

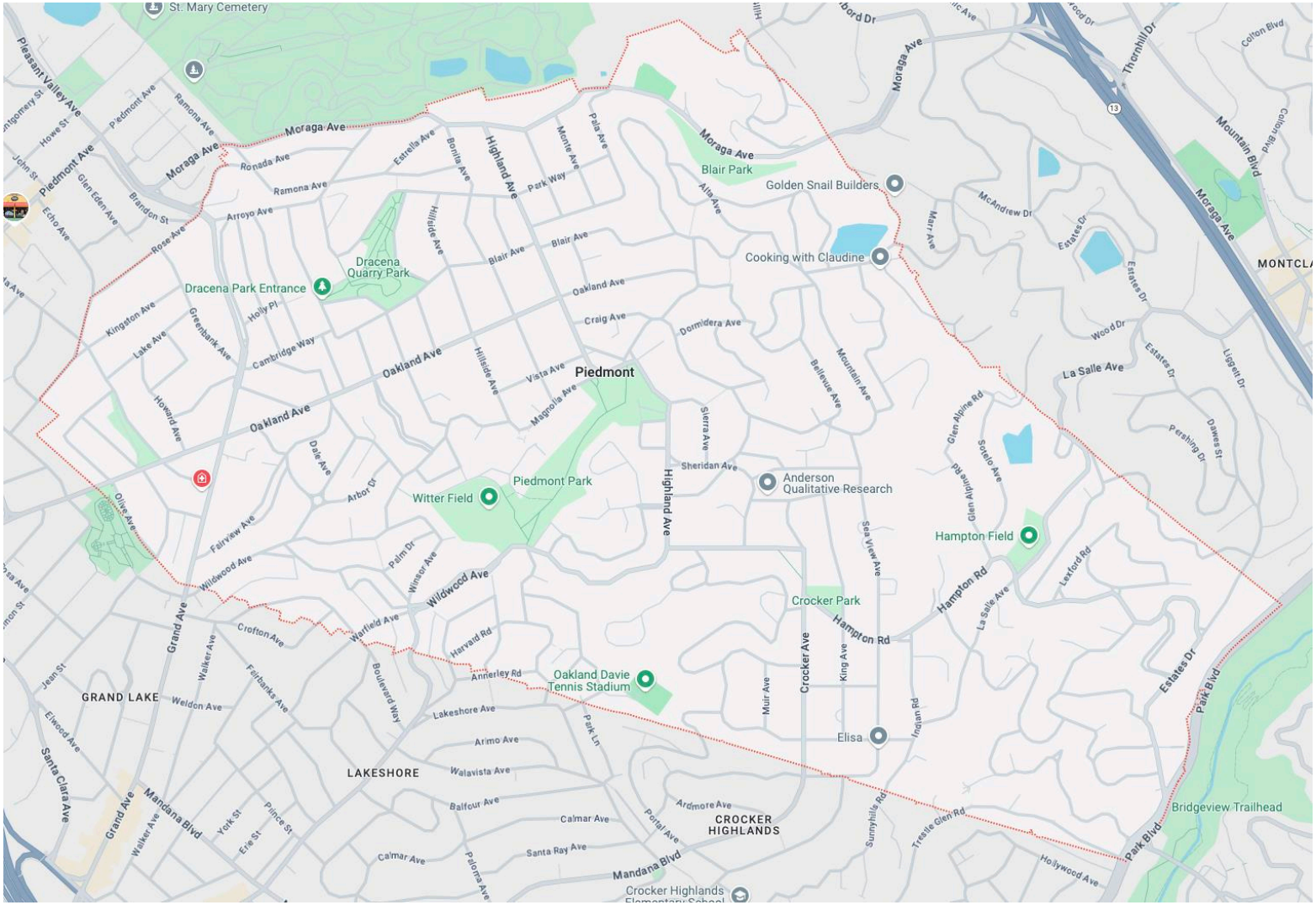


Paramètres de centralité

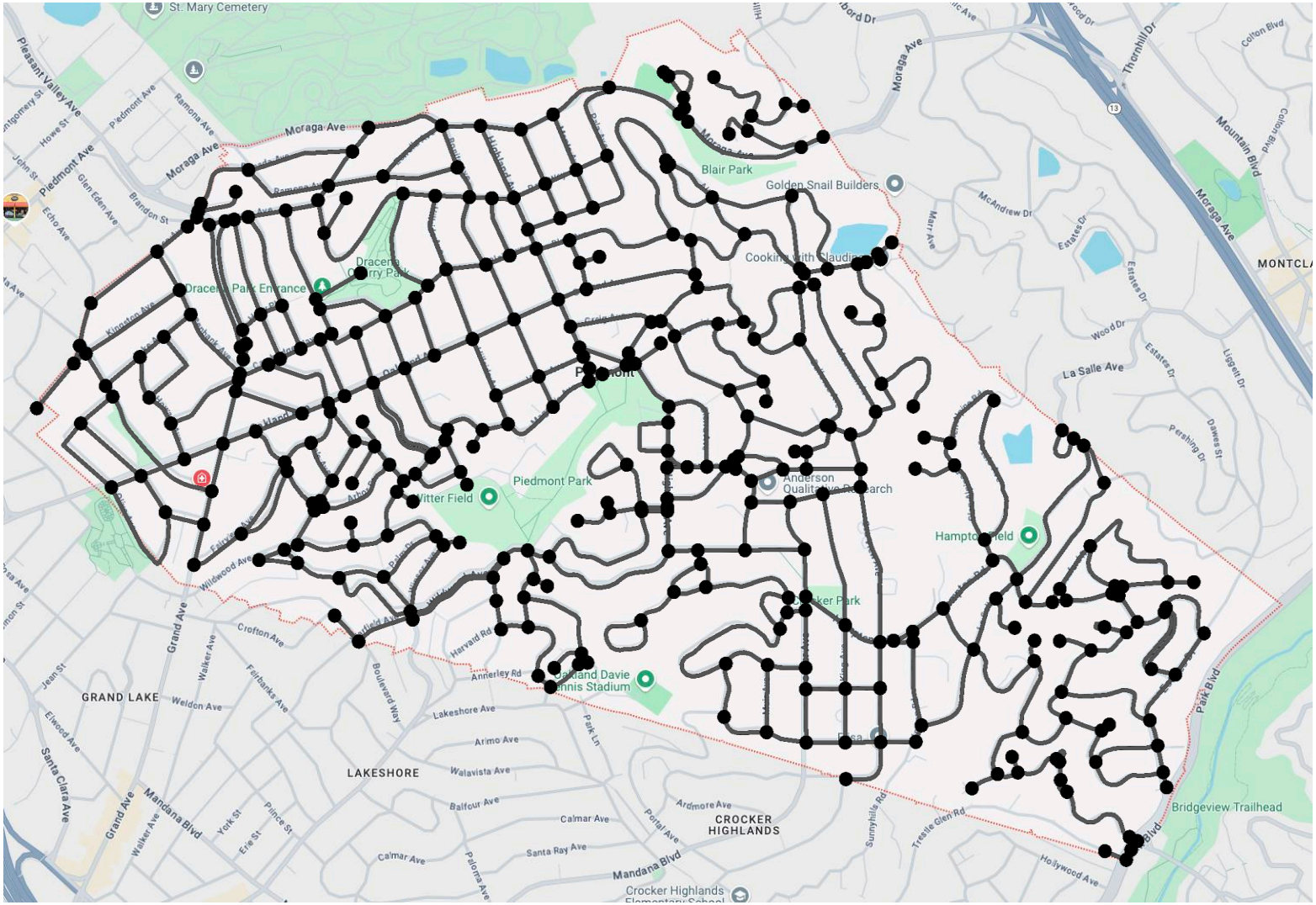
Alexandre Pinlou

alexandre.pinlou@umontpellier.fr

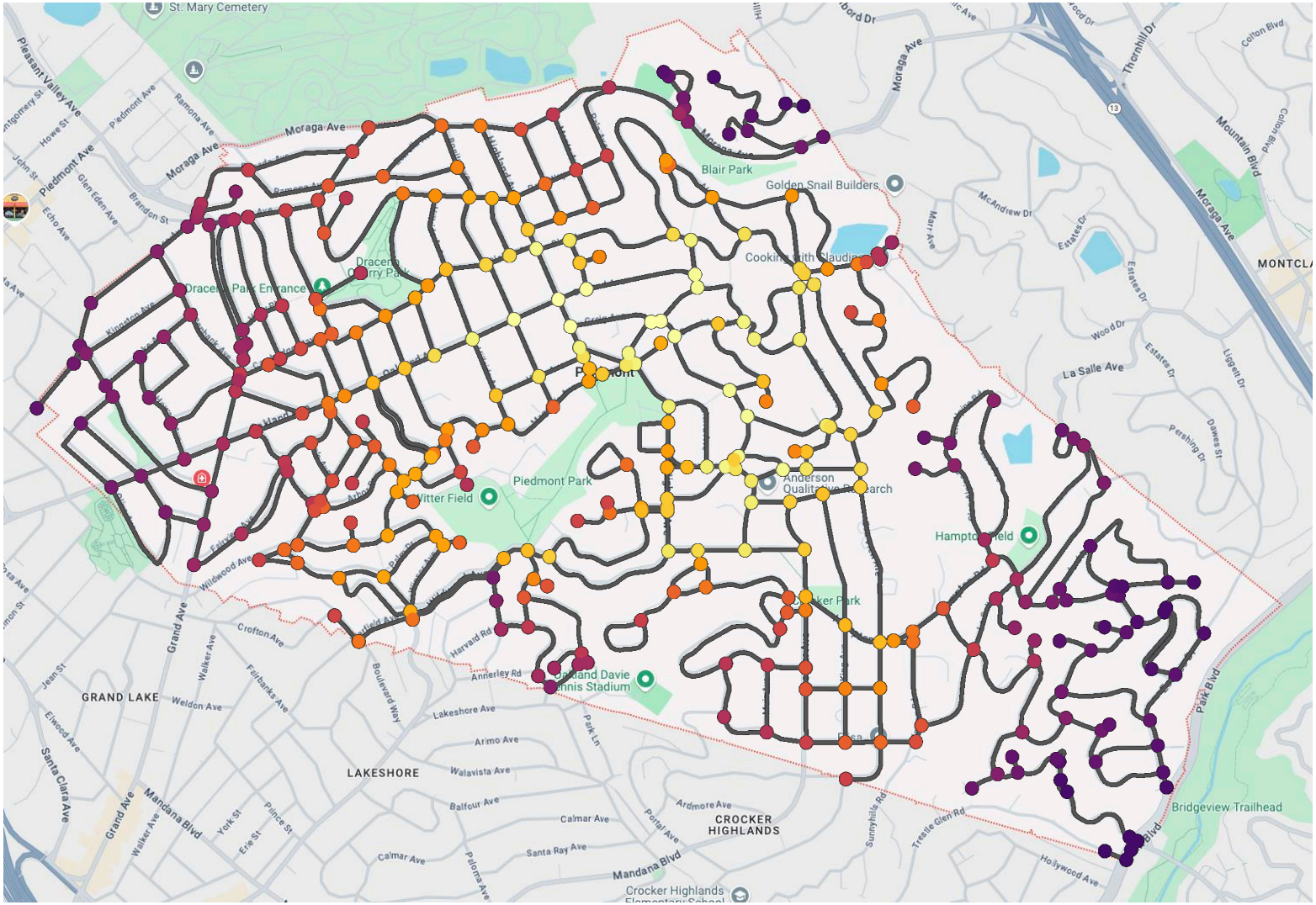
Quelques exemples de réseaux



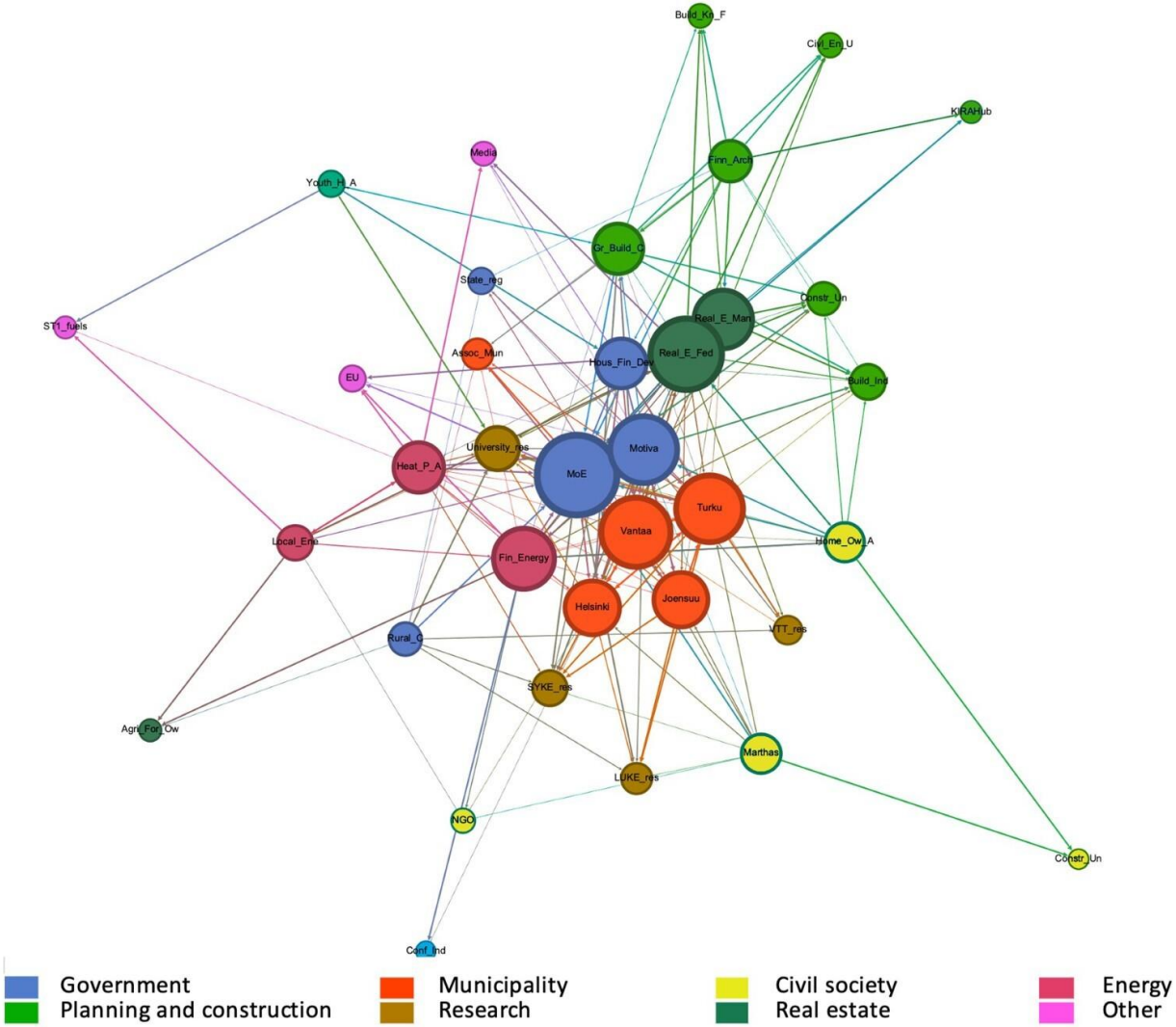
Quelques exemples de réseaux



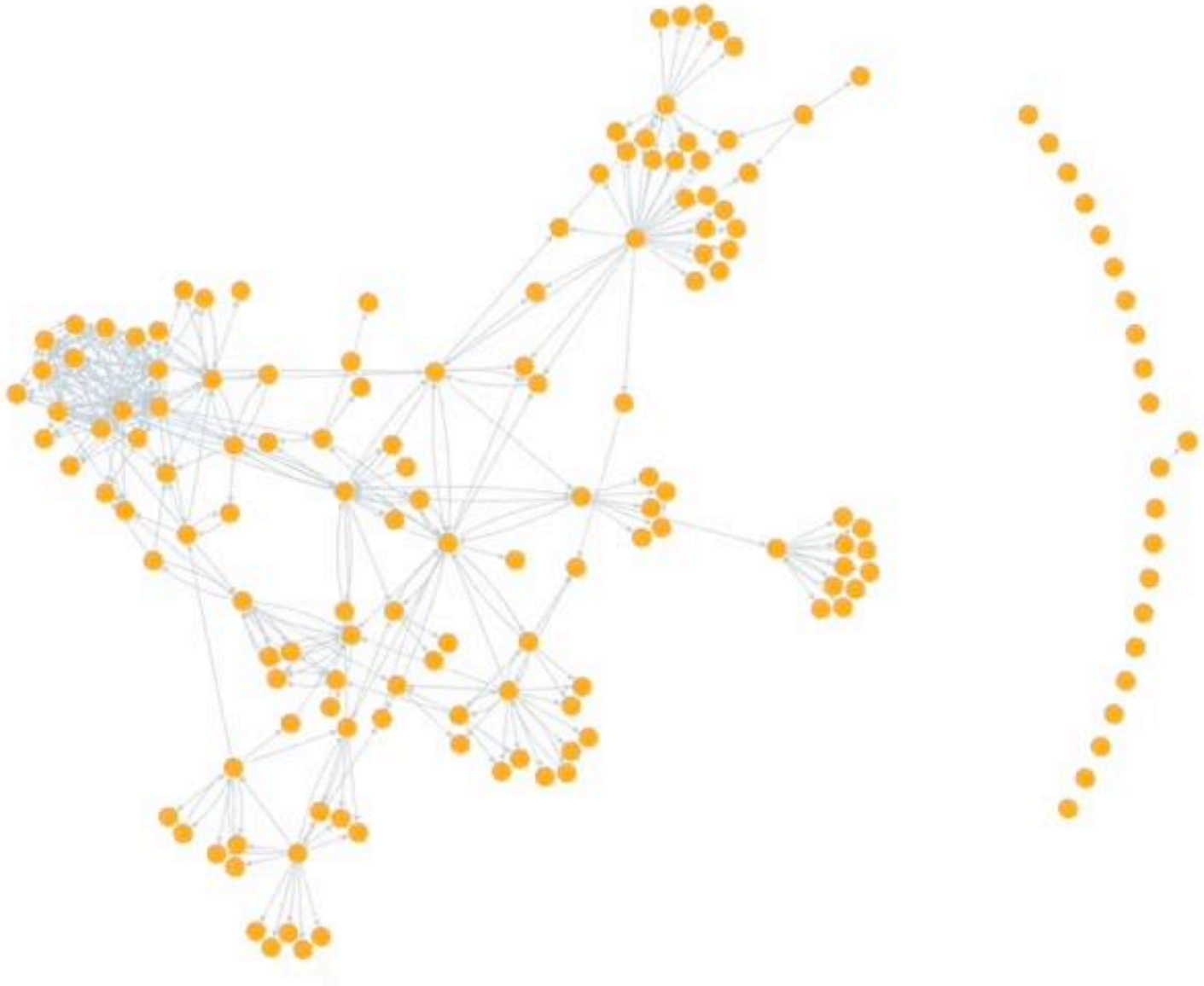
Quelques exemples de réseaux



Quelques exemples de réseaux

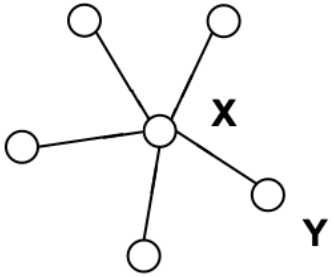


Quelques exemples de réseaux

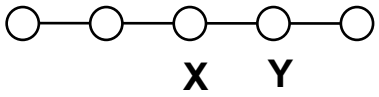


Différentes notions de centralité

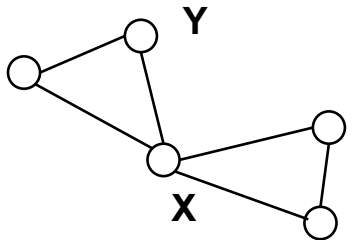
Dans chacun des exemples ci-dessous, X est plus « central » que Y selon la mesure de centralité associée.



Degré
Degree



Proximité
Closeness



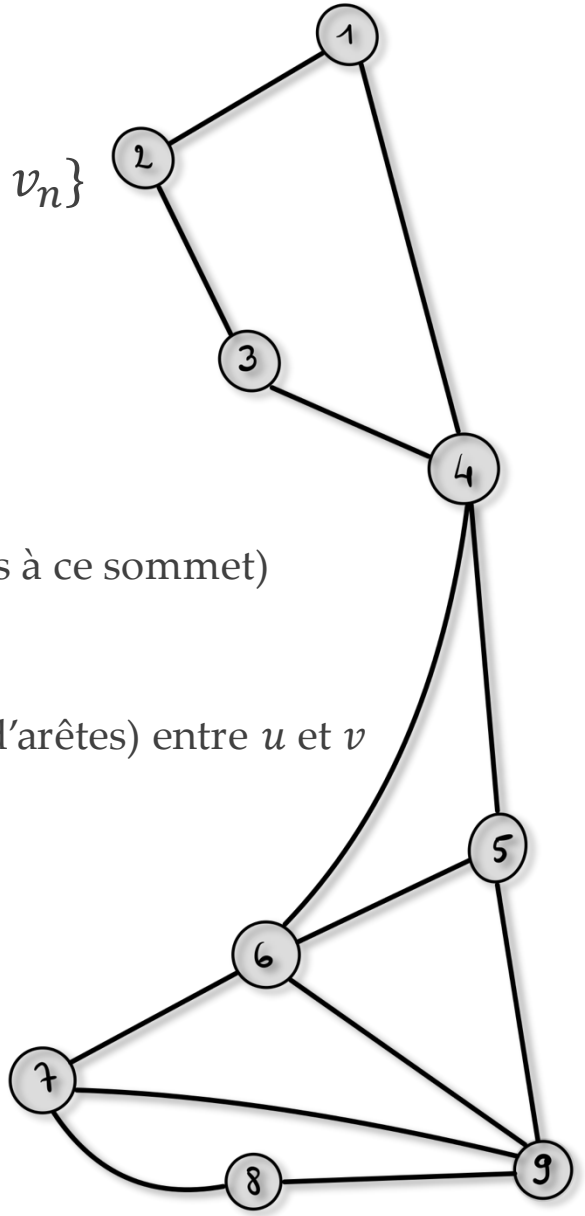
Intermédiarité
Betweenness

Un graphe (un réseau), c'est quoi ? (rappel)

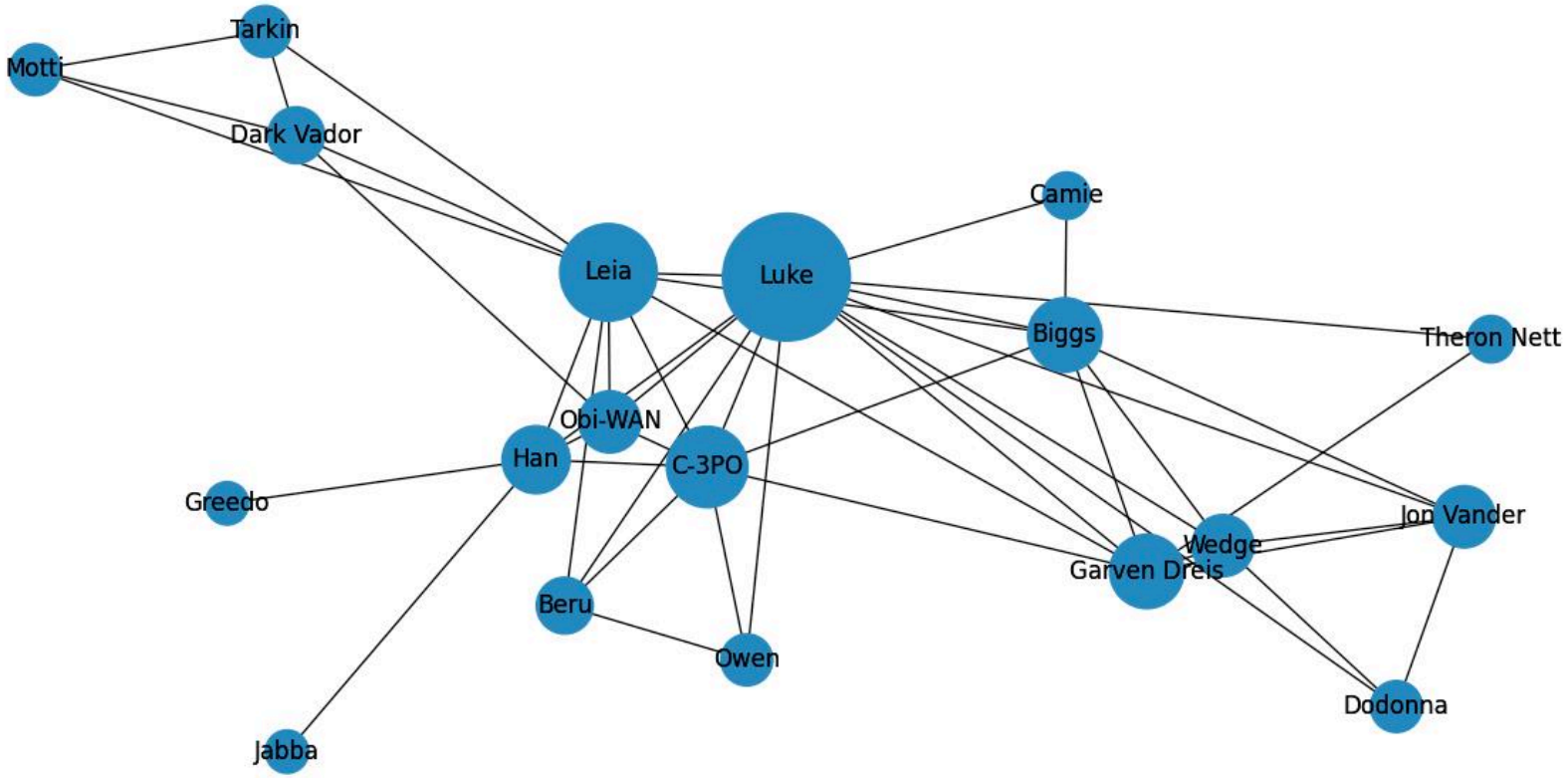
- Un graphe (ou un réseau) : $G = (V, E)$
- Sommets (ou nœuds ou acteurs) : $V = \{v_1, v_2, \dots, v_n\}$
- Arêtes (ou liens) : $E \subseteq V \times V$

- Notations habituelles :
  - n : Ordre du graphe (nombre de sommets)
  - m : Taille du graphe (nombre d'arêtes)
  - $d(v)$: Degré d'un sommet v (nombre d'arêtes incidentes à ce sommet)
  - $\delta(G)$: Degré minimum de G
  - $\Delta(G)$: Degré maximum de G
  - dist(u, v) : longueur du plus court chemin (en nombre d'arêtes) entre u et v

- Algorithmes classiques :
  - Kruskal
  - Parcours en largeur / profondeur
  - Dijkstra



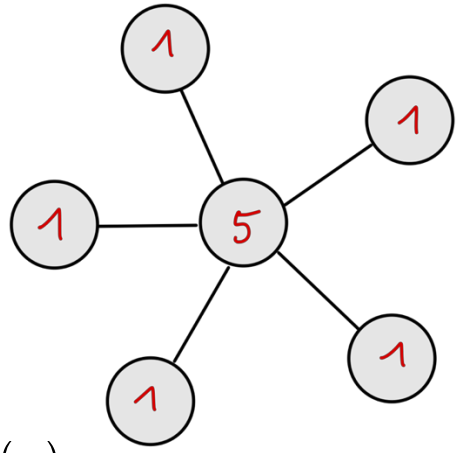
Un premier exemple : centralité de degré



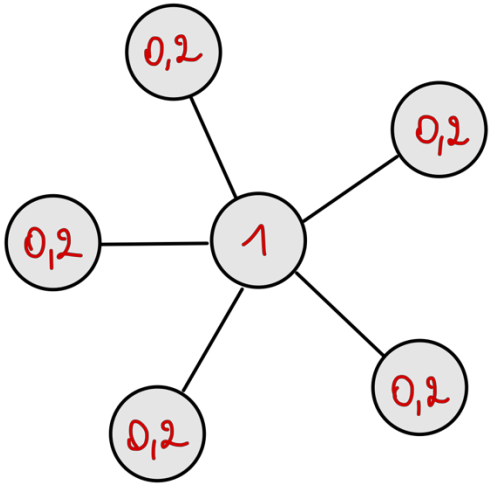
The star wars (1977) : Personnages présents dans une même scène

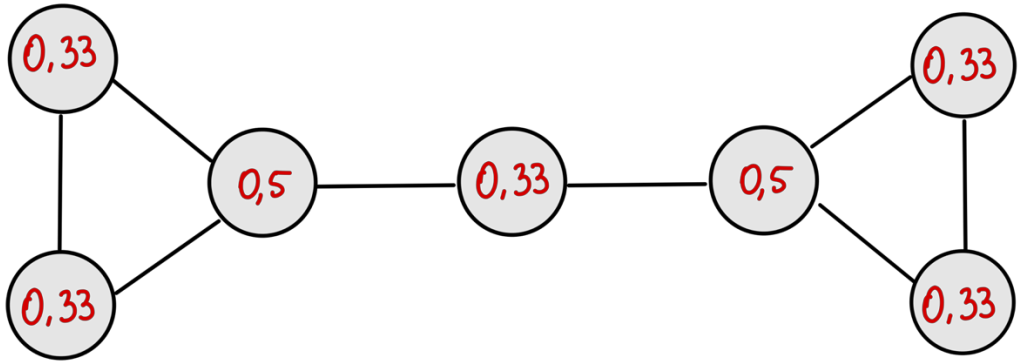
Centralité de degré : quantification et normalisation

- Quantification de la centralité : exemple dans le cas non orienté. **Notée :** $C_d(v)$
- On ne s'intéresse pas ici aux connexions de nos amis (i.e. les amis de nos amis).

