Centro de
Investigación en
Computación

Greedy algorithms

Algoritmos
Voraces, golosos, glotones, ávidos, ambiciosos, etc.

Tema 4

Course

Analysis and design of algorithms

Instructor

Acosta Bermejo Raúl et al.

Lecture notes

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

Introduction

Introducción

Tema

Introduction

Introducción

- Some games (like chess) can be won only by **thinking ahead**: a player who is focused entirely on immediate advantage is easy to defeat.
- Other games (like Scrabble), it is possible to do quite well by simply making whichever move seems best **at the moment** and not worrying too much about future consequences.
  - This sort of myopic behavior is easy and convenient, making it an attractive algorithmic strategy.
  - ***Greedy algorithms*** build up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.
  - Although such an approach can be disastrous for some computational tasks, there are many for which it is optimal.

Introduction

Introducción

Greedy. Adjective

- Having an excessive desire or appetite for food.
- Having or showing an intense and selfish desire for wealth or power.

Introduction

Introducción

- An optimization problem is one in which you want to find, not just a solution, but the best solution.
- A greedy algorithm sometimes works well for optimization problems.
- A greedy algorithm works in phases (small steps). At each phase:
  - You take the best you can get right now (local decisions), without regard for future consequences.
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum.
- Since previous decisions are never reconsidered we use little memory.

Introduction

Introducción

Greedy is a **strategy** that works well on optimization problems with the following characteristics:

1. Greedy-choice property: A global optimum can be arrived at by selecting a local optimum.
2. Optimal substructure: An optimal solution to the problem contains an optimal solution to subproblems.

The second property may make greedy algorithms look like *dynamic programming*. However, the two techniques are quite different.

Introduction

Ejemplos simples

Before to view the 6 problems of the outline, we are going to analyse:

- The counting money problem.
- Huffman encoding.

- Suppose you want to count out a certain amount of money, using the fewest possible bills and coins. A greedy algorithm would do this would be:
- At each step, take the largest possible bill or coin that does not overshoot. Example: To make $6.39, you can choose:
  - A $5 bill
  - A $1 bill, to make $6
  - A 25¢ coin, to make $6.25
  - A 10¢ coin, to make $6.35
  - Four 1¢ coins, to make $6.39
- For US money, the greedy algorithm always gives the optimum solution.

Introduction

Ejemplos simples

CASHIERS-ALGORITHM (x, c_1, c_2, ..., c_n)

SORT n coin denominations so that $c_1 < c_2 < \ldots < c_n$

$S \leftarrow \varphi$ set of coins selected

while $x > 0$

$k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$

if no such k,

RETURN "no solution"

else

$x \leftarrow x - c_k$

$S \leftarrow S \cup \{k\}$

return S

Question

Is cashier's algorithm optimal?

Introduction

Ejemplos simples

- In some (fictional) monetary system, “krons” come in 1 kron, 7 kron, and 10 kron coins.
- Using a greedy algorithm to count out 15 krons, you would get
  - A 10 kron piece.
  - Five 1 kron pieces, for a total of 15 krons.
  - This requires six coins.
- A better solution would be to use two 7 kron pieces and one 1 kron piece.
  - This only requires three coins.
- The greedy algorithm results in a solution, but not in an optimal solution.

How to resolve a problem?

Definiciones

Análisis/Sintesis
Deducción/Inducción

Estrategia

- Palabra derivada del latín strategĭa, que a su vez procede de dos términos griegos: *stratos* (ejército) y *agein* (conductor, guía). Por lo tanto, el significado es:

  Arte de proyectar y dirigir las operaciones militares, especialmente las de guerra.
- La **estrategia** es el conjunto de acciones planificadas y coordinadas sistemáticamente en el tiempo (programa o plan) que se llevan a cabo, para lograr/alcanzar un determinado fin o misión (objetivo propuesto).

Táctica

- Es el método o la forma empleada, con el fin de cumplir un objetivo y que a la vez contribuye a lograr el propósito general, de acuerdo a las circunstancias que tiene que enfrentar.

La estrategia es el plan general para lograr los buenos resultados, y la táctica, son las formas o métodos específico que se aplican de acuerdo a las circunstancias, para cumplir de forma efectiva el plan estratégico.

No sólo se cirscuncribe a la Guerra, economia y deporte, sino donde hay grupos de combate.

Chess example

Ajedrez

Antes de jugar, tenemos que preparar un plan de juego (**estrategia**):

- Inicio.- Desplegar las piezas para no estar encerrado (defenderse y/o atacar).
  - Aperturas
    - Controlar el centro ya que maximiza los posibles movimientos: reyna y caballos al centro.
    - No avanzar pienzas sin protección ni perdida de movimientos (enroque).
- Medio
  - Cambios de piezas segun el Peso y la Posición del tablero.
- Fin
  - Mates típicos
    - Aplicar las mejores combinaciones: reyna y torre (2 lineas), alfiles (2 diagonales), etc.

Luego aprovechar las buenas posiciones de las piezas, para emplear la **táctica**, como:

- Maniobras
- Combinaciones
- Ataque dobles
- Jaques a la descubierta
- Bloqueo de peones

Clavada es una situación en la cual una pieza no puede moverse sin exponer a otra pieza de su color y de mayor valor a ser capturada.
Se produce cuando tres piezas están en la misma fila, columna o diagonal del tablero; la pieza atacante, la pieza atacada o clavada, y la pieza que quedaría expuesta si la pieza clavada se mueve.

Introduction

Introducción

No real consensus on a universal definition.

Greedy algorithms:

- Make decision incrementally in small steps without backtracking.
- Decision at each step is based on improving local or current state in a myopic fashion without paying attention to the global situation
- Decisions often based on some fixed and simple priority rules.

Greedy algorithms come naturally but often are incorrect. A proof of correctness is an absolute necessity.

Interval scheduling

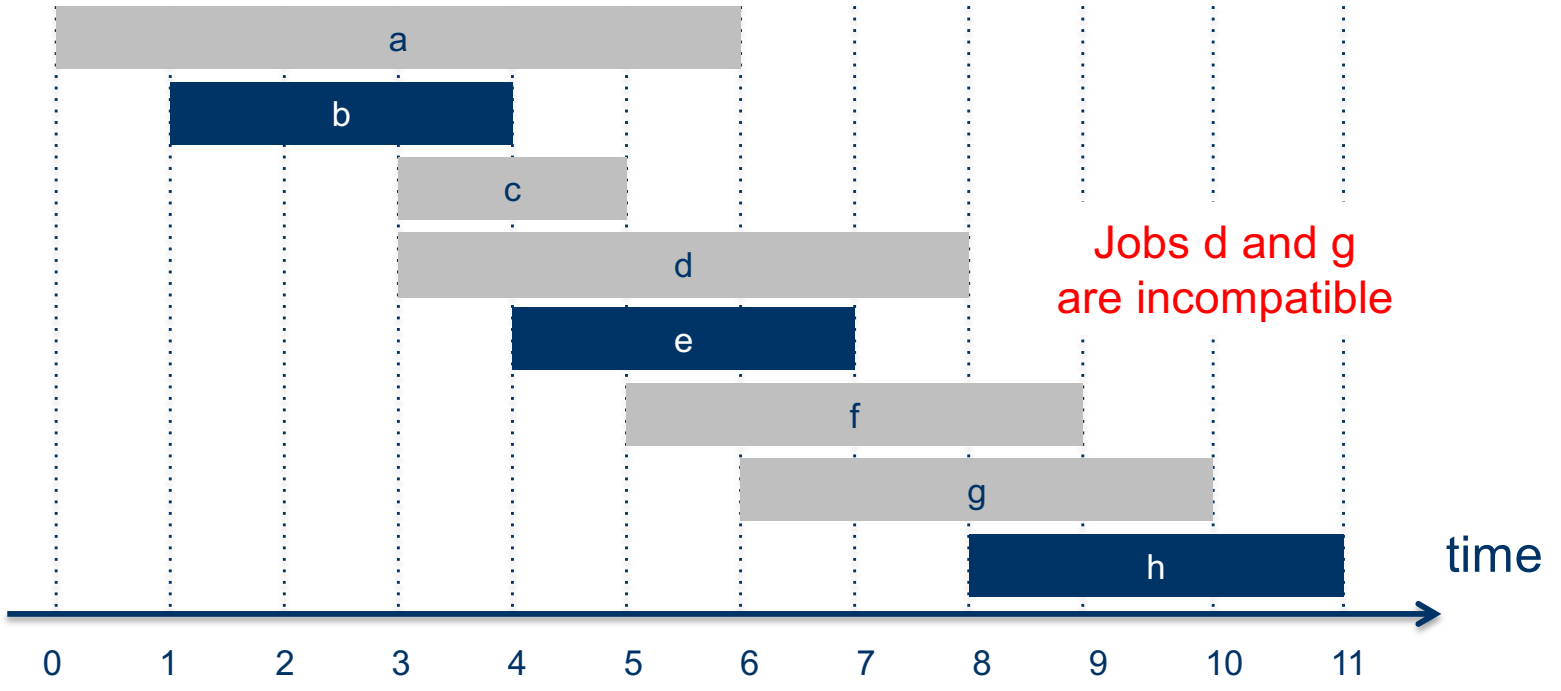
Planificador por intervalos

Tema

Interval scheduling

Calendarización de intervalos, Planificadores

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- **Goal**: find maximum subset of mutually compatible jobs.



Interval scheduling

Análisis

Fuerza bruta

Solución realizando “la mayor cantidad de cálculos”:

1. Comparar una tarea con todas las demás (n-1).
   - Así se obtiene con quienes no tiene traslapa.
2. Repetir el paso anterior para todas las tareas (n).
   - El cálculo general sería hasta aquí $n(n-1)=n^2$ => $O(n^2)$
3. Falta ahora enlazar las tareas que no tengan traslapes.
   - Esto para todas sus combinaciones.

Este algoritmo serviría para encontrar todas las tareas que no se traslapan.

Interval scheduling

Estrategias

This problem was presented in the section 1 and now we are going to use it to analyze the Greedy algorithms.

- The basic idea in a greedy algorithm for interval scheduling is to use a **simple rule** to select a first request i1. Once a request i1 is accepted we reject all requests that are not compatible with i1. We then select the next request i2 to be accepted, and again reject all requests that are not compatible with i2.
- The challenge in designing a good greedy algorithm is in deciding **which simple rule to use** for the selection — and there are many natural rules for this problem that do not give good solutions.

Interval scheduling

Estrategias

Greedy template. Consider jobs in some natural order.

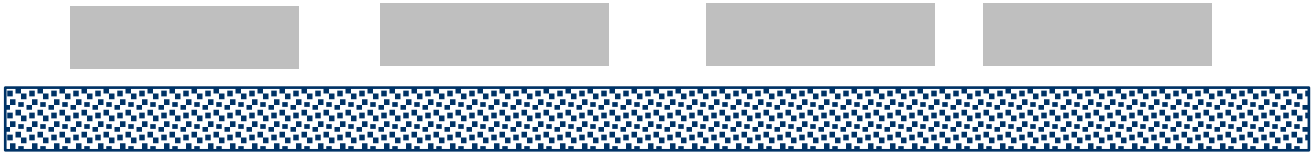
Take each job provided it's compatible with the ones already taken:

- **Earliest start time**: Consider jobs in ascending order of s_j.
- **Earliest finish time**: Consider jobs in ascending order of f_j.
- **Shortest interval**: Consider jobs in ascending order of $f_j - s_j$.
- **Fewest conflicts**: For each job j, count the number of conflicting jobs c_j. Schedule in ascending order of c_j.

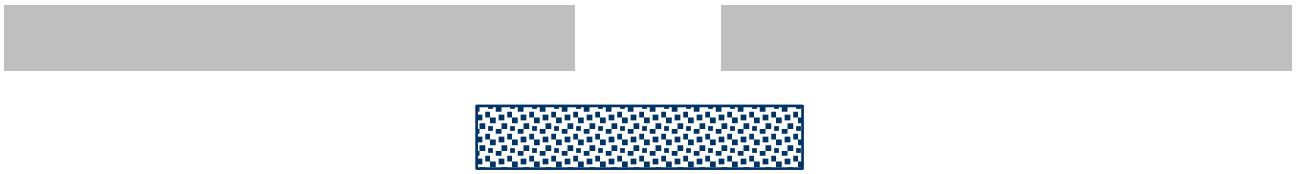
Interval scheduling (counter-example)

Contra-ejemplos

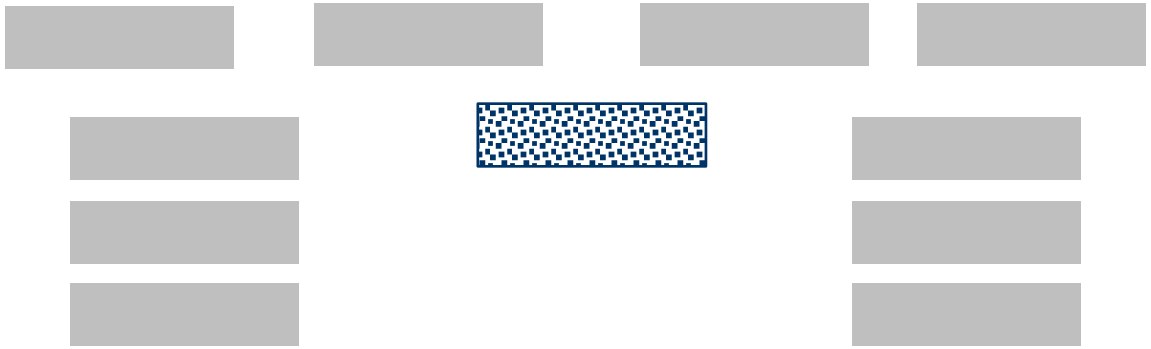
Earliest start time



Shortest interval



Fewest conflicts



Interval scheduling

Algoritmo

```
1.  EARLIEST-FINISH-TIME-FIRST
                (n, s1, s2, ..., sn , f1, f2, ..., fn)
2.      Sort jobs by finish time so that f1 ≤ f2 ≤ ... ≤ fn
3.      A ← φ   (set of jobs selected)
4.      for j = 1 to n
5.          if job j is compatible with A
6.              A ←AU{ j }
7.          Delete all requests from R incompatible with i
8.      return A
```

R = { all requests i with (s_i, f_i) }

2.- O(n log n)

4.- O(n)

O(nlog n + n)

EFTF is O(n log n)

Proposition

- Can implement earliest-finish-time first in O(n log n) time.
- Keep track of job j* that was added last to A.
- Job j is **compatible** with A iff $s_j \geq f_j^*$.
- Sorting by finish time takes O(n log n) time.

