Centro de
Investigación en
Computación

Dynamic Programming

Programación dinámica

Tema 6

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

Algorithmic paradigms

Greedy

Build up a solution incrementally, myopically optimizing some local criterion.

Divide-and-conquer

Break up a problem into independent subproblems, solve each subproblem, and combine solution to subproblems to form solution to original problem.

Dynamic programming

Subproblemas superpuestos y
Subestructuras óptimas

Break up a problem into a series of overlapping subproblems, and build up solutions to larger and larger subproblems.

Introduction

Introducción

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

Etymology

- Dynamic programming = planning over time.
- Secretary of Defense (Wilson, 1953) was hostile to mathematical research.
- Bellman sought an impressive name to avoid confrontation.
- Fancy name for caching away intermediate results in a table for later reuse.

Dynamic programming is both:

- A mathematical optimization method and a computer programming method.
- In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

.

Problemarios

Guía

Algunos ejemplos (links)

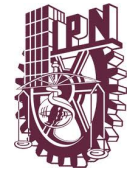
1. Code Chef — Concursos hackathon
   i. https://www.codechef.com/
   ii. https://www.codechef.com/wiki/tutorials Varios Tutoriales de Algoritmia.
   iii. https://www.codechef.com/wiki/tutorial-dynamic-programming
2. Top Coder
   i. https://www.topcoder.com/
   ii. https://www.topcoder.com/community/data-science/data-science-tutorials/

Bellman did a lot of papers:

- https://www.chessprogramming.org/Richard_E._Bellman

6.1 Weighted interval scheduling

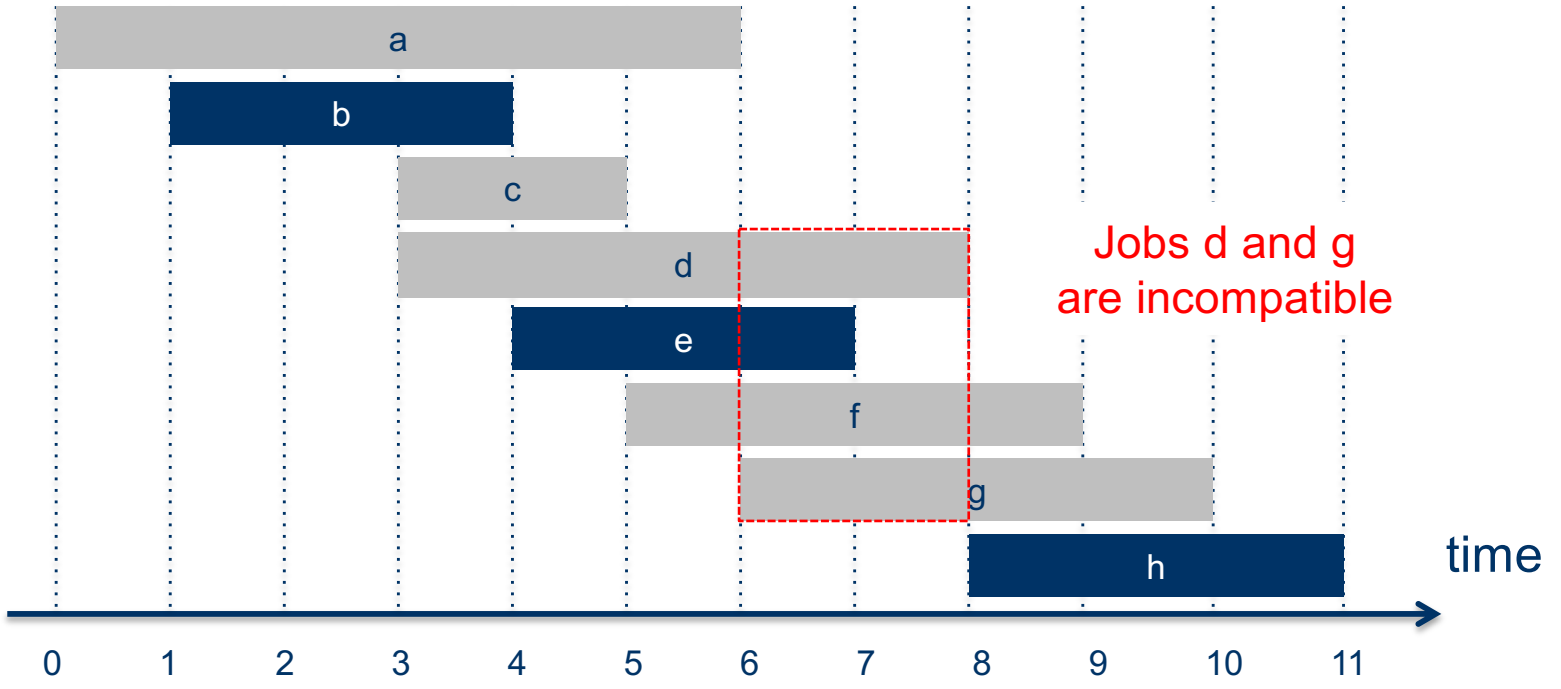
Planificador con intervalos ponderados



6.1 Weighted interval scheduling

Planificador con intervalos ponderados

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



6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Earliest finish-time first

- Consider jobs in ascending order of finish time.
- Add job to subset if it is compatible with previously chosen jobs.

Recall

Greedy algorithm is correct if all weights are 1.

Se vio en la
Unidad 4, T4

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Referencias

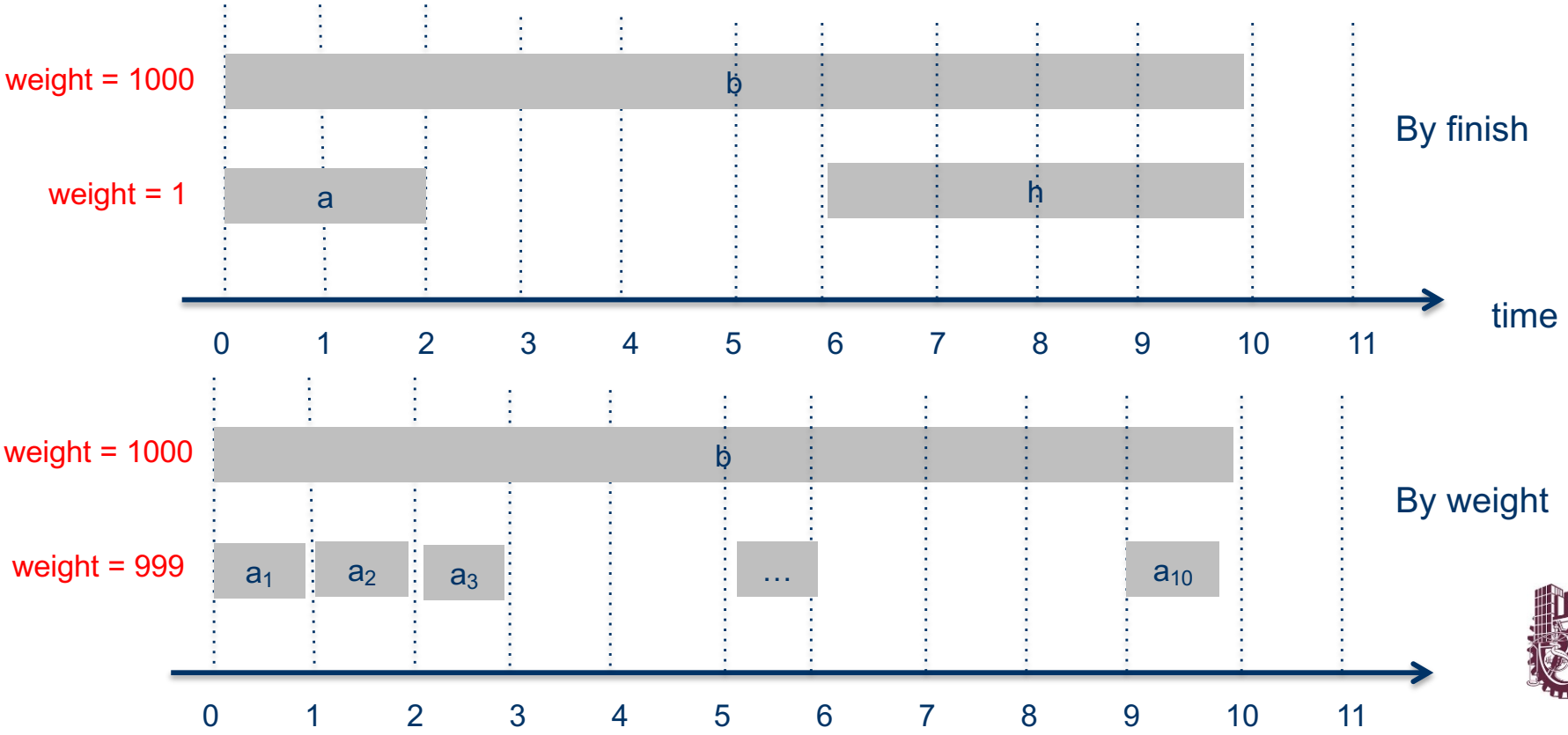
- https://www.techiedelight.com/weighted-interval-scheduling-problem/.
- https://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect10-dp-intv-sched.pdf

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Observation

Greedy algorithm fails spectacularly for weighted version.



