Algorithme de Dijkstra et Algorithme de Floyd-Warshall

Tatiana Babicheva, Sergey Babichev

1 Recherche du chemin le plus court

Chaque personne qui se rend sur la route examine l'état de la circulation. La plupart des gens, qui choisissent un trajet, pensent que l'état des transports ne changera pas tout au long du trajet, c'est-à-dire qu'ils choisissent le chemin le plus court sur un graphe orienté à pondération constante.

Il est très facile de choisir le chemin le plus court sur des graphes simples. Et si nous cherchions le chemin le plus court sur un grand graphe ?

Nous allons faire la rencontre du plus célèbre des algorithmes non réordonnés, l'**Algorithme de Dijkstra**.

Dans l'algorithme de Dijkstra, les chemins les plus courts d'un sommet donné vers tous les autres sont recherchés. Nous pouvons supposer que l'algorithme de Dijkstra construit un graphe orienté des plus courts chemins sans cycles. Les graphes sans cycles sont appelés des arbres, donc l'algorithme de Disktra construit l'arbre des chemins les plus courts, en anglais Shortest Path Tree (SPT). Cet algorithme est nécessaire non seulement dans les tâches de voyage, mais également dans les réseaux informatiques. Il existe de nombreuses variantes, regardons celle que propose Dijkstra lui-même.

Cet algorithme consiste en un nombre fini d'étapes (vous comprenez qu'un algorithme consistant en un nombre infini d'étapes est inutile, car il ne se terminera jamais ?). Cet algorithme est de complexité polynomiale. Plus précisément, pour n sommets et a arcs, le temps est en $O((n+a)\log n)$, voire en $O(a+\log n)$. Chaque étape consiste à ajouter exactement un sommet au SPT déjà construit auparavant. Au tout début de l'algorithme, le SPT est constitué d'un seul sommet *racine*, que nous appellerons s, à partir du mot *source*. Pour l'algorithme, nous avons besoin d'un tableau d, les valeurs des éléments d_i seront la longueur du plus court chemin entre le sommet racine et le sommet i. Initialement, tous les éléments de ce tableau contiennent l'infini, à l'exception de l'élément avec le numéro s. La matrice w contient des éléments w_{ij} qui sont égaux au poids entre le sommet i et le sommet j s'ils sont voisins ou l'infini s'ils ne sont pas voisins.

Une étape, ou comme le disent les algorithmiciens, une *itération* de l'algorithme est la suivante :

1. Le sommet racine (source) est mis dans le SPT.

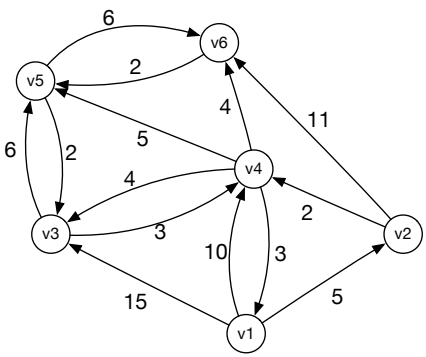
2. À chaque étape, une arête est ajoutée au SPT, qui constitue le chemin le plus court de la source au non-SPT. L'opération consistant à remplacer l'ancienne longueur du plus court chemin par une nouvelle est appelée *relaxation*.

3. Les sommets sont réordonnés dans le SPT dans l'ordre de la longueur du chemin depuis le sommet initial.

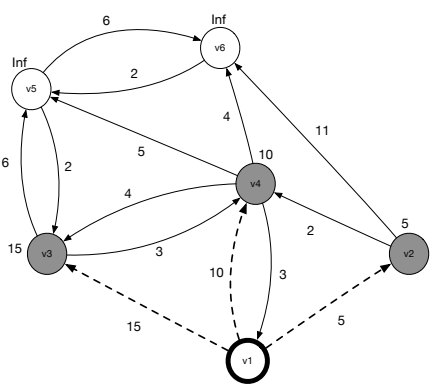
Prenons un exemple.

Écrivons à côté de chaque sommet la valeur de la longueur du chemin déjà atteinte depuis le sommet racine.

