Algoritmos
Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología

Universidad Nacional de Tucumán

2025

Técnicas algorítmicas(3)

Programación Dinámica - Ejemplo

Los **coeficientes binomiales**, números combinatorios o combinaciones se definen por:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Corresponde al número de subconjuntos de k elementos elegidos de un conjunto con n elementos.

Cuando se desarrolla un binomio $(a+b)^n$, el coeficiente binomial corresponde al coeficiente del término $a^{n-k}\, b^k$

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k}\, b^k$$

Programación Dinámica - Ejemplo

Fórmula recursiva de cálculo de **coeficientes binomiales**

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad 0 < k < n \qquad \binom{n}{n} = 1 \quad \binom{n}{0} = 1$$

La función recursiva que surge de esta definición es:

```
FUNCION C(n,k) : entero>0 x entero ≥0 → entero>0
    SI k=0 o k=n ENTONCES
        retorna(1)
    SINO
        retorna (C(n-1,k-1) + C(n-1,k))
FIN
```

El algoritmo hace 2*combinaciones(n,k) -2 llamadas recursivas

Programación Dinámica - Ejemplo

Para resolver por programación dinámica se usa una tabla C(0..n,0..k).

La tabla se llena por fila de izquierda a derecha usando 2 elementos de la fila anterior: $C(i,j) = C(i-1,j-1) + C(i-1,j)$

	0	**1**	**2**	**3**	**…**	**k-1**	**k**
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
…							
n-1	1					C(n-1,k-1)	C(n-1,k)
n	1						C(n,k) = C(n-1,k-1) + C(n-1,k)

El algoritmo tiene: tiempo$\in$O(n.k), y usa almacenamiento$\in$O(n.k)

Programación Dinámica - Ejemplo

Triángulo de Pascal o Triángulo de Tartaglia

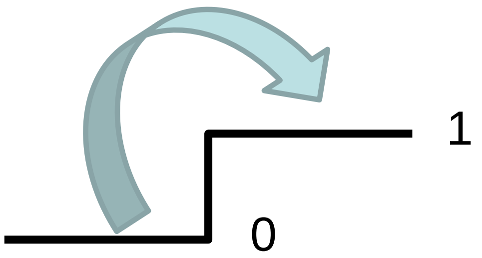
```
                  1
                1   1
              1   2   1
            1   3   3   1
          1   4   6   4   1
        1   5  10  10   5   1
      1   6  15  20  15   6   1
    1   7  21  35  35  21   7   1
  1   8  28  56  70  56  28   8   1
1   9  36  84 126 126  84  36   9   1
```

Programación Dinámica - Ejemplo

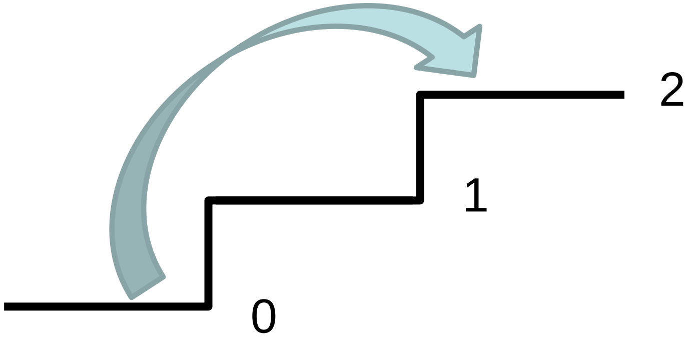
Subir una escalera:

Dos pasos posibles:

