Quadrature nº 81 (2011) 8-15

Les graphes implicites

par Mathias HIRON*

Résumé.

Au travers de cinq sujets, nous montrons comment des algorithmes de graphes classiques peuvent se cacher au sein de problèmes de toutes sortes. Nous présentons une approche qui permet de trouver quel graphe caché dans un problème permet d'exprimer la question posée sous la forme d'un problème classique de graphe. Chacun des problèmes présentés peut se résoudre en appliquant l'un des trois algorithmes classiques que nous présentons brièvement.

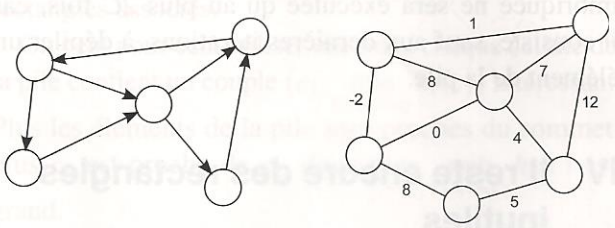
Les graphes sont parmi les objets les plus étudiés en algorithmique, et la théorie des graphes est également une branche importante des mathématiques discrètes. Connaître toutes les propriétés remarquables et algorithmes portant sur les graphes ne prend cependant tout son intérêt que si l'on est capable de reconnaître les graphes présents dans les problèmes auxquels on peut être confronté. Débusquer le ou les graphes cachés dans un problème et déterminer lequel d'entre eux va être la clé de sa résolution est souvent l'étape la plus difficile et la plus intéressante de la résolution d'un problème, tandis que l'application d'un algorithme classique à ce graphe n'est souvent qu'une formalité.

Dans cet article, nous allons vous donner un petit aperçu des différentes manières dont un graphe peut être caché au sein d'un sujet, et vous montrer comment un même algorithme peut être utilisé pour résoudre des problèmes très différents les uns des autres.

I Définition classique d'un graphe

Un graphe est généralement défini comme un ensemble de points reliés par des liens, où les points sont appelés sommets ou nœuds. On peut avoir des graphes orientés ou non orientés, selon si les liens sont unidirectionnels ou bidirectionnels, et ceux-ci sont alors appelés respectivement arcs ou arêtes. Un graphe peut aussi être pondéré si une valeur numérique (un poids) est associée à chaque arête ou arc. Être orienté ou pondéré ne sont que deux des nombreuses caractéristiques possibles d'un graphe, qui peut ainsi être dit simple, connexe, acyclique, planaire, biparti, dense, eulérien, etc. la liste est longue.

Cette définition classique s'accompagne d'une représentation visuelle toute aussi classique, présentée ci-dessous : à gauche un exemple de graphe orienté, à droite un exemple de graphe non orienté, pondéré.



Voici un exemple de définition plus formelle : un graphe orienté pondéré est un triplet $G = (V, E, \omega)$ où V est un ensemble fini de sommets, $E \subseteq V \times V$ est un ensemble d'arcs et $\omega : E \to \mathbb{R}$ est une fonction de pondération.

II Exemples d'algorithmes de graphes

II.1 Recherche du plus court chemin

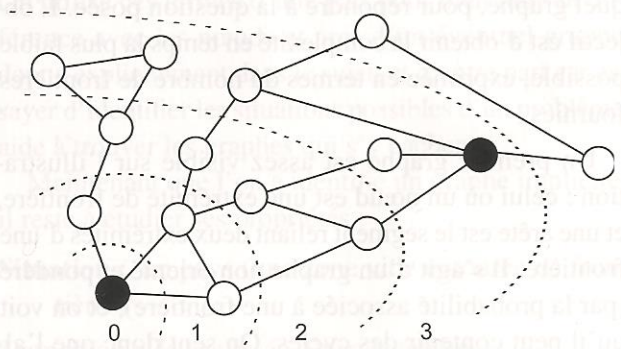
Le problème de graphe le plus classique est probablement celui de la recherche de la longueur du plus

* mathias.hiron@gmail.com

court chemin dans un graphe. Typiquement, on nous donne un graphe, un nœud de départ et un nœud d'arrivée, et le but est de trouver comment passer du sommet de départ au sommet d'arrivée par une séquence d'arcs la plus courte possible, la distance étant soit le nombre d'arcs de la séquence pour un graphe non pondéré, soit la somme des poids des arcs utilisés dans le cas d'un graphe pondéré.

Dans le cas du graphe non pondéré, ce problème peut se résoudre avec l'algorithme de parcours en largeur (ou BFS pour *Breadth First Search*). L'idée de cet algorithme est de lister tous les nœuds à une distance de 0 du nœud de départ (donc ce nœud lui-même), puis d'en déduire tous les nœuds à une distance de 1, puis tous les nœuds à une distance de 2, etc., jusqu'à obtenir une liste qui contient le nœud d'arrivée, dont la distance est alors la réponse à notre question (les étapes du chemin peuvent se retrouver facilement si besoin). Les nœuds à une distance k du point de départ sont les nœuds qui sont adjacents (reliés par une arête) aux nœuds à distance $k-1$ du départ et qui n'ont pas encore été listés comme étant à une distance inférieure du point de départ.

