1. FUNDAMENTOS DE COMBINATORIA

DEFINICIÓN 1

Dado un conjunto finito A, la cardinalidad de A, denotada por $|A|$, es el número de elementos que tiene el conjunto.

TEOREMA 1: Principio de multiplicación

Supongamos que una tarea se puede dividir en dos tareas consecutivas independientes (es decir, que la realización de una de ellas no influye en la realización de la otra). Si la primera tarea se puede realizar de n_1 maneras distintas y la segunda tarea se puede realizar de n_2 maneras distintas, entonces la tarea considerada inicialmente se puede realizar de $n_1 \cdot n_2$ formas distintas.

⚠ De manera general, si una tarea se puede dividir en m tareas independientes y cada tarea i se puede realizar de n_i maneras distintas, entonces la tarea considerada inicialmente se puede realizar de $n_1 \cdot n_2 \cdot \cdots \cdot n_m$ maneras distintas.

TEOREMA 2: Principio de multiplicación generalizado

Dados $A_1, A_2, \ldots, A_n$ conjuntos finitos, entonces

$$|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdots |A_n|.$$

TEOREMA 3: Principio de adición

Si una tarea se puede realizar de n_1 maneras y una segunda tarea se puede realizar de n_2 maneras, y si no se puede realizar las dos a la vez, entonces hay $n_1 + n_2$ maneras de realizar alguna de ellas.

⚠ Si se tiene m tareas, donde cada tarea i se puede realizar de n_i maneras distintas, y ningún par de tareas se puede realizar a la vez, entonces hay $n_1 + n_2 + \cdots + n_m$ maneras de realizar alguna de ellas.

TEOREMA 4: Principio de adición generalizado

Dados $A_1, A_2, \ldots, A_n$ conjuntos finitos, disjuntos dos a dos, entonces

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i|.$$

TEOREMA 5

Dados A_1, A_2 conjuntos finitos tales que $A_1 \subseteq A_2$, entonces

$$|A_2 \setminus A_1| = |A_2| - |A_1|.$$

TEOREMA 6: Principio de inclusión-exclusión

Dados A_1, A_2 conjuntos finitos, entonces

$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|.$$

TEOREMA 7: Principio de inclusión-exclusión generalizado

Dados $A_1, A_2, \ldots, A_n$ conjuntos finitos, entonces

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1\leq i<j\leq n} |A_i \cap A_j| + \sum_{1\leq i<j<k\leq n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap \cdots \cap A_n|.$$

2. PRINCIPIO DEL PALOMAR

TEOREMA 8: Principio del palomar

Si se busca distribuir $k+1$ objetos en k cajas, existe al menos una caja que contiene al menos dos objetos.

TEOREMA 9: Principio del palomar generalizado

Si se busca distribuir N objetos en k cajas, existe al menos una caja que contiene al menos $\left\lceil \frac{N}{k} \right\rceil$ objetos.

Semestre 2022-1 Mat. Andrés Merino

1. PERMUTACIONES Y COMBINACIONES

DEFINICIÓN 1: Permutación

Dado un conjunto A, una *permutación* de los elementos de A es una función $f\colon A \to A$ biyectiva.

⚠ Dado un A un conjunto no vacío, una permutación de los elementos de A es un reordenamiento de los mismos.

TEOREMA 1

Dado un conjunto finito A con $|A| = n$, el número de permutaciones de elementos de A es $n!$.

DEFINICIÓN 2: $r-$Permutación

Dado un conjunto A, una $r-$permutación elementos de A es cualquier función biyectiva $f\colon B \to B$ con $B \subseteq A$ tal que $|B| = r$.

⚠ Dado un conjunto A, una $r-$permutación elementos de A es un ordenamiento de r elementos distintos de A.

⚠ Al número de $r-$permutaciones de elementos de un conjunto A de n elementos lo denotaremos por $P(n,r)$ y se leerá como *el número de permutaciones de elementos de A tomados de r en r*.

OBSERVACIÓN. Se tiene que $P(n,n) = n!$.

