ACT TP 5 – Heuristiques

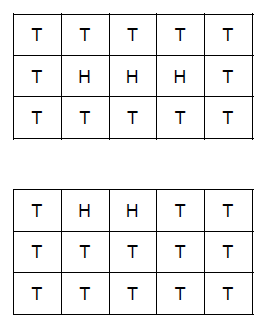
# Présentation du problème

## La pizza

Elle est représentée par une matrice h (hauteur) \* l (largeur)

Morceau de mozzarella : T

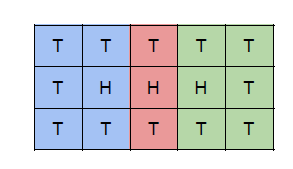
Morceau de jambon : H



## Les parts

Les parts royales :

* Sont rectangulaires, surface max au plus de c cases
* Doit contenir au moins n cases jambon



Part 3

Part 2

Part 1

# Certificat de validation

1. Solution candidate :

Une liste d’objets Part qui contiennent 4 entiers (ligne, colonne, ligneEnd, colonneEnd End) (les indices de la position de la part dans la pizza).

1. Taille :

Meilleur des cas : on a 1 seule part de pizza soit 4 indices => (ligneEnd – ligne + 1) \* ( colonneEnd – colonne + 1) < ( l \* h )

I = nbPart

Pire des cas : on a plusieurs parts de pizza => ∑ (ligneEnd i - lignei) \* (colonneEnd i - colonnei) < ( l \* h )

I = 0

Sinon en général la somme des surfaces des parts n’excède pas la taille de la pizza.

La taille du certificat est bien bornée polynomialement par la taille de l’entrée(la taille de la pizza)

1. Implémentation
   1. Certificat :

Classe Part {

Int ligne ;

Int colonne ;

Int ligneEnd ;

Int colonneEnd ;

}

* 1. Algorithme de vérification :

nbJambon = 0 ;

Pour i ∈ chaque part

Si nbJamboni <= nbJambon

OK

Si ∑ surfacePart <= surfacePizza => OK

1. Complexité :

On parcourt au plus chaque case de la pizza puisque les indices des parts de pizzas ne peuvent être identiques étant donné que 2 parts de pizza ne peuvent se chevaucher.

* Complexité en O (l \* h) 🡪 borné par la taille de l’entrée (la pizza)

1. Classe de complexité du problème :

* Certificat polynomiale par rapport à la taille de l’entrée
* Algorithme de vérification polynomiale par rapport à la taille de l’entrée
* Le problème de la pizza est NP.

# Serveur de validation

## Solution aléatoire

1.2. Implémentation de la génération aléatoire de la solution

1. On génère toutes les parts royales possibles ->list
2. On trie la liste aléatoirement -> ListTriee

Boolean [ List.size ] tableau = false ;

ListTriee ;

ListTriee.add(list(0)) ;

Tant que listTriee.size < list.size

Index = nombre aléatoire ;

Si ! tableau[index]

ListTriee.add(list(index)) ;

Tableau[index] = true ;

1. On ajoute la 1ère part de la liste triée
2. On parcourt la liste triée :

Si la part est compatible avec celles ajoutées -> on l’ajoute

1.3. On pourrait tomber sur la solution optimale mais on a des chances très faibles, on a 1 chances sur le nombre de combinaisons possibles.

Même si la 1ère part tirée au sort correspondait à l’une des parts de la solution optimale, on aurait toujours les mêmes chances soit 1 chances sur (n-1) de tirer au sort une autre part qui fait partie de la solution optimale et ainsi de suite jusqu’à la dernière part.

## Heuristique de pavage glouton

## Libre comme l’air

# Bonus : réduction de problème

# Super bonus : Complexité du problème