Cet algorithme a une complexité en temps et en mémoire de $O(nb_{\text{Nœuds}} + nb_{\text{Arcs}})$, ce qui est le mieux que l'on puisse faire.



Si le graphe est pondéré et que tous les poids sont positifs, un autre algorithme classique peut s'appliquer : l'algorithme de Dijkstra. Il se base sur la même idée, consistant à énumérer les nœuds dans l'ordre de leur distance au nœud de départ. Plusieurs variantes existent, mais la plus utilisée a une complexité en temps de $O((nb_{\text{Nœuds}} + nb_{\text{Arcs}}).\log(nb_{\text{Nœuds}}))$.

II.2 Programmation dynamique

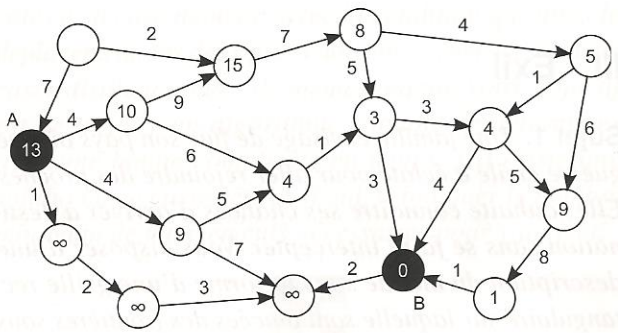
Si le graphe sur lequel on travaille a la propriété d'être un graphe orienté acyclique (DAG pour *Directed Acyclic Graph*), on peut très souvent résoudre le problème en appliquant le principe classique de la programmation dynamique. Il s'agit d'une technique qui n'est pas limitée aux graphes, mais que nous allons considérer comme étant l'un des algorithmes classiques de graphes à notre disposition.

Un graphe orienté est acyclique lorsqu'il est impossible de trouver un chemin qui part d'un nœud, suit une succession d'arcs (au moins un) dans le sens de leur orientation, puis retombe sur le nœud de départ.

L'idée de l'algorithme dynamique est très simple : on pose le problème comme une question sur l'un des nœuds du graphe, et on détermine la réponse à cette question en reposant une question similaire sur chacun des nœuds voisins et en combinant les réponses obtenues. Par exemple, si l'on recherche un plus court chemin du nœud A au nœud B, on va poser la question sur le nœud A : « Quelle est la distance minimale à parcourir du nœud A pour accéder au nœud B ? ». Si le nœud A a k nœuds voisins, V_1 à V_k, auxquels on accède par les arcs E_1 à E_k, on va poser pour chaque V_i la question « Quelle est la distance minimale à parcourir du nœud V_i pour accéder au nœud B ? ». Si on note R_i la réponse à chacune de ces questions, la réponse pour le nœud A est alors $\min_{1 \leqslant i \leqslant k} (poids(E_i) + R_i)$. Si le nœud sur lequel on pose la question est B lui-même, la réponse est 0, et si le nœud sur lequel on pose la question n'a pas de voisin et n'est pas B, la réponse est $+\infty$.

Toute la puissance de cette approche consiste à ne pas calculer deux fois la réponse à la même question : la première fois que la question est posée sur un nœud donné, on calcule sa réponse et on la stocke en mémoire. Toutes les fois suivantes où la même question est posée (à partir de chacun des arcs permettant d'arriver à ce nœud), on se contente de lire la réponse déjà calculée. Sans ce stockage, le nombre de questions posées peut être exponentiel par rapport à la taille du graphe. Grâce à ce stockage, on ne posera qu'une question par arc du graphe, ce qui donne une complexité en temps de $O(nb_{\text{Arcs}} + nb_{\text{Nœuds}})$.

Voici une illustration de l'application de cet algorithme sur un DAG. Les réponses à la question posée sur chaque nœud sont notées à l'intérieur du noeud. Le nœud est vide quand la question n'est jamais posée.



On remarque que l'algorithme ne peut pas du tout s'appliquer sur un graphe comportant des cycles, car pour calculer la réponse à la question sur un nœud, on

aurait alors besoin du résultat de cette même question au moment de retomber sur ce nœud via le cycle. Un graphe comportant des cycles peut cependant cacher un autre graphe acyclique, sur lequel on peut appliquer la programmation dynamique. De nombreux algorithmes de graphes peuvent ainsi être vus comme de la programmation dynamique appliquée sur un graphe acyclique bien choisi. C'est par exemple le cas de l'algorithme de Dijkstra évoqué plus haut, ou de Bellman-Ford, présenté dans un autre article.