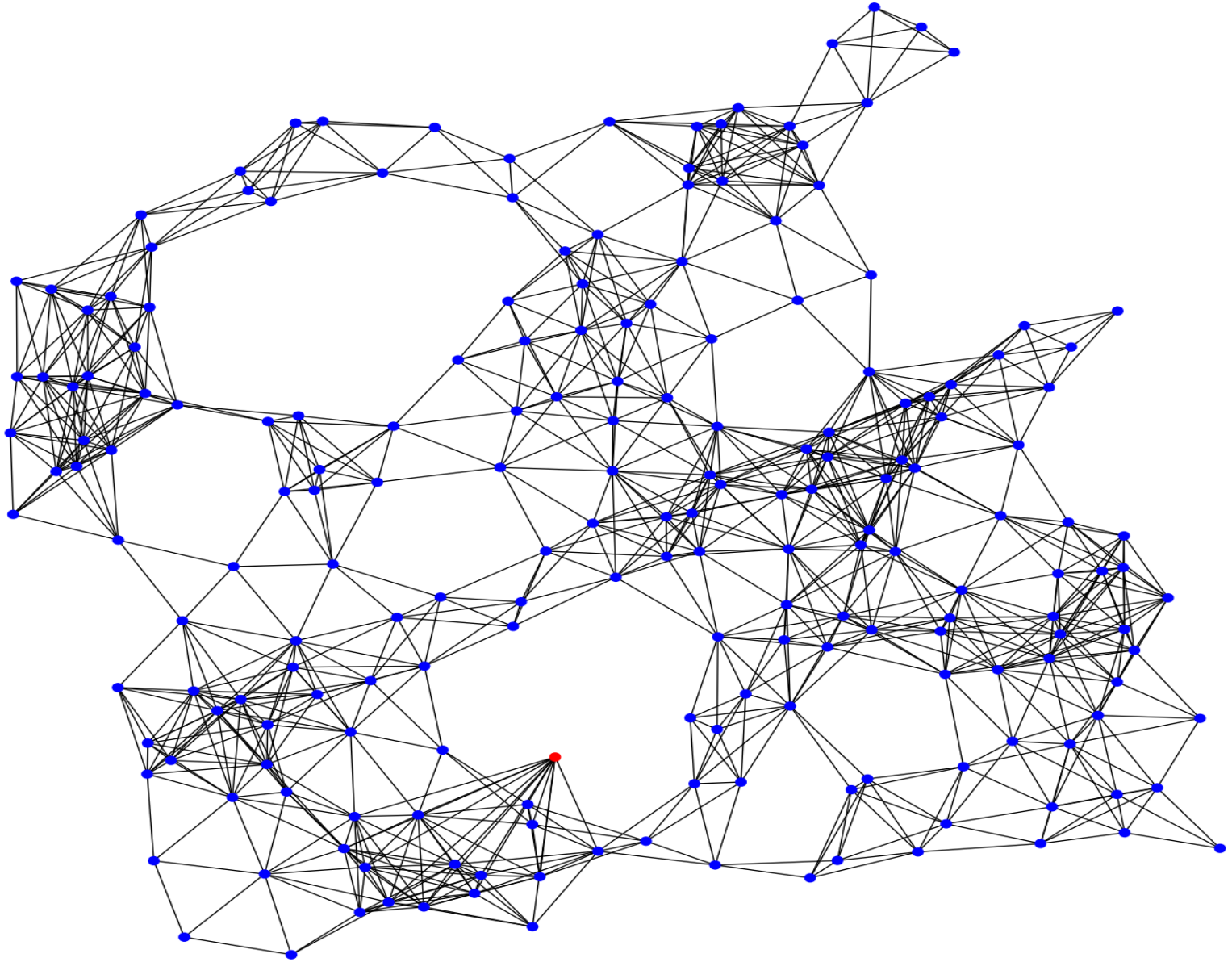


- Les valeurs sont parfois normalisées : $\overline{C_d(v)} = \dfrac{C_d(v)}{n-1}$





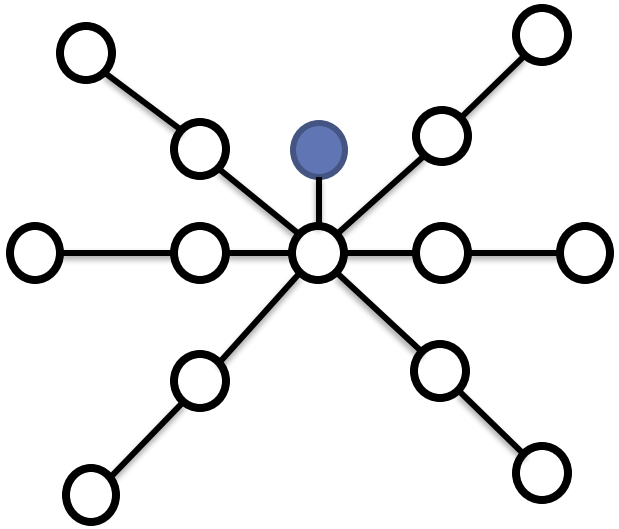
La centralité de degré est-elle suffisante ?



La centralité de Proximité
Closeness (Bavelas, 1950)

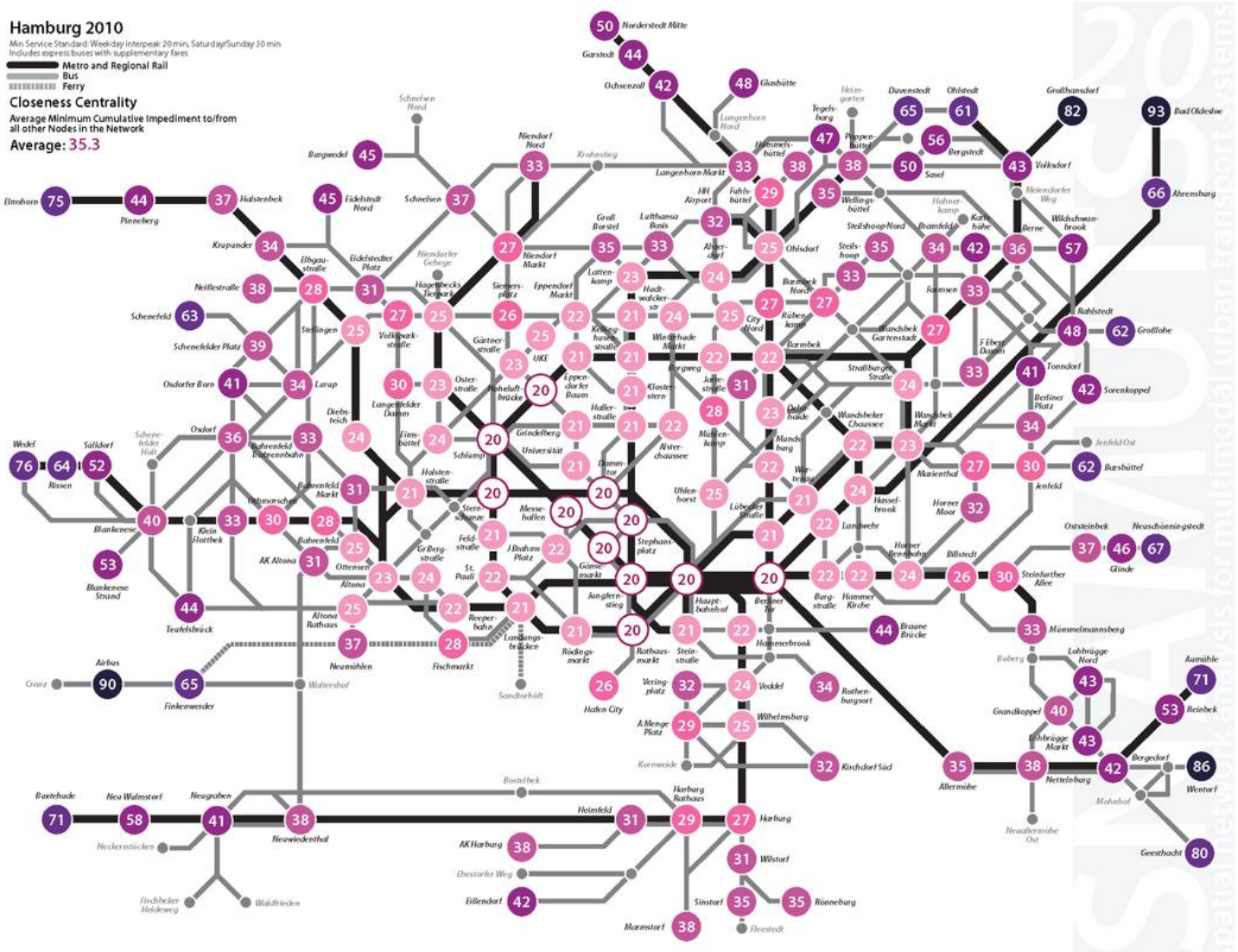
- Degré : caractérise le nombre d'amis directs.

- Si l'important est d'être « au milieu », **non loin** des autres nœuds :
  - Accès facilité (courte distance) à une grande partie du réseau
  - Diffusion rapide d'information



La centralité de Proximité

Closeness (Bavelas, 1950)



La centralité de Proximité
Closeness (Bavelas, 1950)

- La **proximité d'un nœud x** : inverse de la somme des longueurs des plus courts chemins entre x et tous les autres nœuds.

$$C_c(x) = \frac{1}{\sum_{y \neq x} \text{dist}(y, x)}$$

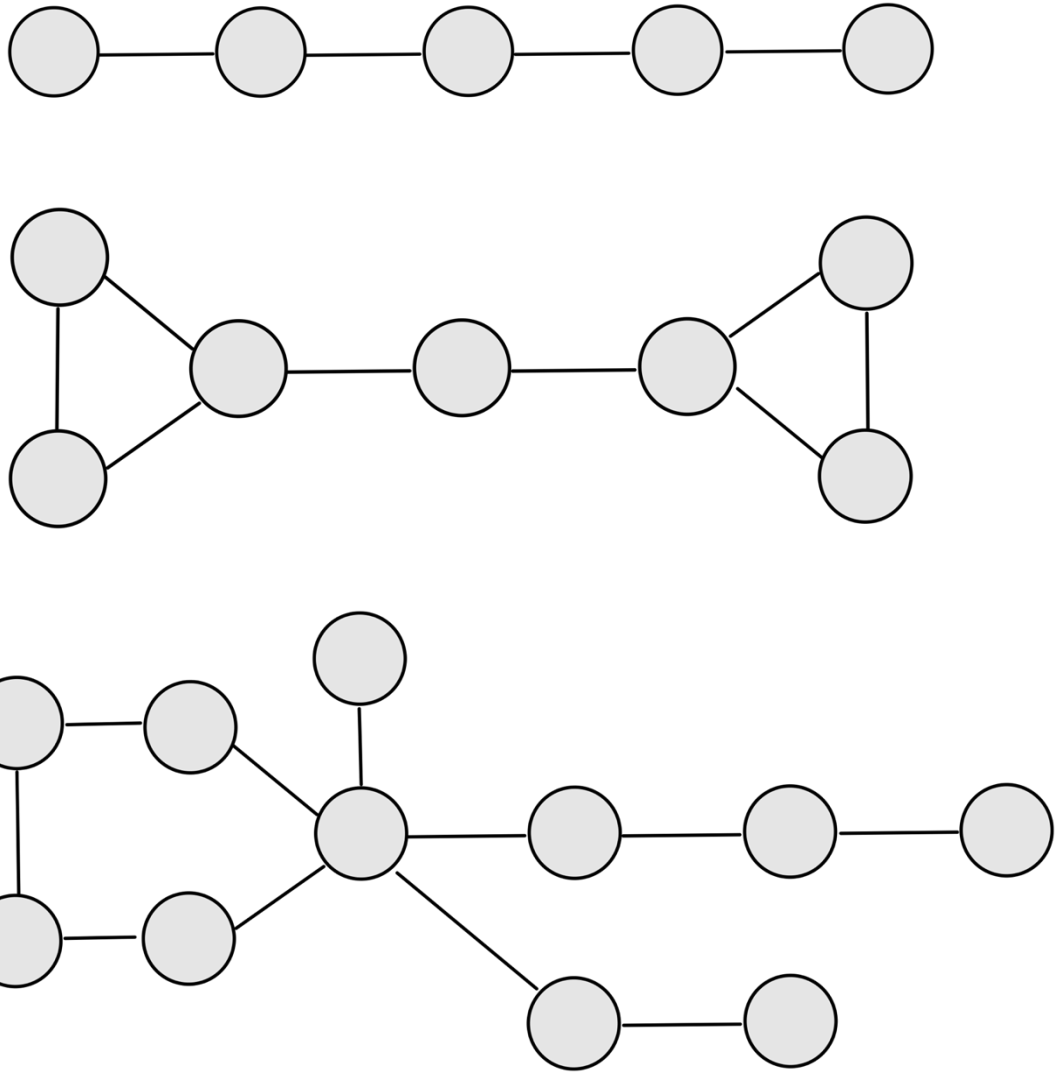
- En général, on normalise la centralité de proximité : $\overline{C_c} = (n-1)C_c$
(permet la comparaison de sommets au sein de graphes de différentes tailles)

- Dans le cas où le graphe n'est pas connexe :

$$C_c(x) = \sum_{y \neq x} \frac{1}{\text{dist}(y, x)} \quad \text{avec la convention} \quad \frac{1}{\infty} = 0$$

La centralité de Proximité

Closeness (Bavelas, 1950)



Centralité de proximité : Algorithmique

- Le calcul de la centralité de proximité de tous les sommets d'un graphe requiert le calcul des plus courts chemins entre toutes les paires de sommets d'un graphe.

- Les calculs peuvent être réalisés en appliquant une version légèrement modifiée de
  - l'algorithme PARCOURS-EN-LARGEUR qui prend un temps O(n(n+m)) dans le cas de graphes non pondérés ;
  - l'algorithme DIJKSTRA qui prend $O(n^3)$ dans le cas de graphes pondérés ;
  - l'algorithme DIJKSTRA qui prend O(n((n+m) log n)) dans le cas de graphes pondérés peu denses.

Centralité de proximité : Algorithmique

Algorithme : Centralité de Proximité (non pondéré)

Données : Un graphe $G = (V, E)$ non pondéré donné par liste de voisins.
Résultat : Une fonction : $C_C : V \to \mathbb{R}$ (la centralité de proximité de chaque sommet).

```
1  début
2      n ⟵ 0 ; // nombre de sommets
3      pour tous les v ∈ V faire
4          C_c(v) ⟵ 0 ;
5          n ⟵ n + 1
6      pour tous les r ∈ V faire
7          pour tous les v ∈ V faire
8              dv(v) ⟵ Faux ; // sommets déjà vus
9          dv(r) ⟵ Vrai ; ℓ(r) ⟵ 0 ; // la racine
10         Enfiler r dans AT ; // sommets à traiter, AT gérée comme une file
11         t ⟵ 2 ; // le temps
12         tant que AT ≠ ∅ faire
13             Prendre v le premier sommet de AT l'enlever de AT ;
14             pour tous les x ∈ Vois(v) faire
15                 si Non dv(x) alors
16                     dv(x) ⟵ Vrai ; // on traite x pour la première fois
17                     Enfiler x dans AT (en dernière position) ;
18                     t ⟵ t + 1 ;
19                     ℓ(x) ⟵ ℓ(v) + 1 ;
20         pour tous les v ∈ V faire
21             C_c(r) ⟵ C_c(r) + l(v) ;
22         si C_c(r) ≠ 0 et n > 1 alors
23             C_c(r) ⟵ (n − 1)/C_c(r) ;
```

Centralité de proximité : Algorithmique

Algorithme : Centralité de Proximité (pondéré)

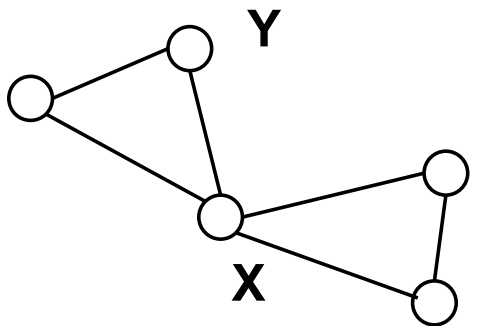
Données : Un graphe $G = (V, E)$ donné par liste de voisin avec ω une fonction de poids **positive** sur les arêtes.

Résultat : Une fonction : $C_C : V \to \mathbb{R}$ (la centralité de proximité de chaque sommet).

```
1  début
2      n ⟵ 0 ; // nombre de sommets
3      pour tous les v ∈ V faire
4          C_c(v) ⟵ 0 ;
5          n ⟵ n + 1
6      pour tous les r ∈ V faire
7          pour tous les v ∈ V faire
8              dist(v) ⟵ +∞ ;
9              traite(v) ⟵ 0 ; // pour marquer les sommets traités
10         dist(r) ⟵ 0 ; // la racine
11         tant que il existe x avec traite(x) = 0 faire
12             Choisir un tel x avec dist(x) minimum ;
13             traite(x) ⟵ 1 ;
14             pour tous les y ∈ Vois(x) faire
15                 si traite(y) = 0 et dist(y) > dist(x) + ω(xy) alors
16                     dist(y) ⟵ dist(x) + ω(xy) ; // x est un raccourci pour atteindre y
17         pour tous les v ∈ V faire
18             C_c(r) ⟵ C_c(r) + dist(v) ;
19         si C_c(r) ≠ 0 et n > 1 alors
20             C_c(r) ⟵ (n − 1)/C_c(r) ;
```

La centralité d'intermédiarité *Betweenness* (Freeman, 1977)

- X joue le rôle d'intermédiaire entre les 2 parties du réseau, se trouve à la frontière de plusieurs communautés.

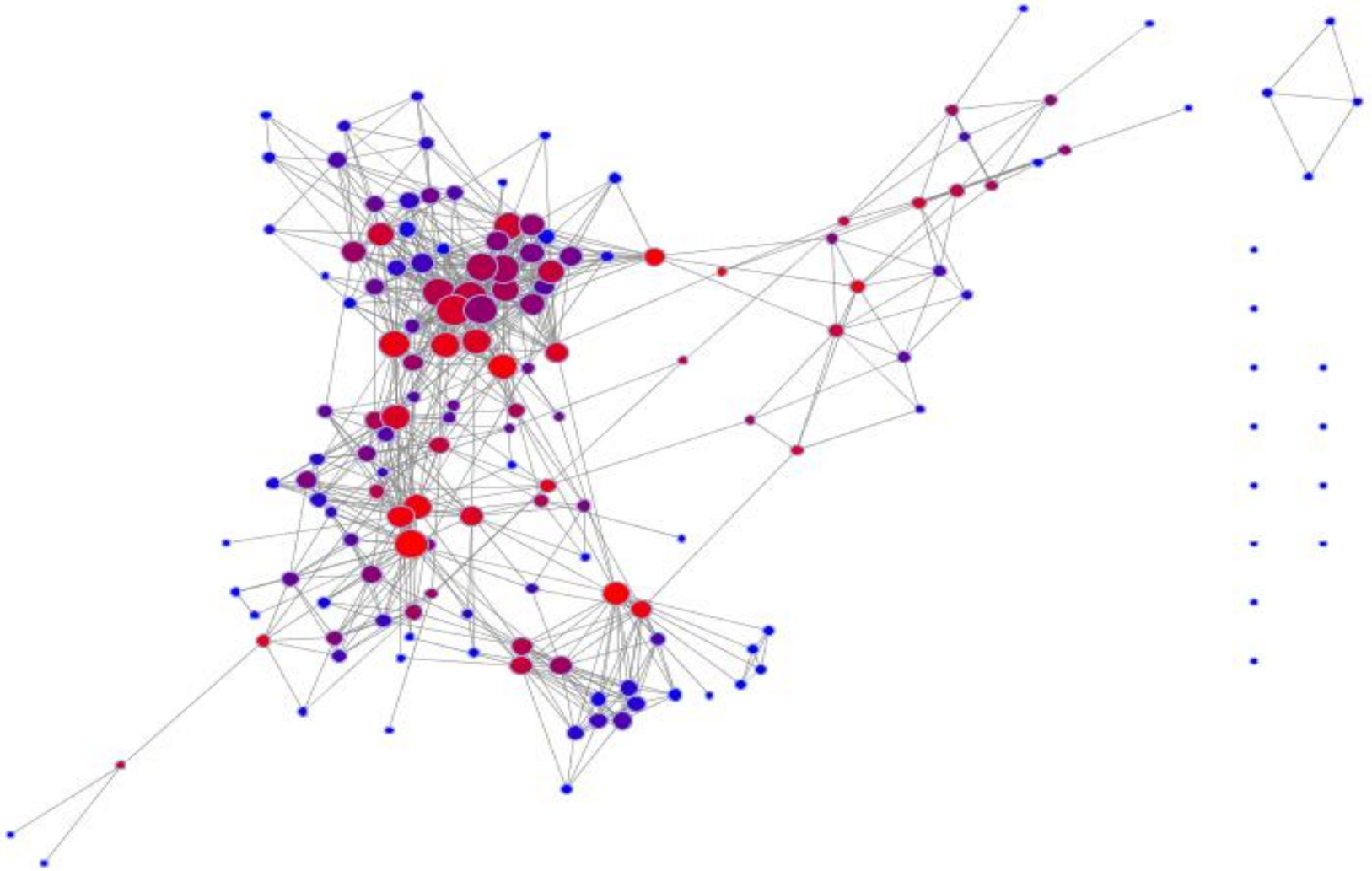


- Étude sociologique sur l'avantage d'occuper cette position
  Burt R. (2005), *Brokerage and closure. An introduction to social capital*, Oxford University Press

  Ils génèrent plus d'idées créatives, leurs idées sont plus facilement acceptées, ils ont plus de crédit.

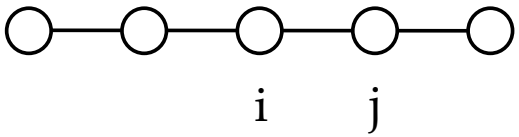
  Contraintes pour les sommets non-intermédiaires que les sommets intermédiaires n'ont pas ?

Centralité d'intermédiarité : Un exemple avec Instagram



La centralité d'intermédiarité *Betweenness* (Freeman, 1977)

- Combien de paires de nœuds devront passer via un nœud donné en un minimum d'étapes pour communiquer ?

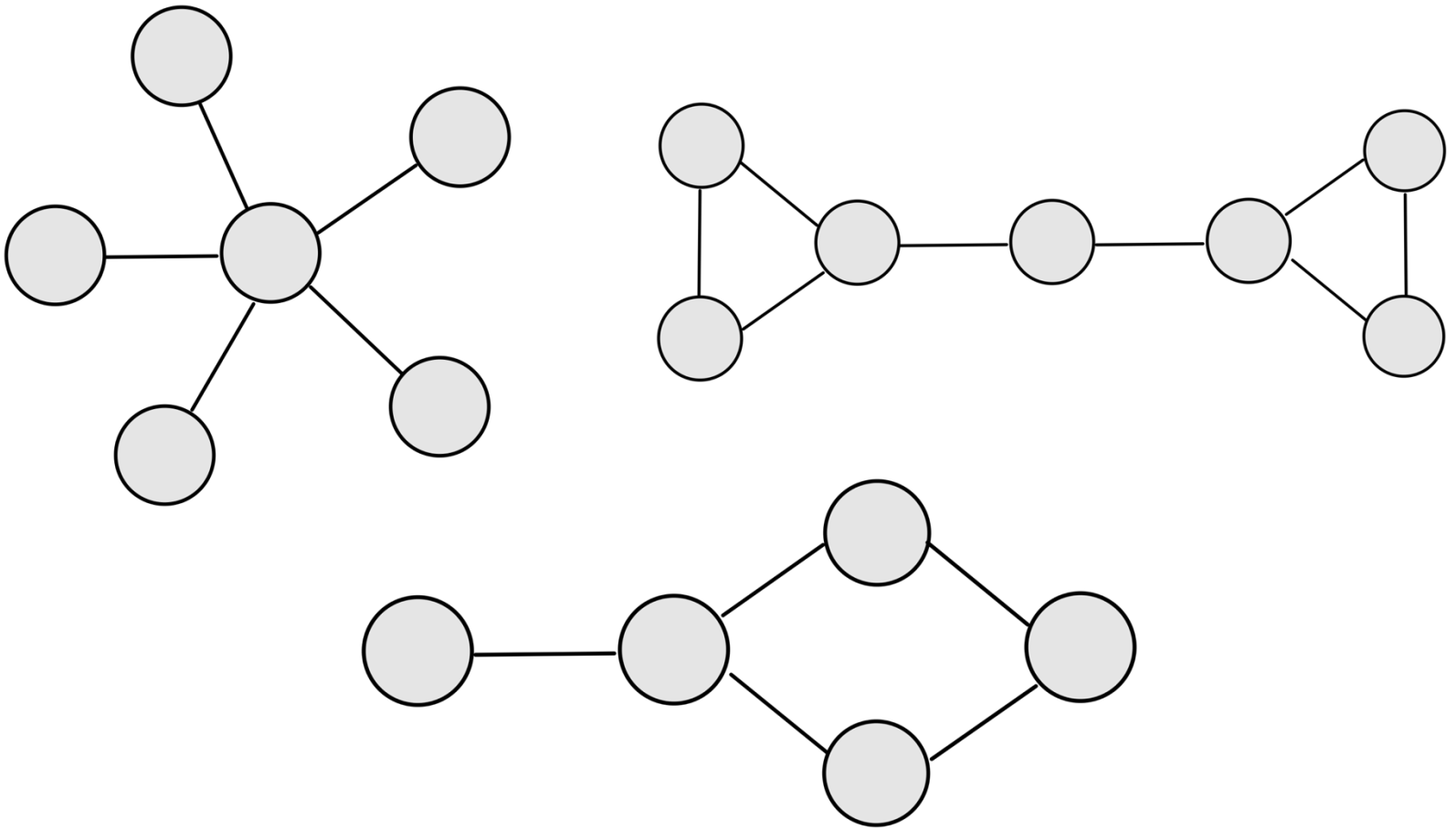


- $$C_b(v) = \sum_{u \neq v \neq w} \frac{\#\text{ plus courts chemins entre } u \text{ et } w \text{ passant par } v}{\#\text{ plus courts chemins entre } u \text{ et } w} = \sum_{u \neq v \neq w} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

- En général, on normalise la centralité d'intermédiarité :

$$\overline{C_b(v)} = \frac{2\, C_b(v)}{(n-1)(n-2)}$$

La centralité d'intermédiarité *Betweenness* (Freeman, 1977)



Comment calculer la centralité d'intermédiarité ?

- σ_{uw} = Nombre de plus courts chemins entre u et w
- $\sigma_{uw}(v)$ = Nombre de plus courts chemins entre u et w passant par v

- Pour chaque **paire de sommets** $\{\boldsymbol{u}, \boldsymbol{w}\}$, on doit trouver et stocker tous les plus courts chemins entre u et w.
- Pour chaque **paire de sommets** $\{\boldsymbol{u}, \boldsymbol{w}\}$ et **pour chaque sommet** $\boldsymbol{v}$, on doit compter tous les plus courts chemins entre u et w qui passent par v.

- Complexité élevée en temps et en espace.

Comment calculer la centralité d'intermédiarité ?

Ulrik Brandes (2001), « *A faster algorithm for betweenness centrality* » Journal of mathematical sociology, 25(2), 163-177.

- Les calculs peuvent être réalisés par l'algorithme de Brandes (2001)
  - qui prend un temps O(n(n+m)) dans le cas de graphes non pondérés (adaptation de PARCOURS-EN-LARGEUR) ;
  - qui prend $O(n^3)$ dans le cas de graphes pondérés (adaptation de DIJKSTRA) ;
  - qui prend O(n((n+m) log n)) dans le cas de graphes pondérés peu denses (adaptation de DIJKSTRA).

Brandes (2001) - The Journal of Mathematical Sociology

Journal of Mathematical Sociology
2001, Vol. 25(2), pp. 163–177
Reprints available directly from the publisher
Photocopying permitted by license only

© 2001 OPA (Overseas Publishers Association) N.V.
Published by license under
the Harwood Academic Publishers imprint,
part of Gordon and Breach Publishing,
a member of the Taylor & Francis Group.

Downloaded by [University of Guelph] at 10:36 01 July 2012

A FASTER ALGORITHM FOR BETWEENNESS CENTRALITY*

ULRIK BRANDES†

University of Konstanz, Department of Computer & Information Science, Box D 188, 78457 Konstanz, Germany

(Received September 25, 2000; In final form December 15, 2000)

The betweenness centrality index is essential in the analysis of social networks, but costly to compute. Currently, the fastest known algorithms require $\Theta(n^3)$ time and $\Theta(n^2)$ space, where n is the number of actors in the network.

Motivated by the fast-growing need to compute centrality indices on large, yet very sparse, networks, new algorithms for betweenness are introduced in this paper. They require $\mathcal{O}(n+m)$ space and run in $\mathcal{O}(nm)$ and $\mathcal{O}(nm+n^2 \log n)$ time on unweighted and weighted networks, respectively, where m is the number of links. Experimental evidence is provided that this substantially increases the range of networks for which centrality analysis is feasible.

KEY WORDS: Social networks, betweenness centrality, algorithms.

1 INTRODUCTION

In social network analysis, graph-theoretic concepts are used to understand and explain social phenomena. A social network consists of a set of actors, who may be arbitrary entities like persons or organizations, and one or more types of relations between them. For a comprehensive overview of methods and applications see Wasserman and Faust (1994) or Scott (1991).

*Part of this research was done while with the Department of Computer Science at Brown University. I gratefully acknowledge financial support from the German Academic Exchange Service (DAAD, Hochschulsonderprogramm III).
† E-mail: ulrik.brandes@uni-konstanz.de.

Downloaded by [University of Guelph] at 10:36 01 July 2012

An essential tool for the analysis of social networks are centrality indices defined on the vertices of the graph (Bavelas, 1948; Sabidussi, 1966; Freeman, 1979). They are designed to rank the actors according to their position in the network and interpreted as the prominence of actors embedded in a social structure. Many centrality indices are based on shortest paths linking pairs of actors, measuring, e.g., the average distance from other actors, or the ratio of shortest paths an actor lies on. Many network-analytic studies rely at least in part on an evaluation of these indices.

With the increasing practicality of electronic data collection and, of course, the advent of the Web, there is a likewise increasing demand for the computation of centrality indices on networks with thousands of actors. Several notions of centrality originating from social network analysis are in use to determine the structural prominence of Web pages (Kleinberg, 1999; Brin *et al.*, 1998; Bharat and Henzinger, 1998). However, there is an $\Omega(n^3)$ bottleneck in existing implementations, due to the particularly important betweenness centrality index (Freeman, 1977; Anthonisse, 1971), which makes comparative centrality analyses of networks with more than a few 100 actors prohibitive. As a remedy, network analysts are now suggesting simpler indices, for instance based only on linkages between the neighbors of each actors (Everett *et al.*, 1999), to at least obtain rough approximations of betweenness centrality.

In this paper, we show that betweenness can be computed exactly even for fairly large networks. We introduce more efficient algorithms based on a new accumulation technique that integrates well with traversal algorithms solving the single-source shortest-paths problem, and thus exploiting the sparsity of typical instances. The range of networks for which betweenness centrality can be computed is thereby extended significantly. Moreover, it turns out that all standard centrality indices based on shortest paths can thus be evaluated simultaneously, further reducing both the time and space requirements of comparative analyses.

The centrality indices relevant here are defined in Section 2. In Section 3, we review methods computing all shortest paths between all pairs of actors in a network. A recursion formula for accumulating betweenness centrality is derived in Section 4, and its practical implications are validated by experiments on real and randomly generated data, as discussed in Section 5.

Brandes (2001) - The Journal of Mathematical Sociology

Downloaded by [University of Guelph] at 10:36 01 July 2012

2 CENTRALITY INDICES BASED ON SHORTEST PATHS

Social and other networks are conveniently described as a graph $G = (V, E)$, where the set V of vertices represents actors, and the set E of edges represents links between actors. We use n and m to denote the number of vertices and edges, respectively. For simplicity, we assume that all graphs are undirected and connected, though they may have loops or multiple edges. Note that our results generalize to directed graphs with only minor modification.

Let ω be a *weight function* on the edges. We assume that $\omega(e) > 0, e \in E$, for *weighted* graphs, and define $\omega(e) = 1, e \in E$, for *unweighted* graphs. Weights are used to measure, e.g., the strength of a link.

Define a *path* from $s \in V$ to $t \in V$ as an alternating sequence of vertices and edges, beginning with s and ending with t, such that each edge connects its preceding with its succeeding vertex. The *length* of a path is the sum of the weights of its edges. We use $d_G(s, t)$ to denote the *distance* between vertices s and t, i.e. the minimum length of any path connecting s and t in G. By definition, $d_G(s, s) = 0$ for every $s \in V$, and $d_G(s, t) = d_G(t, s)$ for $s, t \in V$. We assume familiarity with standard algorithms for shortest-paths problems (see, e.g., Cormen *et al.*, 1990).

Several measures capture variations on the notion of a vertex's importance in a graph. Let $\sigma_{st} = \sigma_{ts}$ denote the number of shortest paths from $s \in V$ to $t \in V$, where $\sigma_{ss} = 1$ by convention. Let $\sigma_{st}(v)$ denote the number of shortest paths from s to t that some $v \in V$ lies on. The following are standard measures of centrality:

$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(v, t)} \quad \textit{closeness centrality} \text{ (Sabidussi, 1966)}$$

$$C_G(v) = \frac{1}{\max_{t \in V} d_G(v, t)} \quad \textit{graph centrality} \text{ (Hage and Harary, 1995)}$$

$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v) \quad \textit{stress centrality} \text{ (Shimbel, 1953)}$$

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad \textit{betweenness centrality} \text{ (Freeman, 1977; Anthonisse, 1971)}$$

Downloaded by [University of Guelph] at 10:36 01 July 2012

High centrality scores thus indicate that a vertex can reach others on relatively short paths, or that a vertex lies on considerable fractions of shortest paths connecting others. For interpretability, i.e. to control for the size of the network, the above indices are usually normalized to lie between zero and one. Though their definitions extend naturally to directed or disconnected graphs, normalization then becomes a problem with some of the above measures. The inhomogeneity of a centrality index is used to define the *centralization* of a graph with respect to that index (Freeman, 1979). A theoretical foundation for centrality measures not based on shortest paths is given in Friedkin (1991). See Wasserman and Faust (1994) for further details and note that we tacitly generalized some of these definitions of centrality to weighted graphs.

