L'ALGORITHME OUBLIÉ

La découverte d'un nouvel algorithme de comptage, reposant sur des principes élémentaires mais pourtant passé inaperçu jusqu'à récemment, étonne les spécialistes.

L'AUTEUR

JEAN-PAUL DELAHAYE
professeur émérite
à l'université de Lille
et chercheur au
laboratoire Cristal
(Centre de recherche
en informatique, signal
et automatique de Lille)

Il est difficile d'imaginer que les chercheuses et les chercheurs, qui sont nombreux et dont l'intelligence collective ne fait aucun doute, aient pu passer à côté d'un résultat simple au sujet d'un problème important, ou d'une méthode élémentaire de calcul. Poser une nouvelle pierre sur le mur des savoirs qui s'accumulent ne peut, pense-t-on, que demander un gros travail, et ce qu'on ajoutera revêtira forcément une certaine complexité. Eh bien ce n'est pas toujours vrai, et une récente découverte dans le domaine des algorithmes les plus fondamentaux vient de le prouver : tout ce qui est simple n'est pas nécessairement connu !

Parmi les questions les plus élémentaires posées en informatique mathématique se trouve celle du décompte du nombre d'éléments distincts d'une suite de données et des algorithmes permettant d'effectuer ce décompte. Par exemple, dans la suite de lettres $S = (a, b, c, a, a, d, b, c, e, c)$, qui est de longueur 10, il y a exactement 5 éléments distincts : a, b, c, d et e. Le problème de leur décompte est ici rapidement résolu. De manière générale, quand la suite S à laquelle on s'intéresse n'est pas très longue, ce problème sera facile à résoudre. Mais dans un flux continu de nombreuses données, déterminer le nombre d'éléments distincts devient délicat à cause de l'impossibilité de stocker tout le passé des données, qui arrivent trop vite et en trop grand nombre.

Des méthodes existaient depuis longtemps pour aborder ce problème, mais toutes étaient assez compliquées et mettaient en œuvre des concepts informatiques délicats, comme celui de fonction universelle de hachage. Trois chercheurs viennent cependant d'effectuer un progrès inattendu : Sourav Chakraborty, de l'Institut indien de statistiques, à Calcutta, Vinodchandran Variyam, de l'École d'informatique de l'université du Nebraska, aux États-Unis, et Kuldeep Meel, professeur au département d'informatique de l'université de Toronto, au Canada.

L'algorithme proposé a provoqué l'étonnement des meilleurs spécialistes, en particulier celui du célèbre mathématicien et informaticien Donald Knuth, auteur du magistral traité *The Art of Computer Programming* et également créateur du traitement de texte TeX, utilisé partout dans le monde pour éditer des documents scientifiques. Venant compléter et préciser le travail des trois découvreurs, Donald Knuth a lui-même proposé un article dans lequel il assure : « L'algorithme n'est pas seulement intéressant, il est aussi extrêmement simple. Il est d'ailleurs parfaitement adapté pour enseigner aux étudiants qui découvrent les bases de l'informatique. Depuis que je l'ai vu, il y a quelques jours, je n'ai pas pu m'empêcher d'essayer d'en expliquer les idées à presque toutes les personnes que j'ai rencontrées. » Nous adopterons la proposition de Donald Knuth de nommer le nouveau procédé de calcul « algorithme CVM », en référence aux initiales de ses découvreurs.

Commençons par le cas évident qui se présente quand la suite des données $S = (x_1, x_2, \ldots, x_n)$

Jean-Paul Delahaye a également publié : **Au-delà du Bitcoin** (Dunod, 2022).

n'est pas très longue et qu'on peut en mémoriser sans difficulté tous les éléments. Pour compter le nombre d'éléments distincts de S, on les fait défiler les uns après les autres en plaçant dans un espace de stockage E certains d'entre eux. On commence avec un espace E vide, puis on observe l'un après l'autre les éléments de la liste S, et pour chaque nouvel élément examiné x on teste si x est déjà présent dans E ; s'il n'y est pas, alors on l'ajoute à E ; s'il y est déjà, on ne fait rien. Une fois la liste S égrainée, le nombre k d'éléments présents dans E constitue la réponse R recherchée (il est clair qu'il s'agit bien du nombre d'éléments distincts de S).

