNCIÓN OBJETIVO, que hay que minimizar o maximizar y depende de las variables x_1, \dots, x_n .
- ▶ DOMINIO, que contiene el conjunto de valores que pueden tomar las variables.
- ▶ FUNCIÓN SOLUCIÓN, que permite saber si unos determinados valores de las variables son o no solución del problema.
- ▶ DECISIÓN, que es la asignación de un valor a una variable.
- ▶ RESTRICCIONES, que son las condiciones que deben cumplir las soluciones.
- ▶ FUNCIÓN FACTIBLE, que permite saber si una solución cumple o no las restricciones.
- ▶ SOLUCIÓN EN CURSO, que es el conjunto de decisiones factibles tomadas hasta el momento.

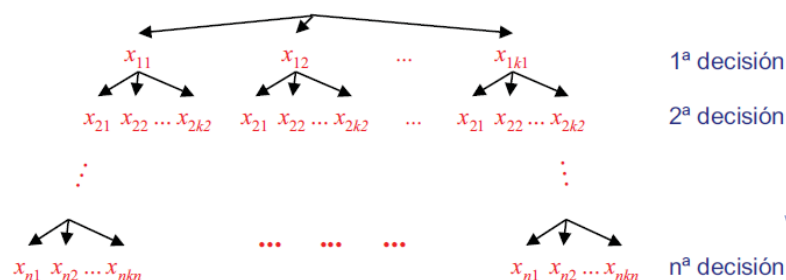
Espacio de Búsqueda

Algorítmicas

de Técnicas

Análisis

- ▶ La solución del problema puede expresarse como una tupla (x_1, \dots, x_n) con $x_i \in C_i$, siendo C_i el dominio de x_i .
- ▶ Así pues, el número total de posibles soluciones es $\prod_{i=1}^n k_i$ siendo k_i el número de elementos de C_i .
- ▶ El conjunto de todas las posibles soluciones se conoce como ESPACIO DE BÚSQUEDA y se representa como un árbol



Recorrido en Profundidad del Espacio de Búsqueda

Análisis
de Técnicas
Algorítmicas

- ▶ Las soluciones se construyen partiendo de la raíz del espacio de búsqueda y tomando decisiones realizando un recorrido en profundidad del mismo.
- ▶ Se dice una solución en curso es **COMPLETABLE** si a partir de ella se puede alcanzar la solución del problema.
- ▶ Mientras que la técnica de fuerza bruta recorrería todo el espacio de búsqueda, *lo que supondría un coste exponencial*, la técnica de vuelta atrás, cuando llega a una solución no completable, da marcha atrás y busca por otro camino del espacio de búsqueda, evitando así recorrer todo el subárbol que queda por debajo de dicha solución no completable.

Diferentes Versiones de Algoritmos con Vuelta Atrás

Análisis
de Técnicas
Algorítmicas

- ▶ Dependiendo del tipo de problema que tengamos que resolver existen diferentes formas de aplicar la vuelta atrás:
 - **Vuelta atrás para una solución:** El algoritmo recorre el espacio de búsqueda hasta que encuentra la primera solución. Es el más sencillo.
 - **Vuelta atrás para todas las soluciones:** El algoritmo recorre el espacio de búsqueda guardando todas las soluciones que encuentra hasta que ya no haya más.
 - **Vuelta atrás para la mejor solución:** El algoritmo recorre el espacio de búsqueda comparando cada solución que encuentra con la mejor solución obtenida hasta el momento, y quedándose con la mejor. Cuando ya no hay más soluciones, devuelve la mejor encontrada. Suele aplicarse a problemas de optimización.

Algoritmo Genérico con Vuelta Atrás para Todas las Soluciones

Algorítmicas

de Técnicas

Análisis

- El siguiente algoritmo calcula todas las soluciones de un problema con la técnica de vuelta atrás

```
funcion vuelta_atras(entero i,sol[n])
  para_todo x en Ci
    sol[i]=x
    si completable(sol,i) entonces
      si solucion(sol) entonces
        guardar(sol)
      fin_si
    si (i<k) entonces
      vuelta_atras(i+1,sol)
    fin_si
  fin_si
fin_para_todo
fin_funcion
```

Poda del Espacio de Búsqueda

Algorítmicas

de Técnicas

Análisis

- El algoritmo anterior no genera el espacio de búsqueda de forma explícita, sino implícitamente mediante llamadas recursivas.
- Cuando se llega a una solución en curso no completable, no se realizan más llamadas recursivas y por tanto no se construye el subárbol correspondiente del espacio de búsqueda. Esto se conoce como PODA del espacio de búsqueda.
- La poda es un mecanismo que permite descartar del recorrido ciertas zonas del espacio de búsqueda, bien porque allí no haya soluciones, bien porque ninguna de las soluciones de esas zonas es la óptima.
- Cuanto mayor sea el número de nodos podados, más eficiente será la búsqueda de soluciones.

