Algorithmique et structures de données

Coûts

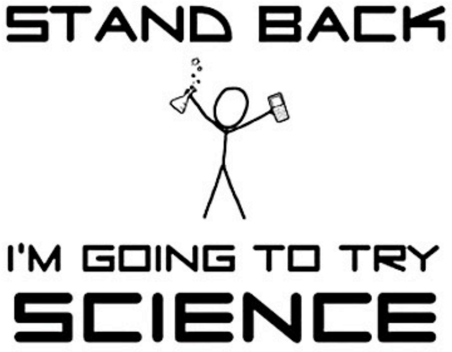
Frédéric Wagner

Ensimag $1^{\text{ère}}$ année – 2022-2023

Sommaire

Liste des gros mots

- modèle
- conjecture
- prédiction
- expérience
- validation / invalidation
- théorème



[a]

a. xkcd

Au programme

- calcul de coûts
- diviser pour régner
- dictionnaires
  - arbres de recherche
  - tables de hachage
  - sortedcontainers
- files de priorité
  - tas
- parcours de graphes

Attention

avoir entendu parler de $\neq$ maîtriser

Fonctionnement

- cours (8 séances)
- projet (3 séances)
- TD (1 / semaine)

Notes

- examen écrit final (2/3 de la note)
- projet (1/3 de la note)

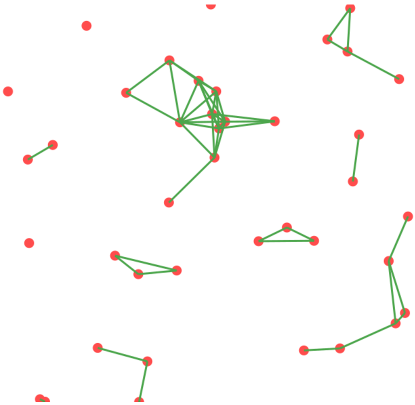
À noter

- c'est l'occasion de remonter si vous êtes forts en math (pas de déboguage)
- beaucoup de bruit en amphi ?

Sommaire

Projet d'algo

- ensemble de points du plan
- distance
- graphe de proximité
- compter la taille de chaque composante connexe



Mode d'emploi

- pas d'indications
- étalé sur tout le semestre
- de nombreux algorithmes possibles
- quelques indices au fil des cours
- projet sur gitlab
- pas de libs externes (pas de numpy)
- high-score
  - un run par nuit et par équipe
  - trois jeux de tests
  - temps limite

Sommaire

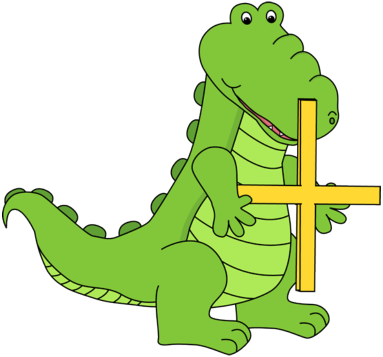
Différentes observations

- on peut regarder :
  - le nombre d'instructions exécutées
  - la place mémoire maximale utilisée
  - le nombre d'appels à certaines fonctions
  - le nombre de branchements exécutés
  - le nombre de comparaisons exécutées
  - le nombre d'œufs lancés
  - ...

Différents types de coûts

- pire cas
  - permet d'être sûr que tout est ok
  - "facile" à calculer
- meilleur cas
  - "facile" à calculer
  - permet d'être sûr que ça ne marchera pas
- en moyenne
  - ce que l'on veut vraiment
  - difficile à calculer
- amorti
  - pour des séquences d'opérations

Coût

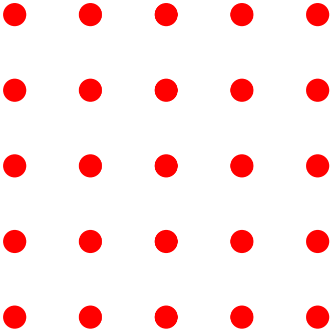


Associativité, commutativité,...

