TD 13

MAXIMISATION D'UNE SOMME

1 Introduction

Le but de ce TD est de résoudre le problème 345 du Projet Euler (`http://projecteuler.net`), à savoir : étant donné une matrice carrée M de taille n, de coefficients $(m_{ij})_{i,j\in[\![0,n[\![^2}$, calculer

$$s(M) = \max_{\sigma\in S_n} \sum_{i=0}^{n-1} m_{ij}$$

où S_n désigne l'ensemble des permutations de $[\![0, n[\![$.

Pour

$$M_1 = \begin{pmatrix} 7 & 53 & 183 & 439 & 863 \\ 497 & 383 & 563 & 79 & 973 \\ 287 & 63 & 343 & 169 & 583 \\ 627 & 343 & 773 & 959 & 943 \\ 767 & 473 & 103 & 699 & 303 \end{pmatrix}$$

on obtient $s(M_1) = 3315$.

Le but est de calculer $s(M_2)$ où M_2 est la matrice 15×15 définie par

$$M_2 = \begin{pmatrix} 7 & 53 & 183 & 439 & 863 & 497 & 383 & 563 & 79 & 973 & 287 & 63 & 343 & 169 & 583 \\ 627 & 343 & 773 & 959 & 943 & 767 & 473 & 103 & 699 & 303 & 957 & 703 & 583 & 639 & 913 \\ 447 & 283 & 463 & 29 & 23 & 487 & 463 & 993 & 119 & 883 & 327 & 493 & 423 & 159 & 743 \\ 217 & 623 & 3 & 399 & 853 & 407 & 103 & 983 & 89 & 463 & 290 & 516 & 212 & 462 & 350 \\ 960 & 376 & 682 & 962 & 300 & 780 & 486 & 502 & 912 & 800 & 250 & 346 & 172 & 812 & 350 \\ 870 & 456 & 192 & 162 & 593 & 473 & 915 & 45 & 989 & 873 & 823 & 965 & 425 & 329 & 803 \\ 973 & 965 & 905 & 919 & 133 & 673 & 665 & 235 & 509 & 613 & 673 & 815 & 165 & 992 & 326 \\ 322 & 148 & 972 & 962 & 286 & 255 & 941 & 541 & 265 & 323 & 925 & 281 & 601 & 95 & 973 \\ 445 & 721 & 11 & 525 & 473 & 65 & 511 & 164 & 138 & 672 & 18 & 428 & 154 & 448 & 848 \\ 414 & 456 & 310 & 312 & 798 & 104 & 566 & 520 & 302 & 248 & 694 & 976 & 430 & 392 & 198 \\ 184 & 829 & 373 & 181 & 631 & 101 & 969 & 613 & 840 & 740 & 778 & 458 & 284 & 760 & 390 \\ 821 & 461 & 843 & 513 & 17 & 901 & 711 & 993 & 293 & 157 & 274 & 94 & 192 & 156 & 574 \\ 34 & 124 & 4 & 878 & 450 & 476 & 712 & 914 & 838 & 669 & 875 & 299 & 823 & 329 & 699 \\ 815 & 559 & 813 & 459 & 522 & 788 & 168 & 586 & 966 & 232 & 308 & 833 & 251 & 631 & 107 \\ 813 & 883 & 451 & 509 & 615 & 77 & 281 & 613 & 459 & 205 & 380 & 274 & 302 & 35 & 805 \end{pmatrix}$$

On se donne la contrainte que le programme doit résoudre le problème en moins de 2 minutes.

On représentera ces deux matrices en Caml par un tableau de lignes, chaque ligne étant représentée par un tableau. Vous pouvez récupérer les matrices sur `http://bianquis.org`.

2 Algorithme naïf

L'algorithme naïf consiste à tirer toutes les permutations de n éléments, à calculer pour chacune la somme des éléments.

1. Estimer le nombre d'opérations nécessaires pour tester une permutation pour une matrice de taille n, puis pour tester pour l'ensemble des permutations et calculer le résultat.
2. Écrire une fonction `nbops` prenant en argument n et rendant cette valeur. *Attention, il faut faire les calculs en flottant pour éviter les dépassements.*

3. Écrire une fonction `tempsop` prenant en argument n et rendant le temps qu'il convient de mettre par opération pour que le temps total d'exécution soit de deux minutes. On rendra le résultat en nanosecondes.
4. Que vaut `tempsop 15` ? Peut-on raisonnablement utiliser l'algorithme naïf pour calculer le résultat pour M_2 ?

