Algorithme de Dijkstra : terminaison, correction et complexité

Julie Parreaux

2018-2019

Référence du développement : Cormen [1, p.609]

Leçons où on présente le développement : 901 (Structure de données) ; 925 (Graphe) ; 926 (Analyse : complexité) ; 927 (Analyse : Correction) ; 931 (Schéma algorithmique).

1 Introduction

L'algorithme de Dijkstra est un grand classique pour calculer le plus court chemin dans un graphe à partir d'une origine unique. Pour la correction de cet algorithme, l'ensemble des poids doivent être positifs ou nuls. De plus, celle-ci se prouve à l'aide d'un invariant de boucle qui n'a rien de trivial. L'algorithme de Dijkstra demande l'implémentation d'une file de priorité (qui n'est pas une structure de données simple). On voit alors apparaître la jeu des structures de données dans le calcul de la complexité d'un algorithme.

Remarques sur le développement

1. Présentation de l'algorithme.
2. Preuve de sa terminaison.
3. Preuve de sa correction.
4. Étude de sa complexité.

2 Étude de l'algorithme de Dijkstra

Présentation de l'algorithme
Objectif : chemin le plus court à origine unique. Pour $a \in V$, on veut calculer $d : V \to \{+\infty\}$ tel que $\forall v \in V$, $d(v) = dist(s, v)$, où s est la source. (Bien implémenté, il y a de meilleurs performances de Bellman-Ford.)

Hypothèse : Soit $G = (S, A)$ un graphe orienté pondéré avec un poids positif ou nul $(\forall (u, v) \in E, w(u, v) \geq 0)$.

Principe : On choisi parmi tous les sommets celui dont la distance *dist* est minimal (le plus court chemin) et on relâche les autres. C'est un algorithme glouton dont le résultat est optimal.

Notation : On note la fonction $dist : V \times V \to \mathbb{N} \cup \{+\infty\}$ la longueur du plus court chemin entre u et v.

Remarque. La file de priorité peut être remplacée par un tri topologique sur le graphe.

Algorithm 1 Algorithme de Dijkstra calculant le plus court chemin à partir d'un sommet initial.

Entrée : G le graphe ; w le poids ; s la source

```
1:  function DIJKSTRA(G, w, s)
2:      for tout v ∈ G.S do                          ▷ Source initiale
3:          v.d ← ∞                                  ▷ Majore le poids jusqu'à l'origine
4:          v.π ← NIL                                ▷ Prédécesseur
5:      end for
6:      s.d = 0                                      ▷ s est la source
7:      E ← ∅
8:      F = G.S                                      ▷ F est une file de priorité
9:      while F ≠ ∅ do
10:         u ← Extraire-Min(F)
11:         E ← E ∪ {u}
12:         for tout v ∈ Adj[u] do
13:             if v.d > u.d + w(u, v) then
14:                 v.d = u.d + w(u, v)
15:                 v.π = u
16:             end if
17:         end for
18:     end while
19: end function
```

Terminaison Si $F = \emptyset$, alors $E = S$. D'où la terminaison. On pose le variant suivant : $F = S \setminus E$.

Initialisation $F = S$ donc $E = \emptyset$.

Hérédité Extraire un sommet de $S \setminus E$ pour l'ajouter à E : ok

Correction On pose l'invariant de boucle : "à chaque itération de la boucle while, $\forall v \in E$, $v.d = dist(s, v)$. On montre par induction cet invariant.

Initialisation Comme $E = \emptyset$, l'invariant est vrai.

Conservation Montrons qu'à chaque itération "$u.d \neq dist(s, u)$ pour le sommet ajouté à E. Par l'absurde, supposons par l'absurde que u est le premier sommet pour lequel $u.d \neq dist(s, u)$ quand on ajoute à E.

- $\rightarrow$ $u \neq s$ car $u.d = 0$ et $dist(s, s) = 0$. Donc $E \neq \emptyset$.
- $\rightarrow$ Il existe un chemin de s à u sinon $u.d = +\emptyset = dist(u, v)$ (contradiction avec l'hypothèse).
- $\rightarrow$ Il existe un plus court chemin p de s à u où $s \in E$ et $u \in S \setminus E$ (car existence d'un chemin). On peut alors décomposé p de la manière suivante

$$s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$$

(p_1 ou p_2 peut ne pas avoir d'arc) où y est le premier sommet appartenant à $S \setminus E$ et x sont prédécesseur.
$y.d = dist(s, y)$ quand u est ajouté à E. Comme $x \in E$, alors $x.d = dist(s, x)$ si s lors de son ajout à E. Dans ce cas, l'arc (x, y) est relâché à cet instant. Donc $y.d = dist(s, y)$ (convergence).

- $\rightarrow$ On souhaite faire apparaître la contradiction : $dist(s, y) \leq dist(s, u)$.
$y.d = dist(s, y) \leq dist(s, u) \leq u.d$ (car y arrive avant u sur le chemin le plus court et $dist$ est minimal).
Mais comme $u, y \in S \setminus E$ lors du choix de u, $u.d \leq y.d$ (car choisi par F). Donc, on a l'égalité (contradiction).

D'où la correction.

Complexité Supposons que le graphe est représenter par une liste d'adjacence. Sa complexité est alors

$$cout(\text{Dikjstra}) \leq |S|(cout(\text{Inserer}) + cout(\text{Extraire}) + \underbrace{|A|(\text{Diminuer-Cle})}_{\text{analyse par agrégat}}$$

Sa complexité dépend de la structure de données implémentant E.

— tableau où on stocke $v.d$ pour chaque sommet (non trié).

cout(Inserer) = 1 = cout(Diminuer-Cle)
cout(Extraire-Min) = O(S) (parcourir tout le tableau)

Dans le cas d'un graphe peu dense $A = O(S^2 / \log S)$.

— tas binaire

cout(Inserer) $O(S)$ pour construire le tas entier ($O(1)$ est amortie)

cout(Diminuer-Cle) $O(\log S)$

cout(Extraire-Min) $O(\log S)$

$O((S + A) \log S) = O(A \log S)$ si tous les sommets accessibles depuis l'origine.

— Tas Fibonacci dans le cas optimal (motive l'existence de ce tas)

cout(Diminuer-Cle) $O(1)$

cout(Extraire-Min) $O(\log S)$ pour S

$O(S \log S + A)$

Références

[1] Rivest R. Stein C. Cormen T., Leiserson C. *Algorithmique, 3ème édition*. Dunod, 2010.