

### Notion de stratégie algorithmique

- La recherche d'une solution à un problème peut se faire de différentes manières, on parle de **stratégies algorithmiques**

### Notion de stratégie algorithmique

- La recherche d'une solution à un problème peut se faire de différentes manières, on parle de **stratégies algorithmiques**
- Ce chapitre introduit les stratégies suivantes :

### Notion de stratégie algorithmique

- La recherche d'une solution à un problème peut se faire de différentes manières, on parle de **stratégies algorithmiques**
- Ce chapitre introduit les stratégies suivantes :
  - La **recherche exhaustives** qui consiste à explorer toutes les solutions possibles.

### Notion de stratégie algorithmique

- La recherche d'une solution à un problème peut se faire de différentes manières, on parle de **stratégies algorithmiques**
- Ce chapitre introduit les stratégies suivantes :
  - La **recherche exhaustives** qui consiste à explorer toutes les solutions possibles.
  - La stratégie **gloutonne** qui consiste pour résoudre un problème d'optimisation à faire un choix local optimal à chaque étape *sans garantie que cela conduise à l'optimal global*.

### Notion de stratégie algorithmique

- La recherche d'une solution à un problème peut se faire de différentes manières, on parle de **stratégies algorithmiques**
- Ce chapitre introduit les stratégies suivantes :
  - La **recherche exhaustives** qui consiste à explorer toutes les solutions possibles.
  - La stratégie **gloutonne** qui consiste pour résoudre un problème d'optimisation à faire un choix local optimal à chaque étape *sans garantie que cela conduise à l'optimal global*.
- D'autres stratégies consistant à **décomposer le problème en sous problèmes** sera vu plus loin dans l'année.

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Définition

La recherche par **force brute** (*brute force*) consiste à parcourir toutes les solutions possibles d'un problème en testant si elles conviennent.

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Définition

La recherche par **force brute** (*brute force*) consiste à parcourir toutes les solutions possibles d'un problème en testant si elles conviennent.

De façon formelle, si on note  $V$  l'ensemble des candidats, et  $P$  une propriété des éléments de  $V$ , on teste (en les énumérant) les  $x \in V$  jusqu'à en trouver un qui vérifie  $P$ .

## C13 Force brute, retour sur trace

### 2. Force brute : généralités

#### Définition

La recherche par **force brute** (*brute force*) consiste à parcourir toutes les solutions possibles d'un problème en testant si elles conviennent.

De façon formelle, si on note  $V$  l'ensemble des candidats, et  $P$  une propriété des éléments de  $V$ , on teste (en les énumérant) les  $x \in V$  jusqu'à en trouver un qui vérifie  $P$ .

#### Remarques

- Dans certains problèmes on cherche à déterminer *toutes* les solutions et donc on ne s'arrête pas à la première rencontrée



# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Définition

La recherche par **force brute** (*brute force*) consiste à parcourir toutes les solutions possibles d'un problème en testant si elles conviennent.

De façon formelle, si on note  $V$  l'ensemble des candidats, et  $P$  une propriété des éléments de  $V$ , on teste (en les énumérant) les  $x \in V$  jusqu'à en trouver un qui vérifie  $P$ .

### Remarques

- Dans certains problèmes on cherche à déterminer *toutes* les solutions et donc on ne s'arrête pas à la première rencontrée
- On doit pouvoir **énumérer de façon exhaustive** les éléments de  $V$  (ce qui peut être difficile dans certains cas).

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Exemples

- La recherche d'un mot de passe par force brute :  $V$  est l'ensemble des chaînes de caractères et  $P(x)$  est vérifié si  $x$  est le mot de passe cherché. Dans ce cas on pourra réaliser l'énumération des candidats en commençant par les chaînes de longueur 1, puis 2, ...

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Exemples

- La recherche d'un mot de passe par force brute :  $V$  est l'ensemble des chaînes de caractères et  $P(x)$  est vérifié si  $x$  est le mot de passe cherché. Dans ce cas on pourra réaliser l'énumération des candidats en commençant par les chaînes de longueur 1, puis 2, ...
- $V$  est l'ensemble des grilles complétées d'un sudoku et  $P$  vérifie si la grille est valide.