TEOREMA 2

Dado un conjunto A con $|A| = n$, se tiene que

$$P(n,r) = n(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}.$$

DEFINICIÓN 3

Dado un conjunto A con $|A| = n$, una $r-$combinación de elementos de A es cualquier subconjunto B de A con $|B| = r$.

OBSERVACIÓN. Al número de $r-$combinaciones de elementos de un conjunto A

con n elementos lo denotaremos por $C(n,r) = \binom{n}{r}$.

TEOREMA 3

Se tiene que

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n(n-1)(n-2)\cdots(n-r+1)}{r!} = \frac{n!}{(n-r)!r!}.$$

2. COEFICIENTES BINOMIALES

TEOREMA 4: Teorema del binomio

Dados $a, b \in \mathbb{R}$ y $n \in \mathbb{N}$, se tiene que

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

Ejemplos 6.7.2-6.7.5 en Johnsonbaugh.

TEOREMA 5

Dados $n, k \in \mathbb{N}$ con $k \leq n$, se tiene que

$$C(n+1,k) = C(n,k-1) + C(n,k).$$

3. ALGORITMOS PARA GENERAR PERMUTACIONES Y COMBINACIONES

DEFINICIÓN 4

Sean $\alpha = s_1 s_2 \cdots s_p$ y $\beta = t_1 t_2 \cdots t_q$. Se dice que α es menor que β ($\alpha < \beta$), en el orden lexicográfico, si

1. $p < q$ y si, para todo $i \in \{1, \ldots, n\}$, se tiene que $s_i = t_i$; o
2. Existe $i \in \{1, \ldots, n\}$ tal que $s_i \neq t_i$ y para el menor valor de i que cumple con lo anterior, se tiene que $s_i < t_i$.

⚠ Para cadenas de igual longitud en el conjunto $\{1, 2, \ldots, n\}$, el orden lexicográfico es el mismo que el orden usual en $\mathbb{R}$.

Una $r-$combinación $\{x_1, x_2 \ldots, x_r\}$, se representará como una cadena $s_1 s_2 \cdots s_r$, donde $s_1 < s_2 < \cdots < s_r$ y $\{x_1, x_2 \ldots, x_r\} = \{s_1, s_2 \ldots, s_r\}$. Es decir, que la cadena que representa a la $r-$combinación está formada por los elementos de la combinación ubicados en orden creciente.

Dado el conjunto $A = \{1, 2 \ldots, n\}$, si consideramos el orden lexicográfico para las $r-$combinaciones de A, se tiene que la primera $r-$combinación es $12 \cdots n$ y la última será $(n-r+1)(n-r+2) \cdots (n-2)(n-1)(n)$.

3.1 Generación de $r-$combinaciones

A continuación se presenta el código en Python para la generación de todas las $r-$combinaciones de un conjunto de n elementos listadas en orden lexicográfico.

```
1  from math import comb
2  def Combinaciones(r,n):
3      s = [0]*r
4      for i in range(r): #generamos la primera r-combinación
5          s[i] = i+1
6      print(s)
7
8      for i in range( 2 , comb(n,r)  + 1 ): #generamos las combinaciones que restan en orden lexicográfico
9          m = r - 1
10         val_max = n
11         while ( s[m] == val_max ): #buscamos el primer dígito, desde la derecha, que no haya alcanzado su valor máximo
12             m = m - 1
13             val_max = val_max - 1
14         s[m] = s[m] + 1 #incrementamos en 1 el m-ésimo dígto
15         for j in range( m + 1 , r ): #construimos el resto de la cadena
16             s[j] = s[j-1] + 1
17         print(s)
```

3.2 Generación de permutaciones

Para motivar la lógica del algoritmo, usar el ejemplo 6.3.12 de Johnsonbaugh.

