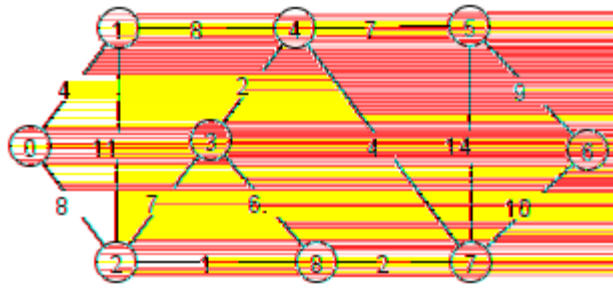


On applique les algorithmes de cours

Exercice 1 – Arbre couvrant minimum



Pour le graphe pondéré ci-dessus on cherche à trouver l'arbre couvrant minimum en appliquant un algorithme de cours.

1. Choisissez un algorithme (écrivez juste son nom s'il s'agit d'un algorithme connu).

Correction. On choisit l'algorithme de Prim.

2. Appliquez l'algorithme (dessinez toutes ses itérations).

Correction. On construit un arbre en partant d'un sommet initial et en ajoutant chaque fois une arête du poids min qui le touche par une extrémité.

–On part du sommet 0.

–On ajoute l'arête 0-1. Les sommets de l'arbre sont 0,1.

–On ajoute l'arête 1-4. Les sommets de l'arbre sont 0,1,4.

–On ajoute l'arête 4-3. Les sommets de l'arbre sont 0,1,3,4.

–On ajoute l'arête 4-7. Les sommets de l'arbre sont 0,1,3,4,7.

–On ajoute l'arête 7-8. Les sommets de l'arbre sont 0,1,3,4,7,8.

–On ajoute l'arête 8-2. Les sommets de l'arbre sont 0,1,2,3,4,7,8.

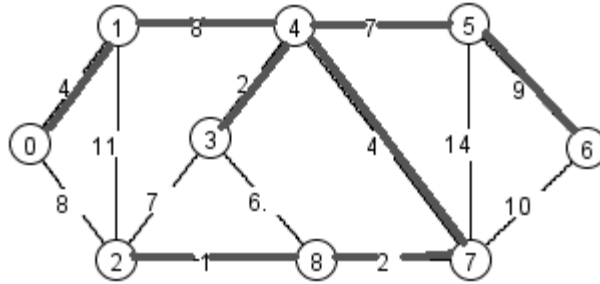
–On ajoute l'arête 4-5. Les sommets de l'arbre sont 0,1,2,3,4,5,7,8.

–On ajoute l'arête 5-6. L'arbre couvre tout le graphe.

On choisit l'algorithme de Prim.

3. Donnez le résultat final : arbre couvrant minimum.

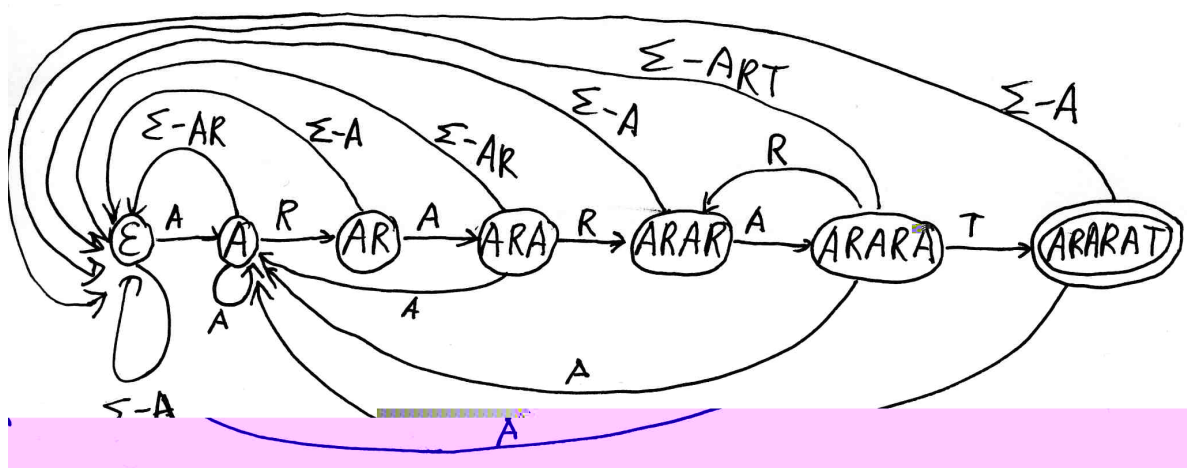
Correction.