The computationally rather involved betweenness centrality index is the one most frequently employed in social network analysis. However, the sheer size of many instances occurring in practice makes the evaluation of betweenness centrality prohibitive. In the following, we therefore focus on computing betweenness. As it turns out, the resulting algorithm can trivially be augmented to compute the other measures as well, at virtually no extra cost. First, recall the following crucial observation.

LEMMA 1 (Bellman criterion) *A vertex $v \in V$ lies on a shortest path between vertices $s, t \in V$, if and only if $d_G(s, t) = d_G(s, v) + d_G(v, t)$.*

Given pairwise distances and shortest paths counts, the *pair-dependency*[1] $\delta_{st}(v) = \sigma_{st}(v)/\sigma_{st}$ of a pair $s, t \in V$ on an intermediary $v \in V$, i.e. the ratio of shortest paths between s and t that v lies on, is given by

$$\sigma_{st}(v) = \begin{cases} 0 & \text{if } d_G(s, t) < d_G(s, v) + d_G(v, t). \\ \sigma_{sv} \cdot \sigma_{vt} & \text{otherwise} \end{cases}$$

To obtain the betweenness centrality index of a vertex v, we simply have to sum the pair-dependencies of all pairs on that vertex,

[1] Note that this definition differs from the one in Freeman (1980, p. 588), where pair-dependency is defined as the dependency of a single vertex on another one. Here, the latter is simply called dependency (defined in Section 4).

Brandes (2001) - The Journal of Mathematical Sociology

$$C_B(v) = \sum_{s \neq v \neq t \in V} \delta_{st}(v).$$

Therefore, betweenness centrality is traditionally determined in two steps:

(1) Compute the length and number of shortest paths between all pairs.
(2) Sum all pair-dependencies.

3 COUNTING THE NUMBER OF SHORTEST PATHS

In this section, we observe that the complexity of determining betweenness centrality is, in fact, dominated by the second step, i.e. the $\Theta(n^3)$ time summation and $\Theta(n^2)$ storage of pair-dependencies. This situation is remedied in the next section.

The two implementations most widely used to compute betweenness are UCINET (Analytic Technologies, Version V, 1999) and SNAPS.[2] Probably because of a reference to Harary *et al.* (1965) in Freeman (1979), the latter appears to make use of the following lemma. Recall that the *adjacency matrix* of a graph is the $n \times n$-matrix $A = (a_{uv})_{u,v \in V}$ with $a_{uv} = 1$ if $\{u, v\} \in E$, and $a_{uv} = 0$ otherwise.

LEMMA 2 (Algebraic path counting) *Let $A^k = \left(a_{uv}^{(k)}\right)_{u,v \in V}$ be the k-th power of the adjacency matrix of an unweighted graph. Then $a_{uv}^{(k)}$ equals the number of paths from u to v of length exactly k.*

Since $\mathcal{O}(n^2)$ pair-dependencies need to be summed for each vertex, the overall running time of the implementation is dominated by the time spend on matrix multiplications.

Clearly, algebraic path counting computes more information than needed. Instead of the number of paths of length shorter than the diameter of the network (the maximum distance of any pair of vertices), we are only interested in the number of shortest paths between each pair of vertices.

[2] A collection of routines for GAUSS (Aptech Systems, Inc.) by Noah Friedkin of University of California, Santa Barbara.

Downloaded by [University of Guelph] at 10:36 01 July 2012

Some superfluous work is avoided in a suitably defined instance, called *geodetic semiring* (Batagelj, 1994), of the closed semiring generalization for shortest paths problems (Aho *et al.*, 1974). It yields an $\Theta(n^3)$ algorithm for betweenness by augmenting the Floyd/Warshall algorithm for the all-pairs shortest-paths problem with path counting.

To exploit the sparsity of typical instances, we count shortest paths using traversal algorithms. Both breadth-first search (BFS) for unweighted and Dijkstra's algorithm for weighted graphs start with a specified source $s \in V$ and, at each step, add a closest vertex to the set of already discovered vertices in order to find shortest paths from the source to all other vertices. In that process, they naturally discover all shortest paths from the source. Define the set of *predecessors* of a vertex v on shortest paths from s as

$$P_s(v) = \left\{u \in V : \{u, v\} \in E, d_G(s, v) = d_G(s, u) + \omega(u, v)\right\}.$$

LEMMA 3 (Combinatorial shortest-path counting) *For $s \neq v \in V$*

$$\sigma_{sv} = \sum_{u \in P_s(v)} \sigma_{su}.$$

Proof Since all edge weights are positive, the last edge of any shortest path from s to v is an edge $\{u, v\} \in E$ such that $d_G(s, u) < d_G(s, v)$. Clearly, the number of shortest paths from s to v ending with this edge equals the number of shortest paths from s to u. The equality now follows from Lemma 1. □

Both Dijkstra's algorithm and BFS are thus easily augmented to count the number of shortest paths according to this lemma. BFS takes time $\mathcal{O}(m)$, and Dijkstra's algorithm runs in time $\mathcal{O}(m + n \log n)$, if the priority queue is implemented with a Fibonacci heap (Fredman and Tarjan, 1987).

COROLLARY 4 *Given a source $s \in V$, both the length and number of all shortest paths to other vertices can be determined in time $\mathcal{O}(m + n \log n)$ for weighted, and in time $\mathcal{O}(m)$ for unweighted graphs.*

Consequently, σ_{st}, $s, t \in V$, can be computed in time $\mathcal{O}(nm)$ for unweighted and in time $\mathcal{O}(nm + n^2 \log n)$ for weighted graphs.

Downloaded by [University of Guelph] at 10:36 01 July 2012

Brandes (2001) - The Journal of Mathematical Sociology

Corollary 4 implies that running time is dominated by the $\Theta(n^3)$ time it takes to sum pair-dependencies. Apparently, this is the approach implemented in UCINET. Clearly, the $\Theta(n^2)$ space bound stems from the need to store the distance matrix and quantities σ_{st}, $s, t \in V$. In the next section we show how to reduce these complexities substantially by accumulating partial sums of pair-dependencies.

4 ACCUMULATION OF PAIR-DEPENDENCIES

To eliminate the need for explicit summation of all pair-dependencies, we introduce first the notion of the *dependency* of a vertex $s \in V$ on a single vertex $v \in V$, defined as

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v).$$

The crucial observation is that these partial sums obey a recursive relation. In the following special case, this relation is particularly easy to recognize.

LEMMA 5 *If there is exactly one shortest path from $s \in V$ to each $t \in V$, the dependency of s on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w : v \in P_s(w)} \Big(1 + \delta_{s\bullet}(w)\Big).$$

Proof The assumption implies that the vertices and edges of all shortest paths from s form a tree. Therefore, v lies on either all or none of the paths between s and some $t \in V$, i.e. $\delta_{st}(v)$ equals either 1 or 0. Moreover, v lies on all shortest paths to those vertices for which it is a predecessor, and on all shortest paths that these lie on (see also Figure 1). □

In the general case, a very similar relation holds.

THEOREM 6 *The dependency of $s \in V$ on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w : v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \Big(1 + \delta_{s\bullet}(w)\Big).$$

Downloaded by [University of Guelph] at 10:36 01 July 2012

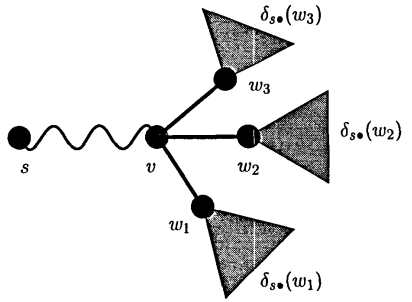


FIGURE 1 With the assumption of Lemma 5, a vertex lies on all shortest paths to its successors in the tree of shortest paths from the source.

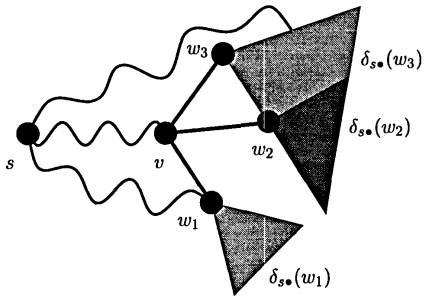


FIGURE 2 In the general case of Theorem 6, fractions of the dependencies on successors are propagated up along the edges of the directed acyclic graph of shortest paths from the source.

Proof Recall that $\delta_{st}(v) > 0$ only for those $t \in V \setminus \{s\}$ for which v lies on at least one shortest path from s to t, and notice that on any such path there is exactly one edge $\{v, w\}$ with $v \in P_s(w)$. This slightly more complicated situation is illustrated in Figure 2.

Downloaded by [University of Guelph] at 10:36 01 July 2012

Brandes (2001) - The Journal of Mathematical Sociology

We extend pair-dependency to include an edge $e \in E$ by defining $\delta_{st}(v, e) = \sigma_{st}(v, e)/\sigma_{st}$ where $\sigma_{st}(v, e)$ is the number of shortest paths from s to t that contain both v and e. Then,

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v) = \sum_{t \in V} \sum_{w : v \in P_s(w)} \delta_{st}\Big(v, \{v, w\}\Big) = \sum_{w : v \in P_s(w)} \sum_{t \in V} \delta_{st}\Big(v, \{v, w\}\Big).$$

Let w be any vertex with $v \in P_s(w)$. Of the σ_{sw} shortest paths from s to w, σ_{sv} many first go from s to v and then use $\{v, w\}$. Consequently, $\sigma_{sv}/\sigma_{sw} \cdot \sigma_{st}(w)$ shortest paths from s to some $t \neq w$ contain v and $\{v, w\}$. It follows that the pair-dependency of s and t on v and $\{v, w\}$ is

$$\delta_{st}\Big(v, \{v, w\}\Big) = \begin{cases} \dfrac{\sigma_{sv}}{\sigma_{sw}} & \text{if } t = w \\ \dfrac{\sigma_{sv}}{\sigma_{sw}} \cdot \dfrac{\sigma_{st}(w)}{\sigma_{st}} & \text{if } t \neq w. \end{cases}$$

Inserting this into the above yields

$$\sum_{w : v \in P_s(w)} \sum_{t \in V} \delta_{st}\Big(v, \{v, w\}\Big) = \sum_{w : v \in P_s(w)} \left(\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t \in V \setminus \{w\}} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \frac{\sigma_{st}(w)}{\sigma_{st}} \right)$$
$$= \sum_{w : v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \Big(1 + \delta_{s\bullet}(w)\Big). \qquad \square$$

COROLLARY 7 *Given the directed acyclic graph of shortest paths from $s \in V$ in G, the dependencies of s on all other vertices can be computed in $\mathcal{O}(m)$ time and $\mathcal{O}(n + m)$ space.*

Proof Traverse the vertices in non-increasing order of their distance from s and accumulate dependencies by applying Theorem 6. We need to store a dependency per vertex, and lists of predecessors. There is at most one element per edge in any of these lists. $\square$

With this result, we can determine the betweenness centrality index by solving one single-source shortest-paths problem for each vertex. At the end of each iteration, the dependencies of the source on each other vertex are added to the centrality score of that vertex. For unweighted graphs, the algorithm can be implemented as described in

Downloaded by [University of Guelph] at 10:36 01 July 2012

Algorithm 1. Note that the centrality scores need to be divided by two if the graph is undirected, since all shortest paths are considered twice. The modifications necessary for weighted graphs are straightforward.

THEOREM 8 *Betweenness centrality can be computed in $\mathcal{O}(nm + n^2 \log n)$ time and $\mathcal{O}(n + m)$ space for weighted graphs. For unweighted graphs, running time reduces to $\mathcal{O}(nm)$.*

The other shortest-path based centrality measures defined in Section 2 are easily computed during the execution of single-source shortest-paths traversals. The same holds for a recently introduced index called *radiality* (Valente and Foreman, 1998)

$$C_R(v) = \frac{\sum_{t \in V} \Big(D(G) + 1 - d_G(v, t)\Big)}{(n - 1) \cdot D(G)},$$

where $D(G) = \max_{s,t \in V} d_G(s, t)$, and for other potential measures as well. This is a significant practical advantage, reducing the combined time and space spent on computing different measures that are to be compared.

Finally, we note that centrality computations for many shortest-path based indices can be sped up heuristically by first decomposing an undirected graph into its biconnected components.

5 PRACTICAL IMPLICATIONS

In this section, we evaluate the practical relevance of the asymptotic complexity improvement achieved by accumulating dependencies. We have implemented weighted and unweighted versions of our algorithm for directed and undirected graphs using the Library of Efficient Data Structures and Algorithms (LEDA, see Mehlhorn and Näher, 1999). Performance is compared with an implementation that uses the same code to determine the length and number of shortest paths between all pairs of vertices, but sums all pair-dependencies explicitly. Note that this is at most faster than implementations currently in use. The experiment was performed on a Sun Ultra 10 SparcStation with 440 MHz clock speed and 256 MBytes main memory.

Figure 3 shows running times for betweenness centrality on 180 random undirected unweighted graphs with 100–2000 vertices. For

Downloaded by [University of Guelph] at 10:36 01 July 2012

Brandes (2001) - The Journal of Mathematical Sociology

ALGORITHM 1
Betweenness Centrality in Unweighted Graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

Downloaded by [University of Guelph] at 10:36 01 July 2012

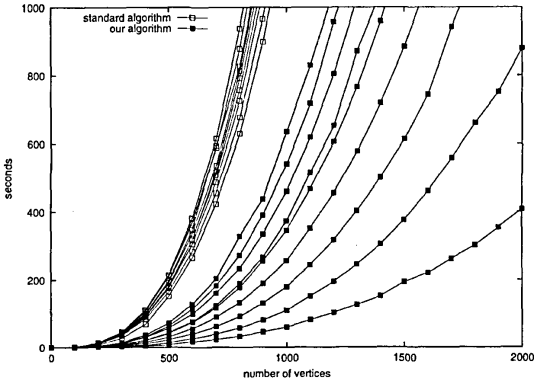


FIGURE 3 Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with 100–2000 vertices and densities ranging from 10–90%.

each number of vertices, there are nine graphs with 10–90% density (defined as the number of edges divided by $n(n-1)/2$). First notice that running times of the standard algorithm vary only slightly, indicating that most of the time is spent on summing pair-dependencies, rather than counting paths. Additional experiments confirmed that the overhead to determine the number of shortest paths in weighted graphs is negligible. As expected, accumulation according to Theorem 6 yields a significant speedup. This is true even for dense graphs, because the $\mathcal{O}(m)$ bound of Corollary 7 is, at least in general, overly pessimistic for the number of edges on shortest paths from some source.

Since large instances arising in social network analysis are typically rather sparse, with density well below 10%, the experiment clearly indicates that betweenness centrality can be computed for graphs of significantly larger size by accumulating dependencies instead of summing pair-dependencies.

The speed-up was also validated in practice, by analysis of an instance of 4259 intravenous drug users with 61,693 directed weighted

Downloaded by [University of Guelph] at 10:36 01 July 2012

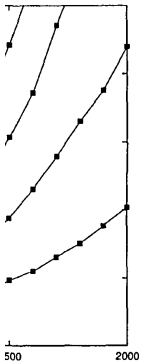
Brandes (2

ology

ALGORITHM 1
Betweenness Centrality in Unweighted Graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

Downloaded by [University of Guelph] at 10:36 01 July 2012



ılity index for random
ensities ranging from

ı 10–90% density
l)/2). First notice
ɔnly slightly, indi-
ɔair-dependencies,
ts confirmed that
paths in weighted
ɾding to Theorem
for dense graphs,
in general, overly
paths from some

ılysis are typically
ɛxperiment clearly
puted for graphs
ɗencies instead of

ɔy analysis of an
directed weighted

Brandes (2001) - The Journal of Mathematical Sociology

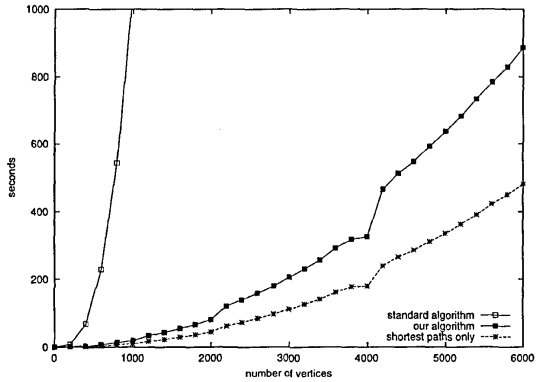


FIGURE 4 Seconds needed to the compute betweenness centrality index for random undirected, unweighted graphs with constant average degree 20. The funny jumps are attributed to LEDA internals.

links, originating from 197,216 unique contacts.[3] Not only because of running time, but also because of the memory required to store the distance and shortest-paths count matrices, betweenness centrality could not be evaluated for this network to date. The largest sub-network previously analyzed had 494 actors with 1774 links (taking 25 minutes on a 200 MHz Pentium Pro PC). Our implementation determined the betweenness centrality index of the whole network in 448 seconds, using less than 8 MBytes of memory.

Our algorithm has also been implemented in the publically available network analysis tool Pajek (Batagelj and Mrvar, 1998) and by researchers performing centrality analyses on networks of words extracted from electronic text. For typical instances, they obtained improvements from about 12 hours CPU time on an SGI Medusa

[3] Courtesy of Robert Foreman and Thomas Valente of the Epidemiology Data House at Johns Hopkins University. See Valente *et al.* (1998) for background on the data.

Downloaded by [University of Guelph] at 10:36 01 July 2012

workstation to less than five minutes on a Pentium III PC with 450 MHz.[4]

Note that the average sum of in- and outdegrees in this network is less than 29, corresponding to 0.3% density. (Clearly, density tends to zero when the average degree is fixed.) Recent experiments estimate an even lower average outdegree of 7.2 for Web pages (Kleinberg *et al.*, 1999). Figure 4 gives running times for betweenness index calculations on random graphs with a fixed average vertex degree of 20. These results imply that our implementation can compute, e.g., the betweenness centrality index for an extract of more than 10,000 Web pages in less than an hour on standard equipment.

[4] As reported by Steven Corman, Department of Communication, Arizona State University, April 2000.

REFERENCES

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974). *Data Structures and Algorithms*. Addison-Wesley.

Anthonisse, J. M. (1971). The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, Amsterdam.

Batagelj, V. (1994). Semirings for social network analysis. *Journal of Mathematical Sociology*, **19**(1): 53–68.

Batagelj, V. and Mrvar, A. (1998). Pajek – Program for large network analysis. *Connections*, **21**(2): 47–57. Project home page at http://vlado.fmf.uni-lj.si/pub/networks/pajek/.

Bavelas, A. (1948). A mathematical model for group structure. *Human Organizations*, **7**: 16–30.

Bharat, K. and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 104–111.

Brin, S., Motwani, R., Page, L. and Winograd, T. (1998). What can you do with a web in your pocket? *IEEE Bulletin of the Technical Committee on Data Engineering*, **21**(2): 37–47.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.

Everett, M. G., Borgatti, S. P. and Krackhardt, D. (1999). Ego-network betweenness. Paper presented at the *19th International Conference on Social Network Analysis (Sunbelt XIX)*, Charleston, South Carolina.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, **34**(3): 596–615.

Downloaded by [University of Guelph] at 10:36 01 July 2012

Brandes (2001) - The Journal of Mathematical Sociology

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, **40**: 35–41.
Freeman, L. C. (1979). Centrality in social networks: Conceptual clarification. *Social Networks*, **1**: 215–239.
Freeman, L. C. (1980). The gatekeeper, pair-dependency and structural centrality. *Quality and Quantity*, **14**: 585–592.
Friedkin, N. E. (1991). Theoretical foundations for centrality measures. *American Journal of Sociology*, **96**(6): 1478–1504.
Hage, P. and Harary, F. (1995). Eccentricity and centrality in networks. *Social Networks*, **17**: 57–63.
Harary, F., Norman, R. Z. and Cartwright, D. (1965). *Structural Models: An Introduction to the Theory of Directed Graphs*. John Wiley & Sons.
Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the Association for Computing Machinery*, **46**(5): 604–632.
Kleinberg, J. M., Kumar, R., Raghavan, P., Rajagopalan, S. and Tomkins, A. S. (1999). The Web as a graph: Measurements, models, and methods. In T. Asano, H. Imai, D. T. Lee, S. Nakano and Tokuyama, T. (Eds.), *Proceedings of the 5th International Conference on Computing and Combinatorics (COCOON '99)*, volume 1627 of *Lecture Notes in Computer Science* (pp. 1–17). Springer.
Mehlhorn, K. and Näher, S. (1999). *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press. Project home page at http://www.mpi-sb.mpg.de/LEDA/.
Sabidussi, G. (1966). The centrality index of a graph. *Psychometrika*, **31**: 581–603.
Scott, J. (1991). *Social Network Analysis: A Handbook*. Sage Publications.
Shimbel, A. (1953). Structural parameters of communication networks. *Bulletin of Mathematical Biophysics*, **15**: 501–507.
Valente, T. W. and Foreman, R. K. (1998). Integration and radiality: Measuring the extent of an individual's connectedness and reachability in a network. *Social Networks*, **20**(1): 89–105.
Valente, T. W., Foreman, R. K., Junge, B. and Vlahov, D. (1998). Satellite exchange in the Baltimore needle exchange program. *Public Health Reports*, **113**(S1): 90–96.
Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

Downloaded by [University of Guelph] at 10:36 01 July 2012

Algorithme de Brandes (2001)

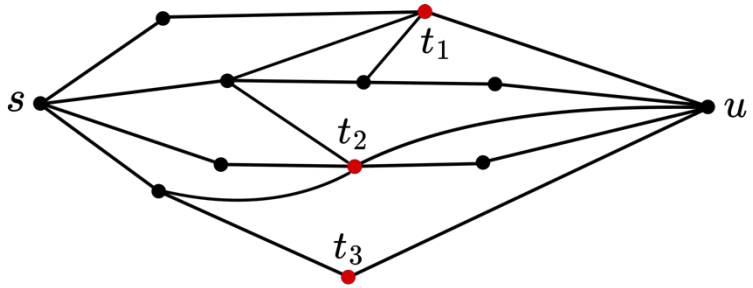
Algorithme : Centralité d'Intermédiarité (Brandes, 2001)
Données : Un graphe non pondéré $G = (V, E)$ donné par liste de voisins
Résultat : Une fonction : $C_B : V \to \mathbb{R}$ (la centralité d'intermédiarité de chaque sommet).

```
1  début
2      pour tous les v ∈ V faire
3          C_B(v) ⟵ 0 ; // la centralité d'intermédiarité.
4      pour tous les r ∈ V faire
5          AT ⟵ file vide ; // sommets à traiter
6          S ⟵ pile vide ; // pour obtenir les sommets par distance non-croissante
7          P ⟵ liste de files ;
8          pour tous les v ∈ V faire
9              ℓ(v) ⟵ −1 ; // la distance (niveau) à la racine r, −1 signifie non visité.
10             σ(v) ⟵ 0 ; // le nombre de PCC pour atteindre v depuis r.
11             P(v) ⟵ file vide ; // prédécesseurs de v sur les PCC de r à v.
12             δ(v) ⟵ 0 ; // centralité d'intermédiarité de v pour les PCC débutant en r
13         ℓ(r) ⟵ 0 ; // la racine
14         σ(r) ⟵ 1 ; // la racine
15         Enfiler r dans AT ;
16         tant que AT ≠ ∅ faire
17             Prendre v le premier sommet de AT, le défiler de AT, l'empiler dans S ;
18             pour tous les x ∈ Vois(v) faire
19                 si ℓ(x) = −1 alors
20                     Enfiler x dans AT ;
21                     ℓ(x) ⟵ ℓ(v) + 1 ;
22                 si ℓ(x) = ℓ(v) + 1 alors
23                     σ(x) ⟵ σ(x) + σ(v) ;
24                     si v ≠ r alors
25                         Enfiler v dans P(x) ;
26         tant que S ≠ ∅ faire
27             Prendre v le premier sommet de S, le dépiler de S ;
28             pour tous les p ∈ P(v) faire
29                 δ(p) ⟵ δ(p) + (σ(p)/σ(v)) · (1 + δ(v)) ;
30             si v ≠ r alors
31                 C_B(v) ⟵ C_B(v) + δ(v)
```

Comment calculer la centralité d'intermédiarité ?

- σ_{uw} = Nombre de plus courts chemins entre u et w
- $\sigma_{uw}(v)$ = Nombre de plus courts chemins entre u et w passant par v
- Conventions :
  - Si $u = w$, alors $\sigma_{uw} = 1$
  - Si $v \in \{u, w\}$, alors $\sigma_{uw}(v) = 0$

- Pour un sommet « racine » s : $\text{Pred}_s(u) = \{t : tu \in E \text{ et } \text{dist}(s,u) = \text{dist}(s,t)+1\}$



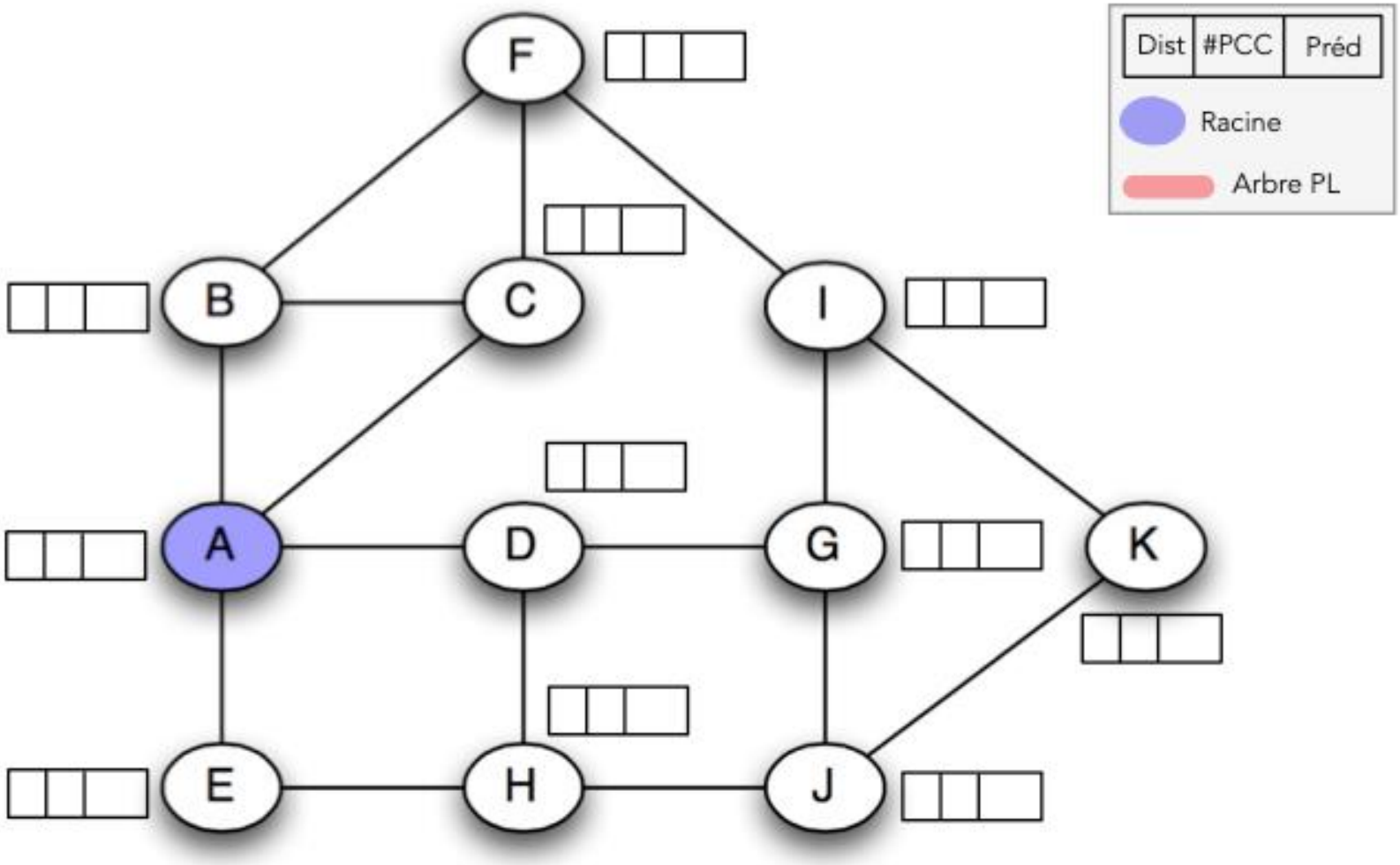
$\text{dist}(s,u) = 3$

$$\text{Pred}_s(u) = \{t_1, t_2, t_3\}$$

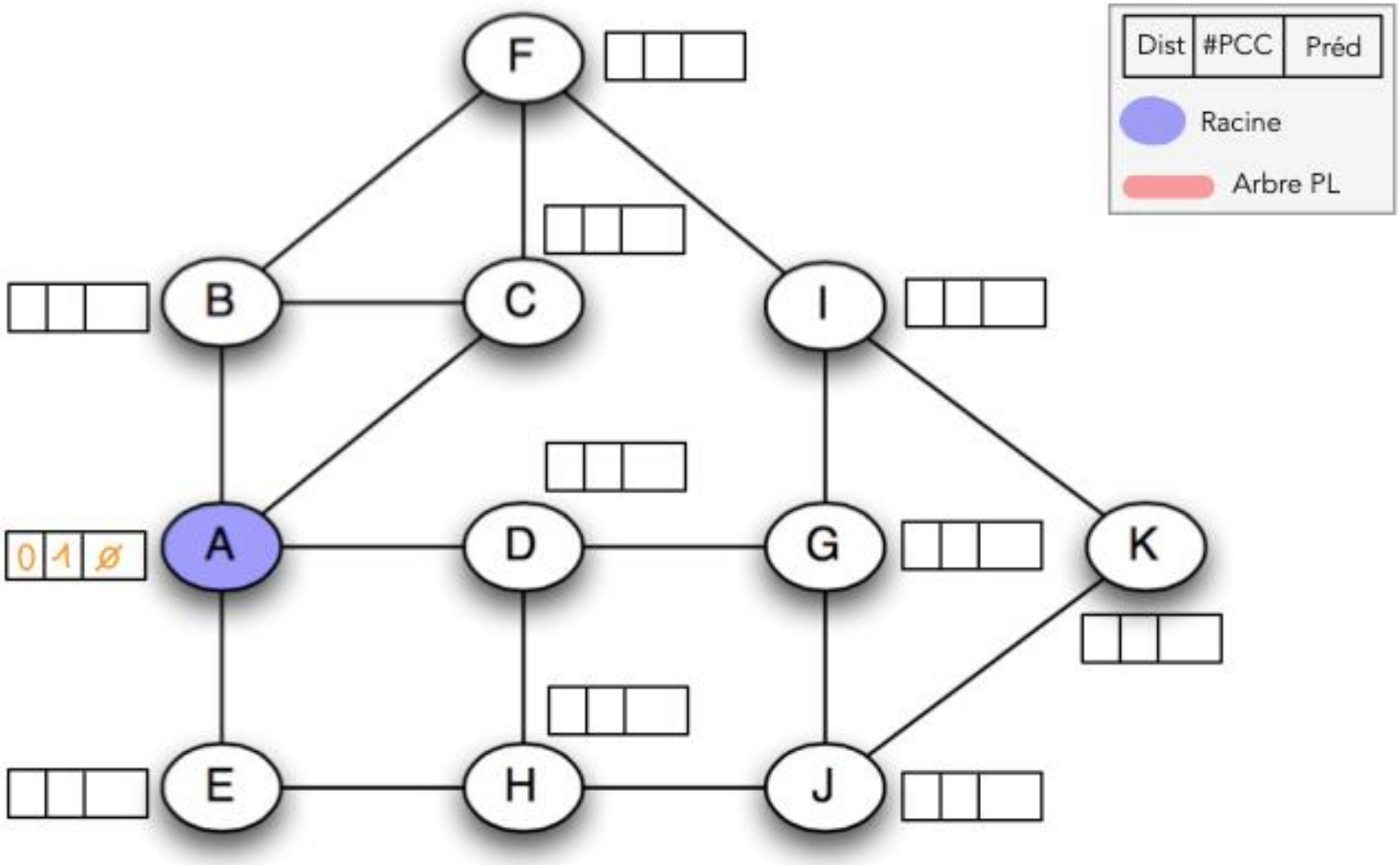
$$\left.\begin{array}{l} \sigma_{st_1} = 2 \\ \sigma_{st_2} = 3 \\ \sigma_{st_3} = 1 \end{array}\right\} \Rightarrow \sigma_{su} = 6$$