1) Al escalón siguiente

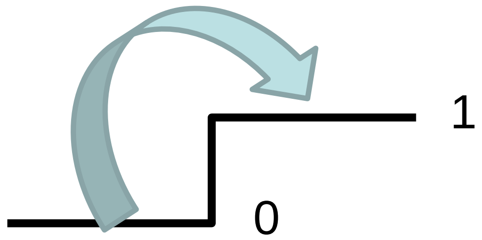


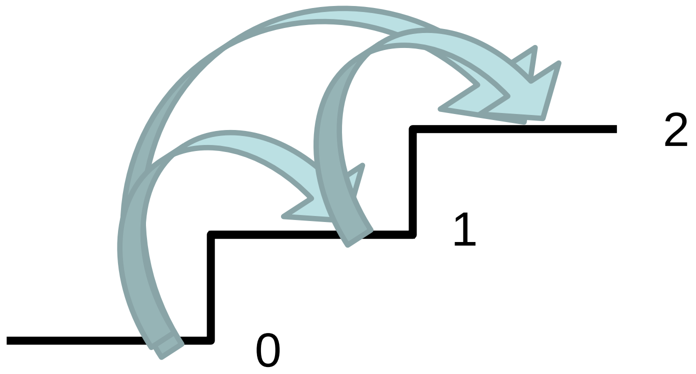
2) Salteando un escalón



Cuántas formas distintas hay para llegar al escalón **n** partiendo de **0** ?

Programación Dinámica - Ejemplo





Cuántas formas distintas hay para llegar al escalón 1 ?

Escalon 1: se llega de una sola manera

NUM(1) = 1

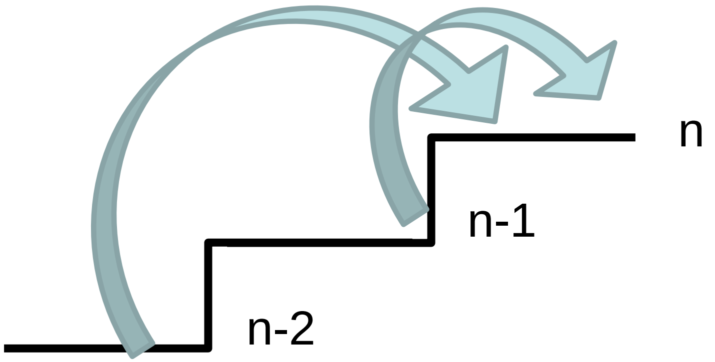
Cuántas formas distintas hay para llegar al escalón 2 ?

Escalon 2: se llega de dos maneras distintas.

NUM(2) = 2

Programación Dinámica - Ejemplo

Cuántas formas distintas hay para llegar al escalón n ?



Escalon n:

NUM(n) = NUM(n-1) + NUM(n-2)   para n>2

Programación Dinámica - Ejemplo

```
FUNCION ESCALERA(n) : entero ≥1 → entero ≥1
   SI n=1 ENTONCES
        retorna (1)
   SINO
        SI n=2 ENTONCES
                          retorna 2
         SINO          // n>2
                          retorna ESCALERA(n-1) + ESCALERA(n-2)
FIN
```

Será eficiente este algoritmo ?

Como sería la implementación con Programación Dinámica ?

Técnica Greedy

El nombre de algoritmo *greedy* se debe a su comportamiento:

- Avanzan siempre hacia "la solución más prometedora".

Los algoritmos greedy son:

- sencillos,
- fáciles de inventar,
- fáciles de implementar,
- cuando funcionan, son eficientes.
- generalmente utilizan la estrategia top-down

- Se usan para resolver *problemas de optimización*.

Técnica Greedy

Se aplica a problemas de optimización que se pueden resolver mediante una ***secuencia de decisiones*.**

Un algoritmo greedy:

- trabaja en una secuencia de pasos
- en cada etapa hace una elección local que se considera una *decisión óptima*,
- luego se resuelve el subproblema que resulta de esta elección,
- finalmente estas soluciones localmente óptimas se integran en una solución global óptima.

Para cada problema de optimización se debe demostrar que la elección óptima en cada paso, lleva a una solución óptima global.

Frecuentemente se preprocesa la Entrada o se usa una Estructura de Datos adecuada para hacer rápida la elección greedy y así tener un algoritmo más eficiente.

Técnica Greedy

- Dado un problema con *n* entradas el método consiste en obtener un subconjunto de éstas que satisfaga una determinada restricción definida para el problema.

- Cada uno de los subconjuntos que cumplan las restricciones se dice que son *soluciones prometedoras.*

- Una solución prometedora que *maximice* o *minimice* una *función objetivo* se denomina *solución óptima*.

