Chapitre 9

Correction des exercices

Exercice 1 Notons $f(n,p)$ le nombre de chemins possibles dans une grille $n \times p$. On dispose naturellement des relations :

$$f(0,p) = f(n,0) = 1 \quad \text{et} \quad f(n,p) = f(n-1,p) + f(n-1,p-1) + f(n,p-1) \text{ si } n \geqslant 1 \text{ et } p \geqslant 1.$$

Ceci conduit à la définition récursive suivante :

```
let rec f = fun
  | 0 p | n 0 -> 1
  | n p -> f (n-1) p + f (n-1) (p-1) + f n (p-1) ;;
```

```
# f 5 5 ;;
- : int = 1683
```

Malheureusement, cette définition montre vite ses limites quand les valeurs de n et p augmentent car les appels récursifs ne sont pas indépendants. On utilise donc la programmation dynamique en créant un tableau bi-dimensionnel destiné à contenir les valeurs $f(n,p)$:

```
let f n p =
  let t = make_matrix (n+1) (p+1) 1 in
  for i = 1 to n do
    for j = 1 to p do
      t.(i).(j) <- t.(i-1).(j) + t.(i-1).(j-1) + t.(i).(j-1)
    done
  done ;
  t.(n).(p) ;;
```

```
# f 20 20 ;;
- : int = 260543813797441
```

Chaque ligne du tableau ne dépendant que de la précédente on peut même se contenter d'un tableau uni-dimensionnel, à condition de bien respecter l'ordre des dépendances :

```
let f n p =
  let t = make_vect (p+1) 1 in
  let v = ref 1 in
  for i = 1 to n do
    for j = 1 to p do
      let a = t.(j) in
      t.(j) <- t.(j-1) + t.(j) + !v ;
      v := a
    done ;
    v := 1
  done ;
  t.(p) ;;
```

Exercice 2 Là encore, la définition récursive est presque immédiate, mais s'avère rapidement limitée :

```
let chemin_max t =
  let n = vect_length t in
  let rec f = fun
    | i j when i = n-1 -> t.(i).(j)
    | i j -> t.(i).(j) + max (f (i+1) j) (f (i+1) (j+1)) in
  f 0 0 ;;
```

Si on se permet de modifier les valeurs du tableau `t`, on préférera la version suivante :

```
let chemin_max t =
  let n = vect_length t in
  for i = n-2 downto 0 do
    for j = 0 to i do
      t.(i).(j) <- t.(i).(j) + max t.(i+1).(j) t.(i+1).(j+1)
    done
  done ;
  t.(0).(0) ;;
```

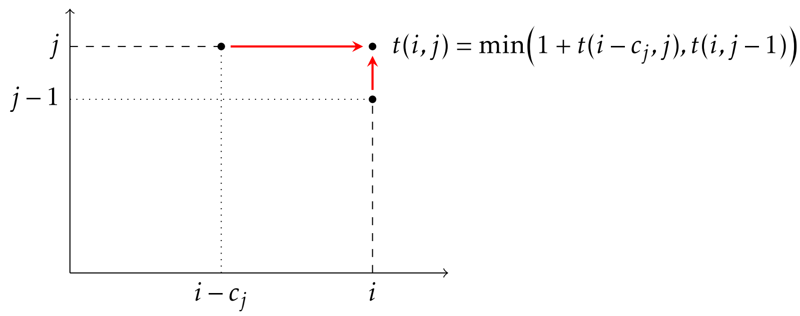
Exercice 3 Notons $S' = \{c_1, c_2, \dots, c_{p-1}\}$.
Si $n < c_p$, alors $f(n,S) = f(n,S')$.
Si $n \geqslant c_p$, deux cas de figure sont possibles :

- si la décomposition optimale de n utilise la pièce c_p, alors $f(n,S) = 1 + f(n-c_p,S)$;
- si la décomposition optimale de n n'utilise pas la pièce c_p, alors $f(n,S) = f(n,S')$.

On en déduit que $f(n,S) = \min\bigl(1 + f(n-c_p,S), f(n,S')\bigr)$.
Nous allons donc utiliser un tableau de taille $(n+1)\times(p+1)$ pour stocker les différentes valeurs de $f\bigl(i,(c_1,\dots,c_j)\bigr)$, avec les valeurs initiales $f(0,j) = 0$ pour $j \geqslant 0$ et $f(i,0) = +\infty$ pour $i \geqslant 1$.

```
let f n c =
  let p = vect_length c in
  let t = make_matrix (n+1) (p+1) 0 in
  for i = 1 to n do t.(i).(0) <- max_int done ;
  for i = 1 to n do
    for j = 1 to p do
      match i < c.(j-1) with
        | true -> t.(i).(j) <- t.(i).(j-1)
        | false -> t.(i).(j) <- min (1 + t.(i-c.(j-1)).(j)) t.(i).(j-1)
    done
  done ;
  t.(n).(p) ;;
```