Coste Temporal del Algoritmo de Vuelta Atrás

Algorítmicas

de Técnicas

Análisis

- ▶ El coste temporal de un algoritmo que utilice la técnica de vuelta atrás, suele depender de:
 - El número de nodos del espacio de búsqueda que se visitan $v(n)$.
 - El coste de las funciones **solucion** y **completable** en cada nodo $p(n)$.

En total $O(p(n)v(n))$.

- ▶ Si se consigue que la función **completable** padezca muchos nodos, el tamaño de $v(n)$ se puede reducir drásticamente:
 - Si se reduce a un solo nodo (solución voraz) tendremos un coste $O(p(n)n)$.
 - Por el contrario, en el peor de los casos (solución por fuerza bruta), tendremos un coste $O(p(n)k^n)$.

Ejemplos de Vuelta Atrás

Algorítmicas

de Técnicas

Análisis

- ▶ Algunos problemas conocidos que pueden resolverse mediante la técnica de vuelta atrás:
 - El problema de las ocho reinas.
 - La suma de subconjuntos.
 - El problema de la mochila entera.
 - El problema del coloreado de grafos.
 - La búsqueda de ciclos Hamiltonianos en grafos.
 - El recorrido de un laberinto.

El Problema del Coloreado de Grafos

Algorítmicas

de Técnicas

Análisis

- ▶ Se dispone de un grafo G y de k colores. ¿Es posible colorear los vértices de G de manera que no haya dos vértices adyacentes con el mismo color?
- ▶ Otro famoso problema reducible a este es el del coloreado de mapas. ¿Puede pintarse un mapa de manera que no haya dos regiones adyacentes con el mismo color, utilizando k colores?
- ▶ Cada región se representa como un vértice del grafo y si dos regiones son adyacentes, sus respectivos vértices de conectan con una arista.

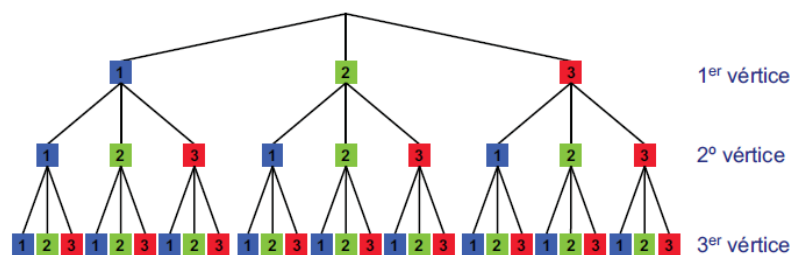
Coloreado de Grafos Espacio de Búsqueda

Algorítmicas

de Técnicas

Análisis

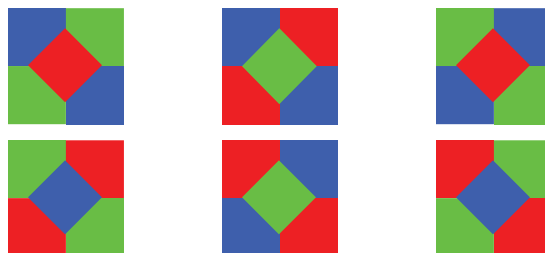
- ▶ Si el grafo tiene n vértices, representaremos el color de cada vértice en un vector $\text{color}[n] = \{c_1, \dots, c_n\}$ siendo, $c_i \in \{1, \dots, k\}$ el color del vértice i .
- ▶ El espacio de búsqueda asociado tiene k^n posibles soluciones. Por ejemplo, para $n=3$ y $k=3$ se tiene el siguiente espacio de búsqueda con 27 posibles soluciones.



Coloreado de Grafos Ejemplo

► Aplicando la función anterior al mapa de 5 regiones

Algorítmicas
de Técnicas
Análisis



Algoritmo Genérico con Vuelta Atrás para la Mejor Solución

Algorítmicas

de Técnicas

Análisis

- El siguiente algoritmo calcula la mejor solución de un problema de optimización con la técnica de vuelta atrás

```
funcion vuelta_atras(entero i,sol[n])
  para_todo x en Ci hacer
    sol[i]=x
    si completable(sol,i) y coste(sol,i) < mcoste entonces
      si solucion(sol) entonces
        mejor_solucion=sol
        mcoste=coste(sol,i)
      fin_si
    si (i<k) entonces
      vuelta_atras(i+1,sol)
    fin_si
  fin_si
fin_para_todo
fin_funcion
```