- Cuando el algoritmo greedy funciona correctamente, la primera solución que encuentre es siempre óptima.

Técnica Greedy

Elementos que deben estar presentes en el problema:

- Un conjunto de ***candidatos***, que corresponden a las *n* entradas del problema.
- Una ***función de selección*** que en cada momento determina el candidato idóneo para formar la solución de entre los que aún no han sido seleccionados ni rechazados.
- Una función que compruebe si un cierto subconjunto de candidatos forman ***una solución prometedor***a.
- Una función que comprueba si un subconjunto de estas entradas es ***solución*** al problema, sea óptima o no.
- Una ***función objetivo*** que determina el valor de la solución encontrada. Es la función que se quiere maximizar o minimizar.

Técnica Greedy

En un **algoritmo Greedy** se tiene generalmente:

C: conjunto de candidatos que nos sirven para construir la solución del problema.

S: conjunto de los candidatos ya usados para armar la solución

solución: una función que chequea si un conjunto es solución del problema, ignorando en principio si esta solución es óptima o no.

factible: una función que chequea si un conjunto de candidatos es factible como solución del problema sin considerar si es óptima o no.

selección: una función que indique cual es el candidato mas prometedor que no se eligió todavía. Está relacionada con la función objetivo.

objetivo: una función que es lo que se trata de optimizar. (Esta función no aparece en el algoritmo).

Esquema Algoritmo Greedy

```
FUNCIÓN Greedy (C): conjunto → conjunto
    S ← ∅
    MIENTRAS not solución (S) AND C ≠ ∅ HACER
        x ← selección (C)
        C ← C - {x}
        SI factible ( S U {x} ) ENTONCES
            S ← S U {x}
    FIN MIENTRAS
    SI solución (S) ENTONCES
        RETORNA (S)
    SINO
        RETORNA (∅ )
FIN
```

Técnica Greedy

En los algoritmos Greedy el proceso no finaliza al diseñar e implementar el algoritmo que resuelve el problema en consideración.

Hay una tarea extra muy importante:

- **SI el algoritmo FUNCIONA BIEN**: la ***demostración formal*** de que el algoritmo encuentra la solución óptima en todos los casos.

- **SI el algoritmo NO FUNCIONA**: la presentación de un ***contraejemplo*** que muestra los casos en donde falla.

Técnica Greedy
Ejemplo: Mínimo de monedas

Funcion darvuelto(vuelto): entero → conjunto de monedas

C ← {25, 10, 5, 1} con suficiente cantidad de c/u

S ← ∅

P1. suma ← 0

P2. MIENTRAS suma ≠ vuelto HACER

x ← el elemento de mayor valor en C

C ← C - {x}

SI suma+x ≤ vuelto ENTONCES

S ← S u {x}

suma ← suma+x

P3. RETORNA S

P4. FIN

Este algoritmo da siempre la solución óptima?

Técnica Greedy
Ejemplo: Mínimo de monedas

- Se puede demostrar que con los valores de monedas sugeridos (valores de 1, 5, 10 y 25 unidades).

- Si hay disponibles de todas las denominaciones de monedas en cantidad suficiente en el conjunto inicial.

- Entonces este algoritmo Greedy que elige en cada paso la moneda de mayor valor***, siempre encontrará la solución optima.***

Técnica Greedy
Ejemplo: Mínimo de monedas

Se puede demostrar que con los valores de monedas de curso legal en Argentina:

- ✓ 1 , 5 , 10 , 25 , 50 Centavos (en desuso)
- ✓ 1 , 2 , 5 , 10 Pesos

el algoritmo Greedy que elige en cada paso la *moneda de mayor valor,*

siempre encontrará la solución optima si es que existe una solución.

Técnica Greedy
Ejemplo: Mínimo de monedas

En Europa las monedas son de valor:
{1, 2, 5, 10, 20, 50 } centavos de euro
y de 1€ y 2€.

Funcionará la técnica Greedy planteada?