On peut n'utiliser qu'un tableau uni-dimensionnel de taille $n+1$ en respectant l'ordre des dépendances :




```
let f n c =
  let p = vect_length c in
  let t = make_vect (n+1) 0 in
  for i = 1 to n do t.(i) <- max_int done ;
  for i = 1 to n do
    for j = 1 to p do
      if i >= c.(j-1) && 1 + t.(i-c.(j-1)) < t.(i) then
        t.(i) <- 1 + t.(i-c.(j-1))
    done
  done ;
  t.(n) ;;
```

Enfin, si on souhaite garder trace de la décomposition optimale, on peut adjoindre à chaque élément du tableau la liste des pièces à utiliser pour le décomposer. Cela donne :

```
let f n c =
  let p = vect_length c in
  let t = make_vect (n+1) (0, []) in
  for i = 1 to n do t.(i) <- (max_int, []) done ;
  for i = 1 to n do
    for j = 1 to p do
      if i >= c.(j-1) then
        let (x, _) = t.(i) and (y, lst) = t.(i-c.(j-1)) in
        if y + 1 < x then t.(i) <- (y+1, c.(j-1)::lst)
    done
  done ;
  t.(n) ;;
```

Exercice 4 Notons $f(i,j)$ le nombre minimal de multiplications nécessaires pour effectuer le produit $\mathrm{M}_i \cdots \mathrm{M}_j$. Si on calcule ce produit en l'écrivant sous la forme $(\mathrm{M}_i \cdots \mathrm{M}_{k-1}) \times (\mathrm{M}_k \cdots \mathrm{M}_j)$, on utilise $f(i,k-1) + f(k,j) + m_i m_k m_{j+1}$ multiplications, ce qui nous conduit aux relations :

$$f(i,i) = 0 \quad \text{et} \quad f(i,j) = \min_{i<k\leqslant j}\Big(f(i,k-1) + f(k,j) + m_i m_k m_{j+1}\Big).$$

Nous allons donc utiliser un tableau bi-dimensionnel dans lequel **`f.(i).(j)`** contiendra la valeur de $f(i,j)$.
Pour des raisons de lisibilité, on commence par une fonction qui calcule le minimum d'une fonction sur un intervalle :

```
let rec minimum h i = function
  | j when i = j -> h i
  | j -> min (h j) (minimum h i (j-1)) ;;
```

Il reste à remplir le tableau en respectant les dépendances :

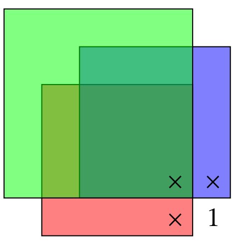
```
let mult_min m =
  let n = vect_length m - 1 in
  let f = make_matrix n n 0 in
  for p = 1 to n-1 do          (* j = i + p *)
    for i = 0 to n-1-p do
      let h k = f.(i).(k-1) + f.(k).(i+p) + m.(i)*m.(k)*m.(i+p+1)
        in f.(i).(i+p) <- minimum h (i+1) (i+p)
    done
  done ;
  f.(0).(n-1) ;;
```

Il est maintenant facile de réaliser l'application numérique qui nous est demandée :

```
  # let m = make_vect 51 0 in
      for k = 0 to 50 do m.(k) <- k+1 done ;
      mult_min m ;;
  - : int = 44198
```

Exercice 5 Nous allons noter $f(i,j)$ la taille maximale d'un carré de 1 dont le coin inférieur droit est à l'emplacement (i,j) ; il s'agit donc de calculer $\max_{i,j} f(i,j)$.

- Si $\mathrm{A}[i,j] = 0$, nous avons bien entendu $f(i,j) = 0$.
- Si $\mathrm{A}[i,j] = 1$, la taille du carré de 1 que l'on cherche dépend des trois carrés maximaux dont les coins inférieurs droits sont situés en $(i-1,j)$, $(i,j-1)$ et $(i-1,j-1)$:



Plus précisément, $f(i,j) = \min\big(f(i-1,j), f(i,j-1), f(i-1,j-1)\big) + 1$.

Cette formule permet de remplir la table des $f(i,j)$, tout en calculant en parallèle le maximum de ces valeurs :

```
let carre a =
  let m = vect_length a and n = vect_length a.(0) in
  let f = make_matrix m n 0 in
  let maxi = ref 0 in
  for i = 0 to m-1 do
    f.(i).(0) <- a.(i).(0) ;  (* valeurs initiales de la première colonne *)
    maxi := max !maxi f.(i).(0)
  done;
  for j = 1 to n-1 do
    f.(0).(j) <- a.(0).(j) ;  (* valeurs initiales de la première ligne *)
    maxi := max !maxi f.(0).(j)
  done;
  for i = 1 to m-1 do
    for j = 1 to n-1 do
      if a.(i).(j) = 1 then
        (f.(i).(j) <- 1 + min f.(i-1).(j) (min f.(i).(j-1) f.(i-1).(j-1)) ;
         maxi := max !maxi f.(i).(j))
    done
  done;
  !maxi ;;
```