Le graphe initial

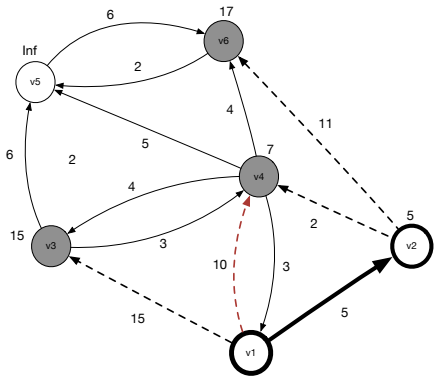


Mettons v_1 dans le SPT, et marquons v_2, v_3 et v_4 en gris, ils sont nos candidats pour les sommets entre lesquels choisir pour la prochaine étape.



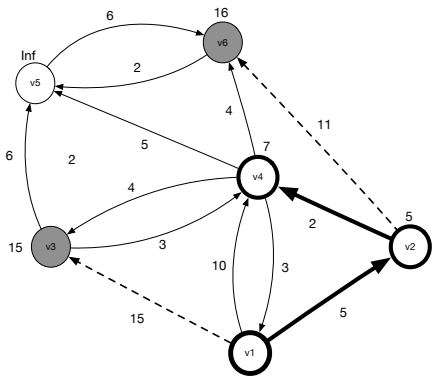
Parcourons tous les sommets gris. Choisissons, par exemple, le sommet v_2. Nous ajustons la distance de celui-ci. Relaxation: l'ancien chemin le plus court ($v_1 \to v_4$)

est remplacé par $(v_1 \to v_2 \to v_4)$. Nous recalculons la longueur du plus court chemin et écrivons le nouveau nombre à côté de v_4. Le sommet v_2 est le plus proche du SPT, colorons-le en noir.

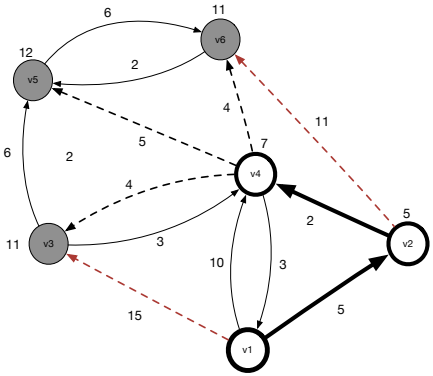


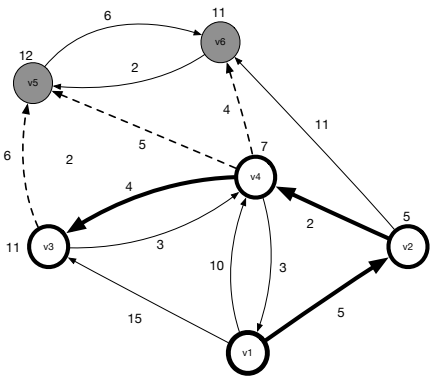
Dans le SPT nous avons v_1 et v_2, et hors du SPT tous les autres (non-SPT).

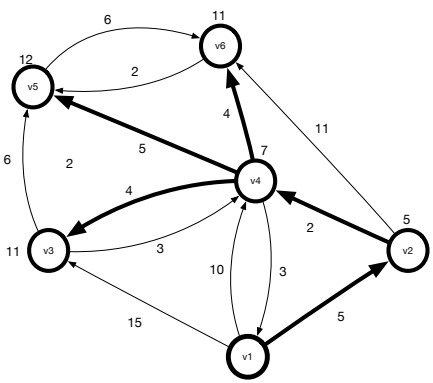
Il est facile de vérifier que le sommet v_4 est maintenant dans le SPT. Faisons toutes les relaxations.



En regardant le sommet v_5 en tant que candidat, un groupe de relaxation est passé: $(v_1 \to v_2 \to v_6)$ est remplacé par $(v_1 \to v_2 \to v_4 \to v_6)$, $(v_1 \to v_3)$ par $(v_1 \to v_2 \to v_4 \to v_3)$







Cet algorithme a beaucoup d'implémentations, nous avons considéré l'une des plus simples.

La sortie de l'algorithme est le SPT souhaité.

