Fonctionnalité et Modularité

Cours 9

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-fm

Plan

- algorithmes gloutons
- exploration exhaustive
- retours en arrière (*backtracking*)
- programmation dynamique

télécharger Ocaml en `http://www.ocaml.org`

Exploration

on distingue 3 méthodes d'exploration

- algorithmes **gloutons**
  - un choix local permet d'obtenir la solution globale

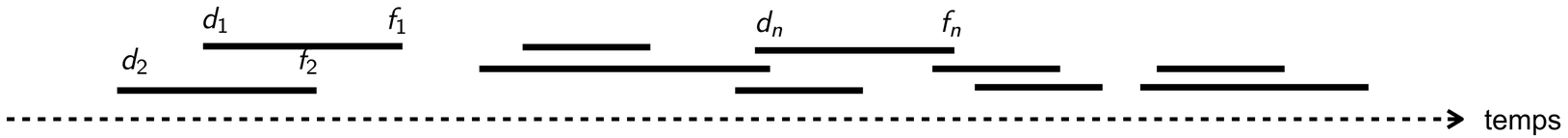
- exploration **exhaustive**
  - on parcourt les solutions globales jusqu'à trouver la bonne solution
  - retours arrière possibles (*backtracking*)

- programmation **dynamique**
  - on mémorise tous les résultats partiels pour obtenir la solution globale
  - demande de la mémoire supplémentaire

Allocation de ressource

Problème

- on gère un magasin et un seul ticket « super-bonus »
- les clients $c_1, c_2, \cdots c_n$ rentrent aux temps $d_1, d_2, \cdots d_n$ et partent aux temps $f_1, f_2, \cdots f_n$
- on veut donner le ticket super-bonus au maximum de clients



Solution

- on trie les clients par ordre croissant de dates de fin
- et on prend les clients dans cet ordre avec la contrainte :

$$i < j \implies f_i < d_j$$

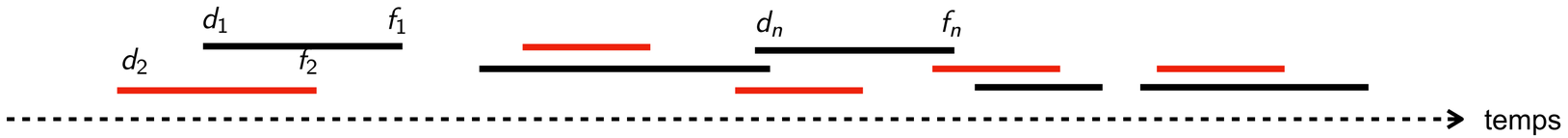
[plus compliqué si plusieurs ressources]

algorithme glouton

Allocation de ressource

Problème

- on gère un magasin et un seul ticket « super-bonus »
- les clients $c_1, c_2, \cdots c_n$ rentrent aux temps $d_1, d_2, \cdots d_n$ et partent aux temps $f_1, f_2, \cdots f_n$
- on veut donner le ticket super-bonus au maximum de clients



Solution

- on trie les clients par ordre croissant de dates de fin
- et on prend les clients dans cet ordre avec la contrainte :

$$i < j \implies f_i < d_j$$

[plus compliqué si plusieurs ressources]

algorithme glouton

Exercice

- montrer que cet algorithme donne la solution maximale !

Algorithme glouton

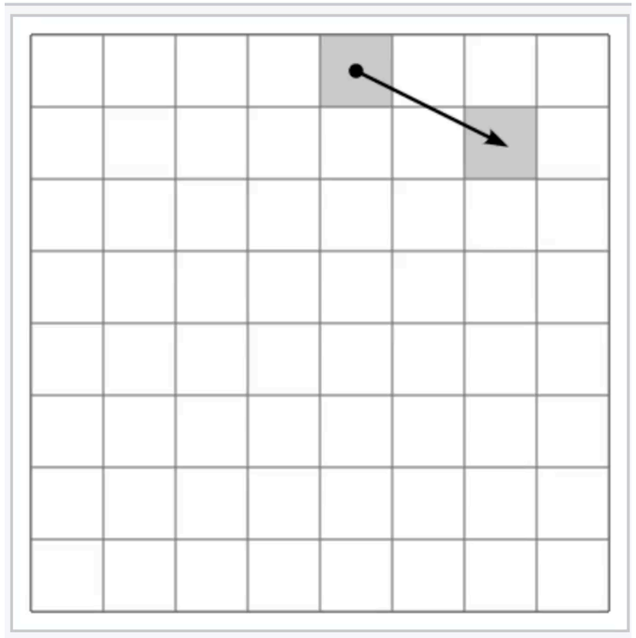
- plus court chemin dans un graphe (`Dijkstra`)

