Jeu du Baguenaudier [je03] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 22 mai 2018

Table des matières

C++ - Jeu du Baguenaudier (Solution)

Mots-Clés Jeu de démontage, Récursivité croisée ■
Requis Axiomatique impérative, Classes, Classes (suite), Récursivité des actions, Complexité des algorithmes ■
Difficulté • • • (1 h 30) ■

Objectif
Cet exercice réalise le jeu du Baguenaudier (jeu de démontage, récursivité croisée) et étudie sa complexité. Dans le même ordre d'idées, l'exercice @[Jeu du Tracassin] réalise le jeu du Tracassin.

1 Présentation du jeu

Jeu du Baguenaudier

Il se joue sur une tablette divisée en n cases. « Jouer » c'est ôter ou placer un pion sur une case selon que celle-ci est ou non remplie. L'unique règle : à chaque coup les seules cases **jouables** sont la première case ou la case qui suit la première case occupée.

Types de jeu

Les deux types de jeux sont :

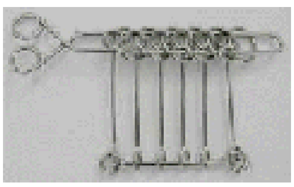
- Mettre n pions sur le baguenaudier initialement vide.
- Prendre n pions du baguenaudier initialement plein.

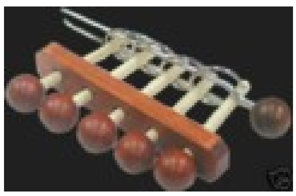
Exemple d'exécution

```
Taille du jeu? 3
On remplit le baguenaudier...
. . .
* . .
* * .
. * .
. * *
* * *
Nombre de coups = 6
0 1 3 2 6 7
Appuyez sur une touche pour continuer...
On vide le baguenaudier...
* * *
. * *
. * .
* * .
* . .
. . .
Nombre de coups = 6
7 6 2 3 1 0
```

Éditions du jeu

Il existe de nombreuses éditions de ce casse-tête, en métal ou en bois.





2 Classe Baguenaudier

On modélise le jeu du baguenaudier par une classe, et on représente un baguenaudier par un tableau b de n booléens telle que $b[k]$ est vrai s'il y a un pion sur la k-ème case et faux sinon.

Écrivez une classe `Baguenaudier` ayant pour attributs :

- Le nombre de pions `npions` (entier).
- La tablette `b` (vecteur de booléens).
- La liste des valeurs `vals` (liste d'entiers) générées lors d'un remplissage (montage) ou vidage (démontage) du baguenaudier.

Écrivez un constructeur à un paramètre d'entier `n` spécifiant le nombre de cases, qui initialise le nombre de pions ainsi que la taille du baguenaudier à `n` et la liste des valeurs à vide.

Écrivez un accesseur interne `getN` du nombre de pions.

Écrivez une méthode interne `initialiser` qui réinitialise la tablette à `Faux` avant de lancer le jeu.

Écrivez une méthode interne `afficher` qui visualise la tablette en affichant une astérisque ($*$) s'il y a un pion dans la case `k` et un point (.) sinon.

Écrivez une méthode interne `eval` qui calcule et renvoie l'évaluation du baguenaudier selon la méthode de Horner en base binaire.

Écrivez une méthode interne `afficherValeurs` qui affiche la liste des valeurs générées.

Validez vos méthodes avec la solution.

Solution C++ @[Baguenaudier.cpp]

```
/**
  Constructeur normal
  @param[in] n - nombre de pions =  taille du baguenaudier
*/
Baguenaudier::Baguenaudier(int n)
: m_npions(n), m_b(n+1), m_vals()
{}
```

```
/**
  Accesseur
  @return le nombre de pions
*/
inline int Baguenaudier::getN() const
{
  return m_npions;
}
```

```
/**
  Reinitialise la tablette avant de lancer le jeu
*/
void Baguenaudier::initialiser()
{
  fill(m_b.begin(), m_b.end(), false);
}
```

```
/**
  Affiche le baguenaudier
*/
void Baguenaudier::afficher() const
{
  for (int k = 1; k <= getN(); ++k)
  {
    cout<<(m_b[k] ? '*' : '.')<<" ";
  }
  cout<<endl;
}
```

```
/**
  Valuation du baguenaudier (method d Hornr n base binaire)
  @return l'Evaluation du baguenaudier
*/
int Baguenaudier::eval() const
{
  int v = 0;
  for (int k = getN(); k >= 1; --k)
  {
    v = v * BASE + static_cast<int>(m_b[k]);
  }
  return v;
}
```

```
/**
  Affiche la liste des valeurs generees
*/
void Baguenaudier::afficherValeurs() const
{
  cout<<"Nombre de coups = "<<m_vals.size()<<endl;
  copy(m_vals.begin(), m_vals.end(), ostream_iterator<int>(cout, " "));
  cout<<endl;
}
```