II.3 Ce qu'il faut retenir pour la suite

Nous vous avons présenté l'idée générale de trois algorithmes, sans trop rentrer dans les détails. Le but de cet article n'est en effet pas de vous apprendre à maîtriser parfaitement ces algorithmes, qui sont présentés dans de nombreux ouvrages (tels que [1]), ou peuvent être découverts sur france-ioi.org.

Ce qu'il est important de retenir pour ce qui va suivre, c'est ce que permettent de faire ces algorithmes :

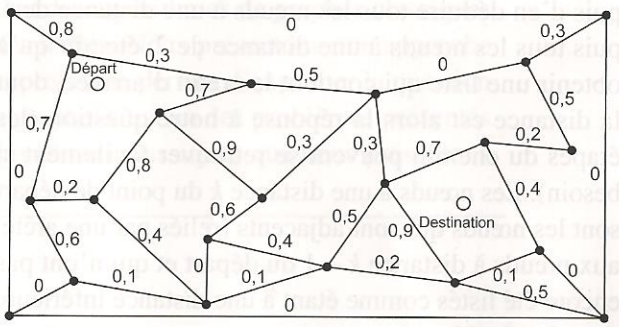
- Si l'on dispose d'un graphe non pondéré, déterminer le plus court chemin entre deux nœuds peut se faire avec l'algorithme BFS avec une complexité en temps de $O(nb_{\text{Nœuds}} + nb_{\text{Arcs}})$.
- Si l'on dispose d'un graphe pondéré dont les poids sont positifs, déterminer le plus court chemin entre deux nœuds peut se faire avec l'algorithme de Dijkstra, avec une complexité en temps de $O((nb_{\text{Nœuds}} + nb_{\text{Arcs}}).\log(nb_{\text{Nœuds}}))$
- Si l'on dispose d'un graphe orienté acyclique (DAG), répondre à de nombreuses questions telles que le plus court chemin, le plus long chemin, ou le nombre de chemins entre deux nœuds, peut se faire avec un algorithme dynamique avec une complexité en temps de $O(nb_{\text{Nœuds}} + nb_{\text{Arcs}})$.

Dans chacun des sujets suivants, votre objectif sera de débusquer le graphe caché dans le problème, sur lequel on peut appliquer l'un de ces trois algorithmes, et de décrire comment cela permet de répondre à la question posée avec une bonne complexité.

III Exil

Sujet 1. *Une famille envisage de fuir son pays où une guerre civile a éclaté pour aller rejoindre des proches. Elle souhaite connaître ses chances d'arriver à destination sans se faire intercepter. Vous disposez d'une description du monde sous la forme d'une grille rectangulaire sur laquelle sont placées des frontières sous la forme de segments de droite. Pour chaque frontière i, vous connaissez les coordonnées de ses deux extrémités A_i et B_i, mais aussi la probabilité P_i réussir à passer de l'autre côté de cette frontière sans être pris. Deux frontières ne s'intersectent jamais, mais peuvent partager la même extrémité. Le périmètre de la grille rectangulaire est formé de 4 frontières avec une probabilité traverser égale à 0. Étant donné les coordonnées du point de départ L et du point de destination F de la famille, déterminer la probabilité maximale de traverser sans se faire prendre, si l'on choisit le meilleur itinéraire entre les deux.*

Voici un exemple d'une telle carte du monde :

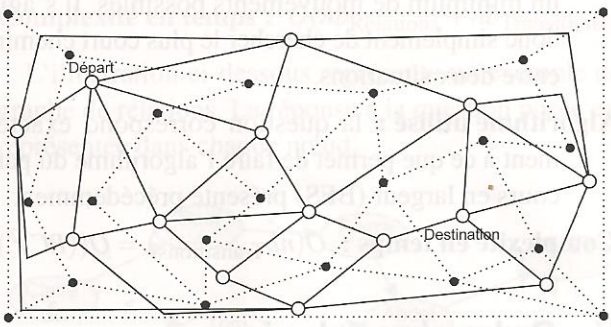


Avant de lire la suite, essayez de déterminer quel algorithme on peut appliquer parmi ceux présentés et sur quel graphe, pour répondre à la question posée. L'objectif est d'obtenir la complexité en temps la plus faible possible, exprimée en termes du nombre de frontières fournies.

Un premier graphe est assez visible sur l'illustration : celui où un nœud est une extrémité de frontière, et une arête est le segment reliant deux extrémités d'une frontière. Il s'agit d'un graphe non orienté et pondéré (par la probabilité associée à une frontière), et on voit qu'il peut contenir des cycles. On sent donc que l'algorithme de Dijkstra pourrait être utilisé, mais il ne semble pas possible d'exprimer la question du sujet sous la forme d'une recherche d'un plus court chemin entre deux nœuds de ce graphe. Il faut donc trouver un autre graphe.

