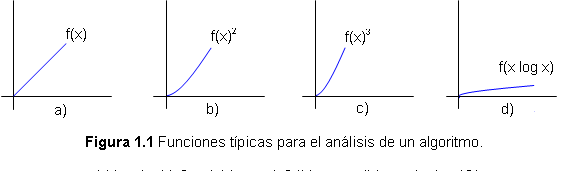
Estructuras de datos avanzadas

Apuntes del curso

Complejidad algorítmica

La complejidad de un algoritmo o complejidad computacional, estudia los recursos y esfuerzos requeridos durante el cálculo para resolver un problema los cuales se dividen en: tiempo de ejecución y espacio en memoria.



La complejidad de espacio, se refiere a la memoria que utiliza un programa para su ejecución; es decir el espacio de memoria que ocupan todas las variables propias del programa.

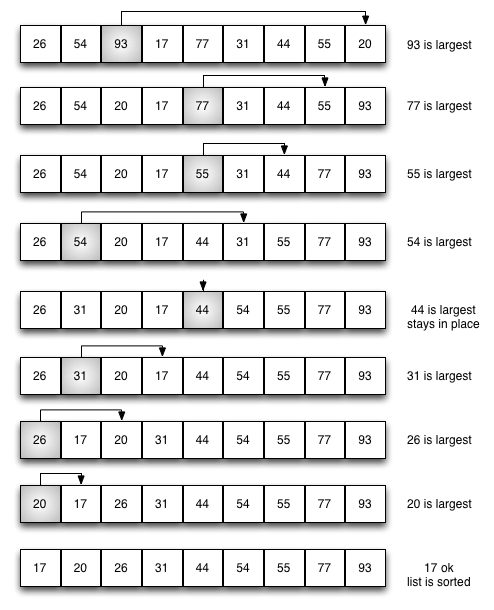
Para calcular la memoria estática, se suman la cantidad de memoria que ocupa cada una de las variables declaradas en el programa.

El cálculo de la memoria dinámica, no es tan simple ya que depende de cada ejecución del programa o algoritmo y el tipo de estructuras dinámicas que se estén utilizando.

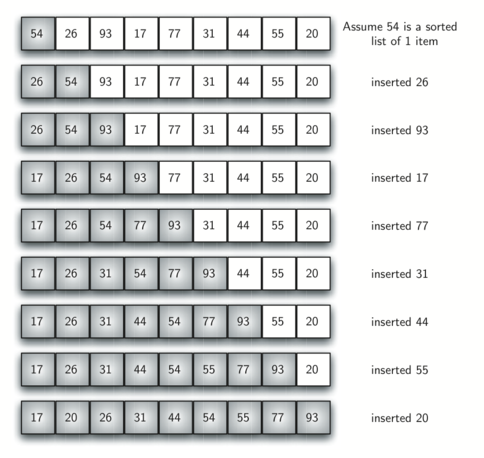
|  |  |  |
| --- | --- | --- |
| O(1) | constante | Todos aquellos algoritmos que responden en un tiempo constante, sea cual sea la talla del problema. Son los que aplican alguna fórmula sencilla, por ejemplo, hallar el máximo de dos valores |
| O(log n) | logarítmico | Los que el tiempo crece con un criterio logarítmico, independientemente de cuál sea la base mientras ésta sea mayor que 1. Por eso, normalmente, ni siquiera se indica la base. No suelen ser muchos, y normalmente están bien considerados, ya que implican que un bucle realiza **menos** iteraciones que la talla del problema, lo cual no suele ser muy común. Por ejemplo, la búsqueda dicotómica en un vector ordenado. |
| O(n) | lineal | El tiempo crece linealmente con respecto a la talla. Por ejemplo, encontrar el máximo de un vector de talla n. |
| O(n·log(n)) | enelogarímico, loglineal, linearítmico o simplememente n por logaritmo de n | Éste orden tiene muchos nombres. Es un orden relativamente bueno, porque la mayor parte de los algoritmos tienen un orden superior. En éste orden está, por ejemplo, el algoritmo de ordenación Quicksort, o la transformada rápida de Fourier. |
| O(nc), con c>1 | polinómico | Aquí están muchos de los algoritmos más comunes. Cuando c es 2 se le llama **cuadrático**, cuando es 3 se le llama **cúbico**, y en general, polinómico. Intuitivamente podríamos decir que éste órden es el último de los aceptables (siempre y cuando c sea relativamente bajo). A partir del siguiente, los algoritmos son complicados de tratar en la práctica cuando n es muy grande. |
| O(cn), con c>1 | exponencial | Aunque pudiera no parecerlo, es mucho peor que el anterior. Crece muchísimo más rápidamente. |
| O(n!) | factorial | Es el típico de aquellos algoritmos que para un problema complejo prueban todas las combinaciones posibles. |
| O(nn) | combinatorio | Tan intratable como el anterior. A menudo no se hace distinción entre ellos.| |

Algoritmos de ordenamiento

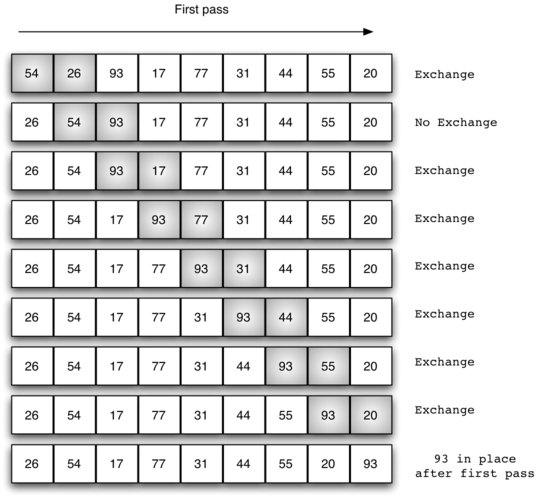
**Selección Directa:** arreglo con números buscar el mas chiquito y ponerlo en la primera posición



