# Synthèse : Algorithmique

1. Introduction :

Un algorithme c’est une suite d’instruction qui permet de résoudre un problème de manière systématique.

Un algorithme est conçu pour répondre à un problème spécifique.

Quelques notions importantes à retenir :

* **Terminaison :**

S’assurer que l’algorithme terminera en un temps fini.

* **Correction :**

S’assurer que le résultat fournit par l’algorithme est la solution au problème.

* **Complétude :**

S’assurer que l’algorithme soit complet face au problème, qu’il puisse donner un résultat correct peu importe l’entrée.

L’idée c’est que l’algorithme puisse répondre à toutes entrées possibles liés au problème de base.

1. Complexité :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

* Classés dans l’ordre croissant. (De haut en bas)
* « n » correspond à la taille de l’entrée.

Il existe deux types de complexité :

* **Spatiale :**

Dans ce cas-ci, il s’agit de l’espace mémoire nécessaire à notre algorithme pour exécuter le calcul.

(Il s’agit généralement de la RAM)

* **Temporelle :**

Dans ce cas-ci, il s’agit du temps nécessaire à notre algorithme pour exécuter le calcul.

1. Les tris :

3.1 Tri à bulle :

1) Parcourir la liste

2) Comparer l’élément avec le suivant, si il est plus petit on change leur position

**Complexité :**

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

3.2 Tri à insertion :

1)Parcourir la liste

2) Comparer à l’élément précédent

3) Déplacer l’élément le plus grand (à droite)

**Complexité :**

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

3.3 Tri par fusion :

Diviser pour mieux régner. Cette méthode de tri, utilise ce qu’on appelle la récursivité.

1)Si la liste n’a qu’un élément, elle est triée

2) Séparer la liste en deux listes +/- égales

3)Trier chacune des listes selon le tri par fusion

4)Fusionner les 2 listes en une seule liste triée

**Complexité :**

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

3.4 Tri rapide :

Cette méthode de tri, utilise également la récursivité et la décomposition d’une liste principale en listes secondaires.

1)Choisir un pivot (par convention on prend le dernier élément)

2)Placer les éléments plus grands au pivot dans une liste + le pivot lui-même

3)Placer les éléments plus petits au pivot dans une liste

4)Lorsque la liste n’a qu’un seul élément alors elle est triée

5)On fusionne toutes les listes secondaires

**Complexité :**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

1. Structure de données :

Il y a deux types de structure de données :

* **Tableau :**

Un tableau contient des éléments qui sont accessibles grâce à un « index ». Le premier élément du tableau aura pour index « 0 ».

* **Liste chaînée :**

Une liste chaînée, est une structure différente. Il n’est pas possible d’aller chercher un élément par un index ou autre.

Pour atteindre un élément, il faudra passer par chaque élément qui le précède.

Illustration :

* **Tableau :**

**Une image contenant capture d’écran, ligne, diagramme, Police

Description générée automatiquement**

* **Liste chaînée :**

Une image contenant Police, diagramme, ligne, Graphique

Description générée automatiquement

N.B : dans les tableaux, on parle d’éléments de tableaux. Pour les listes chaînées, il vaut mieux parler de nœud. Dans chaque nœud, il y a la donnée + le lien vers le prochain nœud de la liste.

Il existe également des listes doublement chaînées, etc. (Revoir Q1).

1. Les arbres :

Une image contenant texte, Police, capture d’écran, diagramme

Description générée automatiquement

5.1 Vocabulaire en lien avec l’illustration :

* **Racine / Root :**

Situé tout au-dessus de l’arbre, on parle alors de nœud initial.

* **Nœud / Node :**

Élément d’un arbre qui contient une valeur, ainsi qu’une référence (lien) vers d’autres nœuds.

* **Feuille / Leaf :**

Élément en bout d’arbre, il n’a pas d’enfant.

* **Parent :**

Nœud qui fait référence à d’autre nœud (enfant). (Le(s) enfants sont en-dessous)

* **Enfant / Child :**

Nœud qui provient d’un autre nœud, qui a été référencé dans un nœud (parent). (Le parent est au-dessus)

* **Frères / Siblings :**

Ce sont des nœuds qui ont le même parent. Sur l’illustration « Enfant » et « Enfant » sont frères.

5.2 Notion de hauteur et profondeur :

* **Hauteur :**

Lorsque l’on parle de hauteur on parle de l’arbre en lui-même, sa hauteur correspond au chemin le plus long entre la « Racine » et la « feuille » la plus éloigné.

* **Profondeur :**

Lorsque l’on parle de profondeur, on parle d’un nœud et non de l’arbre.

La profondeur d’un nœud, correspond au nombre d’arrête entre lui et la « Racine ».

* **Exemple visuel :**

Hauteur de l’arbre :

3

Profondeur de A :

1

Profondeur de B :

2

Profondeur de D :

3

ATTENTION :

Il est possible que la hauteur ainsi que la profondeur change en fonction de la convention utilisée.

La racine peut être à 0 ou 1, et donc forcément tout change.

Une image contenant capture d’écran, horloge, conception

Description générée automatiquement

5.3 Infos supplémentaires :

Les arbres permettent une représentation hiérarchique des données.

1. Les arbres binaires :

6.1 Définition :

Un arbre binaire, est un arbre avec une racine dans lequel chaque nœud possède aux plus 2 fils.

6.2 Méthode de parcours :

Un arbre binaire peut-être parcouru de diffèrente manière.

