Rencontre au milieu

Lycée Thiers

Crédit

- Cette page sur Quora

Position du problème

- Considérons un ensemble E de n nombres distincts et un entier S.

Position du problème

- Considérons un ensemble E de n nombres distincts et un entier S.
- On cherche la plus grande somme d'éléments de E dont la valeur est plus petite que S.

Position du problème

- Considérons un ensemble E de n nombres distincts et un entier S.
- On cherche la plus grande somme d'éléments de E dont la valeur est plus petite que S.
- On peut faire une recherche en force brute. Il s'agit de déterminer la somme pour tous les sous-ensembles de E et de la comparer avec S puis de prendre la plus grande somme plus petite que S.

Position du problème

- Considérons un ensemble E de n nombres distincts et un entier S.
- On cherche la plus grande somme d'éléments de E dont la valeur est plus petite que S.
- On peut faire une recherche en force brute. Il s'agit de déterminer la somme pour tous les sous-ensembles de E et de la comparer avec S puis de prendre la plus grande somme plus petite que S.
- Malheureusement, il y a 2^n sous-ensemble de E, donc la complexité de cette approche serait en $O(2^n)$ au moins (c'est sans compter le coût de la somme elle-même).

Présentation

- *Meet-in-the-middle* est une technique de recherche appliquée lorsque l'entrée est petite (par exemple 40 nombres) mais pas assez petite pour que la recherche en force brute soit envisageable (2^{40} est quand même assez gros).

Présentation

- *Meet-in-the-middle* est une technique de recherche appliquée lorsque l'entrée est petite (par exemple 40 nombres) mais pas assez petite pour que la recherche en force brute soit envisageable (2^{40} est quand même assez gros).
- Comme *diviser pour régner*, le problème est divisé en 2 sous-problèmes. Cependant ici, le travail ne se fait pas récursivement. On travaille sur deux moitiés du tableau, on ne divise pas le tableau davantage.

Algorithme (1)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Diviser l'ensemble des entiers en deux sous-ensemble A, B. A contient $n/2$ éléments et B les autres.

Algorithme (1)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Diviser l'ensemble des entiers en deux sous-ensemble A, B. A contient $n/2$ éléments et B les autres.
- Chercher toutes les sommes possibles d'éléments de A et mettre le résultat dans un tableau X (resp. B, tableau Y). Puisqu'il y a $n/2$ éléments dans A, alors la complexité de cette opération est majorée grossièrement par un $O(\frac{n}{2}2^{n/2})$.

Algorithme (2)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Fusionner les 2 sous-problèmes : trouver les couples $(x, y) \in X \times Y$ tels que $x + y \leq S$:

Algorithme (2)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Fusionner les 2 sous-problèmes : trouver les couples $(x, y) \in X \times Y$ tels que $x + y \leq S$:
  - En force brute, puisque X, Y sont de taille en gros $n/2$, on obtient par une double boucle toutes les sommes en $O((2^{n/2})^2) = O(2^n)$: pas mieux que la force brute initiale.

Algorithme (2)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Fusionner les 2 sous-problèmes : trouver les couples $(x, y) \in X \times Y$ tels que $x + y \leq S$:
  - En force brute, puisque X, Y sont de taille en gros $n/2$, on obtient par une double boucle toutes les sommes en $O((2^{n/2})^2) = O(2^n)$: pas mieux que la force brute initiale.
  - Dans un premier temps on trie Y (et pas X) en $O(\frac{n}{2}2^{n/2})$.

Algorithme (2)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Fusionner les 2 sous-problèmes : trouver les couples $(x, y) \in X \times Y$ tels que $x + y \leq S$:
  - En force brute, puisque X, Y sont de taille en gros $n/2$, on obtient par une double boucle toutes les sommes en $O((2^{n/2})^2) = O(2^n)$: pas mieux que la force brute initiale.
  - Dans un premier temps on trie Y (et pas X) en $O(\frac{n}{2}2^{n/2})$.
  - On mémorise la meilleure somme **m** trouvée jusqu'ici : initialisation $m \leftarrow 0$.

Algorithme (2)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Fusionner les 2 sous-problèmes : trouver les couples $(x, y) \in X \times Y$ tels que $x + y \leq S$:
  - En force brute, puisque X, Y sont de taille en gros $n/2$, on obtient par une double boucle toutes les sommes en $O((2^{n/2})^2) = O(2^n)$: pas mieux que la force brute initiale.
  - Dans un premier temps on trie Y (et pas X) en $O(\frac{n}{2}2^{n/2})$.
  - On mémorise la meilleure somme **m** trouvée jusqu'ici : initialisation $m \leftarrow 0$.
  - Pour chaque $x \in X$, on recherche par dichotomie le plus grand $y \in Y$ tel que $x + y \leq S$. Si $x + y > m$, alors $m \leftarrow x + y$.

Algorithme (2)

Considérons un ensemble de n nombres. Notation $n/2$: division euclidienne par 2.

- Fusionner les 2 sous-problèmes : trouver les couples $(x, y) \in X \times Y$ tels que $x + y \leq S$:
  - En force brute, puisque X, Y sont de taille en gros $n/2$, on obtient par une double boucle toutes les sommes en $O((2^{n/2})^2) = O(2^n)$: pas mieux que la force brute initiale.
  - Dans un premier temps on trie Y (et pas X) en $O(\frac{n}{2}2^{n/2})$.
  - On mémorise la meilleure somme **m** trouvée jusqu'ici : initialisation $m \leftarrow 0$.
  - Pour chaque $x \in X$, on recherche par dichotomie le plus grand $y \in Y$ tel que $x + y \leq S$. Si $x + y > m$, alors $m \leftarrow x + y$.
  - On fait donc $|X| \simeq 2^{n/2}$ fois une recherche en $\log_2(|Y|) \simeq \log(2^{n/2})$. Comme $\log_2(2^{n/2}) = n/2 \log_2 2 = n/2$, La complexité totale est en $O(n/2 \times 2^{n/2}) = O(n2^{n/2})$, coûteuse mais moins que $O(2^n)$

Rappels utiles pour l'implantation

- **1L<<52** calcule 2^{52} mais caste d'abord 1 en un **long**

Rappels utiles pour l'implantation

- **1L<<52** calcule 2^{52} mais caste d'abord 1 en un **long**
- Le codage binaire d'un entier i représente un ensemble E_i d'entiers. Exemple $i = 100101$ représente $E_i \{0, 2, 5\}$.

Rappels utiles pour l'implantation

- **1L<<52** calcule 2^{52} mais caste d'abord 1 en un **long**
- Le codage binaire d'un entier i représente un ensemble E_i d'entiers. Exemple $i = 100101$ représente $E_i\,\{0, 2, 5\}$.
- **i & 1l<<j** cherche si le j-ième bit de i est à 1, donc si $j \in E_i$.