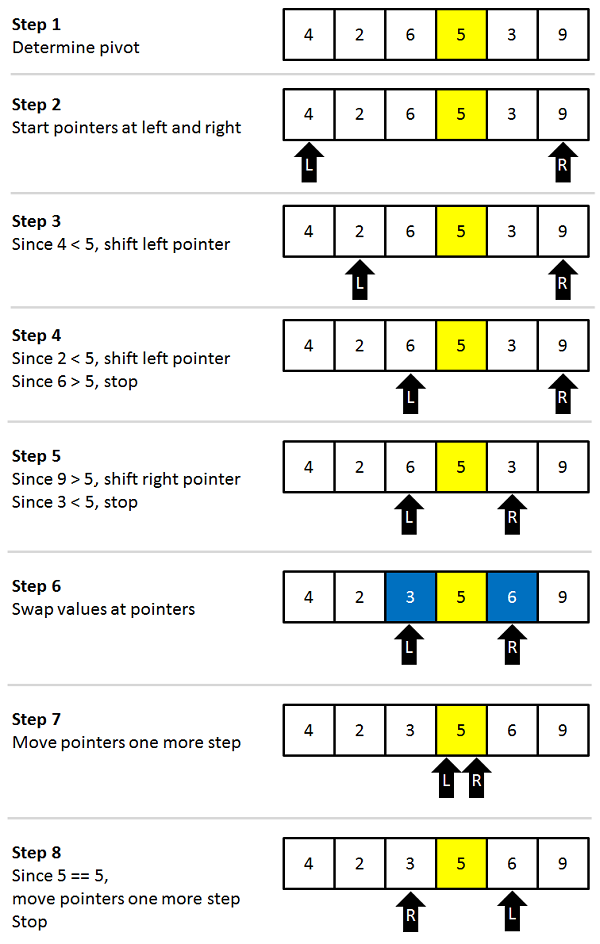
**Insertion Sort:** Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor



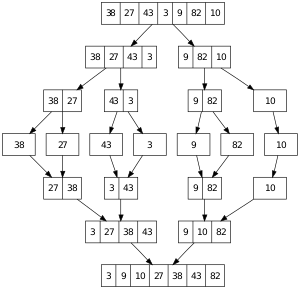
Bubble Sort: Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente



**Quick Sort:** Elegir un pivote poner los números menores a la derecha y los mayores a la izquierda



Merge Sort: dividir a la mitad hasta que solo queden dos números y ordenarlos



Arboles binarios

**Nodos:** Se le llama Nodo a cada elemento que contiene un Árbol.

**Nodo Raíz:** Se refiere al primer nodo de un Árbol, Solo un nodo del Árbol puede ser la Raíz.

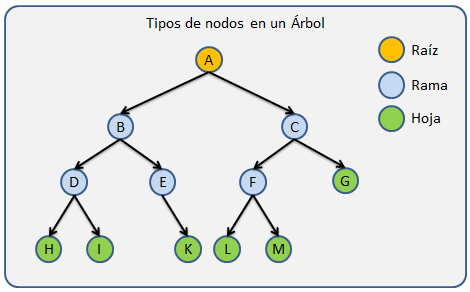
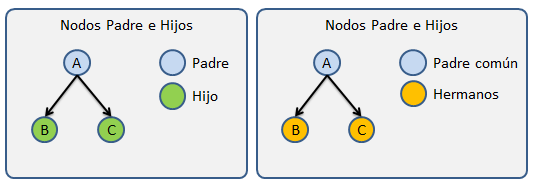
**Nodo Padre:** Se utiliza este termino para llamar a todos aquellos nodos que tiene al menos un hijo.

**Nodo Hijo:** Los hijos son todos aquellos nodos que tiene un padre.

**Nodo Hermano:** Los nodos hermanos son aquellos nodos que comparte a un mismo padre en común dentro de la estructura.

**Nodo Hoja:** Son todos aquellos nodos que no tienen hijos, los cuales siempre se encuentran en los extremos de la estructura.

**Nodo Rama:** Estos son todos aquellos nodos que no son la raíz y que ademas tiene al menos un hijo.

**Grafo** G(V,E) es un conjunto de vértices o nodos V={V1,…Vn} y de aristas (o

ligas) E donde cada arista es una pareja de vértices(Va,Vb)

**Arbol** es un grafo anexo y aciclico en donde un y solo un nodo es designado como la raíz

**Convexo:** un grafo es convexo si para cualquier pareja de vértices Va, Vb existe un conector

(todos están unidos)

**Camino(**Va, Vb) es una secuencia de aristas en E<Va,…Vb>

Acicliclo: que para todo Vi∍V no exista un camino (V0,V1) de longitud mayor o igual a 1

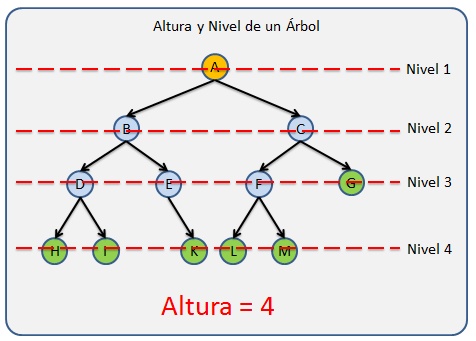
**Nivel:** Nos referimos como nivel a cada generación dentro del árbol.

* Un árbol vacío tiene 0 niveles
* El nivel de la Raíz es 1
* El nivel de cada nodo se calculado contando cuantos nodos existen sobre el, hasta llegar a la raíz + 1, y de forma inversa también se podría, contar cuantos nodos existes desde la raíz hasta el nodo buscado + 1.