On retrouve deux catégories :

1. En profondeur :

Dans cette même catégorie on en retrouve 3 différentes.

* 1. Préfixe :

Une image contenant croquis, dessin, cercle, diagramme

Description générée automatiquement

* 1. Postfixe :

Une image contenant croquis, cercle, diagramme, dessin

Description générée automatiquement

* 1. Infixe :

Une image contenant croquis, cercle, dessin, diagramme

Description générée automatiquement

1. En largeur

Une image contenant croquis, cercle, dessin, diagramme

Description générée automatiquement

1. Transformer un arbre n-aire en arbre binaire :

Pour réaliser cette transformation, suivre ce processus :

Partir de la racine, chaque enfant sera mis sur la gauche et chaque frère sera mis sur la droite.

Exemple :

Une image contenant diagramme, ligne, conception

Description générée automatiquement

🡺 En rouge les frères

🡺 En vert les enfants

1. Arbre binaire de recherche :

8.1 Définition :

Les arbres binaires de recherches (ABR), sont des arbres binaires avec une spécialisation.

De manière simplifié, pour chaque nœud, le fils droit doit être supérieur au nœud parent. Et le fils gauche doit être inférieur au nœud parent.

8.2 Opérations possibles :

* **Rechercher :**

1. Arbre vide => « nœud pas trouvé »
2. Si la valeur = nœud sur lequel on se trouve on renvoie le nœud
3. Si la valeur est plus petite que le nœud sur lequel on se trouve, on l’envoie à gauche, sinon à droite.

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

* **Insérer :**

Concernant l’insertion, on va utiliser dans un premier temps une recherche « fictive ».

On va donc se déplacer dans l’arbre en respectant la spécificité d’un ABR, et lorsque l’on arrive à un endroit où il n’y a pas de nœud, on ajoute la valeur à notre arbre.

* **Supprimer :**

Il y a 3 possibilités :

1. Suppression d’une feuille
2. Suppression d’un nœud qui a 1 enfant
3. Suppression d’un nœud qui a 2 enfants

Lorsque l’on supprime un élément, il faut le remplacer si l’arbre continue

après le nœud concerné.

Deux possibilités :

1. Par son successeur :

Il s’agit du plus petit du côté droit.

Une image contenant capture d’écran, cercle, diagramme

Description générée automatiquement

1. Par son prédécesseur :

Il s’agit du plus grand du côté gauche.

Une image contenant capture d’écran, diagramme, cercle

Description générée automatiquement

8.3 Notion d’équilibrage :

L’équilibrage est important, son principe est simple, essayer de maintenir une profondeur équilibrée entre ses branches.

Cela évite d’avoir certains chemins d’une longueur disproportionnée.

(Lorsqu’un arbre est déséquilibré, on peut parlé d’arbre dégénéré ou arbre peigne.)

1. Tri par tas :

9.1 Notion de tas :

Pour qu’une structure de donnée soit considéré comme un tas, il faut deux choses :

1. **Arbre binaire parfait :**

Il s’agit d’un arbre dans lequel tous les niveaux sont entièrement remplis. (Seul le dernier niveau peut être incomplet)

Illustration :

Une image contenant capture d’écran, horloge, conception

Description générée automatiquement

1. **Propriété du tas :**

Cette propriété varie en fonction du tas, si on a un tas (max) alors chaque enfant d’un nœud aura une valeur inférieure à celle du parent.

Illustration :

Une image contenant cercle, diagramme, ligne

Description générée automatiquement

On remarque très vite que chaque clé enfant est plus petit que son parent. On dit alors que la clé du parent à une plus haute priorité.

On parle de « **max heap** »

Si on a un tas (min) alors chaque enfant d’un nœud aura une valeur supérieure à celle du parent.

Illustration :

Une image contenant cercle, croquis, blanc, diagramme

Description générée automatiquement

On remarque tout de suite la différence puisque c’est l’inverse. Dans ce cas-ci on dira que la clé du parent à une priorité plus basse.

On parle de  « **min heap** »

9.2 Fonctionnement du tri par tas :

Le tri par tas se base sur l’association d’une liste et d’un arbre binaire (tas).

L’arbre binaire doit être sous forme de tas (max heap).

1. Former un tas :

Revoir définition

1. Inverser la racine avec le plus haut index (SWAP) :

Étant donné que l’on utilise également une liste (tableau), chaque donnée à un indice.

On va donc inverser la racine, avec l’indice le plus élevé.

Une image contenant capture d’écran, horloge, cercle, nombre

Description générée automatiquement

Comme on peut le voir, un tas se lit en largeur.

Et donc l’élément avec le plus haut indice sera 9, qui a comme indice 5.

Inversion des deux.

12 se retrouve à l’indice 5 et 9 à l’indice 0 (soit la racine).

Une image contenant capture d’écran, horloge, cercle, nombre

Description générée automatiquement

1. Suppression du dernier élément (REMOVE) :

On va supprimer 12, de notre arbre et de notre tas.

(Il est considéré comme trier, sinon il finirai par remonter et donc boucle infini)

1. Reformer un tas (HEAPIFY) :

Étant donné qu’on a inverser deux données, le tas n’en est plus un. Il faut donc en reformer un.

* Refaire ces 4 étapes en boucles.

INFOS SUPPLÉMENTAIRES :

* Pour trouver le fils de droite d’un parent : 2\*père + 1
* Pour trouver le fils de droite d’un parent : 2\*père + 2

9.3 Complexité :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

9.4 Utilités :

1- OS

2- GUI

3- Moteur de recherche

1. Table de hachage :

10.1 Introduction :

Structure de données reposant sur des tableaux.