$1+(2+3) = (1+2)+3 = (1+3)+2 = 1+1+1+1+1+1, \ldots$

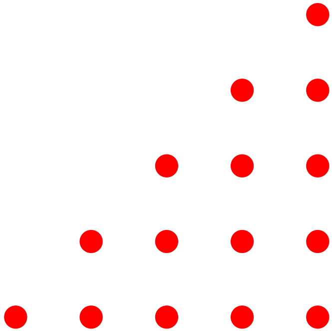
Exemples

```c
void cout1(int n) {
    for (int x = 1 ; x <= n ; x++) {
        for (int y = 1 ; y <= n ; y++) {
            f(x, y)
        }
    }
}
```



Exemples

```
void cout2(int n) {
    for (int x = 1 ; x <= n ; x++) {
        for (int y = 1 ; y <= x ; y++) {
            f(x, y)
        }
    }
}
```



En réalité

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c_{f(x,y)}$$

Grenoble INP Ensimag

En réalité

$$\sum_{x=1}^{n}\sum_{y=1}^{x} c_{f(x,y)}$$

si le coût de f est borné par c :

$$\sum_{x=1}^{n}\sum_{y=1}^{x} c$$

En réalité

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c_{f(x,y)}$$

si le coût de f est borné par c :

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c$$

$$= c \times \sum_{x=1}^{n} x$$

En réalité

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c_{f(x,y)}$$

si le coût de f est borné par c :

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c$$

$$= c \times \sum_{x=1}^{n} x$$

$$= c \times (1 + 2 + 3 + \cdots + n)$$

En réalité

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c_{f(x,y)}$$

si le coût de f est borné par c :

$$\sum_{x=1}^{n} \sum_{y=1}^{x} c$$

$$= c \times \sum_{x=1}^{n} x$$

$$= c \times (1 + 2 + 3 + \cdots + n)$$

$$= c \times \frac{n(n+1)}{2}$$
$$= O(n^2)$$

Sommaire

Pseudo code

Entrées : E : un ensemble d'entiers, k : un entier

Sorties : m : nombre d'entiers supérieurs ou égaux à k

$m \leftarrow 0$;
pour chaque $e \in E$ **faire**
    si $k \leq e$ **alors**
        $m \leftarrow m + 1$;
    finsi
fin

Coût

Combien de fois change m ?

Coût en moyenne

- la moyenne sur quoi ?

Coût en moyenne

- la moyenne sur quoi ?
- sur toutes les entrées possibles
- comment savoir quelles sont les entrées possibles ?
- avec quelles fréquences d'apparitions ?

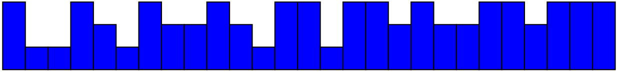
Coût en moyenne

- la moyenne sur quoi ?
- sur toutes les entrées possibles
- comment savoir quelles sont les entrées possibles ?
- avec quelles fréquences d'apparitions ?

Attention

Le code n'est exécuté qu'une seule fois. Il s'agit d'une espérance sur toutes les exécutions possibles et non d'une moyenne sur toutes les exécutions réalisées (coût amorti).

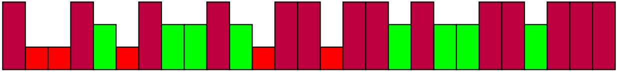
Calculer un coût en moyenne



e_1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
e_2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

- aire sous la courbe / nombre d'entrées

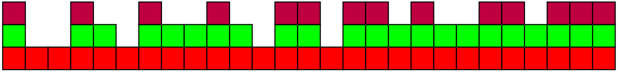
Somme 1



e_1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
e_2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

- $\mathcal{C} = \sum_{c=1}^{\infty} c \times p(\text{coût} = c)$
- $1 \times \frac{5}{27} + 2 \times \frac{8}{27} + 3 \times \frac{14}{27} = \frac{7}{3}$