**Altura** es la longitus del camino mas largo de la raíz a un hijo

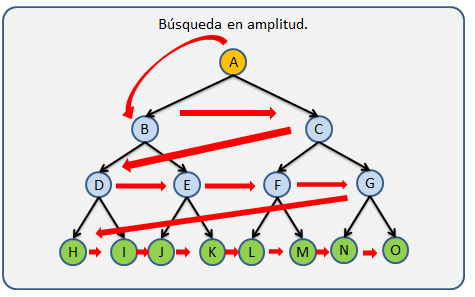
Orden de un árbol es el máximo numero de hijos que puede tener un nodo

Arbol: el conjunto vacio es un árbol. Un nodo conectado a cero o mas arboles es un árbol

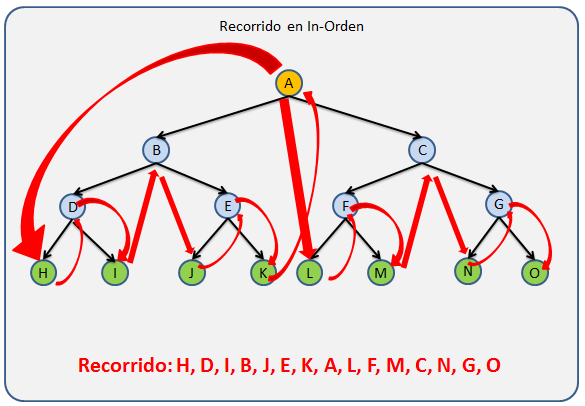


**Recorrido sobre arboles**

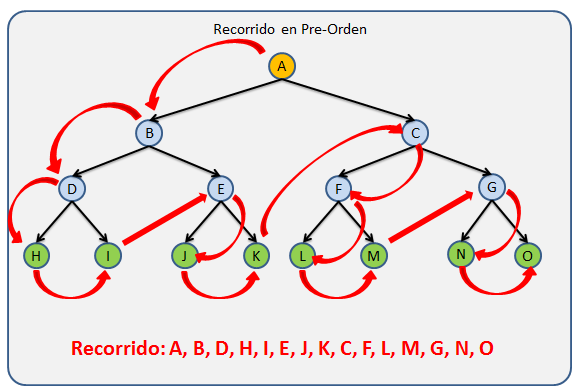
* Busqueda en amplitud: Se recorre primero la raíz, luego se recorren los demas nodos ordenados por el nivel al que pertenecen en orden de Izquierda a derecha. O(n)

****

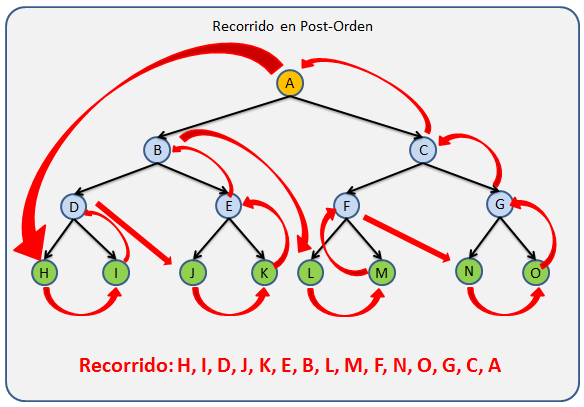
* Inorder: zig-zag de arriba abajo O(n)



* PreOrder: alrededor O(n)



* PostOrder: triángulos O(n)



Arboles Rojo Negro & Arboles AVL

Árboles binarios de búsqueda balanceados:

\* Los más notables son árboles rojo-negro y árboles AVL.

-Su objetivo es que la altura del árbol sea O(log(n)).

\* Árboles AVL (son árboles binarios de búsqueda):

\* Básicamente el criterio para leer un árbol balanceado es que, para cada nodo del arbol, la diferencia en alturas entre su sub-árbol izquierdo y su sub-árbol derecho sea, a lo más, de |1|.

\* Las operaciones de insertado y borrado en AVL son iguales a un ABB, excepto que, después de insertar o borrar, debemos revisar si el árbol sigue balanceado, y, en caso de que no, balancearlo.

\* Las operaciones para reestructurar el árbol se llaman rotaciones y hay cuatro tipos:

1) Rotación Izquierda-Izquierda.

2) Rotación Izquierda-Derecha.

3) Rotación Derecha-Derecha.

4) Rotación Derecha-Izquierda.

\* Cada nodo de un árbol AVL tiene asociado un "factor de equilibrio", que es la resta de la altura de su sub-árbol Derecho menos la altura de su sub-árbol izquierdo.

\* Factor de equilibrio dictará la rotación necesaria para balancear el árbol.

fe=altura(der)-altura(izq). Si fe<-1 será necesaria la rotación tipo 1.

Si fe<-1 será necesaria la rotación tipo 1.

* Un árbol AVL mantiene la atura del árbol en O(logn)
* Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa.
* La estrategia es garantizar que para cada nodo el árbol la diferencia de las alturas de su subárbol izquierdo y el subárbol derecho sea a lo mas de uno
* Para cada operación de inserción o borrado se verifica que no se viole esta propiedad y si se viola se remueve a través de operaciones de rotación
* Para esto vamos a:

1. Modificar la clase nodo para incluir apuntador al padre
2. Incluir un atributo que contenga el factor de equilibrio del nodo. El factor de equilibrio e si
3. Altura(sub-arbol derecho)-Altura(subárbol izquierdo)

* Rotaciones:
  + Izquierda-izquierda
  + Derecha-derecha
  + Izquierda-derecha
  + Derecha-izquierda

fe()=2=Altura()-Altura(A)

2=N-1-Altura(A)

Altura(A)=N-1-2=N-3

Inserción en árboles AVL:

1. El nuevo elemento se inserta como hoja de la misma forma que en un árbol BB.
2. Una vez insertado, “navegamos” hacia la raíz actualizando los factores de equilibrio y rotando cuando sea necesario.