* Donnée statique
* Algorithme de recherche performant

Structure de données reposant sur des tableaux.

10.2 Notion de clé :

Une clé c’est comme un indice que l’on utilise dans les tableaux.

Il s’agit d’un moyen d’identifier de manière certaine un élément.

10.3 Fonction de hachage :

Il s’agit d’une fonction qui prend une entrée et en sort une empreinte « unique ».

Propriété :

* **Longueur des sorties identiques :**

Le nombre de caractères introduit en entrée de la fonction n’influent pas sur le nombre de caractères à la sortie.

* **Unidirectionnel :**

Lorsque l’on hash une donnée, il est impossible de revenir en arrière. Grosse différence avec le cryptage.

* **Sortie impossible à prédire :**

Il est impossible de prédire la sortie de ces fonctions.

* **Sortie unique :**

La fonction permet que chaque sortie soit unique.

ATTENTION :

Il est possible que pour une donnée quasiment similaire. La sortie soit identique une solution existe, voir plus loin.

On parle alors de **collisions.**

10.4 Notion de salage :

Consiste à ajouter un élément à la donnée, permettant ainsi de modifier sa signature.

(Signature = sortie lors de son passage dans une fonction de hachage)

10.4.1 Utilité :

Le salage pemet d’augmenter la sécurité face aux différentes attaques, mais également d’éviter les doublons.

10.4 Exemple de fonction de hachage :

* MD5 :

Bas niveau car collisions possibles et trouvables.

Signature = 128 bits

* SHA-1 :

Meilleur taux de collisions mais reste trouvables.

Signature = 160 bits

* SHA-2 (SHA-256 / SHA-512) :

Standards actuels

Signature = 256 et 512 bits

10.5 Illustration d’une table de hachage :

Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

* Bleu = clé
* Rose = index (hash de clé)
* Vert = clé-valeur (contenu de l’élément)

Dans ce cas-ci tout va bien, mais on a vu qu’il était possible d’avoir des collisions. Si deux éléments on le même index, comment savoir lequel est le bon ?

2 solutions :

1. **Par chaînage :**

Dans ce cas-ci, chaque index aura « derrière » lui une liste chaînées avec les différentes clé-valeur qui auront eu le même hash de sorti.

Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

Dans cette exemple, « Sandra Dee » et « John Smith » ont le même hash de sorti. Ils sont donc contenu dans une liste chaînées derrière le hash « 873 ».

1. **Par adressage ouvert :**

Dans ce cas-ci, lorsqu’une collision aura lieu, on va chercher un emplacement libre dans notre table de hachage.

(On fait ça grâce à un sondage, à approfondir si on veut)

Une image contenant texte, capture d’écran, ligne, diagramme

Description générée automatiquement

Dans cette exemple, « Sandra Dee » et « John Smith » ont le même hash de sorti. « John Smith » ayant comme index 873, on cherche un autre emplacement disponible pour « Sandra Dee » et on trouve 874.

10.6 Utilité :

La complexité temporelle est très intéressante pour chercher / ajouter / supprimer un élément.

On l’utilise pour les bases de données, caches, etc.

Une image contenant texte, Police, capture d’écran, reçu

Description générée automatiquement

1. Fonctionnement du SHA-256 :
2. **Prétraitement des données :**

**Objectif :** avoir des données compatibles avec l’algorithme.

* + **Padding :** rembourrage en français

Si la donnée n’est pas assez grande pour rentrer dans la fonction de hachage, on rajoute des bits afin qu’elle le devienne.

* + **Data Length :** longueur des données en français

1. **Initialisations des constantes :**

**Objectif :** constantes nécessaires à l’algorithme.

* + **Constante (h0-h7)**: elles sont calculées à partir des 32 premiers bits et des 8 premiers nombres premiers.

(2, 3, 5, 7, 11, 13, 17, 19)

1. **Division en blocs :**

**Objectif :** avoir des blocs de taille fixe pour le traitement.

* + **Blocs :** 512 bits

1. **Traitement des blocs :**

**Objectif :** produire le hachage

* + **Traitement :** chaque bloc est traité en plusieurs étapes (appelés « rounds » ou « itérations »). Lors de ces rounds, les données subissent des opérations logiques et arithmétiques.

**(Rotation, permutation, modulo, etc.**)

1. Pathfinding :

12.1 Définition :

Il s’agit de trouver le chemin le plus court entre un point A et un point B.

12.2 Différent critères :

On peut choisir d’établir le chemin le plus court en fonction de diffèrent critères :

* Distance
* Cout
* Vitesse

12.3 Approche intuitive :

1. Prendre l’élément de fin, définir ses coordonnées et initialiser un compteur.
2. Ajouter l’élément à une liste FIFO.
3. Parcourir la liste, inclure les futurs éléments rajouter et réaliser ces opérations :
   1. Créer une liste des 4 cases adjacentes en augmentant le compteur.
   2. Si la case est un mur / existe déjà dans la liste principale on la retire.
   3. Ajouter toutes les cases restantes à la fin de la liste principale.
4. Réaliser a, b et c jusqu’à tomber sur l’élément de début.
5. Démarrer de cet élément et prendre l’élément adjacent avec le compteur le plus bas.
6. On trouve le chemin le plus court.
7. Théorie des graphes :