Interval scheduling

Es correcto?

Theorem. The earliest-finish-time-first algorithm is optimal.

No es obvio que sea óptimo!

Sobre todo si se piensa en un contraejemplo similar.

Ver slides para demostraciones

Proof

La estructura de la demostración es la siguiente:

1. Probar que A es óptimo sin verificar que A = O (donde O es el conjunto de las soluciones optimas) y tan sólo que |A| = |O|.
2. Comparar soluciones parciales de Greedy con las soluciones de O y probar que lo hace mejor o en el peor caso igual (stays ahead).
3. Se usan 2 demostraciones:
    1. Por inducción: para recorrer los elementos.
    2. Por cotradicción/reducción a lo absurdo: si es la óptima.

Minimum lateness scheduling

Planificador de latencia mínima

Tema

Minimum lateness scheduling

Planificadores de latencia mínima

Minimizing lateness problem

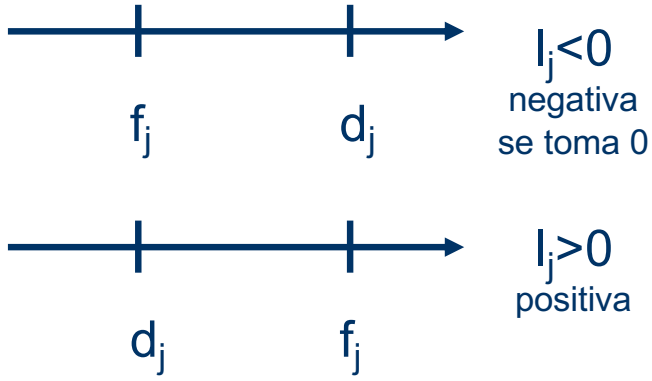
1 procesador
N procesos

- Single resource processes one job at a time.
- Job *j* requires t_j units of processing time and is due at time d_j.
- If *j* starts at time s_j, it finishes at time $f_j = s_j + t_j$.

deadline
(a lo más termina)

- Lateness: $l_j = \max \{ 0, f_j - d_j \}$.
- **Goal**: schedule all jobs to minimize maximum lateness $L = \max_j l_j$.

j	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

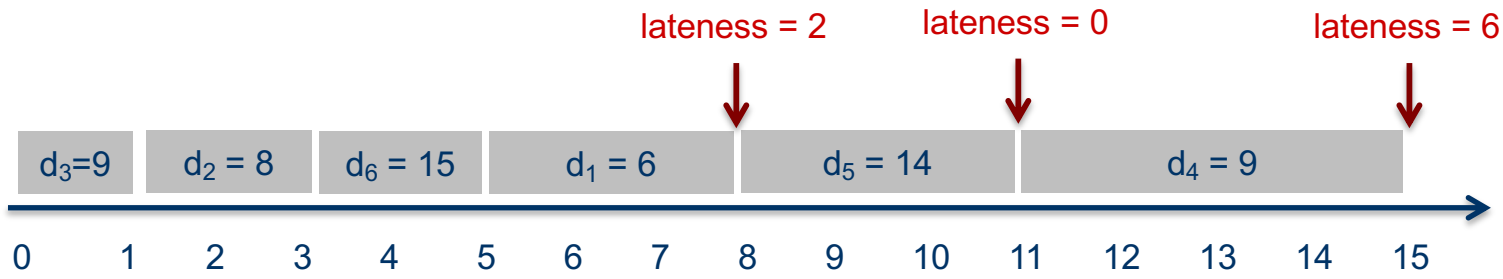


Minimum lateness scheduling

Planificadores de latencia mínima

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

One possible scheduling is:



Cual es la estrategia? Usar t_j, d_j o algún cálculo entre ambas?
Cual usamos aqui? Es la mejor?

Minimum lateness scheduling

Planificadores de latencia mínima

Greedy strategy. Schedule jobs according to some natural order.

- **Shortest processing time first**: Schedule jobs in ascending order of processing time t_j.

counter-example

	1	2
t_j	1	10
d_j	100	10

Si hay varias tareas cortas
con deadlines largos afectan
a las que tienen deadlines cortos
En este caso t_2

- **Smallest slack**: Schedule jobs in ascending order of slack $d_j - t_j$.

slack
Periodo de poca actividad

counter-example

	1	2
t_j	10	1
d_j	10	2

Aqui t_2 es más penalizado:
$latencia_1 = 10-10=0$
$latencia_2 = 11-2=9$

Al reves:
$latencia_1 = 11-2=9$
$latencia_2 = 1-2=0$

- **Earliest deadline first**: Schedule jobs in ascending order of deadline d_j.

Minimum lateness scheduling

Planificadores de latencia mínima

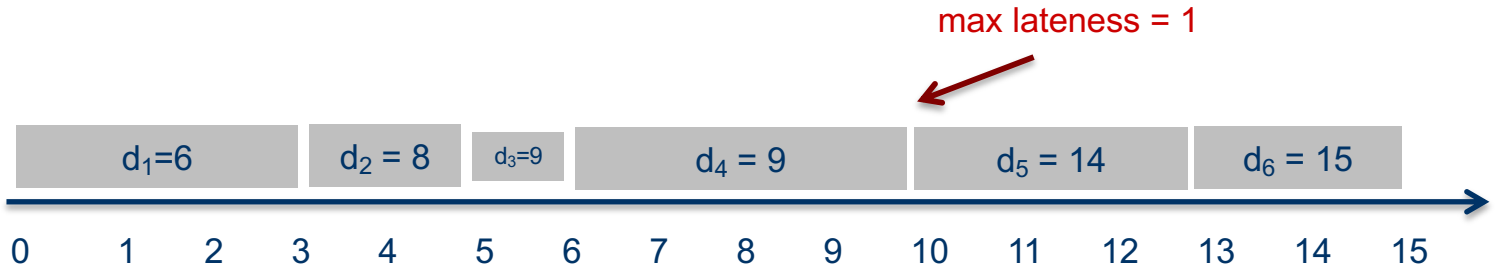
Algorithm

```
1. EARLIEST-DEADLINE-FIRST (n,t1, t2, ..., tn , d1, d2, ..., dn)
2.     Sort n jobs so that d1 ≤ d2 ≤ ... ≤ dn.
3.     t ← 0
4.     for j = 1 to n
5.         Assign job j to interval [t, t + tj ].
6.         sj ← t
7.         fj ← t+tj
8.         t ← t + tj
9. return intervals [s1, f1], [s2, f2], ..., [sn, fn].
```

Minimum lateness scheduling

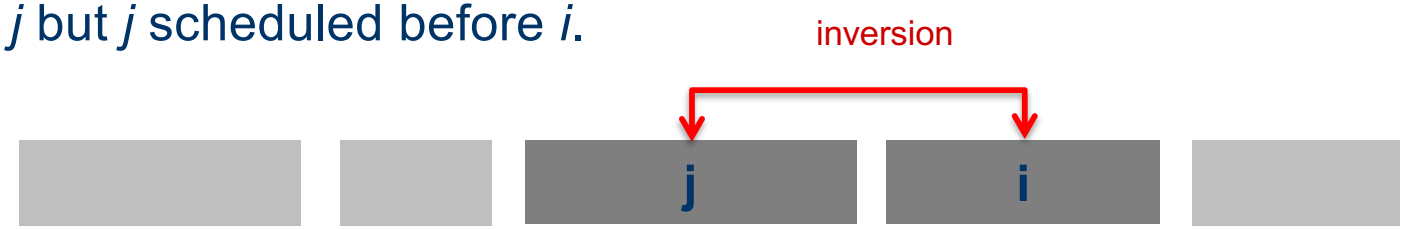
Planificadores de latencia mínima

Example of EARLIEST-DEADLINE-FIRST



Definition

Given a schedule S, an inversion is a pair of jobs *i* and *j* such that: *i* < *j* but *j* scheduled before *i*.



[as before, we assume jobs are numbered so that $d_1 \leq d_2 \leq \ldots \leq d_n$]

Minimum lateness scheduling

Planificadores de latencia mínima

Observations

1. There exists an optimal schedule with no idle time.
2. The earliest-deadline-first schedule has no idle time.
3. The earliest-deadline-first schedule has no inversions.
4. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Claim

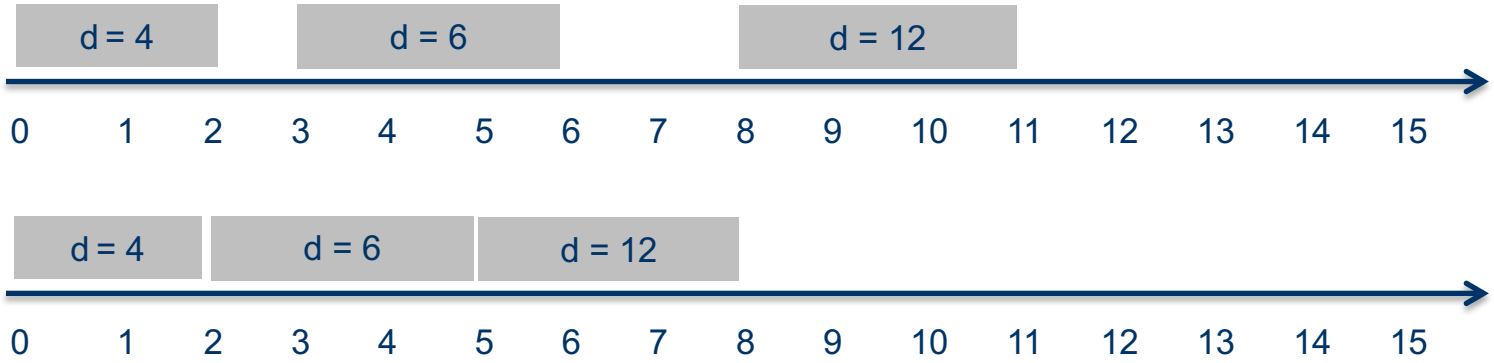
Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Minimum lateness scheduling

Planificadores de latencia mínima

Examples

- Observation 1. There exists an optimal schedule with no idle time.



- Observation 2. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Minimum lateness scheduling

Inversiones

Proof

Let L be the lateness before the swap, and let L' be it afterwards.

- $L'_k = L_k$ for all $k \neq i, j$.
- $L'_i \leq L_i$.
- If job *j* is late,

$$L'_j = f'_j - d_j \quad \text{(definition)}$$
$$= f_i - d_j \quad (j \text{ now finishes at time } f_i)$$
$$\leq f_i - d_i \quad \text{(since } i \text{ and } j \text{ inverted)}$$
$$\leq L_i \quad \text{(definition)}$$

Ver slides para demostraciones

Minimum lateness scheduling

Bibliografia

Papers

There are many, some of them:

- Batch scheduling to minimize maximum lateness
  - Ghosh, Gupta, 1997, Elsevier.
- A review of scheduling research involving setup considerations
  - Allahverdi, Gupta, Aldowaisan, 1999, Pergamon.
- Taxonomy: Scheduling problems with setups (Times or Costs)
  - Batch/Non-batch, Sequence dependent/**In**dependent.
  - Single or parallel machines.
- A survey of scheduling problems with setup times or costs
  - Allahverdi, Cheng, Kovalyov, 2008, Elsevier.
- Etc.

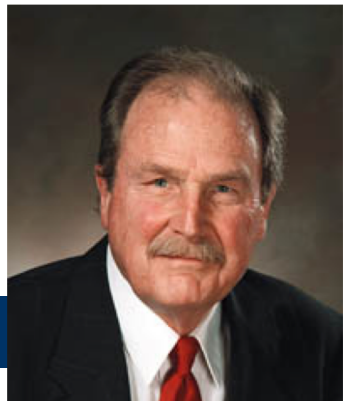
Huffman coding

Del área de Teoría de la información

No vienen en el temario oficial pero
sin en la mayoría de la literatura

Huffman coding

Codificación de Huffman



1925-1999
Estudiante en el MIT

Huffman coding

- In 1951, David A. Huffman and his MIT information theory classmates were given the choice of a term paper or a final exam.
  - A term paper is a research paper written by students over an academic term, accounting for a large part of a grade.
  - Term papers are generally intended to describe an event, a concept, or argue a point.
  - A term paper is a written original work discussing a topic in detail, usually several typed pages in length and is often due at the end of a semester.

Huffman coding

Codificación de Huffman

Huffman coding

- The professor, Robert M. Fano, assigned a term paper on the problem of finding the most efficient binary code.
- Huffman, unable to prove any codes were the most efficient, was about to give up and start studying for the final when he hit upon the idea of using a **frequency**-sorted binary tree and quickly proved this method the most efficient.
- In doing so, the student outdid his professor, who had worked with information theory inventor Claude Shannon to develop a similar code.
- By building the **tree** from the bottom up instead of the top down, Huffman avoided the major flaw of the suboptimal Shannon-Fano coding.

Huffman coding

Codificación de Huffman

Encoding symbols using bits

- Suppose we want to represent symbols using a computer:
  - Letters of the English alphabet (or more broadly ASCII characters).
  - Pixels of an image.
  - Audio information from a sound clip.
- Ultimately these are converted to bits.
- How do we represent these symbols?
  - How do we do it efficiently – using the fewest number of bits?
  - And make sure that we can decode the bits to recover our original symbols.
  - This is a fundamental aspect of **data compression**.