- On a donc : $\sigma_{su} = \sum_{t \in \text{Pred}(u)} \sigma_{st}$

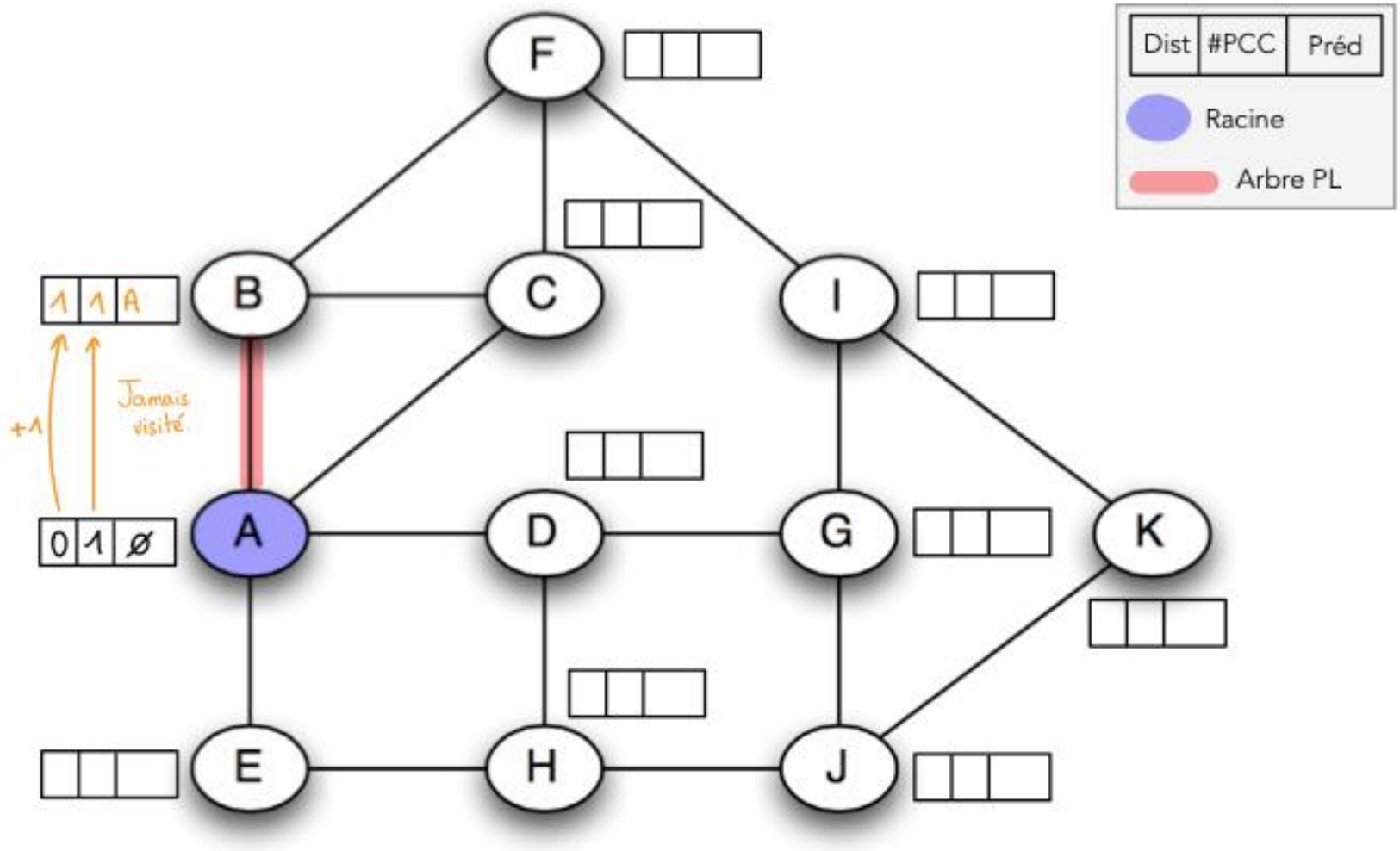
Un exemple



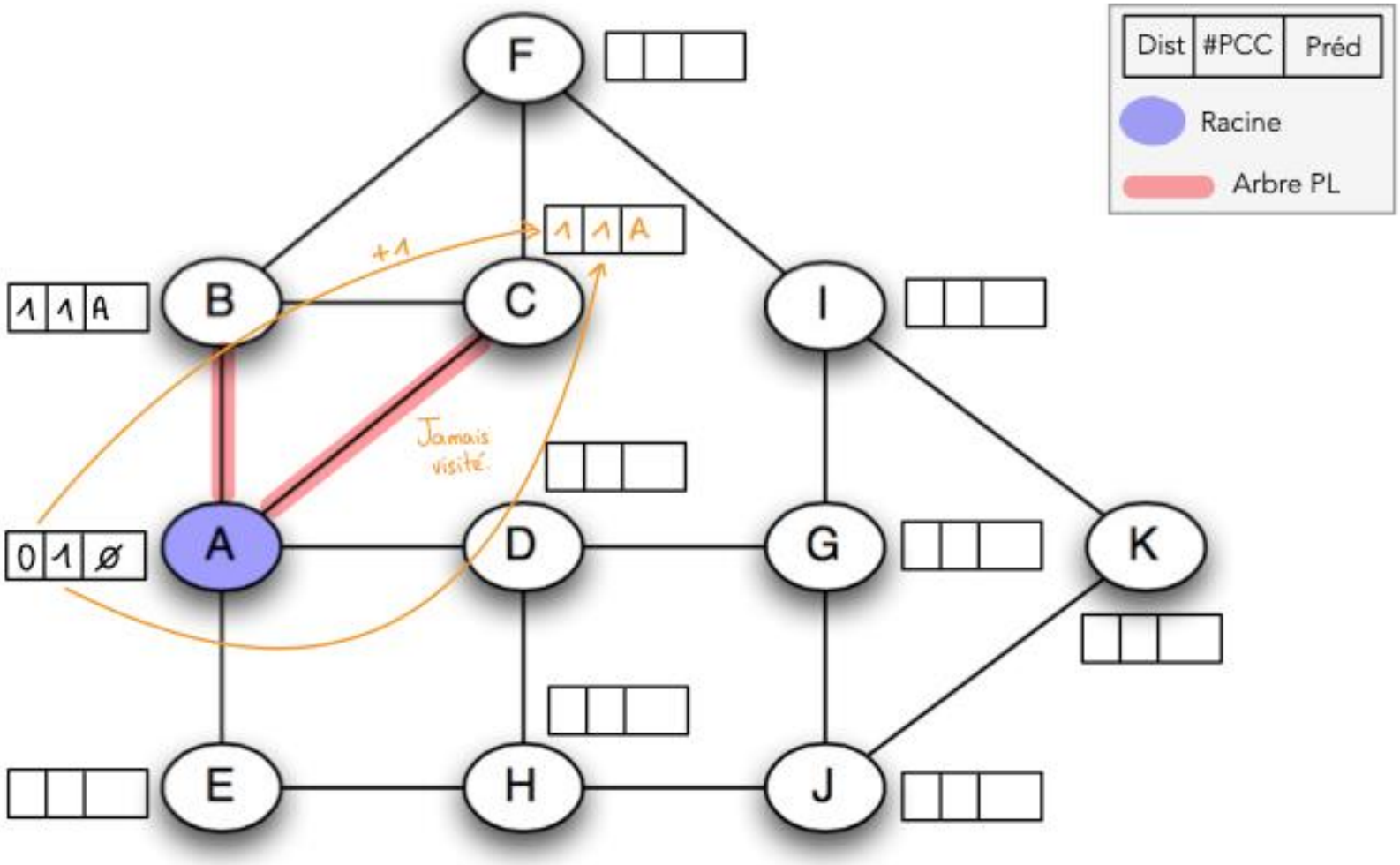
Un exemple



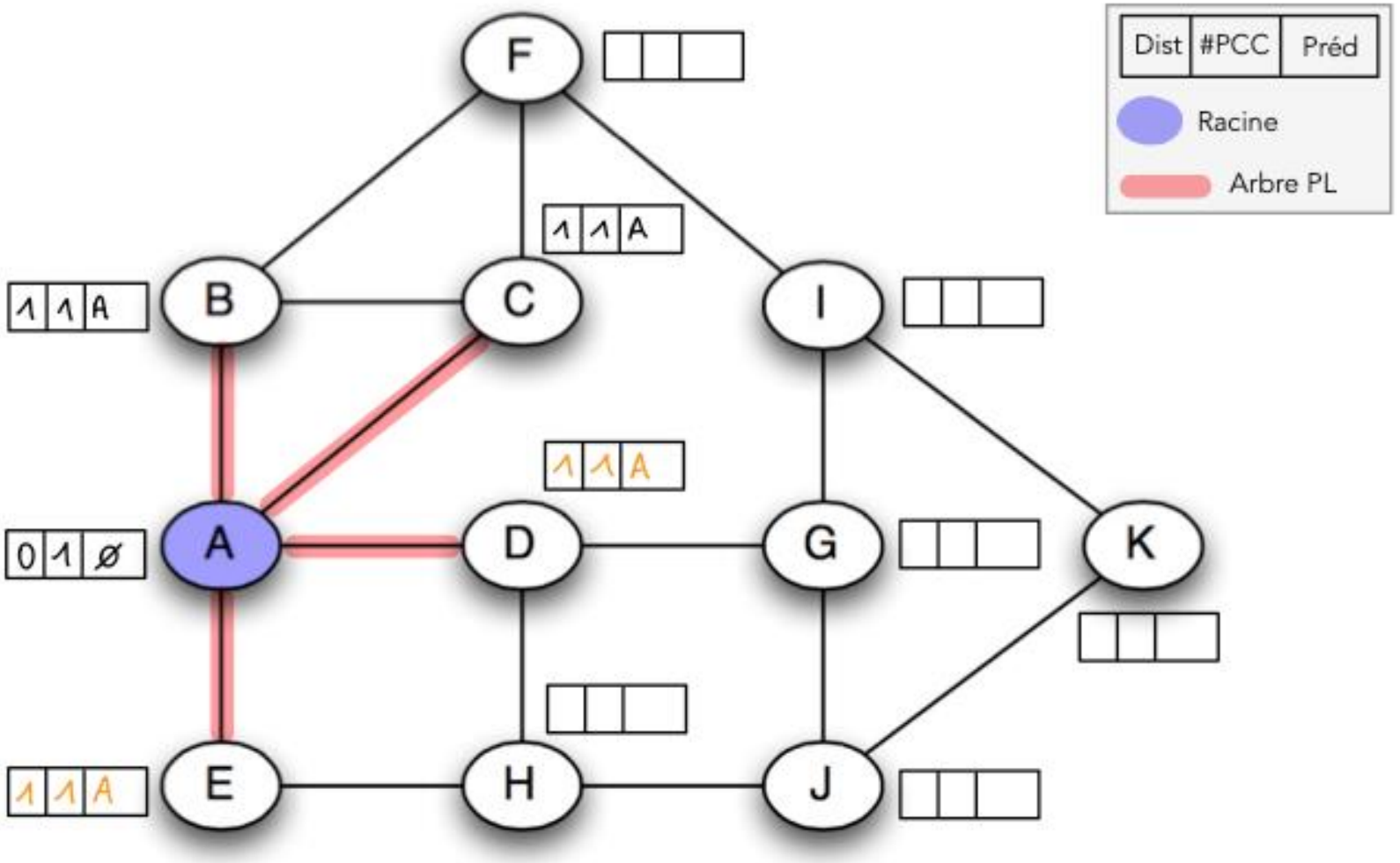
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
1 1 A
0 1 ∅
+1
Jamais visité

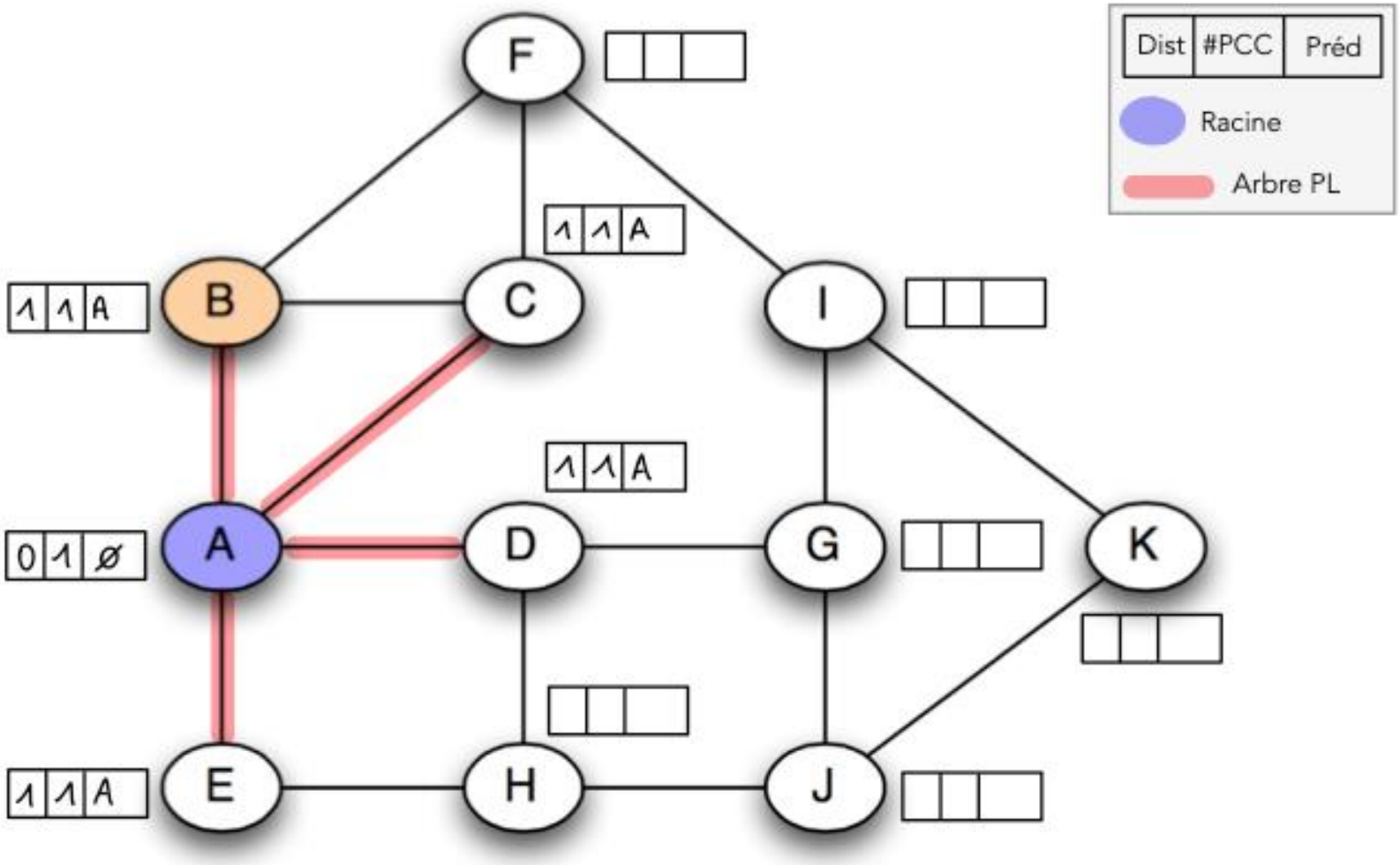
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
1 1 A
+1
Jamais visité
0 1 ∅

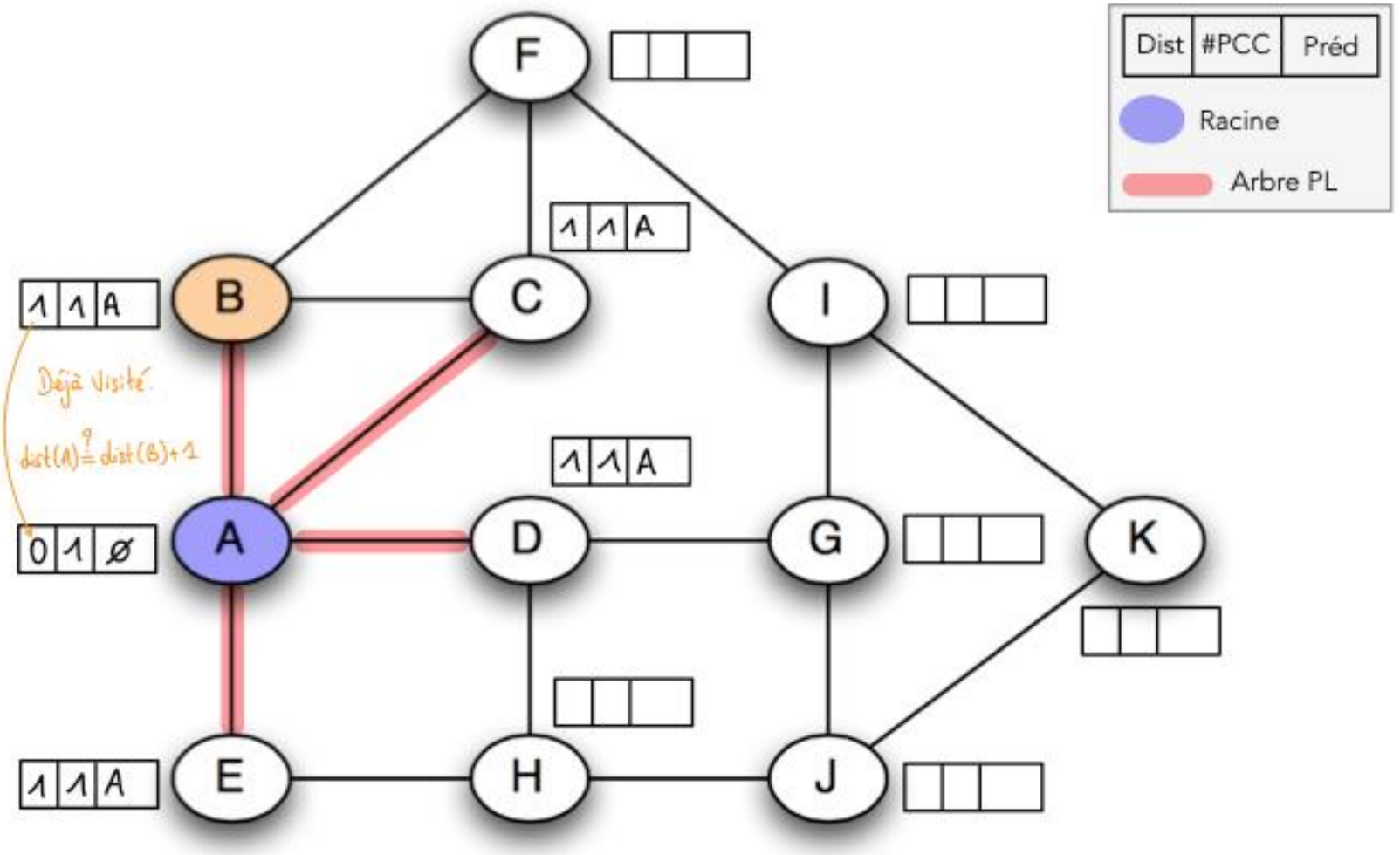
Un exemple



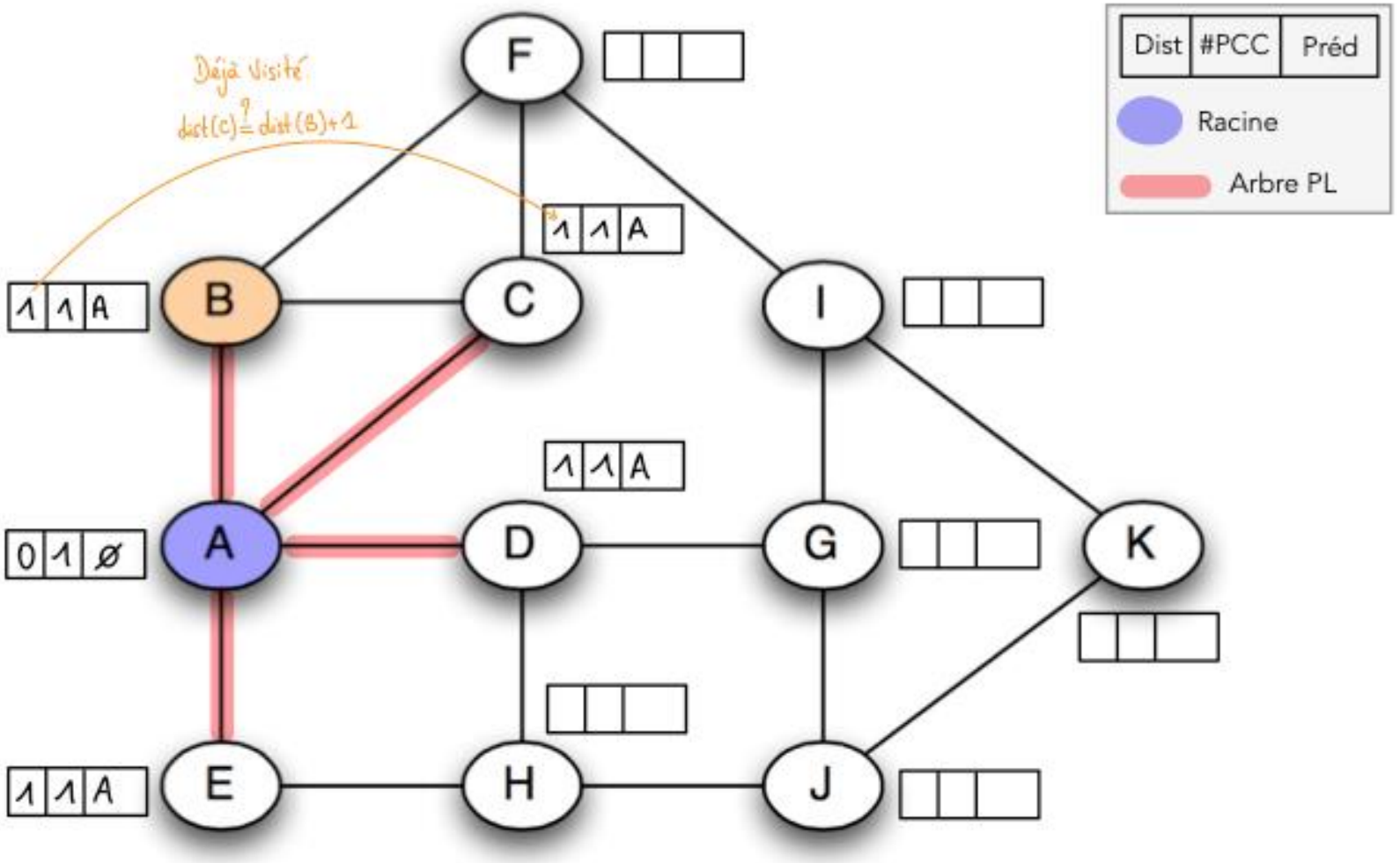
Un exemple



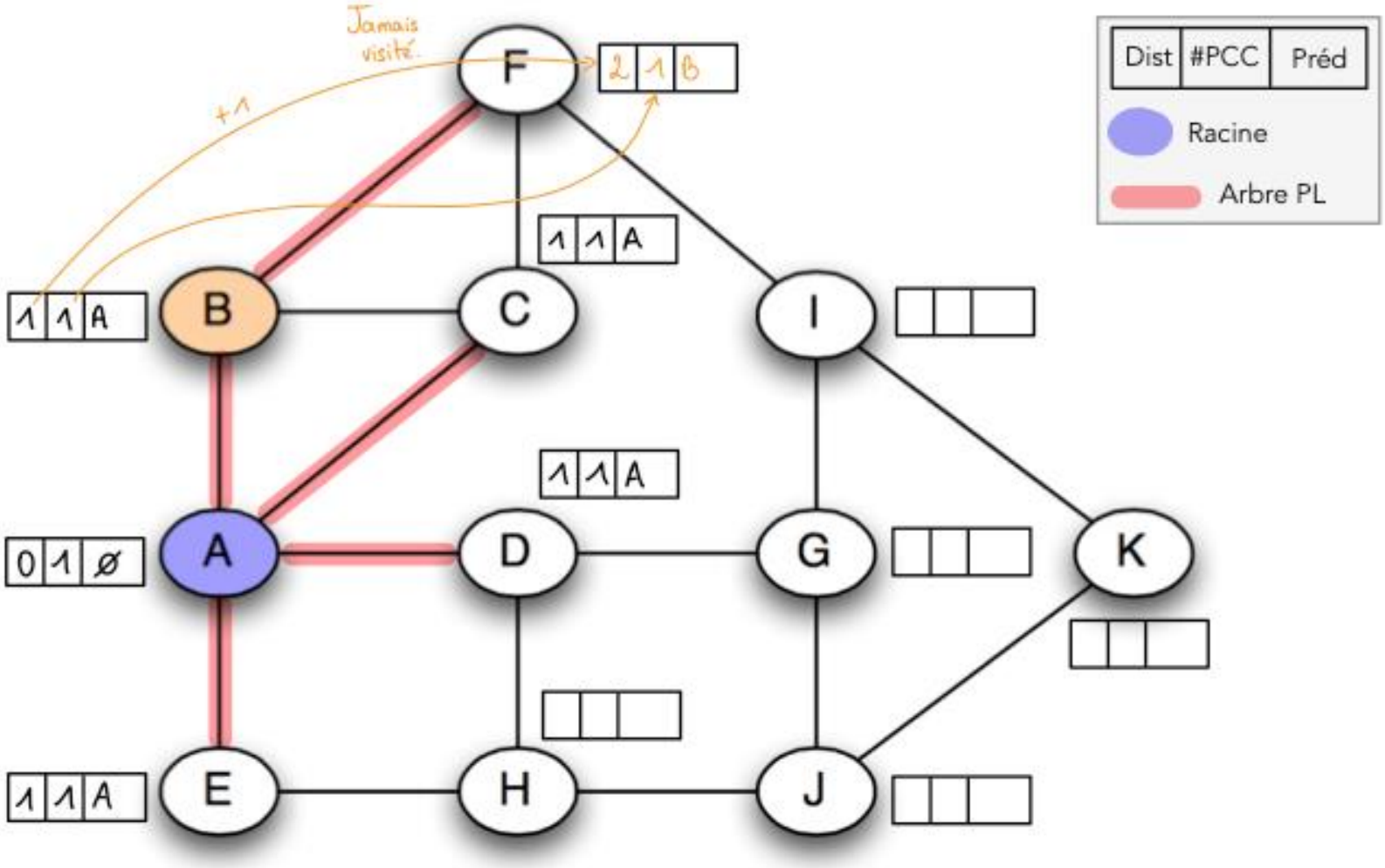
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
Déjà visité
dist(A) ? dist(B)+1

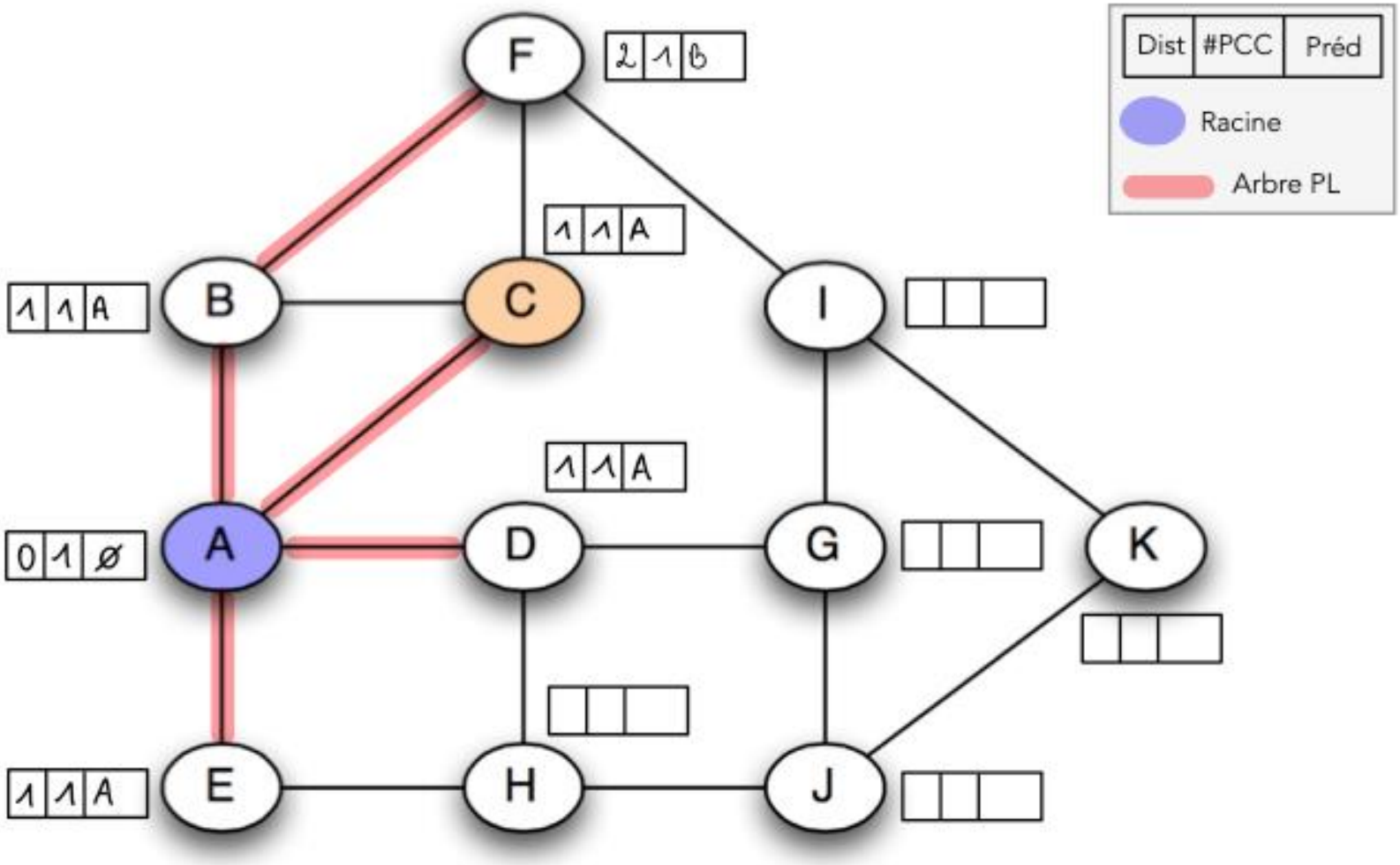
Un exemple


Déjà visité
dist(C) ? = dist(B)+1
Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J