13.1 Définition :

Un graphe est un couple on le note G = (V, E)

* G => Graphe
* V => ensemble de sommets
* E => ensemble d’arêtes (qui sont des paires de sommets, un sommet à chaque extrémités)

13.2 Exemple de graphe :

Il existe énormément de type de graphes, 2 exemples :

Une image contenant trépied

Description générée automatiquement

Une image contenant ligne, blanc, diagramme, Police

Description générée automatiquement

13.3 Informations supplémentaires sur les graphes :

1. Les graphes sont une structure de données abstraites.
2. Opérations de bases :
   1. Adjacents(G, x, y)
   2. Voisins(G, X)
   3. Ajouter\_Sommet(G, x)
   4. Supprimer\_Sommet(G, x)
   5. Ajouter\_Arete(G, x, y)
   6. Supprimer\_Arete(G, x, y)
   7. Retourner\_Valeur(G,x)
   8. Fixer\_Valeur(G, x, v)
3. **Listes d’adjacence :**

Chaque sommet est un objet qui a une liste des sommets adjacents.

1. **Matrice adjacence :**

Illustration :

Matrice adjacence

Graphique

1. 2. 3. 4

Une image contenant ligne

Description générée automatiquement Une image contenant conception, typographie

Description générée automatiquement avec une confiance faible

1. 2. 3. 4

Les lignes représentent les sommets de départ, et les colonnes représente les sommets d’arrivés.

1. **Matrice d’incidence :**

Illustration :

Matrice d’incidence

Graphique

Une image contenant croquis, dessin, diagramme, Dessin au trait

Description générée automatiquement Une image contenant texte, Police, nombre, typographie

Description générée automatiquement

Petite particularité pour cet exemple, on peut voir des « -1 » et des « 1 ».

Les « 1 » apparaissent quand là l’arrête part de cet élément.

Les « -1 » apparaissent quand l’arête pointe sur cet élément.

* Cela se passe lorsque nous avons un graphe orienté uniquement. Dans le cas d’un non-orienté, les « -1 » seront remplacés par des « 1 ».

1. **Tableau récapitulatif :**

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

1. **Méthode de parcours :** 
   1. En largeur :

On utilise une file pour les nœuds voisins.

* 1. En profondeur :

Similaire aux labyrinthes

Marquage des sommets visités

1. Types d’algorithmes :
2. Shortest path :

Algorithme qui permet de trouver le chemin le plus court.

1. Spanning tree :

Algorithme qui vise à trouver l’arbre couvrant de poids minimal.

Un « spanning tree » peut-être illustré comme un arbre, c’est un graphe qui n’a pas de cycle.

1. Min-Cost flow :

Algorithme qui vise à trouver de manière la plus économe d’utiliser un réseau de transport.

1. Algorithme de Dijkstra :

15.1 Informations relatives à cet algorithme :

1. **Graphe pondéré et (non-)orienté**
2. **Objectif : trouver le chemin entre 2 sommets avec le poids minimum**

15.2 Fonctionnement :

Graphique :

Une image contenant texte, diagramme, dessin, cercle

Description générée automatiquement

On va utiliser un tableau, c’est la méthode vue en classe.

Tableau :

Une image contenant nombre, Tracé

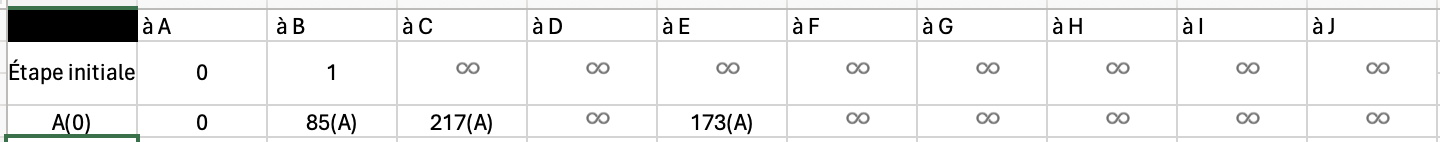
Description générée automatiquement

Voilà à quoi doit ressembler le tableau de départ.

Nous avons qu’un seul nœud pour l’instant, il s’agit de A qui a le poids 0 vu qu’il s’agit du point de départ.

1. On va regarder tous les liens qu’entretient A avec les autres nœuds. Dans ce cas-ci, B / C / E.

On met à jour notre tableau.



Ici on peut voir que de A à B, on a une relation qui a un poids de 85. Dans la case « à B », j’ai indiqué le poids de la relation + du nœud d’où je viens.

J’ai également apporté une modification à ma colonne « à A », car je considère que je ne reviendrai plus en arrière.

Une image contenant texte, nombre, capture d’écran, Police

Description générée automatiquement

1. On regarde maintenant qu’elle est le poids le plus faible, dans ce cas-ci B avec un poids de 85.

B sera notre nouveau point de départ. On va donc recopier exactement la même chose juste en dessous et apporté la même modification que pour la colonne « à A » :

Une image contenant texte, nombre, capture d’écran, Police

Description générée automatiquement

On regarde tous les liens qu’entretient B, dans ce cas-ci F. Avec un poids de 80.

On met à jour le tableau.

Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement

J’ai reproduit exactement la même étape qu’au point 1. Il faut cependant ajouter les poids de B et F. Vu qu’on part de A, qu’on passe par B pour aller à F.

1. On regarde cette fois-ci qu’elle poids est le plus faible. On peut voir que F à le poids le plus faible 165, F sera notre nouveau point de départ.