Si l'on se représente la situation de la famille qui voyage à un moment donné, on va l'imaginer au milieu d'une zone entourée de frontières (un pays). Sa situation va changer significativement lorsqu'elle va traverser l'une des frontières qui l'entoure, et atteindre un autre pays. On a donc un ensemble de situations possibles (se trouver dans chacun des pays), et un ensemble de transitions possibles vers d'autres situations (traverser une frontière). Cet ensemble de situations et transitions forme un graphe, où chaque situation correspond à un nœud, et chaque transition possible correspond à une arête.

On peut ainsi représenter le graphe obtenu sur notre exemple :



On remarque en haut à droite que deux pays peuvent avoir plusieurs frontières communes, donc que l'on peut avoir plusieurs transitions possibles entre deux situations données. Ce type de graphe est appelé multi-graphe par opposition aux graphes simples où deux nœuds ne peuvent être reliés directement que par une seule arête. Notez que notre nouveau graphe est à peu de choses près ce que l'on appelle le graphe dual du graphe des frontières.

Dans la suite et dans tous les problèmes de graphes implicites, nous allons continuer à utiliser ces termes de « situation » et « transition » pour représenter respectivement les nœuds et arcs ou arêtes du graphe caché sur lequel on a décidé de travailler (graphe implicite). On utilise ces termes d'une part pour bien faire la différence avec les nœuds et arcs d'un éventuel graphe donné explicitement dans le sujet, et d'autre part car essayer d'identifier les situations possibles d'un problème aide à trouver les graphes qui s'y cachent.

Maintenant que l'on a identifié un graphe implicite, il reste à étudier ses propriétés :

Situation : un pays (une zone vide entourée de frontières)

Transition : la traversée d'une frontière

Nombre de situations : $O(nb_{\text{Frontières}})$: un pays peut être formé avec un minimum de 3 frontières (zone triangulaire), donc le nombre de situations est du même ordre que le nombre de frontières

Nombre de transitions : $O(nb_{\text{Frontières}})$

Propriétés du graphe : c'est un multi-graphe, il est non orienté, peut contenir des cycles, et est pondéré

Maintenant que l'on a listé les propriétés de ce graphe, il reste à exprimer notre problème sous la forme d'une question « classique » sur ce graphe. On sait que le but est de trouver le chemin entre la situation de départ de la famille et sa destination, de telle sorte que la probabilité de traverser sans être pris soit maximale, donc que le produit des probabilités de traverser sur l'ensemble des transitions soit maximal. Pour se ramener à l'expression classique d'un problème de plus court chemin où il faut minimiser une somme, on peut considérer qu'il faut minimiser le produit des inverses des probabilités, puis passer au log, donc minimiser la somme de leurs logs : si l'on considère que la pondération d'une transition est égale à $-log$(probabilité de traverser sans être pris), alors la question revient à chercher le chemin le plus court entre la situation de départ et la situation d'arrivée.

Pondération d'une transition :
$-log$(probabilité de traverser).
La probabilité étant un nombre entre 0 et 1, la pondération est toujours positive.

Question posée : quelle est la longueur du plus court chemin entre deux situations données ?

On a donc transformé le problème en une simple question de plus court chemin dans un graphe pondéré dont les poids sont positifs.

Algorithme utilisé : on peut appliquer directement l'algorithme de Dijkstra sur ce graphe.

Complexité en temps :
$O(nb_{\text{Frontières}} \cdot \log(nb_{\text{Frontières}}))$

Avant de pouvoir appliquer Dijkstra sur ce graphe, il faudra une étape préalable consistant à le construire à partir de la description des segments. Il s'agit d'un problème intéressant sans être trop difficile, dont la résolution est laissée en exercice au lecteur.

IV Grille de couleurs

Sujet 2. *Un casse-tête est constitué d'une grille rectangulaire de R rangées de C cases colorées. Un pion noir se trouve dans la case située en haut à droite, tandis qu'un pion blanc se trouve dans la case située en bas à gauche, qui est de la même couleur que la case en haut à droite. Le but du jeu est d'échanger la position des deux pions, par une succession de mouvements. Chaque mouvement consiste à déplacer chacun des deux pions vers une case adjacente (partageant un côté) à sa case actuelle, avec la condition qu'après le déplacement, les deux pions doivent se trouver sur deux cases distinctes, mais de même couleur. Votre objectif est de trouver un algorithme capable de déterminer avec une bonne complexité en temps, s'il existe une solution à ce casse-tête et si oui, d'indiquer le nombre minimum de mouvements nécessaires pour y arriver.*

Comme pour tous les sujets présentés ici, le but ne va pas être d'inventer un algorithme mais de déterminer

lequel des trois algorithmes présentés peut s'appliquer et surtout, sur quel graphe implicite. Prenez un moment pour essayer de trouver lesquels avant de lire la suite.

