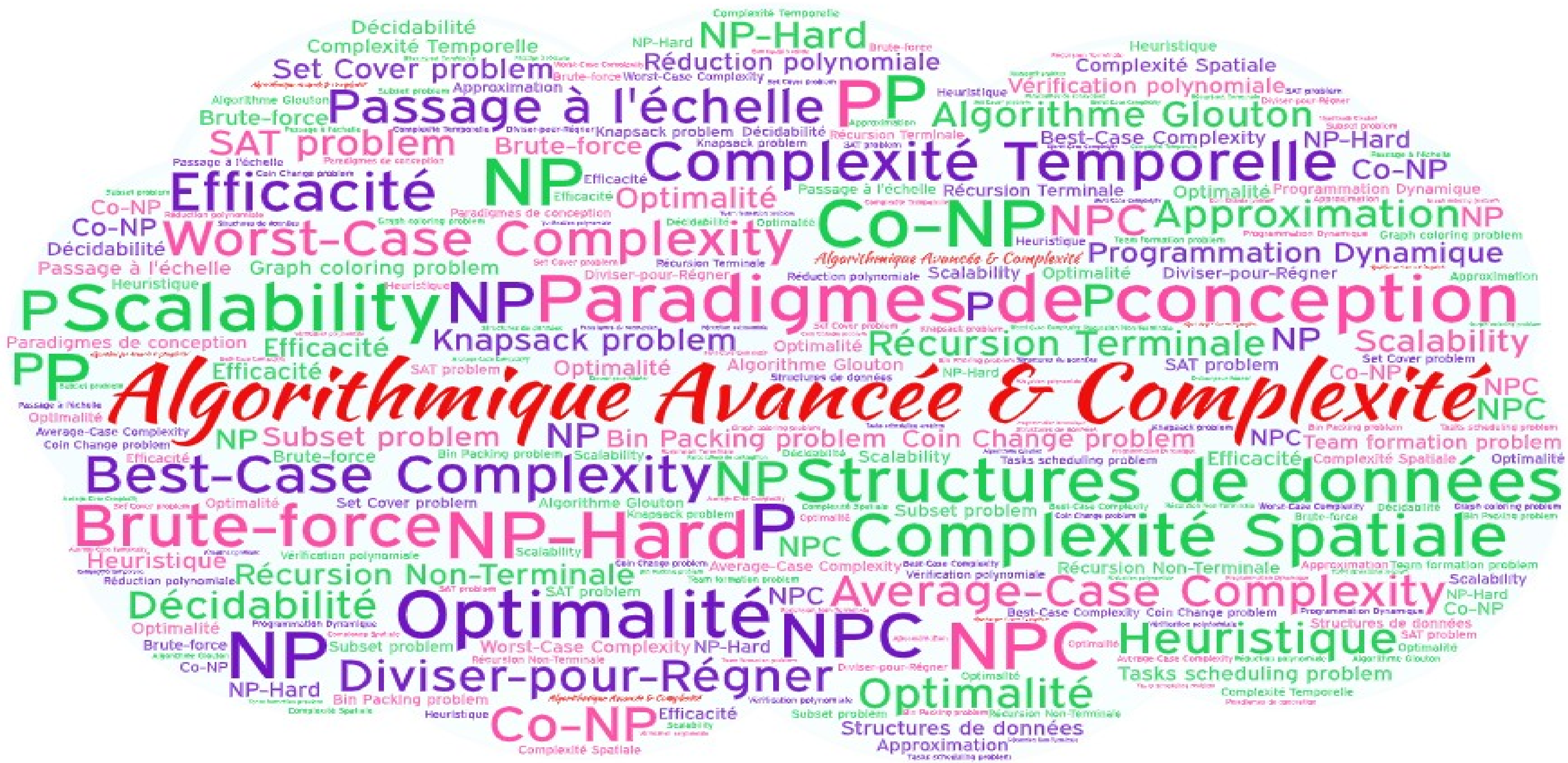


Algorithmes Gloutons (suite)

Dr. Mahfoud Houari
mahfoud.houari@gmail.com
Université Abou-Bakr Belkaïd - Tlemcen

2022/2023

Approche gloutonne – *Récapitulatif*

- Définir un ***choix glouton*** selon la nature du problème.
- Choisir des sous-solutions ***localement optimales*** pour arriver à une solution ***globalement optimale*****.**
- Analyser une ***seule branche*** qui mène à la solution optimale.
- **(+)** Plus efficace par rapport à la PRD.
- **(-)** Choix glouton souvent **difficile à prouver**
- **(-)** L'exactitude de la solution dépend du choix glouton :

  Choix optimal ⇒ ***Solution exacte***

  Choix non optimal ⇒ ***Solution approchée***

Problème du « *Sac à dos 0/1* »

Problème du « *Sac à dos 0/1* »

- Étant donné un sac de capacité maximale ***C***, un ensemble de ***N*** objets ayant chacun un poids p_i et un gain g_i.
- *Variante 0/1 :* veut dire que chaque objet soit il est prix (1) ou non (0).

Peut-on trouver une solution gloutonne pour cette variante du problème ?

Choix gloutons possibles ?

Problème du « *Sac à dos 0/1* »

- Étant donné un sac de capacité maximale C, un ensemble de N objets ayant chacun un poids p_i et un gain g_i.
- *Variante 0/1 :* veut dire que chaque objet soit il est prix (1) ou non (0).

Peut-on trouver une solution gloutonne pour cette variante du problème ?

Choix gloutons possibles :

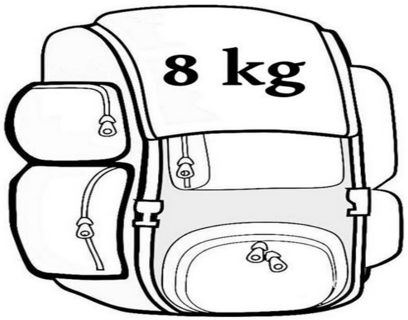
1. Considérer l'objet ayant le plus grand gain.
2. Considérer l'objet ayant le plus petit poids.
3. Considérer l'objet ayant la plus grande fraction g_i / p_i.