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Exemples

- La recherche d'un mot de passe par force brute :  $V$  est l'ensemble des chaînes de caractères et  $P(x)$  est vérifié si  $x$  est le mot de passe cherché. Dans ce cas on pourra réaliser l'énumération des candidats en commençant par les chaînes de longueur 1, puis 2, ...
- $V$  est l'ensemble des grilles complétées d'un sudoku et  $P$  vérifie si la grille est valide.
- $V$  est l'ensemble des permutations possibles d'une liste d'entiers et  $P$  vérifie si la permutation est triée en ordre croissant (**bogosort**).

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Complexité

En supposant l'ensemble  $V$  fini, une recherche exhaustive effectuée au plus  $|V|$  itérations.

⚠ Cela ne signifie pas que la complexité est toujours en  $O(V)$  car tester si une solution vérifie  $P$  ou non peut avoir un coût non constant.

# C13 Force brute, retour sur trace

## 2. Force brute : généralités

### Complexité

En supposant l'ensemble  $V$  fini, une recherche exhaustive effectuée au plus  $|V|$  itérations.

⚠ Cela ne signifie pas que la complexité est toujours en  $O(V)$  car tester si une solution vérifie  $P$  ou non peut avoir un coût non constant.

### Problème d'optimisation

Dans les problèmes du type « déterminer  $x \in V$  tel que  $f(x)$  soit minimale (ou maximale) », l'exploration exhaustive va résoudre le problème en calculant toutes les images par  $f$  des éléments de  $V$ .

# C13 Force brute, retour sur trace

## 3. Mot de passe par force brute

### Exercice

Tableau des temps pour la recherche de mot de passe par force brute :

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 In years	7qd years

On s'intéresse à la recherche d'un mot de passe de 8 lettres minuscules.

# C13 Force brute, retour sur trace

## 3. Mot de passe par force brute

### Exercice

Tableau des temps pour la recherche de mot de passe par force brute :

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 In years	7qd years

On s'intéresse à la recherche d'un mot de passe de 8 lettres minuscules.

- Combien il y a-t-il de mots de passes possibles ? Quel est le temps indiqué dans le tableau ?



# C13 Force brute, retour sur trace

## 3. Mot de passe par force brute

### Exercice

Tableau des temps pour la recherche de mot de passe par force brute :

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 In years	7qd years

On s'intéresse à la recherche d'un mot de passe de 8 lettres minuscules.

- Combien il y a-t-il de mots de passes possibles ? Quel est le temps indiqué dans le tableau ?
- Proposer un algorithme permettant d'énumérer les possibilités.

# C13 Force brute, retour sur trace

## 3. Mot de passe par force brute

### Exercice

Tableau des temps pour la recherche de mot de passe par force brute :

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	1 sec	5 secs
7	Instantly	Instantly	25 secs	1 min	6 mins
8	Instantly	5 secs	22 mins	1 hour	8 hours
9	Instantly	2 mins	19 hours	3 days	3 weeks
10	Instantly	58 mins	1 month	7 months	5 years
11	2 secs	1 day	5 years	41 years	400 years
12	25 secs	3 weeks	300 years	2k years	34k years
13	4 mins	1 year	16k years	100k years	2m years
14	41 mins	51 years	800k years	9m years	200m years
15	6 hours	1k years	43m years	600m years	15 bn years
16	2 days	34k years	2bn years	37bn years	1tn years
17	4 weeks	800k years	100bn years	2tn years	93tn years
18	9 months	23m years	6tn years	100 In years	7qd years

On s'intéresse à la recherche d'un mot de passe de 8 lettres minuscules.

- Combien il y a-t-il de mots de passes possibles ? Quel est le temps indiqué dans le tableau ?
- Proposer un algorithme permettant d'énumérer les possibilités.
- Ecrire en C, une fonction de signature  
`char *bruteforce(char *mdp, char *charset, int size)`  
qui implémente une recherche de mot de passe par force brute.