A continuación se presenta el algoritmo para generar todas las permutaciones de un conjunto de n elementos.

```
1  from math import factorial
2
3  def permutaciones(n):
4      s = [0]*n
5      for i in range(n): #generamos la primera permutación
6          s[i] = i+1
7      print(s)
8
```

```
9       for i in range( 1 , factorial(n) ): #generamos las permutaciones restantes en orden lexicográfico
10          m = n-2
11          while (s[m] > s[m+1]): #buscamos el primer incremento desde la derecha
12              m = m-1
13          k = n-1
14          while (s[m] > s[k]): #buscamos el primer elemento, desde la derecha, que sea mayor que s[m]
15              k = k-1
16          s[m],s[k] = s[k],s[m] #intercambiamos las posiciones de s[m] y s[k]
17          p = m+1
18          q = n-1
19          while (p < q): #tomamos la cadena desde la posición m+1 hasta el final y la invertimos
20              s[p],s[q] = s[q],s[p]
21              p = p+1
22              q = q-1
23          print(s)
```

Semestre 2022-1 Mat. Andrés Merino

1. PERMUTACIONES Y COMBINACIONES GENERALIZADAS

TEOREMA 1

Dada una colección con n objetos de m tipos distintos, donde hay n_i objetos idénticos de cada tipo i, el número de ordenamientos de la colección es

$$\frac{n!}{n_1!n_2!\cdots n_m!}.$$

TEOREMA 2

Dado un conjunto con k elementos, el número de selecciones no ordenadas de t elementos de A, con repeticiones, es

$$C(k+t-1,k).$$

TEOREMA 3

Dado un conjunto con n elementos, el número de maneras de separar estos elementos en k grupos, sin considerar el orden, es

$$C(n+k-1,k-1).$$

En resumen, tenemos la siguiente tabla:

	Sin repetición	Con repetición
Selecciones ordenadas	$n!$	$\frac{n!}{n_1!\cdots n_m!}$
Selecciones no ordenadas	$C(n,r)$	$C(k+t-1,k)$

2. Relaciones de recurrencia

Definición 1

Dada una sucesión $(x_n)_{n\in\mathbb{N}}$ definida recursivamente, a la ecuación que relaciona al término x_n con sus predecesores, se la denomina *relación de recurrencia* de la sucesión $(x_n)_{n\in\mathbb{N}}$. Si, de manera explícita, se especifica una cantidad finita de términos de la sucesión, diremos que estos valores son las *condiciones iniciales* de la sucesión.

Definición 2

Dada una relación de recurrencia, resolver tal relación consiste en encontrar una fórmula explícita para el $n-$ésimo término de la sucesión, es decir, consiste en hallar una fórmula no recursiva para dicho término.

Definición 3

Una relación de recurrencia homogénea lineal de orden k con coeficientes constantes es una relación de recurrencia de la forma

$$a_n = c_1 a_{n-a} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

don $c_k \neq 0$.

⚠ Una relación de recurrencia homogénea lineal de orden k con coeficientes constantes junto con k condiciones iniciales

$$a_0 = C_0, \quad a_1 = C_1, \quad , \ldots, a_{k-1} = C_{k-1},$$

define de manera única una sucesión $(a_n)_{n\in\mathbb{N}}$.

DEFINICIÓN 1

Función de Ackermann Se define la función de Ackermann de manera recursiva de la siguiente manera:

$$\begin{aligned} A(0,n) &= n+1, && m \in \mathbb{N}; \\ A(m,0) &= A(m-1,1) && m \geq 1; \\ A(m,0) &= A(m-1,A(m,n-1)) && m \geq 1 \quad \wedge \quad n \geq 1. \end{aligned}$$

TEOREMA 1

Dada la relación de recurrencia homogénea lineal de segundo orden con coeficientes constantes

$$a_n = c_1 a_{n-1} + c_2 a_{n-2},$$

se tiene que:

1. Si $(S_n)_{n\in\mathbb{N}}$ y $(T_n)_{n\in\mathbb{N}}$ son soluciones de la relación, entonces $(U)_{n\in\mathbb{N}} = (bS_n + dT_n)_{n\in\mathbb{N}}$, con $b, d \in \mathbb{R}$, también es solución de la relación.