Problème du « *Sac à dos 0/1* »

1. Considérer l'objet ayant le plus grand gain : **(optimal ?)**

Contre-exemple pour ce choix:

1	2	3	4
15 da	10 da	9 da	5 da
1 kg	5 kg	3 kg	4 kg



Solution optimale : **1 + 3 + 4 ⇒ 29 da.**

Solution gloutonne selon le choix 1: **1 + 2 ⇒ 25 da.**

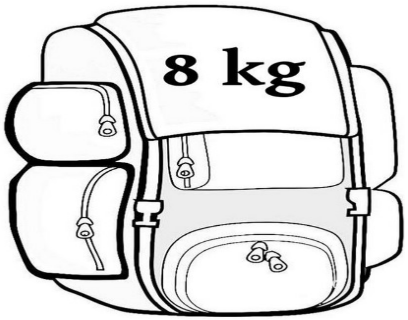
Choix glouton non optimal.

Problème du « *Sac à dos 0/1* »

2. Considérer l'objet ayant le plus petit poids : **(optimal ?)**

Contre-exemple pour ce choix:

1	2	3	4
15 da	15 da	9 da	5 da
1 kg	5 kg	3 kg	4 kg



Solution optimale : **1 + 2 ⇒ 30 da**.

Solution gloutonne selon le choix 2: **1 + 3 + 4 ⇒ 29 da**.

Choix glouton non optimal.

Problème du « *Sac à dos 0/1* »

3. Considérer l'objet ayant la plus grande fraction : **(optimal ?)**

Contre-exemple pour ce choix:

1	2	3
6 da	10 da	12 da
1 kg	2 kg	3 kg

Solution optimale : **2 + 3 ⇒ 22 da**.

Solution gloutonne selon le choix 3: **1 + 2 ⇒ 16 da**.

Choix glouton non optimal.

Problème du « *Sac à dos 0/1* » – *Conclusion*

- Le problème du Sac à dos 0/1 ne peut pas être résolu par une stratégie gloutonne (Il n'existe aucun choix optimal).

- Une solution approximative peut être implémentée selon un des choix non-optimaux trouvés.

Exercice :

Proposer une solution approximative selon le critère « *Choisir l'objet ayant le plus de gain* ».

Problème du « *Sac à dos 0/1* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
//Définir la classe MonObjet ayant comme valeurs poids et gain

ArrayList<MonObjet> SAD_Glouton(int C, MonObjet[] O){
    //trier en ordre décroissant le tableau O selon les gains
    TriDecroissant(O) ;
    ArrayList<MonObjet> Sol = new ArrayList<MonObjet>();
    for(int i = 0 ; i < O.length ; i++){
        if(O[i].poids <= C){
            Sol.add(O[i]);
            C = C – O[i].poids;
        }
    }
    return Sol;
}
```

WCTC : O(????).
WCSC : O(????).

Problème du « *Sac à dos 0/1* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
//Définir la classe MonObjet ayant comme valeurs poids et gain

ArrayList<MonObjet> SAD_Glouton(int C, MonObjet[] O){
    //trier en ordre décroissant le tableau O selon les gains
    TriDecroissant(O) ;
    ArrayList<MonObjet> Sol = new ArrayList<MonObjet>();
    for(int i = 0 ; i < O.length ; i++){
        if(O[i].poids <= C){
            Sol.add(O[i]);
            C = C – O[i].poids;
        }
    }
    return Sol;
}
```

WCTC : $O(|O|.\log_2(|O|)+|O|)$.

WCSC : $O(|O|)$.

Problème de « *Coloration de Graphe* »

Problème de « *Coloration de Graphes* »

Description : Pour n'importe quel graphe non-orienté, l'objectif est d'affecter une couleur à chacun de ses nœuds de telle sorte que les nœuds adjacents ne partagent pas la même couleur.

Objectif : Utiliser le minimum de couleurs.

Applications :

1) Affectations en groupes.

2) Planification des examens.

3) Solution du jeu Sudoku.

Peut-on trouver une solution gloutonne optimale ?

Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell :

1. Trier les nœuds en ordre décroissant selon leur degré.
2. Attribuer une couleur au premier nœud de la liste.
3. Attribuer la même couleur aux nœuds de la liste à condition que leurs adjacents ne la possèdent pas.
4. Répéter les étapes (2-3) pour les nœuds restants.

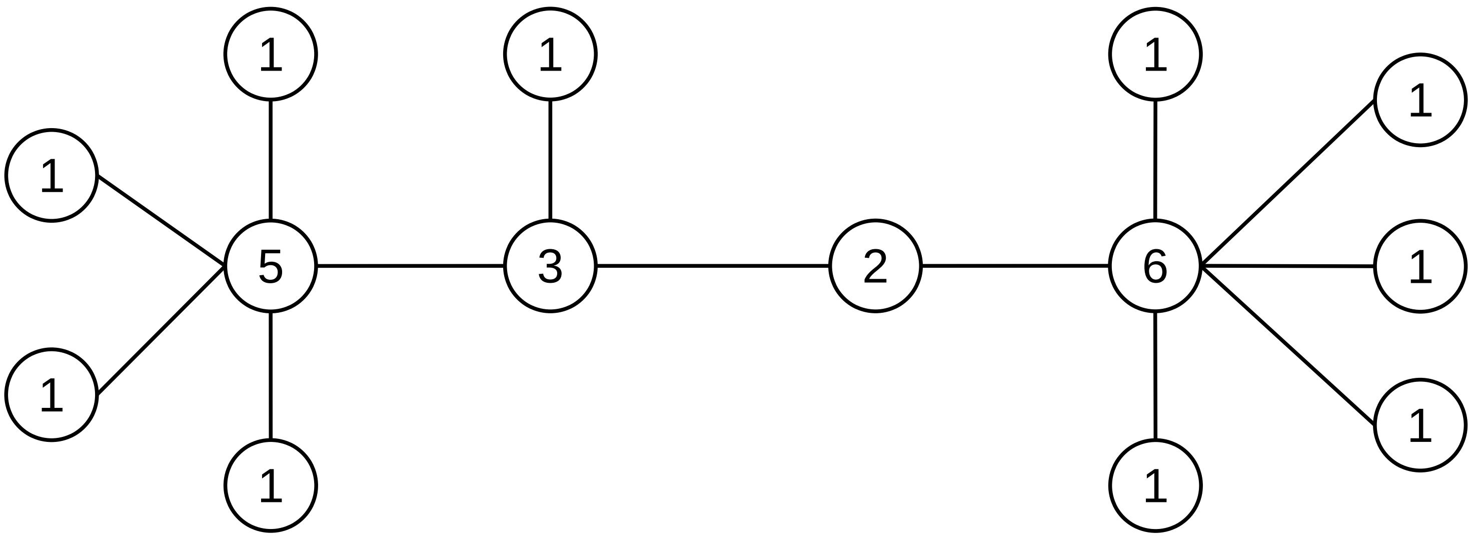
Optimalité ?