Prenons l'exemple $S=(a,b,c,a,a,d,b,c,e,c)$. L'algorithme effectue les 10 étapes suivantes :

```
DEBUT
E=∅;
x=a, E=(a);
x=b, E=(a, b);
x=c, E=(a, b, c);
x=a, E=(a, b, c);
x=a, E=(a, b, c);
x=d, E=(a, b, c, d);
x=b, E=(a, b, c, d);
x=c, E=(a, b, c, d);
x=e, E=(a, b, c, d, e);
x=c, E=(a, b, c, d, e);
k=5
Reponse: R=5
FIN
```

Avant de décrire l'algorithme découvert récemment et pour mieux en comprendre le principe, nous allons en formuler une version simplifiée, qui sera plus facile à justifier et aidera à bien saisir les propriétés de l'algorithme CVM décrit plus loin.

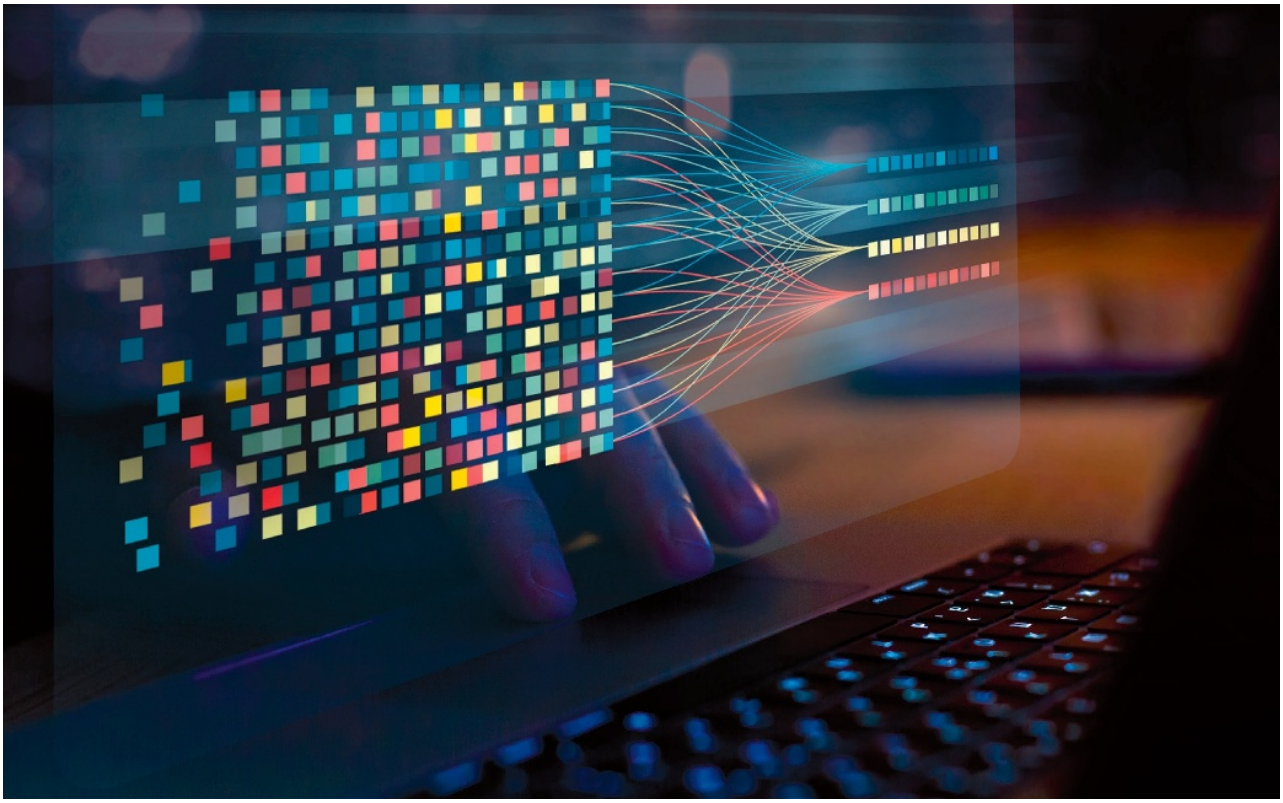
UN PREMIER ALGORITHME PROBABILISTE

Imaginons que nous voulions compter le nombre d'éléments distincts d'une suite S de 100 éléments, mais que nous ne disposions que d'un espace de stockage E dont la taille permet de stocker au maximum 50 éléments. Voici comment procéder.