## 3. Mot de passe par force brute

## Proposition de solution

```
1 char *bruteforce(char *mdp, char *charset, int size)
2 {
3     char *test = malloc(sizeof(char) * size);
4     int nb_char = strlen(charset);
5     int idx[size];
6     int cidx;
7     for (int k = 0; k < size; k++)
8     {
9         idx[k] = 0;
10        test[k] = charset[0];
11    }
12    while (idx[0] < nb_char)
13    {
14        if (strcmp(test, mdp) == 0)
15        {
16            return test;
17        }
18        idx[size - 1]++;
19        cidx = size - 1;
20        while (cidx != 0 && idx[cidx] == nb_char)
21        {
22            idx[cidx] = 0;
23            test[cidx] = charset[0];
24            cidx--;
25            idx[cidx]++;
26            test[cidx] = charset[idx[cidx]];
27        }
28        test[size - 1] = charset[idx[size - 1]];
29    }
30    return "";
31 }
```

# C13 Force brute, retour sur trace

## 4. Résolution par retour sur trace

### Définition

Le **retour sur trace** (*backtracking*) est aussi une méthode de recherche par force brute mais on dispose d'une **méthode de validation d'une solution partielle**. Ainsi, si un début de solution s'avère non valide, on n'explore pas les solutions complètes qui en découlent.

# C13 Force brute, retour sur trace

## 4. Résolution par retour sur trace

### Définition

Le **retour sur trace** (*backtracking*) est aussi une méthode de recherche par force brute mais on dispose d'une **méthode de validation d'une solution partielle**. Ainsi, si un début de solution s'avère non valide, on n'explore pas les solutions complètes qui en découlent.

### Exemple

Pour résoudre un Sudoku :

# C13 Force brute, retour sur trace

## 4. Résolution par retour sur trace

### Définition

Le **retour sur trace** (*backtracking*) est aussi une méthode de recherche par force brute mais on dispose d'une **méthode de validation d'une solution partielle**. Ainsi, si un début de solution s'avère non valide, on n'explore pas les solutions complètes qui en découlent.

### Exemple

Pour résoudre un Sudoku :

- La force brute parcourt l'ensemble des valeurs possibles pour toutes les cases restantes

# C13 Force brute, retour sur trace

## 4. Résolution par retour sur trace

### Définition

Le **retour sur trace** (*backtracking*) est aussi une méthode de recherche par force brute mais on dispose d'une **méthode de validation d'une solution partielle**. Ainsi, si un début de solution s'avère non valide, on n'explore pas les solutions complètes qui en découlent.

### Exemple

Pour résoudre un Sudoku :

- La force brute parcourt l'ensemble des valeurs possibles pour toutes les cases restantes
- Le *backtracking* place des valeurs au fur et à mesure et revient en arrière si une impossibilité est rencontrée.

## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

#### Exemple

Le problème des  $n$  reines consiste à placer  $n$  reines sur une échiquier de taille  $n$  de façon à ce que deux reines ne soit pas sur la même ligne, même colonne ou même diagonale (c'est-à-dire qu'aucune reine n'en menace une autre)



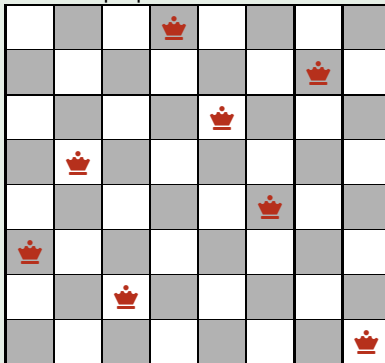
## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

#### Exemple

Le problème des  $n$  reines consiste à placer  $n$  reines sur une échiquier de taille  $n$  de façon à ce que deux reines ne soit pas sur la même ligne, même colonne ou même diagonale (c'est-à-dire qu'aucune reine n'en menace une autre)

Une solution dans le cas  $n = 8$  est proposée ci-dessous :



## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

#### Représentation du problème

- On sait qu'il y a une seule reine par colonne, on peut donc représenter une position de jeu par un tableau de taille  $n$  contenant les numéros de ligne des  $k$  reines déjà placées.

## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

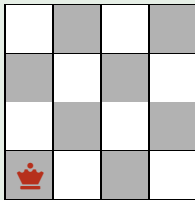
#### Représentation du problème

- On sait qu'il y a une seule reine par colonne, on peut donc représenter une position de jeu par un tableau de taille  $n$  contenant les numéros de ligne des  $k$  reines déjà placées.
- L'algorithme va alors consister à tenter de placer la reine  $k + 1$  sur chacune des lignes  $0, \dots, k - 1$ 
  - si cela génère une menace, on essayer la possibilité suivante, en revenant récursivement à la reine précédente si nécessaire
  - sinon on place la reine suivante, si c'est la dernière reine, une solution est trouvée.

# C13 Force brute, retour sur trace

## 5. Exemple : le problème des $n$ reines

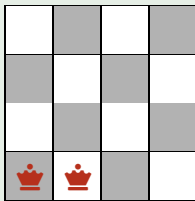
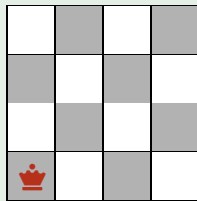
Début de l'algorithme ( $n = 4$ )



## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

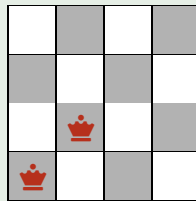
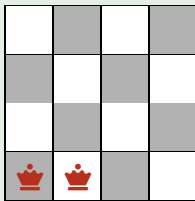
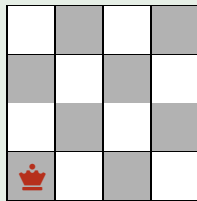
Début de l'algorithme ( $n = 4$ )



# C13 Force brute, retour sur trace

## 5. Exemple : le problème des $n$ reines

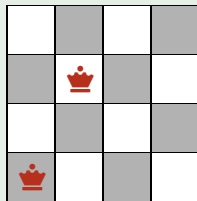
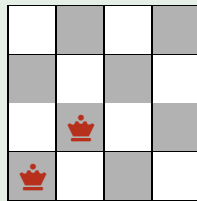
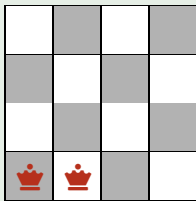
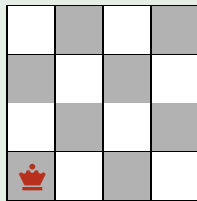
Début de l'algorithme ( $n = 4$ )



# C13 Force brute, retour sur trace

## 5. Exemple : le problème des $n$ reines

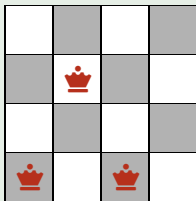
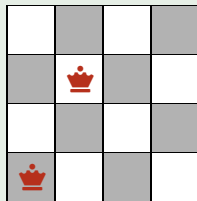
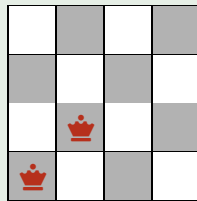
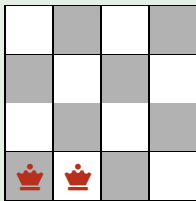
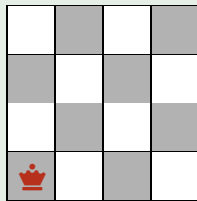
### Début de l'algorithme ( $n = 4$ )



# C13 Force brute, retour sur trace

## 5. Exemple : le problème des $n$ reines

### Début de l'algorithme ( $n = 4$ )

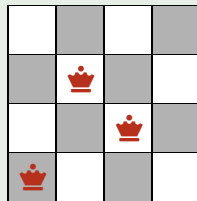
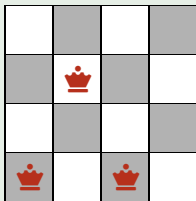
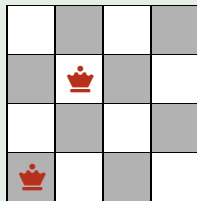
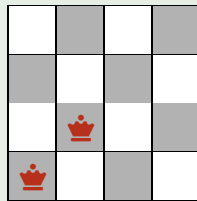
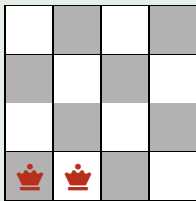
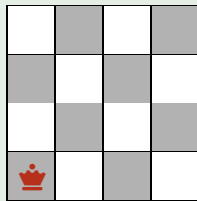