⇒ *L'algorithme n'est pas optimal.*

⇒ *Aucune solution efficace n'existe pour ce problème.*

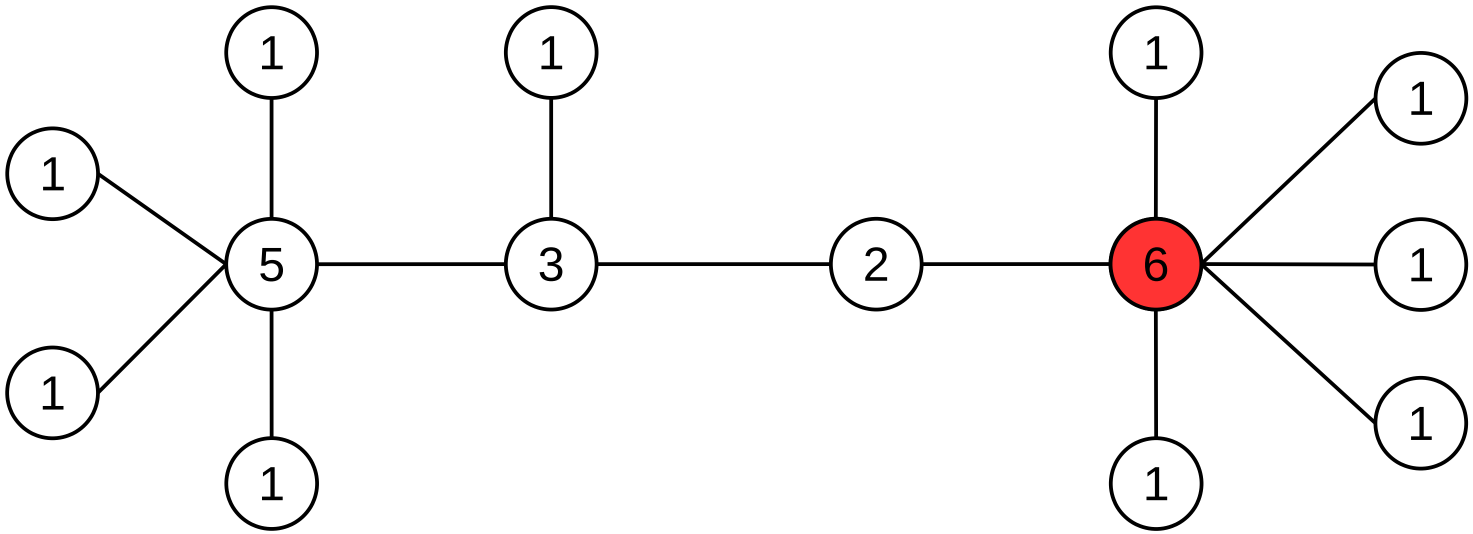
Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell – *Contre-exemple*



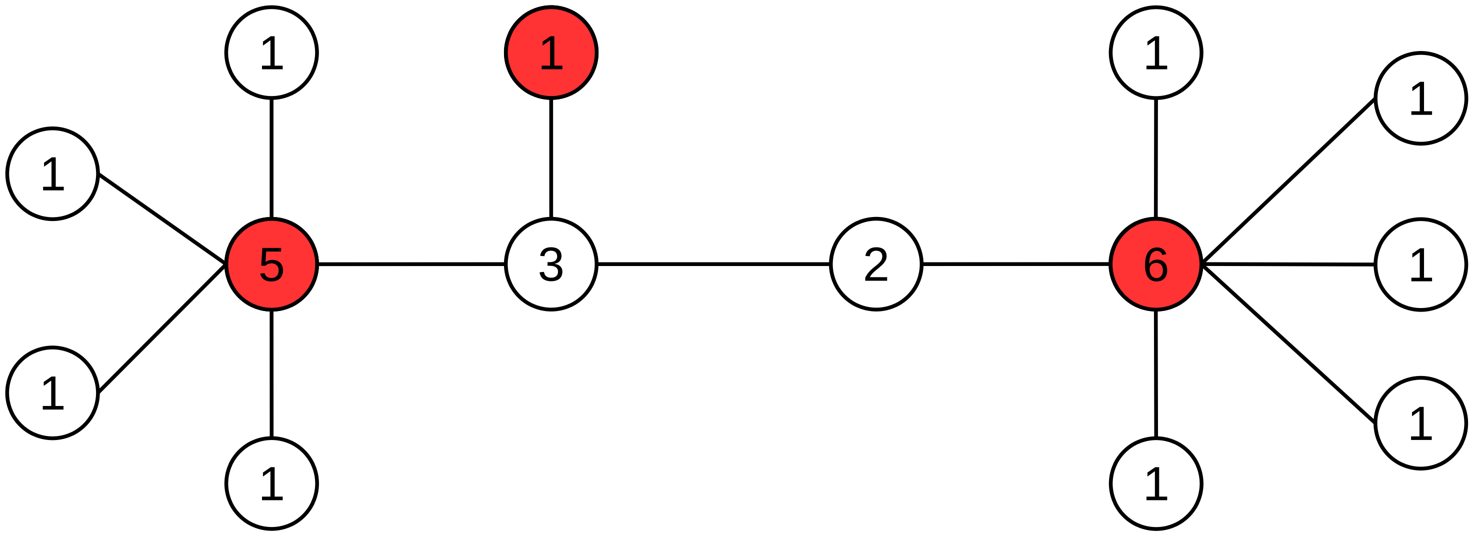
Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell – *Contre-exemple*