2. Si r es una raíz de $t^2 - c_1 t - c_2 = 0$, entonces la sucesión $(r^n)_{n\in\mathbb{N}}$ es una solución de la relación.

3. Si $a_0 = C_0$ y $a_1 = C_1$, y r_1 y r_2 son raíces distintas de $t^2 - c_1 t - c_2 = 0$, entonces existen constantes b y d tales que, para todo $n \in \mathbb{N}$,

$$a_n = br_1^n + dr_2^n$$

TEOREMA 2

Dada la relación de recurrencia homogénea lineal de segundo orden con coeficientes constantes

$$a_n = c_1 a_{n-1} + c_2 a_{n-2},$$

y las condiciones iniciales $a_0 = C_0$ y $a_1 = C_1$, se tiene que, si las raíces de $t^2 - c_1 t - c_2 = 0$ son iguales a r, entonces existen constantes b y d tales que

$$a_n = br^n + dnr^n$$

para todo $n \in \mathbb{N}$.

DEFINICIÓN 2

Dada una sucesión $(a_n)_{n\in\mathbb{N}}$, se define la sucesión de sumas parciales por

$$\begin{aligned} s_0 &= a_0, \\ s_n &= s_{n-1} + a_n, \qquad n \geq 0. \end{aligned}$$

De aquí, se tiene que

$$s_n = \sum_{k=0}^{n} a_k.$$

PROPOSICIÓN 3. Sean $n \in \mathbb{N}$ y $r \in \mathbb{R}^+$.

- $\displaystyle\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$;
- $\displaystyle\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$;
- $\displaystyle\sum_{k=0}^{n} k^3 = \left(\frac{n(n+1)}{2}\right)^2$.

1. Aplicaciones al análisis de algoritmos

1.1 Ordenamiento por selección

La siguiente implementación ordena una lista s dada.

```python
def ordenar(s):
    n = len(s)
    # si la lista tiene solo un elemento, ya está ordenada
    if n == 1:
        return s
    # caso contrario, colocamos el elemento máximo al final
    else:
        # indice donde se encuentra el máximo
        m = 0
        # buscamos el máximo
        for i in range( 2 , n ):
            # si se encuetra un elemento más grande, actualizamo
                el índice
            if s[i] > s[m]:
                m = i
        # colocamos el elemento más grande al final
        s[m], s[n-1] = s[n-1], s[m]
        # ordenamos la lista sin el último elemento
        t = ordenar(s[:-1])
        # añadimos el último elemento
        t.append(s[n-1])
        return t
```

2. Funciones generatrices

Definición 3

Dada una sucesión $(a_n)_{n\in\mathbb{N}}$, a la serie formal

$$G(x) = a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{k=0}^{\infty} a_n x^n$$

se la llama *función generatriz* de la sucesión.

1. NÚMEROS PSEUDOALEATORIOSOS

DEFINICIÓN 1: Método de congruencia lineal

Dados $a, c, m, s \in \mathbb{N}$ tales que

$$2 \leq a < m, \qquad c < m \qquad \text{y} \qquad s < m$$

La sucesión definida por

$$\begin{aligned} x_0 &= s, \\ x_n &= (ax_{n-1} + c) \mod m \end{aligned}$$

es una sucesión de número pseudoaleatorios. Al valor de s se lo conoce como la semilla.

Se suele utilizar

- $m = 2^{31} - 1$
- $a = 7^5$
- $c = 0$

2. TEORÍA DE LA PROBABILIDAD

DEFINICIÓN 2: Espacio muestral

Un conjunto Ω se dice que es un **espacio muestral** si está formado por todos los resultados posibles de un experimento aleatorio.

DEFINICIÓN 3: Suceso

Sea Ω el espacio muestral de experimento aleatorio y $S \subseteq \Omega$. Se dice que S es un **suceso** o un **evento** del experimento aleatorio.