On regarde donc les relations. Il y a I uniquement, avec un poids de 250.

On met à jour le tableau.

Une image contenant texte, nombre, capture d’écran, Police

Description générée automatiquement

Dans ce cas-ci, je ne rentre pas la valeur I dans la colonne tout à gauche, car ce n’est pas le poids le plus faible. On peut voir que C et E ont des poids plus faibles.

I reste donc en « suspend ».

1. Cas particulier, comme expliqué précédemment, I n’est pas le poids le plus faible. On ne va donc plus regarder les relations que possèdent I mais le poids le plus faible dans le tableau.

Dans ce cas-ci, il s’agit de E avec un poids de 173. E sera notre nouveau point de départ. On regarde donc ses relations, et on voit J.

On met à jour le tableau.

Une image contenant texte, nombre, capture d’écran, Police

Description générée automatiquement

Comme on peut le voir sur le tableau, j’ai ajouté E et d’où vient (le plus court) dans ce cas-ci A avec un poids de 173.

Comme il s’agit du chemin le plus court. On sait qu’on ne reviendra pas dessus donc on peut modifier sa colonne.

Photo à la verticale normalement

Une image contenant texte, reçu, Police, nombre

Description générée automatiquement

1. Tous les cas de figure ont été vu, il suffit de les appliquer au bon moment.

Le tableau final ci-dessous.

Une image contenant texte, nombre, capture d’écran, mots croisés

Description générée automatiquement

1. Algorithme de Floyd-Warshall :

16.1 Informations relatives à cet algorithme :

1. **Distance la plus courte entre toutes les paires de sommets**
2. **Besoin d’une représentation en matrice adjacente**

16.2 Fonctionnement :

Graphique :

Une image contenant dessin, croquis, cercle, ligne

Description générée automatiquement

1. Tout d’abord, on va créer la matrice adjacente.

Une image contenant capture d’écran, nombre, ligne

Description générée automatiquement

En jaune ce sont les sommets de départ et en bleu les sommets d’arrivés.

1. Le principe de fonctionnement est simple, le but étant d’avoir les chemins les plus court entre chaque sommet du graphique.

Une image contenant capture d’écran, nombre, ligne

Description générée automatiquement

On va prendre la ligne 1 et la colonne 1.

1. Ensuite lorsque nous avons une intersection entre deux chiffres sur le reste du tableau, on additionne ces deux nombres et on complète la case.

Une image contenant capture d’écran, nombre, ligne

Description générée automatiquement

Ici par exemple, on a pour la première case verte, un « 2 » sur la ligne du 1. Et un « 4 » sur la colonne du 1. On les additionne, cela nous donne 12.

Pour la deuxième case verte on a un « 1 » sur la ligne du 1. Et un « 4 » sur la colonne du 1. On les additionne, cela nous donne 5.

On met à jour la matrice.

Une image contenant capture d’écran, nombre, ligne

Description générée automatiquement

1. Cas particulier :

Une image contenant capture d’écran, nombre, ligne

Description générée automatiquement

Lorsque l’on va prendre la ligne et la colonne 2.

Si on applique le principe donc qu’au croissement on additionne les deux, on obtiendrai « 3 ». Grâce aux deux case orange. Vu que 3 est inférieur à 9, on remplacera « 9 » par « 3 ».

1. On répète l’opération en prenant cette fois-ci la colonne et la ligne de 2. Et ainsi de suite jusque 4.

REMARQUE :

Il faut bien prendre en compte qu’on souhaite avoir le chemin le plus faible entre chaque sommet.

Le but n’est pas de remplacer les infinis présents dans la matrice. Il faut bien faire attention à chaque étape.

1. Matrice finale :

Une image contenant capture d’écran, nombre, ligne

Description générée automatiquement

1. Algorithme de Bellman-Ford :

17.1 Informations relatives à cet algorithme :

1. **Autorise les poids négatifs**
2. **Permet de déterminer les circuits absorbants**
3. Heuristique :

18.1 Définition :

Méthode de calcul qui fournit rapidement une solution réalisable, mais pas forcément optimale (voire exacte).

Avec d’autres mots :

Méthode rapide permettant de trouver une solution qui marche, mais qui n’est pas forcément la meilleure et qui peut parfois être incorrecte.

18.2 Utilisation :

Une heuristique est utilisé lorsque la rapidité de la résolution est plus importante que la précision absolue.

Exemple :

1. **Théorie des graphes**
2. **IA**
3. **Programmation de jeux**

18.3 Exemples :

On va comparer la vitesse d’exécution de l’algorithme de Dijkstra et d’un algorithme utilisant une heuristique.

**Premier exemple :**

**Une image contenant capture d’écran, ligne, diagramme, Tracé

Description générée automatiquement**

Comme on peut le voir, Dijkstra a mis plus de temps, vu qu’il a analyser toute la grille.

L’autre algorithme, lui a fait une « prédiction » du chemin à prendre, cela a donc été plus rapide pour lui.