3 Un deuxième algorithme naïf

Une matrice carrée de taille M et une partie S de $[\![0, n[\![$ étant données, on note $eval(M, S)$ le maximum des $\sum_{i=0}^{\text{Card}\, S-1} m_{i,f(i)}$ où f parcourt les bijections de $[\![0, \text{Card}\, S[\![$ dans S.

On a évidemment $eval(M, [\![0, n[\![) = s(M)$.

1. Que représente $eval(M, S)$ (faire un dessin !).
2. Que vaut $eval(M, S)$ pour $S = \emptyset$?
3. Pour $S \neq \emptyset$, exprimer $eval(M, S)$ en fonction des $eval(M, S \setminus \{i\})$ pour $i \in S$ et des coefficients de M.
4. Écrire une fonction `eval` prenant en argument une matrice carrée M et S et calculant récursivement $eval(M, S)$. On représentera un sous-ensemble de $[\![0, n-1]\!]$ par la liste de ses éléments.
5. Vérifier qu'elle calcule le résultat correct pour M_1.
6. Quelle est la complexité de cette fonction ?

4 Solution en programmation dynamique

1. Pour une matrice M donnée et $S \subset [\![0, n[\![$, combien l'appel de $eval(M, [\![0, n[\![)$ provoque t-il d'appels de $eval(M, S)$?
2. On en déduit qu'il serait intéressant de mémoriser les résultats de ces appels. M étant fixée, on peut imaginer de stocker dans une table globale les valeurs des couples $(S, eval(M, S))$ à chaque fois qu'on les calcule de façon à ne les calculer qu'une seule fois.

   On procède donc comme suit pour calculer $eval(M, S)$:
   - Si la table globale contient déjà une entrée (S, v) on sait que v est la valeur cherchée pour $eval(M, S)$: on la rend directement.
   - Si aucune entrée de cette forme (S, v) n'existe dans la table, on calcule $eval(M, S)$ par la méthode récursive vue plus haut : on calcule récursivement les $eval(M, S \setminus \{i\})$ (en notant au passage dans la table les valeurs calculées si nécessaire). On peut alors calculer la valeur v de $eval(M, S)$. On note (S, v) dans la table et on retourne v.

   En supposant que les recherches et les ajouts dans la table se fassent en temps constant, quel est la complexité de cet algorithme ?
3. Pour écrire la version mémoïsée, on va utiliser les tables de hachage qui sont prédéfinies en Caml. On aura besoin des fonctions suivantes :
   - `Hashtbl.create : int -> ('_a, '_b) Hashtbl.t` qui crée une table de hachage vide (l'entier passé en argument donne la taille initiale de la table, il est souhaitable qu'il soit de l'ordre du nombre d'éléments que l'on compte stocker dans la table) ;
   - `Hashtbl.mem : ('a, 'b) Hashtbl.t -> 'a -> bool` telle que `Hashtbl.mem t k` renvoie `true` si `t` a une valeur associée à `k`, `false` sinon ;
   - `Hashtbl.find : ('a, 'b) Hashtbl.t -> 'a -> 'b` qui prend en argument une table et une clé et renvoie la valeur associée à la clé (ou une exception `Not_found` s'il n'y a pas de telle valeur) ;
   - `Hashtbl.add : ('a, 'b) Hashtbl.t -> 'a -> 'b -> unit` telle que `Hashtbl.add t k v` rajoute l'association (k, v) à la table t.

   Faire quelques essais avec ces fonctions pour se familiariser avec leur fonctionnement.
4. Écrire une fonction `eval_memo` et calculer $s(M_2)$. On utilisera une `(int list, int) Hashtbl.t`. Il faudra prendre garde à ce que les listes soient toutes triées (sinon, deux listes peuvent paraître différentes alors qu'elles correspondent au même ensemble).

5 Optimisation

On peut représenter le sous-ensembles S de $[\![0, n[\![$ par l'entier $\sum_{i \in S} 2^{i-1}$. Du coup, les parties de S sont représentées par des entiers compris entre 0 et $2^n - 1$ inclus. On n'a alors plus besoin d'une table de hachage : on peut utiliser un tableau de 2^n entiers à la place.

Écrire une version *bottom-up* de l'algorithme dynamique précédent en utilisant cette structure de données.