Se puede demostrar que funciona el algoritmo Greedy

Técnica Greedy
Ejemplo: Mínimo de monedas

En el Reino Unido las monedas son de valor:

{1, 2, 5, 10, 20, 25, 50 } peniques (centavos de libra esterlina)

y de 1, 2 y 5 £ (libra esterlina)

Funcionará la técnica Greedy planteada?

Ejemplo:

Para pagar 40 peniques:

Solución greedy usa: 25 + 10 + 5 (3 monedas)

Solución óptima usa: 20 + 20 (2 monedas)

Conclusión: No funciona el algoritmo Greedy

Técnica Greedy
Ejemplo: Problema de Carga

- Un barco se tiene que cargar con contenedores.
- Los contenedores son todos del mismo tamaño, pero de distintos pesos.
- La capacidad de carga del barco esta prefijada.
- Se quiere cargar el barco con el ***máximo número de contenedores.***

Está demostrado que este problema se puede resolver con una técnica greedy obteniendo siempre la solución óptima.

Técnica Greedy
Ejemplo: Problema de Carga

Datos:

n contenedores numerados: $i=1,2,3,\ldots n$

p_i= peso de cada contenedor i,

M= capacidad máxima de carga del barco

Solución: vector X

$x_i = 0$ si el contenedor i no se carga en el barco

$x_i = 1$ si el contenedor i si se carga en el barco

Maximizar la cantidad: $\sum_{i=1}^{n} x_i$

Restricción: $\sum_{i=1}^{n} p_i x_i \leq M$

SOLUCIÓN: Estrategia greedy en peso: en cada paso elegir el contenedor ***de menor peso posible*.**

Técnica Greedy
Ejemplo: Problema de Carga

ALGORITMO Carga

ENTRADA: n: entero
p: vector [1..n] de números reales, pesos de cada cont
M: nro. real, capacidad de carga del barco
SALIDA: X: vector [1..n] de valores: 0 y 1
AUXILIARES: sigue: bool
t: vector (1..n) de números enteros

P1. Leer (n, p, M)
P2. Pre procesar vector de pesos:
Usar un método de ordenación para dar valores a un arreglo de índices t(1..n) de modo que :
p[t(i)] ≤ p[t(i+1)]   para i=1..n

Técnica Greedy
Ejemplo: Problema de Carga

ALGORITMO carga (continuación)

P3. X[1..n] $\leftarrow$ 0
P4. sigue $\leftarrow$ true
P5. i $\leftarrow$1
P6. MIENTRAS (i $\leq$ n) and sigue HACER
    SI p[t(i)] $\leq$ M ENTONCES *// se puede cargar ?*
        X[t(i)] $\leftarrow$ 1 *// se carga*
        M $\leftarrow$ M - p[t(i)]
        i $\leftarrow$ i+1
    SINO
        sigue $\leftarrow$ false *// capacidad colmada*

P7. Escribir (X)
P8. FIN

Técnica Greedy
Ejemplo: Problema de la mochila

Este problema es una variante del problema de carga ya presentado.

- Se tiene ***n*** **objetos** y una mochila para llevarlos.
- Cada objeto tiene un ***peso*** y un ***beneficio*** asociado
- La mochila puede cargar un ***peso máximo*** dado.
- El objetivo es llenar la mochila de tal manera que se ***maximice el beneficio de los objetos transportados***, respetando la limitación de la capacidad impuesta.

Técnica Greedy
Ejemplo: Problema de la mochila

Datos: n objetos con sus pesos y beneficios y capacidad de carga:

$i=1,2,3,\ldots n$

p_i= peso de cada objeto i

b_i= beneficio asociado al objeto i

M= capacidad máxima de la mochila

Solución: vector $\boldsymbol{X}$, $i=1,2,3,\ldots n$

$\boldsymbol{x_i=0}$ si el objeto i no va en la mochila

$\boldsymbol{x_i=1}$ si el objeto i se carga en la mochila

Objetivo: Maximizar la cantidad: $$\sum_{i=1}^{n} b_i x_i$$

Restricción: $$\sum_{i=1}^{n} p_i x_i \leq M$$

Tiene solución con la Técnica Greedy?

Técnica Greedy
Ejemplo: Problema de la mochila

Posibles planteos Greedy para el problema de la mochila:

1. *Greedy al mayor beneficio*.