Huffman coding

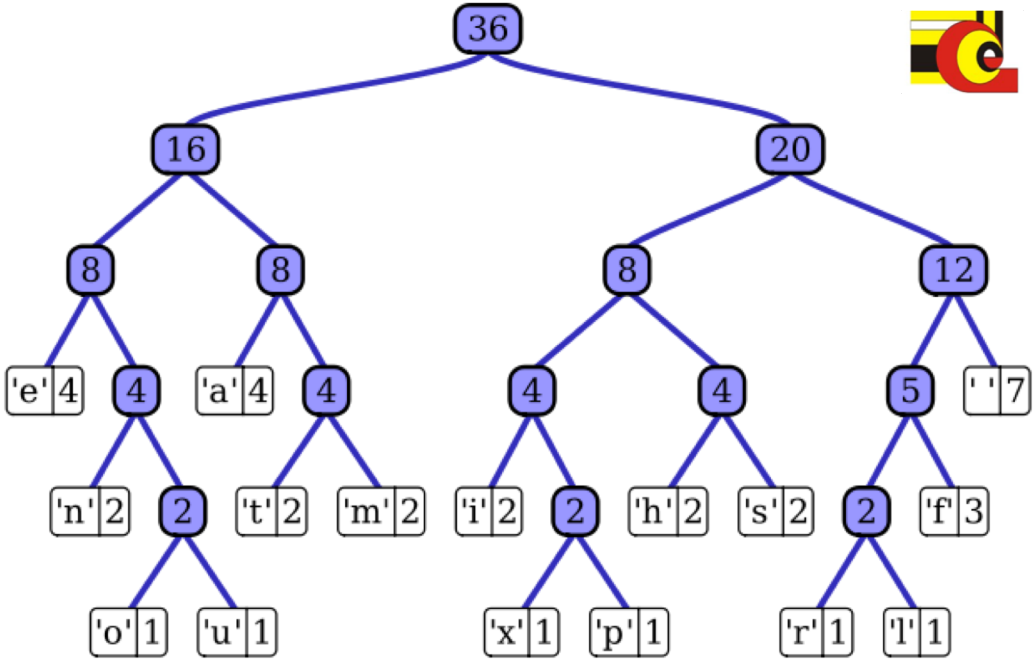
Codificación de Huffman

Características

- Códigos de longitud variable.
- En todos los idiomas hay siempre unas letras y palabras que se repiten más que otras (en español es la letra E).
- Se asignan códigos cortos a los símbolos que más se repiten (frecuencia mayor) y código más largos a los que casi no se repiten.
- Ningún código es prefijo de otro, lo que hace la decodificación rápida y única.

Introduction

Ejemplo



Huffman coding

- Huffman tree generated from the exact frequencies of the text: "this is an example of a huffman tree".
- The frequencies and codes of each character are:

  Space (7,111), a (4,010), e(4,000), f(3,1101), h(2,1010), ...
- Encoding the sentence with this code requires:
  - 135 bits, as opposed to
  - 288 bits if 36 characters of 8 bits were used or
  - 180 bits if 36 characters of 5 bits were used.
- This assumes that the code tree structure is known to the decoder and thus does not need to be counted as part of the transmitted information.

Introduction

Ejemplo

Algoritmo Greedy

Se construye un arbol de la siguiente forma:

Códigos de longitud variable

- Con cada símbolo se crea un nodo (hoja) con (valor, frec), ej. (e,4), (n,2), etc.
- Se crea una lista ordenada con los nodos/hojas (bottom-up).
- Se eligen los 2 nodos con menor frecuencia y se unen en una rama cuya raiz tiene la suma de las frecuencias.
- Se inserta el nodo resultante en la lista ordenada
- Se repite el proceso anterior hasta que sólo quede una rama (árbol).
- El código de los símbolos se genera con el path al nodo, donde una arista derecha se toma como 1, y una aritas izquierda se toma como 0.
- En cada paso se toma una decisión local tomando sólo 2 nodos.

Huffman coding

Codificación de Huffman

procedure **Huffman**(f)

Input: An array f [1 · · · n] of frequencies

Output: An encoding tree with n leaves

let H be a priority queue of integers, ordered by f

for i=1 to n:

  insert(H,i)

for k = n + 1 to 2n − 1:

  i = deletemin(H),

  j = deletemin(H)

  create a node numbered k with children i, j

  f[k] = f[i] + f[j]

  insert(H, k)

$O(n^2)$

$O(n)$

Libro

S. Dasgupta,
C.H. Papadimitriou, and
U.V. Vazirani

Huffman coding

Codificación de Huffman

Informal description

Given

A set of symbols and their weights (usually proportional to probabilities).

Find

A prefix-free binary code (a set of codewords) with minimum expected codeword length (equivalently, a tree with minimum weighted path length from the root).

Formalized description

Input

Alphabet $A = \{a_1, a_2, \ldots, a_n\}$, which is the symbol alphabet of size n.

Set $W = \{w_1, w_2, \ldots, w_n\}$, which is the set of the (positive) symbol weights (usually proportional to probabilities), i.e. $w_i = \text{weight}(a_i)$, $1 \leq i \leq n$.

Output

Code $C(A,W) = (c_1, c_2, \ldots, c_n)$, which is the tuple of (binary) codewords, where c_i is the codeword for a_i, $1 \leq i \leq n$.

Goal

Let $L(C) = \Sigma_{i=1}^{n} w_i \times \text{length}(c_i)$ be the weighted path length of code C. Condition: $L(C) \leq L(T)$ for any code $T(A,W)$.

Optimal caching

Cache óptimo

Tema

Optimal caching

Caché óptimo

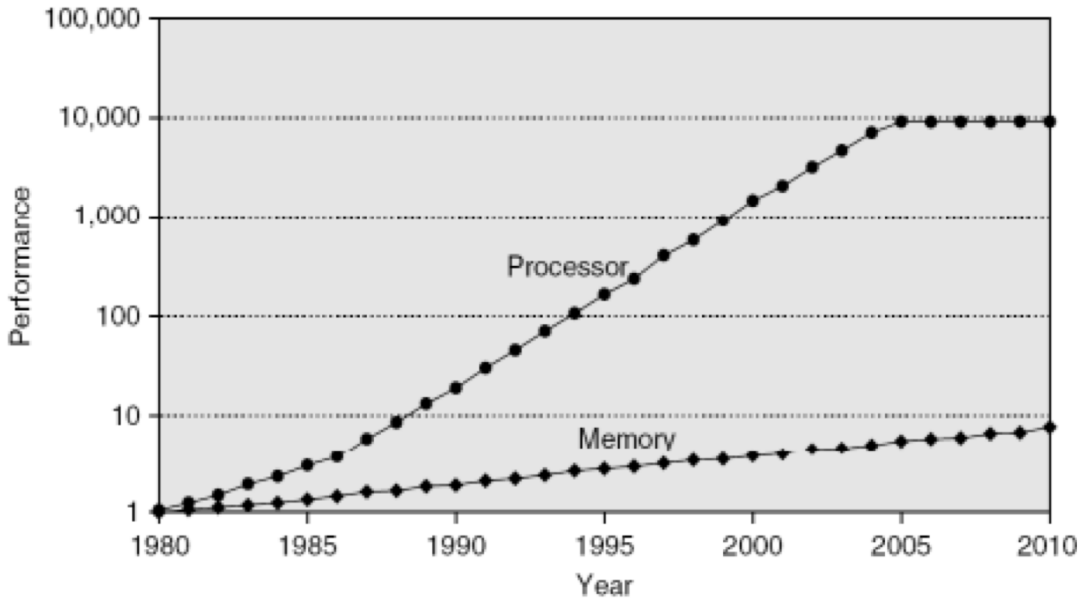
Motivación

- La velocidad de la CPU ha ido aumentado a un ritmo superior al que lo hace la memoria principal.
- La diferencia progresiva a lo largo del tiempo entre la capacidad de la CPU para ejecutar instrucciones y la de la memoria principal (tecnología DRAM) para suministrarlas.
- La memoria caché es una solución estructural a este problema de diferencia de velocidades, y consiste en interponer una memoria más rápida (tecnología SRAM) y más pequeña entre la principal y la CPU.

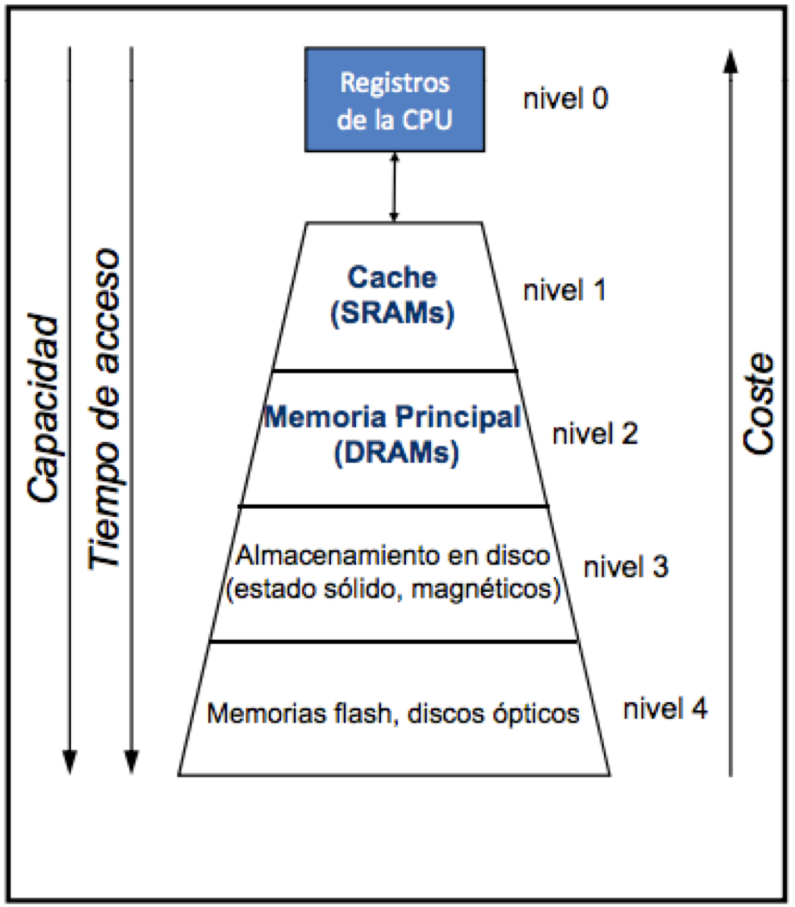
Optimal caching

Caché óptimo

Evolución de los tiempos



Tipo de memoria	Tiempo de acceso (ns)
SRAM	0.5-2.5
DRAM	50-70



Optimal caching

Caché óptimo

- Se utiliza un mecanismo específico (**algoritmo**) fundamentado en la localidad de referencia de los programas hace posible que las palabras de memoria se encuentren en la caché cuando son referenciadas por la CPU.
  - La caché es la memoria de acceso rápido de una computadora, que **guarda temporalmente** los datos recientemente procesados.
  - Es usada por el microprocesador para reducir el tiempo de acceso a datos ubicados en la memoria principal que se utilizan con **más frecuencia**.

Optimal caching

Caché óptimo

Algoritmo genérico

- Cuando se accede por primera vez a un dato, se hace una copia en la caché.
- Los accesos siguientes se realizan a dicha copia, haciendo que sea menor el tiempo de acceso medio al dato.
- Cuando se necesita leer o escribir en una ubicación en memoria principal:
  - Primero verifica si una copia de los datos está en la caché;
  - Si es así, el microprocesador de inmediato lee o escribe en la memoria caché.

Optimal caching

Caché óptimo

Los datos en la memoria caché se alojan en distintos niveles según la frecuencia de uso que tengan, estos niveles son los siguientes:

- Memoria caché nivel 1 (Caché L1).
- Memoria caché nivel 2 (Caché L2)
- Memoria caché nivel 3 (Caché L3)

Cada nivel superior (numérico) es más lento que el precedente (L2 más lento que L1) y en general todos son más rápidos que la memoria principal (RAM).

Optimal caching

Caché óptimo

Specific algorithm

- Cache with capacity to store k items.
- Sequence of m item requests d1, d2, ..., dm.
- *Cache hit*: item already in cache when requested.
- *Cache miss*: item not already in cache when requested: must bring requested item into cache, and evict (desalojar) some existing item, if full.

Goal

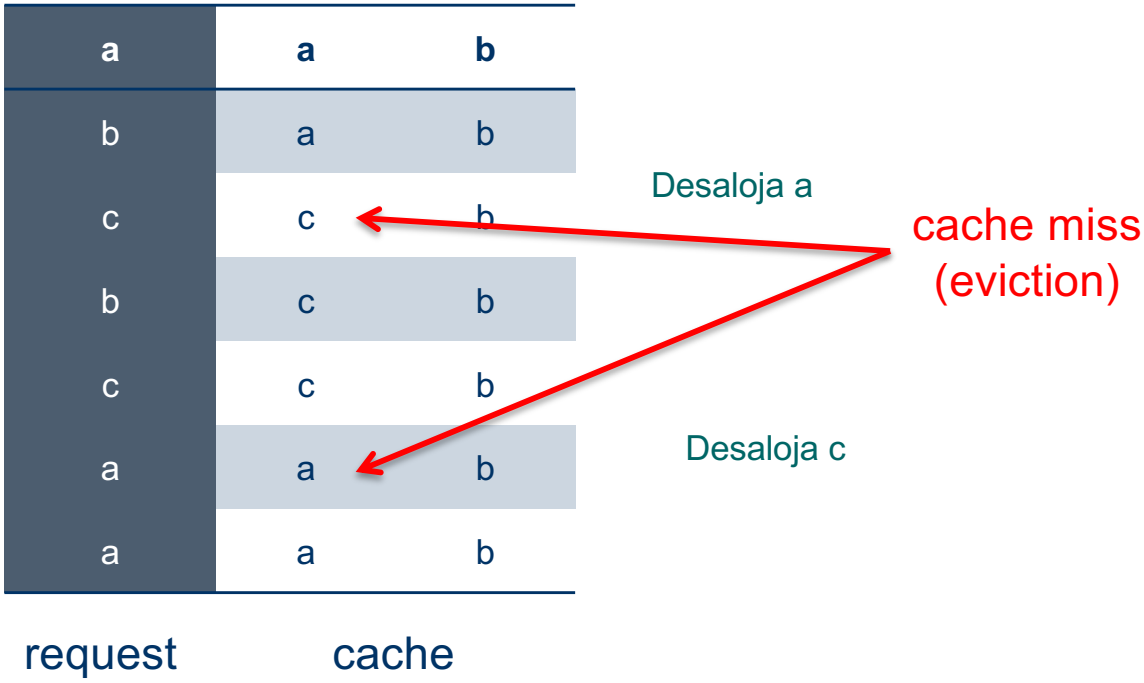
Eviction schedule that minimizes number of evictions.