Somme 2



e_1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2
e_2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2	0	0	0	1	1	1	2	2	2
k	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

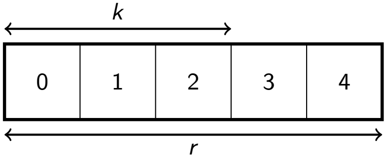
- $\mathcal{C} = \sum_{c=1}^{\infty} 1 \times p(\text{coût} \geq c)$
- $\frac{27}{27} + \frac{22}{27} + \frac{14}{27} = \frac{63}{27} = \frac{7}{3}$

Somme 3

- on peut aussi sommer sur les itérations
  - le coût total est 1 + la somme des coûts de chaque itération
- en notant r le nombre d'entiers, n la taille du tableau, c_i le coût de l'itération i
  - $\mathcal{C} = 1 + \sum_{i=1}^{n} c_i$

Somme 3

- on peut aussi sommer sur les itérations
  - le coût total est 1 + la somme des coûts de chaque itération
- en notant r le nombre d'entiers, n la taille du tableau, c_i le coût de l'itération i
  - $\mathcal{C} = 1 + \sum_{i=1}^{n} c_i$



- $c_i = \frac{\sum_{k=0}^{r-1}\left(0\times\frac{k}{r}+1\times\frac{r-k}{r}\right)}{r} = \frac{r-\frac{\sum_{k=0}^{r-1}k}{r}}{r} = \frac{r-(r-1)/2}{r} = \frac{r+1}{2r}$
- $\mathcal{C} = 1 + \sum_{i=1}^{n}\frac{r+1}{2r} = 1 + \frac{n(r+1)}{2r}$

Collection complète

Entrées : un ensemble infini de cartes à acheter

Sorties : vous n'avez plus d'argent

```
tant que on n'a pas toutes les cartes faire
    acheter la prochaine carte;
fintq
```

Coût en moyenne

- la moyenne sur quoi ?

Coût en moyenne

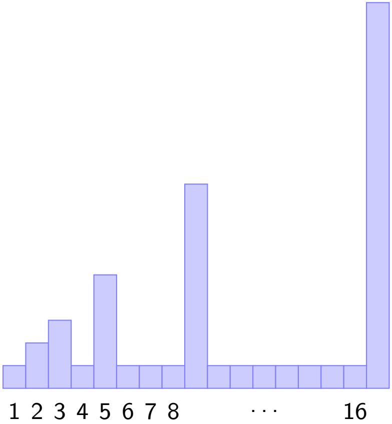
- la moyenne sur quoi ?
- n cartes possibles
- notations
  - c_i le nombre moyen d'achats pour obtenir une $i^{\text{ème}}$ carte
  - a_i le nombre d'achats pour obtenir une $i^{\text{ème}}$ carte
- $\mathcal{C} = \sum_{i=1}^{n} c_i$
- $c_i = \sum_{j=1}^{\infty} p(a_i \geq j) = \sum_{j=1}^{\infty} \left(\frac{i-1}{n}\right)^{j-1} = \frac{1}{1-\frac{i-1}{n}} = \frac{n}{n-i+1}$
- $\mathcal{C} = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{k=1}^{n} \frac{1}{k} = \Theta(n \log(n))$

Sommaire

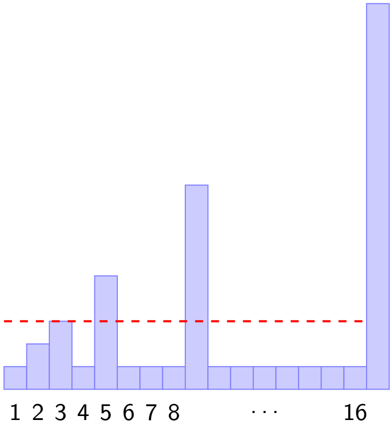
Vecteur

```c
struct vecteur {
  int *tableau;
  unsigned int taille;
  unsigned int nb_elements;
};
void ajout_element(struct vecteur *v, int e) {
  if (v->taille == v->nb_elements) {
    int *nouveau_tableau = malloc(2*v->taille*sizeof(int));
    for(unsigned int i = 0 ; i < v->taille ; i++)
      nouveau_tableau[i] = v->tableau[i];
    free(v->tableau);
    v->tableau = nouveau_tableau;
    v->taille *= 2;
  }
  v->tableau[v->nb_elements] = e;
  v->nb_elements++;
}
```