Ainsi, il est possible, par exemple, d'estimer le temps nécessaire pour se rendre à chaque emplacement de la carte ; des schémas similaires sont utilisés lors de l'estimation du temps de trajet dans diverses applications mobiles.

Une animation d'un algorithme de Dijkstra est montrée sur wikipédia Dijkstra

2 Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall est connu depuis 1962. Il définit les chemins réseau les plus courts, quand le réseau (graphe) est décrit par une matrice d'adjacence. Rappelons que dans la matrice d'adjacence, nous supposons que le nombre dans la i ème rangée et la j ème colonne de la matrice d'adjacence est la métrique de l'arête entre les sommets i et j. En changeant un peu la representation on peut considerer que dans la matrice d'adjacence $C_{ij} = \infty$, dans le cas où les sommets i et j ne sont pas voisins. Comme entrée l'algorithme prend la matrice d'adjacence mise à jour et comme sortie d'algorithme la matrice va contenir en élément C_{ij} la métrique de chemin le plus court de P_i à P_j.

Contrairement à l'algorithme de Dijkstra, qui produit des résultats incorrects quand il existe des chemins avec une métrique négative, cet algorithme permet à de tels chemins d'exister. Il existe une autre restriction : il ne devrait pas y avoir de cycles avec une longueur négative, mais pour résoudre les problèmes de routage dans les systèmes réels (par exemple, les systèmes distribués), ces restrictions ne sont pas significatives.

L'algorithme lui-même peut être décrit sous une forme récurrente :

$$D_{ij}^{(k)} = \begin{cases} C_{ij}, & \text{si } k = 0, \\ \min\left(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\right), & \text{si } k \geqslant 1 \end{cases}$$

et en faisant cela, il constitue l'équation de Bellman pour un problème de programmation dynamique. Cette tâche permet une solution pratique «bottom-up » , ainsi il peut être écrit sous une forme itérative :

```
void FloydWarshall(int **d, int n) { // il est supposé que le tableau est de tail
  for (int i = 0; i < n; i++) {
    for (int s = 0; s < n; s++) {
      for(int t = 0; t < n; t++) {
        if (s != t) {
          if (d[s][t] > d[s][i] + d[i][t]) {
           d[s][t] = d[s][i] + d[i][t];
          }
        }
```

```
      }
    }
  }
}
```

Les éléments inaccessibles dans la matrice doivent être marqués avec de grands nombres, par exemple, initialisés avec une constante `INF=1000000000`.

Cependant, en appliquant l'algorithme de Floyd-Warshall, nous obtenons uniquement les métriques des chemins les plus courts. Pour la plupart des tâches, ce ne sont pas les informations les plus importantes. En fait, nous n'avons pas besoin de la métrique D_{ij}, mais du voisin le long du chemin le plus court. L'algorithme ne change pas en principe, une nouvelle matrice B de taille $N \times N$ est ajoutée, B_{ij} est le «bon» voisin le long du chemin le plus court. Alors, c'est une matrice des meilleurs voisins. Les changements dans l'algorithme, afin de calculer y compris le meilleur voisin, sont très minimes:

```
void floyd(int **dd, int **b, int n) { // d[n][n], b[n][n]
  for (int i = 0; i < n; i++) {
    for (int s = 0; s < n; s++) {
      if(b[s][i] >= 0) {
        for(int t = 0; t < n; t++) {
          if (s != t) {
            if (d[s][t] > d[s][i] + d[i][t]) {
             d[s][t] = d[s][i] + d[i][t];
             b[s][t] = b[s][i];
            }
          }
        }
      }
    }
  }
```

Regardons les étapes de l'algorithme pour le réseau dans la Fig. 2.1. La matrice de connexion d'origine D ressemble à ceci:

$D^{(0)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	∞	∞	∞	3	2
P_1	∞	0	∞	∞	∞	4
P_2	∞	7	0	3	12	∞
P_3	∞	∞	∞	0	∞	∞
P_4	∞	∞	2	∞	0	∞
P_5	∞	∞	4	2	5	0