Optimal caching

Caché óptimo

Example

- k = 2, initial cache = ab, requests: a, b, c, b, c, a, a.
- Optimal eviction schedule. 2 evictions.

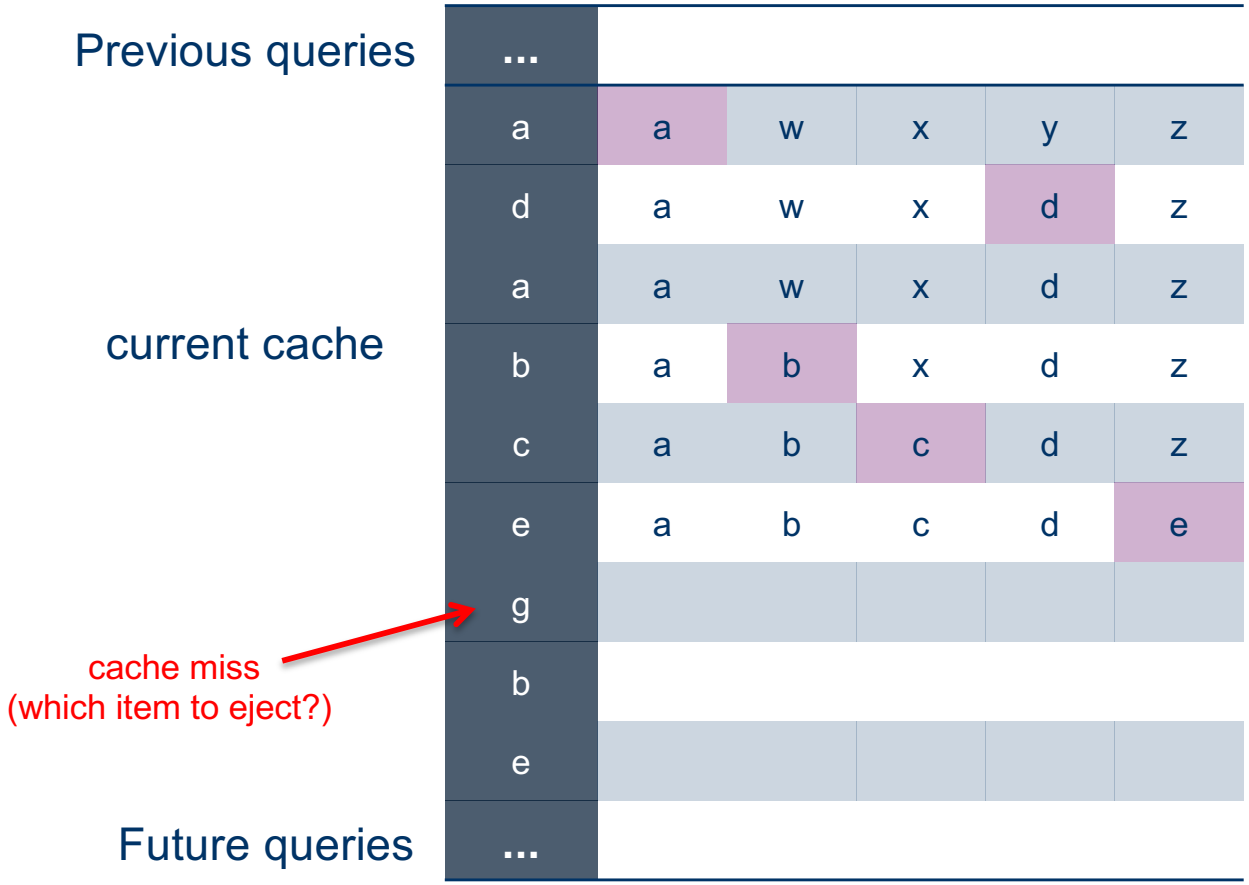


Optimal caching

Strategies

- ***LIFO / FIFO***. Evict element brought in most (east) recently.
- ***LRU***. Evict element whose most recent access was earliest.
- ***LFU***. Evict element that was least frequently requested.

Que TAD usar
Para sacar los datos?



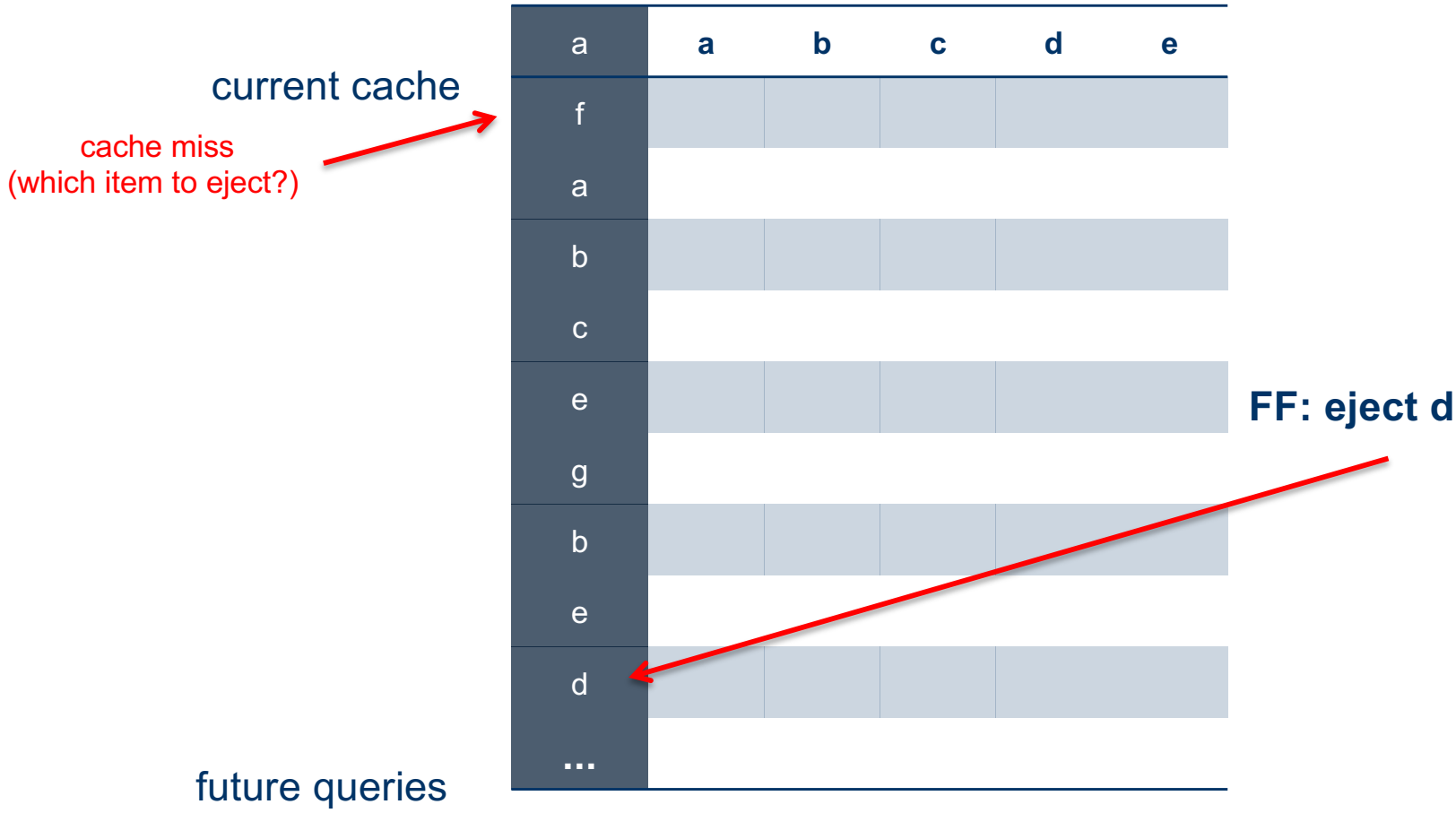
FIFO: eject a / Cola

LRU: eject d

LIFO: eject e / Pila

Optimal caching

- ***Farthest-in-future***. Evict item in the cache that is not requested until farthest in the future.



Theorem [Bélády 1966] FF is optimal eviction schedule. ***Proof*** ?

Optimal caching

Definition

A *reduced schedule* is a schedule that only inserts an item into the cache in a step in which that item is requested.

a	a	b	c
a	a	x	c
c	a	d	c
d	a	d	b
a	a	c	b
b	a	x	b
c	a	c	b
a	a	b	c
a	a	b	c

Aquí **d** no fue solicitado pero fue incertado

item inserted when not requested

an unreduced schedule

a	a	b	c
a	a	b	c
c	a	b	c
d	a	d	c
a	a	d	c
b	a	d	b
c	a	c	b
a	a	b	c
a	a	b	c

an reduced schedule

Optimal caching

Caché óptimo

In **mathematics**, an **invariant** is a property of a mathematical object which remains unchanged, after operations or transformations of a certain type are applied to the objects.
In **CS**, we have similar concepts expresed like: loop invariantes, and assertions.

Estructura de la demostración

Se va a probar:

1. Claim

   Given any unreduced schedule *S*, can transform it into a reduced schedule *S′* with no more evictions.

2. Theorem

   FF is optimal eviction algorithm.

3. Invariant

   There exists an optimal reduced schedule *S* that makes the same eviction schedule as *SFF* through the first *j* requests.

Slides auxiliares
para las demostraciones

Optimal caching

Caché óptimo

Online vs. offline algorithms

- Offline: full sequence of requests is known a priori.
- Online (reality): requests are not known in advance.
- Caching is among most fundamental online problems in CS.

Theorem. FF is optimal offline eviction algorithm.

- Provides basis for understanding and analyzing online algorithms.
- LRU is *k*-competitive.
- LIFO is arbitrarily bad.

Recuerden que hay varios tipos de caches

Shortest path in graph

Camino más corto en grafos

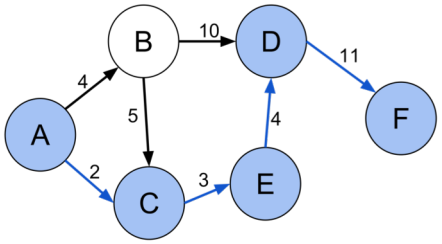
Tema

Euclidean graph

Grafo Euclidiano

Robert Sedgewick, Jeffrey Scott Vitter, **1986**, "Shortest paths in euclidean graphs".

- **Euclidean digraphs** are graphs whose:
  - Vertices are points in the plane (coordinates) and whose
  - Edges are lines connecting them with weights equal to their Euclidean distances, so that
  - You can work with graphical representations.
- But the weights might represent:
  - Time or cost or an entirely different variable and
  - Do not need to be proportional to a distance at all.
- We are emphasizing this point by using mixed-metaphor terminology where we refer to a shortest path of minimal weight or cost.



Shortest path in graph

Camino más corto en grafos

DARPA: 1962-76
TCP/IP: 1983
Internet 1990

- Los primeros algoritmos fueron desarrollados por:
  1. Leyzorek, Gray, Johnson, Ladew, Meaker, Petry y Seitz en 1957
  2. Dantzing en 1958
  3. Edsger Dijkstra en 1959
- Aunque Dijkstra no fue el primero en descubrirlo, pero lo hizo de manera “simultanea” es que se le conoce con su nombre.
- Si se remplaza el heap en el algoritmo de Dijkstra con una cola se obtiene un algoritmo desarrollado por:
  a. Shimbel 1955
  b. Moore en 1957
  c. Woodbury y Dantzing en 1957
  d. Bellman en 1958
- Como Bellman uso la idea de “relajar aristas” la cual propuso primero Ford en 1956, es que se conoce actualmente como Bellman-Ford.
- Al igual que otros autores, Dreyfus, realizo varios estudios de los algoritmos.
- Versiones más avanzadas han sido propuestas por Glover, Hung y Divoky.

Shortest path in graph

Camino más corto en grafos

Variantes

1. Un par de nodos
2. Todos los pares de nodos
3. Un origen todos los destinos
4. Un destino
5. La longitud puede ser:
    1. Todas iguales
    2. No negativa
    3. Negativa pero sin ciclos negativos
    4. Ciclos negativos

Shortest path in graph

Algoritmo de Dijkstra

Edsger Wybe Dijkstra (1930-2002).

- Encuentra la solución a un problema mediante búsquedas locales de soluciones óptimas.
- Usa la técnica de diseño conocida como Greedy.
- Parecido al de Bellman-Ford con algunas diferencias (no se tienen pesos negativos).

Complejidad $O(V^2)$ donde V=Vértices.

Artículo original

- E. Dijkstra, A Note on Two Problems in Connexion with Graphs.
- *Numerische Mathematik*, 1:269-271, 1959.