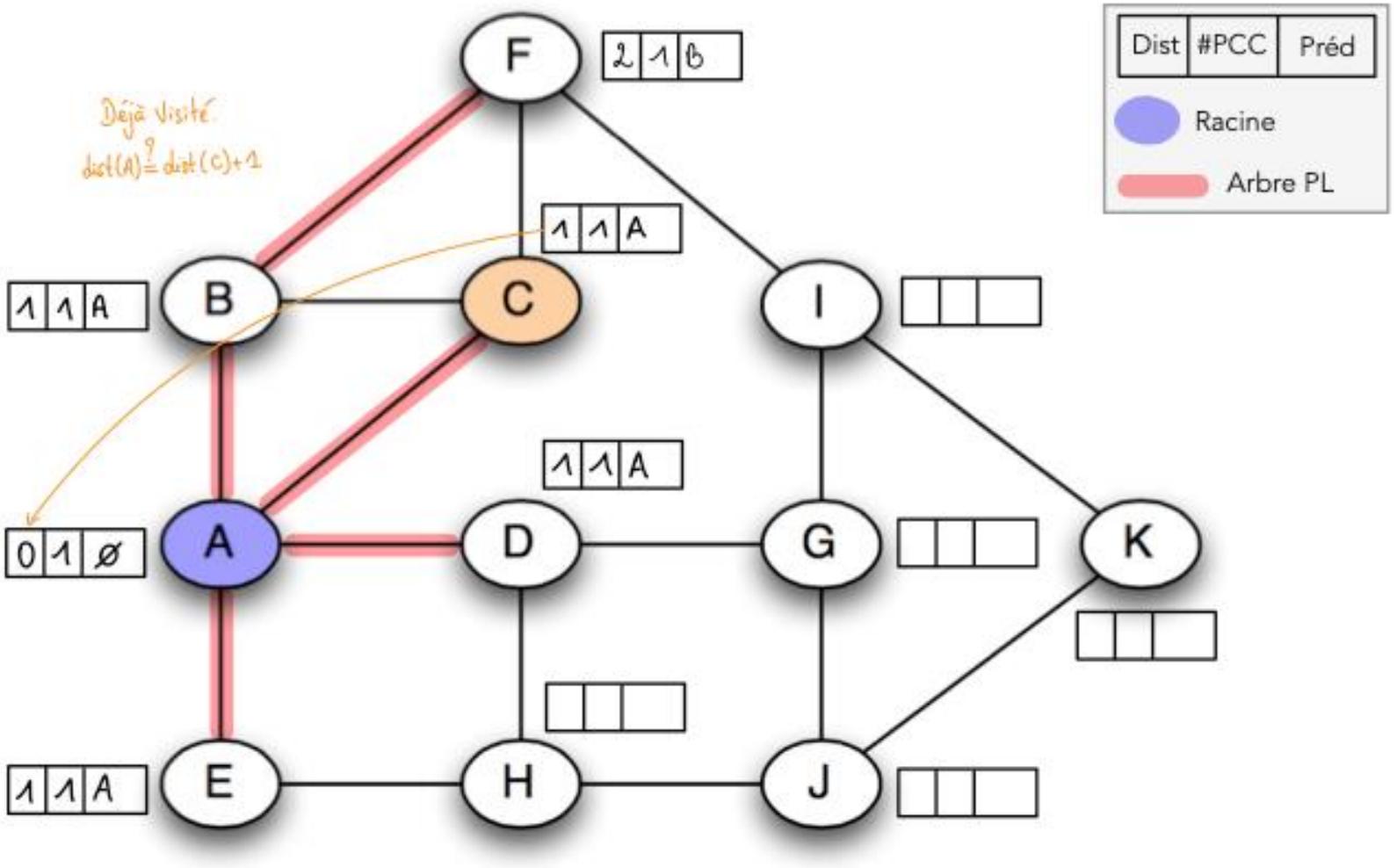
Un exemple


Jamais visité
+1
F
2 1 B
Dist
#PCC
Préd
Racine
Arbre PL
1 1 A
B
C
I
1 1 A
A
D
G
K
0 1 ∅
1 1 A
E
H
J

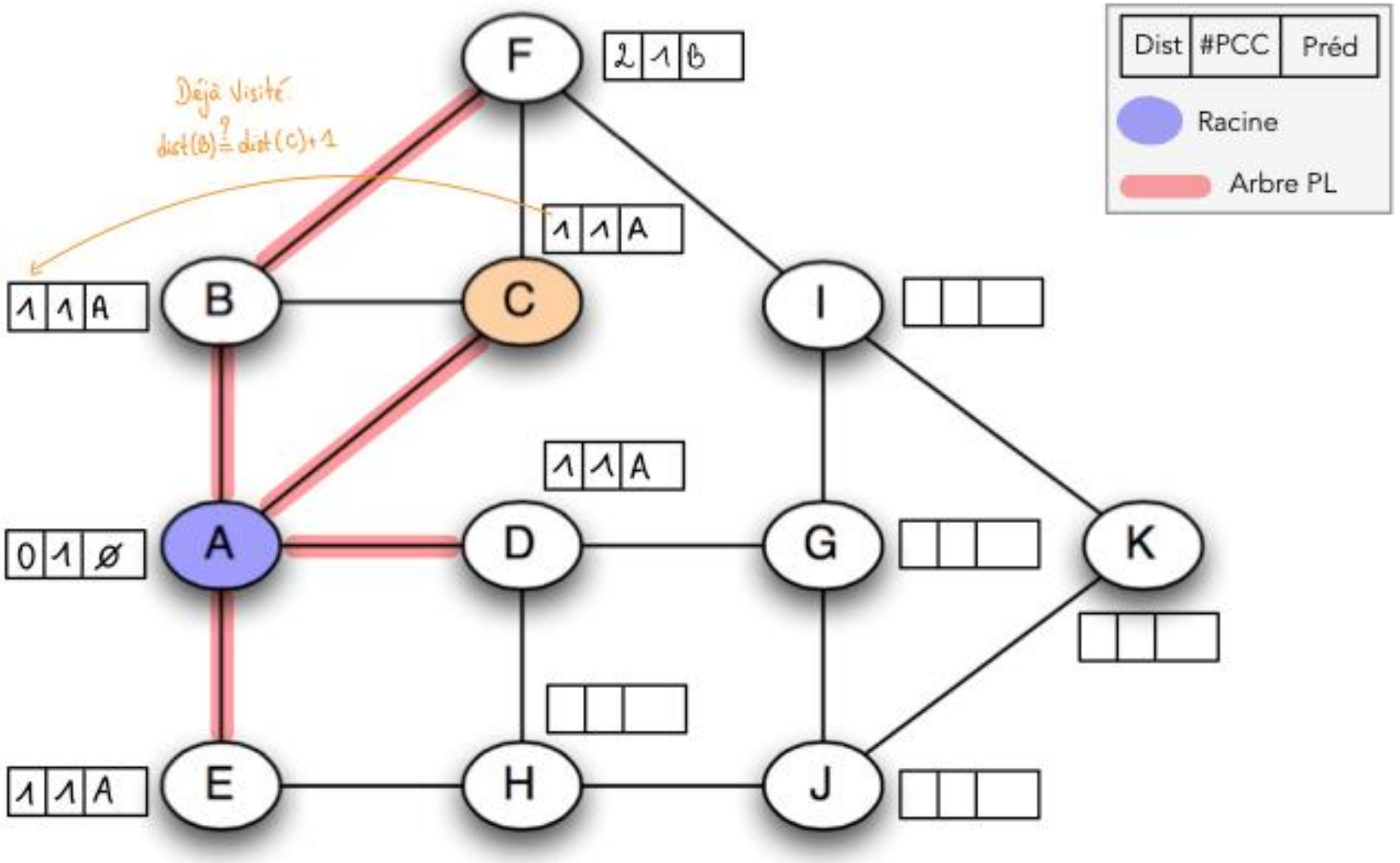
Un exemple



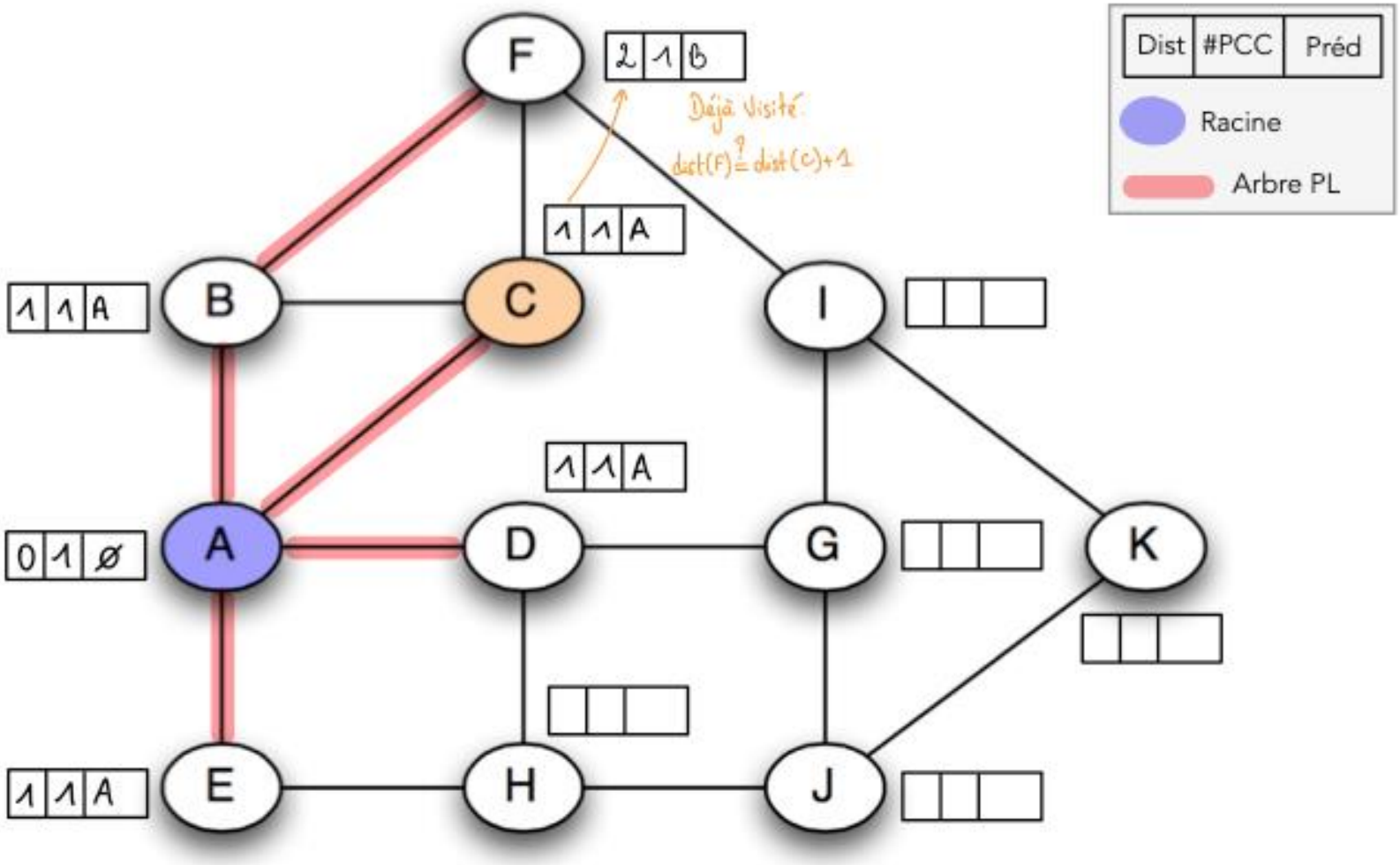
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
Déjà visité
dist(A) ? dist(C)+1
F
B
C
I
A
D
G
K
E
H
J

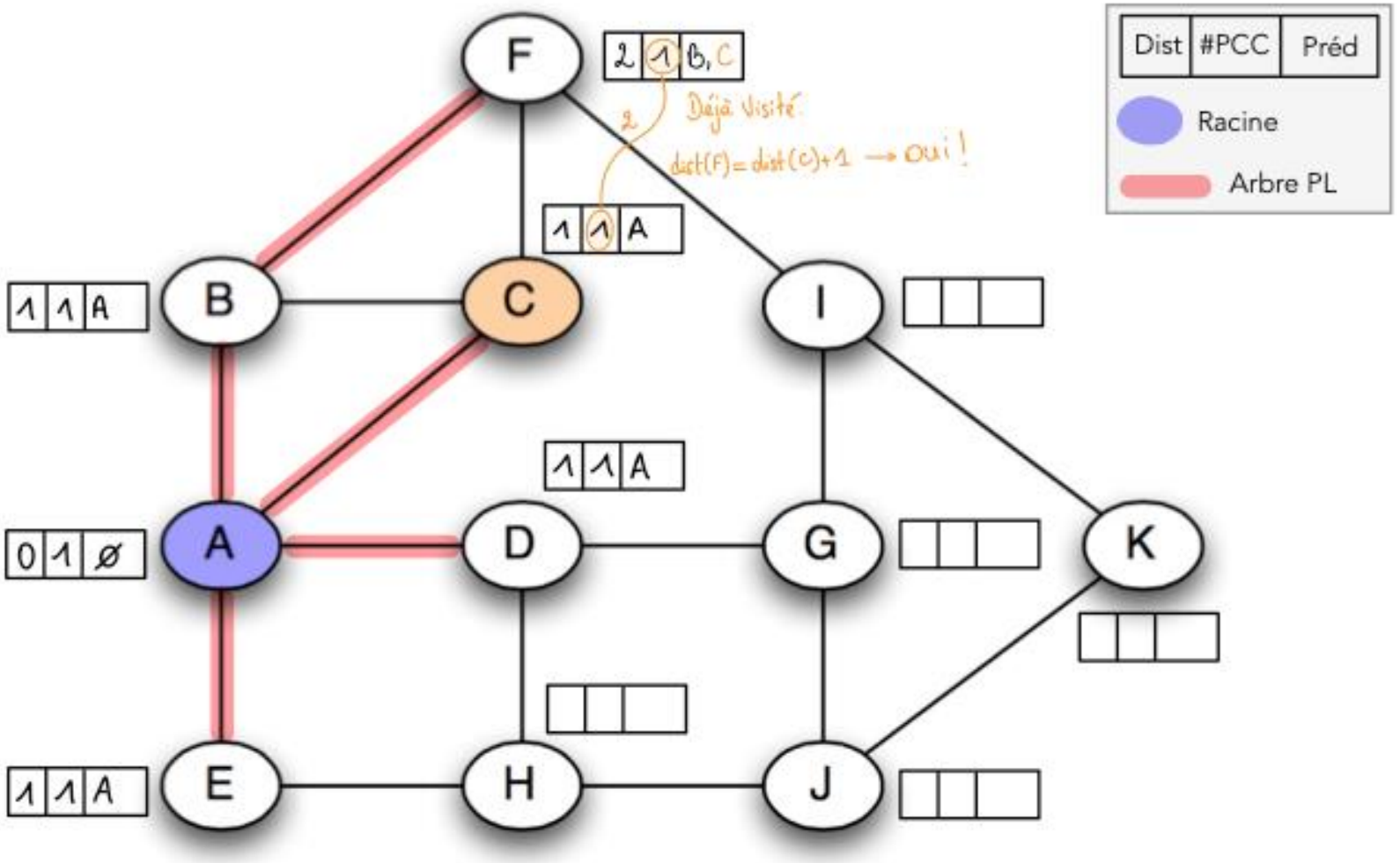
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
Déjà visité
dist(B) ? dist(C)+1
F
B
C
I
A
D
G
K
E
H
J

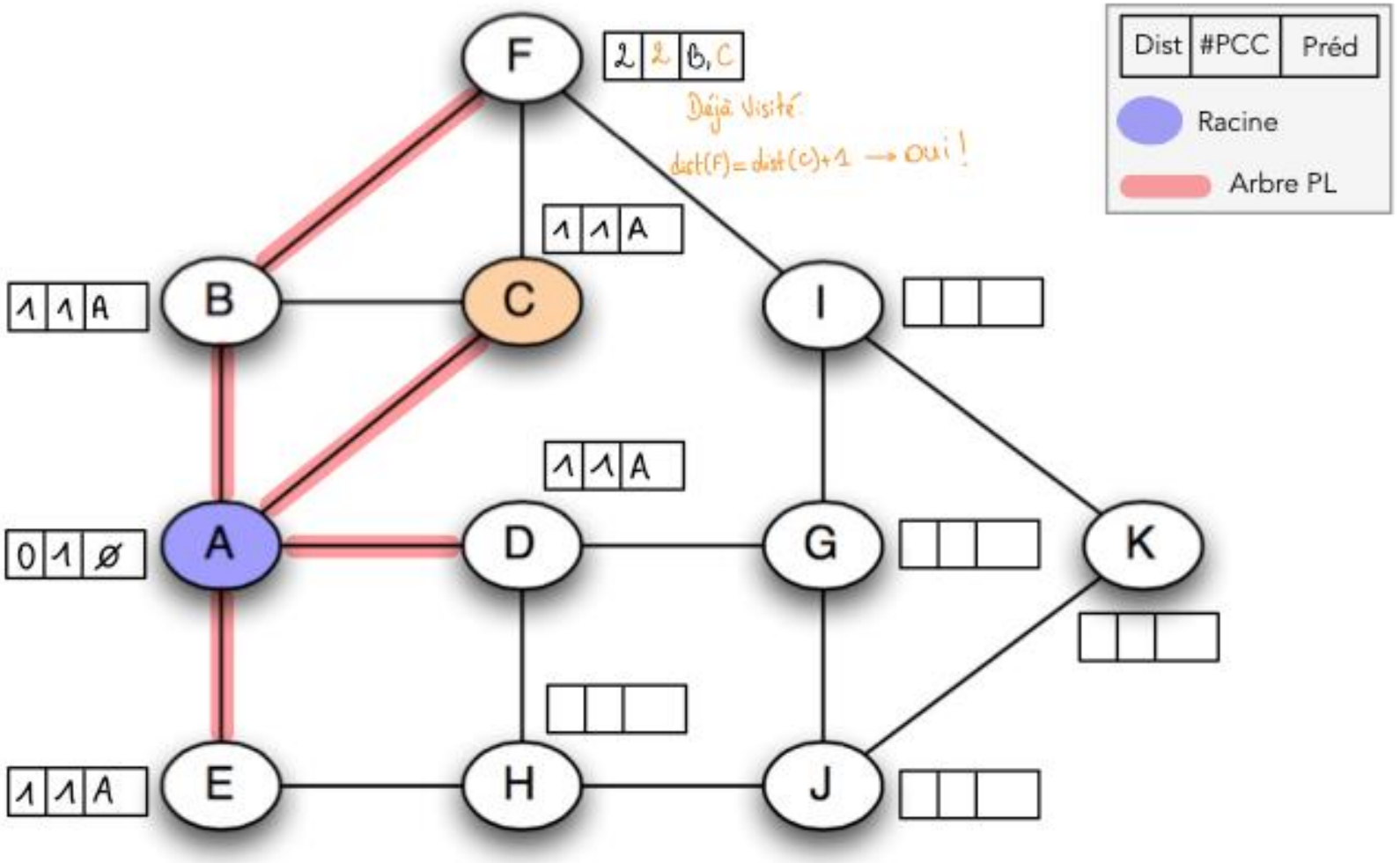
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
Déjà visité
dist(F) ? dist(C)+1

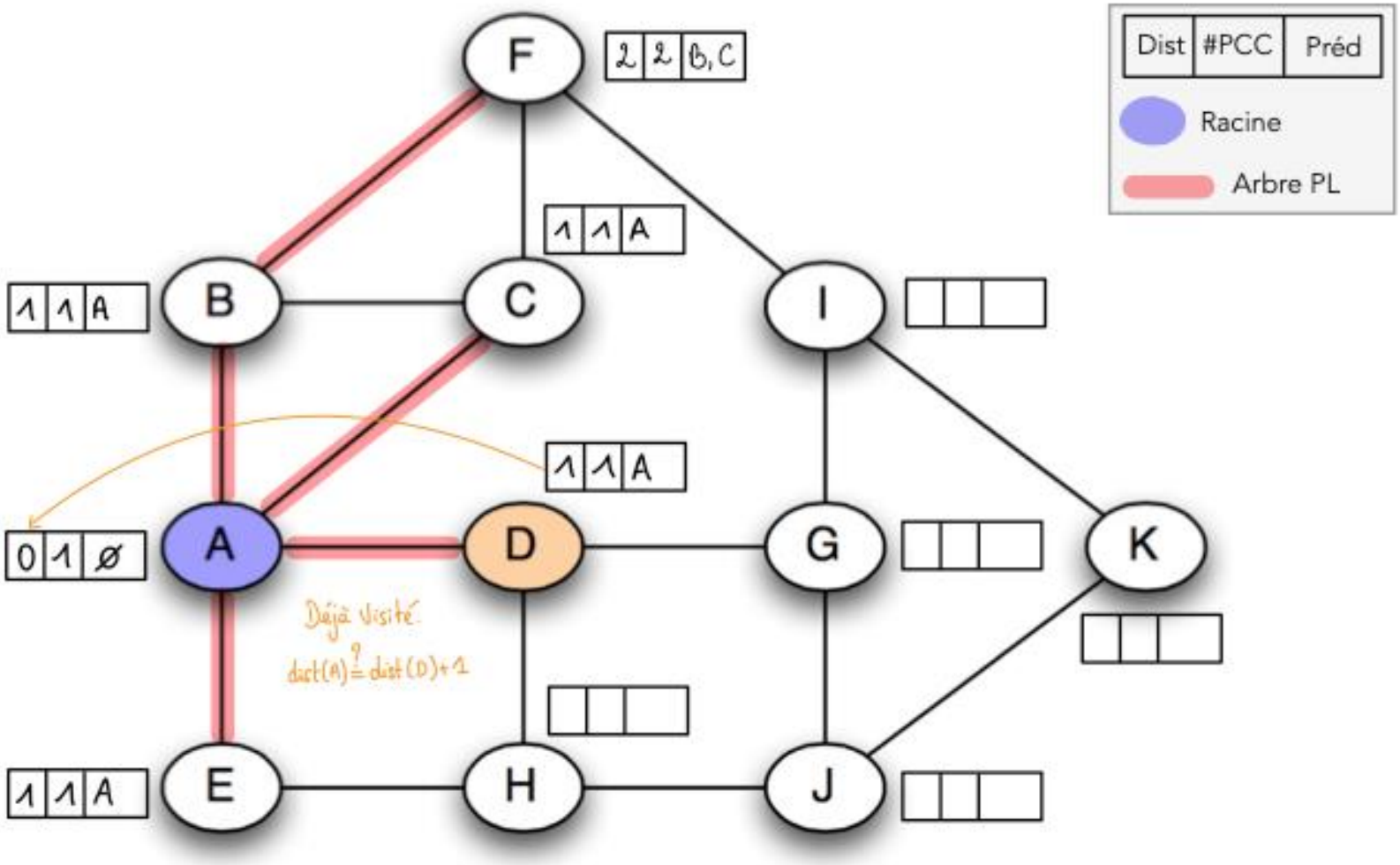
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
Déjà visité
dist(F) = dist(C)+1 → oui !

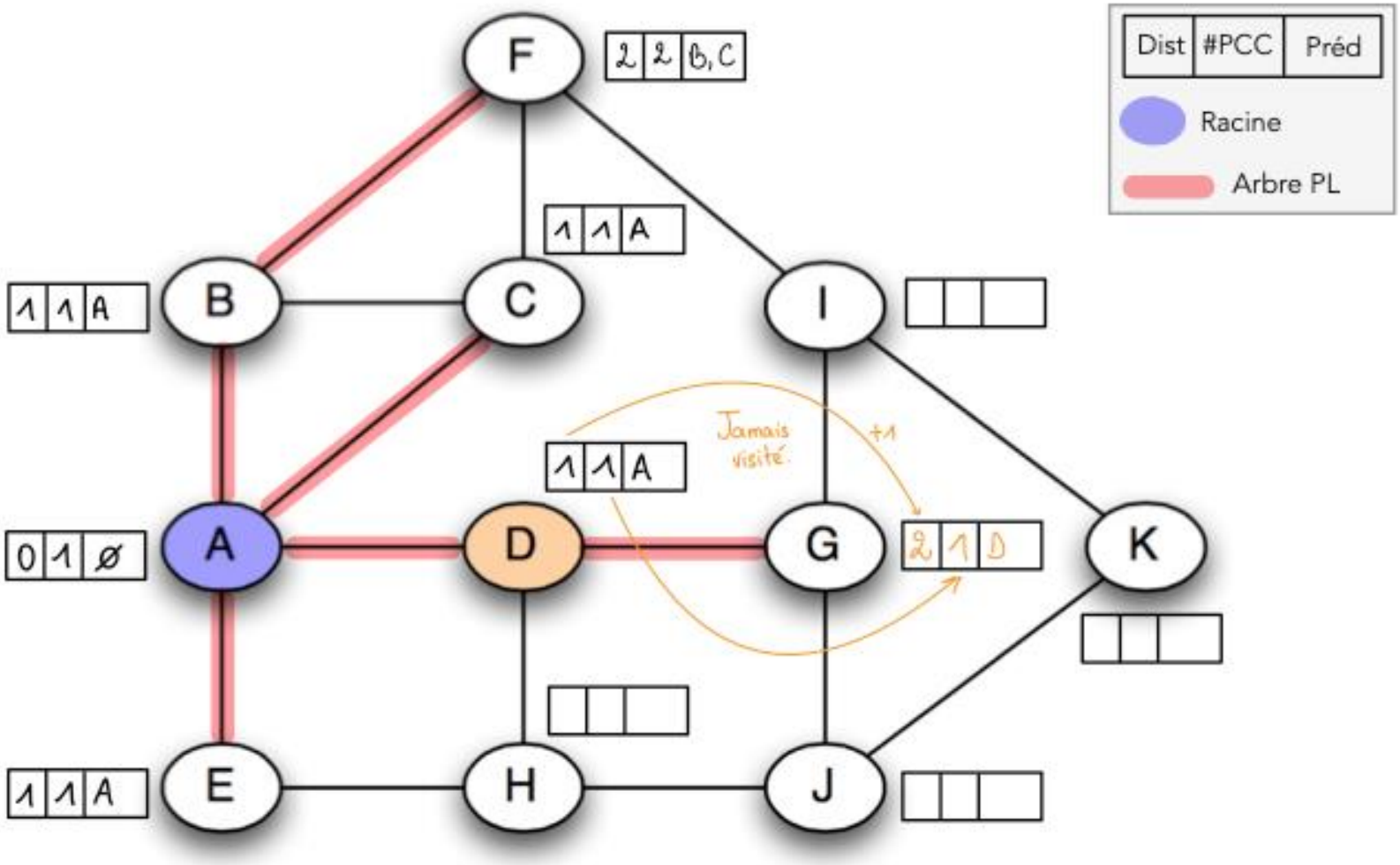
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
Déjà visité
dist(F) = dist(C)+1 → oui !

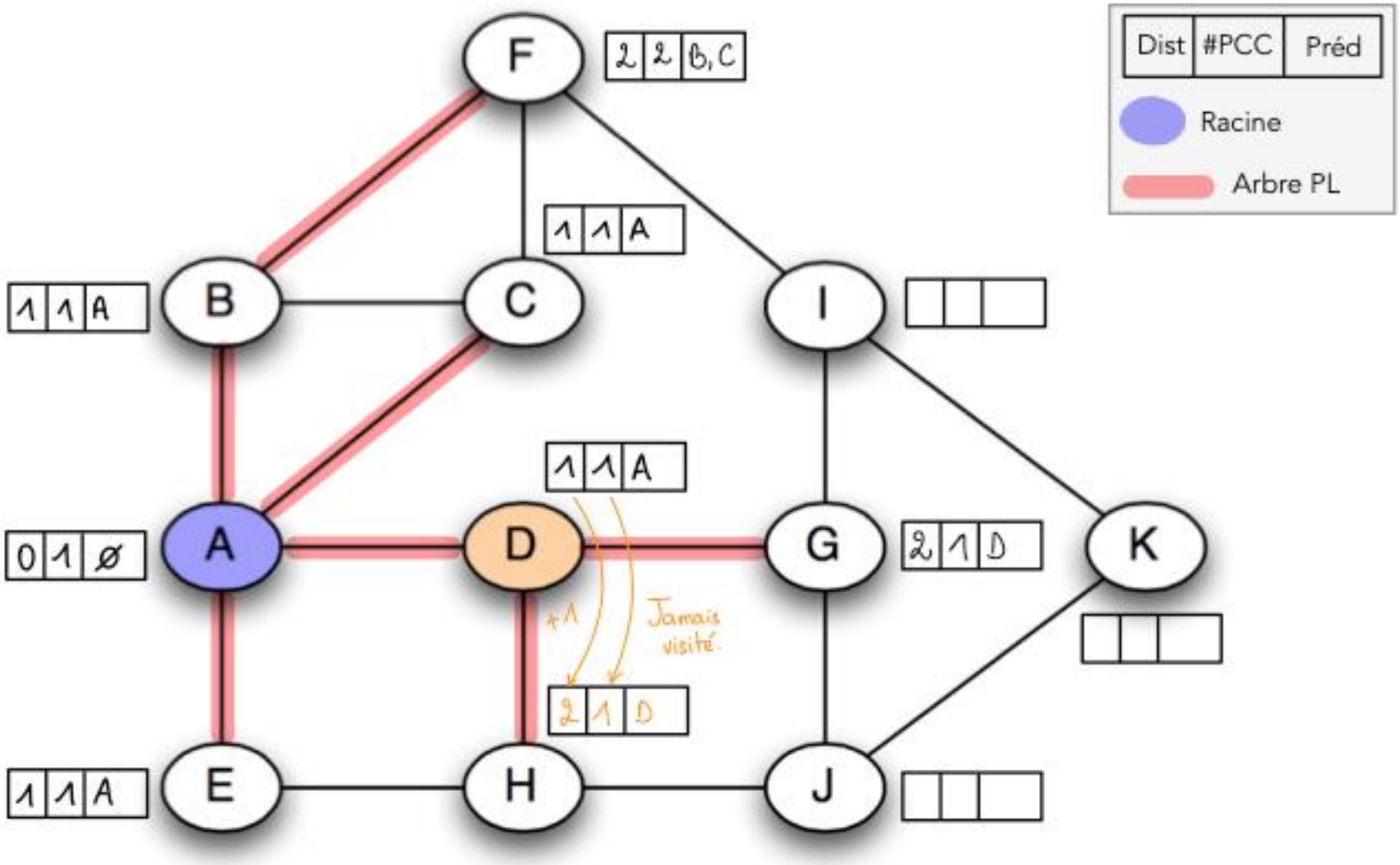
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
Déjà visité
dist(A) ? dist(D)+1