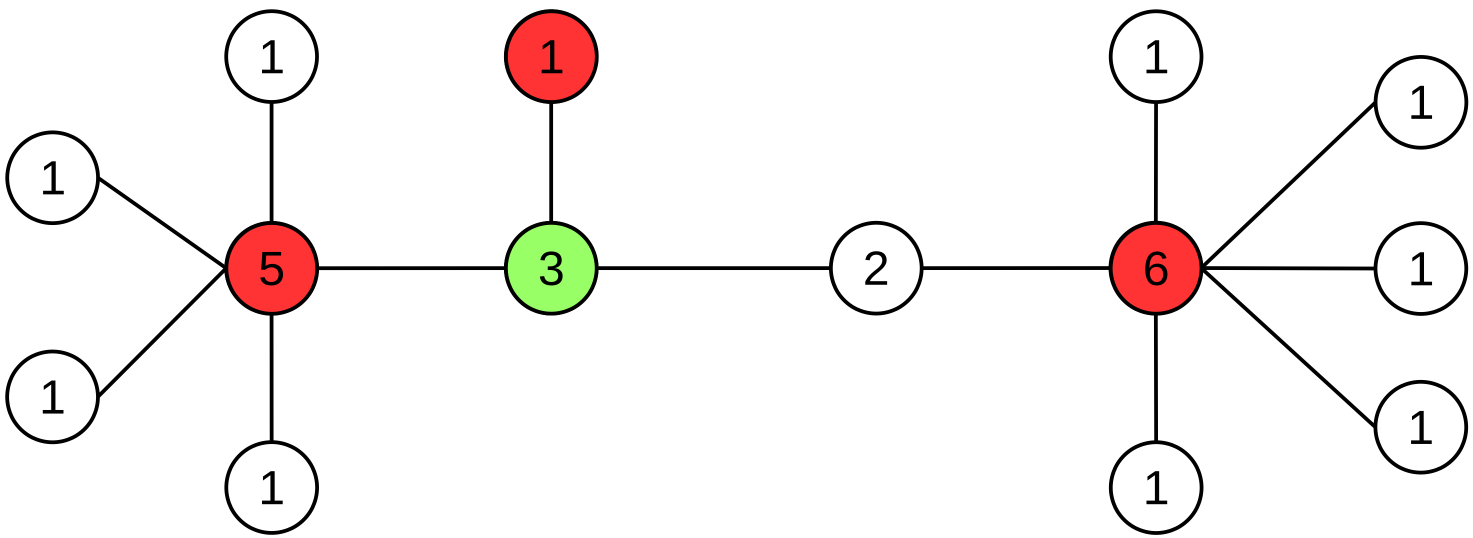
Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell – *Contre-exemple*



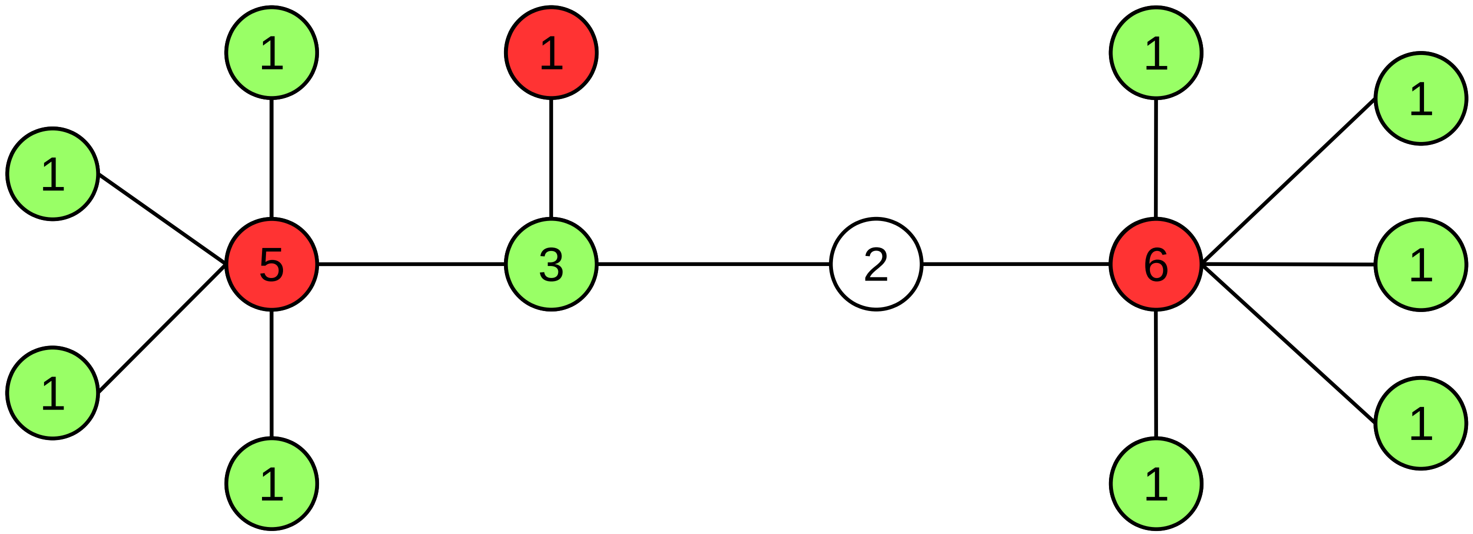
Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell – *Contre-exemple*