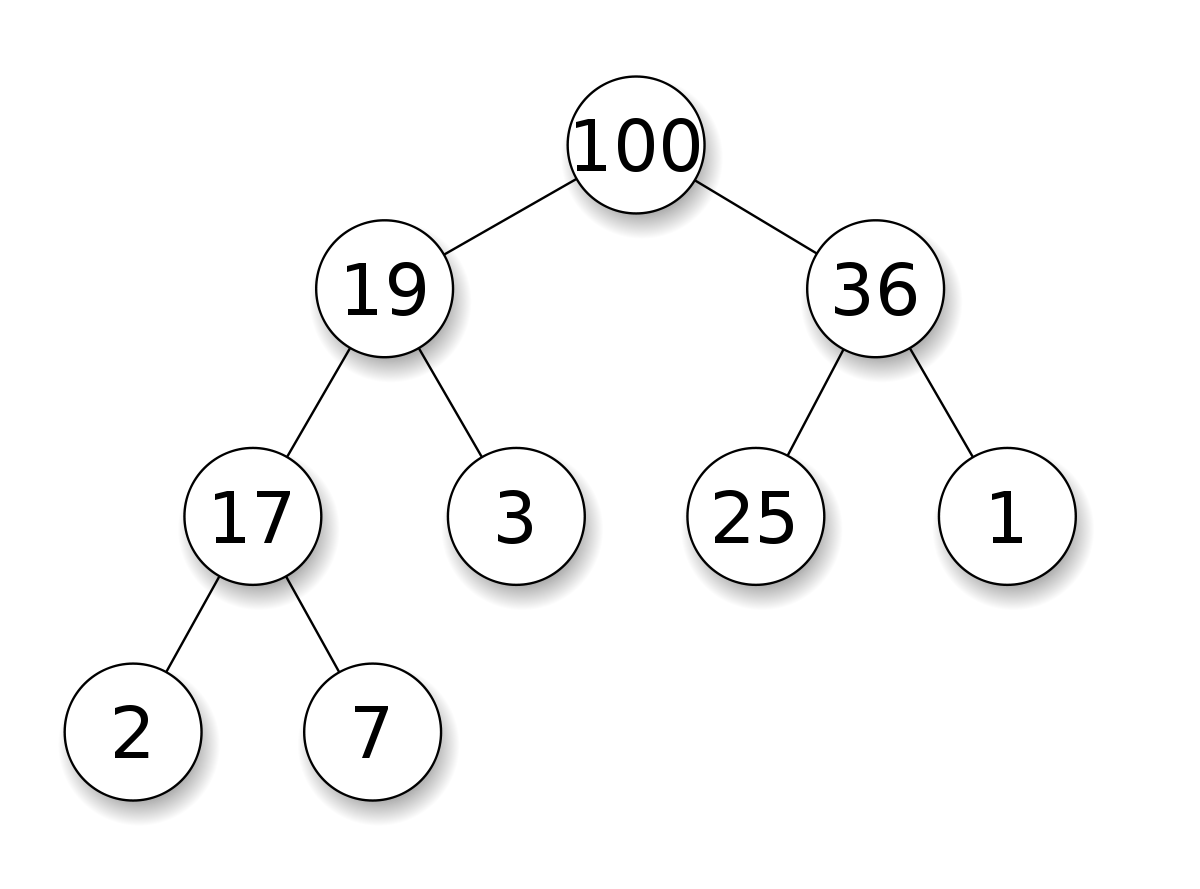
2

1

-1

Heaps

* Son arboles binarios(no de busqueda)
* Son Arboles completos: están balanceados todos los niveles están llenos menos posiblemente el ultimo
* Todas las hojas están recargadas a la izquierda, esto es no existe un nodo con hijo derecho sin que tenga hijo izquierdo, y un nodo no puede tener hijo izquierdo si los nodos de su nivel que esta a su izquierda no tienen hijos derechos
* Hay min-heaps y max-heaps
  + Para un min-heap se cumple que para todo nodo en el árbol el elemento ahí almacenado es menor o igual a todos los elementos almacenados en sus descendientes
  + Un max-heap almacena el elemento
* Operaciones
  + Inserta
  + buscaMin
  + buscaMax



# Arboles multivia

Son una generalización de arboles de búsqueda binarios a no binarios. Los nodos pueden tener mas de 2 hijos pero se sigue conservando en el árbol

Motivación

Podemos guardar mas de un elemento por nodo

El propósito de esto es aprovechar al máximo las lecturas de memoria

Los arboles B son arboles multivia

Arboles 2-3

* + En los arboles 2-3 hay dos tipos de nodo
    - 2-nodo: almacenan un dato E y pueden tener 0 o 2 hijos. Los elementos almacenados en su sub-arbol izquierdo son menores que E y los del subárbol derecho son mayor
    - 3-nodo: almacenan dos datos E1 E2 con E1<E2 puede tener 0 o 3 hijos. Los elementos del subárbol izquierdo son menor a E1. Los elementos del subárbol medio son mayores a E1 y menores A E2. Los elementos del subárbol derecho sea mayor a E1
      * Todos los hijos están al mismo nivel

Insertar

1. Busco la hoja donde va
2. Si la hoja es 2-nodo cambio a 3-nodo
3. Si es 3-nodo lo parto en dos y la raíz es el de en medio

<http://gph.is/2yqGHZY>

borrar

# Skip List

Es una estructura de datos probabilística

|  |  |  |
| --- | --- | --- |
|  | Esperado | Peor de los casos |
| Insercion | O(logn) | O(n) |
| Borrado | - | O(n) |
| Busqueda | - | O(n) |

2↔6↔10↔14↔16↔18↔20↔null O(N)

↔

↔

2 ↔ 10 ↔ 16 ↔ 20 ↔null

O(N/2)

↔

↔

2 ↔ 16 ↔ null

↔

↔

O(N/4)

2 ↔ null O(N/8)

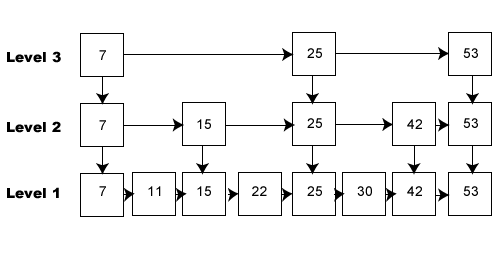


TABLA HASH

Objetivo

Buscamos una estructura de datos en donde en donde instertar buscar tome tiempo constante O(1)