3 Stratégie du jeu

3.1 Analyse

« Jouer » c'est ôter ou placer un pion sur une case selon que celle-ci est ou non remplie. L'unique règle : à chaque coup les seules cases **jouables** sont la première case ou la case qui suit la première case occupée. Ce problème définit les deux types de jeux :

- **Mettre** n pions sur le baguenaudier initialement vide.
- **Prendre** n pions du baguenaudier initialement plein.

En supposant savoir placer p pions ($p \leq n-1$) sur un baguenaudier $b[1..p]$, comment placer le pion n ?

Solution simple
Celui-ci ne peut être joué que s'il suit le premier pion placé sur le jeu. Il faut donc que le jeu contienne un unique pion, le pion $b[n-1]$ et se ramener à cette situation. Ainsi :

- Pour $n = 0$: ne rien faire (cette remarque permet l'initialisation).
- Pour $n = 1$: placer le premier pion (c'est toujours possible).
- Pour $n \geq 2$:
  1. Il faut commencer par en placer $n-1$ (sur $b[1..n-1]$) (situation (o)).
  2. Puis ôter les $n-2$ premiers pions ce qui conduit à la situation où un unique pion est en $b[n-1]$ (a).
  3. Ceci permet de placer un pion sur la n-ème case car c'est maintenant la case qui suit la première case occupée (b).
  4. Enfin replacer les $n-2$ pions dans le baguenaudier réduit $b[1..n-2]$ ce qui met finalement n pions dans b (situation (c)).

	1	2	…	n-2	n-1	n
(o)	•	•	•	•	•	
(a)					•	
(b)					•	•
(c)	•	•	•	•	•	•

Cette analyse se traduit par une procédure récursive.

En supposant savoir prendre p pions ($p \leq n-1$) d'un baguenaudier $b[1..p]$, comment prendre le pion n ?

Solution simple

Pour la prise, il suffit de suivre l'analyse précédente dans l'autre sens.

- Pour $n = 0$: ne rien faire.
- Pour $n = 1$: ôter le premier pion (c'est toujours possible).
- Pour $n \geq 2$:
  1. Il faut d'abord ôter $n - 2$ pions, ce qui amène la situation (a).
  2. Puis ôter le dernier pion, ce qui conduit à la situation (b).
  3. Replacer ensuite les $n - 2$ pions du baguenaudier $b[1..n - 2]$ (c) : les $n - 1$ premières cases de b sont donc les seules occupées, comme dans le cas initial de `Mettre`.
  4. Enfin ôter ces $n - 1$ pions.

	1	2	...	n-2	n-1	n
(o)	•	•	•	•	•	•
(a)					•	•
(b)					•	
(c)	•	•	•	•	•	

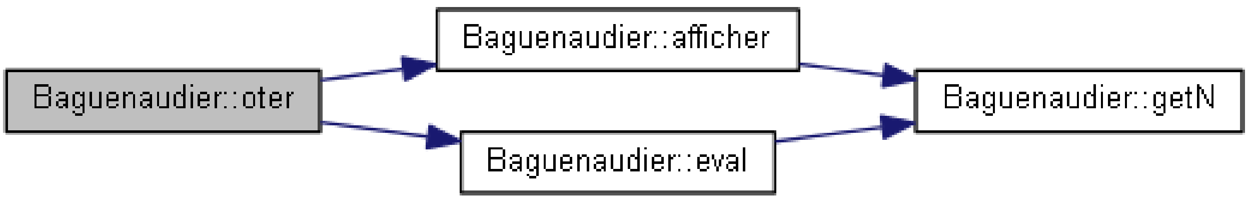
Cette analyse se traduit par une procédure récursive.

3.2 Algorithmique, Programmation

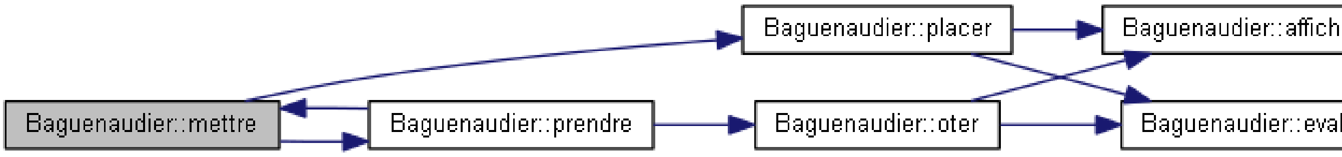
Ce problème réalise les stratégies de jeux.