**Dans ce cas-ci, le deuxième algorithme a été plus rapide et à fournit le chemin le plus rapide d’un point A à un point B.**

**Deuxième exemple :**

**Une image contenant capture d’écran, Rectangle, diagramme, carré

Description générée automatiquement**

**Dans ce cas-ci, on peut voir que le deuxième algorithme est plus rapide en termes de temps d’accès au point B, mais qu’il ne fournit pas le chemin le plus rapide.**

18.4 Critères d’évaluation :

1. **Qualité du résultat :** comparaison avec résultat optimal connu.
2. **Coût de l’heuristique :** complexité de celle-ci.
3. **Étendue du domaine d’application.**
4. Algorithmes probabilistes :
5. **Algorithme de Monte-Carlo**
6. **Algorithme de Las Vegas**
7. **Algorithme d’Atlantic City**
8. Algorithme « Min-cost flow » :
9. **Algorithme de Busacker et Gowen**
10. **Algorithme de Ford-Fulkerson (variante du 1-)**
11. **Algorithme d’Edmonds-Karp**
12. **Algorithme de Goldberg-Tarjan**
13. Algorithme de Kruskal :

21.1 Informations relatives à cet algorithme :

1. **Prend en entrée un graphe pondéré et connexe.**
2. **La sortie sera un arbre (graphe sans cycle)**
3. **Le but étant d’avoir un arbre couvrant le poids minimum**

21.2 Fonctionnement :

Graphique :

Une image contenant ligne, diagramme, capture d’écran, cercle

Description générée automatiquement

1. Lister les arrêtes dans l’ordre croissant :

|  |  |
| --- | --- |
| F,C | 1 |
| A,B | 3 |
| A,C | 4 |
| B,C | 4 |
| E,D | 5 |
| A,F | 7 |
| C,D | 9 |
| E,D | 9 |

1. Dessiner l’arbre :

Pour dessiner l’arbre, on va utiliser notre liste préétablie.

Il faut prendre en compte, qu’on ne veut pas de cycle dans notre arbre.

* 1. F,C

**Une image contenant ligne, diagramme, capture d’écran, Tracé

Description générée automatiquement**

On prend cette arête, aucun cycle lié à cette arête.

* 1. Continuez jusqu’à B, C

**Une image contenant ligne, diagramme, capture d’écran, Tracé

Description générée automatiquement**

On ne prendra pas cette arête car sinon on a un cycle entre A, B et C.

* 1. Terminez l’arbre

1. Autres algorithmes :
2. **Algorithme de Prim (similaire à Dijkstra)**
3. **Algorithme de Boruvka (utilisation des arêtes comme base)**
4. Les algorithmes dans les jeux :

23.1 Jeu à somme nulle :

Un jeu à somme nulle, est un jeu dans lequel le gain d’un joueur entraine la perte de l’autre joueur.

Les gains totaux sont égaux à zéro, exemple : (parti d’échec)

Victoire = +1 Défaite = -1 Nul = +0

Joueur A gagne, B perds => 1 +(-1) = 0

Joueur B gagne, A perds => -1 + 1 = 0

Match nul => 0 + 0 = 0

23.2 Minimax :

Le minimax est un algorithme, qui vise à maximiser les gains pour soi-même et à minimiser les gains pour son adversaire.

Le minimax est limité est limité par la profondeur de recherche maximale. C’est-à-dire que plus son arbre de possibilité sera grand et plus l’algorithme demandera des ressources de calcul. Cela peut vite devenir coûteux.

Comment ça fonctionne :

On peut voir sa sous une forme d’arbre. Chaque niveau de l’arbre correspond à un Min ou Max.

Lorsque le niveau est Max il concerne le joueur, lorsque le niveau est Min il concerne l’opposant.

Une image contenant ligne, diagramme, Rectangle, capture d’écran

Description générée automatiquement

Niveau MAX

Joueur

Niveau MIN

Opposant

Niveau MAX

Joueur

Une image contenant texte, capture d’écran, diagramme, ligne

Description générée automatiquement

Niveau MIN

Opposant prend le coup ou il gagne le -

Niveau MAX

Joueur prend le coup dans lequel il gagne le +

23.3 L’élagage :

L’élagage est lié au Minimax, il permet de supprimer des branches « inutiles » dans l’arbre des possibilités permettant ainsi à l’algorithme d’être plus performant et de ne pas surconsommer en termes de ressources.

23.4 Backtracking :

Le backtracking (retour sur trace) est une technique algorithmique qui va emprunter toutes les solutions possibles, et éliminés celle qui ne sont pas viables.

Il reviendra sur ses traces afin d’explorer une autre solution.

Exemple :

Une image contenant ligne, diagramme, conception

Description générée automatiquement

Parcour 1 : 0 – 1 –3 –7

* Solution pas correcte il revient sur ses traces ici 3 car une autre solution est possible

Parcour 2 : 0 – 1 – 3 – 8

Parcour 3 : 0 – 1 – 4 – 9

(Ainsi de suite)

On parcours en profondeur l’arbre de décision d’un problème.

Utilisation du backtracking pour des problèmes de satisfaction de contraintes, (problème dans lequel un ensemble de contrainte doivent être prise en même temps).

1. Chiffrement :

24.1 Définition :

Le chiffrement est un procédé de cryptographie grâce auquel on souhaite rendre la compréhension d'un document impossible à toute personne qui n'a pas la clé de chiffrement.

Autrement dit :

Moyen qu’on utilise pour rendre la lecture d’un message impossible sans une clé de chiffrement.

24.2 Objectif principaux :

1. **Confidentialité :**

Rendre les données impossibles à lire sauf aux personnes autorisées.

1. **Intégrité :**

Les données envoyées doivent être les mêmes que les données reçus. S’assurer qu’aucune modification ait eu lieu pendant le transit.

1. **Authenticité :**

S’assurer que les données proviennent bien de la source désirée.