6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Notation

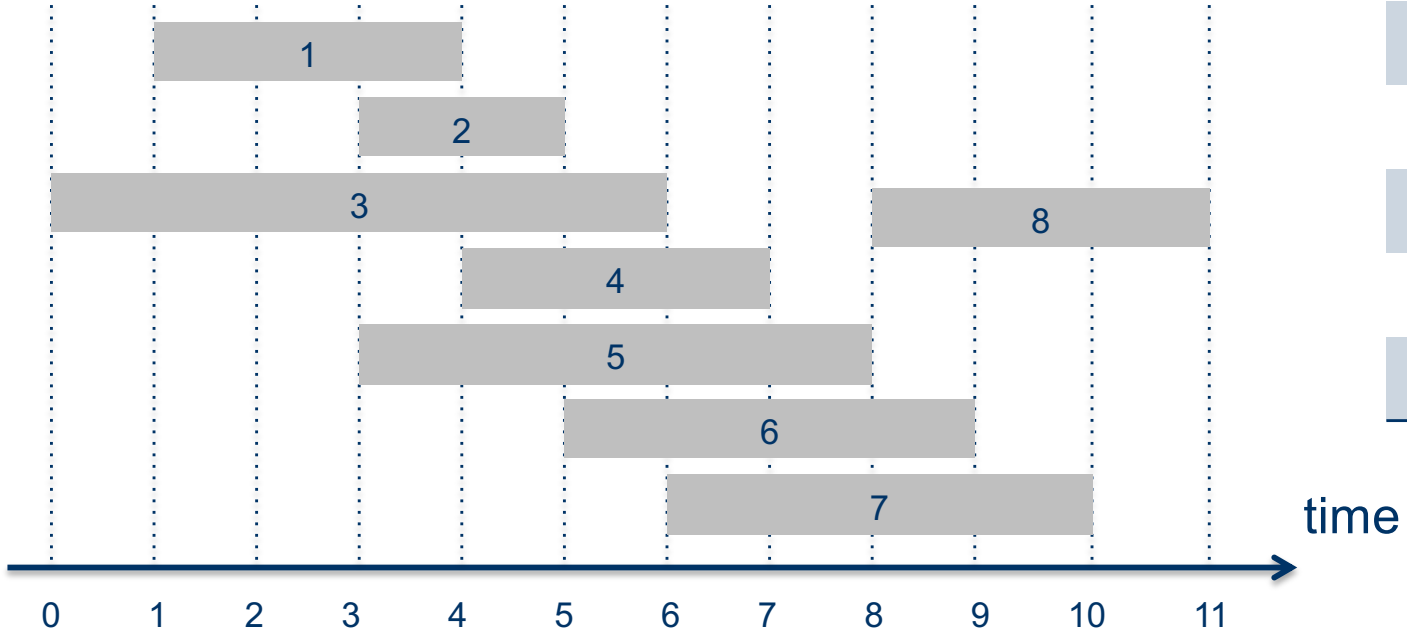
Label jobs by finishing time: $f_1 \leq f_2 \leq \ldots \leq f_n$.

Definition

$p(j)$ = largest index $i < j$ such that job i is compatible with j.

Example

$p(8) = 5$, $p(7) = 3$, $p(2) = 0$.



j	p(j)
0	-
1	0
2	0
3	0
4	1
5	0
6	2
7	3
8	5

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Notation

OPT(*j*) = value of optimal solution to the problem consisting of job requests 1, 2, ..., *j*.

Binary choise

Case 1. *OPT* selects job *j*.

- Collect profit ℓ_j. Es el peso.
- Can't use incompatible jobs { *p*(*j*) + 1, *p*(*j*) + 2, ..., *j* – 1 }.
- Must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *p*(*j*).

optimal substructure property (proof via exchange argument)

Case 2. *OPT* does not select job *j*.

- Must include optimal solution to problem consisting of remaining compatible jobs 1, 2, ..., *j* – 1.

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

$$OPT(j) = \begin{cases} 0 & \text{if } j=0 \\ \max\{ \ell_j + OPT(p(j)), OPT(j-1) \} & \text{Otherwise} \end{cases}$$

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Algorithm by Brute force

Input:

n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n].

Compute p[1], p[2], ..., p[n].

Compute-Opt(j)

```
if j = 0
  return 0
else
  return max(ℓ[j] + Compute-Opt(p[j]), Compute-Opt(j–1))
```

6.1 Weighted interval scheduling

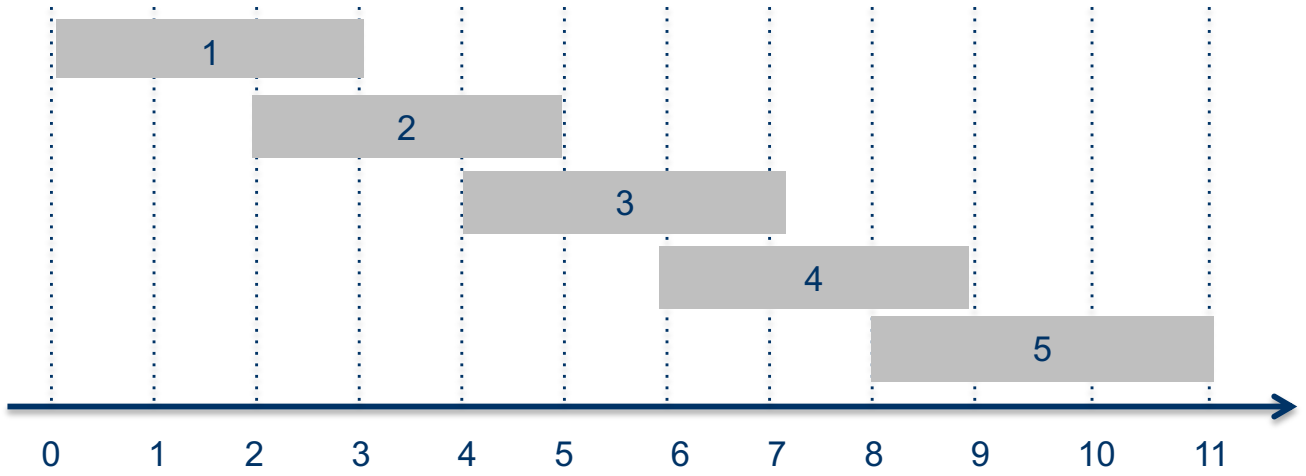
Planificador con intervalos ponderados

Observation

Recursive algorithm fails spectacularly because of redundant subproblems ⇒ exponential algorithms.

Example

Number of recursive calls for family of "layered" instances grows like Fibonacci sequence.



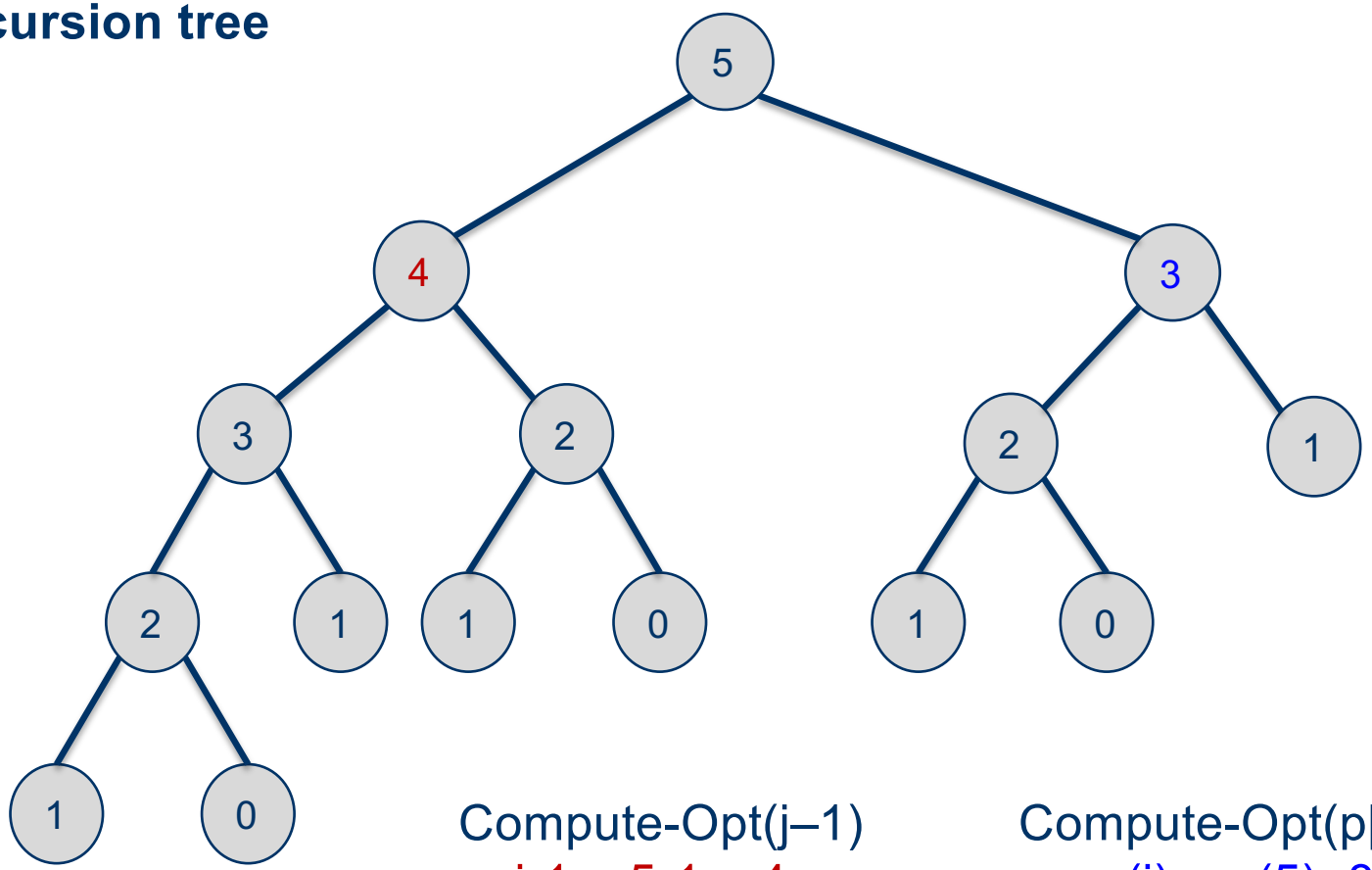
j	p(j)
0	-
1	0
2	0
3	1
4	2
5	3

p(1) = 0
p(j) = j-2

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Recursion tree



Compute-Opt(j–1)
j-1 = 5-1 = 4

Compute-Opt(p[j])
p(j) = p(5)=3

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Algorithm by Memoization

Cache results of each subproblem; lookup as needed

Input:

n, s[1..n], f[1..n], ℓ[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n].