Un premier graphe apparaît dans ce sujet d'une manière relativement classique : celui où chaque case de la grille est un nœud, et où chaque adjacence entre deux cases partageant un côté correspond à une arête entre les deux nœuds associés à ces cases. Nos pions sont ainsi des objets se déplaçant dans ce graphe. Cependant, si l'on cherche à exprimer le problème dans ce graphe, on est grosso modo obligé de réexpliquer l'ensemble du sujet, si ce n'est que l'on parlera de nœuds et déplacement vers des nœuds voisins au lieu de parler de cases et de cases adjacentes. On ne peut pas exprimer ce que l'on cherche par une question simple qui correspond à l'un de nos algorithmes classiques.

Pour trouver le bon graphe implicite, on va se demander quelles informations sont nécessaires pour décrire la situation dans laquelle se trouve si l'on interrompt le jeu en plein milieu. Les seules informations dont on a besoin pour décrire une situation quelconque sont les coordonnées des deux pions dans la grille. Ces situations peuvent tout à fait être considérées comme les nœuds d'un graphe, et il nous reste alors à analyser les propriétés de ce graphe implicite et à déterminer si l'on peut poser le problème de manière classique sur ce graphe :

Situation : les coordonnées de deux cases de même couleur où se trouvent les deux pions

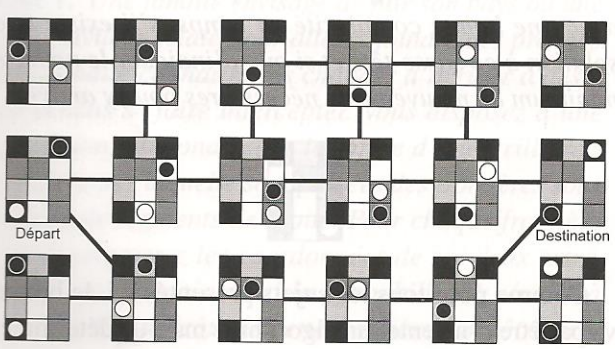
Transition : un mouvement valide qui mène d'une situation à une nouvelle situation où chaque pion s'est déplacé vers une case adjacente en respectant les conditions du sujet

Nombre de situations : $(RC)^2$

Nombre de transitions : $8(RC)^2$ (au maximum 16 mouvements pour une situation donnée)

Propriétés du graphe : il est non orienté, non pondéré

Pour vous aider à bien comprendre, voici une représentation possible du graphe correspondant à la grille donnée en exemple. Seules les situations accessibles depuis la situation de départ sont représentées.



Question posée : on cherche à aller de la situation initiale à la situation finale décrite dans le sujet en un minimum de mouvements possibles. Il s'agit donc simplement de chercher le plus court chemin entre deux situations.

Algorithme utilisé : la question correspond exactement à ce que permet de faire l'algorithme du parcours en largeur (BFS) présenté précédemment.

Complexité en temps : $O(nb_{\text{Transitions}}) = O((RC)^2)$.

V Ordre chronologique

Sujet 3. *On vous décrit une liste de personnes dont on ne dispose pas de l'âge exact mais uniquement des informations de la forme « Paul est strictement plus âgé que Claire », dont vous êtes certains de la validité. Vous souhaitez sélectionner un maximum de personnes de cette liste et les placer côte à côte le long d'une ligne, de la plus jeune à la plus âgée sans risque d'erreur, en vous basant uniquement sur ces informations. Trouvez un algorithme efficace qui détermine le nombre maximum de personnes qu'il est possible de sélectionner et ordonner de cette manière.*

Même si le sujet n'est ici pas présenté explicitement comme un problème de graphe, il est relativement aisé de le voir comme tel une fois que l'on y pense : chaque personne peut être vue comme un nœud, et chaque information peut être vue comme un arc entre deux nœuds. En continuant à utiliser notre vocabulaire de situations et transitions, même s'il est moins adapté ici, étudions les propriétés de ce graphe.

Situation : une personne

Transition : d'une personne à une personne dont une de nos informations dit qu'elle est plus âgée

Nombre de situations : $nb_{\text{Personnes}}$

Nombre de transitions : $nb_{\text{Relations}}$

Propriétés du graphe : orienté, non pondéré, acyclique (DAG). Un cycle signifierait que l'on a une contradiction, du type $(A < B < C < A)$, ce qui est impossible car la validité des informations fournie est garantie

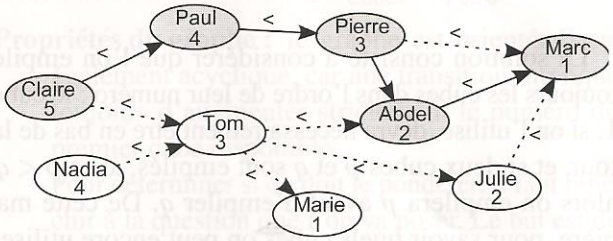
Question posée : quel est le plus long chemin qu'il est possible de former entre deux nœuds du graphe ?