24.3 Techniques de base :

1. **Substitution : (Cesar)**

Décalage similaire entre toute les lettres. Lorsque l’on arrive à la fin, on recommence l’alphabet.

Très simple.

Une image contenant Police, ligne, diagramme, texte

Description générée automatiquement

1. **Transposition**
2. **Par bloc : ( DES, AES)**
3. **Par flux : (RC4)**
4. Chiffrements asymétriques :

25.1 Algorithmes utilisés :

1. **DES (Data Encryption Standard)**

Cette technique de chiffrement utilise des clés de 56 bits.

Elle n’est plus recommandé à l’heure actuelle. (Trop lent + attaque possible)

1. **3DES (Triple DES)**
2. **AES (Advanced Encryption Standard)**
3. **Blowfish**
4. **Twofish**

25.2 Principe de base :

On va utiliser une clé de chiffrement et cette même clé sera utilisé pour déchiffrer le message.

25.3 Vulnérabilités :

1. **Brute force**
2. **Attaque par dictionnaire :**

Cette attaque fonctionne avec des mots couramment utilisés. Elle est particulièrement efficace contre clés courtes et faibles.

1. **Attaque par analyse différentielle :**

Cette attaque fonctionne avec des mots clairs et leur équivalent chiffrés. Pour que cette technique fonctionne correctement, il faut un grand nombre de ces informations.

Le but étant de trouvé des patterns entre ces différentes paires d’informations et ainsi trouvé une vulnérabilité du chiffrement.

1. **Attaque par force brute en temps réel :**

Attaque interceptant le message chiffré et essayant de déchiffrer le message en temps réel en utilisant différentes clés jusqu'à ce que le message chiffré soit déchiffré avec succès.

25.4 Améliorer la sécurité :

1. **Augmenter la longueur de la clé :**

L’attaque par brute force sera pratiquement infaisable si la longueur de la clé est assez longue.

1. **Utilisation de clé aléatoires :**

L’attaque par dictionnaire sera bien plus compliqué voir impossible.

1. **Mode de chiffrement plus sûr / Clé unique**

25.5 DES :

Cet algorithme de chiffrement, est un algorithme par division en bloc.

Plus précisément des blocs de 64 bits. Les clés générées font 56 bits.

**Première illustration :**

**Une image contenant texte, capture d’écran, nombre, Police

Description générée automatiquement**

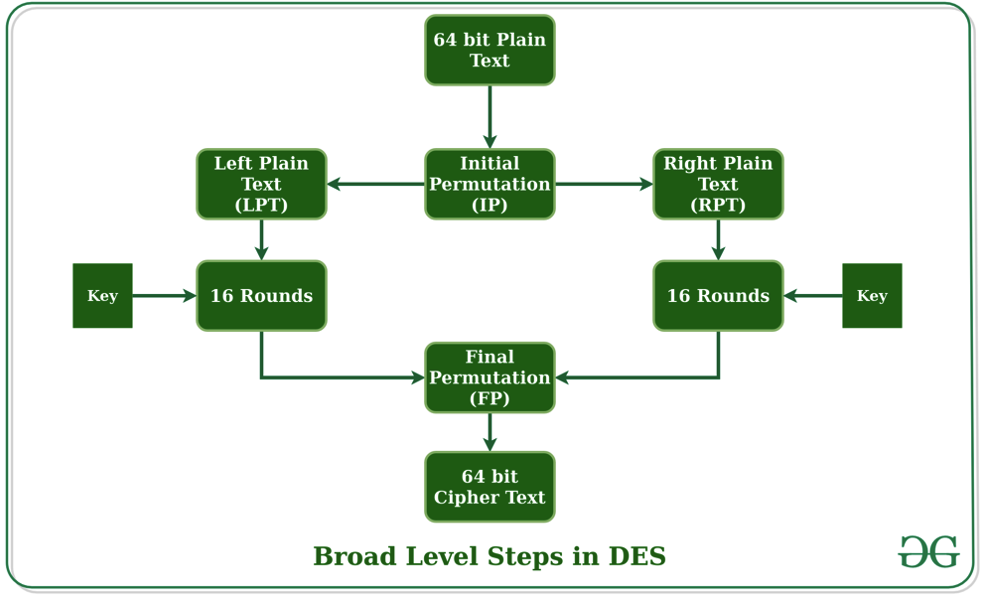
Comme on peut le voir, de blocs se dessinent, puis passent dans l’algorithme et des blocs de 64 (crypté) sortent de l’algorithme.

**Deuxième illustration :**

Ici c’est plutôt à l’intérieur de l’algorithme. Le bloc est redivisé en deux.

Chaque partie se verra être manipulé 16 fois en lien avec la clé de chiffrement.

Et pour finir les deux parties se rassemblent pour sortir.

****

25.6 3DES :

Le 3DES, offre une plus grande résistance aux attaques. Il va appliquer 3 fois l’algorithme DES consécutivement.

On utilise donc 3 clés, et voici comment ça marche :

1. Premier DES : on chiffre avec la première clé
2. Deuxième DES : on déchiffre avec la deuxième clé
3. Troisième DES : on chiffre avec la troisième clé

**Illustration :**

A droite : Chiffrement

A gauche : déchiffrement

Cipher =

Chiffrement

Reverse Cipher =

Déchiffrement

**Une image contenant texte, diagramme, Plan, ligne

Description générée automatiquement**

25.7 AES :

Cet algorithme prend en charge 3 taille de clés différentes : 128, 192, 256 bits.