L'impossibilité d'utiliser l'algorithme de base incite à proposer une méthode approchée, de nature probabiliste. Nous supposerons pour cela que nous disposons d'un système de tirage au sort équivalent à une pièce de monnaie équilibrée qu'on lance et qui donne PILE ou FACE avec la même probabilité, 1/2.

La première version de la nouvelle méthode consiste à remplir l'espace de stockage E au fur et à mesure que les éléments de la suite S défilent, en économisant toutefois la mémoire de la façon suivante. À chaque fois qu'un nouvel élément x arrive, on regarde s'il est déjà dans E, puis :

Trois chercheurs viennent de découvrir une méthode probabiliste simple pour compter le nombre d'éléments distincts dans une liste.



© Song_about_summer/Shutterstock

1 L'ALGORITHME CVM AVEC PROBABILITÉ FIXE

L'algorithme ci-dessous est une méthode probabiliste simple permettant d'économiser de la mémoire pour compter les éléments distincts d'une suite *S*.

DÉBUT
Initialisation : $S = (x_1, x_2, ..., x_n)$; $E = \emptyset$
Pour *i* allant de 1 à *n* faire :
– Si x_i n'est pas dans *E*, tirer à PILE ou FACE et faire :
– Si PILE tombe, ajouter x_i à *E*
– Si FACE tombe, ne rien faire
– Si x_i est déjà dans *E*, le retirer de *E*, puis tirer à PILE ou FACE et faire :
– Si PILE tombe, ajouter x_i à *E*
– Si FACE tombe, ne rien faire
Renvoyer le résultat :
$R = 2 \times$ (nombre d'éléments de *E*)
FIN

Exemple
Soit la suite à 18 éléments *S* = (*a*, *e*, *c*, *b*, *e*, *f*, *g*, *a*, *b*, *c*, *h*, *i*, *j*, *a*, *b*, *a*, *g*, *g*).

```
DEBUT
E=∅;
x=a, PILE, E=(a);
x=e, PILE, E=(a, e);
x=c, FACE, E=(a, e);
x=b, PILE, E=(a, e, b);
x=e, E=(a, b), FACE, E=(a, b);
x=f, FACE, E=(a, b);
x=g, PILE, E=(a, b, g);
x=a, E=(b, g), PILE, E=(b, g, a);
x=b, E=(g, a), FACE, E=(g, a);
x=c, PILE, E=(g, a, c);
x=h, FACE, E=(g, a, c);
x=i, PILE, E=(g, a, c, i);
x=j, FACE, E=(g, a, c, i);
x=a, E=(g, c, i), FACE, E=(g, c, i);
x=b, FACE, E=(g, c, i);
x=a, PILE, E=(g, c, i, a);
x=g, E=(c, i, a), FACE, E=(c, i, a);
x=g, PILE, E=(c, i, a, g);
R=2×4
FIN
```

Le *E* final comporte 4 éléments, la réponse renvoyée par l'algorithme est donc 8. Cette réponse est plutôt satisfaisante, puisque *S* possède 9 éléments distincts : (*a*, *e*, *c*, *b*, *f*, *g*, *h*, *i*, *j*).
En écrivant le programme et en menant l'essai dix fois de suite, voici les réponses que j'ai obtenues : 10, 10, 8, 10, 10, 6, 10, 12, 12 et 6, ce qui en moyenne donne la réponse 9,4. En refaisant l'essai cent fois, j'ai obtenu une moyenne de 8,9, et en faisant mille essais une moyenne de 9,02.
En prenant $p = 1/10$ à la place de $p = 1/2$ (on ne garde qu'une fois sur dix un élément qui passe) et en opérant mille essais, j'ai obtenu une moyenne de 8,45. On constate, sans surprise, que l'algorithme avec *p* petit fait économiser plus de mémoire (environ 90 %), mais que cela se fait au détriment de la précision du résultat.