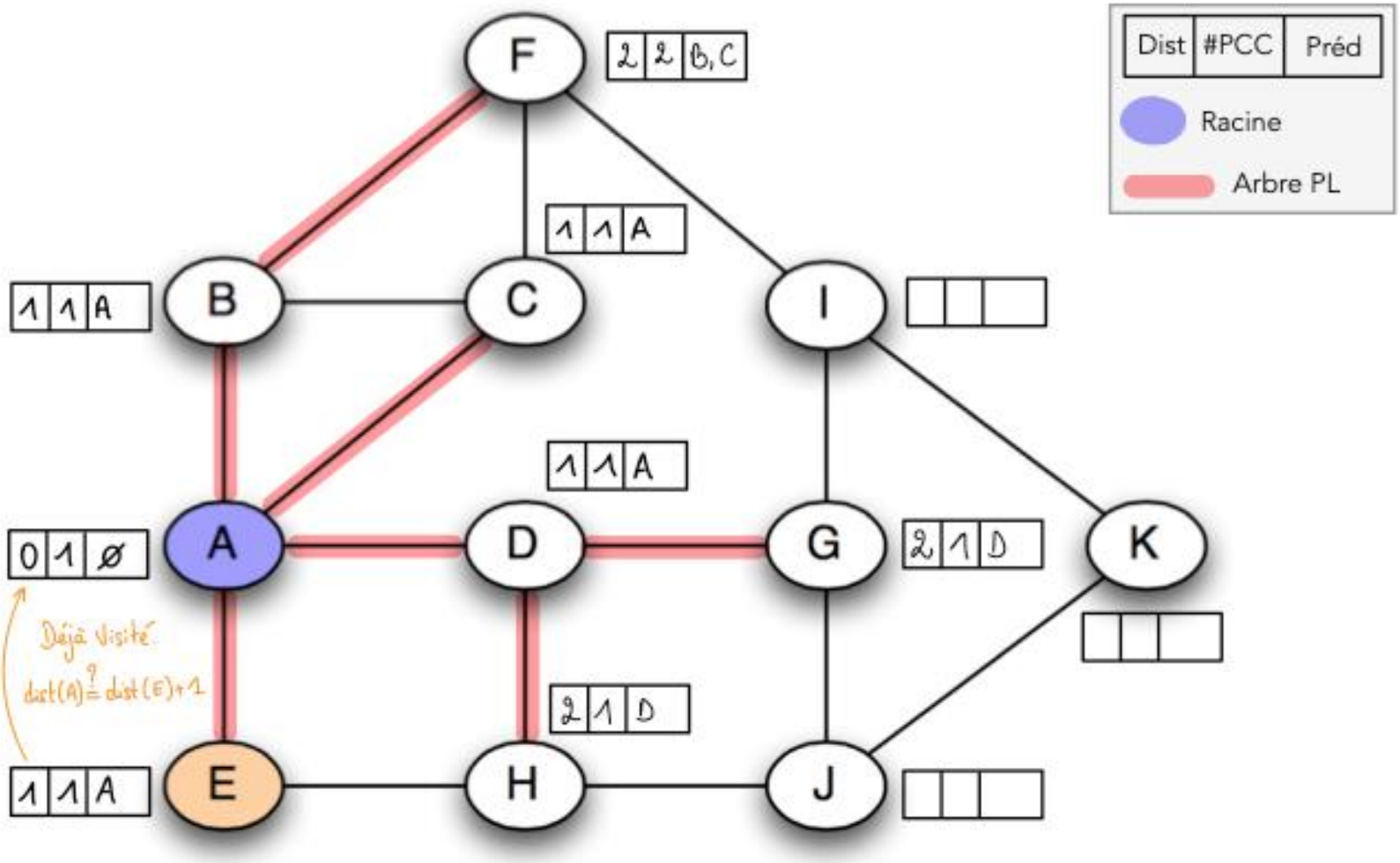
Un exemple



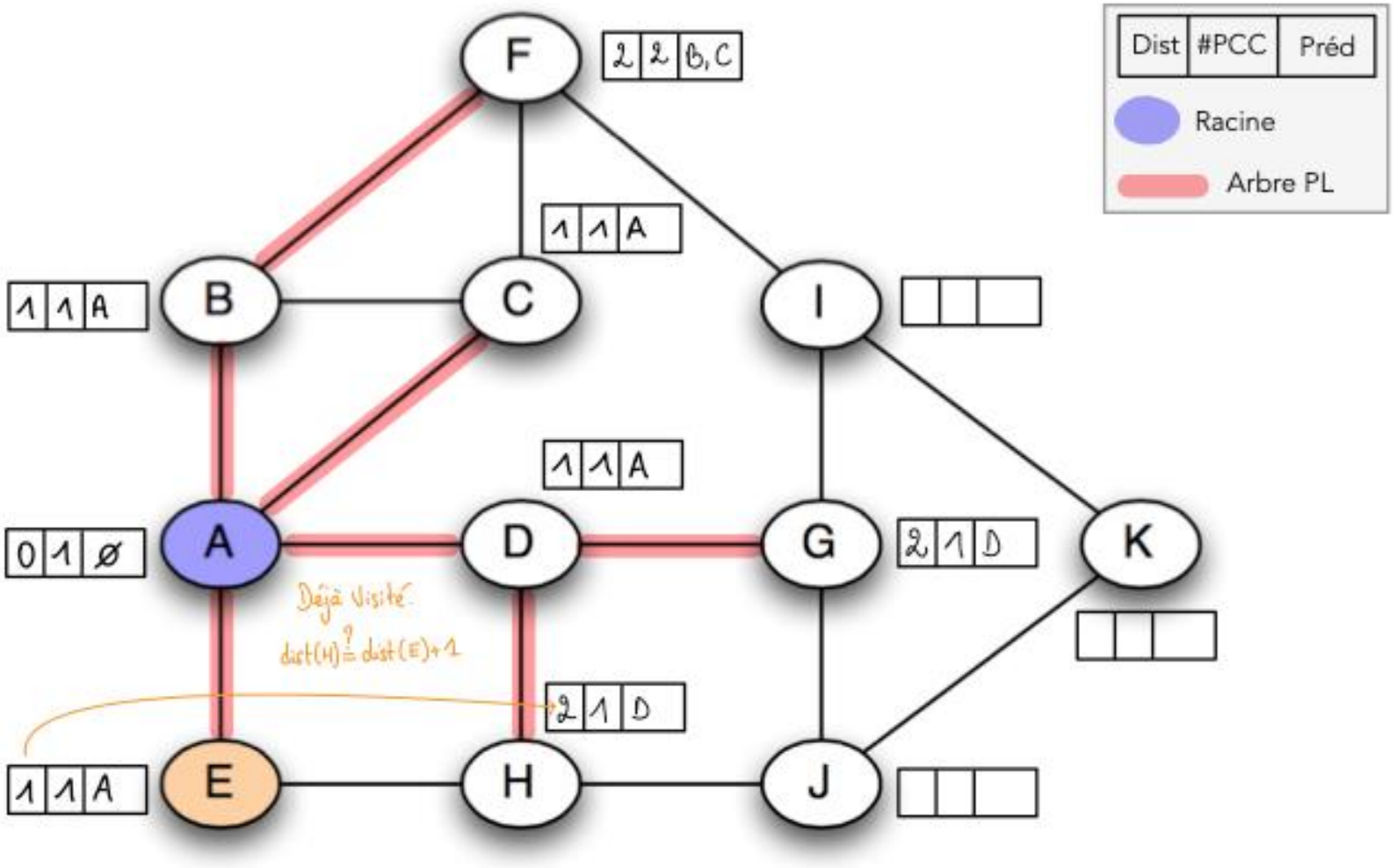
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
C
I
1 1 A
A
D
G
2 1 D
K
0 1 ∅
1 1 A
+1
Jamais visité.
2 1 D
E
H
J
1 1 A

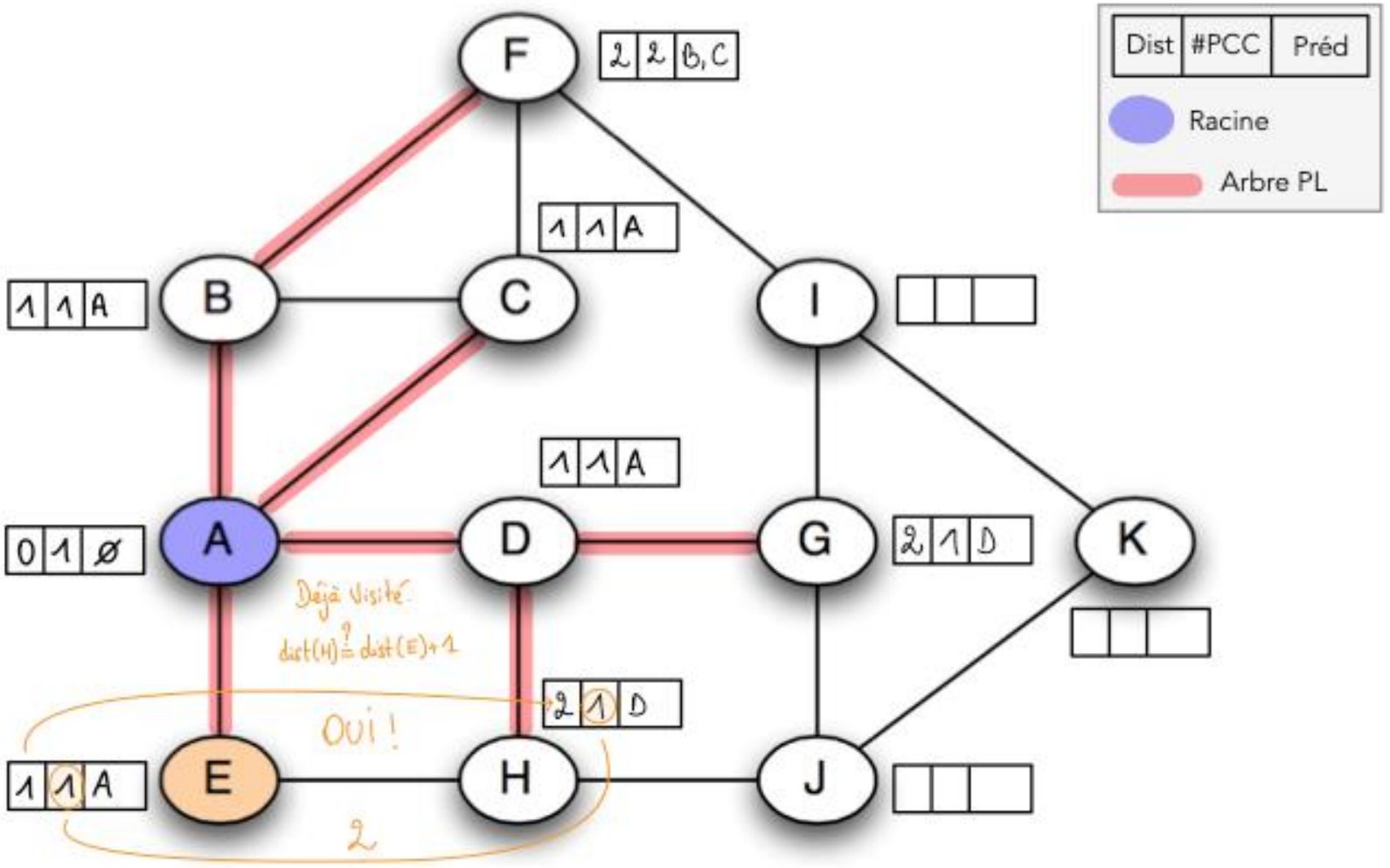
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
1 1 A
C
I
1 1 A
A
0 1 ∅
D
G
2 1 D
K
Déjà visité
dist(A) ? dist(E)+1
2 1 D
E
1 1 A
H
J

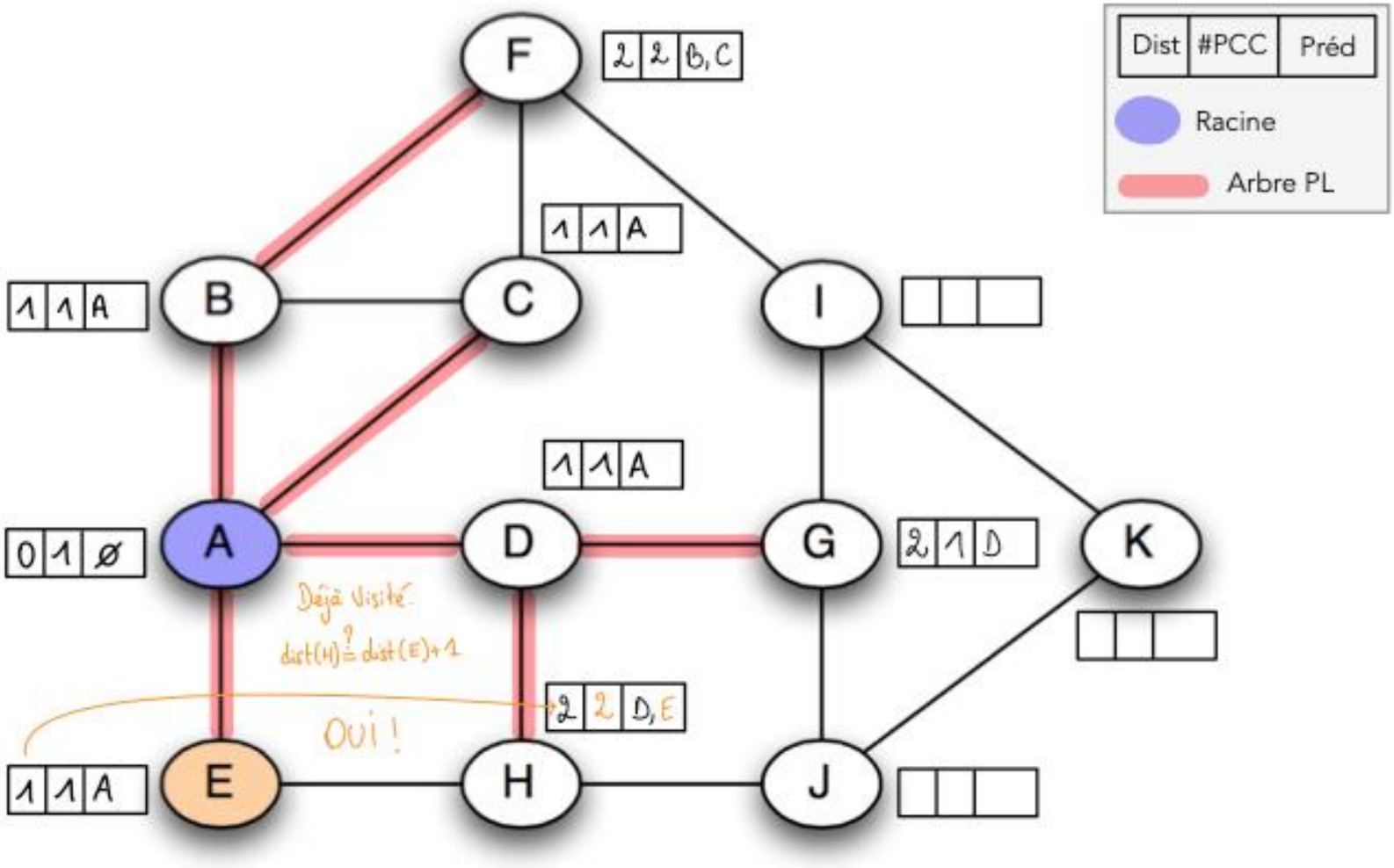
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
1 1 A
A
0 1 ∅
D
G
2 1 D
K
Déjà visité
dist(H) ? dist(E)+1
2 1 D
E
1 1 A
H
J

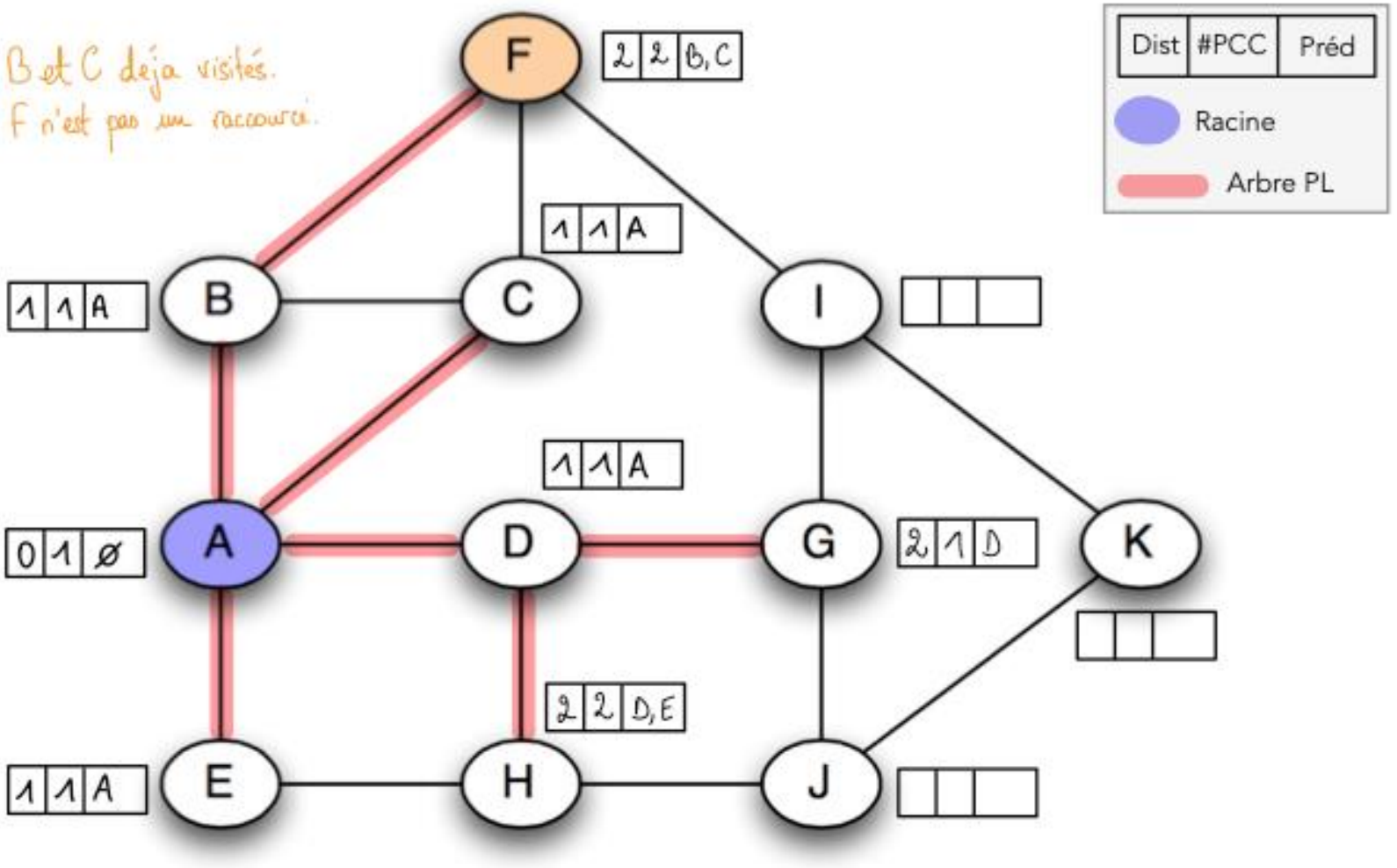
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
A
0 1 ∅
1 1 A
D
G
2 1 D
K
Déjà visité
dist(H) ? dist(E)+1
Oui !
2 1 D
E
1 1 A
H
J
2

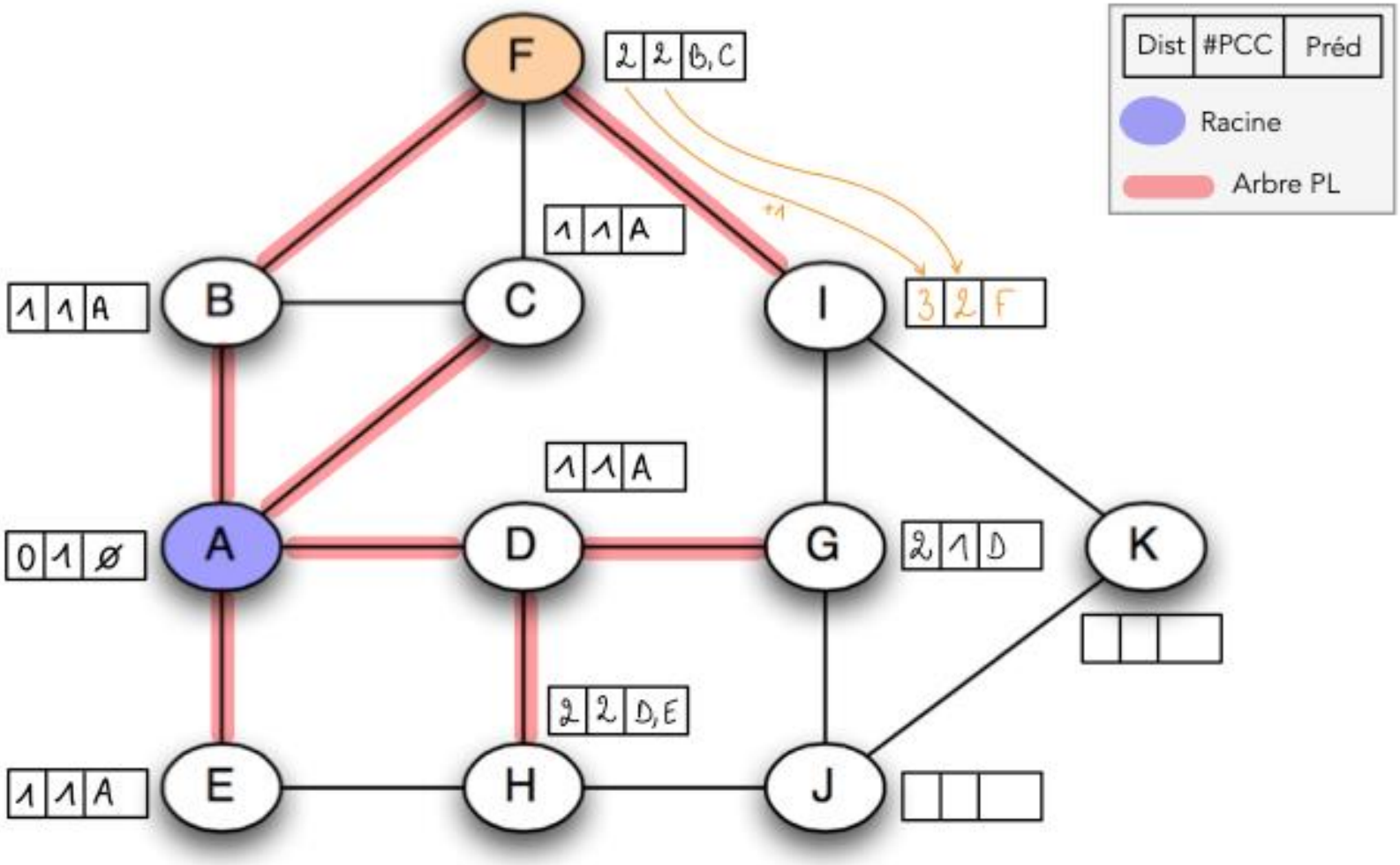
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
A
1 1 A
0 1 ∅
D
G
2 1 D
K
Déjà visité
dist(H) ? dist(E)+1
Oui !
2 2 D, E
E
1 1 A
H
J

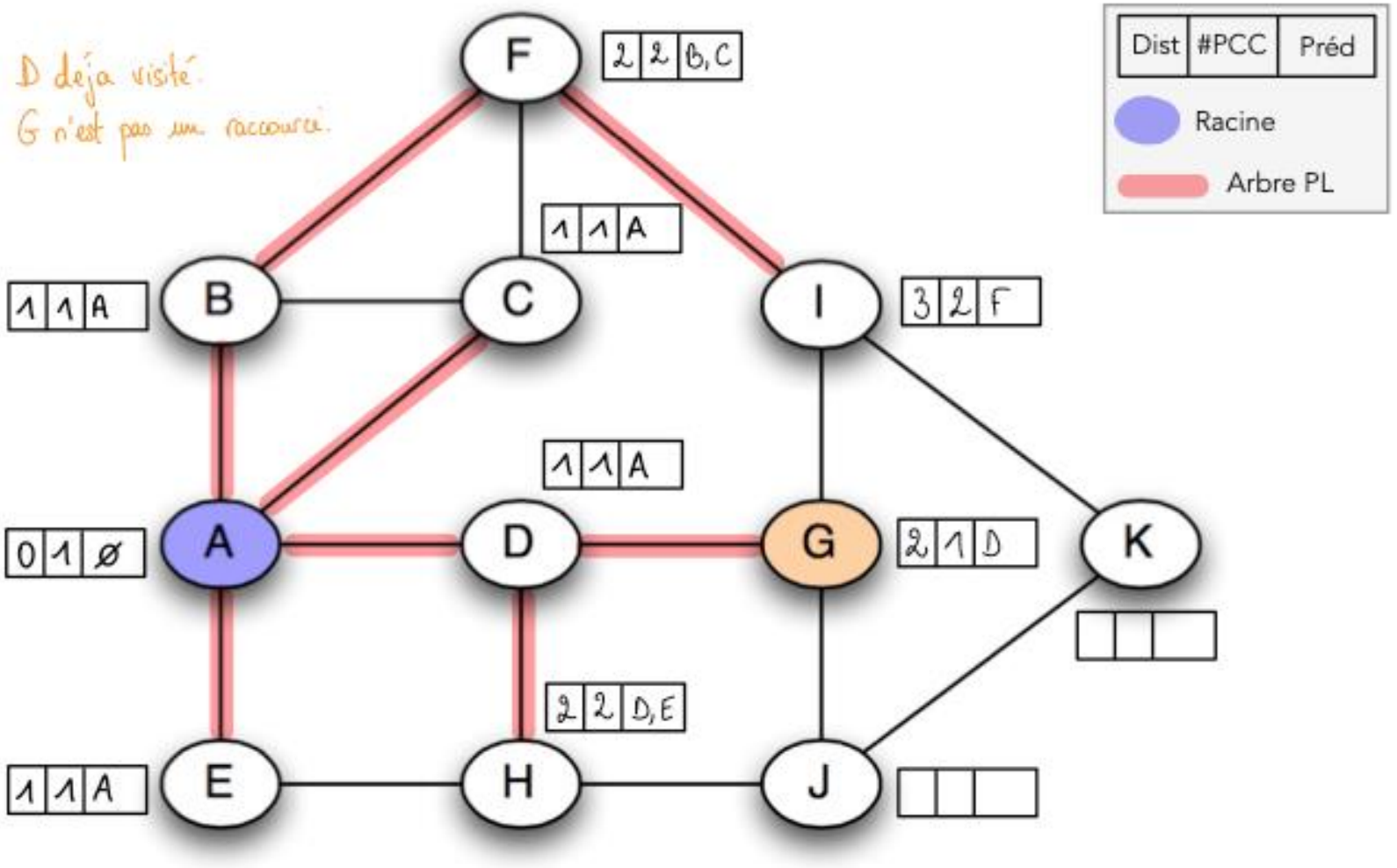
Un exemple


B et C déjà visités.
F n'est pas un raccourci.
Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
C
I
1 1 A
1 1 A
A
D
G
2 1 D
K
0 1 ∅
2 2 D,E
E
H
J
1 1 A

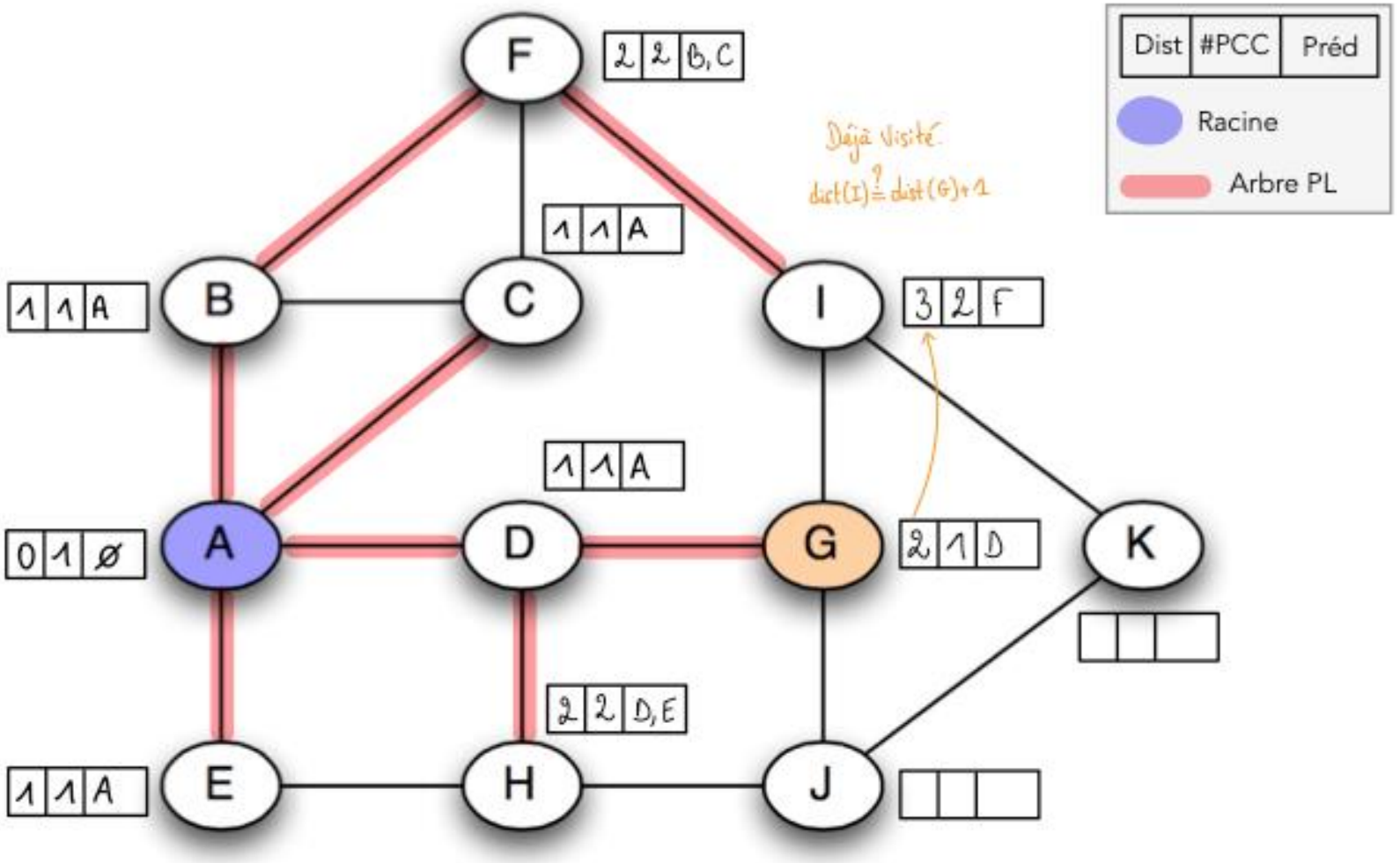
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
B
1 1 A
C
1 1 A
I
3 2 F
A
0 1 ∅
D
1 1 A
G
2 1 D
K
E
1 1 A
H
2 2 D, E
J

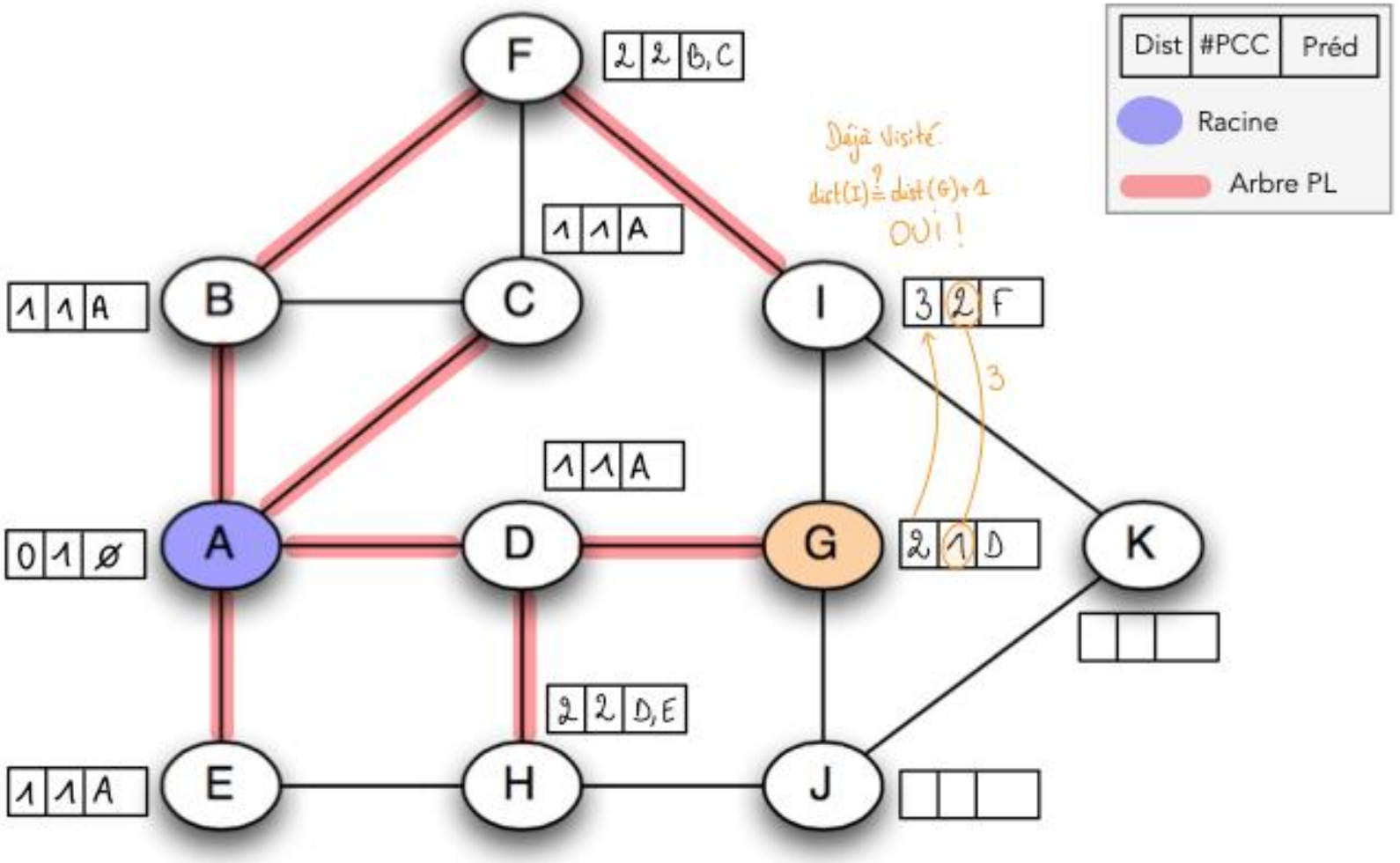
Un exemple


D déjà visité.
G n'est pas un raccourci.
Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
C
I
1 1 A
3 2 F
1 1 A
A
D
G
K
0 1 ∅
2 1 D
2 2 D,E
E
H
J
1 1 A