El costo es como siempre la referencia en datos

Estrategia

Que la ubicación de un dato este dada por una función del mismo dato

La tabla se implementa con un arreglo

La función se conoce como función de hash Es una función matemática que toma un argumento de algún tipo de datos predefinido y regresa un entero

Una tabla de hash toma la llave y arroja la posición en la tabla a almacenar el valor. Son muy difíciles de encontrar porque normalmente no conocemos cuantos son los datos que se van a almacenar

Una colision se da cuando dos llaves ocasionan que dos datos tengan el mismo índice de tabla queremos minimizarlas

Caracteristicas

Regresa un entero

Que mapee uniformemente los valores

Deterministas

Que se calcule en O(1)

ALGUNAS FUNCIONES DE HASH:

Extracción : tomar solo parte del dato. Ej los dígitos del RFC

Doblado: Fragmenta la llave si es numérica y hace XOR entre los fragmentos

Exponencial: Convertir a numero elevar al cuadrado y extraer bits de un nodo(se distribuye uniformemente)

Colision🡪dos llaves que dirijen a la misma celda

Buena función hash minimizar las colisiones (para eso la tabla debe de ser grande)

Exploracion lineal 🡪 visita la siguiente casilla. Si esta libre realiza la inserción. Si no, visita la siguiente casilla

Exploracion cuadrática 🡪 visita la casilla i^2 posiciones mas alla. Si esta ñibre inserta sino lo repite

Encadenamiento enlazado 🡪 en cada posición de la tabla se mantiene una lista enlazada en las que se van insertando los elementos

Rehashing🡪aumenta el tamaño de la tabla

Compuesta por dos partes

Mapeo del key🡪hash

Mapeo del dato🡪compresion

Bloom Filter

* Es una estructura de datos probabilística
* Operaciones: insertar y buscar
* Esta es la probabilidad de tener falsos positivos (pensar que algo esta cuando en realidad no esta)
* No hay falsos negativos
* Muy eficiente en espacio
* Se utiliza para almacenar datos raros

Implementacion

* Arreglo booleano de tamaño n
* K formas de hash independientes

Insertar

* Calcular las posiciones en el filtro arrojadas por las h formas de hash y por las k formas de hash y por su valor en true

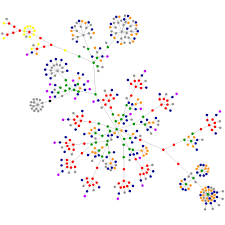
Borrar

* Calcular las posiciones en el filtro arrojados por la h función hash si todos los k valores almacenados son true regresa trye.
* Regresa false de otro modo

Grafos

**Grafo**

Es un conjunto de puntos (nodos o vetices) unidos por líneas (arcos o aristas)



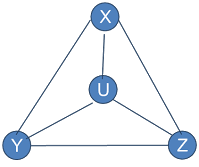
G=(V,E)