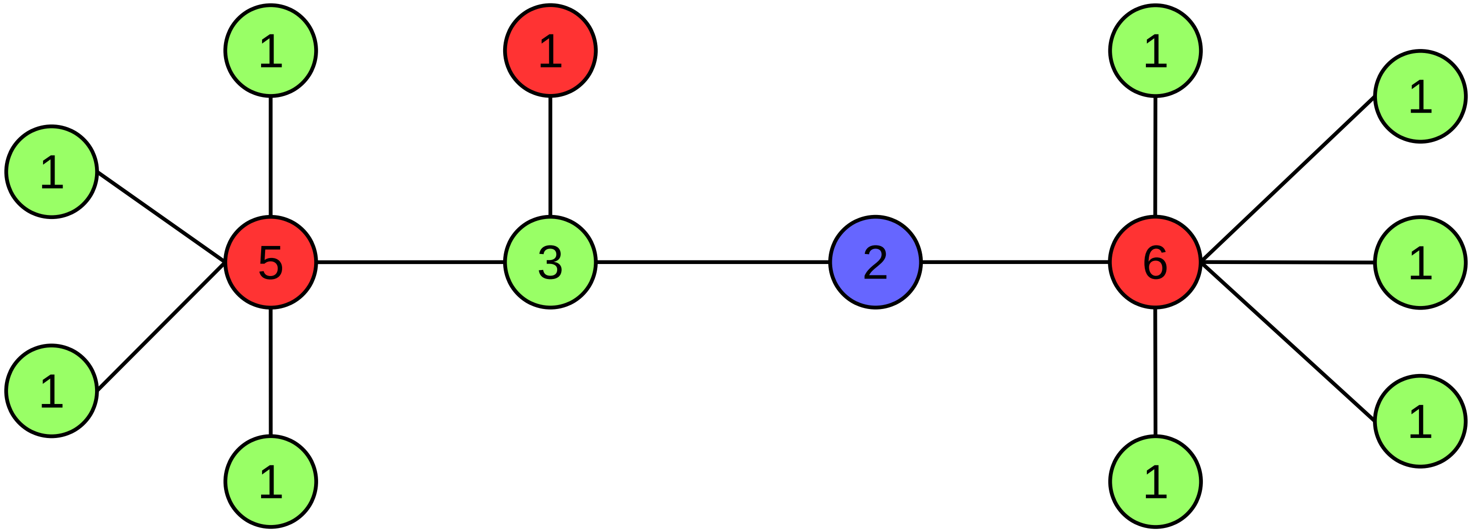
Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell – *Contre-exemple*



Problème de « *Coloration de Graphes* »

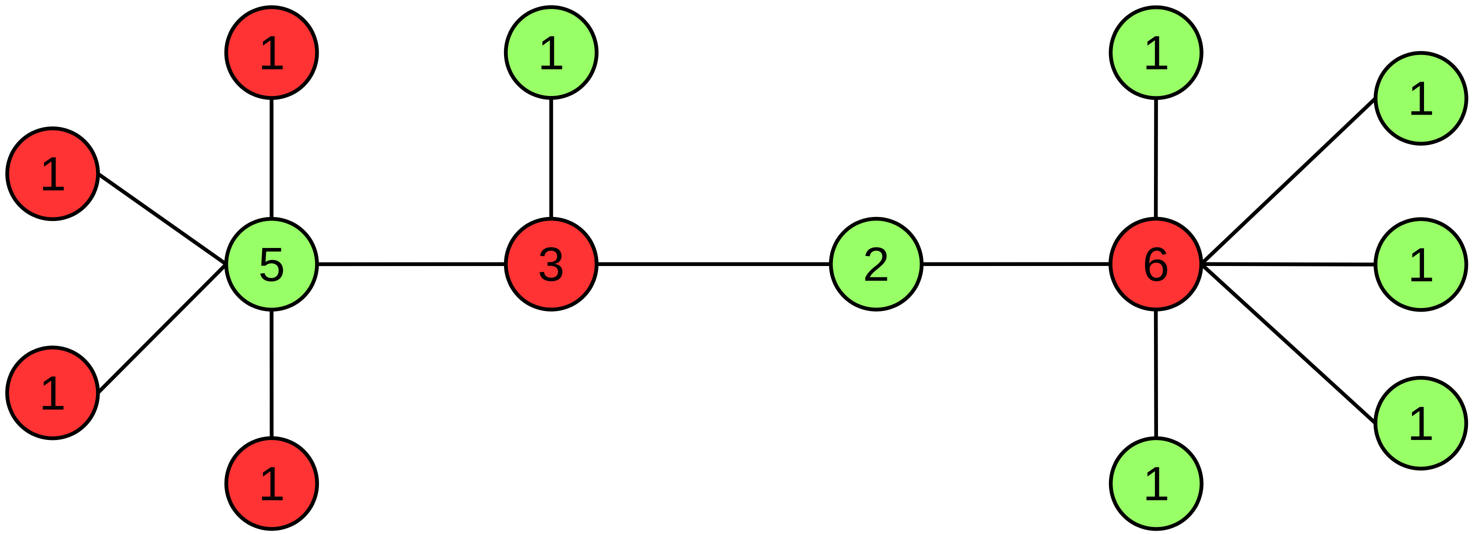
Algorithme de Welsh & Powell – ***Contre-exemple***



Solution trouvée par l'algorithme : *3 couleurs*

Problème de « *Coloration de Graphes* »

Algorithme de Welsh & Powell – *Contre-exemple*



Solution optimale : *2 couleurs.*

Problème du « *Bin Packing* »

(*Remplissage de boites*)

Description :

- Soit un ensemble de N objets de poids p_1, p_2, …, p_N tel que : $p_i > 0$.
- Un ensemble de M boites de même capacité C.