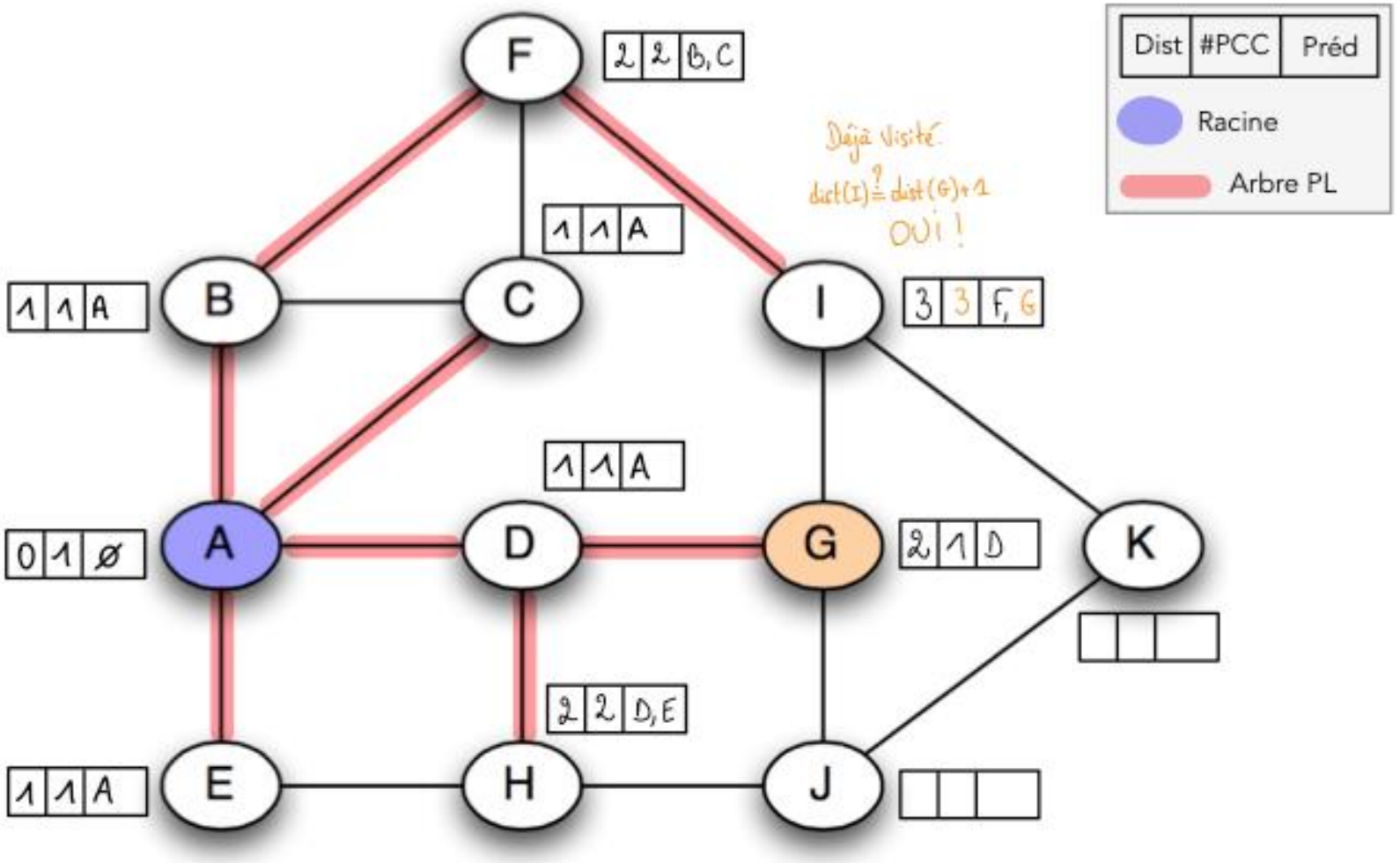
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
Déjà visité
dist(I) ? dist(G)+1
F
2 2 B,C
B
1 1 A
C
1 1 A
I
3 2 F
A
0 1 ∅
D
1 1 A
G
2 1 D
K
E
1 1 A
H
2 2 D,E
J

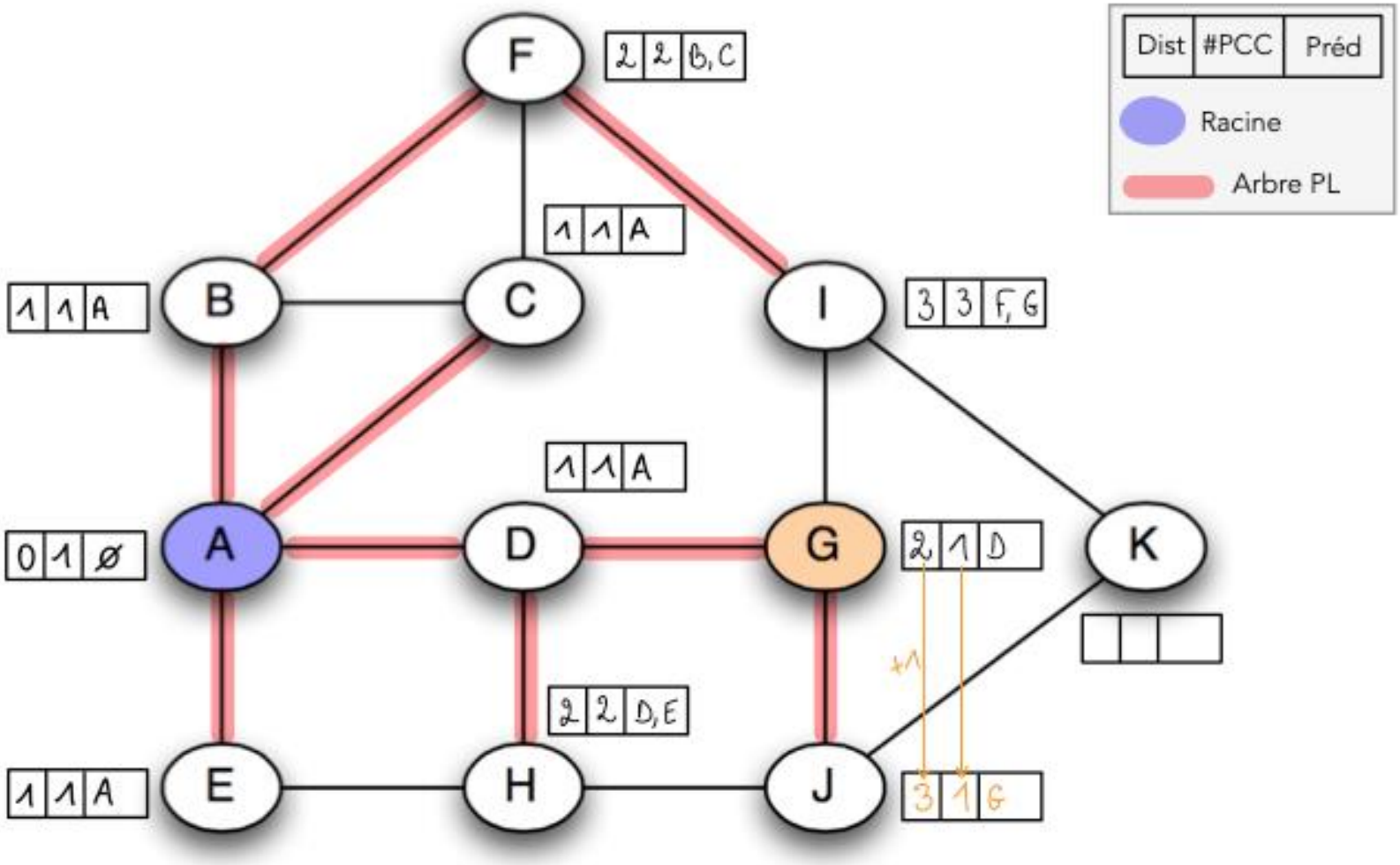
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
Déjà visité
dist(I) = ? dist(G)+1
OUI !
1 1 A
B
C
I
3 2 F
3
1 1 A
1 1 A
A
D
G
2 1 D
K
0 1 ∅
2 2 D,E
E
H
J
1 1 A

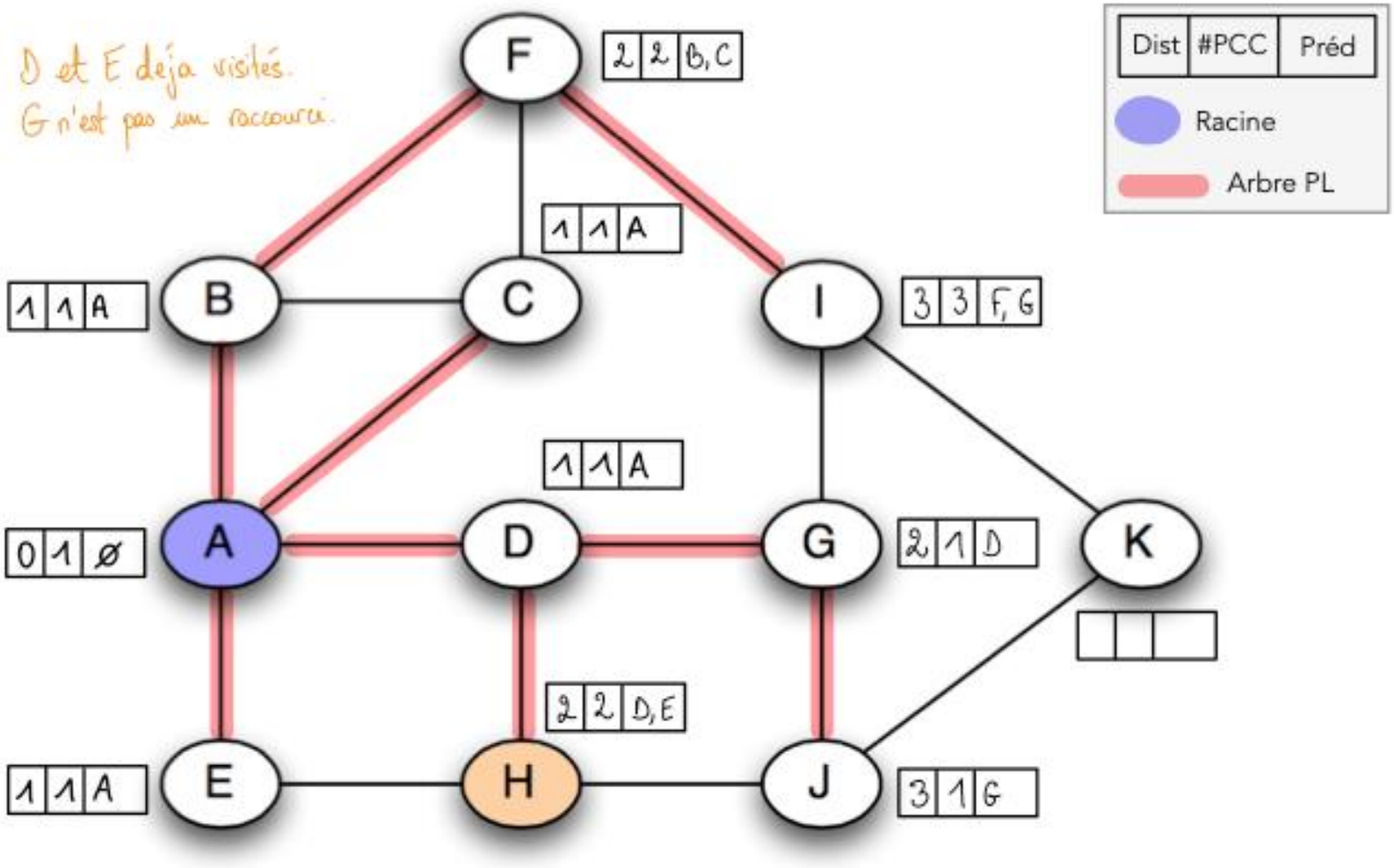
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
Déjà visité
dist(I) = ? dist(G)+1
OUI !
1 1 A
B
C
I
3 3 F, G
1 1 A
0 1 ∅
A
D
G
2 1 D
K
1 1 A
2 2 D, E
E
H
J

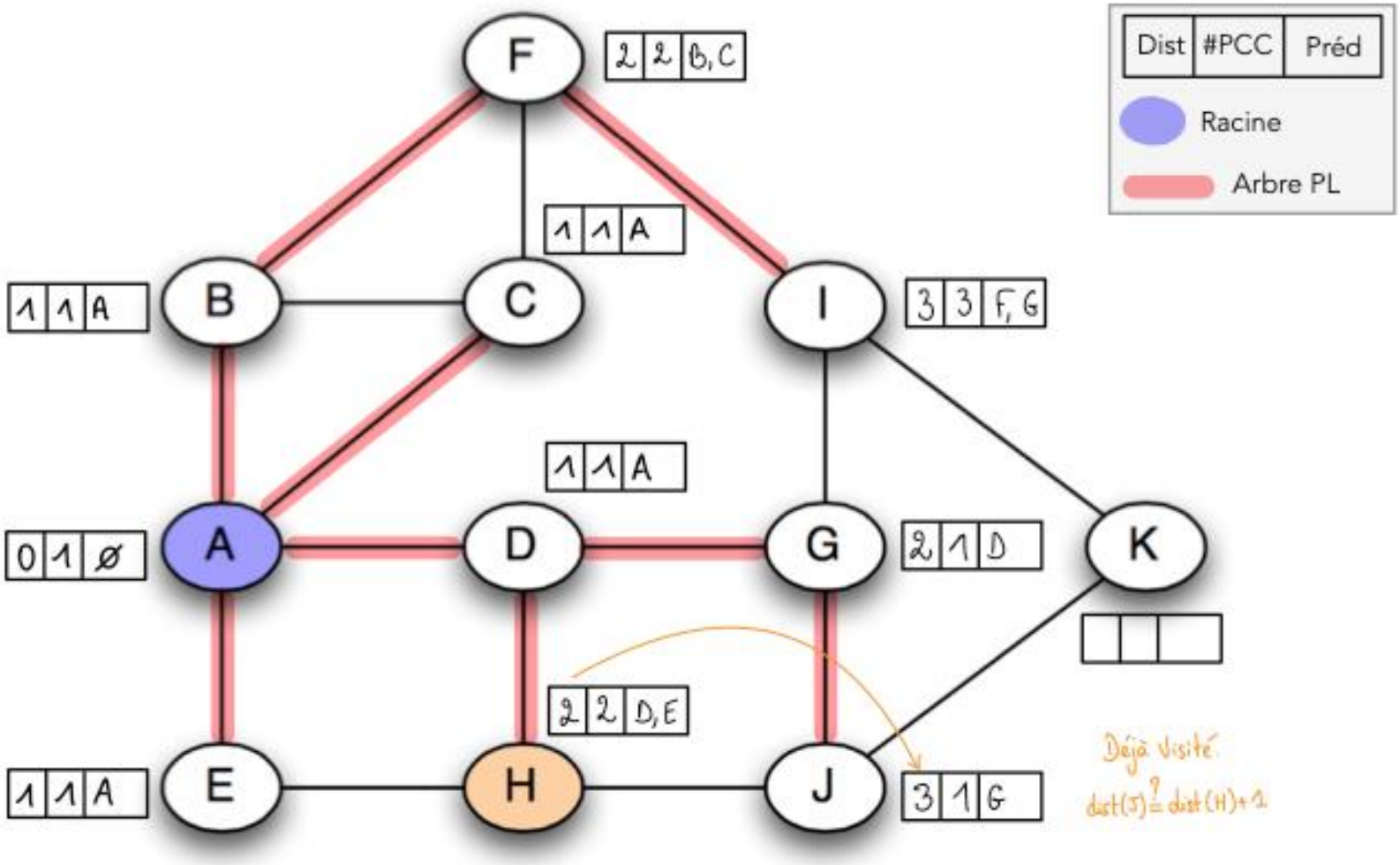
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
C
I
3 3 F, G
1 1 A
1 1 A
A
D
G
2 1 D
K
0 1 ∅
+1
2 2 D, E
E
H
J
3 1 G
1 1 A

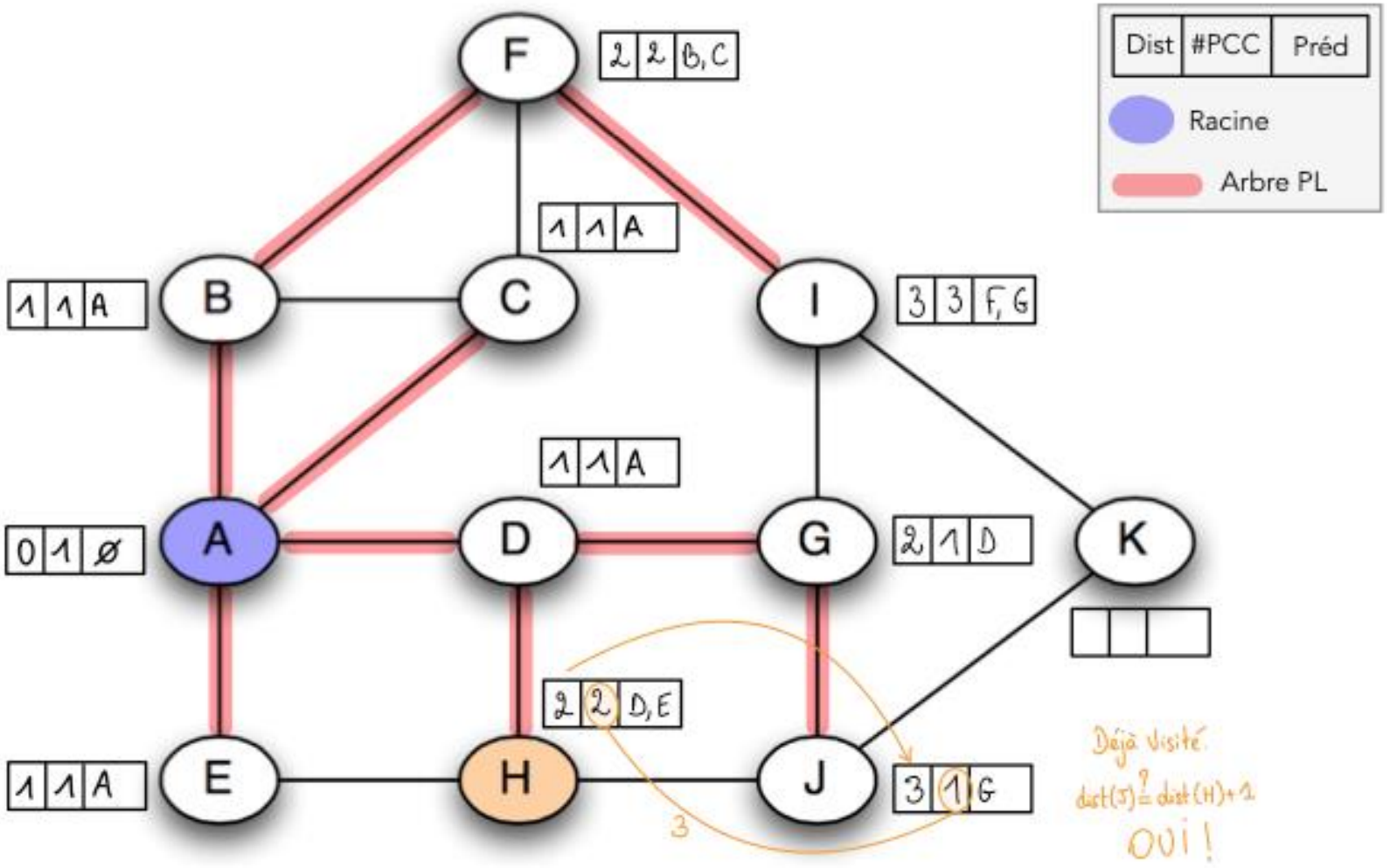
Un exemple


D et E déjà visités.
G n'est pas un raccourci.
Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
C
I
3 3 F,G
1 1 A
1 1 A
A
D
G
2 1 D
K
0 1 ∅
2 2 D,E
E
H
J
3 1 G
1 1 A

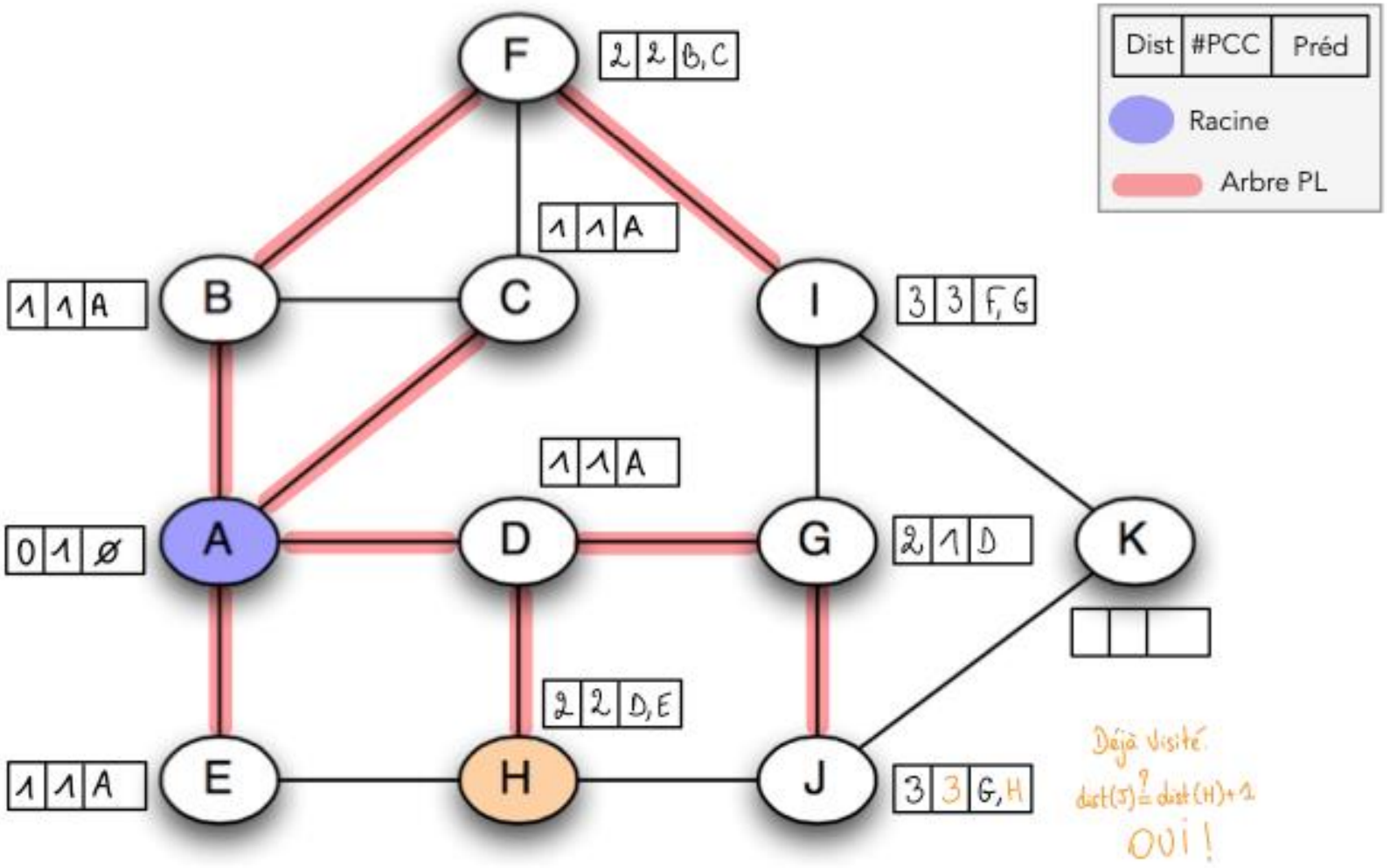
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
3 3 F, G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
2 2 D, E
E
1 1 A
H
J
3 1 G
Déjà visité
dist(J) ? dist(H)+1

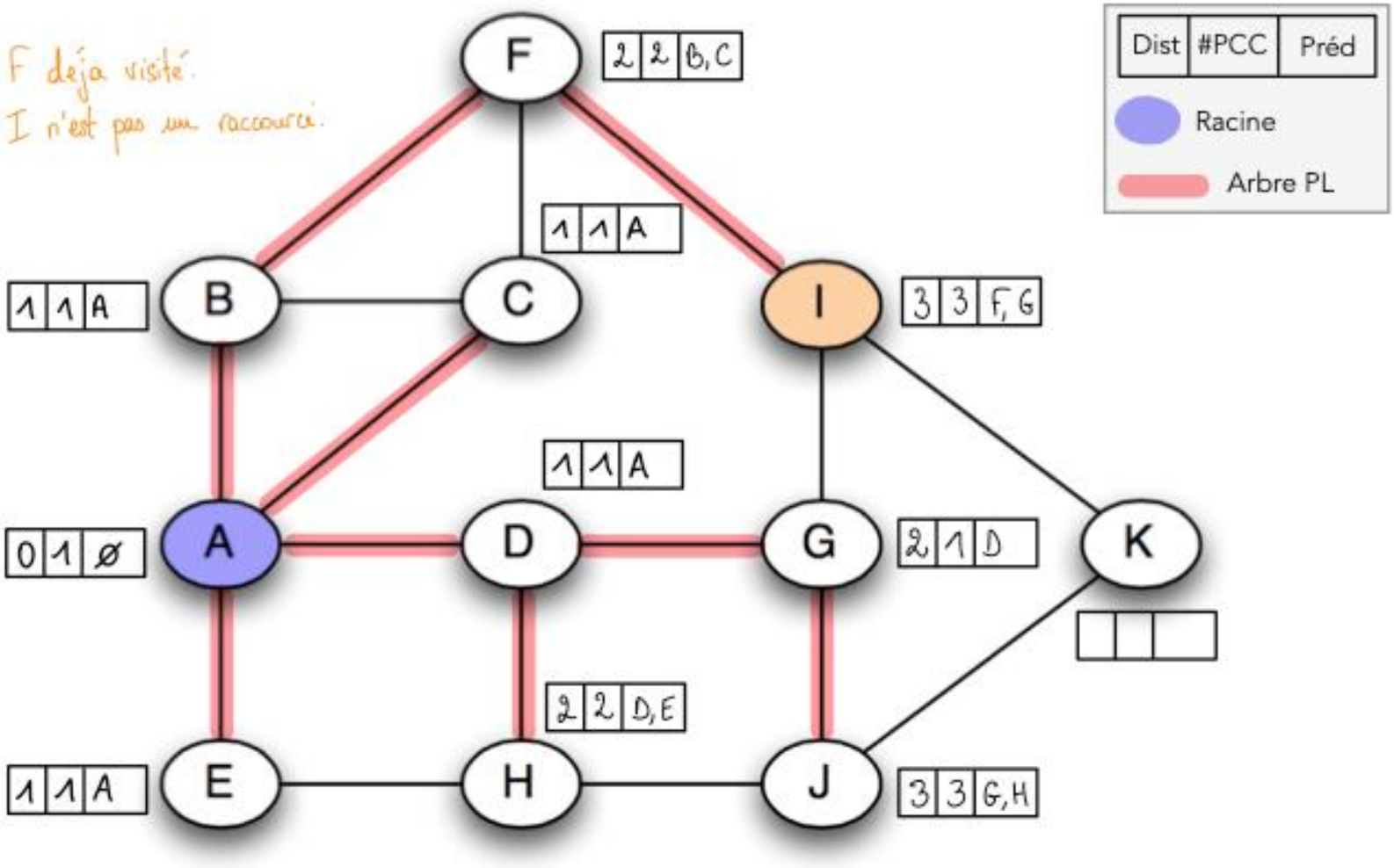
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
3 3 F, G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
2 2 D, E
E
1 1 A
H
J
3 1 G
3
Déjà visité
dist(J) = dist(H)+1 ?
OUI !

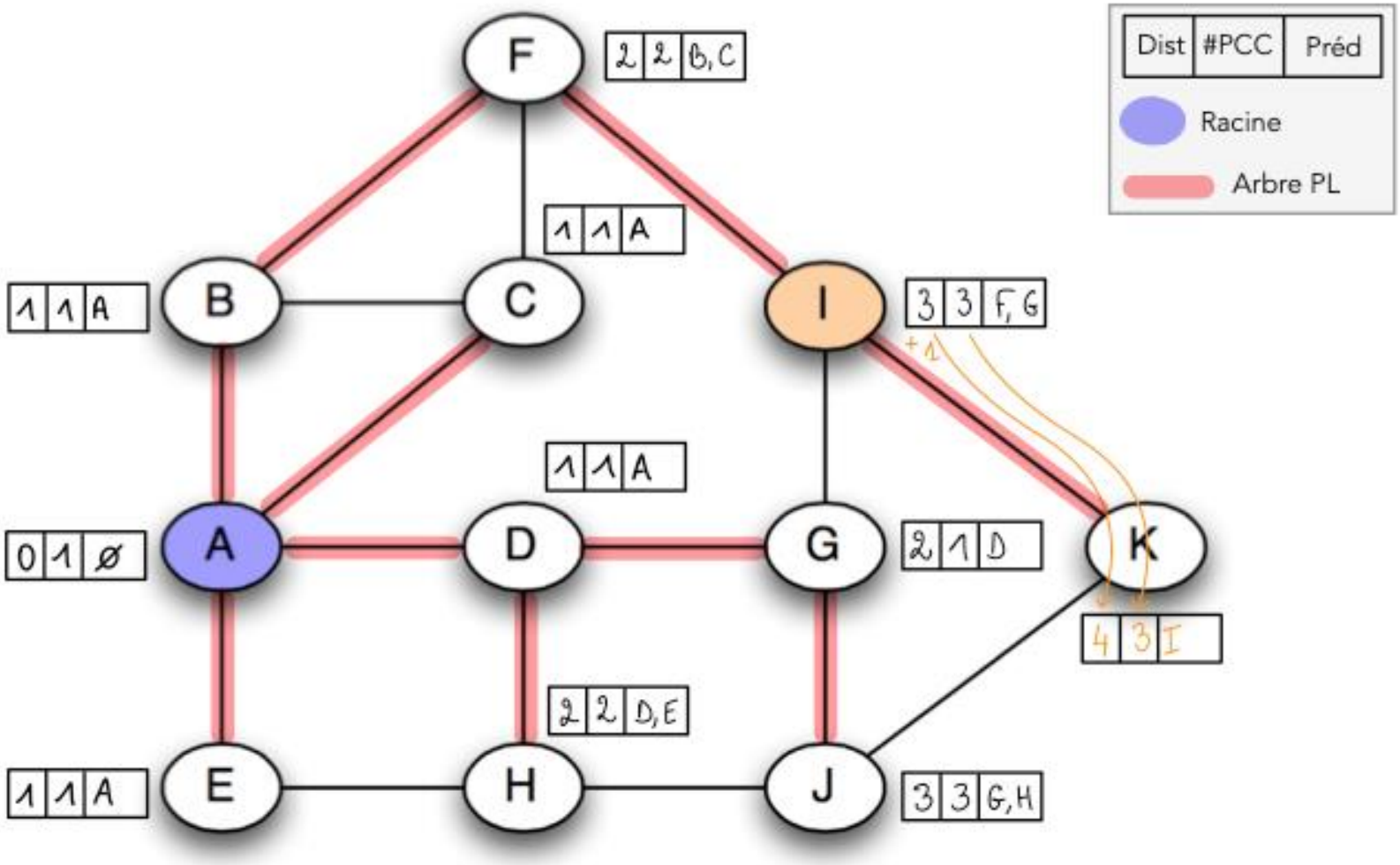
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
3 3 F, G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
2 2 D, E
E
1 1 A
H
J
3 3 G, H
Déjà visité
dist(J) ? = dist(H) + 1
OUI !