DEFINICIÓN 4: Probabilidad de un suceso

Dado Ω el espacio muestral finito de un experimento donde todos los resultados tienen igual probabilidad y S un suceso. La probabilidad de S se define como:

$$\mathbb{P}(S) = \frac{|S|}{|\Omega|},$$

donde,

- $\mathbb{P}(S)$: probabilidad de S,
- $|\Omega|$: número de elementos de Ω y
- $|S|$: número de elementos de S.

TEOREMA 1: Complemento

Sea S un suceso del espacio muestral finito Ω, la probabilidad del complemento de S, S^c, está dado por

$$\mathbb{P}(S^c) = 1 - \mathbb{P}(S).$$

TEOREMA 2: Unión

Sean S_1, S_2 dos sucesos del espacio muestral finito Ω, entonces

$$\mathbb{P}(S_1 \cup S_2) = \mathbb{P}(S_1) + \mathbb{P}(S_2) - \mathbb{P}(S_1 \cap S_2).$$

DEFINICIÓN 5: Axiomas de la probabilidad

Sea Ω un espacio muestral. Una función de probabilidad es una función del conjunto de todos los eventos al intervalo $[0, 1]$ que satisface que

- $\mathbb{P}(\Omega) = 0$ y
- si A y B son conjuntos disjuntos, entonces $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)$.

TEOREMA 3

Sea Ω un espacio muestral finito, $\mathbb{P}$ una función de probabilidad y S un suceso. Se tiene que

$$\mathbb{P}(S) = \sum_{\omega \in S} \mathbb{P}(\{\omega\}).$$

3. Variable aleatoria

Definición 6: Variables aleatorias

A una función que asigna a cada elemento del espacio muestral un número real, se la conoce como **variable aleatoria**.

Definición 7

Dado un espacio muestral Ω, una función de probabilidad $\mathbb{P}$ y una variable aleatoria $X\colon \Omega \to \mathbb{R}$, se define la *probabilidad de que X valga r* por

$$\mathbb{P}(X = r) = \mathbb{P}(\{\omega \in \Omega : X(\omega) = r\}).$$

Teorema 4

Dado un espacio muestral finito Ω, una función de probabilidad $\mathbb{P}$ y una variable aleatoria $X\colon \Omega \to \mathbb{R}$, se tiene que

$$\mathbb{P}(X = r) = \sum_{\substack{\omega \in \Omega \\ X(\omega) = r}} \mathbb{P}(\{\omega\}).$$

4. Valor esperado

Definición 8

Dado un espacio muestral finito Ω, una función de probabilidad $\mathbb{P}$ y una variable aleatoria $X\colon \Omega \to \mathbb{R}$, el **valor esperado** (o esperanza) de X es

$$\mathbb{E}(X) = \sum_{\omega \in \Omega} \mathbb{P}(\{\omega\}) \cdot X(\omega).$$

Cuando el espacio muestral Ω tiene n elementos, es decir,

$$\Omega = \{x_1, x_2, x_3, \ldots, x_n\},$$

el valor esperado se expresa como

$$\mathbb{E}(X) = \sum_{i=1}^{n} \mathbb{P}(\{x_i\}) \cdot X(x_i).$$

TEOREMA 5

Dado un espacio muestral finito Ω, una función de probabilidad $\mathbb{P}$ y una variable aleatoria $X\colon \Omega \to \mathbb{R}$, se tiene que

$$\mathbb{E}(X) = \sum_{r \in X(\Omega)} r \cdot \mathbb{P}(X = r).$$

TEOREMA 6: Complejidad computacional

Sean a_j, con $j \in \{1, 2, \cdots, n\}$ las posibles entradas de un algoritmo y sea X la variable aleatoria que asigna a cada a_j el número de operaciones realizadas por el algoritmo cuando este tiene como entrada a a_j. Entonces la complejidad media del algoritmo es

$$\mathbb{E}(X) = \sum_{j=1}^{n} \mathbb{P}(\{a_j\}) \cdot X(a_j).$$

Semestre 2022-1 Mat. Andrés Merino

1. GRAFOS

DEFINICIÓN 1: Grafo

Sean V un conjunto finito y A un subconjunto de pares no ordenados de V. Se dice que $G = (V, A)$ es un grafo, al conjunto V se lo denomina vértices de grafo y al conjunto A se lo denomina aristas del grafo. Para $e \in A$, si $e = \{u, v\}$, con $u, v \in V$, se dice que la arista e une el vértice u con el vértice v.