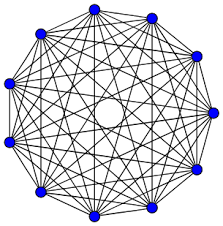
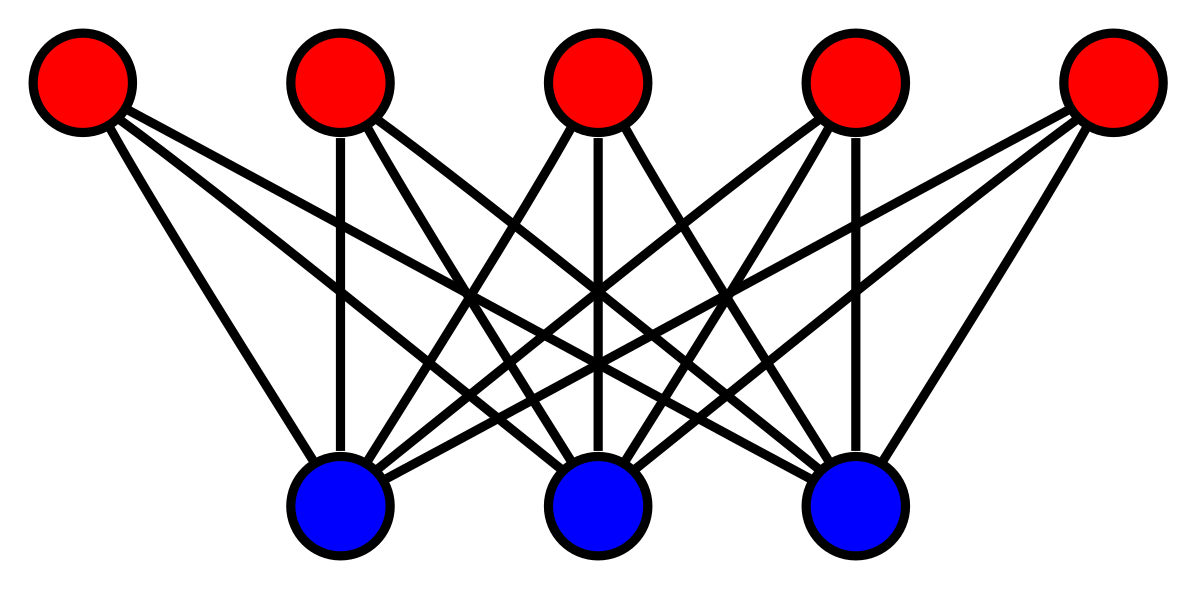
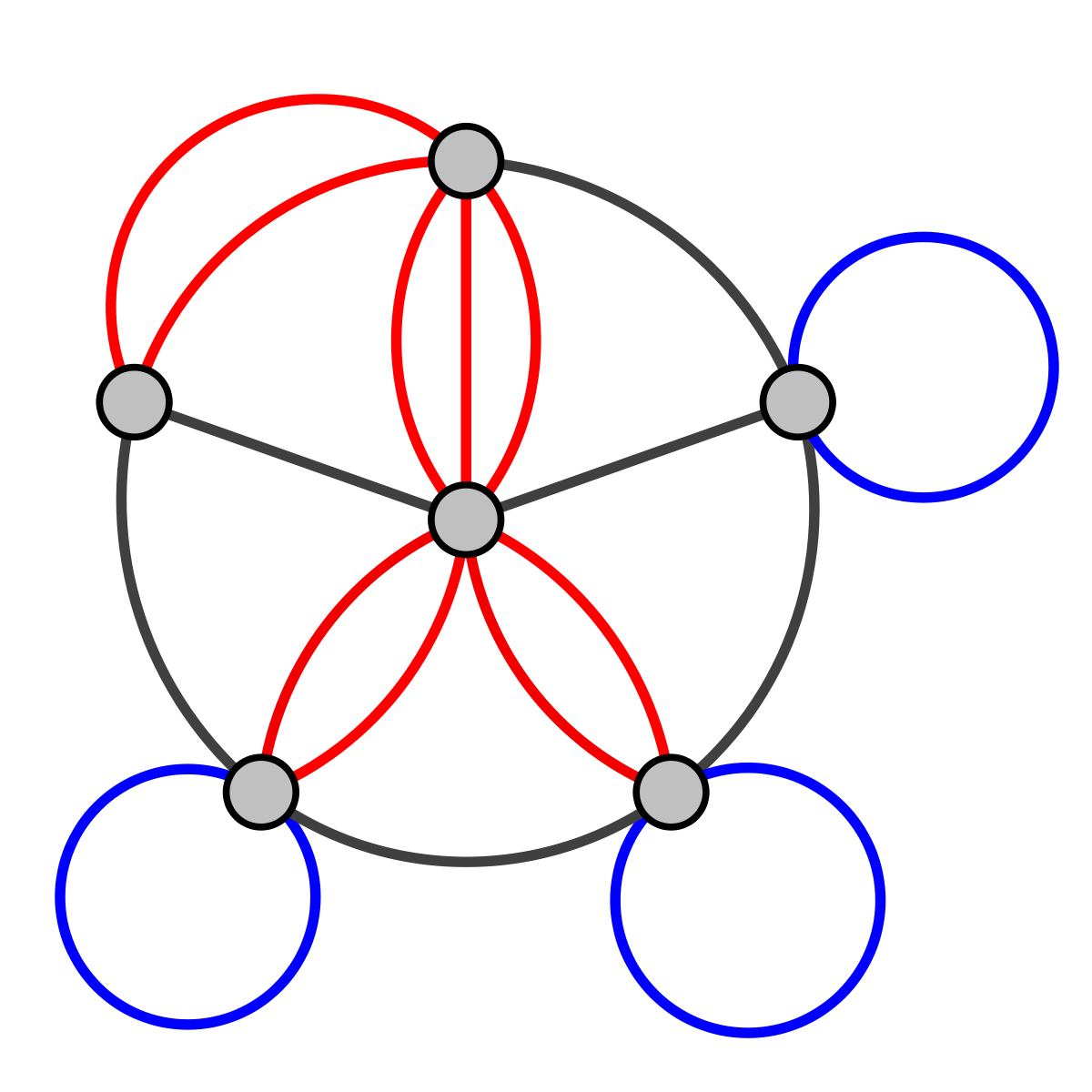
V=vértice 🡪representan objetos

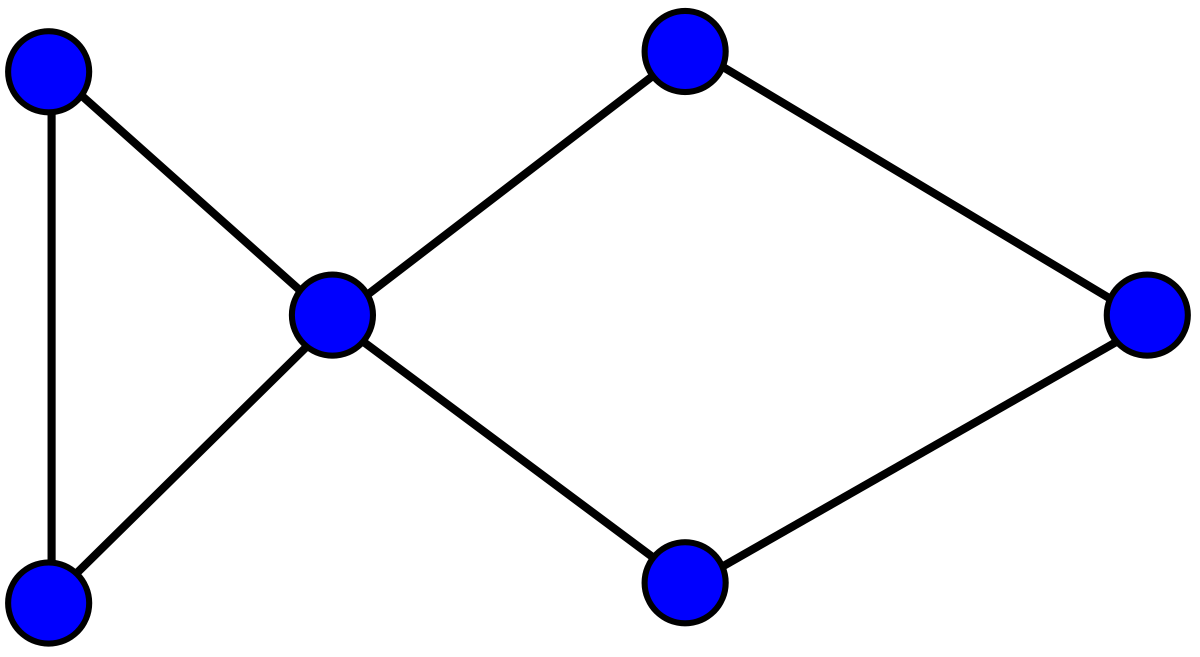
E=arista 🡪 representa relaciones

**Tipos de Grafo**

* **Grafo regular**
  + Todos los vértices tienen el mismo grado



* **Grafo completo**
  + Tiene una arista entre cualquier par de vertices 
* **Grafo bipartitos**
  + Tiene dos partes
  + Sus vértices son la unión de dos grupos de vértices 
* **Multigrafo**
  + Es un grafo que tiene arcos mutipl-
* **Grafo simple** 
  + Es un grafo que no tiene multigrafo