Idea del algoritmo

Cada nodo tiene dos etiquetas:

1. Una con la distancia que tiene desde el nodo origen a lo largo de un camino conocido como el mejor, y
2. Otra con la trayectoria que se va formando.
   - Esta última puede ser tentativa o permanente , al inicio todas son tentativas.
   - Cuando se descubre que una etiqueta representa el camino más corto se hace permanente.

Shortest path in graph

Algoritmo de Dijkstra

Algoritmo en Lenguaje natural

1. Se hace permanente el nodo inicial A, este es el nodo de trabajo.
2. Se examina cada nodo adyacente a A re-etiquetando cada uno de ellos con la distancia a A.
3. Se analizan todos los nodos **con etiqueta tentativa** del grafo y se hace permanente el nodo con el camino más corto, este nodo se convierte en el nuevo nodo de trabajo .
4. Se examinan todos los nodos **adyacentes** al nuevo nodo de trabajo B y si la suma de la etiqueta de B y la distancia de B al nodo adyacente que se esta considerando es menor que la etiqueta de ese nodo, éste se re-etiqueta.
5. Después de que se hayan revisado todos los nodos adyacentes al nodo de trabajo, se examina todo el grafo buscando el nodo tentativo con la menor distancia y se vuelve permanente, este será el nuevo nodo de trabajo.
6. Repetir 4 y 5 hasta llegar al destino.

Shortest path in graph

Algoritmo de Dijkstra

Pseudo-código

```
   Dijkstra(G,s)
1.    for each vertex u ∈ G.V()
2.        u.setd(∞)
3.        u.setparent(NIL)
4.    s.setd(0)
5.    S ← ∅              // Set S is used to explain the algorithm
6.    Q.init(G.V())   // Q is a priority queue ADT
7.     while not Q.isEmpty()
8.       u ← Q.extractMin()
9.       S ← S U {u}
10.      for each v ∈ u.adjacent() do
11.         Relax(u, v, G)
12.         Q.modifyKey(v)
```

Input: Graph *G*, start vertex *s*

Relaxing Edges

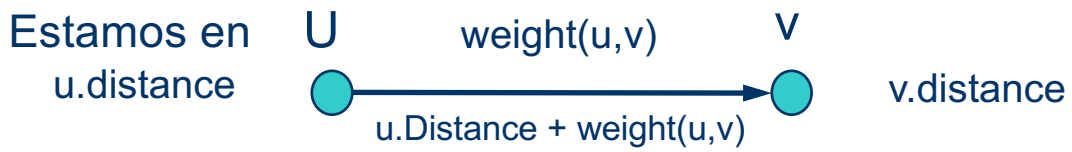
Shortest path in graph

Algoritmo de Dijkstra

Relaxation

- For each vertex *v* in the graph, we maintain *v*.d(), the estimate of the shortest path from *s*, initialized to ∞ at the start.
- Relaxing an edge (*u,v*) means testing whether we can improve the shortest path to *v* found so far by going through *u*.

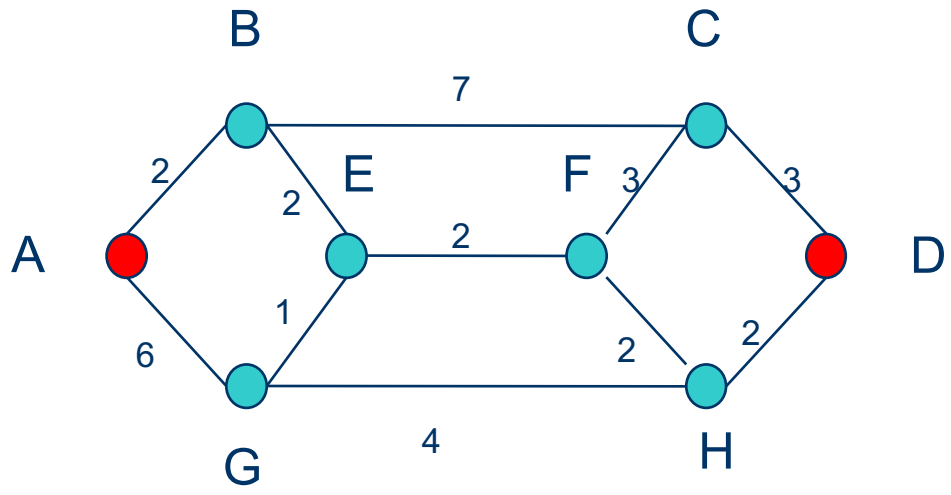
```
Relax (u,v,G)
if v.distance() > u.distance()+G.weight(u,v) then
   v.setDistance(u.distance()+ G.weight (u,v))
   v.setparent(u)
```



Shortest path in graph

Ejemplo

Consider the following graph that represents a net.



Permanente

Tentativo

Para el demo se usaran los slides auxiliares

Shortest path in graph

Ejemplo

Árbol de Trayectorias

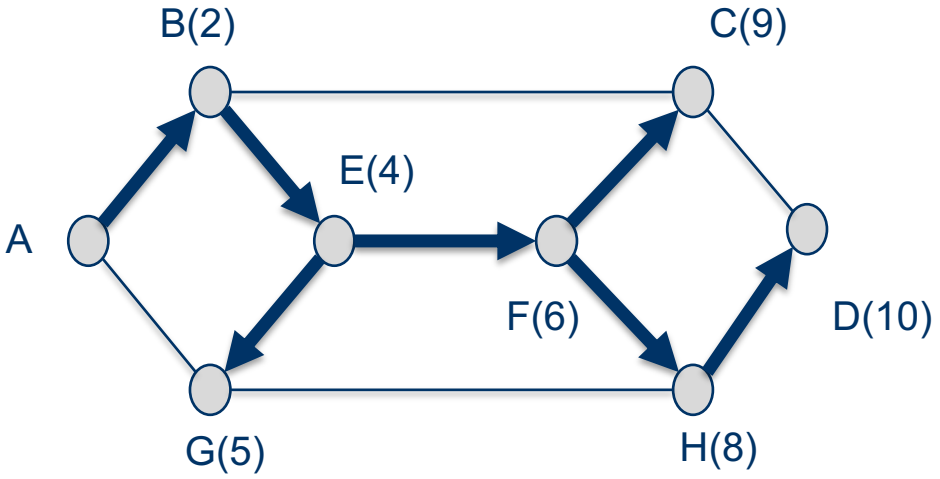


Tabla de A

B	Directo
C	B
D	B
E	B
F	B
G	B
H	B

Suponiendo E origen

Tabla de E

A	B
B	Directo
C	F
D	F
F	Directo
G	Directo
H	F

Shortest path in graph

Complejidad temporal

Analysis

- Extract-Min executed $|V|$ time.
- Decrease-Key executed $|E|$ time.
- Time = $|V|$ TExtract-Min + $|E|$ TDecrease-Key.
- T depends on different Q implementations.

Q	T(Extract-Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$ (amort.)	$O(V \lg V + E)$

Shortest path in graph

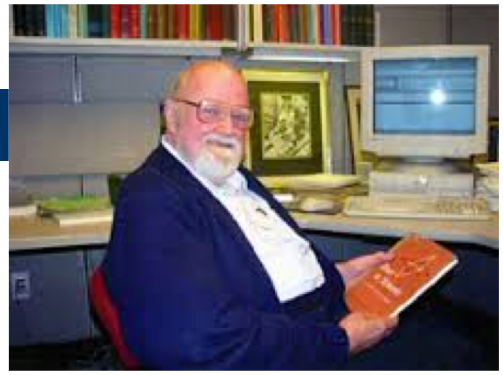
Algoritmo de Bellman-Ford

- Resuelve el problema de la trayectoria más corta de una sola fuente.
- Permite pesos negativos en las aristas, pero no permite ciclos directos de pesos negativos.

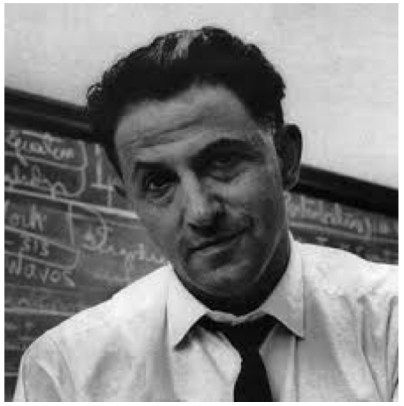
Complejidad **O(VA)** donde V=Vértices y A=Aristas

Artículo original

- R. E. Bellman, *Dynamic Programming*.
- Princeton University Press, Princeton, N.J., 1957.



Lester Randolph **Ford**, Jr.
Born September 23, 1927,
Houston
President of MAA, 1947–48



Richard E. **Bellman**
1920 - 1984
Brooklyn, New York
IEEE Medal of Honor, 1979

Shortest path in graph

Algoritmo de Bellman Ford

Relaxation Algorithm Issues

- Never stop relaxing in a graph with negative-weight cycles: infinite loop.
- A poor choice of relaxation order can lead to exponentially many relaxations.

Shortest path in graph

Algoritmo de Bellman Ford

```
Relax (u,v,G)
if v.distance() > u.distance()+G.weight(u,v) then
   v.setDistance(u.distance()+ G.weight (u,v))
   v.setparent(u)
fi
```

Pseudo-código

```
 Bellman-Ford(G,s)
1.    for v ∈ G.V()
2.       v.setDistance(∞)
3.       v.setParent(NIL)
4.    s.setDistance(0)

5.    for i=1 to |G.V()|-1 do
6.      for each edge(u,v) ∈ G.E() do
7.            Relax (u,v,G)

8.    for each edge (u,v) in G.E() do
9.       if v.d() > u.d() + G.w(u,v) then
10.           return false
11.   return true
```

Inicialización

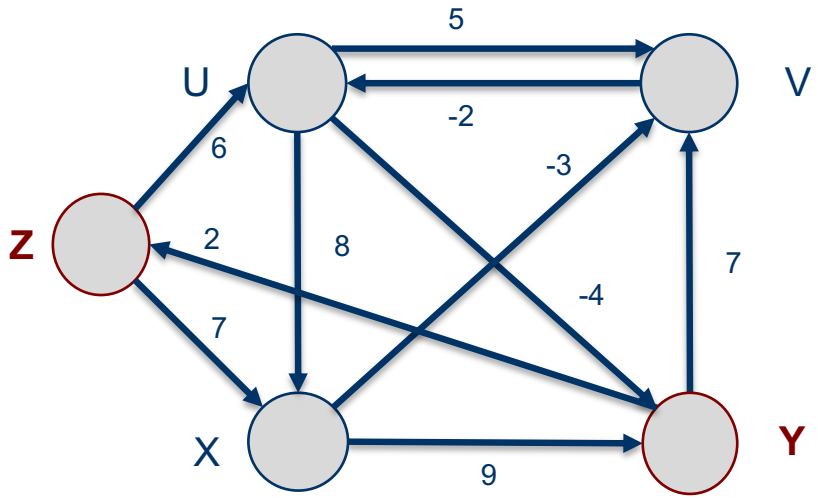
Verificación de ciclos negativos

En realidad es un Continue

Shortest path in graph

Algoritmo de Bellman Ford

Realizaremos una corrida del algoritmo con el siguiente ejemplo:
Source = Z, Target = Y



Para el demo se usaran los slides auxiliares

Shortest path in graph

Algoritmo de Bellman Ford

Pseudo-código

Max { O(V), O(VE), O(E)**}**

```
 Bellman-Ford(G,s)
1.    for each vertex u in G.V()
2.       u.setd(∞)
3.       u.setparent(NIL)
4.    s.setd(0)

5.    for i from 1 to |G.V()|-1 do
6.      for each edge (u,v) in G.E() do
7.            Relax (u,v,G)

8.    for each edge (u,v) in G.E() do
9.       if v.d() > u.d() + G.w(u,v) then
10.           return false

11.    return true
```

O(V)

O(1)

O(E)

O(VE)

O(E)

Shortest path in graph

Algoritmo de Bellman Ford

There are a lot of examples around the web:

- Visual examples
  - In general: http://visualgo.net/
  - Dijkstra y Bellmand-Ford: http://visualgo.net/sssp.html
- Tutorials
  - https://www-m9.ma.tum.de/graph-algorithms/spp-bellman-ford/index_en.html

Shortest path in graph

Algoritmo de Floyd-Warshall

Robert W (Bob) Floyd
1936-2001

Floyd-Warshall algorithm

Finds the shortest (longest) paths among all pairs of nodes in a graph, which:

- It does not contain any cycles of negative length.
- The main advantage is its simplicity.

- Floyd was a college roommate of Carl Sagan.
- Floyd worked closely with Donald Knuth, major reviewer for Knuth's seminal book.
- Turing Award in 1978.

Complejidad $O(V^3)$

Artículo original

- Floyd, Robert W. (junio de 1962). «Algorithm 97: Shortest Path». Communications of the ACM 5 (6): 345.

NO viene en el temario porque usa otra técnica pero es del problema SPP.

Shortest path in graph

Algoritmo de Floyd-Warshall

Floyd-Warshall algorithm

Finds:

- Shortest paths between all pairs of vertices,
- Though it does not return details of the paths themselves.
- Versions of the algorithm can also be used for:
  - Finding the transitive closure of a relation R, or
  - Widest paths between all pairs of vertices in a weighted graph.

It use **Dinamic Programming** that we'll see later.