Compute p[1], p[2], ..., p[n].

for j = 1 to n

M[j] ← empty.

M[0] ← 0.

M-Compute-Opt(j)

if M[j] is empty

M[j] ← max(ℓ[j] + M-Compute-Opt(p[j]), M-Compute-Opt(j–1))

return M[j]

Etymology

- The term "memoization" was coined by Donald Michie in 1968.
- It is derived from the Latin word "memorandum" ("to be remembered").
- It carries the meaning of "turning [the results of] a function into something to be remembered."
- While "memoization" might be confused with "memorization" (because they are etymological cognates), "memoization" has a specialized meaning in computing.

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Claim.

Memoized version of algorithm takes $O(n \log n)$ time.

- Sort by finish time: O(n log n).
- Computing p(·) : O(n log n) via sorting by start time.
- M-COMPUTE-OPT(*j*): each invocation takes $O(1)$ time and either:
  - Returns an existing value M[j].
  - Fills in one new entry M[j] and makes two recursive calls.
- Progress measure Φ = # nonempty entries of M[].
  - Initially $\Phi = 0$, throughout $\Phi \leq n$.
  - (ii) increases Φ by 1 $\Rightarrow$ at most $2n$ recursive calls.
- Overall running time of M-COMPUTE-OPT(n) is $O(n)$.

Remark. $O(n)$ if jobs are presorted by start and finish times**.**

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Question

DP algorithm computes optimal value. How to find solution itself?

Answer

Make a second pass.

```
Find-Solution(j)
  if j = 0
    return ∅
  else if (ℓ[j] + M[p[j]] > M[j–1])
    return { j } ∪ Find-Solution(p[j])
  else
    return Find-Solution(j–1)
```

This If-Else implements The max.

Analysis # of recursive calls ≤ $n \Rightarrow O(n)$

6.1 Weighted interval scheduling

Planificador con intervalos ponderados

Bottom-up DP

Unwind recursion.

BOTTOM-UP (n, s_1, ..., s_n , f_1, ..., f_n , ℓ_1, ..., ℓ_n)

Sort jobs by finish time so that $f_1 \le f_2 \le \ldots \le f_n$

Compute $p(1)$, $p(2)$, ..., $p(n)$

$M[0] \leftarrow 0$

for j = 1 to n

$M[j] \leftarrow \max \{ \ell_j + M[p(j)], M[j-1] \}$

6.2 Principles of Dynamic Programming

Principios de Programación dinámica

6.2 Principles of Dynamic Programming

Principios de Programación dinámica

Working principles

1. Divide into subproblems
2. Construction of table for storage
3. Combining using Bottom-up means

Selection of optimal (decision rules)

1. The principle of Optimality
   - An optimal sequence of decision is obtained iff each subsequence must be optimal.
   - That means if the initial state and decisions are optimal then the remaining decision must constitute an optimal sequence.
   - Combinatorial problems may have this property but may exploit too much memory and/or time towards efficiency.
2. Polynomial Break up.

6.2 Principles of Dynamic Programming

Principios de Programación dinámica

Limitations

What kind of problems can be solved using DP?

1. Evidently, optimization problems.
2. It works best on objets which are linearly ordered an cannot be rearranged such as:
    1. Characters in a string,
    2. Points around the boundary of a polygon,
    3. Matrices in a chain,
    4. The left-to-right order of leaves in a search tree, etc.

Shortcomings (mistake, error(defectos)

- It is often nontrivial to write code that evaluates the subproblems in the most efficient order.
- The challenge of devising a good solution method is in steps forward to make decisions what are the subproblems, how they would be computed and in what order.

6.2 Principles of Dynamic Programming

Principios de Programación dinámica

Strengths (fuerza, fortaleza)

1. Creativity is necessary before we can distinguish that a particular problem can be casted effectively as a dynamic program.
2. This idea of reusing sub-problems is the main advantage.
3. Selection of optimal decision rules (Optimality and Polynomial) which optimizes performance criterion.

6.3 Segmented Least Squares

Mínimos Cuadrados

6.3 Segmented least squares

Mínimos cuadrados

Gauss
1777-1855
Aleman

Historia

- El día de Año Nuevo de 1801, el astrónomo italiano Giuseppe Piazzi descubrió el planeta enano Ceres. Fue capaz de seguir su órbita durante 40 días.
- Durante el curso de ese año, muchos científicos intentaron estimar su trayectoria con base en las observaciones de Piazzi: resolver las ecuaciones no lineales de Kepler de movimiento es muy difícil.
- La mayoría de las evaluaciones fueron inútiles; el único cálculo suficientemente preciso para permitir a Franz Xaver von Zach, astrónomo alemán, reencontrar a Ceres al final del año fue el de **Carl Friedrich Gauss** (tenia 24 años): los fundamentos de su enfoque ya los había planteado en 1795, cuando aún tenía 18 años.
- Sin embargo, su método de mínimos cuadrados no se publicó sino hasta **1809**, y apareció en el segundo volumen de su trabajo sobre mecánica celeste, "*Theoria Motus Corporum Coelestium in sectionibus conicis solem ambientium*".
- El francés Adrien-Marie Legendre desarrolló el mismo método de forma independiente en 1805.
- En 1829, Gauss fue capaz de establecer la razón del éxito maravilloso del procedimiento: el argumento concreto se conoce como **teorema de Gauss-Márkov**.

6.3 Segmented least squares

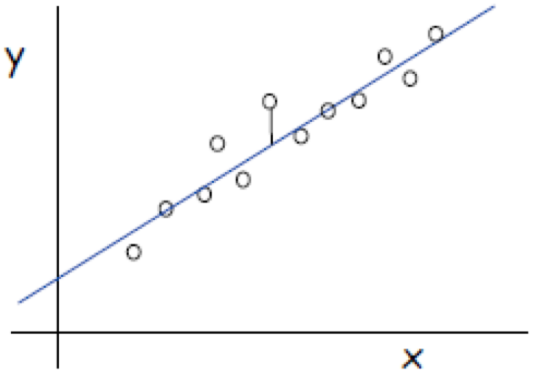
Mínimos cuadrados

Least squares. Foundational problem in statistics.

- Given n points in the plane: (x_1, y_1), (x_2, y_2) , ..., (x_n, y_n).
- Find a line $y = ax + b$ that minimizes the **S**um of the **S**quared **E**rror:

$$SSE = \sum_{i=1}^{n} (y_i - ax_i - b)^2$$

Error $e_k = y_k - f(x_k)$



Ecuación de una recta
f(x)=y = mx +b

- Slope (pendiente) is often denoted by the letter m (not a). It is no clear answer to the question why the letter m is used for, but it might be from the "m for multiple".

6.3 Segmented least squares

Mínimos cuadrados

Varias formas de medir el error (diferencia):

$$\text{Error}_{\text{Máximo}} = \max(e_k), \quad \text{Error}_{\text{Medio}} = \left(\sum_{k=1}^{n} e_k\right) / n$$

Solución. Calcular ⇒ El error mínimo que se alcanza cuando:

$$m = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2} \qquad b = \frac{\sum_i y_i - m \sum_i x_i}{n}$$

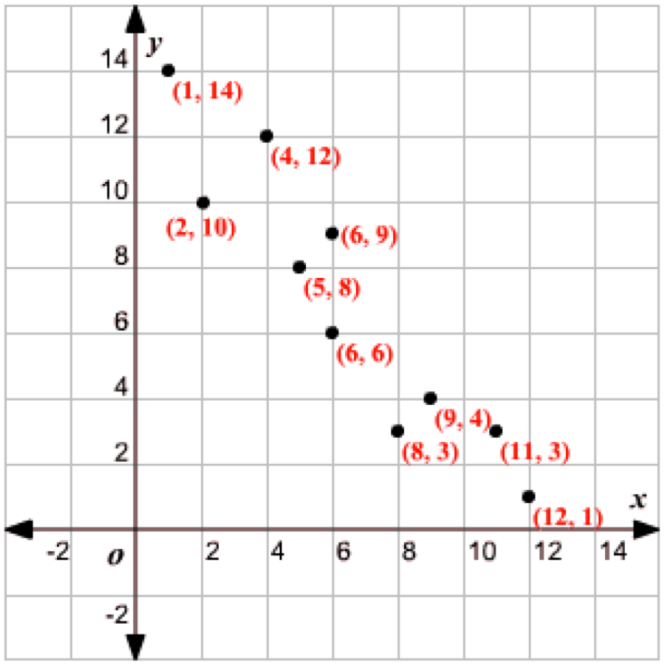
6.3 Segmented least squares

Mínimos cuadrados

Ejercicio

Calcular y=mx+b para los siguientes datos (puntos):

	1	2	3	4	5	6	7	8	9	10
x	8	2	11	6	5	4	12	9	6	1
y	3	10	3	6	8	12	1	4	9	14



Reescribiendo las ecuaciones de m y b anteriores se obtienen las siguientes:

6.3 Segmented least squares

Mínimos cuadrados

Ejercicio

Calcular y=mx+b

Para eso se requiere:

$$m = \frac{\Sigma xy - \frac{(\Sigma x)(\Sigma y)}{n}}{\Sigma x^2 - \frac{(\Sigma x)^2}{n}}$$

media(x) = Σ x /n

media(y) = Σ y /n

b = media(y) - m*media(x)

