Maximisation de 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 $M = (m_{i,j}) \in \mathcal{M}_n(\mathbb{N})$, calculer

$$s(M) = \max_{\sigma \in \mathcal{S}_n} \sum_{i=0}^{n-1} m_{i,\sigma(i)}$$

où $\mathcal{S}_n$ désigne l'ensemble des permutations de $E_n = \{0 \dots n-1\}$.

Pour la matrice M_1 ci-dessous, on obtient $s(M_1) = 3315$.

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

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 d'une minute.

On représentera ces deux matrices en OCaml par un tableau de lignes, chaque ligne étant représentée par un tableau ; elles sont définies dans le fichier joint.

2 Algorithme naïf

L'algorithme naïf consiste à énumérer toutes les permutations de $\mathcal{S}_n$ et à calculer pour chacune la somme des éléments.

1. En ne comptant que les additions entre éléments de la matrice, combien d'opérations faut-il faire avec cette méthode pour traiter une matrice de taille n ?
2. En utilisant la « recette de cuisine » du cours sur la complexité, donner alors une fourchette réaliste pour le temps d'exécution sur des matrices de taille 8, 10, 12, 15 et 20. Que peut-on conclure pour notre problème ?

3 Récursion naïve

Une matrice carrée de taille M et une partie S de E_n étant données, on définit

$$eval(M, S) := \max \left\{ \sum_{i=0}^{|S|-1} m_{i,f(i)} \mid f : [0 \dots |S|-1] \to S \text{ avec } f \text{ bijective} \right\}$$

3.1 Solution récursive naïve

1. Que valent $eval(M, \varnothing)$? $eval(M, E_n)$?
2. Que représente $eval(M, S)$? *Faire un dessin !*
3. Pour $S \neq \varnothing$, 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 `range : int -> int -> int list` telle que `range a b` renvoie la liste `[a; a + 1; ...; b - 1]`.
5. Écrire une fonction `prive_de : int list -> int list` telle que l'appel `prive_de u x` renvoie une liste ayant les mêmes éléments que `u`, sauf que la première occurrence de `x` a été supprimée :

```
# prive_de [1; 3; 5; 7] 3;;
- : int list = [1; 5; 7]
```

6. Écrire une fonction `max_liste : int list -> int` calculant le maximum d'une liste d'entiers.
7. Écrire une fonction `eval : int array array -> int list -> int` calculant récursivement $eval(M, S)$. Le premier argument de `eval` est la matrice M, le deuxième l'ensemble S représenté par la liste de ses éléments.
8. Vérifier qu'elle calcule le résultat correct pour M_1.

3.2 Analyse de la solution récursive naïve

1. On note $\varphi(k)$ le nombre total d'appels effectués lors du calcul de $eval(M, S)$ quand $|S| = k$. Justifier que $\varphi(k) \geqslant k!$.
2. Combien de valeurs *différentes* S prend-il lors de ces appels ?

4 Solution en programmation dynamique

Comme nous venons de le voir, il n'y a que 2^n valeurs de $eval(M, S)$ à calculer pour une matrice M donnée de taille n. En mémoïsant ces appels, on peut donc espérer réduire la complexité à « un peu plus » de 2^n (peut-être 2^n, peut-être $n2^n$, peut-être $n^2 2^n \ldots$). Or, autant 15! est « vraiment grand », autant 2^{15} (qui vaut 32768) ne l'est pas : pour *notre problème précis*, ça devrait marcher !

M étant fixée, on va utiliser un dictionnaire dans lequel on va stocker les associations $(S, eval(M, S))$ au fur et à mesure du calcul. Le principe de calcul de $eval(M, S)$ sera donc le suivant :

— Si la table globale contient déjà une entrée (S, v) on sait que v est la valeur cherchée pour $eval(M, S)$: on la renvoie 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 $eval(M, S \setminus \{i\})$ (ces appels récursifs ajouteront eux-mêmes leur résultat dans la table). On peut alors calculer la valeur v de $eval(M, S)$. On note (S, v) dans la table et on retourne v.

Pour réaliser le dictionnaire, on va utiliser les tables de hachage qui sont prédéfinies en OCaml. 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.

1. Écrire une fonction `eval_mem` et calculer $s(M_2)$. Il faudra une fonction principale, non récursive, dont le travail sera simplement de créer une table vide et d'appeler la fonction auxiliaire. Cette fonction auxiliaire sera, elle, récursive, et procédera comme décrit ci-dessus.
   Ici, il est très important que les listes correspondant aux ensembles S soient triées, sinon un même ensemble pourrait être représenté par deux listes différentes et on ne retrouverait pas nos données dans la table.
2. **Question délicate que je vous invite à sauter dans un premier temps.** En supposant que l'accès à un élément de la table se fasse en temps constant, quelle est la complexité de la fonction `eval_mem` ?

5 Optimisation

On peut représenter le sous-ensembles S de $[\![0, n[\![$ par l'entier $\sum_{i \in S} 2^i$. Ainsi, les parties de $[\![0, n[\![$ 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 de l'algorithme dynamique précédent en utilisant cette structure de données ; le plus simple est d'utiliser une approche *top-down*, mais du *bottom-up* est également possible (et, bien programmé, est même encore un peu plus rapide).

Il existe en fait une solution en temps polynomial à ce problème, mais l'algorithme est plus compliqué et ne se comprend bien qu'avec un peu de théorie des graphes.