Algorithme utilisé : on pose une question de type « plus long chemin » sur un graphe orienté acyclique, donc on s'oriente vers un algorithme dynamique. On peut en effet résoudre le problème en posant pour chaque nœud la question « Quel est le plus long chemin que l'on peut former à partir de ce nœud ? », et de manière évidente, on peut

répondre à cette question en reposant la question sur chacun des nœuds voisins.

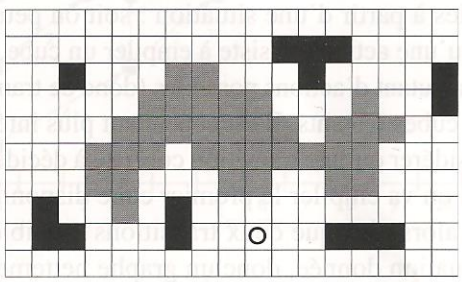
Complexité en temps : $O(nb_{\text{Relations}} + nb_{\text{Transitions}})$

L'illustration ci-dessous représente un exemple de graphe de relations. La réponse à la question posée est représentée dans chaque nœud.



VI Le tour du lac

Sujet 4. *Vous avez entrepris de faire une randonnée autour d'un grand lac, mais craignez d'avoir surestimé vos forces et souhaitez donc minimiser vos efforts en trouvant la manière la plus rapide possible de le faire. Vous disposez d'une carte sous la forme d'une grille rectangulaire de $R \times C$ cases, chacune pouvant être soit une case libre, soit une partie du lac, soit un obstacle. On vous garantit que le lac est connexe (en un seul morceau). Étant donné votre case de départ (une case libre), trouvez un algorithme qui calcule le nombre minimum de cases par lesquelles vous devez passer pour faire le tour du lac et revenir à la case de départ en ne passant que par des cases libres. Un déplacement consiste à aller vers l'une des cases libres qui partagent un côté avec la case où vous vous trouvez.*

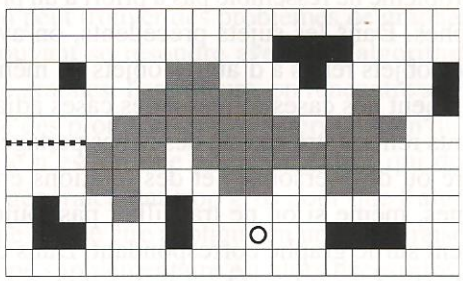


Comme pour la grille de couleurs, on reconnaît un graphe relativement classique dont les nœuds sont les cases et les arêtes sont les adjacences entre cases. Il est cependant difficile d'exprimer la notion de « faire le tour » dans ce graphe, et là encore la solution va consister à trouver un autre graphe, dans lequel nous pourrons poser une question beaucoup plus classique.

Raisonnons à nouveau en termes de situation : imaginons que vous êtes en train de faire la randonnée, que vous êtes tout d'un coup interrompu et devez demander à quelqu'un d'autre de prendre le relais. Comment pouvez-vous décrire votre situation actuelle en un minimum d'informations à cette personne, de sorte qu'elle soit en mesure de déterminer la meilleure suite possible ? Décrire l'ensemble du chemin déjà parcouru demanderait de fournir beaucoup trop d'informations. Il faut au minimum donner vos coordonnées actuelles, mais ce n'est pas suffisant : supposons par exemple que ces coordonnées soient celles de la case juste en dessous de votre point de départ. Si l'on ne dispose que de cette information, comment différencier le cas où on vient tout juste de commencer la randonnée, de celui où l'on a déjà fait tout le tour et où on est sur le point d'arriver ?

On sent qu'il faut indiquer si l'on est déjà passé autour du lac ou non, mais définir ce que signifie « être déjà passé autour du lac » est loin d'être évident. Réfléchissez-y avant de lire la suite.

Une solution consiste à considérer une ligne entre le lac et le bord de la grille. On peut par exemple considérer une demi-droite qui part du coin haut gauche d'une case du lac qui se trouve le plus vers la gauche de la grille possible et va vers la gauche. Quel que soit son point de départ, faire le tour du lac une fois implique de traverser cette ligne au moins une fois. Attention cependant : si on la traverse dans un sens puis qu'on la traverse dans l'autre sens juste après, c'est comme si on ne l'avait pas traversée du tout. Pour être certain d'être bien passé de l'autre côté, il faut en fait l'avoir traversé un nombre impair de fois. On peut ainsi décrire sa situation avec deux informations : ses coordonnées, et le fait que l'on ait ou non traversé cette ligne un nombre impair de fois.



Ceci nous donne le graphe implicite suivant :

Situation : les coordonnées de la case où l'on se trouve, et un booléen indiquant si l'on a traversé la demi-droite un nombre impair de fois

Transition : d'une situation donnée, on a une transition vers chacune des cases libres adjacentes, en inversant la valeur du booléen si l'on doit traverser la demi-droite pour atteindre la nouvelle case

Nombre de situations : $2RC$

Nombre de transitions : $4RC$

Propriétés du graphe : non orienté, non pondéré

Question posée : quel est le plus court chemin pour aller de la case de départ avec le booléen à faux (pas traversé la demi-droite), vers la case de départ mais le booléen à vrai (traversé la demi-droite un nombre impair de fois) ?