Écrivez une méthode interne `placer(k)` qui place le pion d'indice `k` (entier) sur le baguenaudier puis insère son évaluation (eval) dans la liste des valeurs.

De même, écrivez une méthode interne duale `oter(k)` qui ôte le pion d'indice `k` (entier) du baguenaudier puis insère son évaluation (eval) dans la liste des valeurs.

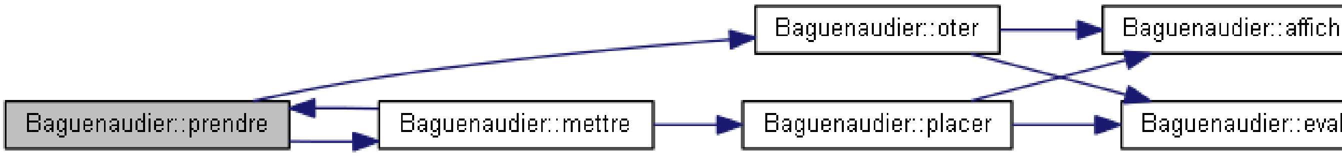


Écrivez une méthode interne **récursive croisée** `mettre(n)` qui met `n` (entier) pions sur le baguenaudier initialement vide.



Aide simple
Traduisez l'[Analyse].

De même, écrivez une méthode interne **récursive croisée** `prendre(n)` qui prend `n` (entier) pions du baguenaudier initialement plein.



Aide simple
Traduisez l'[Analyse].

Validez vos méthodes avec la solution.

Solution C++ @[Baguenaudier.cpp]

```
/**
  Place un pion sur le baguenaudier et insère son evaluation dans la liste
  @param[in] k - indice du pion
*/
void Baguenaudier::placer(int k)
{
  m_b[k] = true;
  afficher();
  m_vals.push_back(eval());
}
```

```
  Ote un pion du baguenaudier et insere son evaluation dans la liste
  @param[in] k - indice du pion
*/
void Baguenaudier::oter(int k)
{
  m_b[k] = false;
  afficher();
  m_vals.push_back(eval());
}

/**
```

```
  Met n pions sur le baguenaudier
  @param[in] n - nombre de pions a placer
*/
void Baguenaudier::mettre(int n)
{
  // si n == 0: ne rien faire
  if (n == 0)
  {
```

```
/**
  Met n pions sur le baguenaudier
  @param[in] n - nombre de pions a placer
*/
void Baguenaudier::mettre(int n)
{
  // si n == 0: ne rien faire
  if (n == 0)
  {
    return;
  }
  // si n == 1: placer l’unique pion 1
  else if (n == 1)
  {
    placer(n);
  }
  // sinon
  else
  {
    mettre(n-1);    // mettre n-1 pions
    prendre(n-2);   // prendre les n-2 premiers
    placer(n);      // placer le pion n
    mettre(n-2);    // mettre les n-2 premiers
  }
}
```

```
/**
  Prend n pions du baguenaudier
  @param[in] n - nombre de pions a prendre
*/
void Baguenaudier::prendre(int n)
{
  if (n == 0)
  {
    return;
  }
  else if (n == 1)
  {
    oter(n);
  }
  else
  {
    prendre(n-2);
    oter(n);
    mettre(n-2);
    prendre(n-1);
  }
}
```

Déduisez une méthode interne `remplir` qui remplit le baguenaudier initialement vide.

De même, déduisez une méthode interne `vider` qui vide le baguenaudier initialement plein.

Écrivez une méthode `jouer` qui initialise le baguenaudier, le remplit puis le vide.

Validez vos méthodes avec la solution.

Solution C++ @[Baguenaudier.cpp]

```
/**
  Remplit le baguenaudier initialement vide
*/
void Baguenaudier::remplir()
{
  cout<<"On remplit le baguenaudier..."<<endl;
  // reinitialise la liste des valeurs
  m_vals.clear();
  // affiche le jeu initial
  afficher();
  // evalue le premier jeu
  m_vals.push_back(eval());
  // met les pions sur le jeu
  mettre(getN());
  // liste des valeurs generees
  afficherValeurs();
}
```

```
/**
  Vide le baguenaudier initialement plein
*/
void Baguenaudier::vider()
{
  cout<<"On vide le baguenaudier..."<<endl;
  m_vals.clear();
  afficher();
  m_vals.push_back(eval());
  prendre(getN());
  afficherValeurs();
}
```

```
/**
  Lance le jeu
*/
void Baguenaudier::jouer()
{
  initialiser();
  remplir();
  system("PAUSE");
//  cout<<"Taper [Entree] pour vider le jeu...";
//  cin.ignore(10, '\n');
//  cout<<endl;
```

```
  vider();
}
```

Validez votre classe avec la solution.