Poda Basada en la Mejor Solución

Algorítmicas

de Técnicas

Análisis

- Cuando estemos ante un problema de optimización, el objetivo del algoritmo es recorrer el espacio de búsqueda para encontrar la mejor solución.
- En este caso, además de la poda que realiza la función **completable**, podemos realizar otra poda basada en el coste de la mejor solución en curso.
- La idea es que una solución en curso se poda, cuando, a pesar de ser completable, no es posible conseguir una solución mejor que la mejor solución en curso, aún cuando se genere todo el subárbol del espacio de búsqueda que cuelga de ella (**coste(sol,i) >= mejorcoste**).
- Como antes, esta poda es más efectiva cuantos más nodos consiga eliminar.

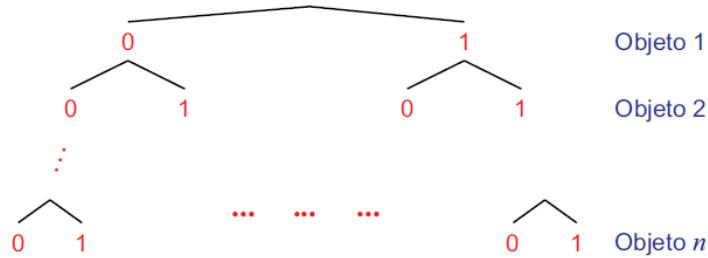
El Problema de la Mochila Entera



- ▶ El problema de la mochila entera también puede resolverse mediante la técnica de vuelta atrás.
- ▶ Si tenemos n objetos y representamos el contenido de la mochila con un vector $\text{sol}[n]=\{x_1, \dots, x_n\}$ donde

$$x_i = \begin{cases} 0 & \text{si el objeto } i \text{ no está en la mochila} \\ 1 & \text{si el objeto } i \text{ está en la mochila} \end{cases}$$

entonces el espacio de búsqueda asociado es



El Problema de la Mochila Entera Poda Basada en la Mejor Solución



- ▶ Puesto que se trata de un problema de optimización, aplicaremos también la poda basada en la mejor solución en curso.
- ▶ Para ello resulta útil disponer de una función $cota(sol,i)$ que calcule una cota superior del valor de la mochila en las soluciones que pueden obtenerse a partir de la solución en curso.
- ▶ Con esta función, cada vez que $cota(sol,i) < maxval$, siendo $maxval$ el máximo valor de la mochila encontrado hasta el momento (al comienzo $maxval=0$), podaremos el subárbol del espacio de búsqueda que cuelgue de dicha solución en curso y haremos la vuelta atrás.
- ▶ En nuestro caso, esta cota puede obtenerse mediante el algoritmo voraz para el problema de la mochila fraccionada.

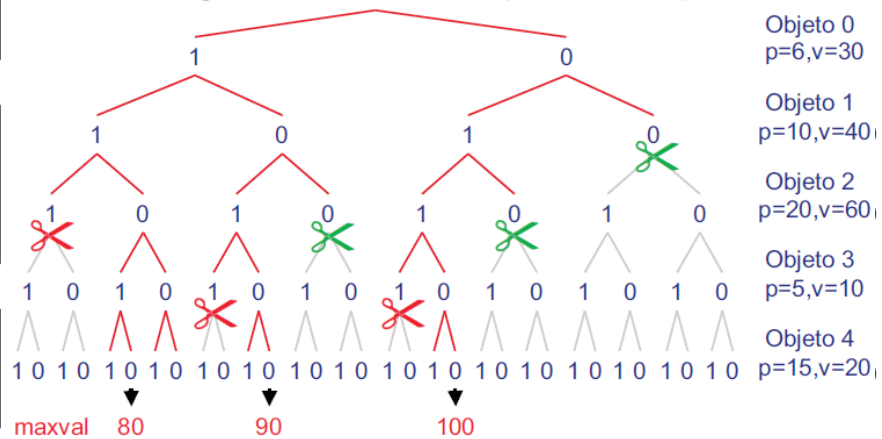
Mochila Entera. Ejemplo para una mochila con peso máx 30k

Algorítmicas

de Técnicas

Análisis

- Si hay 5 objetos de valores $\text{valor}[] = \{30, 40, 60, 10, 20\}$ y pesos $\text{peso}[] = \{6, 10, 20, 5, 15\}$, entonces la función realiza el siguiente recorrido del espacio de búsqueda



Mejoras de la Vuelta Atrás

Algorítmicas

de Técnicas

Análisis

- Una posible mejora sería comenzar buscando una solución factible por algún procedimiento rápido, e inicializar el valor de la mejor solución con el valor de dicha solución. De esta manera, la poda basada en mejor solución comenzaría a funcionar antes.