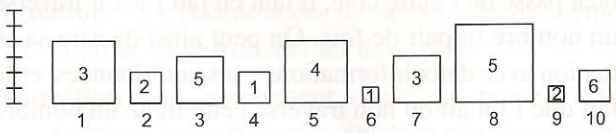
Algorithme utilisé : dans ce graphe implicite, la question du sujet a été transformée en une simple question de longueur de plus court chemin entre deux nœuds d'un graphe non orienté, non pondéré, ce qui est exactement ce que permet de résoudre l'algorithme du parcours en largeur.

Complexité en temps : $O(nb_{\text{Transitions}}) = O(RC)$.

VII Tour de babel

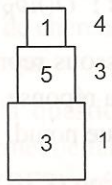
Sujet 5. *Vous disposez de N cubes, chacun ayant un poids P_i et une hauteur entière H_i (avec $N \leqslant 100$, $P_i \leqslant 1000$, $H_i \leqslant 1000$). Vous souhaitez empiler certains de ces cubes pour former une tour aussi haute que possible, mais sans dépasser un poids limite $lim_{Poids} \leqslant 10000$. Écrivez un algorithme qui détermine la hauteur maximale que vous pourrez atteindre.*

Par exemple, on peut disposer des 10 cubes présentés sur l'illustration suivante. La valeur inscrite au centre des carrés est le poids des cubes.



Ce problème ne ressemble pas a priori à un problème de graphes. Dans les sujets précédents, on avait toujours des objets reliés à d'autres objets du même type, que ce soient des cases reliées à des cases adjacentes, des points reliés entre eux ou des zones partageant une frontière ou des personnes et des relations entre ces personnes, même si on ne travaillait pas toujours directement sur le graphe correspondant. Dans ce sujet, les seuls objets sont les cubes, et ils n'ont pas de relations particulières entre eux qui puissent constituer un graphe.

On peut cependant faire apparaître un graphe en réfléchissant en termes de situations. Imaginons que l'on a commencé à empiler des cubes et que l'on est interrompu. De quelle informations a-t-on besoin au minimum si l'on veut pouvoir reprendre où l'on en était, et continuer à chercher à atteindre la plus grande hauteur possible ? Lister tous les cubes que l'on a empilé n'est pas envisageable, car on aurait un nombre de situations possibles égal à 2^N, ce qui donnerait une complexité en temps exponentielle quel que soit l'algorithme utilisé. Il nous faut cependant un moyen de savoir quels cubes on a encore le droit d'utiliser pour continuer à construire la tour.



La solution consiste à considérer que l'on empile toujours les cubes dans l'ordre de leur numéro : le cube 1, si on l'utilise, devra nécessairement être en bas de la tour, et si deux cubes p et q sont empilés, avec $p < q$, alors on empilera p avant d'empiler q. De cette manière, pour savoir quels cubes on peut encore utiliser pour continuer à former la tour, on a uniquement besoin de connaître le numéro du premier cube que l'on a le droit d'utiliser. Le numéro du premier cube disponible ne suffit cependant pas à décrire la situation assez précisément pour continuer : on a besoin de vérifier que l'on ne va pas dépasser la limite de poids, donc on va ajouter à la description de notre situation le poids que l'on a déjà accumulé. On pourrait avoir envie de stocker également la hauteur déjà atteinte, mais celle-ci ne va pas influencer la manière dont on va continuer à empiler des cubes : quelle que soit la hauteur déjà atteinte, on va empiler ce qui reste de sorte d'ajouter la plus grande hauteur possible sans dépasser la limite de poids.

En exprimant une situation de manière concise, on a en fait considéré un graphe dont les nœuds sont ces situations. Les transitions correspondent aux différentes actions que l'on peut effectuer à partir d'une situation donnée. Il y a deux manières de considérer les actions possibles à partir d'une situation : soit on peut considérer qu'une action consiste à empiler un cube, auquel cas on a autant d'actions possibles (donc de transitions) que de cubes restants. Il est cependant plus intéressant de considérer qu'une transition consiste à décider si oui ou non on va empiler le premier cube disponible, car on n'a alors plus que deux transitions possibles pour une situation donnée, donc un graphe nettement plus réduit.

Pour bien comprendre notre nouveau graphe, considérons la situation présentée dans l'illustration précédente : le premier cube disponible est le cube 5, et nous avons déjà atteint un poids de 9. Si l'on décide d'empiler le cube 5, on se retrouve dans une nouvelle situation, où le premier cube disponible est le cube 6, et où nous avons atteint un poids de 14. Si par contre on décide de ne pas empiler le cube 5, on se retrouve alors dans une situation où le premier cube disponible est le cube 6, mais nous sommes toujours avec un poids total de 9.