Ejemplo: n=3, M=105 , p=[100,10,10], b=[20,15,15]

Solución Greedy eligiendo en cada paso el objeto de mayor beneficio:

x=[1,0,0] , Beneficio Total = 20

Solución óptima :

x=[0,1,1] , Beneficio Total = 30

Técnica Greedy
Ejemplo: Problema de la mochila

Posibles planteos Greedy para el problema de la mochila:

2. *Greedy al menor peso*.

Ejemplo: n=2, M=25, p=[10,20], b=[5,100]

Solución Greedy eligiendo en cada paso el objeto de menor peso:

x=[1,0] , Beneficio Total = 5

Solución óptima :

x=[0,1] , Beneficio Total = 100.

Técnica Greedy
Ejemplo: Problema de la mochila

Posibles planteos Greedy para el problema de la mochila:

3. *Tampoco se consigue siempre la solución optima con estrategia Greedy para beneficio/peso*

Ni con ninguna estrategia…

Conclusión:

El problema de la mochila
NO tiene solución con la Técnica Greedy

Técnica Greedy
Ejemplo: Planificación 1 servidor

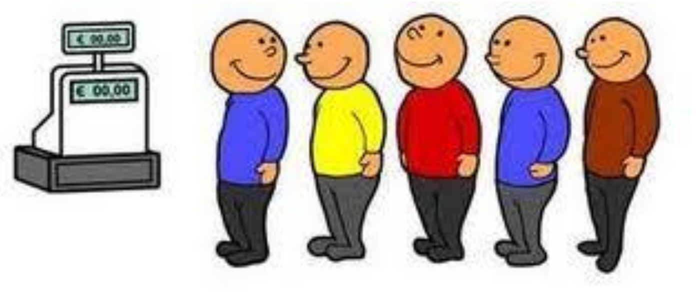
Considere ***un solo servidor*** que tiene que atender n clientes.
El tiempo de atención que necesita cada cliente se conoce de antemano.
Se quiere determinar el orden de atención de los clientes de manera de *minimizar el tiempo medio invertido por cada cliente* en el sistema.

Datos:

número de cada cliente: $i=1,n$

t_i: tiempo de atención del cliente i

E_i: tiempo de espera del cliente i



Objetivo:

Minimizar: $$E = \frac{\sum_{i=1}^{n} E_i}{n}$$

Cuál es la estrategia Greedy ?

Técnica Greedy
Ejemplo1: Planificación 1 servidor

Ejemplo: n=3 clientes: C1, C2, C3, tiempos: t1=5, t2=8, t3=3
Tiempo total de atención para estos clientes = ∑ ti = 16 .
Existen n!=6 posibles esquemas de orden de atención.

ORDEN	**Tiempo de Espera**			
	C1 (5)	**C2** (8)	**C3** (3)	**TIEMPO MEDIO**
C1C2C3	5	5+8	5+8+3	34/3
C1C3C2	5	5+3+8	5+3	29/3
C2C1C3	8+5	8	8+5+3	37/3
C2C3C1	8+3+5	8	8+3	35/3
C3C1C2	**3+5**	**3+5+8**	**3**	**27/3**
C3C2C1	3+8+5	3+8	3	30/3

El algoritmo greedy atiende a los clientes en orden creciente de *ti* y garantiza siempre la solución óptima en este problema.

Técnica Greedy
Ejemplo2: Planificación 1 servidor

Ejemplo: n=3 clientes: C1, C2, C3, tiempos: t1=8, t2=4, t3=6

ORDEN	**Tiempo de Espera**			
	C1	**C2**	**C3**	**TIEMPO MEDIO**
C1C2C3	8	8+4	8+4+6	
C1C3C2	8	8+6+4	8+6	
C2C1C3	4+8	4	4+8+6	
C2C3C1	4+6+8	4	4+6	
C3C1C2	6+8	6+8+4	6	
C3C2C1	6+4+8	6+4	6	

Será suficiente que la técnica greedy de la solución óptima en 2 ejemplos para decir que funciona siempre?