# C13 Force brute, retour sur trace

## 5. Exemple : le problème des $n$ reines

### Début de l'algorithme ( $n = 4$ )



## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

#### Implémentation en langage C

- 1 Ecrire une fonction de signature `bool menace(int tab[], int idx)` qui renvoie `true` si la reine située en colonne `idx` menace l'une des reines situés en colonne `0`, ..., `idx-1`.

# C13 Force brute, retour sur trace

## 5. Exemple : le problème des n reines

### Implémentation en langage C

- ① Ecrire une fonction de signature `bool menace(int tab[], int idx)` qui renvoie `true` si la reine située en colonne `idx` menace l'une des reines situées en colonne `0`, ..., `idx-1`.

```
1  bool menace(int tab[], int idx)
2  {
3      for (int i = 0; i <= idx - 1; i++)
4      {
5          if (tab[i] == tab[idx] || abs(tab[i] - tab[idx]) == idx - i)
6          {
7              return true;
8          }
9      }
10     return false;
11 }
```

# C13 Force brute, retour sur trace

## 5. Exemple : le problème des $n$ reines

### Implémentation en langage C

- 1 Ecrire une fonction de signature `bool menace(int tab[], int idx)` qui renvoie `true` si la reine située en colonne `idx` menace l'une des reines situées en colonne `0`, ..., `idx-1`.

```
1  bool menace(int tab[], int idx)
2  {
3      for (int i = 0; i <= idx - 1; i++)
4      {
5          if (tab[i] == tab[idx] || abs(tab[i] - tab[idx]) == idx - i)
6          {
7              return true;
8          }
9      }
10     return false;
11 }
```

- 2 Ecrire une fonction qui renvoie la première solution rencontrée sous la forme du tableau des positions par colonne des  $n$  reines.

## Proposition de solution

```
1  bool solve(int sol[], int size, int idx)
2  {
3      if (idx == size)
4      {
5          return true;
6      }
7      else
8      {
9          for (int p = 0; p < size; p++)
10         {
11             sol[idx] = p;
12             if (!menace(sol, idx) && solve(sol, size, idx + 1))
13             {
14                 return true;
15             }
16         }
17         return false;
18     }
19 }
```

## C13 Force brute, retour sur trace

### 5. Exemple : le problème des $n$ reines

#### Exercice

Modifier la fonction précédente afin qu'elle calcule le nombre total de solutions au problème.

## Exercice

Modifier la fonction précédente afin qu'elle calcule le nombre total de solutions au problème.

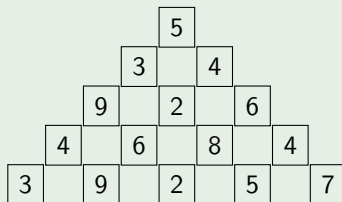
```
1 void nreines(int sol[], int size, int index, int *nb_sol)
2 {
3     if (index == size)
4     {
5         *nb_sol = *nb_sol+1;
6     }
7     else
8     {
9         for (int p = 0; p < SIZE; p++)
10        {
11            sol[index] = p;
12            if (!menace(sol, index))
13            {
14                nreines(sol, size, index + 1, nb_sol);
15            }
16        }
17    }
18 }
```

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

On s'intéresse à la recherche de la somme maximale d'un chemin qui part du haut de la pyramide et atteint sa base.



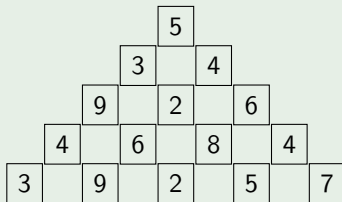


## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

On s'intéresse à la recherche de la somme maximale d'un chemin qui part du haut de la pyramide et atteint sa base.