Links

- https://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall

Shortest path in graph

Algoritmo de Floyd-Warshall

Floyd-Warshall algorithm

El algoritmo usa memoria cuadrática.

```
1. let dist be a |V| × |V| ∞
2. for each vertex v
3.     dist[v][v] ← 0

4. for each edge (u,v)
5.    dist[u][v] ← w(u,v)

6. for k=1 to |V|
7.    for i=1 to |V|
8.        for j=1 to |V|
9.            if dist[i][j] > dist[i][k] + dist[k][j]
10.               dist[i][j] ← dist[i][k] + dist[k][j]
11.           fi
```

1.- Array of minimum distances initialized to ∞ (infinity) $O(V^2)$

3.- Local loops set to 0. $O(V)$

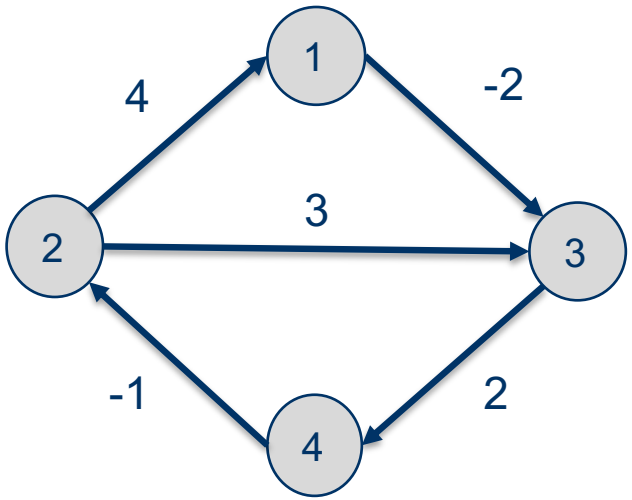
5.- The weight of the edge (u,v) $O(E)$

$O(V^3)$

Shortest path in graph

Algoritmo de Floyd-Warshall

Example



Usaremos
los
Slides auxiliares

The minimum spaning tree problem

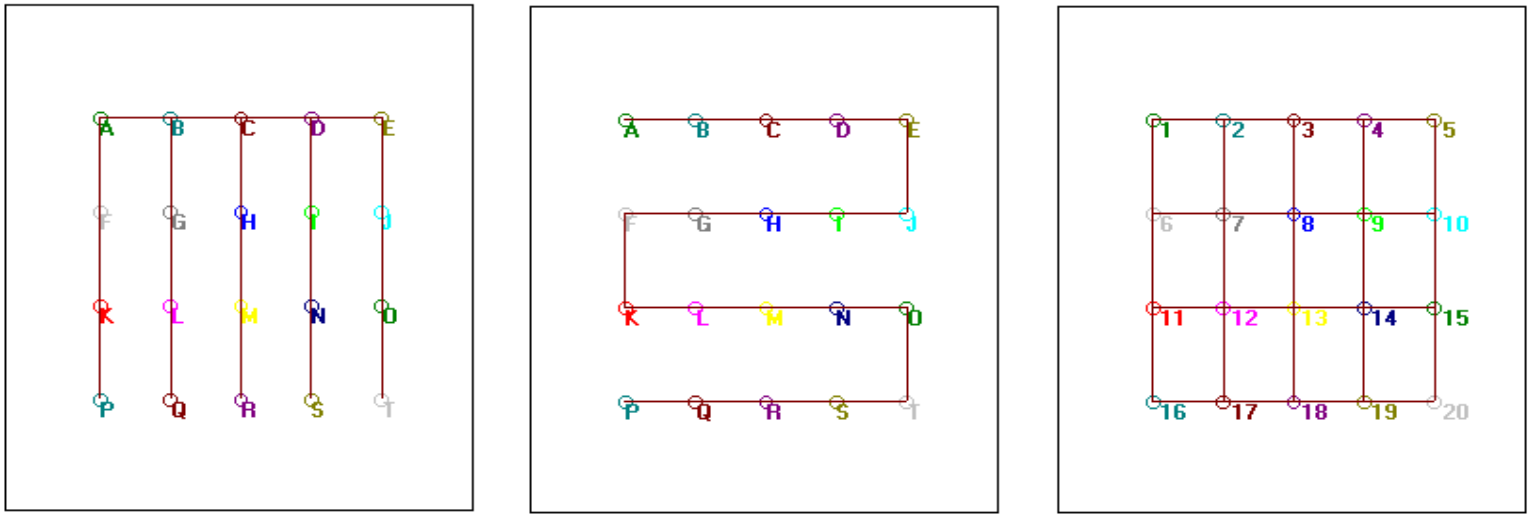
Árbol de expansión mínimo recubrimiento

Tema

The minimum spaning tree problem

El problema del árbol de expansión mínimo

- A *spanning tree* for a connected graph *G* is a connected subgraph of *G* that is a tree (i.e. has no circuits) and contains all the vertices of *G*. If G is a tree then it has only one spanning tree (*G* itself), otherwise there will be more than one spanning tree.
- **Pbm**: El MST de un grafo ponderado (weighted) es un conjunto de aristas cuyo peso total es mínimo y forma un árbol de expansión (todos los vértices están conectados).
- **Algoritmos**: Prim (1957), Kruscal (1956)
- En matemática existe la función MinimumSpanningTree [*g*].



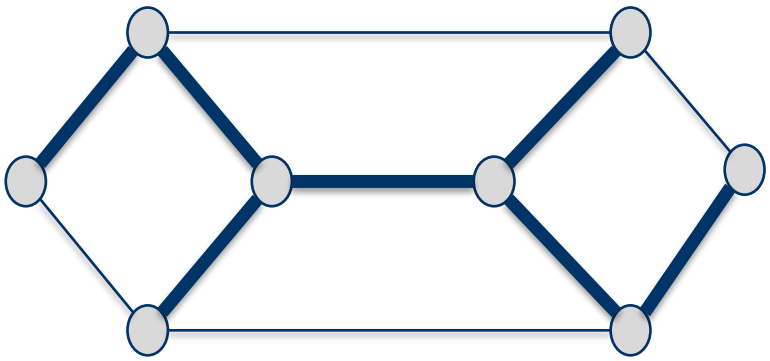
Spaning tree properties

Propiedades

The abbreviation "**TFAE**" is shorthand for "the following are equivalent". It is used before a set of equivalent conditions (each implies all the others).

Proposition. Let $T = (V, F)$ be a subgraph of $G = (V, E)$. TFAE:

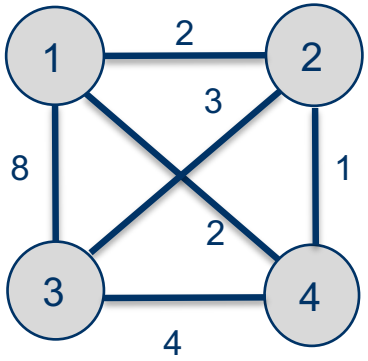
- T is a spanning tree of G.
- T is acyclic and connected.
- T is connected and has $n - 1$ edges.
- T is acyclic and has $n - 1$ edges.
- T is minimally connected: removal of any edge disconnects it.
- T is maximally acyclic: addition of any edge creates a cycle.
- T has a unique simple path between every pair of nodes.



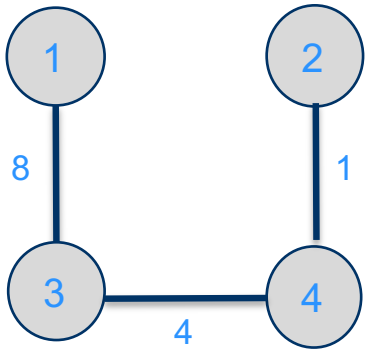
MST example

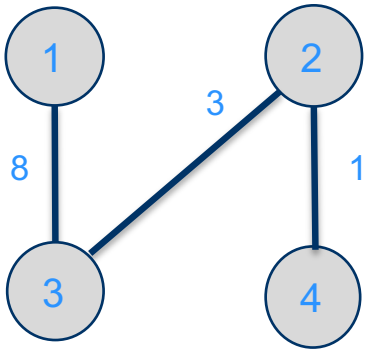
Ejemplo

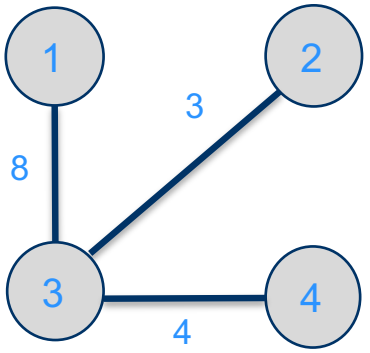
The graph:



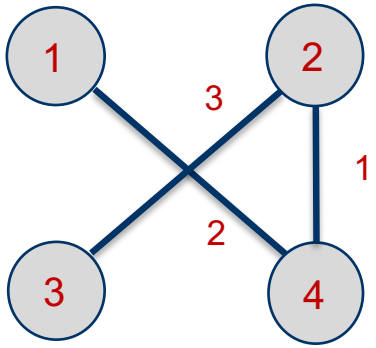
Has 16 spanning trees. Some are:

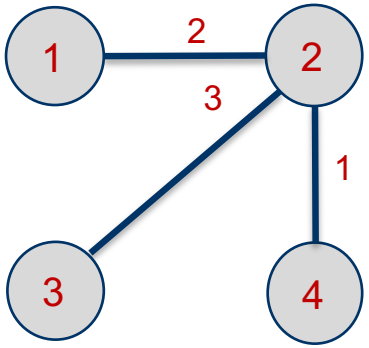


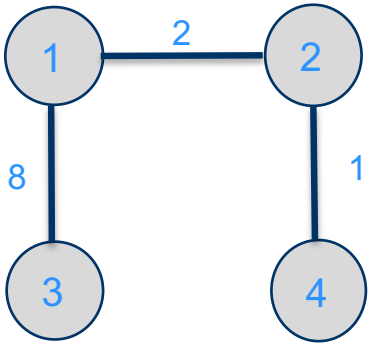


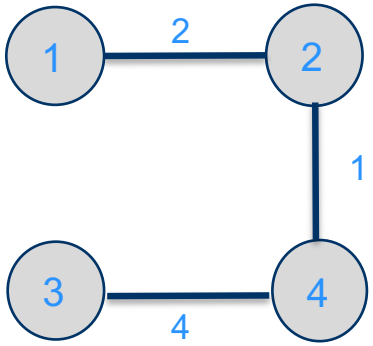


The graph has two minimum-cost spanning trees, each with a cost of 6:









Spaning tree properties

Algorithms

Some of the most known:

- Prim's algorithm, 1957.
- Kruscal's algorithm, 1956

} Se estudiarán estos 2.

- Boruvka/Solin 's algorithm.
- Hybrid algorithm (Prim & Boruvka).

MST Prim's algorithm

Algoritmo de Prim

1921 EU
Matemático
Computólogo

Prim's algorithm

- The algorithm was developed in 1930 by Czech mathematician Vojtěch Jarník.
- Later rediscovered and republished by:
  - Robert C. Prim in 1957.
  - Edsger W. Dijkstra in 1959.
  - Therefore, it is also sometimes called the ***DJP algorithm***, ***Jarník's algorithm***, ***Prim-Jarník algorithm***, or ***Prim-Dijkstra algorithm***.
- It finds a minimum spanning tree for a weighted undirected graph. Other well-known algorithms for this problem include: Kruskal's algorithm and Borůvka's algorithm. These algorithms find the minimum spanning forest in a **possibly disconnected** graph.
- In contrast, the most basic form of Prim's algorithm only finds minimum spanning trees in **connected graphs**.
- However, running Prim's algorithm separately for each connected component of the graph, it can also be used to find the minimum spanning forest.

MST Prim's algorithm

Algoritmo de Prim

In terms of their asymptotic time complexity:

- The three algorithms are equally fast for sparse (**escaso**) graphs, but slower than other more sophisticated algorithms.
- However, for graphs that are sufficiently dense, Prim's algorithm can be made to run in linear time, meeting or improving the time bounds for other algorithms.

The time complexity of Prim's algorithm depends on the data structures used for the graph and for ordering the edges by weight, which can be done using a priority queue. The following table shows the typical choices:

Minimum edge weight data structure	Time complexity (total)
Adjacency matrix, searching	$O(V^2)$
Binary heap and adjacency list	$O((V+E) \log V) = O(E \log V)$
Fibonacci heap and adjacency list	$O(E + V \log V)$

MST Prim's algorithm

Algoritmo de Prim

The algorithm may informally be described as performing the following steps:

1. Select any vertex.
   Initialize a tree T with that vertex.
2. Select the shortest edge connected to that vertex.

3. Select the shortest edge connected to any vertex already connected.
   It is better if you sort edges (out of T) in ascending order.
4. Repeat step 3 until all vertices have been connected.

MST Prim's algorithm

Algoritmo de Prim

```
Prim (Grafo G)
    For cada u en V[G] do
      distancia[u] = INFINITO
      padre[u] = NULL

    Añadir(cola,<u,distancia[u]>)

    distancia[u]=0
    while ! esta_vacia(cola)
       u = extraer_minimo(cola) // Q.extractMin
        for cada v adyacente a u do
           si ((v ∈ cola) && (distancia[v] > peso(u, v)) entonces
               padre[v] = u
               distancia[v] = peso(u, v)
                actualizar(cola,<v,distancia[v]>)  // Q.decreaseKey
```

MST Prim's algorithm

Algoritmo de Prim

```
Prim (Grafo G)
    For cada u en V[G] do
      distancia[u] = INFINITO
      padre[u] = NULL

    Añadir(cola,<u,distancia[u]>)

    distancia[u]=0
    while ! esta_vacia(cola)
       u = extraer_minimo(cola) // Q.extractMin
        for cada v adyacente a u do
           si ((v ∈ cola) && (distancia[v] > peso(u, v)) entonces
               padre[v] = u
               distancia[v] = peso(u, v)
                actualizar(cola,<v,distancia[v]>)  // Q.decreaseKey
```

O(V)

O(V)

O(log V)

O(log V)

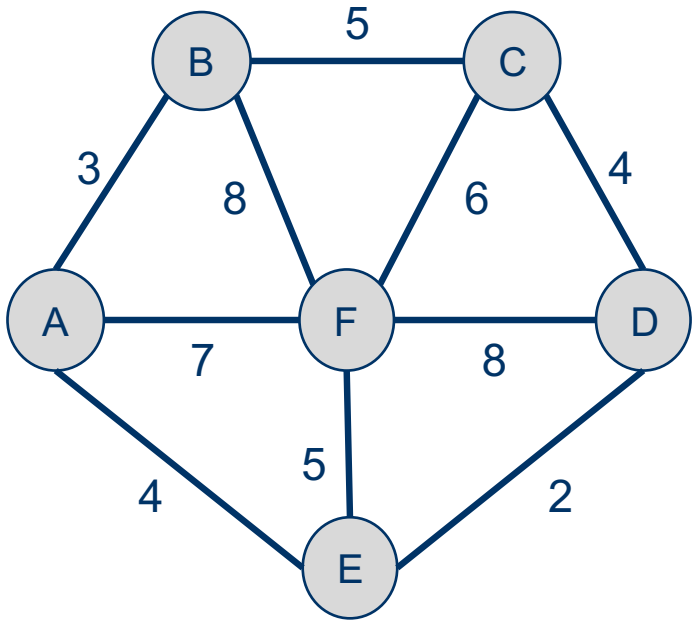
Total O((V+E) log V)

Terminar el análisis
Determinar donde surge E

MST Prim's algorithm

Algoritmo de Prim

Example



Usaremos
los
Slides auxiliares

MST Kruskal's algorithm

Algoritmo de Kruskal

1928 – 2010
Maplewood, Nueva Jersey
Fue un matemático y estadístico estadounidense.

