Generar y probar: n-tuplas y el problema de la mochila

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

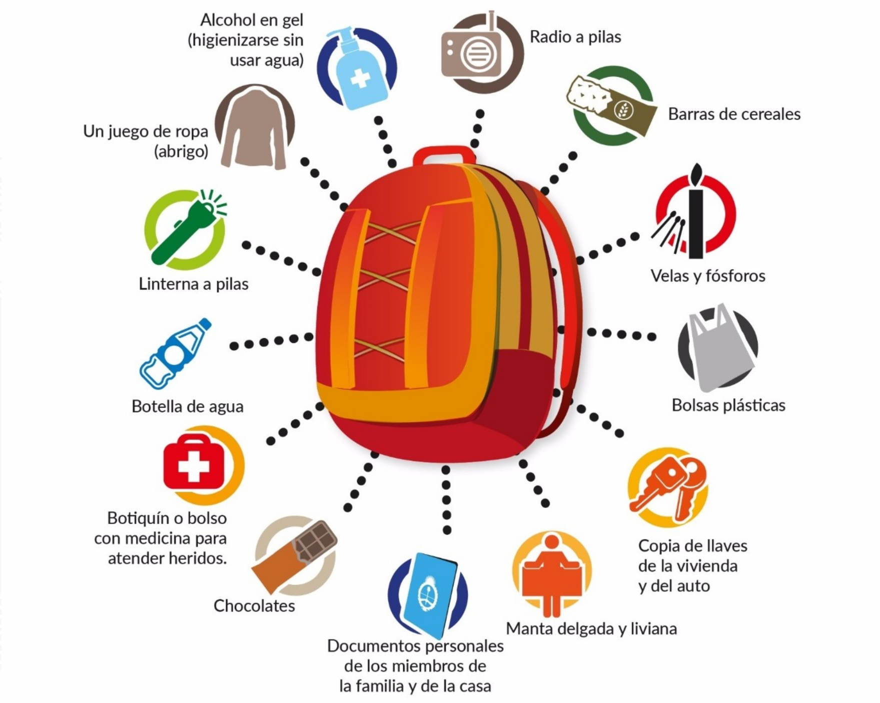
vpodberezski@fi.uba.ar

Generar y probar: n-tuplas

- **Espacio de soluciones**
  - Trabajaremos sobre la elección de un subconjunto entre n elementos
  - Los elementos son únicos e indivisibles.
  - Un elementos puede estar seleccionado o no.
  - No importa el orden de los elementos seleccionados
- **Función generativa**
  - Generación de las n-tuplas
  - Corresponderá a las restricciones explicitas del problema

Ejemplo práctico: Problema de la mochila

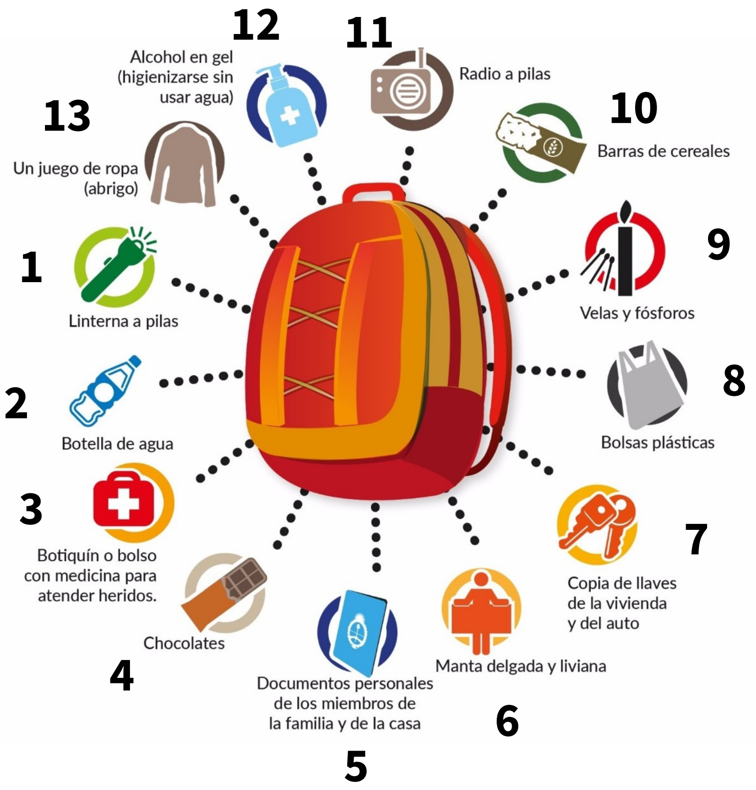
- **Contamos con:**
  - una mochila con una capacidad de K kilos
  - un subconjunto del conjunto E de "n" elementos
- **Cada elemento i tiene:**
  - un peso de k_i kilos
  - un valor de v_i.
- **Queremos seleccionar un subconjunto de E**
  - con el objetivo de maximizar la ganancia.
  - el peso total seleccionado no puede superar la capacidad de la mochila.