Resultados:

m = -1.1

media(x)=6.4

media(y)=7.0

b=14.0

y = -1.1 x + 14

6.3 Segmented least squares

Mínimos cuadrados

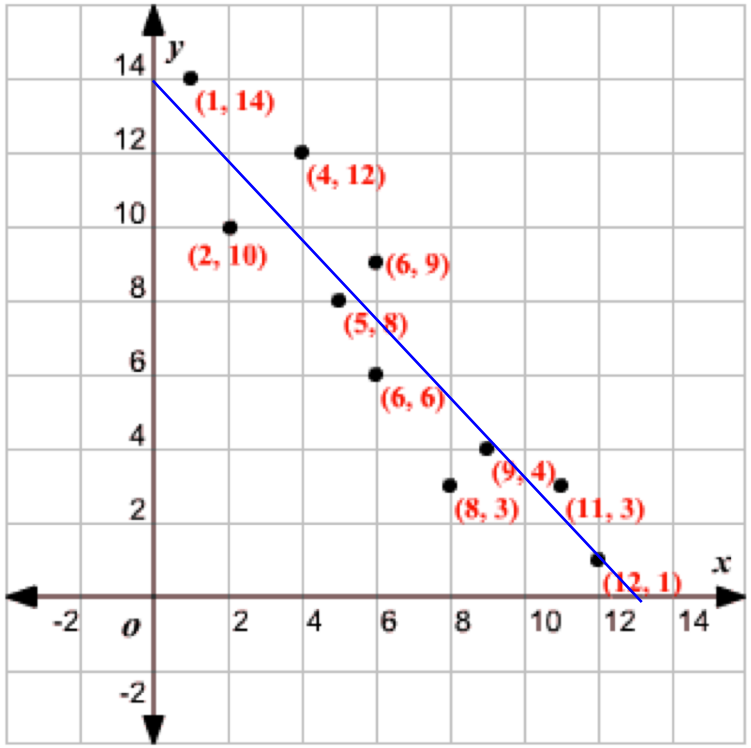
Ejercicio

Calcular y = mx+b = -1.1 x + 14

Si x=0 => y=14

Si y=0 => x = (-14/-1.1)

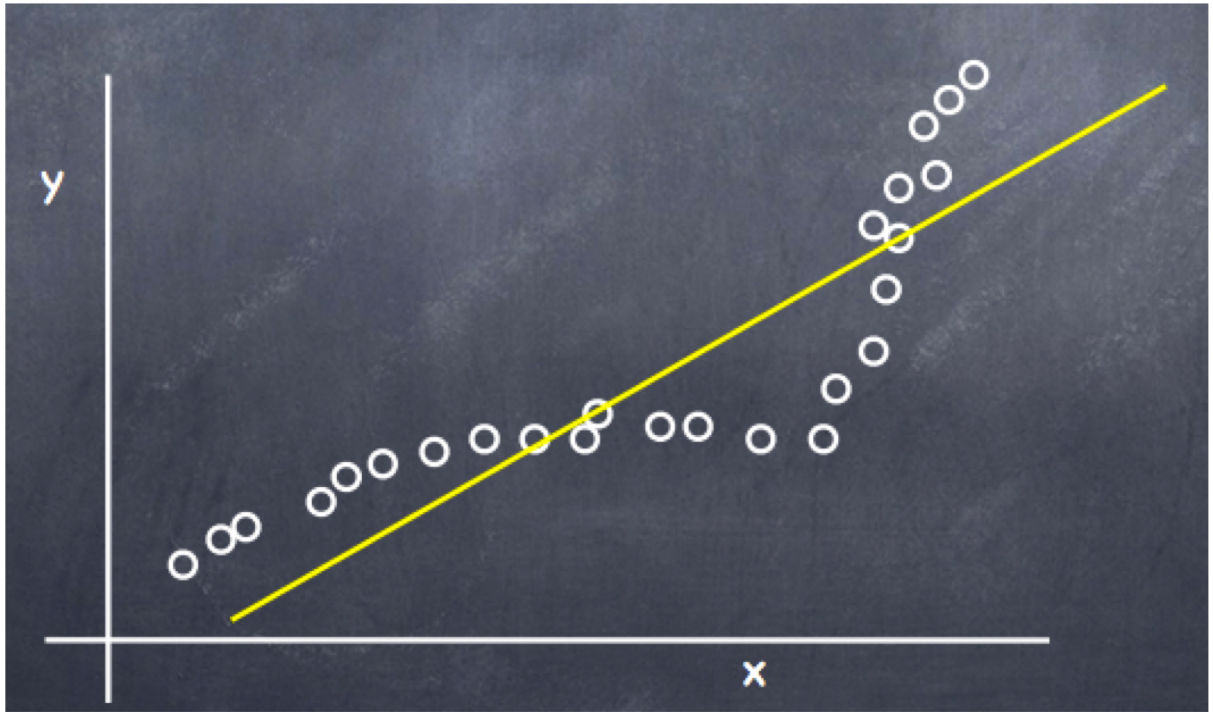
x=12.72



6.3 Segmented least squares

Mínimos cuadrados

Sometimes a single line does not work very well:



6.3 Segmented least squares

Mínimos cuadrados

Segmented least squares

- Points lie roughly on a sequence of several line segments.
- Given n points in the plane: (x_1, y_1), (x_2, y_2) , ..., (x_n, y_n) with:

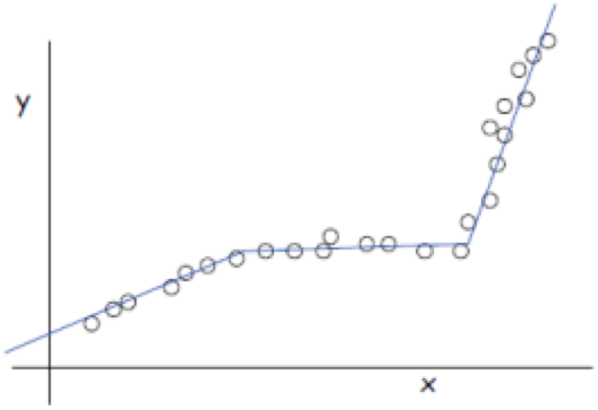
$$x_1 < x_2 < \ldots < x_n,$$

find a sequence of lines that minimizes $f(x)$.

Question. What is a reasonable choice for $f(x)$ to balance accuracy and parsimony?

goodness of fit

number of lines



6.3 Segmented least squares

Mínimos cuadrados

- Given n points in the plane:

$$(x_1, y_1), (x_2, y_2) , ..., (x_n, y_n)$$

with $x_1 < x_2 < ... < x_n$ and a constant $c > 0$,

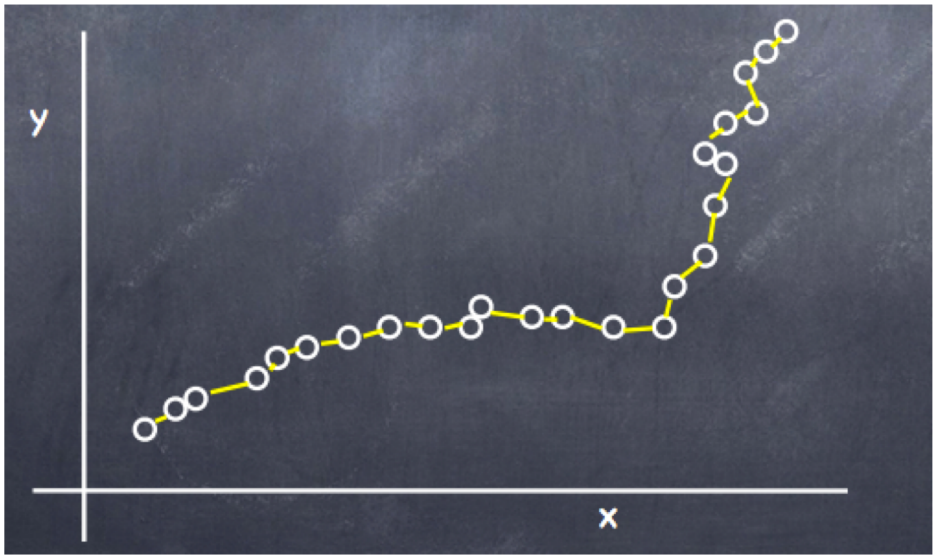
- Find a sequence of lines that minimizes $f(x) = E + c\,L$:
  - E = the sum of the sums of the squared errors in each segment.
  - L = the number of lines.

6.3 Segmented least squares

Mínimos cuadrados

Issue: how many lines?

- With too many lines, you can get a perfect solution, but there may be a much simpler explanation (e.g., two lines).



6.3 Segmented least squares

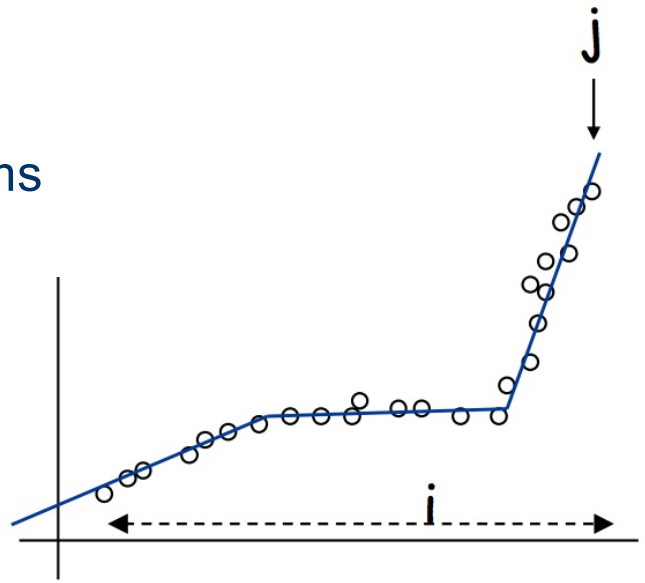
Mínimos cuadrados

Designing the solution

Notation

- $OPT(j)$ = minimum cost for points p_1, p_2, ..., p_j.
- $e(i, j)$ = minimum sum of squares errors for points p_i, p_{i+1}, ..., p_j.

We want to
create subpbms
How?