Escrito por Joseph Kruskal y publicado en *Proceedings of the American Mathematical Society*, pp. 48–50 en 1956.

The algorithm may informally be described as performing the following steps:

1. Select the shortest edge in a network.
2. Select the next shortest edge which does not create a cycle.
3. Repeat step 2 until all vertices have been connected.

MST Kruskal's algorithm

Algoritmo de Kruskal

```
function Kruskal(G)

  for cada v en V[G] do
    Nuevo conjunto C(v) ← {v}.
  Nuevo heap Q (con todas las aristas de G, ordenando por peso)
  Crear árbol T ← Ø     (bosque)
  while T tenga menos de n-1 vertices do // n número total de vértices
   (u,v) ← Q.sacarMin()
   // Previene ciclos en T.
   // Agrega (u,v) si u y v están diferentes componentes en el conjunto.
   if C(v) ≠ C(u) then // C(u) devuelve la componente a la que pertenece u.
      Agregar arista (v,u) a  T.
      Merge C(v) y C(u) en el conjunto
  return arbol T
```

MST Kruskal's algorithm

Algoritmo de Kruskal

Kruskal's complexity

- Si M el número de aristas del grafo y N el número de vértices.
- La complejidad es O(m log m) o O(m log n), cuando se ejecuta sobre estructuras de datos simples.

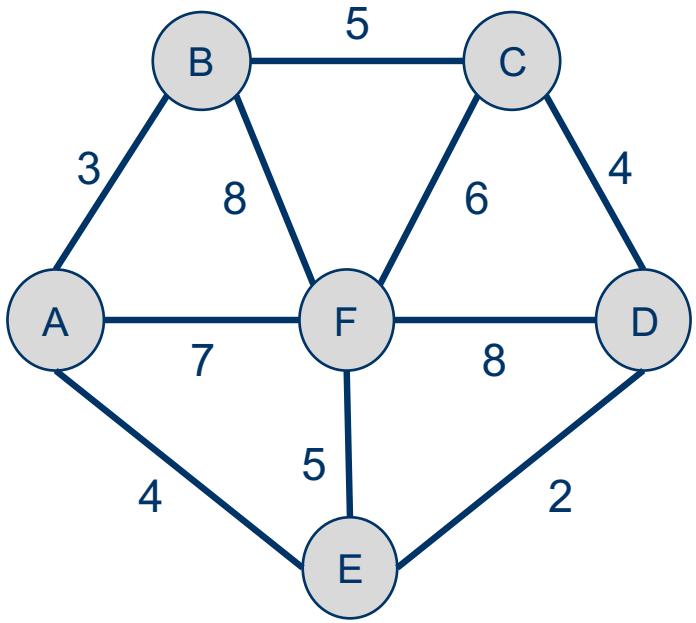
Proof

- Los tiempos de ejecución son equivalentes porque:
  - m es a lo sumo n^2 y log n^2 = 2 log n que a su vez es O(log n).
  - Ignorando los vértices aislados, los cuales forman su propia componente del árbol de expansión mínimo, n ≤ 2m, así que log n es O(log m).

MST Kruskal's algorithm

Algoritmo de Kruskal

Example



Usaremos
los
Slides auxiliares

MST Kruskal's algorithm

Algoritmo de Kruskal

Inverse Kruskal's algorithm

- It is an algorithm to find the MST
- Similar name is Reverse-delete algorithm.

MST problem

Prim vs Kruskal

Comparación

- Prim's algorithm initializes with a **node**, whereas Kruskal's algorithm initiates with an **edge**.
- Connectivity
  - In Prim's, you **always keep a connected component**, starting with a single vertex. Prim's algorithms span from one node to another.
  - In Kruskal's, you **do not keep one connected component** but a forest. Kruskal's algorithm select the edges in a way that the position of the edge is not based on the last step.
- In prim's algorithm, graph must be a **connected graph** while the Kruskal's can function on **disconnected graphs** too.
- Time complexity
  - Prim's algorithm is $O(V^2)$
  - Kruskal's is O(E log V).

MST problem

Prim vs Kruskal

Some points to note

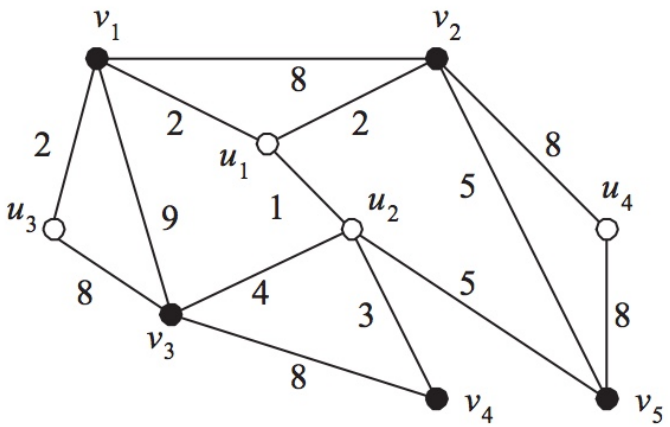
- Both algorithms will always give solutions with the same length.
- They will usually select edges/nodes in a different order.
  - This may happen when you have to choose between edges with the same length.
  - In this case, there is more than one minimum connector for the network.

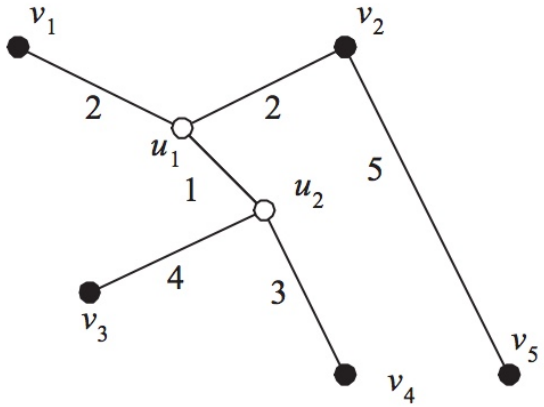
MST problem

Problemas similares

Steiner Tree Problem (STP)

- Given an undirected graph with non-negative edge weights and a subset of vertices, usually referred to as **terminals,** the Steiner tree problem in graphs requires a tree of minimum weight that contains all terminals (but may include additional vertices).





MST problem

Problemas similares

Steiner Tree Problem (STP)

- One well-known variant, which is often used synonymously with the term Steiner tree problem, is the Steiner tree problem in **graphs.**
- The Steiner tree problem in graphs can be seen as a generalization of two other famous combinatorial optimization problems:
  - If a Steiner tree problem in graphs contains exactly two terminals, it reduces to finding a shortest path (problem).
  - If, on the other hand, all vertices are terminals, the Steiner tree problem in graphs is equivalent to the minimum spanning tree (problem).

Links

- https://en.wikipedia.org/wiki/Steiner_tree_problem

MST problem

Problemas similares

Suizo
1796-1863
mathematician

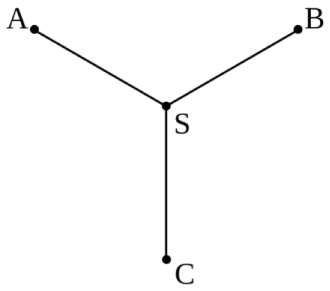
Steiner Tree Problem (STP)

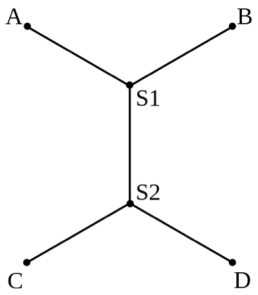
- **STP**, or **Minimum Steiner Tree Problem**, named after Jakob Steiner, is an umbrella term for a class of problems in combinatorial optimization.
- While Steiner tree problems may be formulated in a number of settings, they all require an optimal interconnect for a given set of objects and a predefined objective function.
- Further well-known variants are:
  - Euclidean Steiner tree problem.
  - Rectilinear minimum Steiner tree problem.
- Most versions of the Steiner tree problem are NP-hard, but some restricted cases can be solved in polynomial time.
  - Despite the pessimistic worst-case complexity, several Steiner tree problem variants, including the Steiner tree problem in graphs and the rectilinear Steiner tree problem, can be solved efficiently in practice, even for large-scale real-world problems.

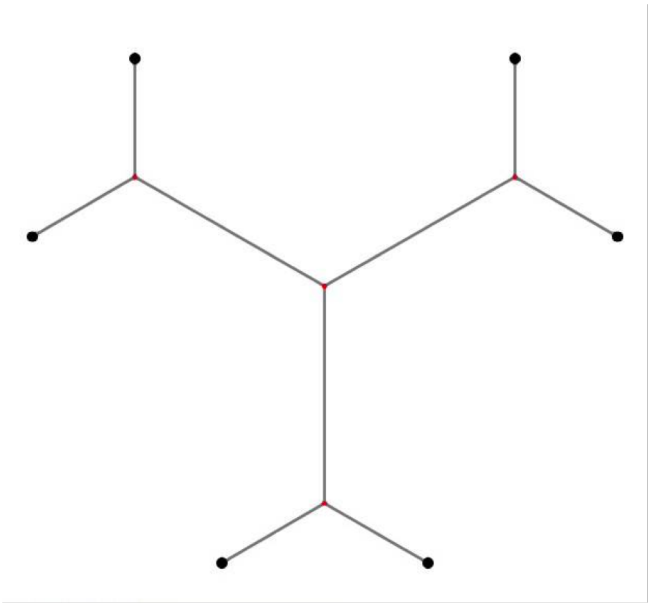
MST problem

Problemas similares

Steiner Tree Problem (STP)







- The **decision variant** of the Steiner tree problem in graphs is NP-complete (which implies that the optimization variant is NP-hard).
- In fact, the decision variant was among **Karp**'s original.

MST problem

Otros problemas

Union-find

- The problem of maintaining the connected components of the growing forest in Kruskal's algorithm is a special case of a more general problem called the **union-find problem**.
- A **union-find data structure** maintains a family of disjoint nonempty sets. Each set in the family has an element called its **label**. The data structure supports the following three operations:
  - MAKESET(x) creates a new set {x} and adds it to the family. The element x must not be an element of any existing set in the family.
  - UNION(x, y) changes the family by replacing two sets, the one containing x and the one containing y, by a single set that is the union of these two sets. It is a no-op if x and y are already in the same set.
  - FIND(x) returns the label of the set containing x.
- Note that the changes in the family of sets are monotone. Once an element appears it remains in some set in the family, and once two elements are in the same set they stay in the same set.

MST problem

Otros problemas

Union-find

- It is “easy” to see how to use such a data structure to imple- ment Kruskal’s algorithm. We first perform a MAKESET for each vertex of the graph. Then for each edge (u, v) we do a FIND for both u and v, followed by UNION(u, v) if they are not already in the same set.

Clustering

Agrupamiento

Tema

Clustering

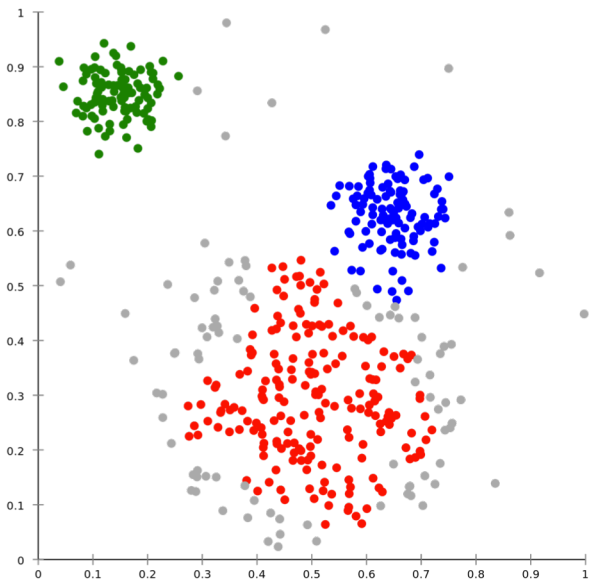
Agrupamiento

Goal

Given a set *U* of *n* objects labeled *p*1, ..., *pn*, partition into clusters so that objects in different clusters are far apart.

Applications

- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.
- ...



Clustering

Agrupamiento

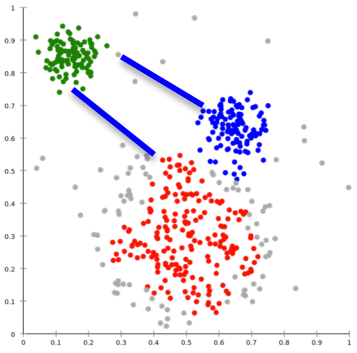
k-clustering. Divide objects into k non-empty groups.
Distance function. Numeric value specifying "closeness" of two objects.