Exercice 2 – Automate d'occurrences

1. Construire l'automate déterministe minimal qui reconnaît toutes les occurrences du mot "ARARAT" dans un texte quelconque. Pour cet exercice seul le résultat final sera évalué. Inutile donc d'expliquer votre méthode.

Correction.



On adapte les algorithmes de cours

Exercice 3 – Poids max de camion

Un réseau routier connecte les villages d'une île. Pour chaque route $e = (u, v)$ on connaît le poids maximum (en tonnes) $p(u, v)$ du camion qui peut emprunter cette route. On cherche à calculer pour toutes les paires de villages u, v le poids maximum de camion qui peut aller de u à v via le réseau routier tout en respectant la contrainte de poids pour chaque route empruntée.

1. Formalisez le problème en complétant le texte ci-dessus.

On a un graphe (V, E) et une fonction de poids $p : E \rightarrow \mathbb{R}^+$. On définit la capacité d'un chemin $\sigma = v_1, v_2, \dots, v_k$ comme $C(\sigma) = \dots$. Pour deux sommets u et v on définit la capacité $C(u, v) = \dots$. Le problème algorithmique CamionMax consiste à trouver $C(u, v)$ pour toutes les paires de sommets u, v .

Correction.

$$-C(\sigma) = \min_i p(v_i, v_{i+1})$$

$$-C(u, v) = \max C(\sigma) \text{ où le maximum est sur tous les chemins } \sigma \text{ de } u \text{ à } v.$$

2. Quel algorithme de cours allez-vous modifier et quelles sont les modifications nécessaires ?

Correction. On modifiera l'algorithme de Floyd-Warshall. Les modifications sont les suivantes :

-Pour initialiser la matrice de poids $W[i, j]$ on posera $W[i, i] = \infty$ (pas de limites sur le poids si on reste dans le même village) ; $W[i, j] = p(v_i, v_j)$ si le graphe possède une arête (v_i, v_j) (on peut emprunter la route qui relie les deux villages, et sa capacité est $p(v_i, v_j)$) ; et, finalement, $W[i, j] = 0$ si le graphe n'a pas d'arête (v_i, v_j) (pas de route connue).

-A l'itération principale on prendra le max entre la capacité du chemin déjà connu et celle du meilleur chemin qui passe par le nouveau sommet v_k :

$$W^{(k)}[i, j] = \max \left(W^{(k-1)}[i, j], \min \left(W^{(k-1)}[i, k]; W^{(k-1)}[k, j] \right) \right)$$

3. Donnez l'algorithme CamionMax(...)

Correction.

algorithme CamionMax(V, E, P) $n = |V|$

pour i de 1 à n faire

 pour i de 1 à n faire

 si $i=j$ alors $W^0[i, j] = \infty$

 sinon si $(v_i, v_j) \in E$ alors $W^0[i, j] = p[i, j]$

 sinon $W^0[i, j] = 0$

pour k de 1 à n faire (*)

 pour i de 1 à n faire

 pour j de 1 à n faire

$$W^{(k)}[i, j] = \max \left(W^{(k-1)}[i, j], \min \left(W^{(k-1)}[i, k]; W^{(k-1)}[k, j] \right) \right)$$

retourner $W^{(k)}$

4. Expliquez pourquoi votre algorithme est correct.

Correction. L'invariant de l'algorithme est le suivant : après k itérations de la boucle (*) la matrice $W^{(k)}$ contient dans sa case $[i, j]$ le poids maximum du camion capable d'aller du village v_i au village v_j en passant seulement par les villages v_1, \dots, v_k . On le prouve facilement par récurrence. La correction de l'algorithme est une conséquence directe de cet invariant.