DEFINICIÓN 2: Vértices adyacentes

Sean $G = (V, A)$ un grafo y $u, v \in V$. Se dice que u y v son adyacentes en G si existe $e \in A$ tal que e une u con v.

DEFINICIÓN 3: Grade de un vértice

Sea $G = (V, A)$ un grafo. Para $u \in V$, se define el grado de u, denotado por $\text{grad}(u)$, por el número de aristas en A que contienen a u como uno de sus extremos (en caso de que existan lazos, estos cuentan por dos).

DEFINICIÓN 4: Grafo dirigido

Sean V un conjunto finito y A un subconjunto de pares ordenados de V. Se dice que $G = (V, A)$ es un grafo dirigido o digrafo, al conjunto V se lo denomina vértices de grafo y al conjunto A se lo denomina arcos del grafo. Para $e \in A$, si $e = (u, v)$, con $u, v \in V$, se dice que el arco e empieza en el vértice u y termina en el vértice v.

Sean $G = (V, A)$ un grafo dirigido y $u \in V$. Se denota por $\text{suc}(u)$ al conjunto de todos los sucesores de u, es decir,

$$\textbf{suc}(u) = \{v \in V : (u, v) \in V\}.$$

DEFINICIÓN 5: Grado de entrada y salida

Sea $G = (V, A)$ un grafo dirigido. Para $u \in V$, se define el grado de salida de u, denotado por $\text{grad}_s(u)$, por el número de aristas en A que contienen a u como su extremo inicial. De manera similar, se define el grado de entrada de u, denotado por $\text{grad}_e(u)$, por el número de aristas en A que contienen a u como su extremo final.

Dados un grafo dirigido $G = (V, A)$ y $v \in V$, si $\text{grad}_s(v) = 0$, se dice que v es un sumidero, y si $\text{grad}_e(v) = 0$, se dice que v es una fuente.

DEFINICIÓN 6: Camino

Sean $G = (V, A)$ un grafo y $a, b \in V$. Un camino entre a y b es una secuencia

$$C = (v_0, e_1, v_1, e_2, v_2, \ldots, e_{n-1}, v_{n-1}, e_n, v_n)$$

tal que $v_0, v_1, \ldots, v_n \in V$, $e_1, \ldots, e_n \in A$, $v_0 = a$, $v_n = b$ y para todo $k \in \{1, 2, \ldots, n\}$, e_k conecta v_{k-1} con v_k. Al número de aristas que contiene el camino se lo denomina longitud del camino y se lo denota por $\text{long}(C)$.

Dados $G = (V, A)$ un grafo, $a, b \in V$ y α un camino entre a y b, a α también se lo representará únicamente por sus aristas.

DEFINICIÓN 7: Camino cerrado o circuito

Sean G un grafo y α un camino en G. Se dice que α es cerrado o que es un circuito si su vértice inicial y final coinciden.

DEFINICIÓN 8: Camino simple

Sean G un grafo y α un camino en G. Se dice que α es simple si todos sus vértices son diferentes.

DEFINICIÓN 9: Recorrido

Sean G un grafo y α un camino en G. Se dice que α es un recorrido si todas sus aristas son diferentes.

DEFINICIÓN 10: Circuito simple

Sean G un grafo y α un circuito en G. Se dice que α es un circuito si es un recorrido.

DEFINICIÓN 11: Ciclo

Sean G un grafo y α un camino en G. Se dice que α es un ciclo si todos sus vértices son diferentes menos el primero y último que son iguales. Un ciclo de longitud k se denomina un k-ciclo.

DEFINICIÓN 12: Grafo conexo

Sea $G = (V, A)$ un grafo. Se dice que G es conexo si para todo $u, v \in V$, existe un camino entre u y v.