– Si x n'est pas dans E, on tire à PILE ou FACE ; si c'est PILE qui tombe on ajoute x à E ; si c'est FACE on ne change pas E ;

– Si x est déjà dans E, on le retire de E puis on tire à PILE ou FACE ; si c'est PILE qui tombe on remet x dans E ; si c'est FACE on ne change pas E (qui vient donc de perdre x).

Puisqu'on ne met dans E qu'environ un élément de S sur deux, on s'en sortira avec un espace E dont la mémoire est de taille approximativement deux fois moindre que le nombre d'éléments différents de S. Le sens exact des termes « environ » et « approximativement », dans la phrase précédente, sera précisé dans la suite de l'article.

Une fois tous les éléments de S égrenés, on compte le nombre k d'éléments de E et on propose la réponse $R = 2k$. Cette réponse est une approximation du résultat car, en procédant comme on l'a fait, chacun des éléments distincts de S à une chance sur deux de se trouver dans le E final, qui contient donc environ deux fois moins d'éléments qu'il n'y a d'éléments distincts dans S. Précisons le raisonnement assurant que « chacun des éléments distincts de S a une chance sur deux de se trouver dans le E final » :

– Si l'élément x n'apparaît qu'une fois dans S, alors quand il se présente il est ajouté à E avec une probabilité 1/2, puis la situation ne change plus. L'élément x est donc dans le E final avec une probabilité 1/2.

– Si l'élément x apparaît plusieurs fois dans S, seule sa dernière apparition est importante, car s'il est déjà présent dans E à l'instant de sa dernière apparition, il en est d'abord retiré, puis un tirage à PILE ou FACE détermine s'il y est remis ou non. Un tel élément x a donc, lui aussi, exactement une chance sur deux de se trouver dans le E final.

Remarquons que si, pour chaque nouvel élément de S, on avait choisi de l'ajouter à E avec la probabilité $p = 1/10$ au lieu de $p = 1/2$, le même raisonnement nous indiquerait qu'en multipliant le k final par 10, on obtiendrait une approximation de la valeur recherchée. Bien sûr, 1/10 peut être remplacé par n'importe quelle probabilité p entre 0 et 1, et la réponse $R = k/p$ constituera une approximation de la valeur recherchée.

Il semble donc qu'on dispose d'une méthode qui, même quand on a peu de mémoire, permet de compter les éléments distincts d'une suite S, même extrêmement longue. Il suffit de procéder avec p assez petit, car plus p est petit moins l'on utilisera de mémoire en faisant défiler les éléments de S. On comprend toutefois intuitivement que plus p est petit, plus l'approximation finale risque d'être imprécise.

L'algorithme CVM consiste justement à ne pas choisir un p petit au départ, mais à le faire

varier pendant le déroulement du calcul ce qui permet de ne prendre p petit que quand cela est véritablement nécessaire. Voici donc l'algorithme de comptage utilisant cette idée d'un p ajusté au mieux, qui a surpris la communauté informatique.

L'ALGORITHME CVM

On cherche à évaluer le nombre d'éléments distincts d'une longue suite $S = (x_1, x_2, ..., x_n)$. On ne dispose que d'un espace de stockage E de taille m fixée une fois pour toutes au départ. Cette taille m détermine donc le nombre maximum d'éléments de la liste S que l'algorithme pourra stocker temporairement en mémoire. On suppose que m est plus petit que le nombre total n d'éléments de la liste S. L'algorithme procédera donc de manière probabiliste, en ne proposant qu'une valeur approchée du nombre d'éléments distincts de S.