**TIPOS DE GRAFOS (DIRECCION)**

* **Grafos no dirigidos**

Las aristas no tienen dirección (se pueden recorrer de ambos sentidos)



* **Grafos dirigidos**

Las aristas son parejas ordenadas de vértices

DAG🡪DIRECTED ACYCLE GRAPH



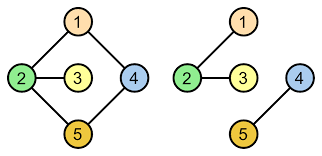
* **Grafo paralelo**

Puede ser dirigido y no dirigido

Cada arista tiene asociado un valor

**Grafo convexo**

* Existe un camino entre cualquier par de nodos



**Matriz adyacencia**

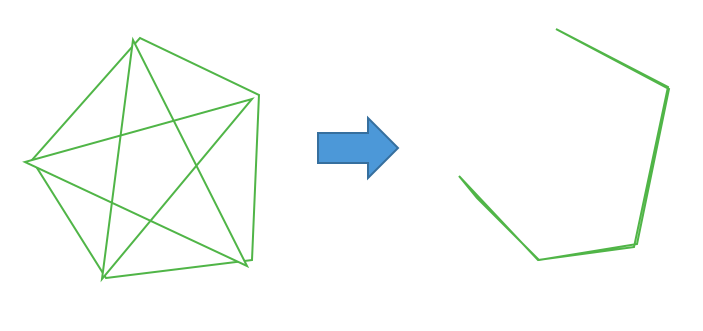
****

****

**Arbol de expansión**

Informalmente, un árbol de expansión de G es una selección de aristas de G que forman un árbol que cubre todos los vértices. Esto es, cada vértice está en el árbol, pero no hay ciclos.

* Distribuir algo a las n vértices
* Interactuar pines en un chip
* Conectar n ciudades sin ciclos
* Arbol de expansión (ST) dado un grafo G=(V,E) un donde E´ <= E y G´es un árbol donde no tiene ciclos es convexo y hay un nodo raíz