5. Analysez la complexité de votre algorithme.

Correction. $O(n^3)$ à cause de trois boucles imbriquées.

Exercice 4 – Le jeu de doublets

Ce jeu attribué à Lewis Carroll consiste à relier deux mots donnés en une chaîne de mots différant d'une seule lettre de leurs voisins. Ainsi on va de CINQ à SEPT par la chaîne : CINQ, CINE, CENE, CENT, SENT, SEPT.

1. Représentez ce jeu comme recherche d'un chemin dans un graphe. Décrivez précisément le graphe.

Correction. Il s'agit de graphe $G=(V,E)$ suivant. L'ensemble de sommets V est l'ensemble de tous les mots français. Deux mots u et v sont reliés par une arête s'ils diffèrent d'une seule lettre.

2. L'approche naïve consiste à construire la totalité du graphe et à appliquer un des algorithmes de parcours vu en cours. Quelles sont les difficultés de cette approche ? (*une ou deux phrases svp*)

Correction. Le graphe contient beaucoup de mots (à peu près un million), et énormément d'arêtes. Pour construire ces arêtes il faut tester $25n$ possibilités pour chaque mot de taille n . Si on construit ce graphe complètement, il prendra beaucoup de place dans la mémoire, beaucoup de temps pour le construire, et en fait une grande partie de ce graphe ne sera pas utilisé pour trouver un chemin spécifique.

3. L'approche plus raisonnable consiste à appliquer un algorithme de parcours à la volée, c-à-d sans charger dans la mémoire la totalité du graphe. Supposons qu'on est fourni d'une fonction ("oracle") booléenne $EstUnMot(w)$ qui renvoie vrai si et seulement si w est un mot de la langue française. Écrivez un algorithme $Doublets(u,v)$ qui cherche une chaîne de mots reliant u et v .

Correction. Voici un algorithme de type BFS. On génère les arêtes et les sommets (mots) à la volée. La file d'attente s'appelle *Waiting*. On mémorise tous les mots parcourus pour ne pas les re-visiter (et pour éviter que le parcours boucle) dans *Visited*. On stocke avec chaque mot l'information sur son prédécesseur, ainsi *Visited* doit être un ensemble de couples clé-valeur. Pour le reste, il s'agit de BFS standard.

```
algo doublets(u,v: mots)
{INITIALISATION}
m=longueur(u);
si longueur(v)!=m retourner faux;

Visited={(u, NIL)}: ensemble de couples clé-valeur de type mot-mot;
Waiting={u}: file de mots;
Trouvé=faux;

{BOUCLE PRINCIPALE - RECHERCHE DE CHEMIN}
tant que non Trouvé et Waiting.nonVide() faire
    motCourant=Waiting.défiler();
    si motCourant=v
    alors
        Trouvé= vrai
    sinon
        pour pos de 1 à m faire
            pour lettre de a à z faire
                motSuivant = remplacerLettre(motCourant, pos, lettre)
                si EstUnMot(MotSuivant) et non Visited.contient(motSuivant)
                alors
                    Visited.ajouter(motSuivant,motCourant);
                    Waiting.enfiler(motSuivant);

{IMPRESSION DU RESULTAT EN ORDRE INVERSE}
si Trouvé alors
    motCourant=v;
    tant que motCourant!= NIL faire
        imprimer(motCourant)
        motCourant=Visited.trouverValeur(motCourant)

retourner trouvé
```

On invente un algorithme

Exercice 5 – La plus longue sous-séquence croissante

On a une séquence de m nombres différents $\alpha = a_1, a_2, \dots, a_m$. Le problème SSC consiste à trouver sa plus longue sous-séquence croissante et la longueur de cette sous-séquence.