Objectif : Il s'agit de déterminer le nombre minimum de boites pour ranger tous les objets.

Existe-il une solution gloutonne pour ce problème ?

Choix gloutons possibles ?

Existe-il une solution gloutonne pour ce problème ?

Choix gloutons possibles :

1. Considérer l'objet ayant le plus grand poids.
2. Considérer l'objet ayant le plus petit poids.

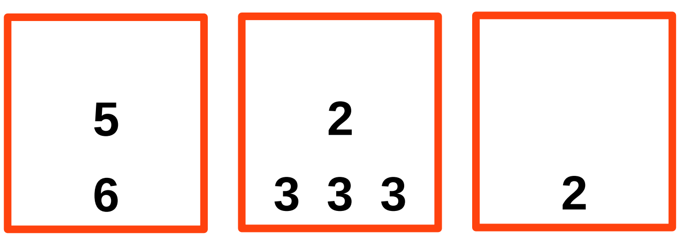
Problème du « *Bin Packing* »

1. Considérer l'objet ayant le plus grand poids :

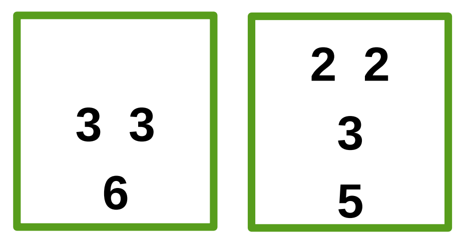
Contre-exemple pour ce choix:

Les poids des objets : 2 2 3 3 3 5 6

Capacité des boites : 12



Solution gloutonne



Solution optimale

Choix glouton non optimal.

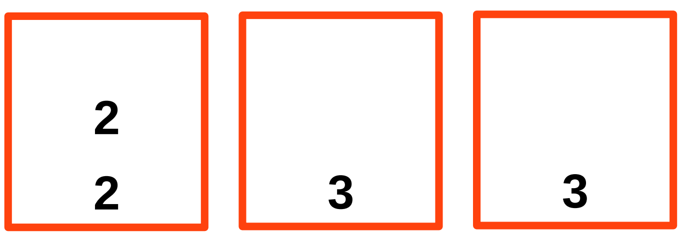
Problème du « *Bin Packing* »

2. Considérer l'objet ayant le plus petit poids :

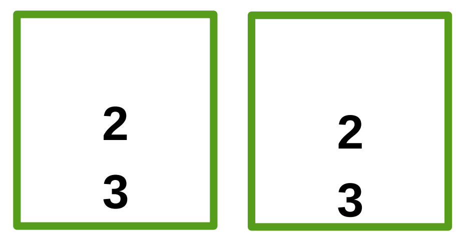
Contre-exemple pour ce choix:

Les poids des objets : 2 2 3 3

Capacité des boites : 5



Solution gloutonne



Solution optimale

Choix glouton non optimal.

Problème du « *Bin Packing* » – *Conclusion*

- Aucun critère glouton optimal ne peut être trouvé pour ce problème.

- Il s'agit d'un problème NP-complet (*à voir plus tard*) qui n'accepte pas une solution praticable.

Problème du « *Bin Packing* » – *Implémentation*

Exercice :

- Proposez une méthode approchée qui implémente le deuxième critère (objet ayant le plus grand poids).

- Calculez la WCTC et WCSC de la méthode proposée.

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
class Objet {
    int poids ;
    public Objet(int p){  this.poids = p ;  }
}

class Boite {
    int C ;
    ArrayList<Objet> objets ;
    public Boite(int c){
        this.C = c ;
        this.objets = new ArrayList<Objet>() ;
    }
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
	TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
	ArrayList<Boite> Sol = new ArrayList<Boite>();










	return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
    TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
    ArrayList<Boite> Sol = new ArrayList<Boite>();
    for(int i = 0 ; i < O.length ; i++){
        boolean range = false ;








    }
    return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
    TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
    ArrayList<Boite> Sol = new ArrayList<Boite>();
    for(int i = 0 ; i < O.length ; i++){
        boolean range = false ;
        for(int j = 0 ; j < Sol.size() ; j++){




        }



    }
    return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
    TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
    ArrayList<Boite> Sol = new ArrayList<Boite>();
    for(int i = 0 ; i < O.length ; i++){
        boolean range = false ;
        for(int j = 0 ; j < Sol.size() ; j++){
            if(Sol.get(j).C >= O[i].poids){




            }
        }




    }
    return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
    TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
    ArrayList<Boite> Sol = new ArrayList<Boite>();
    for(int i = 0 ; i < O.length ; i++){
        boolean range = false ;
        for(int j = 0 ; j < Sol.size() ; j++){
            if(Sol.get(j).C >= O[i].poids){
                Sol.get(j).C -= O[i].poids ;
                Sol.get(j).objets.add(O[i]) ;
                range = true ; break ;
            }
        }



    }
    return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
	TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
	ArrayList<Boite> Sol = new ArrayList<Boite>();
	for(int i = 0 ; i < O.length ; i++){
		boolean range = false ;
		for(int j = 0 ; j < Sol.size() ; j++){
			if(Sol.get(j).C >= O[i].poids){
				Sol.get(j).C -= O[i].poids ;
				Sol.get(j).objets.add(O[i]) ;
				range = true ; break ;
			}
		}
		if(! range){

