- Programació dinàmica
  - Programació dinàmica de dalt a baix
  - Programació dinàmica de baix a dalt
- Problemes

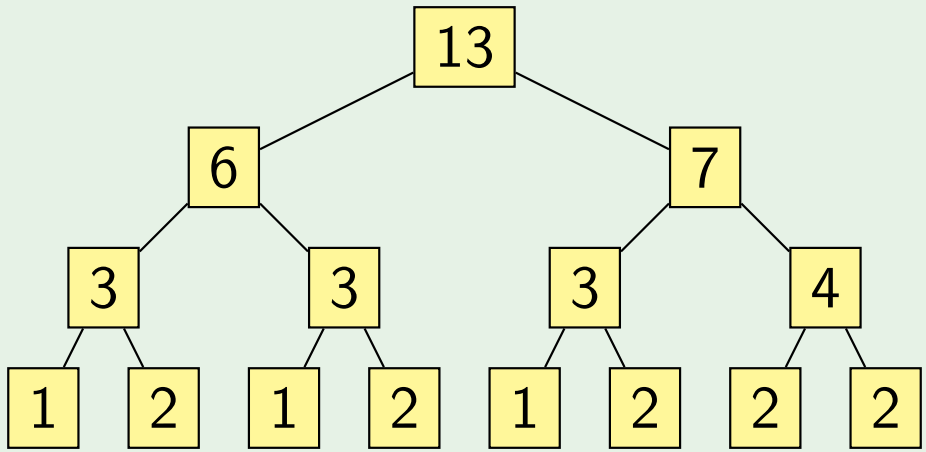
Basat en:

- G. Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, 2002.

- Recordem que la tècnica de dividir i vèncer es pot aplicar a aquells problemes que mostren el principi d'independència, segons el qual una instància del problema es pot dividir en una sèrie d'instàncies més petites i independents entre sí.
- En aplicar-lo a la cerca de solucions de cost mínim, el principi d'independència se sol anomenar principi d'optimalitat, segons el qual una solució de cost mínim d'una instància del problema es pot dividir en una sèrie de solucions de cost mínim d'instàncies més petites i independents entre sí. Aquestes instàncies més petites i independents se solen solapar, però, per la qual cosa un algorisme de dividir i vèncer farà un esforç computacional innecessari per recalcular solucions de cost mínim d'instàncies del problema que se solapen.
- La tècnica de programació dinàmica comporta una millora sustancial de la tècnica de dividir i vèncer per a aquells problemes que mostren el principi d'optimalitat, en emmagatzemar solucions a subproblemes. Amb això, s'evita el recàlcul de solucions a subproblemes que se solapen durant la cerca d'una solució òptima al problema.

Exemple (Màquina expenedora, cont.)

- **procedure** monedes (y: nat)
  if $y \in \{1, 2, 5, 10, 20, 50\}$ **then**
  print y
  else
  monedes($\lfloor y/2 \rfloor$)
  monedes($\lceil y/2 \rceil$)
- Aquest algorisme de dividir i vèncer, però, no és correcte. Per a la instància $y = 13$, per exemple, dóna 8 monedes ($13 = 5 \cdot 2 + 3 \cdot 1$), però hi ha una solució amb 3 monedes ($13 = 1 \cdot 10 + 1 \cdot 2 + 1 \cdot 1$).



Exemple (Màquina expenedora, cont.)