DEFINICIÓN 13: Vértices conectados

Sean $G = (V, A)$ un grafo y $u, v \in V$. Se dice que u y v están conectados si existe un camino entre u y v.

DEFINICIÓN 14: Distancia entre vértices

Sean $G = (V, A)$ un grafo conexo y $u, v \in V$. Se define la distancia de u a v por

$$d(u, v) = \text{mín}\{\text{long}(C) : C \text{ es un camino entre } u \text{ y } v\}.$$

DEFINICIÓN 15: Diámetro de un grafo

Sea $G = (V, A)$ un grafo conexo. El diámetro de G se define por

$$\text{diam}(G) = \text{máx}\{d(u, v) : u, v \in V\}.$$

2. FLUJO SOBRE GRAFOS

DEFINICIÓN 16: Grafos ponderados

Sean $G = (V, A)$ un grafo y $p\colon A \to [0, +\infty[$. A p se la llama función de pesos o de ponderaciones del grafo G. A

$$\sum_{e \in A} p(e)$$

se lo denomina peso del grafo. Al grafo ponderado se lo denota por (V, A, p)

DEFINICIÓN 17: Peso de un camino

Sean G un grafo, p una función de pesos de G y C un subgrafo de G o un camino en G. El peso de C se define por

$$\sum_{e \in B} p(e),$$

donde B es el conjunto de aristas de C.

DEFINICIÓN 18: Red de transporte

Sean $G = (V, A, c)$ un grafo dirigido ponderado. Se dice que es una red de transporte si tiene una sola fuente y un solo sumidero. Al peso de cada arista se lo conoce como capacidad de la arista.

DEFINICIÓN 19: Flujo

Sea $G = (V, A, c)$ una red de transporte. Un flujo sobre la red es una función $f\colon A \to [0, +\infty[$ tal que

- $f(u, v) \leq c(u, v)$ para todo $(u, v) \in A$,
- para todo vértice u que no es fuente ni sumidero, se tiene que

$$\sum_{(v,u)\in A} f(v, u) = \sum_{(u,x)\in A} f(u, w).$$

TEOREMA 1

Sean $G = (V, A, c)$ una red de transporte y f un flujo sobre la red. Si a es la fuente de la red y z el sumidero, entonces se tiene que

$$\sum_{(a,v)\in A} f(a, v) = \sum_{(u,z)\in A} f(u, z).$$

TEOREMA 2

Sean $G = (V, A, c)$ una red de transporte, f un flujo sobre la red y a la fuente de la red. El valor de

$$\sum_{(a,v)\in A} f(a, v)$$

se lo llama valor del flujo f.

1. GENERACIÓN DE ÁRBOLES DE EXPANSIÓN

La generación de árboles de expansión está ligada a algunos algoritmos de búsqueda sobre un grafo. Analizaremos estos algoritmos y su implementación para la generación de árboles de expansión.

La búsqueda sobre un grafo se refiere a un mecanismo para visitar cada vértice de un grafo de una manera secuencial.

1.1 Búsqueda en anchura

A continuación presentamos el algoritmo de búsqueda en anchura (BFS por sus siglas en inglés *Breadth First Search*), el cual tiene como objetivo ir marcando cada vértice visitado (incluirlo en un conjunto).

Dados $G = (V, A)$ un grafo y $v \in V$, el pseudocodigo para búsqueda en anchura es:

```
1  Función BFS(G,v):
2      Q := (v)                          /* Vértices que se visitarán */
3      M := {v}                          /* Vértices ya visitados */
4      mientras Q ≠ ∅ hacer
5          u := Ultimo(Q)                /* Vértice actual */
6          Q := SinUltimo(Q)
7          para w ∈ V[u] hacer
8              si w ∉ M entonces
9                  M := M ∪ {w}
10                 Q := Concatenar(w,Q)
```

El código en Python para la función que realiza búsqueda en anchura es:

```python
def BFS(G,v):
    Q = [v]
    M = {v}
    while Q != []:
        u = Q[-1]
        Q = Q[:-1]
        for w in G[u]:
            if w not in M:
                M.add(w)
                Q.insert(0,w)
```

Con esto, podemos generar un árbol de expansión marcando cada arista que vamos recorriendo en el algoritmo BFS.