		}
	}
	return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
	TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
	ArrayList<Boite> Sol = new ArrayList<Boite>();
	for(int i = 0 ; i < O.length ; i++){
		boolean range = false ;
		for(int j = 0 ; j < Sol.size() ; j++){
			if(Sol.get(j).C >= O[i].poids){
				Sol.get(j).C -= O[i].poids ;
				Sol.get(j).objets.add(O[i]) ;
				range = true ; break ;
			}
		}
		if(! range){
			Boite B = new Boite(C – O[i].poids) ; B.objets.add(O[i]) ; Sol.add(B) ;
		}
	}
	return Sol;
}
```

Problème du « *Bin Packing* » – *Approximation*

Algorithme glouton (*Solution Approximative*):

```
ArrayList<Boite> RB_Glouton(int C, Objet[] O){
	TriDecroissant(O) ; //trier en ordre décroissant le tableau O selon les poids
	ArrayList<Boite> Sol = new ArrayList<Boite>();
	for(int i = 0 ; i < O.length ; i++){
		boolean range = false ;
		for(int j = 0 ; j < Sol.size() ; j++){
			if(Sol.get(j).C >= O[i].poids){
				Sol.get(j).C -= O[i].poids ;
				Sol.get(j).objets.add(O[i]) ;
				range = true ; break ;
			}
		}
		if(! range){
			Boite B = new Boite(C – O[i].poids) ; B.objets.add(O[i]) ; Sol.add(B) ;
		}
	}
	return Sol;
}
```

WCTC : O(????).

WCSC : O(????).

Exercice n°01:

Étant donné un ensemble de candidats qui doivent passer à l'administration d'une entreprise pour déposer leurs dossiers. Sachant que chaque candidat précise un intervalle d'horaire durant lequel il peut passer, l'objectif est de ***trouver le nombre minimum d'horaires*** durant lesquels le bureau de candidature doit être ouvert pour recevoir les dossiers.

Exemple :

Pour les horaires **[14h,16h],[9h, 12h],[8h,10h],[15h,17h]**

Solutions optimales possibles (avec 02 horaires):

Sol1 : Ouvrir le bureau à **10h** et à **16h**.

ou

Sol2 : Ouvrir le bureau à **09h** et à **15h**.

Procédez à une résolution gloutonne pour ce problème

Exercice n°02:

Un bureau d'expert a reçu plusieurs demandes d'expertise telle que chacune est caractérisée par un gain et une durée. Sachant que l'expert peut travailler uniquement dans un intervalle horaire **[D,F]**, l'objectif est de ***trouver les expertises à réaliser afin de maximiser le gain total*** et ce tout en respectant l'intervalle de l'expert.

Exemple :

Pour l'expert : **[8h,12h]**

Pour les demandes: **[2h,120 DA], [4h,150 DA], [1h, 100 DA]**.

La solution optimale consiste à prendre deux demandes **[2h,120 DA] et [1h, 100 DA]** pour gagner finalement 220 DA.

Procédez à une résolution gloutonne pour ce problème

Paradigme « *Branch & Bound* »

(*B&B*)

Principe :

- C'est un paradigme de résolution exacte.
- C'est une amélioration de l'approche exhaustive.
- Similairement à l'approche exhaustive, on crée un arbre des appels récursifs contenant des branches (***Branch***).
- L'idée est d'arrêter l'exploration des branches qui ne mènent pas à une solution optimale (***Pruning***).
- Une borne optimale (***Bound***) est attribuée à chaque nœud de l'arbre.
- L'exécution est faite en largeur en sélectionnant le nœud ayant la plus grande borne.
- Les bornes peuvent être calculées via un critère glouton.
- En pratique, le paradigme assure de très bons résultats.
- Théoriquement, une WCTC exponentielle peut être atteinte.

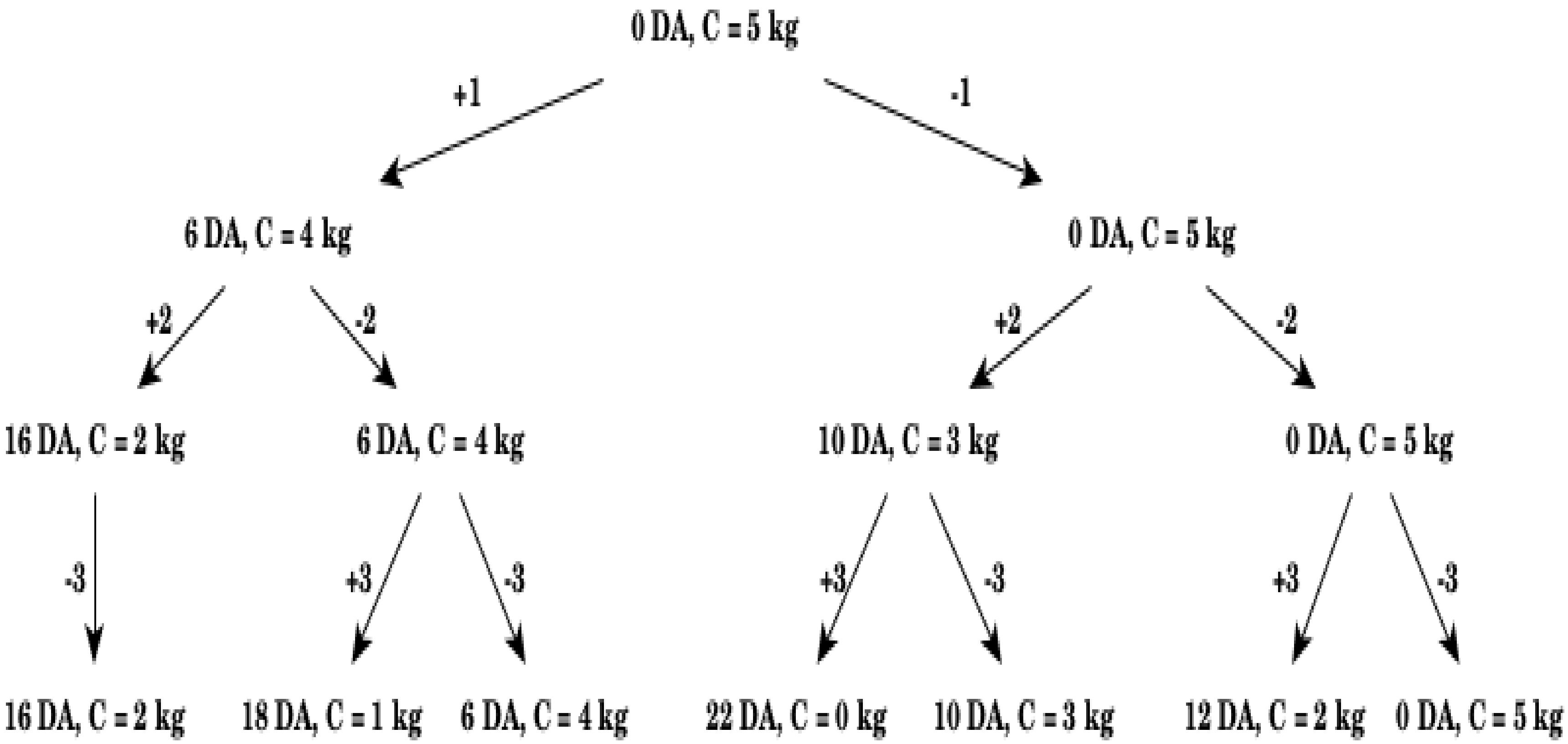
Exemple d'application: Problème de *Sac à dos*

Voici une instance du ***Sac à dos 0/1*** :

1	2	3
6 da	10 da	12 da
1 kg	2 kg	3 kg

Exemple d'application: Problème de *Sac à dos*

Résolution exhaustive :

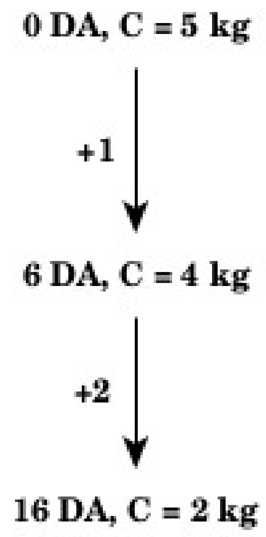


Solution optimale : Objets **2** et **3** ⇒ gain de **22 da**.

Appels récursifs : 14

Exemple d'application: Problème de *Sac à dos*

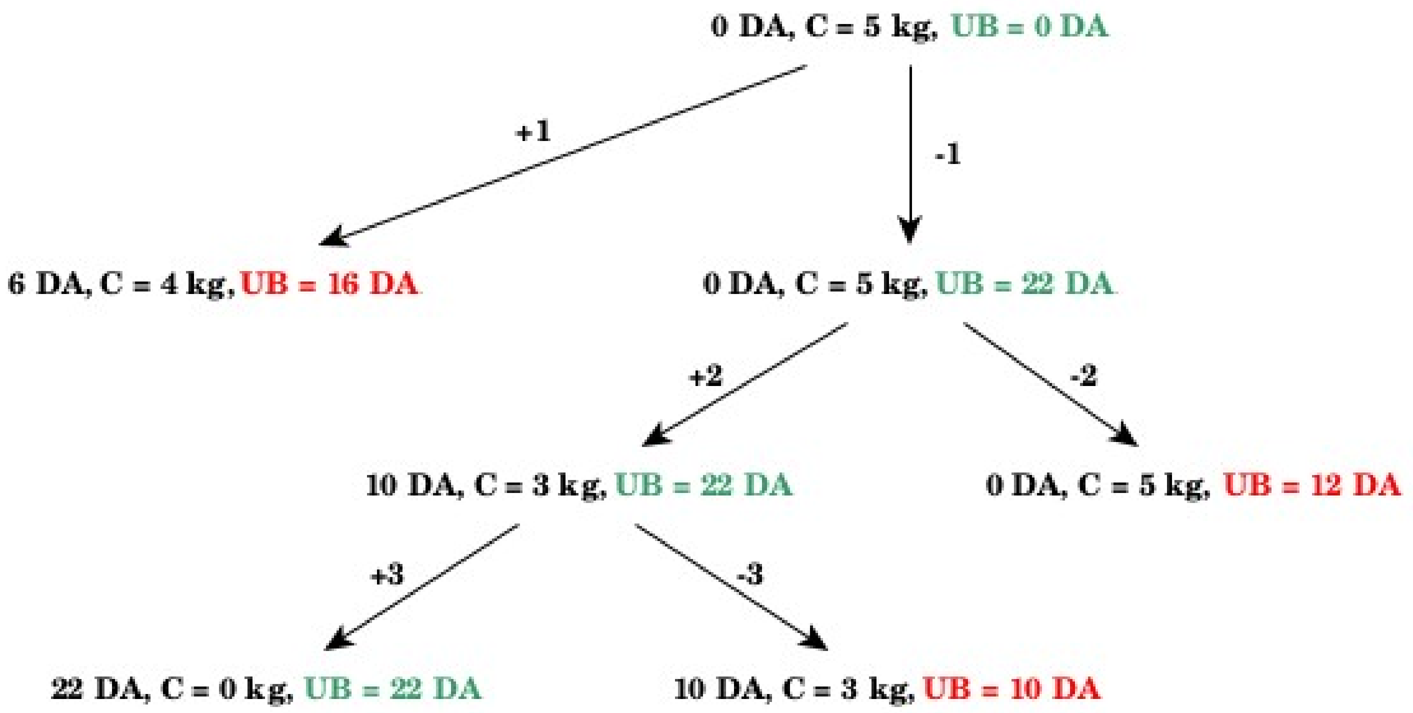
Résolution gloutonne :



Solution non optimale : Objets **1** et **2** ⇒ gain de **16 da**.
Appels récursifs : **03**

Exemple d'application: Problème de *Sac à dos*

Résolution Branch & Bound :



Solution optimale : Objets **2** et **3** ⇒ gain de **22 da**.

Appels récursifs : **07**