Il est itératif, c’est-à-dire qu’il y a un certain nombre de passage dans l’algorithme, le nombre d’itération et donner par la taille de la clé :

* + 128 = 10 itérations (rondes)
  + 196 = 12 itérations (rondes)
  + 256 = 14 itérations (rondes)

Il fonctionne sur des blocs également de 128 bits. Sur lesquels ils appliquent des actions.

**Illustration :**

**Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement**

25.8 Mode de chiffrement :

1. **ECB (Electronic Codebook) :**

Ce mode divise les données en taille fixe.

Chaque bloc sera chiffré avec la même clé (même bloc clair = même bloc chiffré = problème de sécurité)

Vulnérable aux attaques par dictionnaire.

1. **CBC (Cipher Block Chaining) :**

Ce mode divise également les données en bloc.

Lorsqu’il va chiffré un bloc, il va d’abord combiné le bloc précédent chiffré avec le bloc de donnée clair et ensuite chiffré le tout.

Attaque par dictionnaire plus difficile.

1. **CTR (Counter) :**

Ce mode utilise un compteur (nombre binaire).

Le compteur, ainsi que la clé sont envoyés dans la fonction de chiffrement. La sortie est un flux (série de bit).

Ensuite la donnée en clair, subira l’opération XOR avec ce flux.

**Une image contenant capture d’écran, diagramme, texte, ligne

Description générée automatiquement**

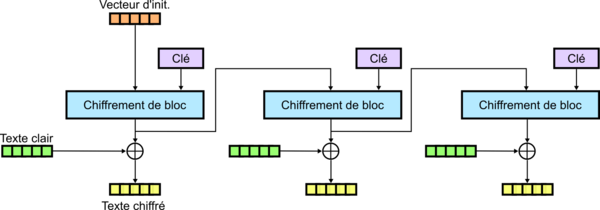
1. **OFB (Output Feedback) :**

Ce mode utilise un vecteur d’initialisation (série de bit).

Ensuite on combine vecteur + clé que l’on envoie à la fonction de chiffrement, celle-ci renvoie un flux. (pour la première étape)

La donnée subira l’opération XOR avec ce flux.

Pour les autres étapes, on utilisera le flux précédent + la clé pour produire un nouveau flux. Etc.



25.9 Problème liés au chiffrement symétriques :

1. Échange de clés :

Étant donné que la même clé est utilisé pour le chiffrement et le déchiffrement, il faut pouvoir la transmettre de manière sécurisée.

1. Distribution des clés
2. Rotation des clés :

Il est fréquemment utilisés de changer la clé de chiffrement, pour garantir une sécurité accrue.

25.10 Solutions :

1. On chiffre la clé
2. PKI (Public Key Infrastructure) : la PKI est un système de chiffrement asymétrique qui permet d'authentifier et de sécuriser les communications électroniques. La PKI utilise des certificats numériques pour vérifier l'identité des parties et assurer l'intégrité des données.
3. Il est possible d’utiliser des systèmes de gestion de clés centralisés, facilitant aux organisation les problèmes vus au point 25.9.
4. Chiffrements asymétriques :

26.1 Principe de base :

1. Utilisation d’un couple de clés :
   1. Publique => permet de chiffre le message.
   2. Privée => permet de déchiffrer le message.
2. Présence de deux fonctions chiffrement et déchiffrement.

26.2 Algorithmes utilisés :

1. **RSA**
2. **Chiffrement El Gamal**
3. **Chiffrement de Merkle-Hellman**

26.3 RSA :

Cet algorithme est basé sur la difficulté de factorise de grands nombres entiers en facteur premier.

**Comment générer une clé RSA :**

1. Prendre 2 nombres premiers différents ‘p’ et ‘q’
2. Calculer le module de chiffrement ‘n’ = ‘p’ \* ‘q’
3. Calculer l’indicatrice d’Euler
4. Choisir un nombre premier tel que ‘e’ < et premier avec
5. Calculer le nombre entier ‘d’ = ‘e’-1

**Pourquoi le chiffrement RSA est-il considéré comme sur ?**

Il est pratiquement impossible de déchiffrer la clé privée sur base de (n, e) dans un temps raisonnable.

**Attaques possibles :**

* Brute force :

Sur ‘p’ et ‘q’

* Attaque de Wiener :

Retrouver ‘d’ en fonction de ‘n’ et ‘e’.

* Hastad
* Time attack
* Adaptive chose cyphertext attack

1. RegEx :

27.1 C’est quoi ?

RegEx signifie « Regular Expression » ou « expression régulière ».

Une expression régulière est une chaîne de caractère qui en décrit un ensemble.

27.2 Comment ça marche ?

On les représente avec des [ ].

* [a-z] => lettres minuscules
* [0-9] => chiffres
* [&é"'(è\_çà)=] => caractères spéciaux (à compléter si nécessaire)
* [^a] => pas dans l’ensemble
* [aei-ou] => a, e, tout ce qui se trouve entre i et o, u
* \w => lettres + chiffres + \_
* \d => chiffres
* \s => caractères blancs
* . => n’importe quel caractère (\. pour détecter le point)
* Chaque raccourci en majuscule donne la négation
* a|b => les a OU les b
* ? => 0 ou 1 élément
* \* => 0 ou plusieurs éléments
* + => au moins 1 élément
* {n,m} => un nombre défini d’éléments

27.3 Exemple :

Une image contenant texte, Police, capture d’écran, algèbre

Description générée automatiquement