[on cherche le minimum local — cf cours 8]

- arbre de recouvrement minimal dans un graphe non-orienté valué (`Kruskal, Prim`)

- allocation de ressources, etc

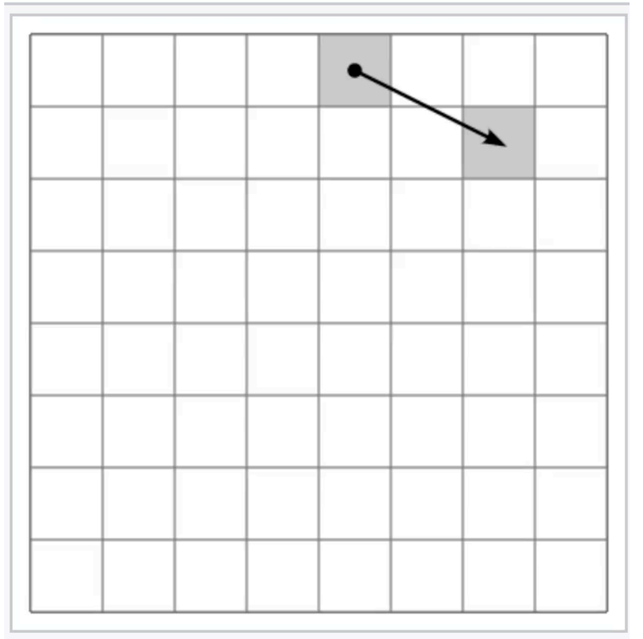
- marche du cavalier pour couvrir toutes les cases d'un échiquier



Marche du cavalier

- marche du cavalier pour couvrir toutes les cases d'un échiquier

- on se déplace vers la case où il y aura le moins de déplacements possibles pour le cavalier



```
let rec marche a i j k =
    a.(i).(j) <- k ;
    let (i', j') = caseMinCoupsJouables a i j in
    if (i', j') = (i, j) then k else
          marche a i' j' (k+1) ;;
```

```
let une_solution n i j =
    let a = Array.make_matrix n n libre in
    if marche a i j 1 < n*n then
       Printf.printf "pas de solution! \n"
       else print_matrix a ;;
```

une_solution 8 0 4

```
47  14  61  32   1  16  19  34
64  31  46  15  60  33   2  17
13  48  57  62  45  18  35  20
30  63  42  53  56  59  40   3
49  12  55  58  41  44  21  36
26  29  52  43  54  39   4   7
11  50  27  24   9   6  37  22
28  25  10  51  38  23   8   5
```

Marche du cavalier

- marche du cavalier pour couvrir toutes les cases d'un échiquier
- on se déplace vers la case où il y aura le moins de déplacements possibles pour le cavalier

```
let libre = -1 ;;
let infini = max_int ;;
let dX = [| 2; 1; -1; -2; -2; -1;  1; 2 |] ;;
let dY = [| 1; 2; 2;  1; -1; -2; -2; -1 |] ;;
let range_dXY = List.init (Array.length dX) (fun i -> i) ;;
```

```
let jouable a i j =
    0 <= i && i < Array.length a &&
    0 <= j && j < Array.length a.(0) &&
      a.(i).(j) = libre ;;
```

```
let nbDeCoupsJouables a i j =
    if not (jouable a i j) then infini else
      List.fold_left (fun r k ->
        if jouable a (i + dX.(k)) (j + dY.(k)) then r+1 else r)
      0 range_dXY ;;
```

