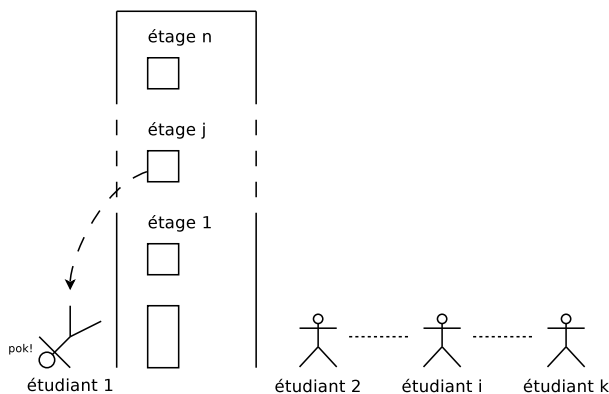


TD n°8

1 Ce à quoi vous avez échappé

1.1 Le grand saut

Le problème est de déterminer à partir de quel étage d'un immeuble, sauter par une fenêtre est fatal. Vous êtes dans un immeuble à n étages (numérotés de 1 à n) et vous disposez de k étudiants. Les étudiants sont classés par notes de partiel croissantes. Il n'y a qu'une opération possible pour tester si la hauteur d'un étage est fatale : faire sauter le premier étudiant de la liste par la fenêtre. S'il survit, vous pouvez le réutiliser ensuite (évidemment, l'étudiant survivant reprend sa place initiale dans la liste triée), sinon vous ne pouvez plus.



Vous devez proposer un algorithme pour trouver la hauteur à partir de laquelle un saut est fatal (renvoyer $n + 1$ si on survit encore en sautant du n -ème étage) en faisant le minimum de sauts.

Question 1.1 Si $k \geq \lceil \log_2(n) \rceil$, proposer un algorithme en $\mathcal{O}(\log_2(n))$ sauts.

Solution : La complexité en $\mathcal{O}(\log_2(n))$ qui est indiquée nous aiguille vers une dichotomie. En effet, en supposant que l'on a $k \geq \lceil \log_2(n) \rceil$, on obtient le résultat sur les étages de i à j en jetant un étudiant depuis le $n^{\text{ème}}$ étage, où $n = j - i/2$, puis en itérant le procédé sur les étages de i à $n - 1$ si la chute à été fatale, et sur les étages de n à j dans le cas contraire. La méthode dichotomique nous garantit alors que l'on obtient le bon résultat (lorsqu'il ne reste plus qu'un seul étage, c'est-à-dire lorsque l'on calcule pour les étages de $i = j$), et ce avec une complexité logarithmique dans le pire des cas. \square

Question 1.2 Si $k < \lceil \log_2(n) \rceil$, proposer un algorithme en $\mathcal{O}(k + \frac{n}{2^{k-1}})$ sauts.

Solution : Puisqu'ici on ne dispose que de $k < \lceil \log_2(n) \rceil$ étudiants, on ne peut pas appliquer directement la méthode dichotomique proposée précédemment. Cependant, on va remédier au problème de manière simple en appliquant une recherche dichotomique avec $k - 1$ étudiants, de manière à délimiter un intervalle d'étages dans lequel se trouve l'étage recherché. On se sert

alors du dernier étudiant restant pour parcourir l'intervalle de façon linéaire, donc en le jetant de chaque étage en partant du plus bas de l'intervalle, jusqu'au plus haut. Après avoir jeté les $k - 1$ premiers étudiants, si l'on n'a pas encore trouvé le bon étage, il reste exactement $n/2^{k-1}$ étages dans l'intervalle de recherche, d'où une complexité dans le pire des cas en $O(k + n/2^{k-1})$ sauts. \square

Question 1.3 Si $k = 2$, proposer un algorithme en $O(\sqrt{n})$ sauts.

Solution : Dans le cas particulier où l'on a $k = 2$, on ne veut pas avoir à tester chaque étage de façon linéaire, c'est pourquoi on va reprendre à notre compte les idées précédentes, et notamment celle qui consiste à délimiter un intervalle de recherche. Nous découpons donc l'ensemble des étages en "tranches" de \sqrt{n} étages, avant de jeter le premier étudiant de chacun des étages de début de tranche. Lorsque l'étudiant y laisse sa peau, on se ramène au dernier étage n testé qui ne soit pas fatal, et on n'a plus qu'à parcourir de manière linéaire l'intervalle allant de l'étage $n + 1$ à l'étage fatal trouvé précédemment. On a ainsi deux séries d'essais en $O(\sqrt{n})$, et donc une complexité finale dans le pire des cas également en $O(\sqrt{n})$ sauts. \square

1.2 Chercher la star

Dans un groupe de n personnes (numérotées de 1 à n pour les distinguer), une *star* est quelqu'un qui ne connaît personne mais que tous les autres connaissent. Pour démasquer une star, s'il en existe une, vous avez juste le droit de poser des questions, à n'importe quel individu i du groupe, du type "est-ce que vous connaissez j ?" (noté " $i \rightarrow j$?"), on suppose que les individus répondent la vérité. On veut un algorithme qui trouve une star, s'il en existe, ou sinon qui garantit qu'il n'y a pas de star dans le groupe, en posant le moins de questions possibles.

Question 1.4 Combien peut-il y avoir de stars dans le groupe ?

Solution : Il ne peut y avoir qu'une seule star dans le groupe, puisque s'il y en a une, elle ne connaît personne, et donc tous les autres sont inconnus d'au moins une personne, et ne peuvent donc être eux aussi des stars. \square

Question 1.5 Ecrire le meilleur algorithme que vous pouvez et donner sa complexité en nombre de questions (on peut y arriver en $O(n)$ questions). **Solution :** Lorsqu'on effectue le test " $i \rightarrow j$?

", c'est-à-dire lorsque l'on cherche à savoir si la personne i connaît la personne j , on obtient le résultat suivant :

- si oui, alors i n'est pas une star, mais j en est potentiellement une.
- si non, alors j n'est pas une star, mais i en est potentiellement une.

L'algorithme consiste alors à parcourir le tableau des personnes une fois, en gardant en mémoire à chaque instant la personne i qui est jusqu'ici reconnue par tous, tandis qu'au $j^{\text{ème}}$ test, toutes les autres personnes, dont les indices sont les $k < j$ (avec $k \neq i$, bien sûr) ne peuvent pas être des stars. Cet algorithme s'écrit donc de la manière suivante :

début

```

i ← 1 et j ← 2;
tant que j ≤ n faire
  | si "j → i" alors j ← j + 1;
  | sinon i ← j et j ← j + 1;
istar ← vrai et j ← 1;
tant que j < i et istar faire
  | si "j → i" alors j ← j + 1;
  | sinon istar ← faux;
k ← 1;
tant que k ≤ n et istar faire
  | si k ≠ i et "i → k" alors istar ← faux;
  | sinon k ← k + 1;
si istar alors retourner "i est la star";
sinon retourner "il n'y a pas de star";

```

fin

□

Question 1.6 Donner une borne inférieure sur la complexité (en nombre de questions) de tout algorithme résolvant le problème. ((*Difficile*) prouver que la meilleure borne inférieure pour ce problème est $3n - \lfloor \log_2(n) \rfloor - 3$).

Solution : En observant notre algorithme, on constate que l'on peut utiliser comme borne inférieure pour le nombre de questions la valeur $3n - 3$, puisque l'on fait ici trois boucles sur $n - 1$ personnes. Pour ce qui est de la borne inférieure optimale, il s'agit d'une question difficile, dont on n'explicitera pas la solution ici. □