Conception d'algorithmes

Force brute

Force brute

- Simple
- On essaie toutes les possibilités sans se préoccuper d'une quelconque stratégie
- On trouve la solution si elle existe

Force brute : principe

- Il faut pouvoir énumérer toutes les possibilités de solutions
  - Définir une manière de parcourir toutes les possibilités
- Savoir si une possibilité est satisfaisante
- ***Recherche exhaustive***
- Manière la plus directe d'aborder un problème

- Exemple
  - Cracker un mot de passe
    - Prendre un dictionnaire et essayer tous les mots
  - Deviner l'âge d'une personne
    - Énumérer les entiers

Algo abstrait (wikipedia)

- Soit un problème P à résoudre
- Soit ***E l'ensemble des états*** muni des opérations suivantes
  - first() → une solution candidate (ou null)
  - next(c) → solution candidate suivante de c (ou null)
  - valid(c,P) → booléen vrai si solution valide
  - output(c) → stocke la solution (si on les veut toutes)
- c← first(E)
- Tant que c ≠ null faire
  - Si valid(c,P) alors output(c)
  - c← next(c)

- Complexité = ??

Exemple introductif

- Soit différentes pièces de monnaies
  - en centimes : 100, 1, 2, 2,10, 5, 2, 20, 2, 100, 200
- Somme à rendre 37

- Comment écrire un algorithme qui calcule si on peut rendre la monnaie ?

- Solutions possibles
  - 100
  - 100 +1
  - 100+1+2
  - 100+1+2+2
  - …
  - 1
  - 1+2
  - 1+2+2
  - etc

Au bout de combien d'étapes trouve t-on la solution ????

Exemple: trouver un nombre *x* entre 1 et N

- Les états -> 1… N
- fisrt → 1
- next→ incrémenter
- output → stocker dans une liste
- valid → comparer l'état avec x

Exemple: trouver un nombre x entre 1 et N (bis)

- Les états -> 1… N (représentés sous forme d'ensemble)
- next→ tirer un nb au hasard et le retirer de l'ensemble
- output → stocker dans une liste
- valid → comparer le nb avec x

Critique

- Facile à mettre en œuvre (dès qu'on sait énumérer les solutions !)
- Le parcours étant fait sans connaissance du problème et de ses caractéristiques
  - On veut faire 37 et on essaie avec 200 !
  - Connaissances heuristiques :
    - Ex réduire l'espace de recherche ; ne pas considérer les pièces de valeurs supérieures à la somme à rendre

    ➔ introduire des choix à chaque étape
- Explosion combinatoire !!
  - Ici 2^8=256 possibilités (c'est encore possible)
  - Si on a 70 pièces en caisse ….

Retour sur le problème des pièces à rendre

- Donner un algorithme de force brute résolvant ce problème
  - Exercice/programmation

Pièces de monnaie

- Solution 1 (force très brute)

```
for(int i0=0;i0<=1;i0++){
    for(int i1=0;i0<=1;i0++){
    Etc.
    if (i0*200+i1*100+ …==somme) { s++;}
    }
}
```

Inconvénients ?

Pièces de monnaie

- Solution 2 (force brute)
  - valeur[0] =200; valeur[1]=100;

```
for(int i0=0;i0<=1;i0++){
      for(int i1=0;i0<=1;i0++){
      Etc.
      if (i0*valeur[0]+i1*valeur[1]+ …==somme) { s++;}
      }
 }
```

Inconvénients ?

Retour sur le problème des pièces à rendre

- Donner un algorithme de force brute (itératif ou non)
  - Exercice/programmation
    - On dispose de N pièces
    - Idée 1:
      - Soit pièce un tableau tq piece[i] correspond à la valeur de la ième pièce
      - Soit encompte[i] un tableau à valeur dans 0/1
      - Écrire un programme avec n itérations → critique ?
    - Idée 2: parcours par récursivité (-> backtrack)
    - Idée 3: générer toutes les solutions/se donner une manière d'énumérer cf exemple