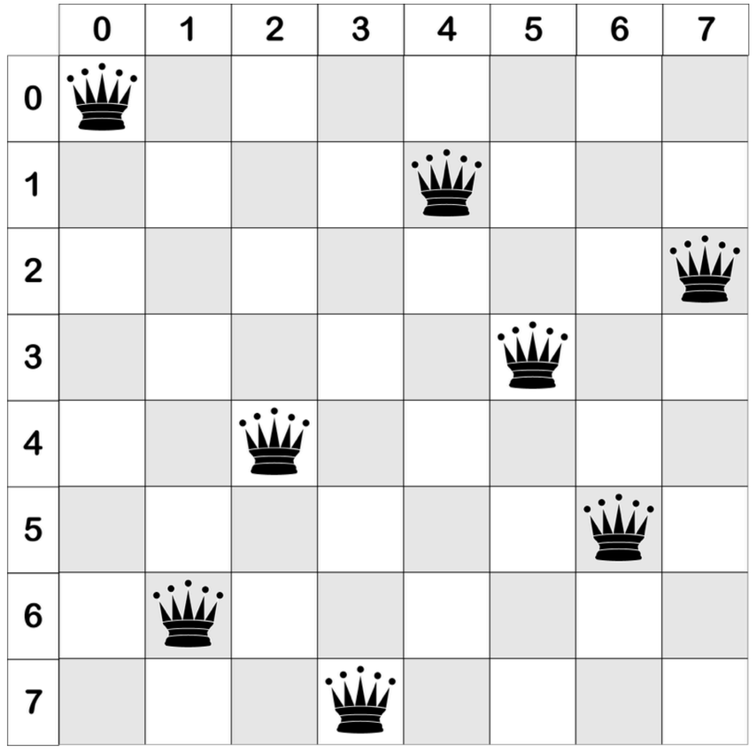
```
let caseMinCoupsJouables a i j =
    let i1 = ref i and j1 = ref j in
    let min = ref infini in
    Array.iter2 (fun dx dy ->
        let i' = i + dx and j' = j + dy in
        let nk = nbDeCoupsJouables a i' j' in
        if nk < !min then begin
            min := nk ; i1 := i'; j1 := j'
          end) dX dY;
    (!i1, !j1) ;;
```

Recherche exhaustive

- problème du sac à dos (ranger le maximum d'objets dans un sac)
- voyageur de commerce et tous les problèmes NP
- les 8 reines (placer 8 reines sur un échiquier sans qu'elle ne soit en prise par une autre reine)

```
let conflit i1 j1 i2 j2 =
  i1 = i2 || j1 = j2 ||
  abs (i1 - i2) = abs (j1 - j2);;
```

teste si la reine en `(i2, j2)` peut prendre la reine en `(i1, j1)`



Les 8 reines

- les 8 reines (placer 8 reines sur un échiquier sans qu'elle ne soit en prise par une autre reine
  on explore les solutions avec possibles retours arrière (***backtracking***)

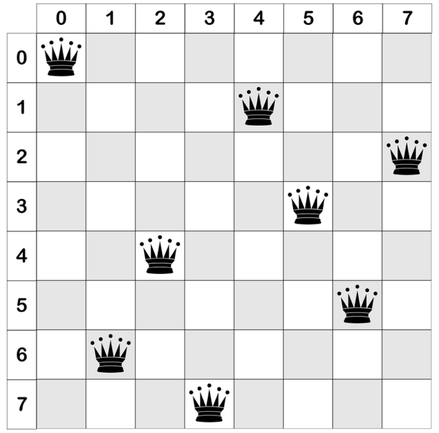
```
let compatible i j pos =
  List.for_all
    (fun i' -> not (conflit i' pos.(i') i j))
       (range i) ;
```

```
let rec reines n i pos =
   if i >= n then
     imprimerSolution pos
   else
     List.iter (fun j ->
        if compatible i j pos then begin
           pos.(i) <- j;
           reines n (i+1) pos
        end)
      (range n) ;;
```

ici *backtracking* si pas de solution à partir de ligne `i+1` (on passe alors à la colonne suivante)

```
let nReines n =
    let pos = Array.make n 0 in
    pos.(0) <- 4;
    reines n 1 pos ;;
```

on imprime toutes les solutions en partant de la reine en (0, 4)




```
p = [| 0; 4; 7; 5; 2; 6; 1; 3 |]
```