Par exemple, pour 7, 1, 9, 2, 8, 4, 5, 3 la réponse est 1, 2, 4, 5, longueur 4.

On utilisera la programmation dynamique pour concevoir un algorithme qui résolve ce problème.

1. Soit $L(i)$ la longueur de la plus longue sous-séquence croissante dans i premiers éléments a_1, a_2, \dots, a_i de la séquence α . Essayez de trouver des équations de récurrence pour $L(i)$. Expliquez pourquoi c'est difficile.

Correction. Pour décider si l'on ajoute a_i à la sous-séquence croissante déjà trouvée il ne suffit pas de connaître sa longueur $L(i-1)$, il faut aussi connaître son plus grand élément. Ceci rend impossible la récurrence sur la fonction $L(i)$.

2. Pour pallier à cette difficulté il faut prendre une fonction plus "fine" qui admette des équations de récurrence. On vous propose deux telles fonctions au choix :

$M(i)$ qui est la longueur de la plus longue sous-séquence croissante de α qui se termine par a_i (la sous-séquence doit donc inclure a_i).

$H(i, x)$ qui est la longueur de la plus longue sous-séquence croissante de a_1, a_2, \dots, a_i , telle que tous les éléments de cette sous-séquence sont inférieurs ou égaux à x .

Choisissez une des deux fonctions H ou M (*ne faites pas les deux!*) Écrivez les équations de récurrence pour cette fonction sans oublier les cas de base.

Correction. On choisit la fonction M . Chaque sous-séquence croissante qui se termine par a_i commence par une sous-séquence dont le dernier élément a_j est inférieur à a_i (et, bien sûr $j < i$). On doit trouver la plus longue parmi telles sous-séquences, d'où les équations ci-dessous.

$$M(i) = \begin{cases} 1 & \text{si } i = 1 \\ 1 + \max\{M(j) \mid j < i \wedge a_j < a_i\} & \text{si } i > 1, \end{cases}$$

ici le max vaut 0 si l'ensemble est vide.

3. Écrivez un algorithme efficace (récursif avec "marquage" ou itératif) pour calculer H ou M .

Correction. On remplira un tableau $M[i]$ avec toutes les valeurs de la fonction M . Puisque chaque valeur $M(i)$ ne dépend que des $M(j)$ pour $j < i$, le tableau peut être rempli d'une manière itérative dans l'ordre croissant d'indices.

1. $M[1]=1$
2. pour i de 2 à m faire
3. $lemax=0$
4. pour j de 1 à $i-1$
5. si $a[j] < a[i]$ et $lemax < M[j]$
6. alors $lemax=M[j]$
7. $M[i]=lemax+1$

4. En sachant calculer la fonction choisie H ou M , comment répondre à la question initiale : trouver la longueur de la plus longue sous-séquence croissante de α .

Correction. C'est juste $\max\{M[i] \mid 1 \leq i \leq m\}$. On notera l'indice du maximum (dernier élément de la plus longue sous-séquence croissante) par i_0

5. Comment trouver la plus longue sous-séquence croissante elle-même (toujours en sachant calculer la fonction choisie H ou M).

Correction. On peut mémoriser pour chaque a_i l'élément précédent de la sous-séquence. Pour ceci on crée un autre tableau $pre[i]$ qu'on pré-rempli par des 0. On ajoute à la ligne 6 de l'algorithme ci-dessus l'affectation

$pre[i] = j$

La plus longue sous-séquence croissante sera (dans l'ordre inverse) $i_0, pre[i_0], pre[pre[i_0]], \dots$ jusqu'à ce que le pre devienne 0.

6. Analysez la complexité de votre algorithme.

Correction. Les deux boucles pour, avec la taille limitée par m , et avec le corps de complexité $O(1)$. Donc, la complexité totale est $O(m^2)$.