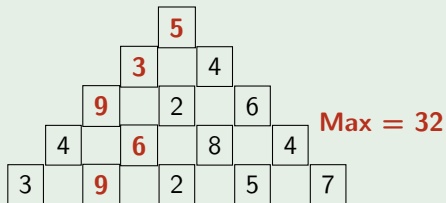
- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

On s'intéresse à la recherche de la somme maximale d'un chemin qui part du haut de la pyramide et atteint sa base.



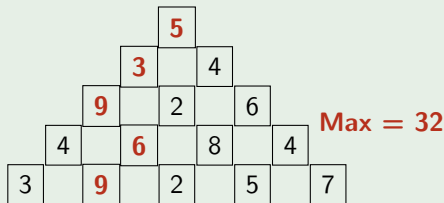
1. déterminer cette somme et un chemin possible sur l'exemple ci-dessus.

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

On s'intéresse à la recherche de la somme maximale d'un chemin qui part du haut de la pyramide et atteint sa base.



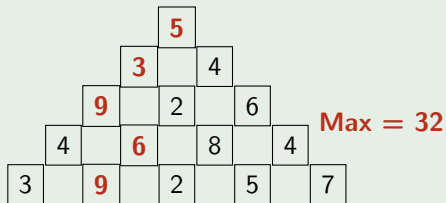
- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 On suppose qu'on applique la stratégie suivante : « à chaque niveau on choisit de descendre vers le cube de plus grand valeur ». Quel chemin obtient-on et quelle somme à ce chemin ?

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

On s'intéresse à la recherche de la somme maximale d'un chemin qui part du haut de la pyramide et atteint sa base.



- 1 déterminer cette somme et un chemin possible sur l'exemple ci-dessus.
- 2 On suppose qu'on applique la stratégie suivante : « à chaque niveau on choisit de descendre vers le cube de plus grand valeur ». Quel chemin obtient-on et quelle somme à ce chemin ?

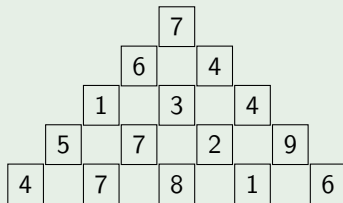
On obtient le chemin 5-4-6-8-5 de somme 28.

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

Même question pour la pyramide suivante :

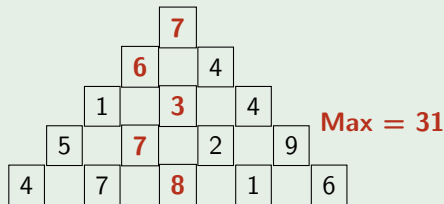


## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

Même question pour la pyramide suivante :

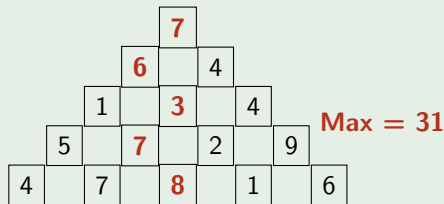


## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Chemin de somme maximale dans une pyramide

Même question pour la pyramide suivante :



Cette fois l'algorithme qui consiste à choisir de descendre vers le cube de plus grand valeur permet d'obtenir la plus grande somme.

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Implémentation en OCaml

- On représente une pyramide par un *tableau de tableau*, chacune des tableaux contient les coefficients d'un niveau de la pyramide.

Par exemple, la pyramide précédente correspond à la liste

```
[ [ 1 7 ] ; [ 1 6 ; 4 ] ; [ 1 1 ; 3 ; 4 ] ; [ 1 5 ; 7 ; 2 ; 9 ] ; [ 1 4 ; 7 ; 8 ; 1 ; 6 ] ]
```



## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Implémentation en OCaml

- On représente une pyramide par une *tableau de tableau*, chacune des tableaux contient les coefficients d'un niveau de la pyramide.  
Par exemple, la pyramide précédente correspond à la liste  
`[ [ 1 7 1 ]; [ 1 6; 4 1 ]; [ 1 1; 3 4 1 ] ; [ 1 5; 7 2; 9 1 ]; [ 1 4; 7 8; 1 6 1 ] ]`
- On écrit alors une fonction `glouton : int array array -> int` qui prend en argument un tableau de tableaux et renvoie la somme obtenue en suivant la stratégie gloutonne (descendre vers le cube de plus grand valeur).