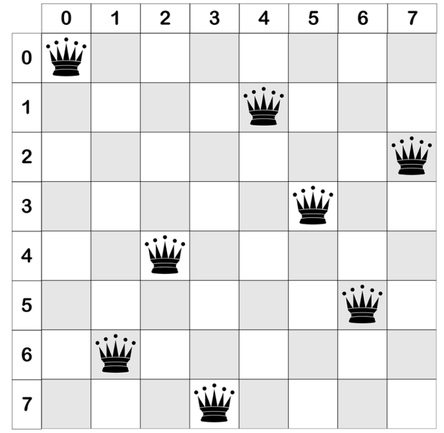
```
let range i = List.init i (fun k -> k) ;;
```

Les 8 reines

- impression de la solution

```
let print_matrix a =
    Array.iter (fun line ->
      Array.iter (Printf.printf "%2s ") line ;
                 Printf.printf "\n") a ;;

let imprimerSolution pos =
    let n = Array.length pos in
    let a = Array.make_matrix n n "." in
    List.iter (fun i -> a.(i).(pos.(i)) <- "R")
       (range n) ;
    print_matrix a ;
    Printf.printf "----------------------------\n" ;;
```

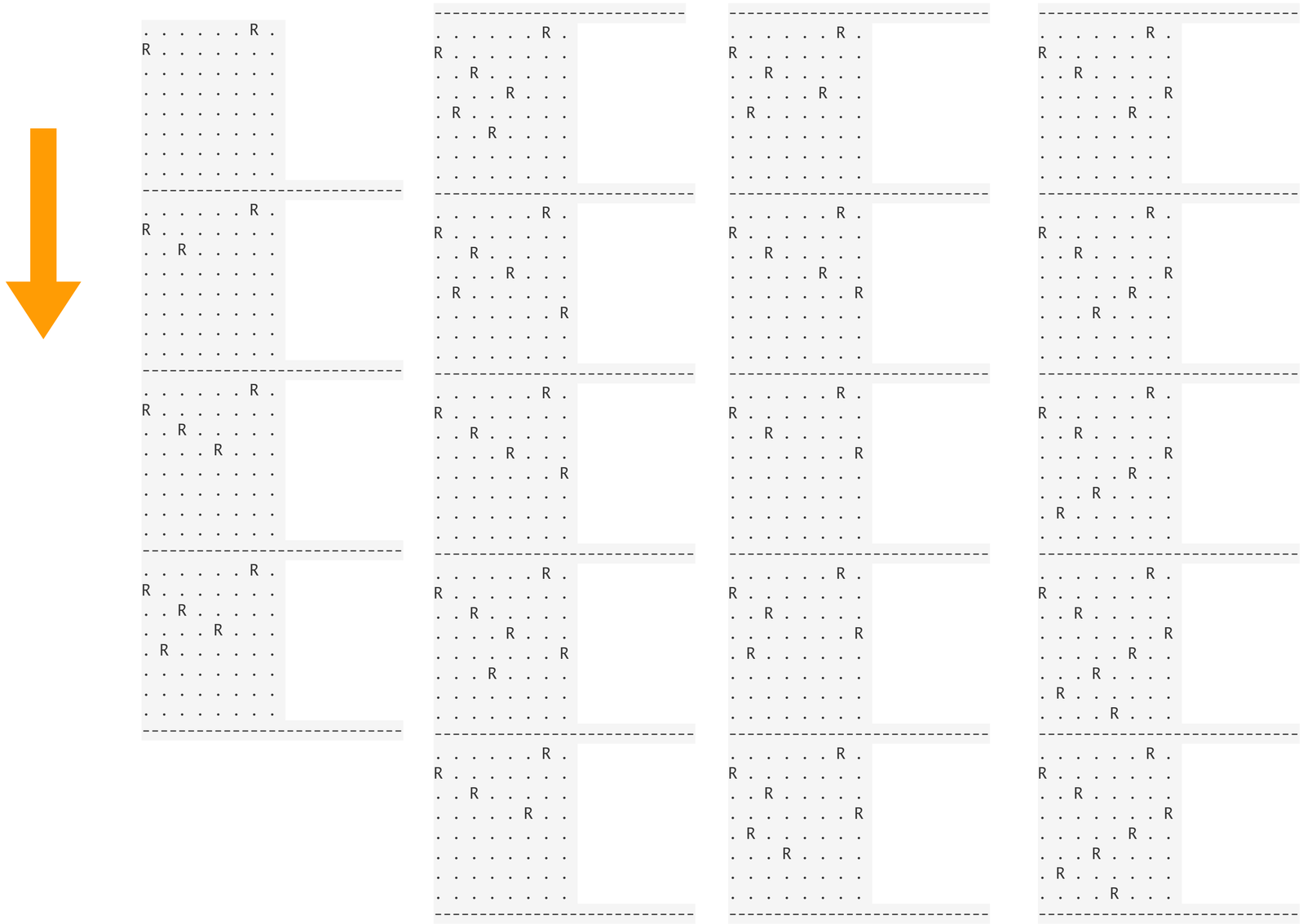



```
p = [| 0; 4; 7; 5; 2; 6; 1; 3 |]
```

```
R . . . . . . .
. . . . R . . .
. . . . . . . R
. . . . . R . .
. . R . . . . .
. . . . . . R .
. R . . . . . .
. . . R . . . .
```