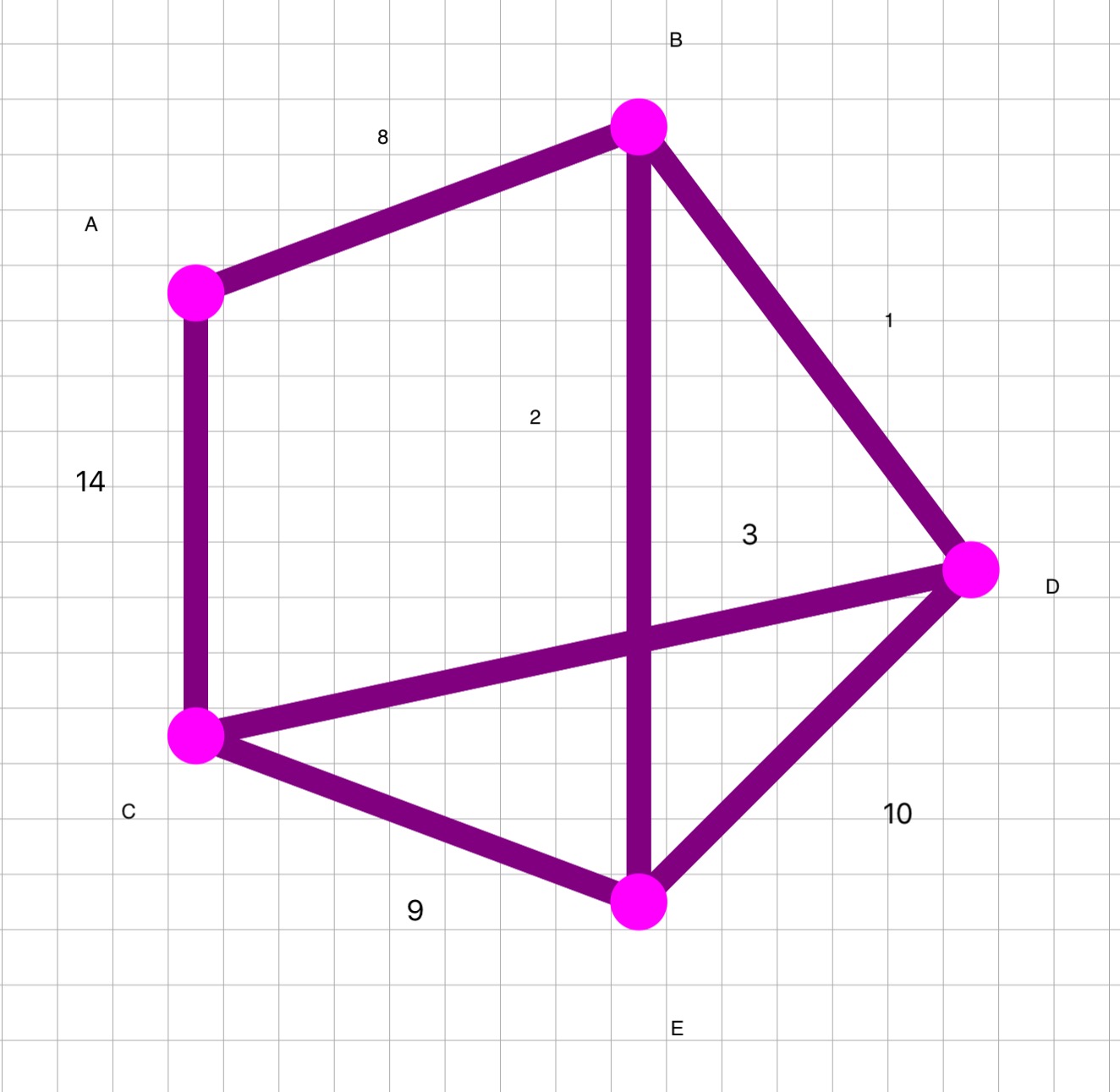
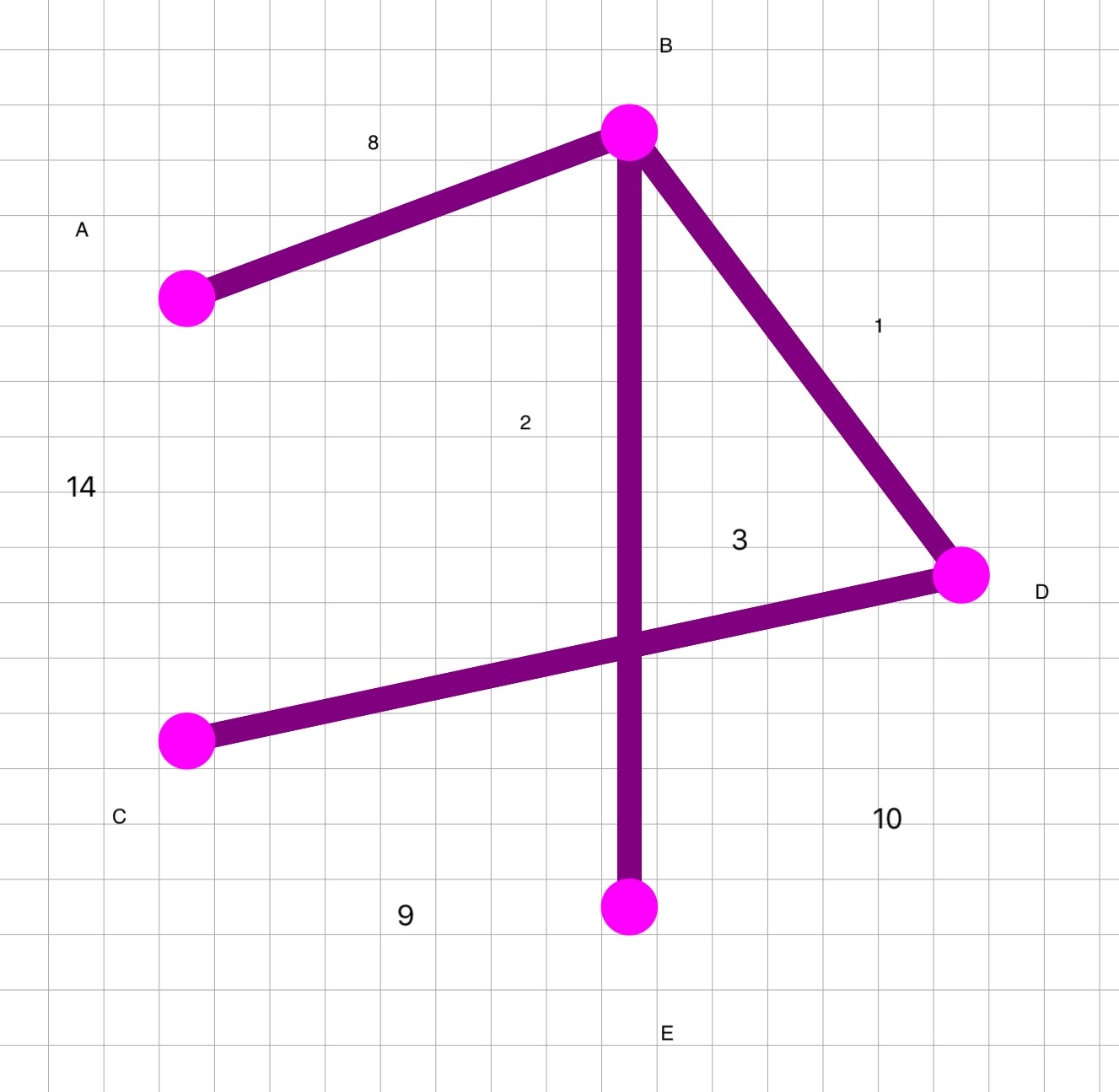


Arbol de expansión minima (mínimum spanning tree)

Supongamosque G es un grafo ponderado y que el resto de un grafo es u(G)= donde w(u,v) es

* la ponderación de la arista (u,v)
* lo que queremos es encontrar el árbol





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| key | a | b | c | d | E |
|  | ∞ | ∞ | ∞ | ∞ | ∞ |
|  | 0 | ∞ | ∞ | ∞ | ∞ |
|  |  | 8 | 14 | ∞ | ∞ |
|  |  |  | 14 | 1 | 2 |
|  |  |  | 3 |  |  |
| ¶ |  | a | d | b | b |

**Camino más corto**

En la teoría de grafos, el problema del camino más corto es el problema que consiste en encontrar un camino entre dos vértices (o nodos) de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima. Un ejemplo de esto es encontrar el camino más rápido para ir de una ciudad a otra en un mapa.

Camino mas corto desde una fuente. Dado un grafo G=(v,z) y una familia de peso wi EàR El peso de un camino p=<V1, V2, V3, V4…> es la suma de las aristas que lo construye a

W(P)=

El peso del camino mas corto

F(u,r)

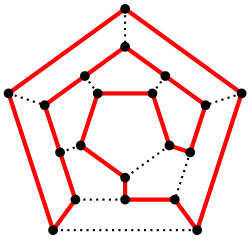
Min {w(p) u si hay camino

**Recorrido hamiltoneano**

Un camino hamiltoniano, en el campo matemático de la teoría de grafos, es un camino de un grafo, una sucesión de aristas adyacentes, que visita todos los vértices del grafo una sola vez.

Pasa solo una vez

¡existe?



Flujo máximo

Existe un flujo que viaja desde un único lugar de origen hacia un único lugar de destino a través de arcos que conectan nodos intermediarios. Los arcos tienen una capacidad máxima de flujo y se trata de enviar desde la fuente al destina la mayor cantidad posible de flujo.