- Siguin $D_1 > D_2 > \cdots > D_n = 1$ les denominacions disponibles.
- El mínim nombre $C_{i,j}$ de monedes possible per donar canvi de j cèntims en monedes de denominacions $D_i, \ldots, D_n$ cèntims, ve donat per

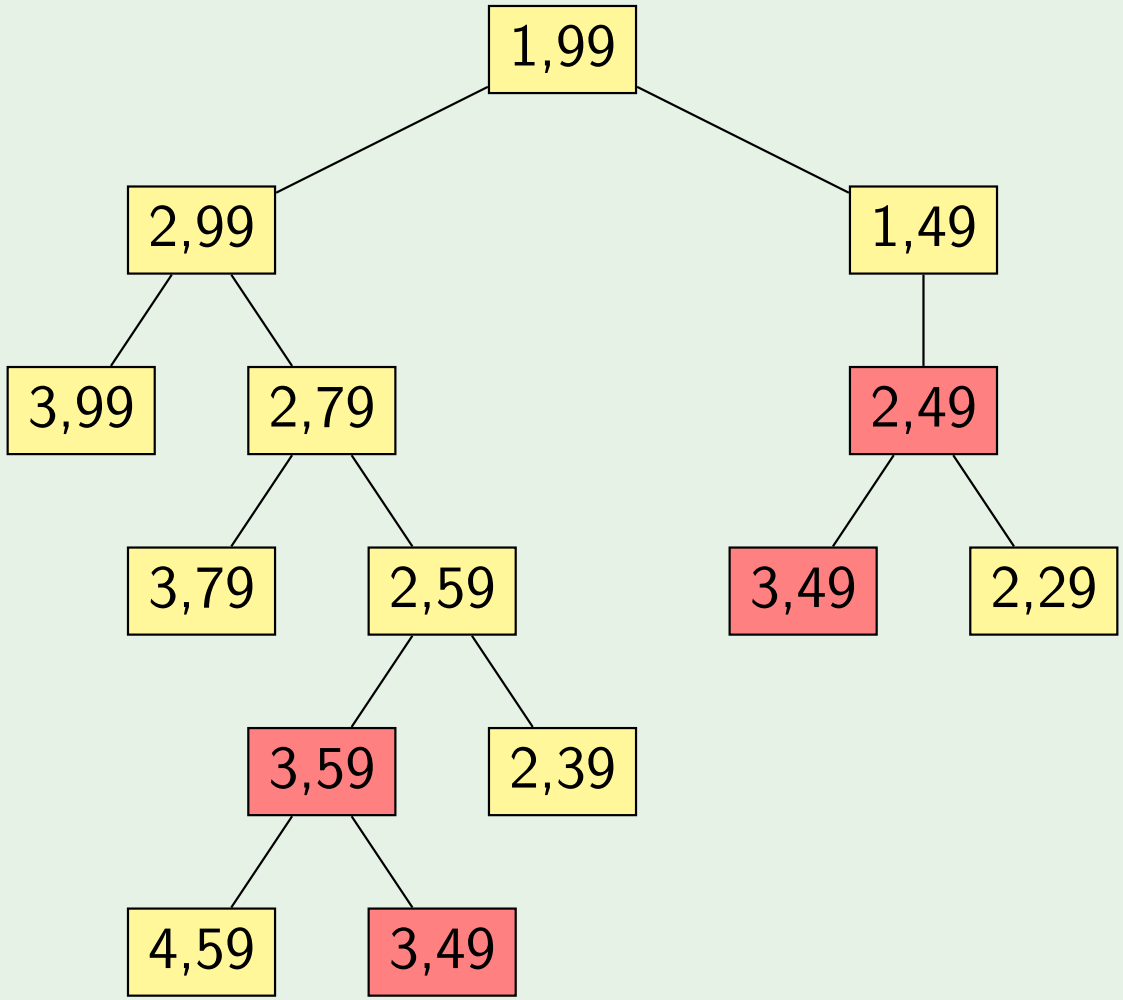
$$C_{i,j} = \begin{cases} C_{i+1,j} & \text{si } D_i > j \\ \min(C_{i+1,j}, 1 + C_{i,j-D_i}) & \text{si } D_i \leqslant j \end{cases}$$

  on $1 \leqslant i \leqslant n$.

- **procedure** monedes (D: **array**[1..n] **de** nat; i, j: nat)
  if $D[i] > j$ **then**
  return monedes$(D, i+1, j)$
  else
  return min(monedes$(D, i+1, j)$, $1 +$ monedes$(D, i, j - D[i])$)

Exemple (Màquina expenedora, cont.)