Ramificación y Poda (Branch & Bound)

Algorítmicas
de Técnicas
Análisis

- ▶ La técnica de RAMIFICACIÓN Y PODA es básicamente una mejora de la técnica de vuelta atrás que se basa en un recorrido dinámico más eficiente del espacio de búsqueda.
- ▶ La principal diferencia es que, mientras que la técnica de vuelta atrás realiza un *recorrido ciego* del espacio de búsqueda, la técnica de ramificación y poda realiza un *recorrido informado*.
- ▶ En un **recorrido ciego**, bien sea en profundidad o en anchura, se conoce perfectamente el siguiente nodo a visitar (siguiente hijo o hermano del nodo actual), es decir, el orden del recorrido está establecido a priori, mientras que en un **recorrido informado**, se busca el siguiente nodo más prometedor en base a la información de que se disponga.
- ▶ Se aplica fundamentalmente a problemas de optimización con restricciones, donde el espacio de búsqueda es un árbol.

Nodos Prometedores

Algorítmicas
de Técnicas
Análisis

- ▶ Diremos que un nodo del espacio de búsqueda es PROMETEDOR, si la información que tenemos de ese nodo en el recorrido indica que expandiéndolo se puede conseguir una solución mejor que la mejor solución obtenida hasta el momento.
- ▶ Esta información puede provenir de la evaluación del camino ya recorrido hasta el nodo, o bien de la evaluación de los caminos quedan por recorrer desde el nodo a posibles soluciones.
- ▶ Para guiar la búsqueda es fundamental disponer de una función que, no sólo diga si un nodo es o no prometedor, sino que además estime lo prometedor que es.

Caracterización del Espacio de Búsqueda en Ramificación y Poda

Algorítmicas

de Técnicas

Análisis

- ▶ El espacio de búsqueda se representa mediante un árbol en el que hay tres clases de nodos:
 - **Nodo vivo:** Es un nodo hasta el cual la solución en curso es factible, prometedora, y del que no se han generado aún todos sus hijos (no se ha completado la solución). Indican caminos abiertos de búsqueda.
 - **Nodo muerto:** Es un nodo del que no van a generarse más hijos porque, o bien ya se han generado todos sus hijos, o bien no es factible, o bien no es prometedora. Indican caminos cerrados de búsqueda.
 - **Nodo en expansión:** Es aquel del que se están generando sus hijos en un determinado instante.
- ▶ En un determinado instante de la búsqueda puede haber muchos nodos vivos y muertos pero sólo uno en expansión.

Funcionamiento de la Búsqueda en Ramificación y Poda

Algorítmicas

de Técnicas

Análisis

- ▶ En la técnica de vuelta atrás, cada vez que se generaba un hijo del nodo en expansión, este pasaba inmediatamente a ser el nodo en expansión. En consecuencia, los únicos nodos vivos son los que están en el camino de la raíz al nodo en expansión.
- ▶ Por el contrario, en la técnica de ramificación y poda se generan todos los hijos del nodo en expansión antes de que cualquier otro nodo vivo pase a ser el nuevo nodo en expansión.
Esto supone que puede haber nodos vivos en distintos caminos y tendremos que guardar la lista de nodos vivos en alguna estructura de datos (suelen ser pilas o colas).

Estrategias de Búsqueda

Algorítmicas

de Técnicas

Análisis

- ▶ Existen diferentes estrategias para elegir el siguiente nodo en expansión de la lista de nodos vivos:
 - **Más antiguo.** Se elige como nodo en expansión el más antiguo de la lista. En este caso la lista se implementa como una cola. Produce un recorrido en anchura.
 - **Menos antiguo.** Se elige como nodo en expansión el menos antiguo de la lista. En este caso la lista se implementa como una pila. Produce un recorrido en profundidad.
 - **Más prometedor.** Se elige como nodo de expansión el nodo más prometedor de la lista. La lista se implementa como una cola con prioridades, donde la prioridad de un nodo es lo prometedor que es. Produce un recorrido informado y *¡es el que suele utilizar la técnica de ramificación y poda!*

El Problema de la Mochila Entera Mejora con Ramificación y Poda

Algorítmicas

de Técnicas

Análisis

- ▶ Se puede mejorar la solución dada al problema de la mochila entera con la técnica de vuelta atrás, si, además de las podas que realizaba esta, se dirige la búsqueda con la función de prioridad.
- ▶ Tomando como función de prioridad la misma función que la que calculaba una cota superior del valor de la mochila para cada nodo del espacio de búsqueda, se expandirán primero los nodos que tengan una cota mayor, y que probablemente llevarán a una solución con un valor de la mochila mayor.