Les 8 reines

- avancées et retours en arrière de `reines (n, i, pos)`



	0	1	2	3	4	5	6	7
0	♛							
1					♛			
2								♛
3						♛		
4			♛					
5							♛	
6		♛						
7				♛				

```
p = [| 0; 4; 7; 5; 2; 6; 1; 3 |]
```

```
R . . . . . . .
. . . . R . . .
. . . . . . . R
. . . . . R . .
. . R . . . . .
. . . . . . R .
. R . . . . . .
. . . R . . . .
```

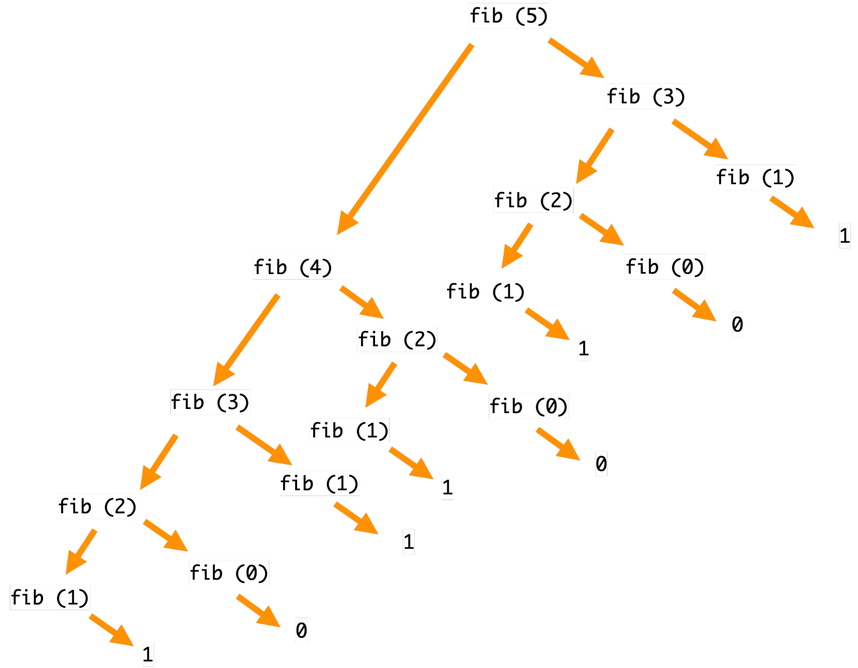
Programmation dynamique

- la fonction de Fibonacci fait beaucoup d'appels récursifs

```
let rec fib n = if n <= 1 then n else fib (n-1) + fib (n-2) ;;
```