6.3 Segmented least squares

Mínimos cuadrados

Designing the solution

optimal substructure property
(proof via exchange argument)

To compute $OPT(j)$:

- Last segment uses points p_i, p_{i+1}, ..., p_j for some i.
- Cost = $e(i, j) + C + OPT(i - 1)$.

$$\text{OPT(j)} = \begin{cases} 0 & \text{if } j=0 \\ \min_{1 \le i \le j} \{ e(i,j) + C + OPT(i-1) \} & \text{Otherwise} \end{cases}$$

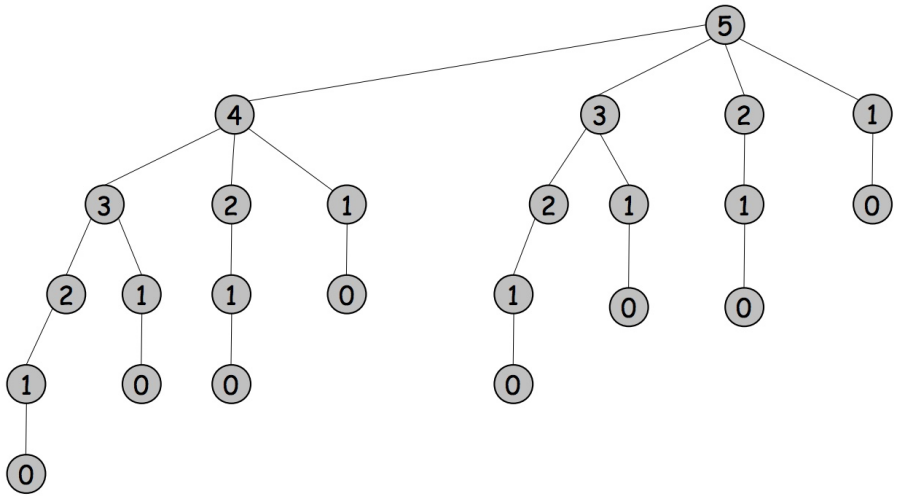
6.3 Segmented least squares

Mínimos cuadrados

Designing the solution

$$OPT(j) = \begin{cases} 0 & \text{if } j=0 \\ \min_{1 \le i \le j} \{ e(i,j) + C + OPT(i-1) \} & \text{Otherwise} \end{cases}$$

OPT(j) call graph



6.3 Segmented least squares

Mínimos cuadrados

The penalty of a partition is defined to be the sum of following two terms:

1. The number of segments multiplied by a given multiplier $C > 0$.
2. For each segment, the error value of the optimal line through that segment.

The goal is to find a partition of minimum penalty:

- Increasing the number of segments reduces the penalty term (2),
- But increases the penalty term (1).

The multiplier C is part of the input.

6.3 Segmented least squares

Mínimos cuadrados

SEGMENTED-LEAST-SQUARES (n, p_1, ..., p_n , c)

for j = 1 to n
  for i = 1 to j
    Compute $e(i, j)$ for the segment p_i, p_{i+1}, ..., p_j.

$M[0] \leftarrow 0$

for j = 1 to n
  $M[\,j\,] \leftarrow \min_{1 \le i \le j} \{\, e(i,j) + C + M[\,i - 1]\,\}$.

return $M[n]$

M[] is OPT() with memoization

6.3 Segmented least squares

Mínimos cuadrados

Theorem [Bellman 1961]

The dynamic programming algorithm solves the segmented least squares problem in $O(n^3)$ time and $O(n^2)$ space.

Proof

- There are $O(n^2)$ pairs (i,j).
- For which we need to compute e(i, j).
  - Each e(i,j) takes O(n) time.
- Thus all e(i,j) values can be computed in $O(n^3)$.
- Once e(i,j) values are computed, the OPT array can be filled in $O(n^2)$ time.

Remark

Can be improved to $O(n^2)$ time and $O(n)$ space by precomputing various statistics. How? Tarea optativa.

6.3 Segmented least squares

Mínimos cuadrados

SEGMENTED-LEAST-SQUARES (n, p_1, ..., p_n , c)

for j = 1 to n In other words For all pairs i<j $O(n^3)$
  for i = 1 to j
    Compute $e(i, j)$ for the segment p_i, p_{i+1}, ..., p_j.

$M[0] \leftarrow 0$
for j = 1 to n $O(n^2)$
  $M[j] \leftarrow \min_{1 \le i \le j} \{ e(i,j) + C + M[i-1] \}$.
return $M[n]$

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

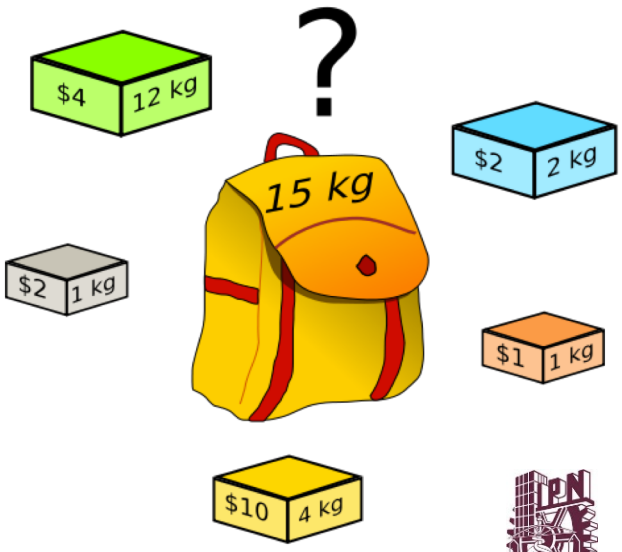
Introduction

1 pound = 0.45359 Kg
libra

- Knapsacks is one of the 21 NP-com problems of Richard Karp.

Problem

- During a robbery, a burglar finds much more loot than he had expected and has to decide what to take.
- His bag (or "knapsack") will hold a total weight of at most W pounds.
- There are n items to pick from, of weight $w_1, \ldots, w_n$ and dollar value $v_1, \ldots, v_n$.
- What's the most valuable combination of items he can fit into his bag?



6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

knapsacks = backpack = Mochila de espalda

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ and has value $v_i > 0$.
- Knapsack has capacity of W.
- Goal: fill knapsack so as to maximize total value.

Examples

$Items_i$ = {1, 2, 5} has value 35 & weight 10

{3, 4} has value 40 & weight 11

{3, 5} has value 46 (but exceeds weight limit!)

Knapsack instance
Weight limit W=11

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Formal definition

Bounded

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to} \sum_{i=1}^{n} w_i x_i \leq W \; and \; xi \in \{0, \ldots, ci\}$$

Unbounded $x_i \geq 0$

Hay varios objetos iguales
Y se pueden tomar varias veces

$$x_i \in \{0, 1\}$$

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Greedy by:

- Value. Repeatedly add item with maximum v_i.
- Weight. Repeatedly add item with minimum w_i.
- Ratio. Repeatedly add item with maximum ratio v_i / w_i.

Observation. None of greedy algorithms is optimal.

Tarea optativa:

1. Encontrar un ejemplo.
2. Realizar la demostración.

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Variations

There are two points of view:

- Repetition
  1. With: If there are unlimited quantities of each item available.
  2. Without: If there is one of each item.
- Bound.
  1. Unbounded knapsack problem (UKP)
  2. Bounded knapsack problem (BKP)

0-1 knapsack problem

Which restricts the number x_i of copies of each kind of item to zero or one.

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Variations

- Multi-objective knapsack problem

  This variation changes the goal of the individual filling the knapsack. Instead of one objective, such as maximizing the monetary profit, the objective could have several dimensions.

- Multi-dimensional knapsack problem

  In this variation, the weight of knapsack item *i* is given by a D-dimensional vector $w_i = (w_{i1}, \ldots, w_{iD})$ and the knapsack has a D-dimensional capacity vector $(W_1, \ldots, W_D)$. The target is to maximize the sum of the values of the items in the knapsack so that the sum of weights in each dimension *d* does not exceed W_d.

Bin packing problem, objects of different volumes must be packed into a finite number of bins or containers each of volume *V* in a way that minimizes the number of bins used.

- Multiple knapsack problem

  This variation is similar to the Bin Packing Problem. It differs from the Bin Packing Problem in that a subset of items can be selected, whereas, in the Bin Packing Problem, all items have to be packed to certain bins.

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Definition

OPT(i) = max profit subset of items 1, …, i.

Case 1. OPT does not select item i.

OPT selects best of { 1, 2, …, i – 1 }. → optimal substructure property (proof via exchange argument)

Case 2. OPT selects item i.

- Selecting item i does not immediately imply that we will have to reject other items.
- Without knowing what other items were selected before i, we don't even know if we have enough room for i.

Conclusion

Need more subproblems!

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

OPT(0,w)=0 no items
OPT(j,0)=0 no space

Definition

OPT(i, w) = max profit subset of items 1, …, i with weight limit w.

Case 1. OPT does not select item i.

OPT selects best of { 1, 2, …, i – 1 } using weight limit w.

Case 2. OPT selects item i.

- New weight limit = $w - w_i$.
- OPT selects best of { 1, 2, …, i – 1 } using this new weight limit.

optimal substructure property
(proof via exchange argument)

$$OPT(i,w) = \begin{cases} 0 & \text{if } i=0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-wi)\} & \text{otherwise} \end{cases}$$

Case 1 Case 2

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Algorithm bottom-up

KNAPSACK (n, W, w_1, …, w_n, v_1, …, v_n)

for w = 0 **to** W
  M [0, w] ← 0.

O(n W)

for i = 1 **to** n
  for w = 1 **to** W
    if (w_i > w) M [i, w] ← M [i – 1, w].
    else M [i, w] ← max { M [i – 1, w], v_i + M [i – 1, w – w_i] }.

return M[n, W].

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

i	v_i	w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Demo

$$OPT(i,w) = \begin{cases} 0 & \text{if } i=0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w), vi + OPT(i-1,w-wi)\} & \text{otherwise} \end{cases}$$

3 elementos
Items = {2,3}
W = 2 + 5 <=7
V = 6 + 18 = 24

Weight limit w

Subset of items 1,…,i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
{}	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1,2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1,2,3 }	0	1	6	7	7	18	19	24	25	25	25	25
{1,2,3,4}	0	1	6	7	7	18	22	24	28	29	29	40
{ 1,2,3,4,5 }	0	1	6	7	7	18	22	28	29	34	34	40

6.4 Subset sums and knapsacks

Suma de subconjuntos y mochila

Theorem

There exists an algorithm to solve the knapsack problem with n items and maximum weight W in Θ(n W) time and Θ(n W) space.