- $d(pi, pj) = 0$ iff $pi = pj$ [identity of indiscernibles]
- $d(pi, pj) \geq 0$ [nonnegativity]
- $d(pi, pj) = d(pj, pi)$ [symmetry]

Spacing. Min distance between any pair of points in different clusters.

Goal. Given an integer k, find a k-clustering of maximum spacing.

3-clustering



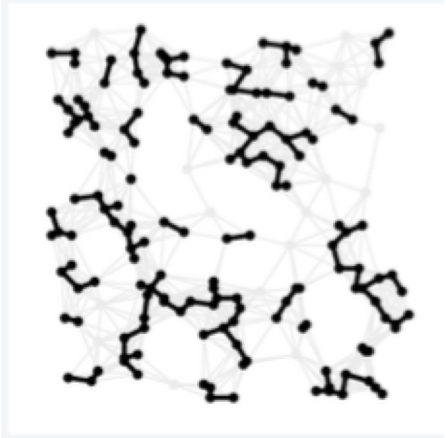
Distance between two clusters

Clustering

Agrupamiento

"Well-known" algorithm for single-linkage k-clustering:

- Form a graph on the node set *U*, corresponding to *n* clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat *n* – *k* times until there are exactly *k* clusters.



Key observation. This procedure is precisely Kruskal's algorithm.

Except we stop when there are *k* connected components.

Alternative. Find an MST and delete the *k* – 1 longest edges.

Clustering

Agrupamiento

Clasification of algorithms

- Exclusive clustering
  - K-means
- Overlapping clustering
  - Fuzzy C-means
- Hierarchical clustering (agglomerative)
- Probabilistic clustering
  - Mixture of Gaussians

No usan la estrategia Greedy
Usan una estrategia **Heuristica**
Pero que es?

Methodology/Strategy

Nuevas estrategias

George Pólya
1887-1985
Matemático hungaro

Heurística

- La palabra heurística procede del término griego εὑρίσκειν, que significa «hallar, inventar» (etimología que comparte con eureka).
- Actualmente se han hecho adaptaciones al término en diferentes áreas, así definen la heurística como un *Arte, técnica o procedimiento práctico o informal, para resolver problemas*.
- La heurística usualmente propone estrategias heurísticas que **guían el descubrimiento**. El término fue utilizado por:
  - George Pólya, con su libro *Cómo resolverlo* (*How to solve it*). Habiendo estudiado tantas pruebas matemáticas desde su juventud, quería saber cómo los matemáticos llegan a ellas. El libro contiene la clase de recetas heurísticas que trataba de enseñar a sus alumnos de matemáticas.
  - Albert Einstein en la publicación sobre efecto fotoeléctrico (1905), con el cual obtuvo el premio Nobel en Física en el año 1921 y cuyo título traducido al idioma español es: “Sobre un punto de vista *heurístico* concerniente a la producción y transformación de la luz”.

Methodology/Strategy

Nuevas estrategias

How to solve it (Cómo plantear y resolver problemas)

El libro (publicado en 1945) contiene la clase de recetas heurísticas que trataba de enseñar a sus alumnos de matemáticas.

- Técnica de cuatro etapas:
  1. Entender el problema.
  2. Crear un plan.
  3. Llevar a cabo el plan.
  4. Revisar e interpretar el resultado (mediante el *método científico*)
- Cuatro ejemplos que ilustran sus conceptos:
  1. Si no consigues entender un problema, dibuja un esquema.
  2. Si no encuentras la solución, haz como si ya la tuvieras y mira qué puedes deducir de ella *(razonando a la inversa)*.
  3. Si el problema es abstracto, prueba a examinar un ejemplo concreto.
  4. Intenta abordar primero un problema más general (es la “paradoja del inventor”: el propósito más ambicioso es el que tiene más posibilidades de éxito).

Methodology/Strategy

Nuevas estrategias

https://es.wikipedia.org/wiki/Cómo_plantear_y_resolver_problemas

How to solve it (Cómo plantear y resolver problemas)

Heurística	Descripción informal
Analogía	¿Puedes encontrar un problema análogo a tu problema y resolverlo?
Generalización	¿Puedes encontrar un problema más general que tu problema?
Inducción	¿Puedes resolver un problema a partir de una generalización de algunos ejemplos?
Variaciones del problema	Puedes modificar o cambiar el problema para crear un nuevo problema (o un conjunto de problemas) cuya solución pueda ayudarte a resolver el problema original?
Especialización	¿Puedes considerar un problema más restringido o especializado?
Dibuja un esquema	¿Puedes trazar un esquema del problema?
Trabajando hacia atrás a partir del objetivo	¿Puedes empezar con el objetivo y trabajar de manera inversa hasta algo conocido?
Aparece un problema relacionado con el tuyo y previamente resuelto	¿Puedes encontrar un problema relacionado con el tuyo que ya haya sido resuelto?

Clustering

Agrupamiento

K-means

Algorithm Basic K-means algorithm

Which points?
Heuristic choice

1: Select K points as initial centroids.
2: repeat
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: until Centroids do not change.

Clustering

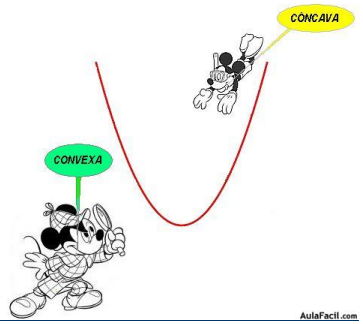
Agrupamiento

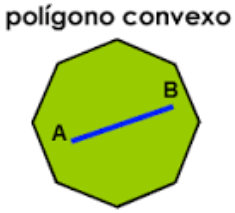
Centroide

- En Geometría, el centroide o baricentro de un objeto X perteneciente a un espacio n-dimensional es la intersección de todos los hiperplanos que dividen a X en dos partes de igual n-volumen con respecto al hiperplano.
- En Física, el centroide, el centro de gravedad y el centro de masas pueden, bajo ciertas circunstancias, coincidir entre sí, aunque designan conceptos diferentes.
  - El centroide es un concepto puramente geométrico que depende de la forma del sistema;
  - El centro de masas depende de la distribución de materia, mientras que
  - El centro de gravedad depende del campo gravitatorio.

Clustering

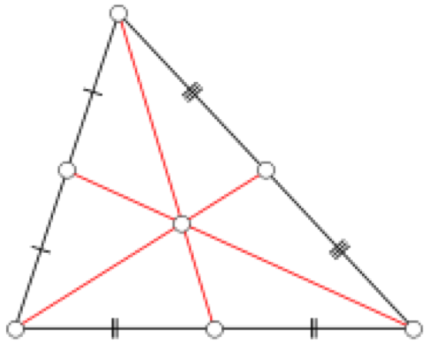
Agrupamiento





Centroide

- Una figura **cóncava** puede tener su centroide en un punto situado fuera de la misma figura.
- El centroide de una lámina con forma de cuarto de Luna estará en algún punto fuera de la lámina.
- El centroide de un triángulo (también llamado baricentro) se encuentra en el punto donde se intersecan sus transversales de gravedad:
  - Líneas que unen un vértice con el punto medio del lado opuesto.
  - Este punto es también el centroide de la superficie del triángulo.



Clustering

Agrupamiento

Questions

- How do you represent a cluster of more than one point?
- How do you determine the “nearness” of clusters?
- When to stop combining clusters?

Answers

- Each cluster has a well-defined centroid
  - Average across all the points
- Represent each cluster by its centroid.
- Distance between clusters = distance between centroids.

Clustering

Agrupamiento

“Closest” point?

Non-euclidean Distances

Possible meanings:

1. Smallest maximum distance to the other points.
2. Smallest average distance to other points.
3. Smallest sum of squares of distances to otherpoints.
4. Etc.

Distance measure

- Minkowski metric

Several topics

Varios temas

Notas históricas

Problemas abiertos

Tareas

URLs

A recordar

Historical notes

Notas históricas

- **George Dantzig** arrived late to class (1939), found some problems on the board and copied them down assuming they were homework.
- He took them home, worked them out. Six weeks later he gave his work to his professor, while apologizing for taking so long, because the questions were a bit harder than usual.
- What he didn't know was that those questions on the board weren't homework, but some examples of unsolved problems that were being discussed in class.
- A year later he asked his professor for help choosing his thesis topic and the professor just told him to put those two problems in a binder and he'll accept them as his thesis.

Turing Award

Premios Turing

Es un premio de las Ciencias de la Computación que es otorgado anualmente desde 1966 por la Asociación para la Maquinaria Computacional (**ACM**) a quienes hayan contribuido de manera trascendental al campo de las ciencias computacionales.

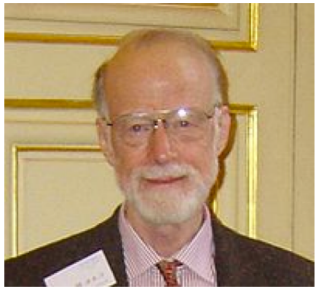
El galardón rinde tributo a Alan Turing y desde 2014 es patrocinado por Google, que recompensa con un premio de 1 000 000 de dólares estadounidenses al ganador.

1969 - Marvin Minsky
IA

1971 – John McCarthy
IA

1972 – Edsger Dijkstra
Lenguajes de Program.

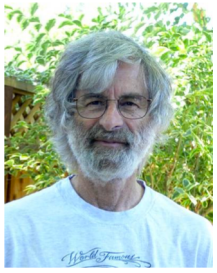
1974 – Donald Knuth
Alg. y Lenguajes



1980 – CAR Hoare
Lenguajes de Prog.

...

2004 – Cerf - Kahn
TCP/IP



2013 – L. Lamport
Sist. Distribuidos

2015 – Diffie - Hellman
Criptografía

Open problems

Problemas sin resolver o difíciles

Richard Karp
1935 EU
Premio Turing 1965.

Lista de 21 problemas NP-completos de Karp

- Es una lista de 21 problemas computacionales famosos, que tratan sobre **combinatoria y teoría de grafos**, y que cumplen la característica en común de que todos ellos pertenecen a la clase de complejidad de los NP-completos.
- Esta lista fue elaborada en 1972 por el informático teórico Richard Karp, en su trabajo seminal "Reducibility Among Combinatorial Problems" (Reducibilidad entre Problemas Combinatorios), como profundización del trabajo de Stephen Cook, quien en 1971 había demostrado uno de los resultados más importantes y pioneros de la complejidad computacional: la NP-completitud del Problema de satisfacibilidad booleana.
- El descubrimiento de Karp de que todos estos importantes problemas eran NP-completos, motivó el estudio de la NP-completitud y de la indagación en la famosa pregunta, de si P = NP.

Open problems

Problemas sin resolver o difíciles

Lista de 21 problemas NP-completos de Karp

- SAT (Problema de satisfacibilidad booleana, para fórmulas en forma normal conjuntiva)
  - 0-1 INTEGER PROGRAMMING (Problema de la programación lineal entera)
  - CLIQUE (Problema del clique, véase también Problema del conjunto independiente)
    - SET PACKING (Problema del empaquetamiento de conjuntos)
    - VERTEX COVER (Problema de la cobertura de vértices)
      - SET COVERING (Problema del conjunto de cobertura)
      - FEEDBACK NODE SET
      - FEEDBACK ARC SET
      - DIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano dirigido)
        - UNDIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano no dirigido)
  - 3-SAT (Problema de satisfacibilidad booleana de 3 variables por cláusula)
    - CHROMATIC NUMBER (Problema de la coloración de grafos)
      - CLIQUE COVER (Problema de la cobertura de cliques)
      - EXACT COVER (Problema de la cobertura exacta)
        - HITTING SETSTEINER TREE3-DIMENSIONAL MATCHING (Problema del matching tridimensional)
        - KNAPSACK (Problema de la mochila)
          - JOB SEQUENCING (Problema de las secuencias de trabajo)
          - PARTITION (Problema de la partición)
            - MAX-CUT (Problema del corte máximo)

Homeworks

Tareas

H4.1

Research the list of algorithms and each one is goint to implement one differente.

Input file

```
5          K=5 clusters
1,2        Point 1
…          Poin N
3,4
```

Output file

```
1
1,2
2
3,4
```

Make a graph with gnuplot putting the same color for points in a cluster.

Recommended reading and Web sites

Lecturas recomendadas

Remember that this can change at any time (o estar fuera de linea)

- http://scanftree.com/Data_Structure/kruskal's-algorithm
- http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/kruskalAlgor.htm

Clustering

- http://scikit-learn.org/stable/modules/clustering.html

To remember

A recordar

Lo más importante es:

- Un algoritmo greedy/paradigma:
  - Siempre hace la elección que parece la mejor en el momento.
  - Es decir hace una elección optima local esperando que le lleve a la solución óptima global.
  - NO siempre lleva a la solución optima.
- Identificar las características de los problemas que se resuelven con este paradigma. Los problemas son:
  - El camino más corto: una fuente, alg. dijkstra.
  - El árbol de expansión mínimo.
  - Heurística de set-covering (Chvatal).

To remember

A recordar

Problemas/Algoritmos, Comentarios

- The counting money problem
  - Cashiers Algo. Da una solución O(nlg n), pero no óptima.
- Interval scheduling
  - Earliest_deadline_first Algo.

Problems

Problemario

Diseñar algoritmo

Probar su correctitud

Calcular su complejidad

List of problems

Lista

1. Interval Partitioning
   - Input: A set of lectures, with start and end times.
   - Goal: Find the minimum number of classrooms, needed to schedule all the lectures such two lectures do not occur at the same time in the same room.

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652

Homeworks

Tareas

Homework (opcional)

Biography

1. Cartel
2. Biografia de un computólogo. Extensa
3. Mini-biografia (hoja).

Cada alumno un computólogo diferente.

Libro

Se hara una "publicación" de divulgación científica.

Organizada por campos de la computación.

Con secciones como Premios (Turing)