## C13 Force brute, retour sur trace

### 6. Stratégie gloutonne : exemple introductif

#### Implémentation en OCaml

- On représente une pyramide par une *tableau de tableau*, chacune des tableaux contient les coefficients d'un niveau de la pyramide.  
Par exemple, la pyramide précédente correspond à la liste  
`[ [ 1 7 1 ]; [ 1 6; 4 1 ]; [ 1 1; 3 4 1 ] ; [ 1 5; 7 2; 9 1 ]; [ 1 4; 7 8; 1 6 1 ] ]`
- On écrit alors une fonction `glouton : int array array -> int` qui prend en argument un tableau de tableaux et renvoie la somme obtenue en suivant la stratégie gloutonne (descendre vers le cube de plus grand valeur).
- Pour chaque niveau `i` de la pyramide, on doit donc comparer les deux cubes se trouvant en dessous c'est à dire les cubes :
  - Situé juste en dessous c'est à dire en `(i+1,colonne)`
  - Situé en dessous et à droite c'est à dire en `(i+1,colonne+1)`

# C13 Force brute, retour sur trace

## 7. Notion d'algorithme glouton

### Stratégie gloutonne

Afin de résoudre un problème d'optimisation, on peut adopter une stratégie dite **gloutonne** :

# C13 Force brute, retour sur trace

## 7. Notion d'algorithme glouton

### Stratégie gloutonne

Afin de résoudre un problème d'optimisation, on peut adopter une stratégie dite **gloutonne** :

- à chaque étape on effectue le choix qui correspond à l'optimal **local** d'une grandeur.

# C13 Force brute, retour sur trace

## 7. Notion d'algorithme glouton

### Stratégie gloutonne

Afin de résoudre un problème d'optimisation, on peut adopter une stratégie dite **gloutonne** :

- à chaque étape on effectue le choix qui correspond à l'optimal **local** d'une grandeur.
- Ces choix successifs ne conduisent pas forcément à la solution optimale **globale**.

### Stratégie gloutonne

Afin de résoudre un problème d'optimisation, on peut adopter une stratégie dite **gloutonne** :

- à chaque étape on effectue le choix qui correspond à l'optimal **local** d'une grandeur.
- Ces choix successifs ne conduisent pas forcément à la solution optimale **globale**.
- Cette stratégie ne fournit donc pas toujours la **meilleure solution**.

# C13 Force brute, retour sur trace

## 7. Notion d'algorithme glouton

### Stratégie gloutonne

Afin de résoudre un problème d'optimisation, on peut adopter une stratégie dite **gloutonne** :

- à chaque étape on effectue le choix qui correspond à l'optimal **local** d'une grandeur.
- Ces choix successifs ne conduisent pas forcément à la solution optimale **globale**.
- Cette stratégie ne fournit donc pas toujours la **meilleure solution**.

### Définition

Un algorithme est dit **glouton** lorsqu'il procède par choix successifs, en sélectionnant à chaque étape l'option qui correspond à un optimum local sans garantie que cela conduise à l'optimum globale.




# C13 Force brute, retour sur trace

## 8. Exemple résolu : problème du sac à dos

### Position du problème

On dispose d'un sac à dos et d'une liste objet ayant chacun un poids et une valeur. Le problème du sac à dos consiste à remplir ce sac en maximisant la valeur des objets qu'il contient tout en respectant une contrainte sur le poids du sac.

### Exemple

180 €  0.3 kg	2050 €  4.1 kg	280 €  0.6 kg	810 €  1.7 kg
990 €  2 kg	1275 €  2.9 kg	2570 €  5.7 kg	920 €  2.1 kg

Quels objets doit-on prendre pour maximiser la valeur contenu si le poids doit rester inférieur à **8 kg** ?



## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Mise en place d'une stratégie gloutonne

On propose de résoudre ce problème en adoptant la stratégie gloutonne suivante :

- 1 On classe les objets selon un critère pertinent

# C13 Force brute, retour sur trace

## 8. Exemple résolu : problème du sac à dos

### Mise en place d'une stratégie gloutonne

On propose de résoudre ce problème en adoptant la stratégie gloutonne suivante :

- 1 On classe les objets selon un critère pertinent

Ici la valeur par unité de poids semble un critère intéressant

## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Mise en place d'une stratégie gloutonne

On propose de résoudre ce problème en adoptant la stratégie gloutonne suivante :

- 1 On classe les objets selon un critère pertinent  
Ici la valeur par unité de poids semble un critère intéressant
- 2 On parcourt dans l'ordre la liste *triée* des objets

# C13 Force brute, retour sur trace

## 8. Exemple résolu : problème du sac à dos

### Mise en place d'une stratégie gloutonne

On propose de résoudre ce problème en adoptant la stratégie gloutonne suivante :

- 1 On classe les objets selon un critère pertinent  
Ici la valeur par unité de poids semble un critère intéressant
- 2 On parcourt dans l'ordre la liste *triée* des objets
- 3 Si l'objet considéré rentre dans le sac en respectant la contrainte de poids on l'ajoute (en diminuant le poids restant), sinon on passe à l'objet suivant.

## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Utilisation de `sorted` pour classer les objets

- La fonction `sorted` de Python prend en argument une liste et renvoie cette liste triée dans l'ordre croissant

## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Utilisation de `sorted` pour classer les objets

- La fonction `sorted` de Python prend en argument une liste et renvoie cette liste triée dans l'ordre croissant

Par exemple `sorted([2, 1, 6, 4])` renvoie la liste `[1, 2, 4, 6]`

## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Utilisation de `sorted` pour classer les objets

- La fonction `sorted` de Python prend en argument une liste et renvoie cette liste triée dans l'ordre croissant  
Par exemple `sorted([2, 1, 6, 4])` renvoie la liste `[1, 2, 4, 6]`
- Le paramètre optionnel `reverse` classe dans l'ordre *décroissant* lorsqu'il vaut `True` Par exemple `sorted([2, 1, 6, 4], reverse=True)` renvoie la liste `[6, 4, 2, 1]`

## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Utilisation de `sorted` pour classer les objets

- La fonction `sorted` de Python prend en argument une liste et renvoie cette liste triée dans l'ordre croissant  
Par exemple `sorted([2, 1, 6, 4])` renvoie la liste `[1, 2, 4, 6]`
- Le paramètre optionnel `reverse` classe dans l'ordre *décroissant* lorsqu'il vaut `True` Par exemple `sorted([2, 1, 6, 4], reverse=True)` renvoie la liste `[6, 4, 2, 1]`



## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Utilisation de `sorted` pour classer les objets

- La fonction `sorted` de Python prend en argument une liste et renvoie cette liste triée dans l'ordre croissant  
Par exemple `sorted([2, 1, 6, 4])` renvoie la liste `[1, 2, 4, 6]`
- Le paramètre optionnel `reverse` classe dans l'ordre *décroissant* lorsqu'il vaut `True` Par exemple `sorted([2, 1, 6, 4], reverse=True)` renvoie la liste `[6, 4, 2, 1]`
- On peut préciser une *clé de tri* avec le paramètre `reverse`.

## C13 Force brute, retour sur trace

### 8. Exemple résolu : problème du sac à dos

#### Utilisation de `sorted` pour classer les objets

- La fonction `sorted` de Python prend en argument une liste et renvoie cette liste triée dans l'ordre croissant  
Par exemple `sorted([2, 1, 6, 4])` renvoie la liste `[1, 2, 4, 6]`
- Le paramètre optionnel `reverse` classe dans l'ordre *décroissant* lorsqu'il vaut `True` Par exemple `sorted([2, 1, 6, 4], reverse=True)` renvoie la liste `[6, 4, 2, 1]`
- On peut préciser une *clé de tri* avec le paramètre `reverse`.  
Par exemple si on souhaite trier une liste de points (représentés par des tuples) par ordonnée croissante, on crée une fonction renvoyant l'ordonnée d'un point : Puis on l'utilise comme clé de tri :  
`res = sorted([(2,2), (1,5), (6,2), (4,7)], key = ordonnee)`