Proof

- Takes O(1) time per table entry.
- There are Θ(n W) table entries.
- After computing optimal values, can trace back to find solution: take item i in OPT(i, w) iff M [i, w] > M [i – 1, w].

Remarks Pseudo-polynomial

- Not polynomial in input size!
- Decision version of knapsack problem is NP-COMPLETE. [Unite 10]
- There exists a poly-time algorithm that produces a feasible solution that has value within 1% of optimum.

6.5 RNA secondary structure

Estructura secundaria del RNA

6.5 RNA secondary structure

Estructura secundaria del RNA

Introducción

Existen dos grandes tipos celulares:

- Las ***procariotas***: que comprenden las células de arqueas y bacterias.
  - Son pequeñas y menos complejas que las eucariotas.
  - Contienen ribosomas pero carecen de sistemas de endomembranas (esto es, orgánulos delimitados por membranas biológicas, como puede ser el núcleo celular).
- Las ***eucariotas***: divididas tradicionalmente en animales y vegetales, si bien se incluyen además hongos y protistas, que también tienen células con propiedades características.
  - Las células eucariotas son el exponente de la complejidad celular actual.
  - Presentan una estructura básica relativamente estable caracterizada por la presencia de distintos tipos de orgánulos intracitoplasmáticos especializados, entre los cuales destaca el núcleo, que alberga el material genético. Especialmente en los organismos pluricelulares, las células pueden alcanzar un alto grado de especialización.

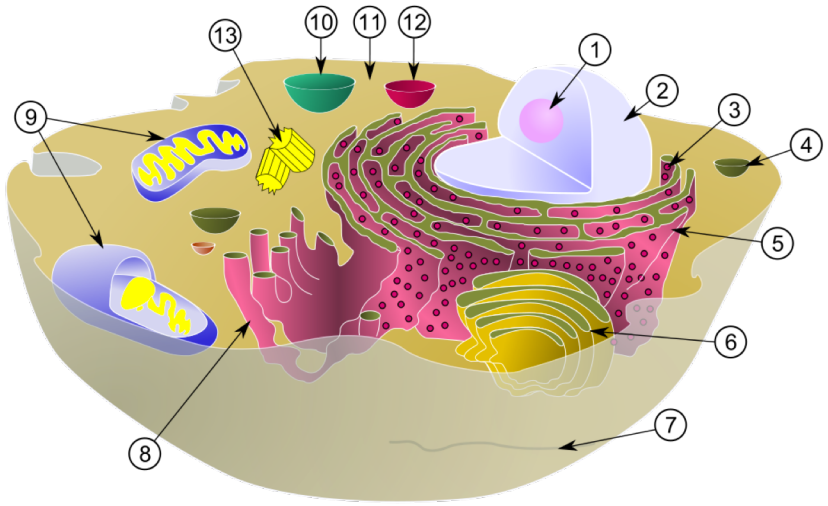
6.5 RNA secondary structure

Estructura secundaria del RNA

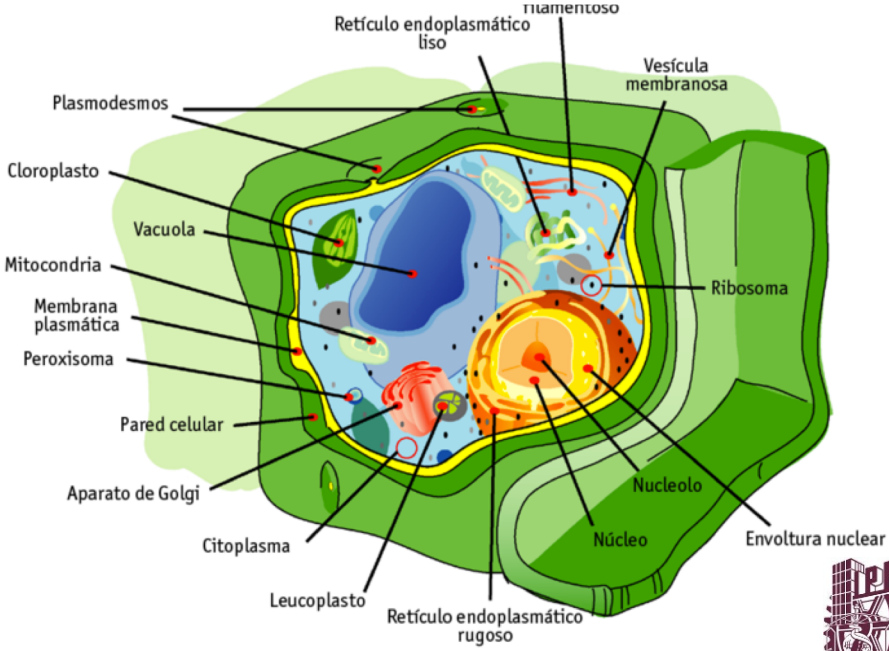
Introducción

Diagrama de una célula:

Animal



Vegetal



6.5 RNA secondary structure

Estructura secundaria del RNA

Introducción

Las células eucariotas poseen su material genético en:

- Un solo núcleo celular, delimitado por una envoltura consistente en dos bicapas lipídicas atravesadas por numerosos poros nucleares, y
- Continuidad con el retículo endoplasmático.

En su interior, se encuentra el material genético:

- El ADN, observable, en las células en interfase, como cromatina de distribución heterogénea.
- A esta cromatina se encuentran asociadas multitud de proteínas, entre las cuales destacan las histonas, así como ARN, otro ácido nucleico.

6.5 RNA secondary structure

Estructura secundaria del RNA

Introducción

Las proteínas o prótidos son biomoléculas formadas por cadenas lineales de aminoácidos.

Por sus propiedades físico-químicas, las proteínas se pueden clasificar en:

- ***Simples*** (holoproteidos): formadas solo por aminoácidos o sus derivados.
- ***Conjugadas*** (heteroproteidos): formadas por aminoácidos acompañados de sustancias diversas.
- ***Derivadas***: sustancias formadas por desnaturalización y desdoblamiento de las anteriores.

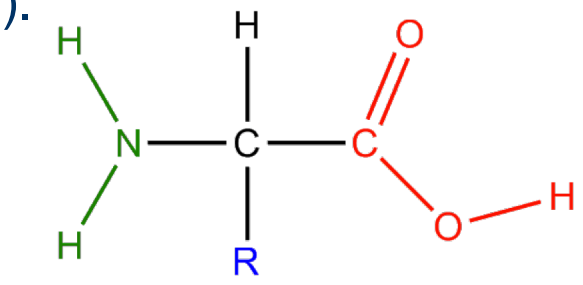
6.5 RNA secondary structure

Estructura secundaria del RNA

Introducción

Un aminoácido es una molécula orgánica:

- Con un grupo amino (-NH2) y un grupo carboxilo (-COOH).
- Los aminoácidos más frecuentes y de mayor interés son aquellos que forman parte de las proteínas.
- Todos los aminoácidos componentes de las proteínas son L-alfa-aminoácidos. Esto significa que el grupo amino está unido al carbono contiguo al grupo carboxilo (carbono alfa).
- Dos aminoácidos se combinan en una reacción de descomposicion formando dipéptidos, tripéptidos, y en general polipéptidos.



6.5 RNA secondary structure

Estructura secundaria del RNA

ADN

- El ácido desoxirribonucleico, es un ácido nucleico que contiene las instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos conocidos y algunos virus, y es responsable de su transmisión hereditaria.
- La función principal de la molécula de ADN es el almacenamiento a largo plazo de información.

ARN o RNA

- El ácido ribonucleico es un ácido nucleico formado por una cadena de ribonucleótidos.
- Está presente tanto en las células procariotas como en las eucariotas, y es el único material genético de ciertos virus (virus ARN).
- El ARN celular es lineal y monocateriana (de una sola cadena), pero en el genoma de algunos virus es de doble hebra.

6.5 RNA secondary structure

Estructura secundaria del RNA

RNA

- En los organismos celulares desempeña diversas funciones.
  - Es la molécula que dirige las etapas intermedias de la síntesis proteica.
  - El ADN no puede actuar solo, y se vale del ARN para transferir esta información vital durante la síntesis de proteínas (producción de las proteínas que necesita la célula para sus actividades y su desarrollo).
  - Varios tipos de ARN regulan la expresión génica, mientras que otros tienen actividad catalítica.
- En conclusión es mucho más versátil que el ADN.

6.5 RNA secondary structure

Estructura secundaria del RNA

Estructura del RNA

- Está formado por una cadena de monómeros repetitivos llamados **nucleótidos**.
- Los nucleótidos se unen uno tras otro mediante enlaces fosfodiéster cargados negativamente.
- Cada nucleótido está formado por tres componentes:
  - Un monosacárido de cinco carbonos (pentosa) llamada ribosa.
  - Un grupo fosfato
  - Una base nitrogenada, que puede ser:

• Adenina (A)	• Guanina (G)
• Citosina (C)	• Uracilo (U)

6.5 RNA secondary structure

Estructura secundaria del RNA

Estructura primaria

- Se refiere a la secuencia lineal de nucleótidos en la molécula de ARN.
- Los siguientes niveles estructurales (estructura secundaria, terciaria) son consecuencia de la estructura primaria.

Estructura secundaria

- El ARN se pliega como resultado de la presencia de regiones cortas con apareamiento intramolecular de bases, es decir, pares de bases formados por secuencias complementarias más o menos distantes dentro de la misma hebra.
- La estructura secundaria se refiere, entonces, a las relaciones de apareamiento de bases: “El término ‘estructura secundaria’ denota cualquier patrón plano de contactos por apareamiento de bases.
- Es un concepto topológico y no debe ser confundido con algún tipo de estructura bidimensional.