Dados $G = (V, A)$ un grafo y $v \in V$, el pseudocodigo para la función que devuelve un árbol de expansión de G con raíz en v utilizando BFS es:

```
1  Función ArbolExpansionBFS(G,v):
2      Q := (v)                                  /* Vértices que se visitarán */
3      M := {v}                                  /* Vértices ya visitados */
4      T := ∅                                    /* Aristas del árbol resultante */
5      mientras Q ≠ ∅ hacer
6          u := Ultimo(Q)                        /* Vértice actual */
7          Q := SinUltimo(Q)
8          para w ∈ V[u] hacer
9              si w ∉ M entonces
10                 M := M ∪ {w}
11                 T := T ∪ {{w, u}}
12                 Q := Concatenar(w,Q)
13     devolver (M,T)
```

1.2 Búsqueda en profundidad

A continuación presentamos el algoritmo de búsqueda en profundidad (DFS por sus siglas en inglés *Deep First Search*), el cual tiene como objetivo ir marcando cada vértice visitado (incluirlo en un conjunto).

Dados $G = (V, A)$ un grafo y $v \in V$, el pseudocodigo para búsqueda en profundidad es:

```
1  Función DFS(G,v):
2      Q := (v)                                  /* Vértices que se visitarán */
3      M := {v}                                  /* Vértices ya visitados */
4      mientras Q ≠ ∅ hacer
5          u := Ultimo(Q)                        /* Vértice actual */
6          si V[u] ⊆ M entonces
7              Q := SinUltimo(Q)
8          en otro caso
9              para w ∈ V[u] hacer
10                 si w ∉ M entonces
11                     M := M ∪ {w}
12                     Q := Concatenar(Q,w)
13                     Salir;
```

El código en Python para la función que realiza búsqueda en anchura es:

```python
def DFS(G,v):
    Q = [v]
    M = {v}
    while Q != []:
        u = Q[-1]
        if all(a in M for a in G[u]):
            Q = Q[:-1]
        else:
            for w in G[u]:
                if w not in M:
                    M.add(w)
                    Q.append(w)
                    break
```

Con esto, podemos generar un árbol de expansión marcando cada arista que vamos recorriendo en el algoritmo DFS.

Dados $G = (V, A)$ un grafo y $v \in V$, el pseudocodigo para la función que devuelve un árbol de expansión de G con raíz en v utilizando DFS es:

```
Función DFS(G,v):
    Q := (v)                                  /* Vértices que se visitarán */
    M := {v}                                  /* Vértices ya visitados */
    T := ∅                                    /* Aristas del árbol resultante */
    mientras Q ≠ ∅ hacer
        u := Ultimo(Q)                        /* Vértice actual */
        si V[u] ⊆ M entonces
            Q := SinUltimo(Q))
        en otro caso
            para w ∈ V[u] hacer
                si w ∉ M entonces
                    M := M ∪ {w}
                    T := T ∪ {{w,u}}
                    Q := Concatenar(Q,w)
                    Salir;
    devolver (M,T);
```

1.3 Generación de árboles de expansión mínima - Algoritmo de Kruskal

Dado $G = (V, A)$ un grafo ponderado donde el conjunto A está ordenado por el peso de cada arista de manera creciente, el pseudocodigo para el algoritmo de Kruskal es:

```
1  Función Kruskal(G):
2      T := ∅                                   /* Aristas del árbol resultante */
3      para v ∈ V hacer
4          C_v := {v}                           /* Componente del vértice v */
5      para {u, v} ∈ A hacer
6          si C_u ≠ C_v entonces
7              T := T ∪ {{u, v}}
8              N_C := C_u ∪ C_v                 /* Nueva componente de los vértices */
9              para w ∈ N_C hacer
10                 C_w := N_C
11     devolver T;
```