Solution C++ @[Baguenaudier.hpp]

```
#ifndef BAGUENAUDIER_CLASS
#define BAGUENAUDIER_CLASS
#include <ostream>
#include <list>
#include <vector>
using namespace std;

/**
  Jeu du Baguenaudier
*/
class Baguenaudier
{
  public:
    explicit Baguenaudier(int n);
    void jouer();

  private:
    //== Methodes utilitaires
    int getN() const;
    void initialiser();
    void afficher() const;
    //== Poses et prises des pions
    void placer(int k);
    void oter(int k);
    void mettre(int n);
    void prendre(int n);
    void remplir();
    void vider();
    //== Evaluation des configurations
    enum { BASE = 2 };    // base binaire pour eval()
    int eval() const;
    void afficherValeurs() const;
    //== Attributs
    int m_npions;         // # de pions
    vector<bool> m_b;     // la tablette
    list<int> m_vals;     // liste des valeurs du jeu
};
#include "Baguenaudier.cpp"
#endif
```

Écrivez un programme qui saisit la taille du jeu dans un entier `n`, instancie un `Baguenaudier` puis lance le jeu pour `n|` pions.

Testez. Exemple d'exécution :

```
Taille du jeu? 4
On remplit le baguenaudier...
. . . .
* . . .
* * . .
. * . .
. * * .
* * * .
* . * .
. . * .
. . * *
* . * *
* * * *
Nombre de coups = 11
0 1 3 2 6 7 5 4 12 13 15
Appuyez sur une touche pour continuer...
On vide le baguenaudier...
* * * *
* . * *
. . * *
. . * .
* . * .
* * * .
. * * .
. * . .
* * . .
* . . .
. . . .
Nombre de coups = 11
15 13 12 4 5 7 6 2 3 1 0
```

Validez votre programme avec la solution.

Solution C++ @[pgbguedier.cpp]

```cpp
#include <iostream>
using namespace std;

#include "Baguenaudier.hpp"
int main()
{
  cout << "Taille du jeu? ";
  int n;
  cin >> n;
  Baguenaudier jeu(n);
  jeu.jouer();
}
```

4 Etude, Complexité, Décurryfication

4.1 Nombres générés

Chaque case d'un baguenaudier b ne peut prendre que deux états : libre ou occupée. En considérant que chaque $b[k]$ est équivalent à un chiffre binaire (0 pour libre et 1 pour occupé), chaque configuration est la représentation d'un entier en base 2.

Exemple

La configuration $b = [F, V, F, V, V, F, V, V]$ (où F est `Faux`, V est `Vrai`) représente l'entier 0b11011010 (valeur dans l'ordre inverse).

Remarque

Ce jeu engendre une suite de nombres et le passage d'un nombre au suivant s'effectue en changeant un unique chiffre binaire : c'est un **code de Gray**.

Quelle est la suite des nombres engendrés par le jeu `mettre` pour $n = 4$?

Solution simple

On a : 0b0001=1, 0b0011=3, 0b0010=2, 0b0110=6, 0b0111=7, 0b0101=5, 0b0100=4, 0b1100=12, 0b1101=13, 0b1111=15. Tous les nombres inférieurs à 8 ont été utilisés mais seulement 12, 13 et 15 pour ceux supérieurs ou égaux à 8.

Vérifiez qu'il y a 218 nombres générés pour $n = 8$.

Déterminez la suite engendrée par chacun des jeux.

Généralisez par récurrence.

4.2 Complexité du jeu

Ce problème étudie la complexité du jeu en nombre d'opérations, en supposant que les opérations `oter` et `placer` un pion nécessitent le même temps.

Soient $P(n)$ et $M(n)$ le nombre d'opérations effectuées par `prendre(n)` et `mettre(n)`. Écrivez les relations vérifiées par $P(n)$ et $M(n)$.