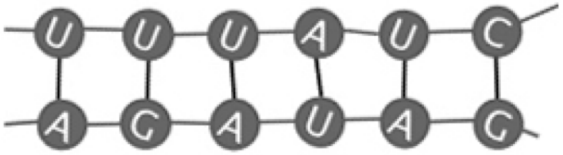
Estructura terciaria

- Es el resultado de las interacciones en el espacio entre los átomos que conforman la molécula.
- Algunas interacciones de este tipo incluyen el apilamiento de bases y los apareamientos de bases distintos a los propuestos por Watson y Crick, como el apareamiento Hoogsteen, los apareamientos triples y los zippers de ribosa.

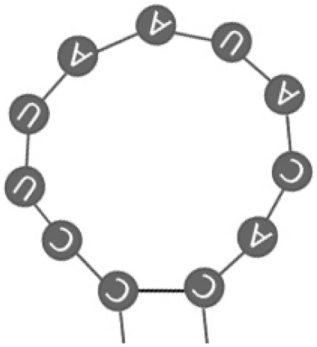
6.5 RNA secondary structure

Estructura secundaria del RNA

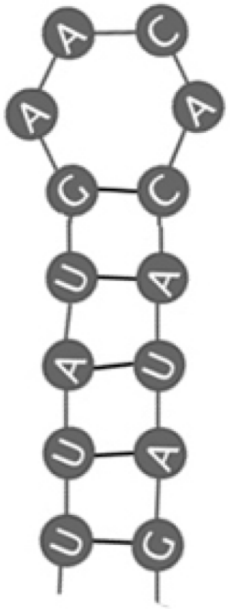
Ejemplos de Estructura secundaria



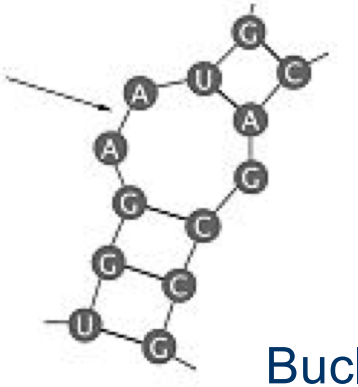
Hélice



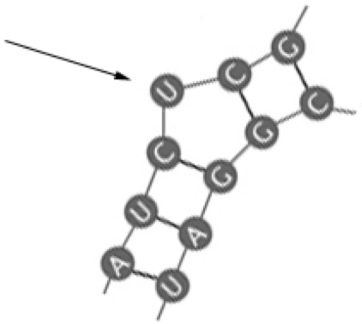
Bucle



Bucle en horquilla



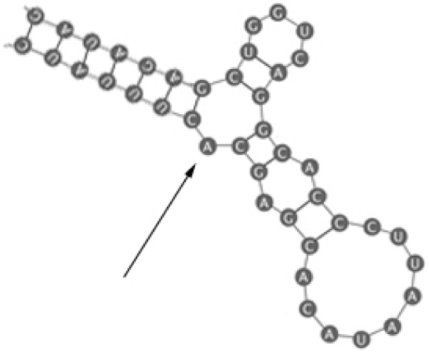
Bucle interno



Protuberancia

Bucle multiple

Pseudo nudo



6.5 RNA secondary structure

Estructura secundaria del RNA

RNA

String B = $b_1 b_2 \ldots b_n$ over alphabet { A, C, G, U }

Secondary structure

RNA is single-stranded so it tends to loop back and form base pairs with itself. This structure is essential for understanding behavior of molecule.

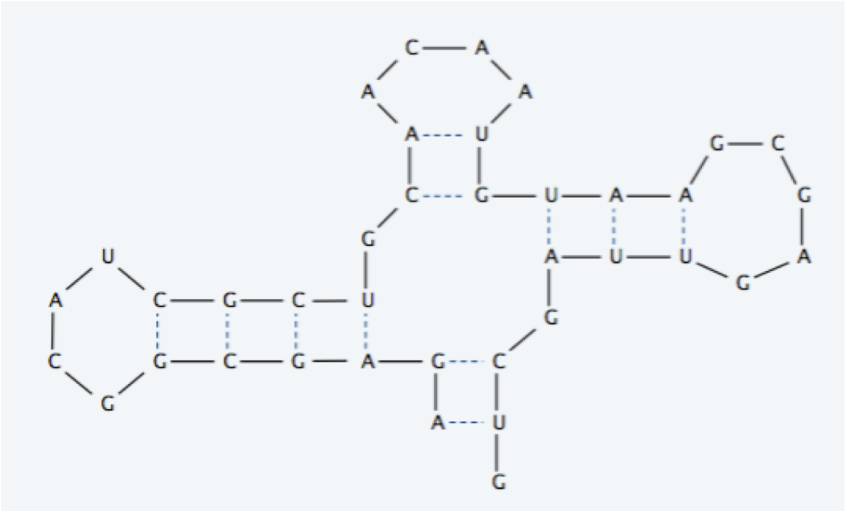
Example

RNA secondary structure for:

GUCGAUUGAGCGAAU

GUAACAACGUGGCUAC

GGCGAGA



6.5 RNA secondary structure

Estructura secundaria del RNA

Secondary structure. A set of pairs S = { (b_i, b_j) } that satisfy:

- [Watson-Crick] S is a matching and each pair in S is a Watson-Crick complement: A–U, U–A, C–G, or G–C.
- [No sharp turns] The ends of each pair are separated by at least 4 intervening bases. If (bi, bj) ∈ S, then i < j – 4.
- [Non-crossing] If (b_i, b_j) and (b_k, b_l) are two pairs in S, then we cannot have i < k < j <l.

Free energy

Usual hypothesis is that an RNA molecule will form the secondary structure with the minimum total free energy.

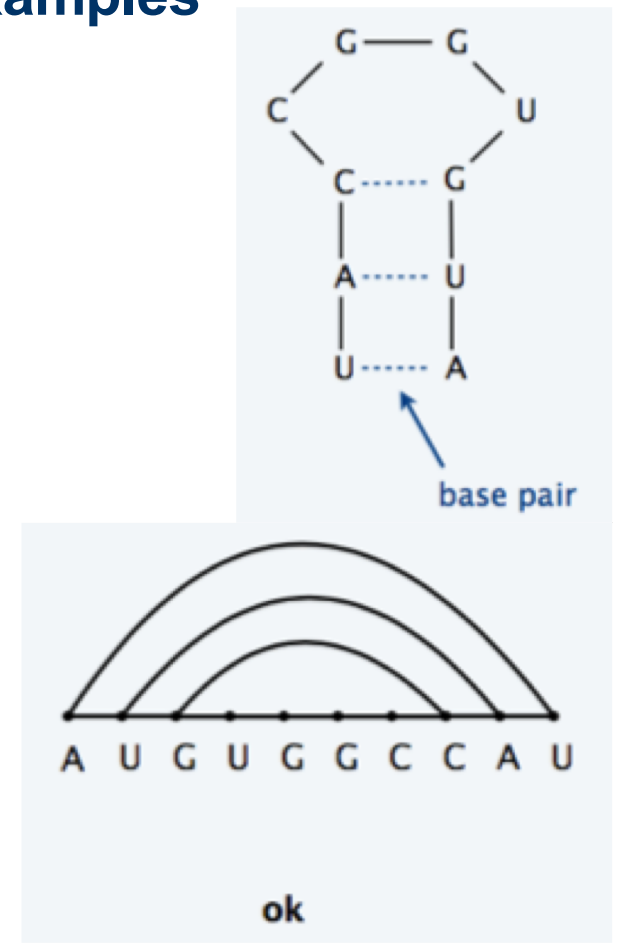
Goal

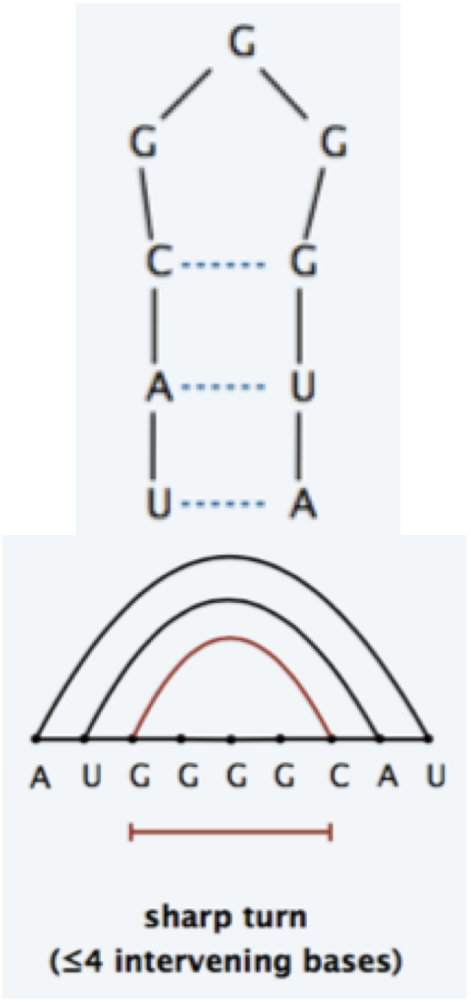
Given an RNA molecule $B = b_1 b_2 ... b_n$, find a secondary structure S that maximizes the number of base pairs.