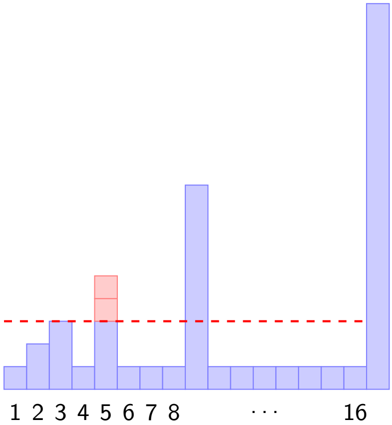
Coût amorti



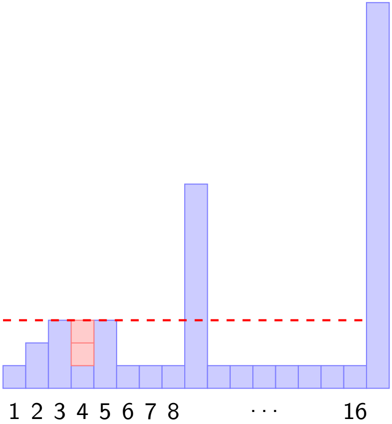
Coût amorti



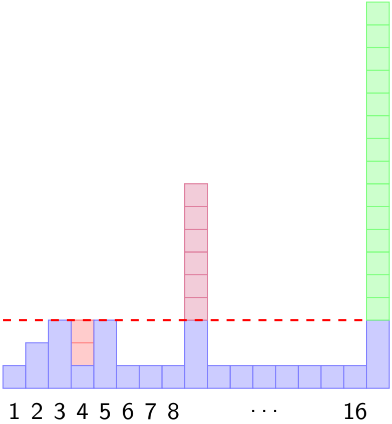
Coût amorti



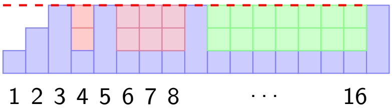
Coût amorti



Coût amorti



Coût amorti



Potentiel

- potentiel Φ
  - "compte épargne" ou "mensualisation"
  - on paye plus cher les opérations peu chères
    - en posant sur le compte
  - on paye moins cher les opérations chères
    - en **retirant** du compte

Potentiel

- potentiel Φ
  - "compte épargne" ou "mensualisation"
  - on paye plus cher les opérations peu chères
    - en posant sur le compte
  - on paye moins cher les opérations chères
    - en **retirant** du compte
- ici :
  - soit t_i la taille du tableau à l'instant i
  - soit c_i le nombre de cases vides à l'instant i
  - $\Phi_i = 1 + t_i - 2 \times c_i$

Attention

Pas de découvert. Le potentiel n'est jamais en négatif.

Potentiel : vecteur

- $\Phi_i = 1 + t_i - 2 \times c_i$
- le potentiel est positif
  - le tableau est toujours à moitié rempli (sauf tableau vide)
- coût d'un ajout
  - ajout simple
    - une case vide en moins
    - le potentiel augmente de 2
    - le coût compensé est donc de 1+2=3
  - ajout redimensionnant
    - t_i passe de t_i à $2t_i$ et c_i de 0 à $t_i - 1$
    - Φ_i passe donc de $1 + t_i$ à $1 + 2t_i - 2t_i + 2$
    - différence de $t_i - 2$ pour un coût réel de $t_i + 1$
    - le coût compensé est donc de 3