- on peut simplement la calculer en mémorisant les valeurs intermédiaires

```
let fib_dyn n =
  let res = Array.make (n+1) 0 in
    res.(0) <- 0; res.(1) <- 1;
    for i = 2 to n do
      res.(i) <- res.(i-1) + res.(i-2)
    done;
    res.(n) ;;
```



Programmation dynamique

- plus longue sous-séquence commune entre 2 chaînes de caractères (commande Unix `diff`)

  [on mémorise les solutions partielles — $m \times n$ opérations]

```
(* gauche, haut, diagonale, stop *)
type sens = G | H | D | S ;;


let longueurSSC u v =
  let m = String.length u in
  let n = String.length v in
  let lg = Array.make_matrix (m+1) (n+1) 0 in
  let p = Array.make_matrix (m+1) (n+1) S in
  for i = 1 to m do
    for j = 1 to n do
       if u.[i-1] = v.[j-1] then begin
         lg.(i).(j) <- 1 + lg.(i-1).(j-1);
         p.(i).(j) <- D
       end else if lg.(i).(j-1) > lg.(i-1).(j) then begin
         lg.(i).(j) <- lg.(i).(j-1) ;
         p.(i).(j) <- G
       end else begin
         lg.(i).(j) <- lg.(i-1).(j) ;
         p.(i).(j) <- H
      end
    done
  done ;
  (lg.(m).(n), p) ;;
```

u = ‘abcadefg’ v = ‘fbcexyg’

lg

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1
0 0 1 2 2 2 2 2
0 0 1 2 2 2 2 2
0 0 1 2 2 2 2 2
0 0 1 2 3 3 3 3
0 1 1 2 3 3 3 3
0 1 1 2 3 3 3 4
```

p

```
0 0 0 0 0 0 0 0
0 2 2 2 2 2 2 2
0 2 3 1 1 1 1 1
0 2 2 3 1 1 1 1
0 2 2 2 2 2 2 2
0 2 2 2 2 2 2 2
0 2 2 2 3 1 1 1
0 3 2 2 2 2 2 2
0 2 2 2 2 2 2 3
```

Programmation dynamique

- plus longue sous-séquence commune entre 2 chaînes de caractères (commande Unix `diff`)

  [on mémorise les solutions partielles — $m \times n$ opérations]

```
let ssc u v =
  let m = String.length u in
  let n = String.length v in
  let (lg, p) = longueurSSC u v in
  let rec loop lg i j r =
    if lg = 0 then r else begin
      match p.(i).(j) with
      | D -> loop (lg-1) (i-1) (j-1) ((String.make 1 u.[i-1]) ^ r)
      | G -> loop lg i (j-1) r
      | H -> loop lg (i-1) j r
      | _ -> failwith "Impossible" end in
  loop lg m n "";;


print_string (ssc "abcadefg" "fbcexyg") ;;
```

u = ‘abcadefg’ v = ‘fbcexyg’

lg

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1
0 0 1 2 2 2 2 2
0 0 1 2 2 2 2 2
0 0 1 2 2 2 2 2
0 0 1 2 3 3 3 3
0 1 1 2 3 3 3 3
0 1 1 2 3 3 3 4
```

p

```
0 0 0 0 0 0 0 0
0 2 2 2 2 2 2 2
0 2 3 1 1 1 1 1
0 2 2 3 1 1 1 1
0 2 2 2 2 2 2 2
0 2 2 2 2 2 2 2
0 2 2 2 3 1 1 1
0 3 2 2 2 2 2 2
0 2 2 2 2 2 2 3
```

Conclusion

VU:

- algorithme glouton
- exploration exhaustive
- programmation dynamique

TODO list

- objets
- parallélisme
- autres langages fonctionnels