On commence par fixer un paramètre $p = 1$. Ce p sera amené à changer de valeur au cours des calculs. Puis, comme dans le premier algorithme décrit ci-dessus, on fait défiler un à un les éléments x de S. Pour chacun de ces éléments x, si x n'est pas déjà dans E, il y est ajouté avec la probabilité p. S'il est déjà présent, l'algorithme choisit alors de le conserver dans E avec une probabilité p (autrement dit, x est retiré de E, puis on ne le remet dans E qu'avec la probabilité p). Lorsque l'espace de stockage E est rempli car on y a déposé m éléments (le nombre maximum), on modifie p en le divisant par 2, et on procède à une « purge » de E, consistant à retirer chaque élément présent dans E avec une probabilité 1/2. On continue le processus avec la nouvelle valeur de p jusqu'à ce que E soit de nouveau rempli. Quand cela se produit, on divise de nouveau p par 2 et on effectue une nouvelle purge. On poursuit ces cycles jusqu'à ce que tous les éléments de la suite S aient été égrenés.

Si, par malchance, une purge ne permet de retirer aucun élément de E, l'algorithme s'arrête et répond « erreur ». C'est une éventualité qu'une parfaite rigueur oblige à considérer, car cette situation ne peut pas être exclue. Le cas est cependant peu probable dès que m est assez grand, puisqu'il signifie qu'aucun des m éléments présents quand on a purgé n'a été retiré, ce qui ne se produit qu'avec une probabilité $1/2^m$. En clair, on ne peut pas omettre cette éventualité, mais dès que m dépasse 20 ou 30, le risque qu'elle se présente peut être négligé. Les lecteurs courageux qui désireraient des informations plus fines sur la

2 L'ALGORITHME CVM AVEC PROBABILITÉ VARIABLE

Le nouvel algorithme CMV, plus efficace que celui présenté dans l'encadré 1, peut être décrit comme suit.

DÉBUT
Initialisation : $S = (x_1, x_2, ..., x_n)$; $p = 1$; $E = \emptyset$; m
Pour i allant de 1 à n faire :
– Si x_i est dans E, retirer x_i de E
– Avec une probabilité p, ajouter x_i à E
– Si le nombre d'éléments de E atteint m, alors :
– Changer p en $p/2$
– Retirer chaque élément de E avec une probabilité 1/2 ;
– Si le nombre d'éléments de E est encore m alors :
– Renvoyer le résultat : « erreur »
Renvoyer le résultat :
R = (nombre d'éléments de E) / p
FIN

Les lecteurs amateurs de programmation adapteront sans mal dans leur langage préféré cette structure générale.

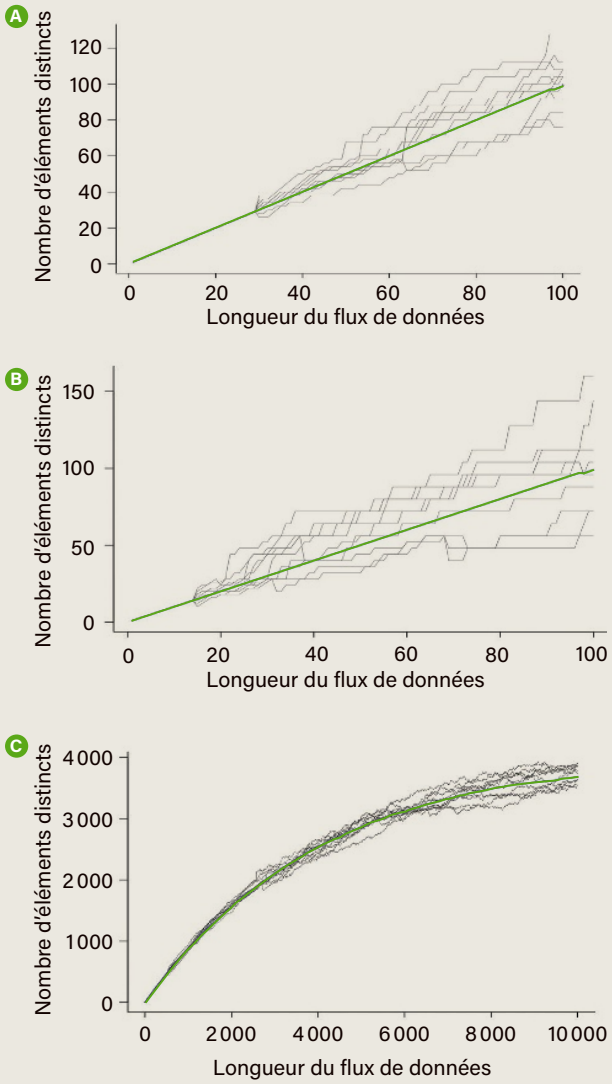
Exemple
Reprenons la suite à 18 éléments de l'encadré 1 :
$S = (a, e, c, b, e, f, g, a, b, c, h, i, j, a, b, a, g, g)$.
Pour les tirages au sort, quand $p = 1/2$ on imagine qu'on tire à PILE ou FACE, et que le x testé est mis dans E si on obtient PILE. Quand $p = 1/4$ on tire à PILE ou FACE deux fois de suite, et le x testé est mis dans E si on obtient deux fois PILE, etc.

```
DEBUT
m=4; E=∅; p=1;
x=a, E=(a);
x=e, E=(a, e);
x=c, E=(a, e, c);
x=b, E=(a, e, c, b);
Purge: PILE-FACE-PILE-FACE,
E=(a, c), p=1/2;
x=e, PILE, E=(a, c, e);
x=f, FACE, E=(a, c, e);
x=g, FACE, E=(a, c, e);
x=a, E=(c, e), FACE, E=(c, e);
x=b, PILE, E=(c, e, b);
x=c, E=(e, b), PILE, E=(e, b, c);
x=h, PILE, E=(e, b, c, h);
Purge: PILE-PILE-FACE-FACE,
E=(e, b), p=1/4;
x=i, FACE-PILE, E=(e, b);
x=j, FACE-FACE, E=(e, b);
x=a, PILE-PILE, E=(e, b, a);
x=b, E=(e, a), PILE-FACE, E=(e, a);
x=a, E=(e), PILE-PILE, E=(e, a);
x=g, PILE-FACE, E=(e, a);
x=g, FACE-FACE, E=(e, a);
R=2/(1/4)
FIN
```

Le E final comporte deux éléments, la réponse renvoyée par l'algorithme est donc 2 / p = 8.
En exécutant l'algorithme 10 fois de suite, j'ai obtenu successivement les résultats 8, 12, 4, 4, 12, 8, 4, 16, 24 et 4, ce qui donne une moyenne de 9,6. En refaisant l'essai cent fois, j'ai obtenu une moyenne de 9,5. En faisant l'essai mille fois, la moyenne était de 8,73.
Les résultats obtenus, conformément à ce qu'annoncent les résultats théoriques, convergent vers la bonne réponse, qui est 9.
Le déroulement de l'algorithme CVM est assez semblable à ce qu'on obtient avec certains jeux des constructions permettant de faire rouler des billes sur des circuits complexes. Chaque bille est une donnée, les billes passent dans divers dispositifs qui les sélectionnent, avec parfois des accumulations en certains points du circuit conduisant à des opérations équivalentes à des purges. Les billes qui arrivent au bout du parcours produisent le résultat.

3 OBSERVATIONS EXPÉRIMENTALES

Le statisticien Kenneth Tay a mené des expériences sur l'algorithme CVM. Il les a présentées dans un billet de blog intitulé « A simple probabilistic algorithm for estimating the number of distinct elements in a data stream », sur son site « Statistical Odds & Ends ». Ces expériences illustrent clairement les propriétés du nouvel algorithme.
Faisant défiler dix fois de suite le même flux de données de longueur 100, en considérant un espace de stockage *E* de taille 30, et en comparant à chaque instant la valeur réelle du nombre d'éléments distincts *(courbes vertes ci-dessous)*, et la valeur estimée par l'algorithme CVM *(courbes grises ci-dessous)*, le chercheur obtient les dix courbes de la figure A. Les courbes diffèrent les unes des autres car les tirages au sort ne sont pas toujours les mêmes. Remarquez que, bien sûr, tant que moins de 30 éléments sont passés, l'évaluation est parfaitement correcte.
Avec une mémoire disponible de 15 éléments les résultats sont moins satisfaisants, comme le montre la figure B.
Au contraire, pour un flux de données plus long, de 10 000 éléments, et une mémoire disponible de taille 500, on constate que l'évaluation est très satisfaisante et ne s'écarte jamais de plus de 10 % du résultat exact, comme le montre la figure C.



manière dont ce cas « erreur » est abordé en pratique pourront consulter l'article de Donald Knuth cité dans la bibliographie.

Lorsque tous les éléments de la liste S ont défilé, l'estimation R proposée du nombre d'éléments distincts de S est le nombre k d'éléments finalement présents dans la mémoire de travail E divisé par la dernière valeur du paramètre utilisé dans l'algorithme : $R=k/p$.

UNE APPROXIMATION SATISFAISANTE

Pour comprendre que la réponse proposée est une approximation satisfaisante du résultat attendu, il faut comprendre que chaque élément distinct x présent dans S se retrouve dans l'espace de travail final E avec la probabilité p, où p est la valeur finale du paramètre au moment de l'arrêt. Il y a en effet deux cas à considérer.

Si x n'est présent qu'une fois dans S, quand il est traité par l'algorithme, il n'est gardé qu'avec la probabilité p, où p est la valeur du paramètre à l'instant où x est traité. Ensuite, à chaque purge, x est enlevé de E avec une probabilité 1/2, et au même moment la valeur du paramètre est divisée par deux. En définitive, x sera donc dans la mémoire finale E avec la probabilité p finale, car p a été divisé par deux autant de fois qu'il y a eu de purges depuis le traitement de x par l'algorithme.

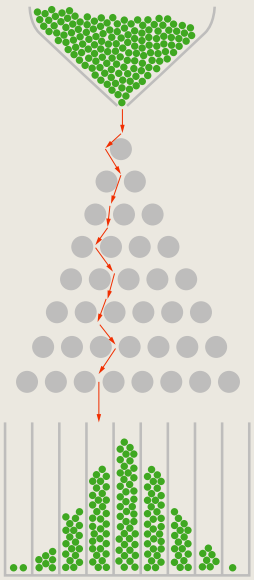
Si x est présent plusieurs fois dans S, on est en fait ramené au premier cas. En effet, seule sa dernière apparition dans S compte vraiment, puisqu'à chaque nouvelle apparition de x lors du parcours des éléments de S, on commence par retirer x de E avant de dérouler les étapes décrites ci-dessus.

Il résulte bien de ceci que, dans le E final il y a environ $k=r\times p$ éléments, où p est la valeur finale du paramètre et r le nombre d'éléments distincts présents dans S. En renvoyant comme résultat $R=k/p$, l'algorithme CVM renvoie donc bien une approximation de r. On aimerait cependant mieux quantifier la qualité de cette approximation : en pratique, ce $R=k/p$ sera-t-il toujours proche de la réponse exacte, qui est un nombre entier, ou pourra-t-il s'en éloigner beaucoup par malchance ? Les trois découvreurs de l'algorithme CVM ont proposé une réponse très précise à cette question, réponse dont nous détaillons ci-dessous la formulation.

Quand ε et δ sont de petits nombres positifs, on dit qu'un algorithme probabiliste du type CVM « fournit une approximation à (ε, δ) près de la bonne réponse B » si la réponse obtenue R s'écarte en proportion de la bonne réponse B de plus de ε avec une probabilité inférieure à δ. Autrement dit, on affirme cela lorsque la probabilité que « $R/B \geq (1+\varepsilon)$ ou $R/B \leq (1-\varepsilon)$ » est inférieure à δ, ou encore que la probabilité que « $(1-\varepsilon)B<R<(1+\varepsilon)B$ » est supérieure

4 LES ALGORITHMES PROBABILISTES

Même pour des problèmes dont les réponses sont parfaitement déterminées et ne dépendent en rien du hasard, il est parfois utile, en algorithmique, d'utiliser des tirages aléatoires – comme c'est le cas dans l'algorithme CVM. Le tout premier exemple de cette situation est ce qu'on nomme la « planche de Galton », en hommage à l'anthropologue et inventeur britannique Francis Galton. Il s'agit d'un dispositif physique, proposé en 1889, dont on déduit facilement un algorithme. La planche permet d'obtenir des approximations des coefficients de la loi binomiale en utilisant des billes qu'on fait tomber sur des clous régulièrement disposés sur une planche *(voir le dessin ci-contre)* et qui viennent s'entasser dans des petites boîtes en bas de la planche. Comme pour les données retenues ou non dans le fonctionnement l'algorithme CVM, à chaque fois qu'une bille frappe un clou, elle va vers la droite ou vers la gauche avec une probabilité 1/2. Une fois toutes les billes descendues et réparties dans les boîtes réceptrices, en les comptant – comme on compte le nombre d'éléments de *E* à la fin du fonctionnement de l'algorithme CVM –, on obtient une approximation des coefficients de la loi binomiale. Pour obtenir des approximations des coefficients de la *n*-ième ligne du triangle de Pascal avec une planche de Galton, il faut utiliser 2^n billes et disposer de *n* + 1 boîtes.



ou égale à $(1-\delta)$. Pour $\varepsilon=10\%$ et $\delta=1\%$, cela donne par exemple : « La réponse R donnée par l'algorithme est une approximation à (10%, 1%) près de la bonne réponse B si et seulement si R ne s'écarte de plus de 10% de B qu'avec une probabilité inférieure à 1%. » On comprend bien que plus ε et δ sont petits, plus cela assure que la réponse R sera proche de B et qu'elle ne s'en écartera qu'avec une faible probabilité.

Le résultat concernant l'algorithme CVM utilise cette notion d'approximation à (ε, δ) près. Il indique que, pour une suite S de longueur n, si la taille m de l'espace de stockage E vérifie $m \geq [(12/\varepsilon^2) \times log(8L/\delta)]$, alors l'algorithme CVM produit une réponse R qui est une approximation à (ε, δ) près du nombre d'éléments distincts de S. La notation $[x]$ désigne la partie entière par excès de x, c'est-à-dire le plus petit entier supérieur ou égal à x : $[2{,}718]=3$; $[6{,}1]=7$.

Ainsi, pour une liste S d'un million d'éléments, on peut obtenir une approximation à (10%, 1%) près du nombre d'éléments distincts de S si l'on dispose d'un espace de stockage E de taille supérieure ou égale à $m=[(12/(1/100)) \times log(8000000/0{,}01)]=10684$. Autrement dit, dans un tel cas, avec environ 90 fois moins de mémoire que la taille de S, on arrive à une bonne évaluation du nombre d'éléments distincts. L'algorithme CVM est donc très puissant, puisqu'avec beaucoup moins d'espace que la longueur de S, on s'en tire assez bien. Mais sans surprise, plus l'on met à sa disposition un espace de stockage conséquent, meilleure sera la réponse qu'il donnera.

Très concrètement, considérons le texte de la pièce *Hamlet*, de William Shakespeare, qui comporte en tout 30557 mots, mais seulement 3967 mots distincts. En utilisant une mémoire de 100 mots, en répétant l'expérience 5 fois de suite et en prenant la moyenne des résultats obtenus, l'équipe CMV a estimé ce nombre de mots distincts à 3955, ce qui est assez bon (moins de 1% d'erreur). Avec une mémoire de travail de 1000 mots le résultat est, logiquement, encore meilleur : 3964.

Depuis que l'algorithme CVM s'est répandu, des variantes et des perfectionnements très subtils ont été proposés. Il ne fait aucun doute que, dans tous les livres et cours d'algorithmique, on réservera maintenant une place pour présenter et expliquer ce nouvel outil qui, c'est aussi certain, sera largement utilisé en pratique. ■

BIBLIOGRAPHIE

K. Tay, A simple probabilistic algorithm for estimating the number of distinct elements in a data stream, *billet de blog « Statistical Odds & Ends »*, 2024.

D. Knuth, The CVM algorithm for estimating distinct elements in streams, *Stanford Computer Science Department*, 2023.

M. Nandi et S. Paul, Analysis of Knuth's Sampling Algorithm D and D', *arXiv preprint*, 2023.

S. Chakraborty *et al.*, Distinct elements in streams : An algorithm for the (text) book, *arXiv preprint*, 2023.

D. Kane *et al.*, An optimal algorithm for the distinct elements problem, *Proceedings of PODS*, 2010.

M. Durand et P. Flajolet, Loglog counting of large cardinalities, *Algorithms-ESA 2003*, 2003.