- Les instàncies $C_{3,59}$ i $C_{2,49}$, per exemple, se solapen.



- El nom mateix de programació dinàmica no fa referència a una metodologia de programació sinó a un mètode tabular, amb el qual les solucions òptimes dels subproblemes es calculen només un cop i s'emmagatzemen en un diccionari per poder esser consultades més endavant, en trobar el subproblema durant la cerca d'una solució òptima del problema original.
- Essencialment, hi ha dues maneres en què es pot implementar aquest mètode tabular.
- La programació dinàmica de dalt a baix, també coneguda com a *memorització*, consisteix a estendre un algorisme recursiu de dividir i vèncer per a un problema d'optimització, comprovant si un subproblema ha estat resolt abans (i la seva solució està disponible en un diccionari) i resolent-lo recursivament en cas contrari.
- La implementació eficient de la programació dinàmica de dalt a baix requereix consultes de diccionaris en temps constant, per la qual cosa se solen usar taules i tècniques de hashing perfecte.

Exemple (Màquina expenedora, cont.)

- La conversió de l'algorisme de dividir i vèncer en un algorisme de programació dinàmica de dalt a baix per calcular $C_{i,j}$, requereix un diccionari C de naturals indexat per parells de naturals.
- **procedure** monedes (C: dicc; D: **array**$[1..n]$ **de** nat; i, j: nat)
  if not pertany$(C, \langle i+i, j\rangle)$ **then**
  inserir$(C, \langle i+1, j\rangle,$ monedes$(C, D, i+1, j))$
  if $D[i] > j$ **then**
  return consultar$(C, \langle i+1, j\rangle)$
  else
  if not pertany$(C, \langle i, j-D[i]\rangle)$ **then**
  inserir$(C, \langle i, j-D[i]\rangle,$ monedes$(C, D, i, j-D[i]))$
  return
  $\min($consultar$(C, \langle i+1, j\rangle), 1 +$ consultar$(C, \langle i, j-D[i]\rangle))$
- El cost d'aquest algorisme és $\Theta(ij)$ si es representa el diccionari amb una taula.

- En la programació dinàmica de baix a dalt, el càlcul s'efectua de baix a dalt i les solucions òptimes dels subproblemes són disponibles per a la resolució d'un problema de talla més gran.
- Un algorisme de programació dinàmica de baix a dalt pot resultar més eficient que un algorisme de programació dinàmica de dalt a baix, tot i que per un factor constant, si cal resoldre tots els subproblemes almenys un cop, degut al cost addicional de la recursió i de manteniment del diccionari.
- Un algorisme de programació dinàmica de dalt a baix pot resultar més eficient que un algorisme de programació dinàmica de baix a dalt, però, quan no cal resoldre tots els subproblemes, perquè la solució de dalt a baix té l'avantatge de resoldre només aquells subproblemes que són estrictament necessaris per trobar la solució òptima del problema.
- Hi ha problemes, però, en els quals el patró regular d'accés al diccionari de l'algorisme de programació dinàmica de baix a dalt es pot aprofitar per reduir encara més el seu cost temporal o espacial.

Exemple (Màquina expenedora, cont.)

- **procedure** monedes (D: **array**$[1..n]$ **de** nat; y: nat)
  for $j := 0$ **to** y **do**
  $C[n, j] := j$
  for $i := n - 1$ **downto** 1 **do**
  for $j := 0$ **to** y **do**
  if $D[i] > j$ **or** $C[i + 1, j] < 1 + C[i, j - D[i]]$ **then**
  $C[i, j] := C[i + 1, j]$
  else
  $C[i, j] := 1 + C[i, j - D[i]]$
- Amb aquest algorisme es fan $y + 1$ iteracions seguides de $(n - 1)(y + 1)$ iteracions. El seu cost és doncs $\Theta(ny)$.

Exemple (Màquina expenedora, cont.)

- $D_1 = 50, D_2 = 20, D_3 = 10, D_4 = 5, D_5 = 2, D_6 = 1$.
- Mínim nombre $C_{i,j}$ de monedes possible per donar canvi de j cèntims en monedes de denominacions $D_i, \ldots, D_n$.

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

- Mínim nombre $C_{i,0}, \ldots, C_{i,j}$ de monedes possible per donar canvi de $y = 0, \ldots, j$ cèntims en monedes de denominacions $D_i, \ldots, D_n$.

Exemple (Màquina expenedora, cont.)

- L'algorisme anterior es pot estendre fàcilment per trobar també el multiconjunt de denominacions corresponent a una solució òptima.
- Només cal mantenir una taula U de denominacions usades.
- **procedure** monedes (D: **array**$[1..n]$ **de** nat; y: nat)

```
for j := 0 to y do
    C[n, j] := j
    U[n, j] := true
for i := n − 1 downto 1 do
    for j := 0 to y do
        if D[i] > j or C[i + 1, j] < 1 + C[i, j − D[i]] then
            C[i, j] := C[i + 1, j]
            U[i, j] := false
        else
            C[i, j] := 1 + C[i, j − D[i]]
            U[i, j] := true
```

Exemple (Màquina expenedora, cont.)

- $D_1 = 50, D_2 = 20, D_3 = 10, D_4 = 5, D_5 = 2, D_6 = 1$.
- $C_{i,j}$ és el mínim nombre de monedes possible per donar canvi de j cèntims en monedes de denominacions $D_i, \ldots, D_n$.
- $U_{i,j}$ indica si s'han usat monedes de denominació D_i per donar canvi de j cèntims en monedes de denominacions $D_i, \ldots, D_n$.

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

Exemple (Màquina expenedora, cont.)

- El vector U de denominacions usades permet reconstruir fàcilment les solucions òptimes.
- **procedure** òptim (i, j: nat; D, U: **array**[1..n] **de** nat)
  if $j \neq 0$ **then**
  if $U[i, j]$ **then**
  print "moneda" $D[i]$
  òptim$(i, j - D[i], D, U)$
  else
  òptim$(i + 1, j, D, U)$
- Per exemple,
  - òptim$(1, 13, D, U)$
  - òptim$(2, 13, D, U)$
  - òptim$(3, 13, D, U)$
  - moneda 10
  - òptim$(3, 3, D, U)$
  - òptim$(4, 3, D, U)$
  - òptim$(5, 3, D, U)$
  - moneda 2
  - òptim$(5, 1, D, U)$
  - òptim$(6, 1, D, U)$
  - moneda 1
  - òptim$(6, 0, D, U)$

Els problemes marcats amb una estrelleta ($\star$) són més difícils. Si no es diu el contrari i us cal, considereu que tots els logaritmes són en base 2.

- 8.1 (en teoria)
- 8.2 (en teoria)
- 8.5
- 8.8
- 8.9
- 8.10
- 8.12
- 8.15 (en teoria)

Problema (8.1)

Els nombres de Fibonacci es defineixen inductivament com segueix:

$$F(n) = \begin{cases} 0, & \text{si } n = 0 \\ 1, & \text{si } n = 1 \\ F(n-1) + F(n-2), & \text{si } n \geqslant 2. \end{cases}$$

Implementeu i analitzeu

- *un algorisme recursiu per calcular* $F(n)$,
- *un algorisme que usi programació dinàmica per calcular* $F(n)$,
- *un algorisme recursiu per calcular alhora* $F(n)$ *i* $F(n+1)$.

Problema (8.2)

Dissenyeu algorismes diferents per calcular $\binom{n}{k}$, *és a dir, el nombre de maneres d'escollir* k *elements d'entre* n. *Lògicament,* $\binom{n}{k} = 0$ *si* $k < 0$ *o si* $k > n$. *Per definició,* $\binom{0}{0} = 1$.

- *El primer algorisme ha de ser recursiu i usar la igualtat*

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

- *El segon algorisme ha de fer servir la mateixa igualtat, però usant programació dinàmica.*
- *El tercer algorisme ha de ser recursiu i usar la igualtat*

$$\binom{n}{k} = \binom{n}{k-1} \cdot \frac{n-k+1}{k}$$

Quins dels tres algorismes són eficients?

Problema (8.5)

Considereu les seqüències generades per la gramàtica següent:

$$\begin{array}{rcl} \textit{seqüència} & \longrightarrow & \textit{nombre} \mid \textit{nombre}\ \ \textit{operació}\ \ \textit{seqüència} \\ \textit{operació} & \longrightarrow & + \mid * \end{array}$$

on "nombre" indica qualsevol nombre real positiu. Dissenyeu un algorisme que calculi el màxim resultat possible d'aplicar les operacions d'una seqüència donada, suposant que es poden afegir tants parèntesi com es vulgui.

Per exemple, si la seqüència fos $2 * 2.5 + 1$, *podríem avaluar-la com* $(2 * 2.5) + 1 = 6$, *o bé com* $2 * (2.5 + 1) = 7$. *Per tant l'algorisme hauria de retornar 7.*

Funciona el vostre algorisme quan els nombres són negatius?

Problema (8.8)

Considereu un sistema de n segells, amb valors naturals $0 < v_1 < v_2 < \cdots < v_n$. *Dissenyeu i analitzeu un algorisme que calculi de quantes maneres diferents es pot aconseguir un cert valor positiu* V *usant tants segells com calgui de cada tipus.*

Per exemple, si els valors són 2, 5 i 7, llavors per a $V = 12$ *la resposta és 3, per a* $V = 7$ *la resposta és 2, per a* $V = 4$ *la resposta és 1, i per a* $V = 3$ *la resposta és 0.*

Problema (8.9)

Torneu a resoldre el problema anterior, però suposant ara que a una carta només hi caben com a molt K segells. (Així doncs, al problema anterior teníem $K = \infty$.)

Per exemple, per als valors 2, 5 i 7, amb $V = 12$ i $K = 3$, la resposta ha de ser 2 (corresponent a les combinacions $7 + 5$ i $5 + 5 + 2$).

Problema (8.10)

Torneu a resoldre el problema anterior, però suposant ara que a una carta s'han de posar exactament 2 segells.

Per exemple, per als valors 2, 5 i 7, amb $V = 12$, *la resposta ha de ser 1 (corresponent a la combinació* $7 + 5$*).*

Problema (8.12)

Suposeu que sou els responsables d'una empresa dedicada a tallar barres d'acer. Donada una barra de M metres, s'obtenen $n+1$ barres més curtes tallant la barra original en n punts, $0 < p_1 < p_2 < \cdots < p_n < M$. Cada tall costa un euro per cada metre que tingui la barra abans de tallar-la. Dissenyeu i analitzeu un algorisme que calculi el cost mínim de tallar una barra, donats M, n i $p_1, p_2, \ldots, p_n$.

Per exemple, si $M = 9$, $n = 2$, $p_1 = 2$ i $p_2 = 6$, tallant primer a p_1 el cost és $9 + 7 = 16$, mentre que tallant primer a p_2 el cost només és $9 + 6 = 15$.

Problema (8.15)

Dissenyeu un algorisme que, donada un col·leció C de n nombres enters positius (amb possibles repeticions) i un nombre enter positiu S, determini si hi ha un subconjunt de C que sumi exactament S.

Solucioneu aquest problema amb programació dinàmica, tot suposant que cada element de la col·lecció està fitat per una certa constant coneguda (com ara 1000). Quin és el cost del vostre algorisme? I si no es coneix cap fita superior per al pes dels objectes?