Ejemplo práctico: Problema de la mochila

- Podemos asignarle a cada elemento identificador único (un valor entero)
- Asignar un orden a los elementos según su identificador

1	2	3	4	5	6	7	8	9	10	11	12	13
k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}



Ejemplo práctico: Problema de la mochila

- Cualquier subconjunto de los identificadores corresponde a una posible solución al problema
- Una solución es factible si su peso combinado es menor a la capacidad de la mochila
- La solución óptima es aquella solución factible con mayor suma de valor entre sus elementos

1	2	3	4	5	6	7	8	9	10	11	12	13
		X				X			X			
k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}

¿ $k_3 + k_7 + k_{10} \leq K$?

Valor = $v_3 + v_7 + v_{10}$

Ejemplo práctico: Problema de la mochila

- Podemos expresar una posible solución como un vector de "n" posiciones
- Espacio de soluciones: Podemos tener 2^n posibles soluciones.
- Llamamos al proceso de construir estas posibles soluciones como la generación de todas las n-tuplas

1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	1	0	0	0	1	0	0	1	0	0	0

¿ $k_3 + k_7 + k_{10} \leq K$?

Valor = $v_3 + v_7 + v_{10}$

Generación de n-tuplas

- **Se generarán cada uno de los elementos posibles en orden lexicográfico.**
  - Cada vector que representa una posible solución también representa en binario el orden en el que es encontrado y evaluado esa posible solución.
- **Para la generación de estas soluciones podemos utilizar un contador binario.**

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Generación de n-tuplas: Contador binario

Para inicializar:

comenzar con una mochila vacía Ningún elemento seleccionado

Complejidad temporal: O(n)

Obtener la próxima posible solución:

Comenzando por la derecha del vector, mientras que el contenido sea un 1 reemplazar por un 0

Establecer el siguiente elemento del vector en 1.

Complejidad amortizada temporal: O(1)

```
Sea C un vector de n posiciones (soluciones)

Inicializar C:
    Desde x=0 a n
        C[x]=0


Incrementar C:
    Sea pos=n-1
    Mientras C[pos]==1 y pos>0
        C[pos]=0
        Decrementar pos
    Si pos==-1
        Retornar 'fin'  //Overflow!
    C[pos]=1
    retornar C
```

Generación de n-tuplas: Contador binario

Para inicializar:

0	0	0

Obtener la próxima posible solución (Ejemplo):

0	1	1

1	0	0

```
Sea C un vector de n posiciones (soluciones)

Inicializar C:
    Desde x=0 a n
        C[x]=0


Incrementar C:
    Sea pos=n-1
    Mientras C[pos]==1 y pos>0
        C[pos]=0
        Decrementar pos
    Si pos==-1
        Retornar 'fin'   //Overflow!
    C[pos]=1
    retornar C
```

Verificación de la solución

Factibilidad:

Sumar los pesos de los elementos seleccionados (en el vector con un "1") y comparar con la capacidad de la mochila

Complejidad temporal: O(n)

Complejidad espacial: O(1)

Ganancia:

Sumar los valores de los elementos seleccionados.

Complejidad temporal: O(n)

Complejidad espacial: O(1)

```
Es Factible C:
    Sea pesoNecesario = 0
    Desde i=0 hasta n-1
        pesoNecesario += C[i] * k[i]
    retornar ( pesoNecesario <=K)


Ganancia C:
    Sea valorTotal = 0
    Desde i=0 hasta n-1
        valorTotal += C[i] * v[i]
    retornar valorTotal
```

Generar y probar: Problema de la mochila

Unificando:

Se recorren todas las combinaciones posibles

Se verifica por cada uno si es una solución factible

Se verifica por cada uno factible es mejor que la mayor solución previamente encontrada

Complejidad temporal total: $O(n2^n)$

Complejidad espacial total: $O(n)$

```
Sea C un vector representando un subconjunto
de elementos
Inicializar C
Sea maximaGanancia = 0
Sea soluciónMáxima = C

Mientras Incrementar C <> ‘fin’
     Si Es Factible C y Ganancia C >
                                  maximaGanancia
          maximaGanancia = Ganancia C
          soluciónMáxima = C

retornar soluciónMáxima y maximaGanancia
```