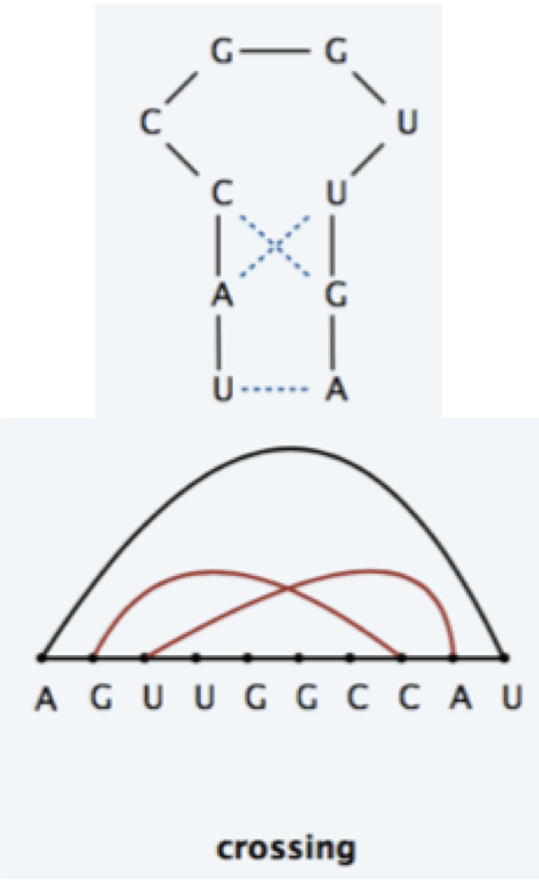
6.5 RNA secondary structure

Estructura secundaria del RNA

Examples







6.5 RNA secondary structure

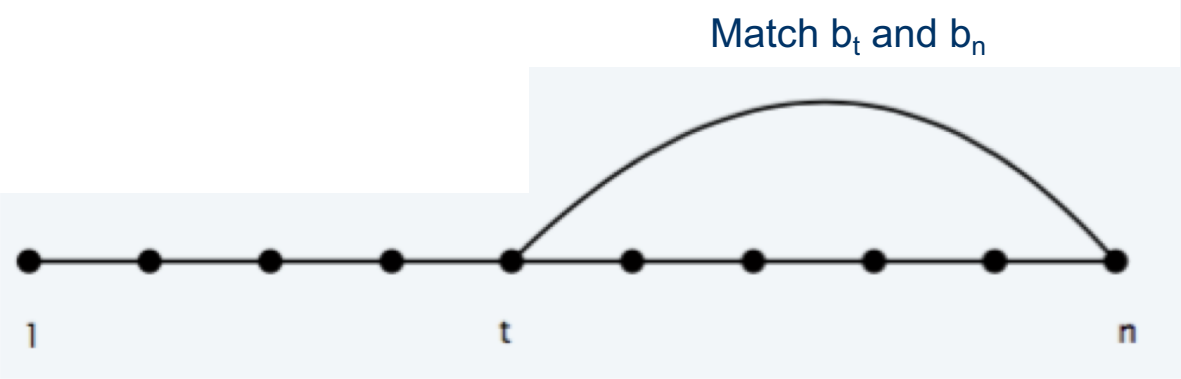
Estructura secundaria del RNA

First attempt

OPT(j) = maximum number of base pairs in a secondary structure of the substring $b_1\ b_2\ \ldots\ b_j$.

Choice

Match b_t and b_n.



Difficulty. Results in two subproblems but one of wrong form.

- Find secondary structure in $b_1\ b_2\ \ldots\ b_{t-1}$. OPT(t–1)
- Find secondary structure in $b_{t+1}\ b_{t+2}\ \ldots\ b_{n-1}$.

6.5 RNA secondary structure

Estructura secundaria del RNA

Notation

OPT(i, j) = maximum number of base pairs in a secondary structure of the substring $b_i\ b_{i+1}\ \ldots\ b_j$.

Case 1. If $i \geq j - 4$.

- OPT(i, j) = 0 by no-sharp turns condition.

Case 2. Base b_j is not involved in a pair.

- OPT(i, j) = OPT(i, j – 1).

Case 3. Base b_j pairs with b_t for some $i \leq t < j - 4$.

- Noncrossing constraint decouples resulting subproblems.
- OPT(i, j) = 1 + max t { OPT(i, t – 1) + OPT(t + 1, j – 1) }.

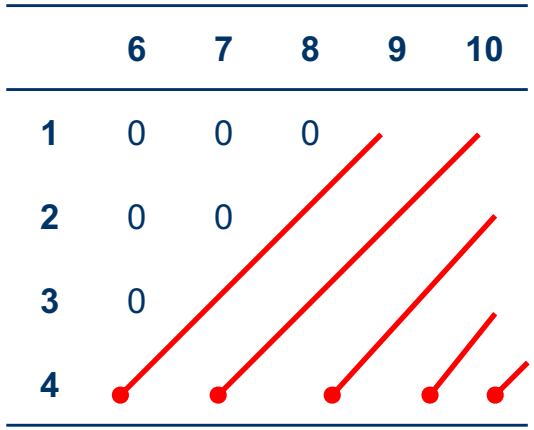
6.5 RNA secondary structure

Estructura secundaria del RNA

Q. In which order to solve the subproblems?

A. Do shortest intervals first.

```
RNA (n, b1, ..., bn )
   for k = 5 TO n – 1
      for i = 1 TO n – k
         j← i+k.
         Compute M[i, j] using formula.
return M[1,n].
```



Theorem

The dynamic programming algorithm solves the RNA secondary substructure problem in $O(n^3)$ time and $O(n^2)$ space.

6.6 Shortest path in a graph

El camino más corto en un grafos

6.6 Shortest paths in a graph

El camino más corto en un grafo

6.7 Extra problems

Más problemas

6.7.1

Longest increasing subsequence

La subsecuencia incremental más larga

Longest increasing subsequence

La subsecuencia incremental más larga

Definition

It is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.

Examples

- { 15, 27, 14, 38, 26, 55, 46, 65, 85 }
  - Subsequence={ 15, 27, 38, 55, 65, 85 }
  - Long=6.
- { 10, 9, 2, 5, 3, 7, 101, 18 }
  - Subsequence={ 2, 3, 7, 101 }
  - Long=4.

Longest increasing subsequence

La subsecuencia incremental más larga

Examples

- In the first 16 terms of the binary Van der Corput sequence
  - 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
- A longest increasing subsequence is:
  - 0, 2, 6, 9, 11, 15. Length six.
- The longest increasing subsequence in this example is not unique, for instance:
  - 0, 4, 6, 9, 11, 15
  - 0, 4, 6, 9, 13, 15
- The input sequence has no seven-member increasing subsequences.

6.7 Extra problems

Más problemas

6.7.2

Longest common subsequence

La subsecuencia incremental más larga

Longest common subsequence

La subsecuencia incremental más larga

- The longest increasing subsequence problem is closely related to the longest common subsequence problem.
- The longest increasing subsequence of a sequence S is the longest common subsequence of S and T, where T is the result of sorting S.
- It has a quadratic time dynamic programming solution.
- Examples:
  - Input sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.
  - Input sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

6.7 Extra problems

Más problemas

6.7.3
Matrix-Chain Multiplication

Multiplicación de matrices

Matrix-Chain Multiplication

Multiplicación de matrices

- Given a sequence of matrices, the goal is to find the most efficient way to multiply these matrices.
- The problem is not actually to perform the multiplications, but merely to decide the sequence of the matrix multiplications involved.
- We have many options because matrix multiplication is associative.
  - In other words, no matter how we parenthesize the product, the result obtained will remain the same.
  - $((AB)C)D = ((A(BC))D) = (AB)(CD) = A((BC)D) = A(B(CD))$.
- However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the *efficiency*.

Matrix-Chain Multiplication

Multiplicación de matrices

Example

- Suppose
  - A is a 10 × 30 matrix.
  - B is a 30 × 5 matrix.
  - C is a 5 × 60 matrix.
- Then
  - $(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1{,}500 + 3{,}000$
    $= 4{,}500$ operations
  - $A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9{,}000 + 1{,}8000$
    $= 27{,}000$ operations.

Matrix-Chain Multiplication

Multiplicación de matrices

Algorithm

```
// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
MatrixChainOrder(int p[]) { // length[p] = n + 1
    n = p.length - 1;
    for (i = 1; i <= n; i++)
      m[i,i] = 0;

    for (L=2; L<=n; L++) { // L is chain length
      for (i=1; i<=n-L+1; i++) {
        j = i+L-1;
        m[i,j] = MAXINT;
        for (k=i; k<=j-1; k++) {
          q = m[i,k] + m[k+1,j] + p[i-1]*p[k]*p[j];
          if (q < m[i,j]) {
              m[i,j] = q;
              s[i,j]=k;
} } } } }.
```

m[i,j] = Minimum number of scalar multiplications (i.e., cost) needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]
cost is zero when multiplying one matrix

// q = cost/scalar multiplications

s[i,j] = Second auxiliary table that stores k
k = Index that achieved optimal cost

Matrix-Chain Multiplication

Multiplicación de matrices

Complexity

- It can be shown that DP (with some simple trick) brings the runtime down to $O(n^3)$ from $O(2^n)$, which is more than efficient enough for real applications.
- There are algorithms that are more efficient than the $O(n^3)$ dynamic programming algorithm, though they are more complex.
  - An algorithm published in 1981 by Hu and Shing achieves $O(n \log n)$ complexity.

6.7 Extra problems

Más problemas

6.7.4
Word segmentation problem

Segmentación de palabras (chinese)

Word segmentation problem

Segmentación de palabras (chinese)

Definition

Given a string x of letters $x_1x_2\ldots x_n$, give an efficient algorithm to split x into words (substrings) such that the sum of the quality of these words is maximized.

Example

"mogenzeslapen"

quality(mo, gen, ze, sla, pen) = 7

quality(mogen, ze, slapen) = 10

Word	quality
mo	0
mogen	4
enz	1
gen	2
sla	2
pen	2
slapen	5
ze	1
en	1

punish uncommon words

6.7 Extra problems

Más problemas

6.7.5

Sequence alignement

Alineación de secuencias

Sequence alignment

Alineación de secuencias

.

Several topics

Varios temas

Notas históricas

Problemas abiertos

Tareas

URLs

A recordar

Recommended reading and Web sites

Lecturas recomendadas

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

Wikipedia

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

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

Historical notes

Notas históricas

- .

Open problemas

Problemas sin resolver

Problems

Homeworks

Tareas

- .

To remember

A recordar

Lo más importante es: