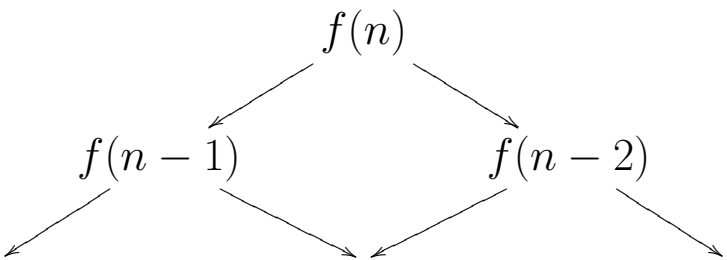
Programació Dinàmica

- Plantejament recursiu de la resolució de problemes.
- Si un problema es pot descomposar en subproblemes sempre es pot considerar la possibilitat de combinar les subsolucions corresponents.
- Sota certes condicions, la solució recursiva basada en la divisió en subproblemes i combinació de resultats es pot convertir en un procediment iteratiu (significativament) més eficient

Programació Dinàmica

Problema: calcular el n-èssim nombre de Fibonacci, $f(n)$.
Subproblemes: calcular $f(n-1)$ i $f(n-2)$.
Combinació: calcular la suma.



L'arbre de recursió té una profunditat màxima i mínima de n i $\frac{n}{2}$, respectivament. El cost estarà doncs entre 2^n i $(\sqrt{2})^n$.

Programació Dinàmica

Ara be, quans subproblemes **diferents** realment hi ha?

Efectivament, només n. I com que cada subproblema es pot resoldre mijantçant una combinació de les solucions de subproblemes inferiors,

es pot "recórrer" d'alguna manera convenient l'espai de solucions?

Equivalentment: es pot calcular $f(n)$ resolent només n subproblemes?

$$f(0) \longrightarrow f(1) \longrightarrow f(2) \longrightarrow \cdots \longrightarrow f(n-2) \longrightarrow f(n-1) \longrightarrow f(n)$$

Programació Dinàmica

En general, donada la solució recursiva a un problema:

Esquema RR (d)

si casbase(d) aleshores soluciódirecta(d)

si no

$(d_1, d_2, \ldots d_k) \leftarrow$descomposar$(d)$

combinar$(RR(d_1), \ldots, RR(d_k))$

Programació Dinàmica

Si es poden emmagatzemar els resultats parcials dels subproblemes, $r_i = RR(d_i)$, en una certa estructrura de dades, indexada pels valors d_i. És a dir, $A[d_i] = r_i$

I es disposa d'una enumeració dels valors d_i que garantesca

$d_i < d$, $1 \leq i \leq k$ si $(d_1, d_2, \ldots d_k) =$descomposar(d)

Aleshores,

Programació Dinàmica

Esquema PD (d)

$d' \leftarrow$ mínim

mentre $d' < d$ fer

si casbase(d') aleshores soluciódirecta(d')

si no

$(d_1, d_2, \ldots d_k) \leftarrow$descomposar$(d')$

combinar$(A[d_1], \ldots, A[d_k])$

$d' \leftarrow$ següent(d')

El cost ara ja no depén de l'arbre de recursió sino del nombre de subproblemes que cal resoldre fins arribar al problema d.

Programació Dinàmica

Normalment, la solució iterativa encara pot ser més eficient si els casos base es poden inicialitzar fora del bucle.

Esquema PD (d)

Inicialitzar casos base
$d' \leftarrow$ menor valor no trivial
mentre $d' < d$ fer
$(d_1, d_2, \ldots d_k) \leftarrow$descomposar$(d')$
combinar$(A[d_1], \ldots, A[d_k])$
$d' \leftarrow$ següent(d')

Programació Dinàmica

Els d_i són normalment enters o tuples d'enters (mínim nombre de paràmetres que caracteritza cada subproblema).

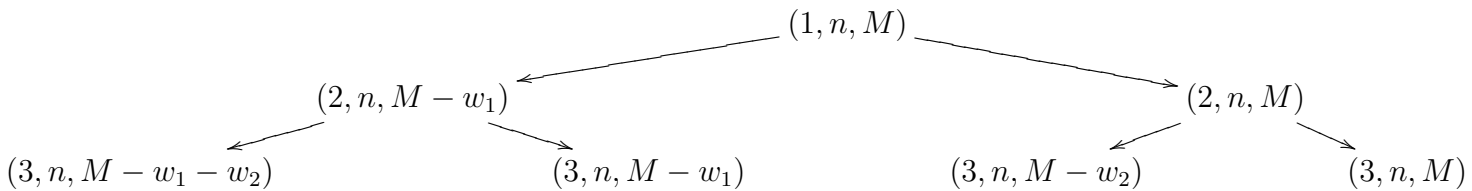
A és un vector indexat pels valors d_i. El cost espacial ve donat per l'espai requerit per emmagatzemar una solució multiplicat pel nombre de subproblemes que cal resoldre.

En molts casos, no cal emmagatzemar **tot** el vector A. Es pot anar reutilitzant l'espai corresponent a subproblemes que ja no faran falta.

El problema de la motxilla

Una motxilla de capacitat M. n objectes de pes i valor w_i i p_i, $1 \leq i \leq n$, respectivament. Quina és la combinació d'objectes que maximitza el valor sense superar la capacitat màxima?

Representem com a $(1, n, M)$ el problema original i com a (i, n, M') el subproblema restringit als $n - i + 1$ darrers objectes i capacitat $M' \leq M$.



El problema de la motxilla

Si les capacitats i els pesos venen donats per enters, cada subproblema es pot caracteritzar per un parell d'enters: el nombre d'objectes (suposada una ordenació d'aquests) i la capacitat.

Un subproblema és trivial si no queden objectes o si la capacitat és menor o igual que zero.

El problema es pot resoldre amb un bucle que es repetirà $n \cdot M$ vegades i en cada iteració caldrà calcular el mínim de dos valors.

El problema de la motxilla

Per resoldre el problema de la motxilla (amb pesos discrets) caldria un vector $A[0..n, 0..M]$.

Si només es vol calcular el valor de la solució òptima, només cal emmagatzemar els valors de les diferents subsolucions.

Si es vol calcular la solució òptima, cal aleshores emmagatzemar cada una de les $n \cdot M$ solucions en el vector.

Normalment, es calcula primer el valor de la solució òptima i s'emmagatzemen les "decisions" preses en cada posició (no tota la solució). Aleshores, es pot construir la solució òptima recorreguent el vector A al revés.

El problema de la motxilla

Dades: w, p : `vector`$[1..n]$ `de` $\mathbb{R}$; $M : \mathbb{N}$
Resultats: $val : \mathbb{R}$ // l'algorisme torna *només* el valor òptim //

```
per a m ← 0 fins a M fer
  per a i ← 0 fins a n fer
      si i = 0 aleshores A[i, m] ← 0
      si no
          si w[i] > m aleshores S ← 0
          si no
              S ← p[i] + A[i − 1, m − w[i]]
          fsi
          N ← A[i − 1, m]
          si S > N aleshores
              A[i, m] ← S
          si no
              A[i, m] ← N
          fsi
      fsi
      fper a
  fper a
val ← A[n, M]
```

El problema de la motxilla

Dades: w, p : `vector`$[1..n]$ `de` $\mathbb{R}$; $M : \mathbb{N}$

Resultats: sol : `vector`$[1..n]$ `de` $\{0,1\}$ // solució òptima //

$val : \mathbb{R}$ // valor de la solució òptima //

Auxiliar: Ant, Act : `vector`$[0..M]$ `de` $\mathbb{R}$

C : `vector`$[1..n, 0..M]$ `de` $\{0,1\}$

Inici: $Ant[m] = 0 \; m = 0, \ldots, M$

El problema de la motxilla

```
per a i ← 1 fins a n fer
        per a m ← 0 fins a M fer
                si w[i] > m aleshores S ← 0
                si no
                        S ← p[i] + Ant[m − w[i]]
                fsi
                N ← Ant[m]
                si S > N aleshores
                        C[i, m] ← 1
                        Act[m] ← S
                si no
                        C[i, m] ← 0
                        Act[m] ← N
                fsi
                fper a
        Ant ← Act          // s'actualitza Ant per a la següent iteració //
        fper a
val ← Act[M]
```

El problema de la motxilla

// obtenció de la solució òptima //

$m \leftarrow M$
per a $i \leftarrow n$ **fins a** 1 **amb** -1 **fer**
    $sol[i] \leftarrow C[i, m]$
    si $sol[i] = 1$ **aleshores** $m \leftarrow m - w[i]$ **fsi**
    fper a

El principi d'optimalitat

- Un cas particularment interessant són els problemes d'optimització.
- La solució de molts d'aquests es pot veure com una seqüència de decisions "locals" (en el cas de la motxilla, posar o no l'objecte i).
- Es diu que un problema d'optimització de la forma "trobar la seqüència òptima" compleix el principi d'optimalitat si

> tota subseqüència de la seqüència òptima és també òptima (respecte al subproblema corresponent).

El principi d'optimalitat

El problema de la motxilla compleix el principi d'optimalitat:

donats els objectes $1, \ldots i, \ldots j, \ldots n$ i les decisions associades $x_i \in \{0, 1\}$, corresponents a la solució òptima,

tota subseqüència $x_i, \ldots x_j$ ha de ser òptima per al subproblema restringit als objectes $i, \ldots j$.

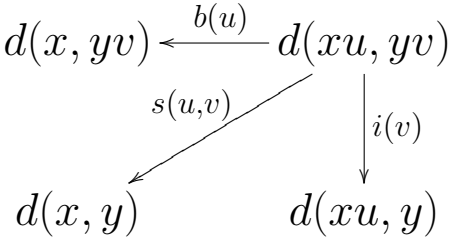
Exercici!

Distàncies entre cadenes

- com es poden definir distàncies entre cadenes de símbols?
- una possibilitat és considerar operacions simples sobre cadenes (inserció, borrat i substitució).
- donades dues cadenes, sempre es pot convertir una en l'altra aplicant una seqüència d'operacions.
- Es defineix la **distància d'edició entre cadenes** com el menor número d'operacions que calen per convertir una cadena en l'altra.
- Es poden assignar pesos a les operacions i definir aleshores la distància com el pes de la seqüència de mínim pes (que converteix una en l'altra).

Distàncies entre cadenes. Descomposició

Donades dues cadenes, x, y i dos símbols, u i v, es té

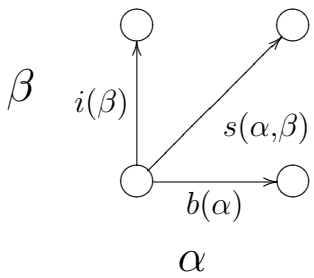


$$d(xu, yv) = \min\{d(x, yv) + b(u), d(x, y) + s(u, v), d(xu, y) + i(v)\}$$

Una vegada fixades les cadenes cada subproblema ve donat per dos índexs sobre elles.

Interpretació gràfica. Quadrícules

Donades dues cadenes x i y, es defineix una quadrícula de $(|x|+1)(|y|+1)$ nodes i arcs horitzontals, verticals i diagonals.



El problema de la distància d'edició és equivalent al problema de trobar el camí entre dos nodes de la quadrícula de pes mínim.

Interpretació gràfica. Quadrícules

En aquest cas no ens interessa la solució sinó el valor (mínim) de la solució.

El problema es pot resoldre en temps $|x| \cdot |y|$ i en espai $\min\{|x|, |y|\}$.

Camins mínims. L'algorisme de Floyd

Calcular (el valor de) tots els camins mínims (entre tot parell de nodes) en un graf dirigit i ponderat.

Siga un graf de n nodes, $u_1, \ldots, u_n$. Siga $P(n)$ el problema de trobar tots els camins mínims entre els n nodes.

S'introdueix un nou paràmetre, k, $1 \leq k \leq n$ i es defineix $P(n,k)$ com el problema de trobar tots els camins mínims entre els n nodes fent servir com a nodes de pas $u_1, \ldots, u_k$.

Es compleix que $P(n)$ és equivalent a $P(n,n)$.

El plantejament recursiu consisteix a relacionar $P(n,k)$ amb $P(n,k-1)$.

Arbres binaris de cerca òptims

Siguen n claus, $k_1, \ldots, k_n$ amb probabilitats d'ocurrència $p_1, \ldots, p_n$. Quin seria l'arbre binari de cerca òptim en aquest cas concret?

O, quin és l'arbre que minimitzaria el nombre esperat de comparacions (suposant sempre cerques exitoses)?

(el arbre equilibrat només és l'òptim si les claus són equiprobables)

Exemple: claus 1, 2, 3 amb probabilitats .8, .1 i .1, respectivament.

L'arbre òptim: $1 \searrow 2 \searrow 3$ Cost mitjà: $1 \cdot (.8) + 2 \cdot (.1) + 3 \cdot (.1) = 1.3$

Arbres binaris de cerca òptims

Plantejament recursiu:

- Donades n claus, l'arbre òptim haurà de tindre com a arrel una d'elles.
- Fixada k_i com a arrel, tant a la dreta com a l'esquerra es té el subarbre òptim format per les claus $k_1, \ldots, k_{i-1}$ i $k_{i+1}, \ldots, k_n$, respectivament.
- Cada subproblema és de la forma $k_i, \ldots k_j$ i està identificat per dos enters, i i j.