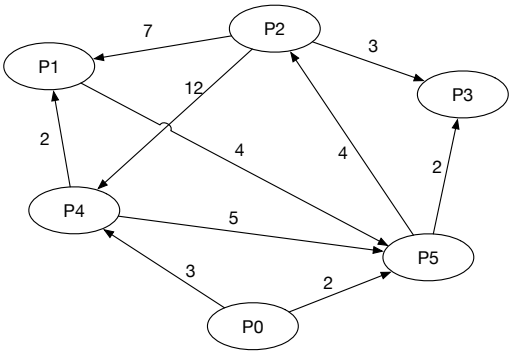


Рис. 2.1: Réseau simple

Avant le début de l'algorithme, la matrice des meilleurs voisins est initialisée comme suit:

$$B_{ij}^{(0)} = \begin{cases} j, & \text{si } C_{ij} \neq 0, \\ NIL, & \text{si } C_{ij} = 0. \end{cases}$$

Pour plus de commodité, nous noterons dans le tableau l'absence de chemin par le symbole '-' et non par le symbole NIL.

$B^{(0)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	-	-	-	4	5
P_1	-	1	-	-	-	5
P_2	-	1	2	3	4	-
P_3	-	-	-	3	-	-
P_4	-	-	-	-	4	-
P_5	-	-	2	3	4	5

La matrice initiale $D^{(0)}$ contient les métriques de tous les meilleurs itinéraires de longueur unitaire et la matrice $B^{(0)}$ les meilleurs voisins pour ces itinéraires. Chaque itération suivante de l'algorithme ajoute à la matrice $B^{(i+1)}$ les éléments associés à des routes de longueur i incrémenté de 1, et à la matrice $D^{(i+1)}$ ces métriques itinéraires.

Après la première itération, les matrices ne changent pas.

Après la deuxième itération, on obtient ce qui suit (les caractères en gras marquent les éléments modifiés des tableaux):

$D^{(2)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	∞	∞	∞	3	2
P_1	∞	0	∞	∞	∞	4
P_2	∞	7	0	3	12	**11**
P_3	∞	∞	∞	0	∞	∞
P_4	∞	∞	∞	∞	0	∞
P_5	∞	∞	4	2	5	0

$B^{(2)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	-	-	-	4	5
P_1	-	1	-	-	-	5
P_2	-	1	2	3	4	**1**
P_3	-	-	-	3	-	-
P_4	-	-	-	-	4	-
P_5	-	-	2	3	4	5

Après :

$D^{(3)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	∞	∞	∞	3	2
P_1	∞	0	∞	∞	∞	4
P_2	∞	7	0	3	12	11
P_3	∞	∞	∞	0	∞	∞
P_4	∞	∞	∞	∞	0	∞
P_5	∞	**11**	4	2	5	0

$B^{(3)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	-	-	-	4	5
P_1	-	1	-	-	-	5
P_2	-	1	2	3	4	1
P_3	-	-	-	3	-	-
P_4	-	-	-	-	4	-
P_5	-	**2**	2	3	4	5

Le résultat des quatrième et cinquième itérations coïncide avec le résultat de la troisième.

La sixième itération (la dernière) :

$D^{(6)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	**13**	**6**	**4**	3	2
P_1	∞	0	**8**	**6**	**9**	4
P_2	∞	7	0	3	12	11
P_3	∞	∞	∞	0	∞	∞
P_4	∞	∞	∞	∞	0	∞
P_5	∞	11	4	2	5	0

$B^{(6)} =$

	P_0	P_1	P_2	P_3	P_4	P_5
P_0	0	**5**	**5**	**5**	4	5
P_1	-	1	**5**	**5**	**5**	5
P_2	-	1	2	3	4	1
P_3	-	-	-	3	-	-
P_4	-	-	-	-	4	-
P_5	-	2	2	3	4	5

Ceci termine l'algorithme. En comparant ses résultats avec ceux de l'utilisation répétée de l'algorithme de Dijkstra, nous sommes convaincus de leur identité.

Les questions sur lesquelles réfléchir

1. La constante `INF` peut-elle avoir une valeur de 2 000 000 000? Si oui, dans quelles conditions?

2. Quels problèmes peuvent survenir dans la fonction qui implémente l'algorithme de Floyd-Warshall, en supposant que les métriques peuvent être des nombres négatifs ? Suggérez un moyen de les résoudre. Et si les métriques sont représentées sous forme de nombres réels ?