Solution simple

Le nombre d'opérations par `prendre(n)` et `mettre(n)` vérifient les relations :

$$\begin{cases} P(n) = P(n-2) + M(n-2) + P(n-1) + 1,\ \forall n \geq 2 \\ M(n) = M(n-1) + P(n-2) + M(n-2) + 1,\ \forall n \geq 2 \\ P(0) = M(0) = 0 \\ P(1) = M(1) = 1 \end{cases}$$

Écrivez la relation de récurrence de la suite A définie par :

$$\forall n \geq 0,\ P(n) = M(n) = A(n) + a \qquad (*)$$

Solution simple

La suite A vérifie alors la relation :

$$\begin{aligned} \forall n \geq 2,\ A(n) + a &= (A(n-2) + a) + (A(n-2) + a) + (A(n-1) + a) + 1 \\ &= 2A(n-2) + A(n-1) + 3a + 1 \end{aligned}$$

D'où :

$$\forall n \geq 2,\ A(n) = A(n-1) + 2A(n-2) + 2a + 1$$

Résolvez la relation pour que A soit linéaire.

Solution simple

Pour que A soit linéaire, il faut choisir a pour que $2a + 1 = 0$ d'où $a = -\frac{1}{2}$ et donc :

$$\begin{cases} A(n) = A(n-1) + 2A(n-2),\ \forall n \geq 2 \\ A(0) = P(0) - a = \frac{1}{2} \\ A(1) = P(1) - a = \frac{3}{2} \end{cases}$$

Le polynôme caractéristique associé à cette suite est :

$$Q(X) = X^2 - X - 2 = (X+1)(X-2)$$

et donc :

$$\forall n \geq 0,\ A(n) = \alpha 2^n + \beta(-1)^n \qquad (**)$$

où α et β désignent deux constantes. Pour $n = 0$ et $n = 1$, il résulte que :

$$\begin{cases} \alpha + \beta = \frac{1}{2} \\ 2\alpha - \beta = \frac{3}{2} \end{cases} \qquad (S)$$

Ce système admet une unique solution $(\alpha, \beta) = \left(\frac{2}{3}, \frac{-1}{6}\right)$. Compte tenu des relations (*) et (**) et du système (S), on a :

$$\forall n \geq 0,\ P(n) = M(n) = \frac{1}{6}\left(2^{n+2} - (-1)^n - 3\right) \in \mathbb{N}$$

Concluez en terme de complexité des procédures récursives.

Solution simple

Le nombre d'opérations est donc un $O(2^n)$. Comme le temps d'exécution est proportionnel au nombre d'opérations, les deux algorithmes sont de complexité temporelle exponentielle.

4.3 Décurryfication des procédures récursives

Ce problème (optionnel) réalise la décurryfication des procédures récursives.

Montrez que le jeu `mettre` est entièrement déterminé à chaque coup.

Aide simple

Vérifiez que le premier coup est imposé, puis utilisez le fait qu'un coup ne doit pas défaire ce qui a été fait au coup précédent pour constater que :

1. Le premier pion est joué un coup sur deux.
2. Les autres coups sont complètement déterminés.

Ainsi il n'y a aucun choix dans le jeu `mettre` : il est entièrement déterminé à chaque coup.

Solution simple

Pour le jeu `mettre` :

- Initialement il n'y a rien sur le jeu : la seule possibilité est de jouer la case $b[1]$.
- Au coup suivant, il est ridicule d'ôter le pion qui vient d'être placé, et la première case étant occupée il faut jouer sur la case $b[2]$. Il serait ridicule d'ôter le pion qui vient d'être mis : il faut donc jouer sur la case $b[1]$ (la vider).
- La première occupée est maintenant la case $b[2]$: ridicule de remettre en $b[1]$ le pion qui vient d'être ôté et donc il faut jouer sur la case $b[3]$, etc., aucun choix n'est donc possible.

De même, montrez que le jeu `prendre` est entièrement déterminé à chaque coup.

Solution simple

Pour le jeu `prendre` :

- Pour vider un jeu à une case, il faut ôter le pion $b[1]$.
- Pour prendre le jeu à deux cases, il n'est pas possible d'ôter le pion $b[1]$ (sinon il est alors impossible d'ôter le pion $b[2]$) et donc il faut d'abord ôter $b[2]$ puis $b[1]$.
- Pour le jeu à trois pions, la suite des coups est : ôter $b[1]$, ôter $b[3]$, placer $b[1]$, ôter $b[2]$ puis ôter $b[1]$.
- Si n est pair, il faut commencer par ôter le pion qui suit le premier pion placé sur le jeu et sinon (n impair) il faut commencer par ôter le pion $b[1]$.

Décurryfiez les procédures récursives.

5 Références générales

Comprend [] ■