Début

```
// initialisation
Graphe ← V, E, k[V], g[V][V]

//creatEdgesG(Graphe)
//créer un liste d'arête à partir la matrice d'adjacence du graphe G
//retourner le nombre totale d'arête
Pour (i allant de 0 à V-1) faire
        Pour (j allant de i+1 à V-1) faire
            Si (g[i] [j] ≠ NULL) alors
                p(src) ← i
                p(dest) ← j
                p(poids)  ← g[i][j]
                edges ← edges + p // liste d'arête du graphe
            Fin Si
    Fin Pour
Fin Pour

// trouver(Graphe)
// trouver l'arbre couvrant de version naïve en testant s'il est satisfait la limitation
Pour (iterE allant de 0 à strlen(edges)) faire
        // vérifier si la source et destinataire existe ou leur limite est la même que le tableau de limite utilisé
        Si ((src && dest) || (m[src] == k[src] || m[dest] == k[dest])) alors
            nonEdges ← nonEdges + iterE  // tableau des arêtes qui ne peuvent plus utilisé
    Fin Si
    PlusEdge(arbre, iterE) // ajouter l'arête iterE dans l'arbre
Fin Pour

// optimizer(Graphe)
//optimiser l'arbre couvrant pour trouver les autres arbres couvrants de poids minimum
Pour (iterE allant de 0 à strlen(nonEdges)) faire
    Pour (iterA allant de 0 à strlen(Tmini)) faire
        Si (m[src] < k[src] && m[dest] < k[dest]) alors
            max ← dfs1(iterA, iterE(src), iterE(dest), -1)
            Si (poids ≥ max) faire
                Pour (itE allant de 0 à strlen(dp1[iterE(dest)])) faire
                    minusEdge(arbre, itE)       //supprimer l'arête itE de l'arbre
                    plusEdge(arbre, iterE)
                    Tmini ← Tmini + arbre // tableau des arbres couvrants
                Fin Pour
            Fin Si
        Fin Si
        Si (m[src] == k[src] && m[dest] < k[dest]) alors
            max ← dfs2(iterA, iterE(src), iterE(src), -1, iterE(dest))
        Fin Si
        Si (m[src] < k[src] && m[dest] == k[dest]) alors
            max ← dfs2(iterA, iterE(dest), iterE(dest), -1, iterE(src))
        Fin Si
        Si (poids ≥ max) faire
            Pour (itE allant de 0 à strlen(dp1[iterE(dest)])) faire
                minusEdge(arbre, itE)       //supprimer l'arête itE de l'arbre
```

```
                                    plusEdge(arbre, iterE)
                                    Tmini ← Tmini + arbre // tableau des arbres couvrants
                            Fin Pour
                    Fin Si
            Fin Pour
Fin Pour

Fin


// dfs1(arbre, src, dest, precedent)
// Trouver des arêtes qui sont de poids maximum permis les arêtes pour allant de src à dest
Pour (i allant de 0 à N-1) faire
        Si (i ≠ precedent && g[src][i]) alors
                Si (dp[i] == -1) alors
                        Si (dp[i] > g[src][i]) alors
                                Pour (itE allant de 0 à strlen(dp1[src])) faire
                                        dp[i] ← dp[i] + itE
                                Fin Pour
                                dp[i] ← dp[src]
                        Sinon
                                p(src) ← src
                                p(dest) ← dest
                                p(poids)  ← g[src][i]
                                dp[i] ← dp[i] + p
                        Fin Si
                Fin Si
                dfs1(arbre, i, dest, src)
        Fin Si
Fin Pour


/ dfs2(arbre, src, src2,  precedent, dest)
// Trouver un arête qui est de poids maximum et relié avec point src parmis les arêtes pour allant de  // src au dest

Pour (i allant de 0 à N-1) faire
        Si (i ≠ precedent && g[src][i]) alors
                Si (i == dest) alors
                        flag ← 1
                Sinon
                        dfs2(arbre, i, src2,  src, dest)
                Fin Si
                Si (src == src2 && flag) alors
                        dp2(src) ← src
                        dp2(dest) ← i
                        dp2(poids) ← g[src][i]
                Fin Si
        Fin Si
Fin Pour
```