Analysons les propriétés de ce graphe :

Situation : une paire (c,P) : le premier cube disponible et le poids déjà atteint

Transitions : empiler le premier cube disponible si cela ne dépasse pas le poids limite $(c+1,P+Pc)$, ou ne pas l'empiler $(c+1,P)$

Nombre de situations : $nb_{\text{Cubes}}lim_{\text{Poids}}$

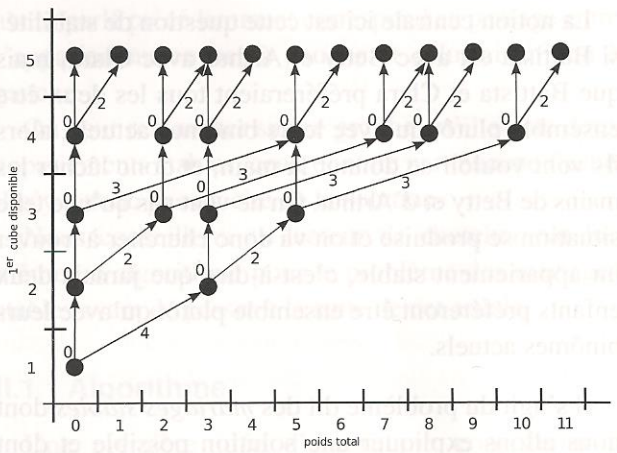
Nombre de transitions : $2nb_{\text{Cubes}}lim_{\text{Poids}}$

Propriétés du graphe : le graphe est orienté. Il est également acyclique, car une transition implique toujours d'augmenter strictement le numéro du premier cube disponible.

Pour déterminer si on doit le pondérer, il faut réfléchir à la question que l'on va poser. Le but est de maximiser la hauteur totale atteinte, or cette hauteur n'est pas décrite par notre situation. Si l'on utilise comme poids d'une transition la hauteur que l'on ajoute à la tour en effectuant cette transition, alors notre but va être de trouver un chemin dont la somme des poids est la plus grande. Cette pondération nous convient donc très bien.

Pondération : la hauteur ajoutée à la tour.

L'illustration suivante représente une partie du graphe. On place les nœuds dans un repère, avec le poids total en abscisse, et le numéro du premier cube disponible en ordonnée.



Question posée : quelle est la longueur du plus long chemin partant de la situation initiale (cube 1, poids 0)

Algorithme utilisé : notre graphe est un graphe orienté acyclique, et on pose une question de type plus long chemin, donc on va utiliser un algorithme dynamique.

Complexité en temps :

$O(nb_{\text{Transitions}}) = O(nb_{\text{Cubes}}lim_{\text{Poids}})$

VIII Conclusion

Au travers de cinq sujets, nous avons pu constater que les problèmes de graphes peuvent prendre diverses formes qui peuvent être éloignées de la présentation usuelle d'un graphe. Nous avons vu qu'avec seulement trois algorithmes classiques, il était possible de résoudre une grande variété de problèmes, et que la difficulté de la résolution consiste avant tout à trouver sur quel graphe travailler, celui-ci étant rarement le premier qui vient à l'esprit à la lecture du sujet.

Nous avons introduit une technique qui facilite la reconnaissance des graphes implicites cachés dans les sujets, et qui consiste à s'imaginer en train de résoudre le sujet sur un exemple et à se demander comment on pourrait décrire en un minimum d'informations la situation dans laquelle on se trouve au moment où l'on est interrompu. Ces informations doivent être tout juste suffisantes pour pouvoir reprendre la résolution. Les situations possibles selon cette description forment alors les nœuds d'un graphe, et les transitions possibles à partir de ces situations constituent ses arcs ou arêtes.

Une fois les situations et transitions du graphe implicite déterminées, nous avons pu voir qu'en analysant les propriétés du graphe obtenu et en exprimant la question posée par le sujet du point de vue de ce graphe, on se ramenait à des questions relativement classiques posées sur ce graphe, pouvant être résolues par des algorithmes classiques appliqués tels quels.

Les exemples présentés ont cependant été sélectionnés pour être relativement simples. Tout d'abord car nous avons sélectionné des sujets correspondant aux trois algorithmes présentés dans la première partie alors que l'on peut trouver des problèmes de graphes implicites pouvant se résoudre avec tout algorithme classique de graphes. Par ailleurs, on rencontre souvent en pratique des problèmes où la détermination du graphe implicite n'est qu'une première étape, qui donne un graphe généralement trop gros pour que l'algorithme classique puisse être appliqué en un temps raisonnable. Une étape supplémentaire est alors nécessaire, consistant à appliquer un ensemble de techniques de réduction de la taille du graphe. Il s'agit souvent d'une des étapes les plus intéressantes de la résolution de ce style de sujets.

Références

[1] Cormen, T., Leiserson, C., Rivest, R., Stein, C., *Introduction à l'algorithmique*, Dunod, 2004.