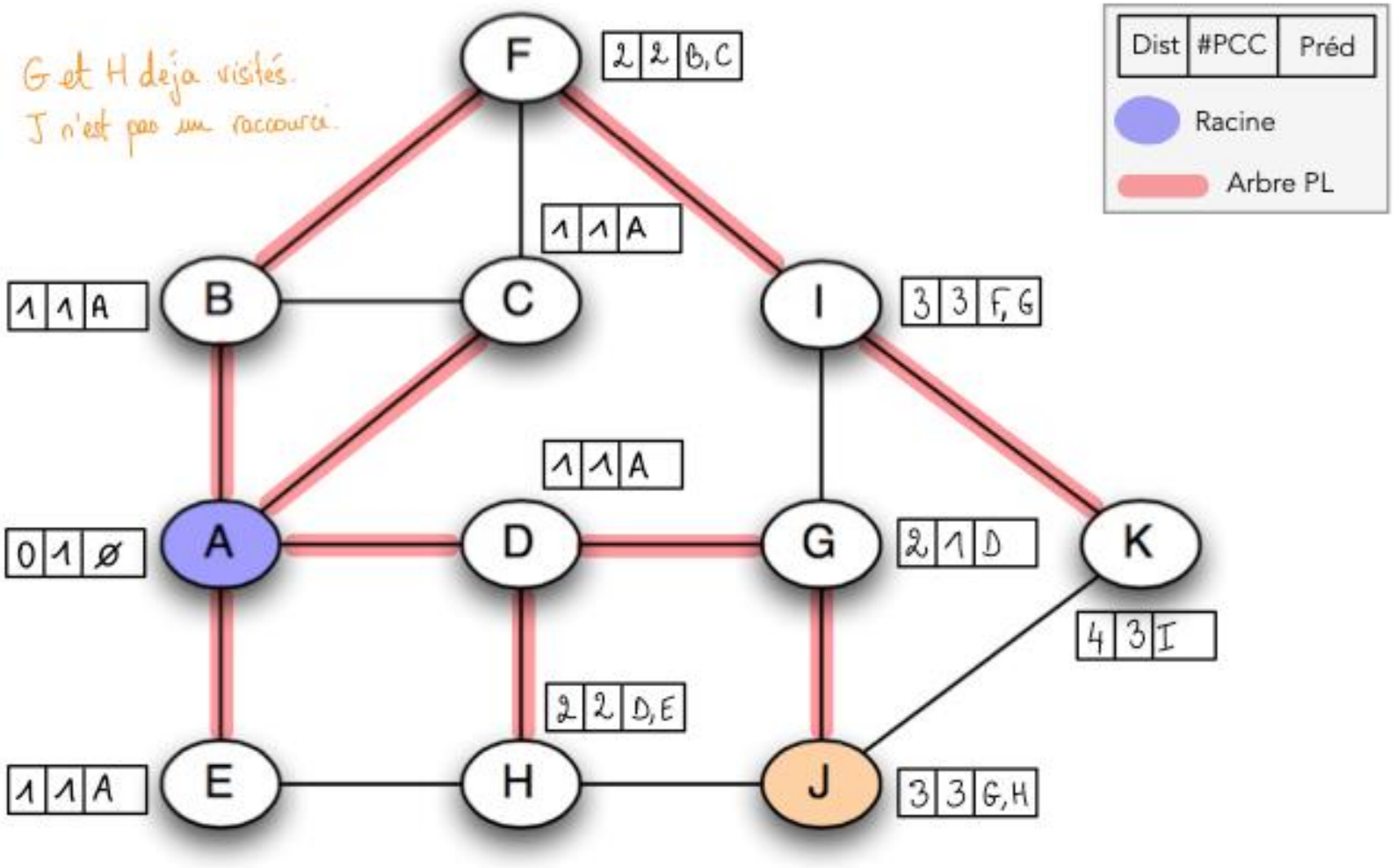
Un exemple


F déjà visité.
I n'est pas un raccourci.
Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
B
1 1 A
C
1 1 A
I
3 3 F,G
A
0 1 ∅
D
1 1 A
G
2 1 D
K
E
1 1 A
H
2 2 D,E
J
3 3 G,H

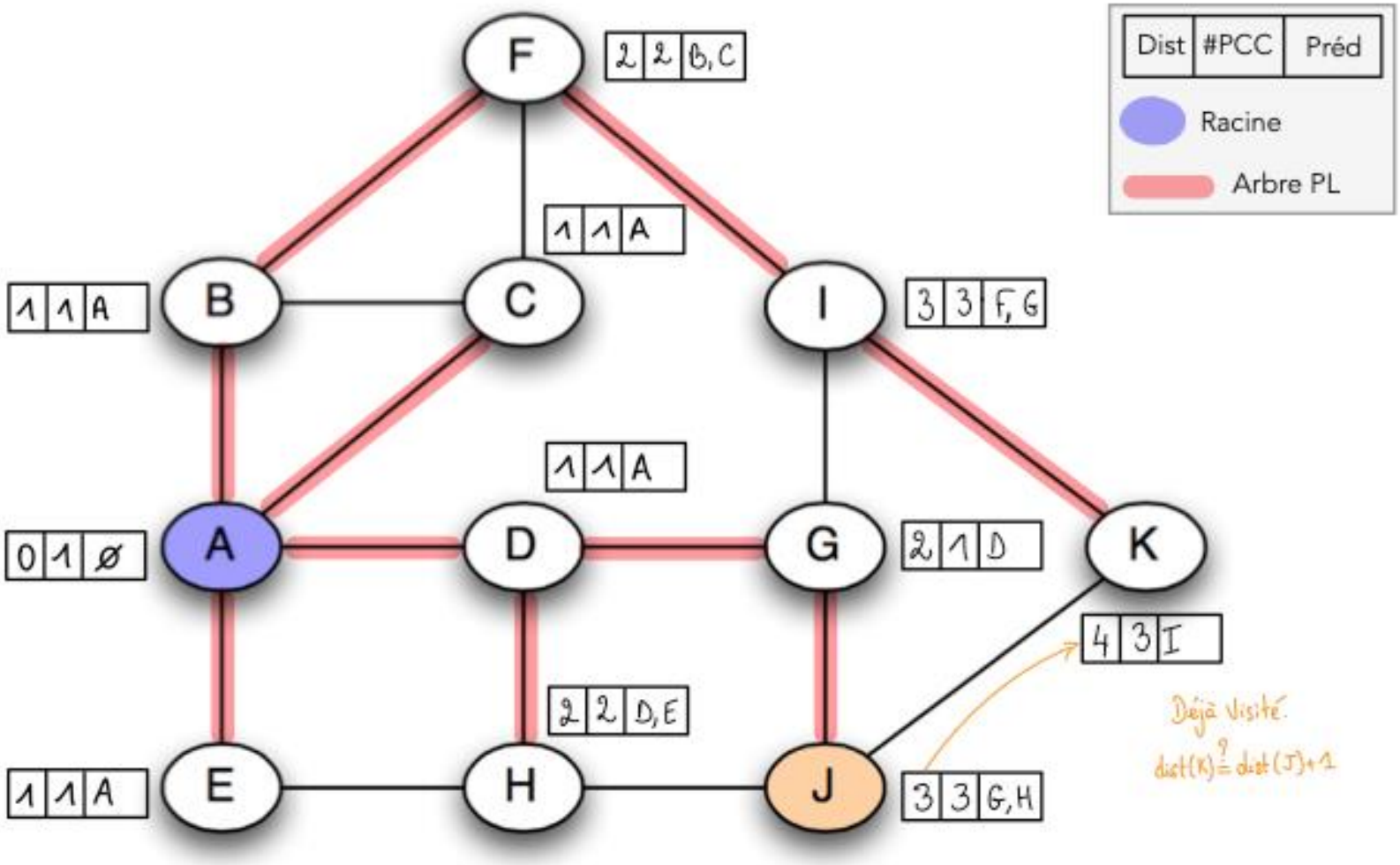
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
C
I
3 3 F, G
+1
1 1 A
A
D
G
2 1 D
K
4 3 I
0 1 ∅
2 2 D, E
1 1 A
E
H
J
3 3 G, H

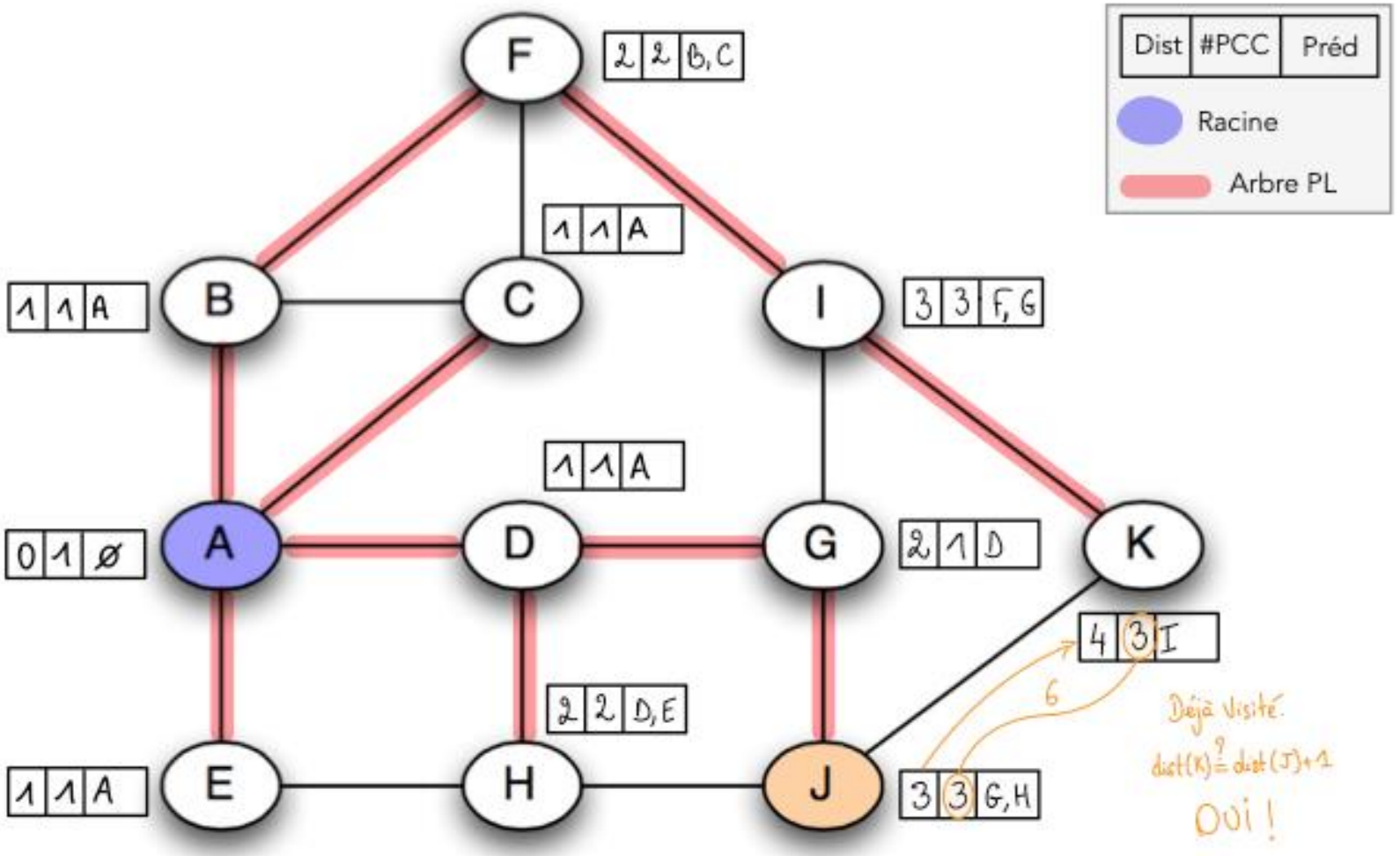
Un exemple


G et H deja visités.
J n'est pas un raccourci.
Dist | #PCC | Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
1 1 A
C
I
3 3 F,G
1 1 A
0 1 ∅
A
D
G
2 1 D
K
4 3 I
2 2 D,E
E
1 1 A
H
J
3 3 G,H

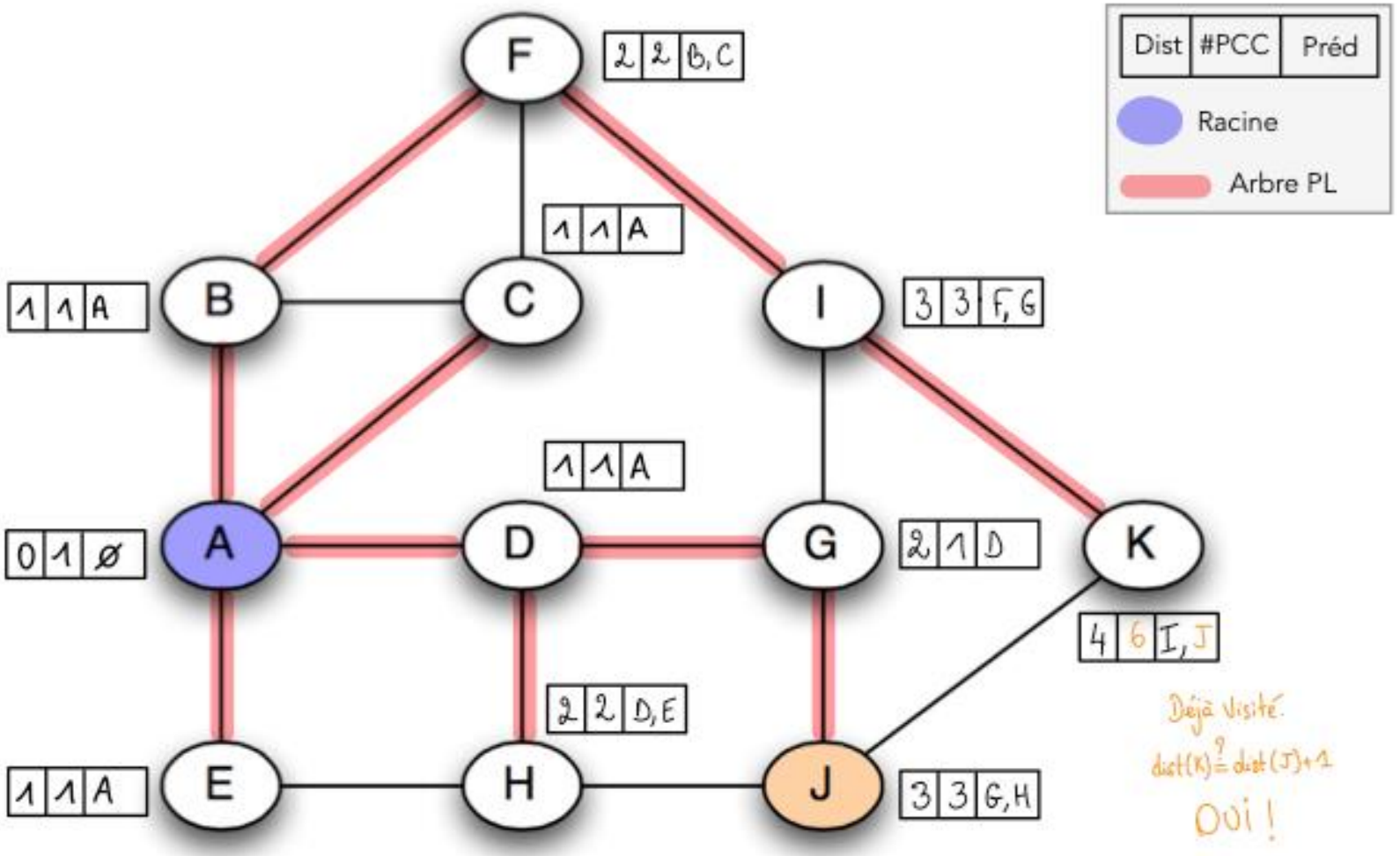
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
3 3 F, G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
4 3 I
Déjà visité.
dist(K) ? dist(J)+1
2 2 D, E
E
1 1 A
H
J
3 3 G, H

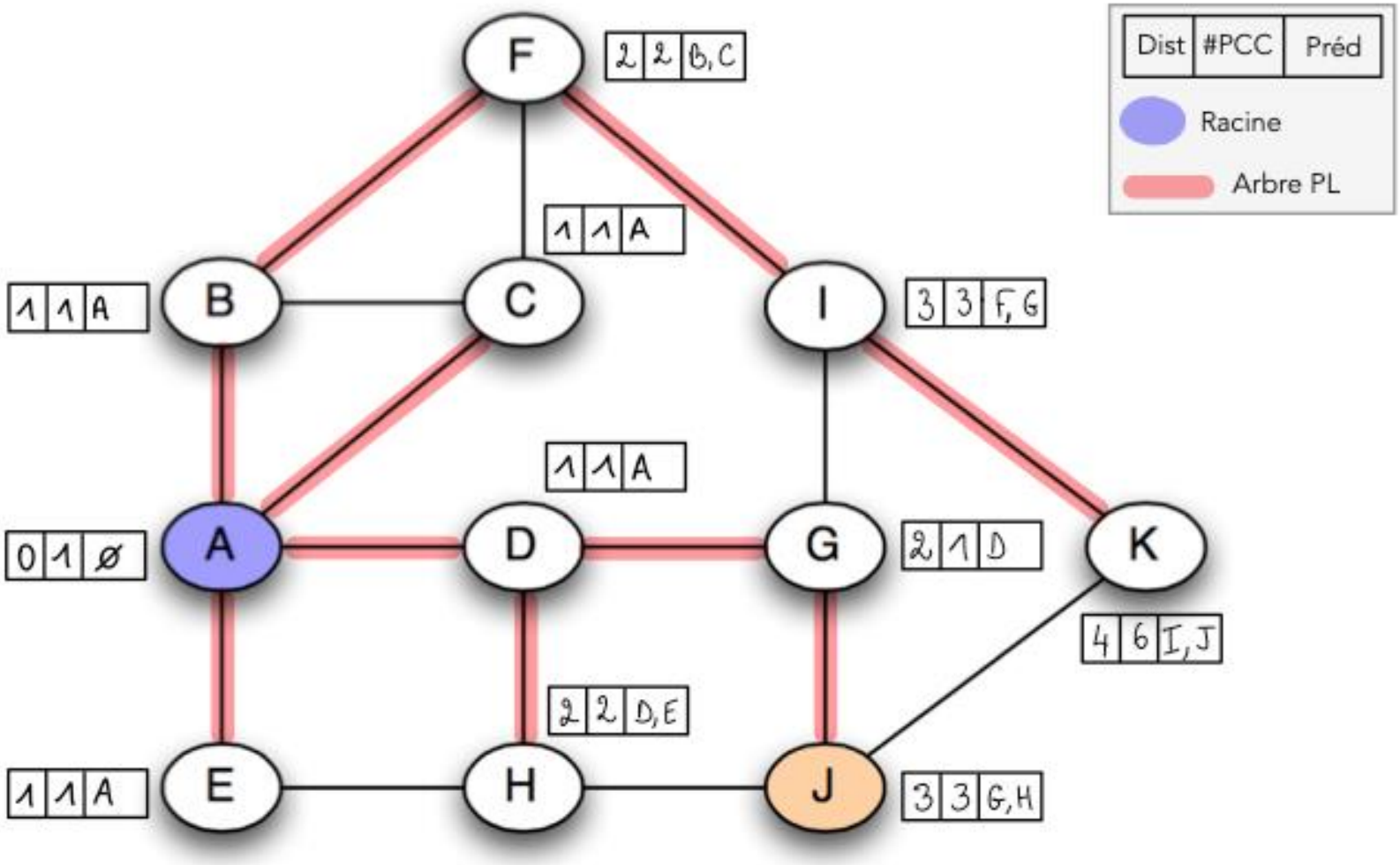
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
3 3 F, G
1 1 A
0 1 ∅
A
D
G
2 1 D
K
4 3 I
6
Déjà visité.
dist(K) ? dist(J)+1
Oui !
2 2 D, E
E
1 1 A
H
J
3 3 G, H

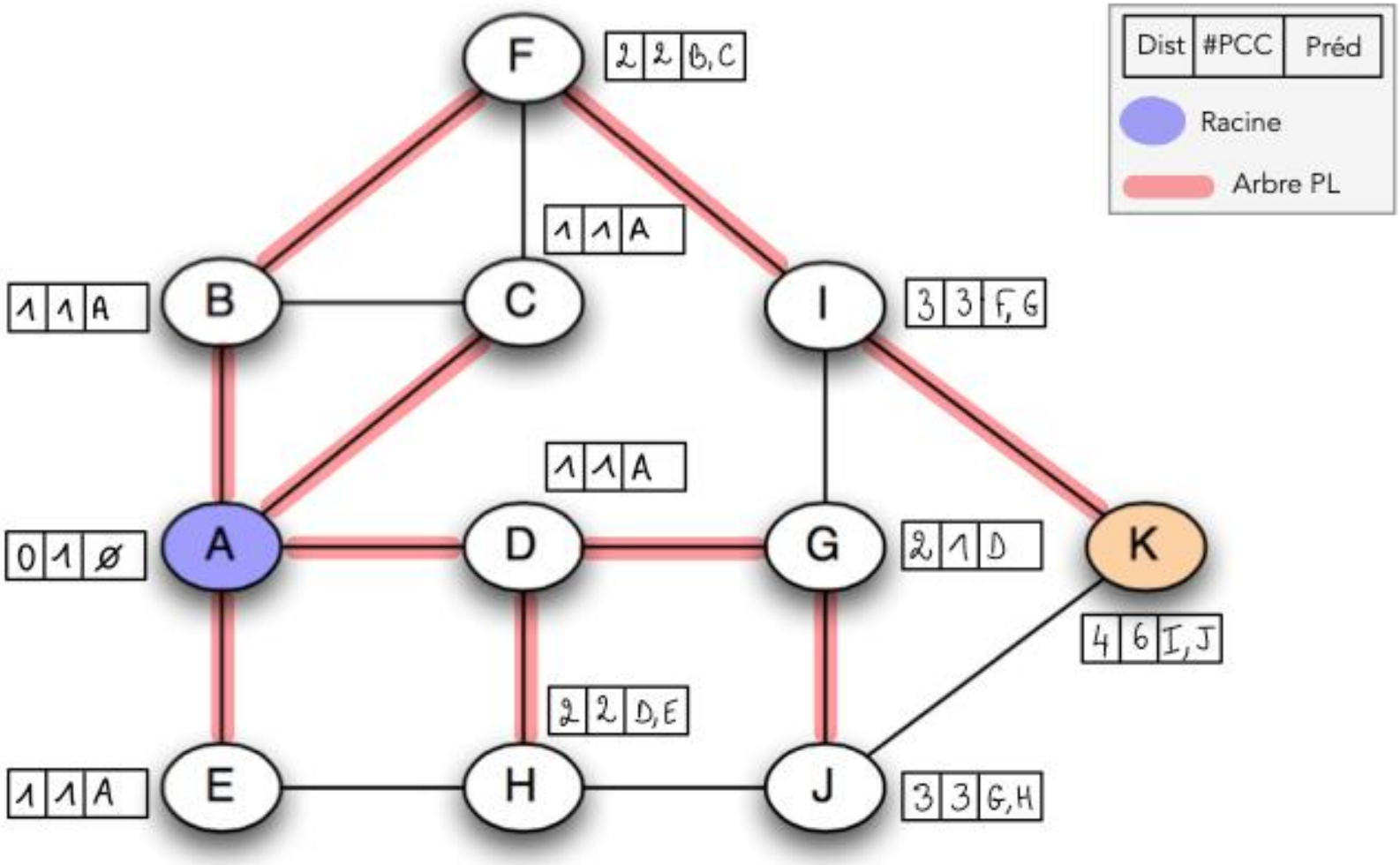
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B, C
1 1 A
B
1 1 A
C
I
3 3 F, G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
4 6 I, J
2 2 D, E
E
1 1 A
H
J
3 3 G, H
Déjà visité.
dist(K) ? dist(J)+1
Oui !

Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
1 1 A
C
I
3 3 F,G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
4 6 I,J
2 2 D,E
E
1 1 A
H
J
3 3 G,H

Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
1 1 A
C
I
3 3 F,G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
4 6 I,J
2 2 D,E
E
1 1 A
H
J
3 3 G,H

Comment calculer la centralité d'intermédiarité ?

- En appliquant n fois l'algorithme précédent, on obtient, pour chaque paire de sommets $\{u, w\}$, le nombre de PCC entre u et w (i.e. σ_{uw})

  Complexité : $0\big(n(n+m)\big)$.

- Comment faire pour calculer $\sigma_{uw}(v)$?

  Pour tout $u \in V$, on pose :

$$\sigma_u(v) = \sum_{w \in V} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

  On a donc :

$$C_b(v) = \sum_{u \in V} \sigma_u(v)$$

Rappels :

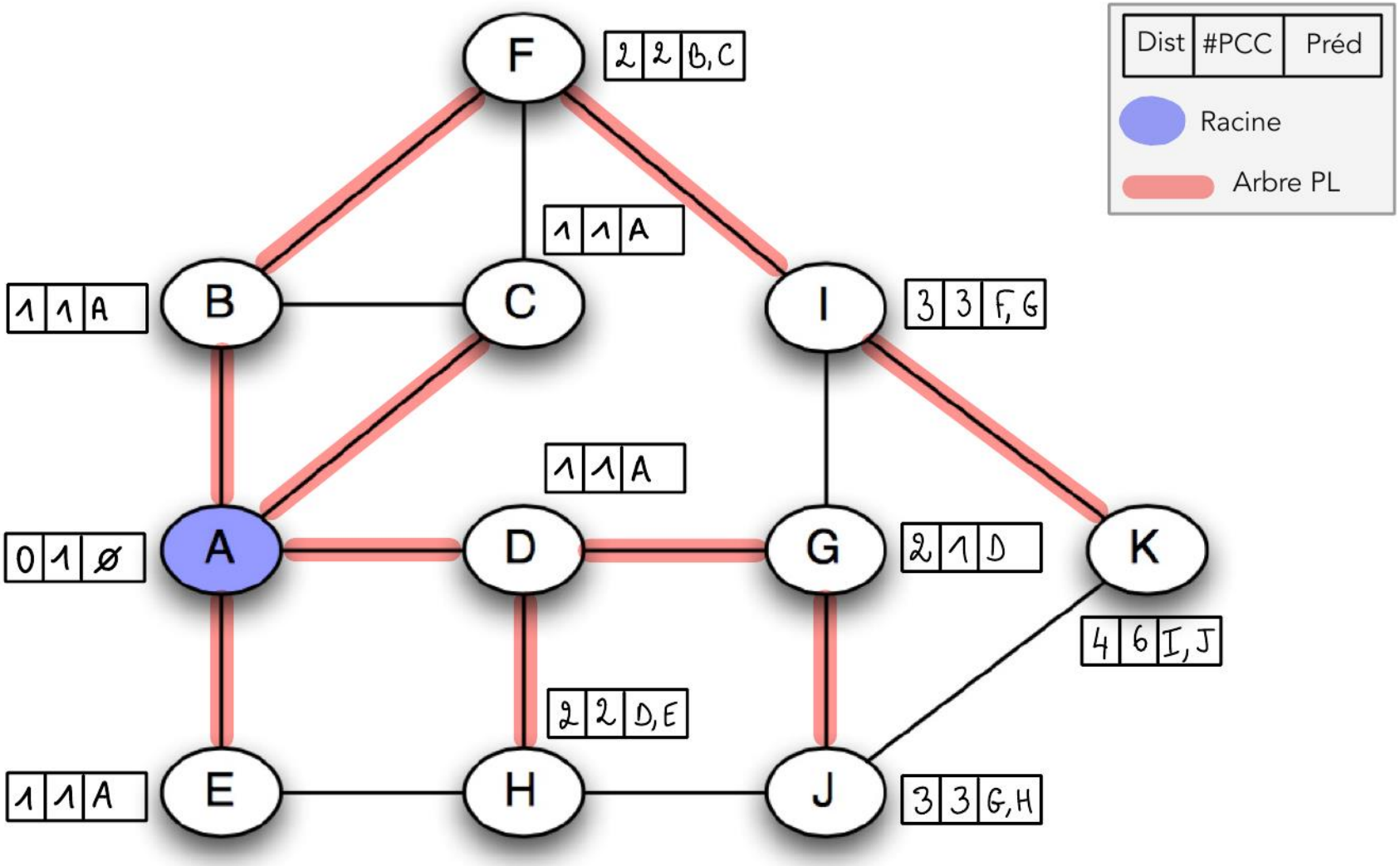
Si $u = w$, alors $\sigma_{uw} = 1$
Si $v \in \{u, w\}$, alors $\sigma_{uw}(v) = 0$

$$C_b(v) = \sum_{u \neq v \neq w} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

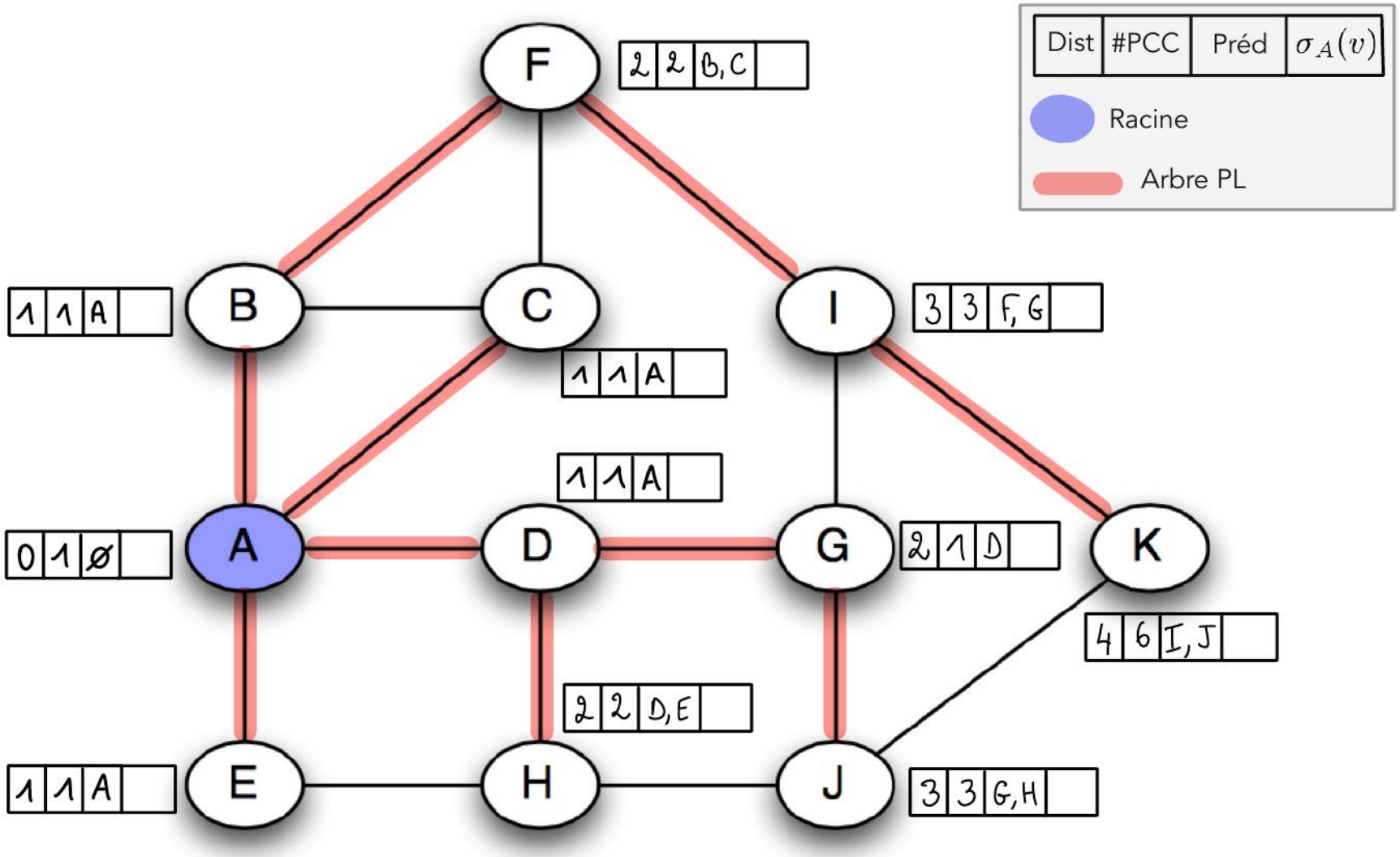
Brandes montre que $\sigma_u(v)$ est calculable facilement et qu'il n'est pas utile de savoir quels sont exactement les PCC impliqués.

Voyons l'intuition sur un exemple.

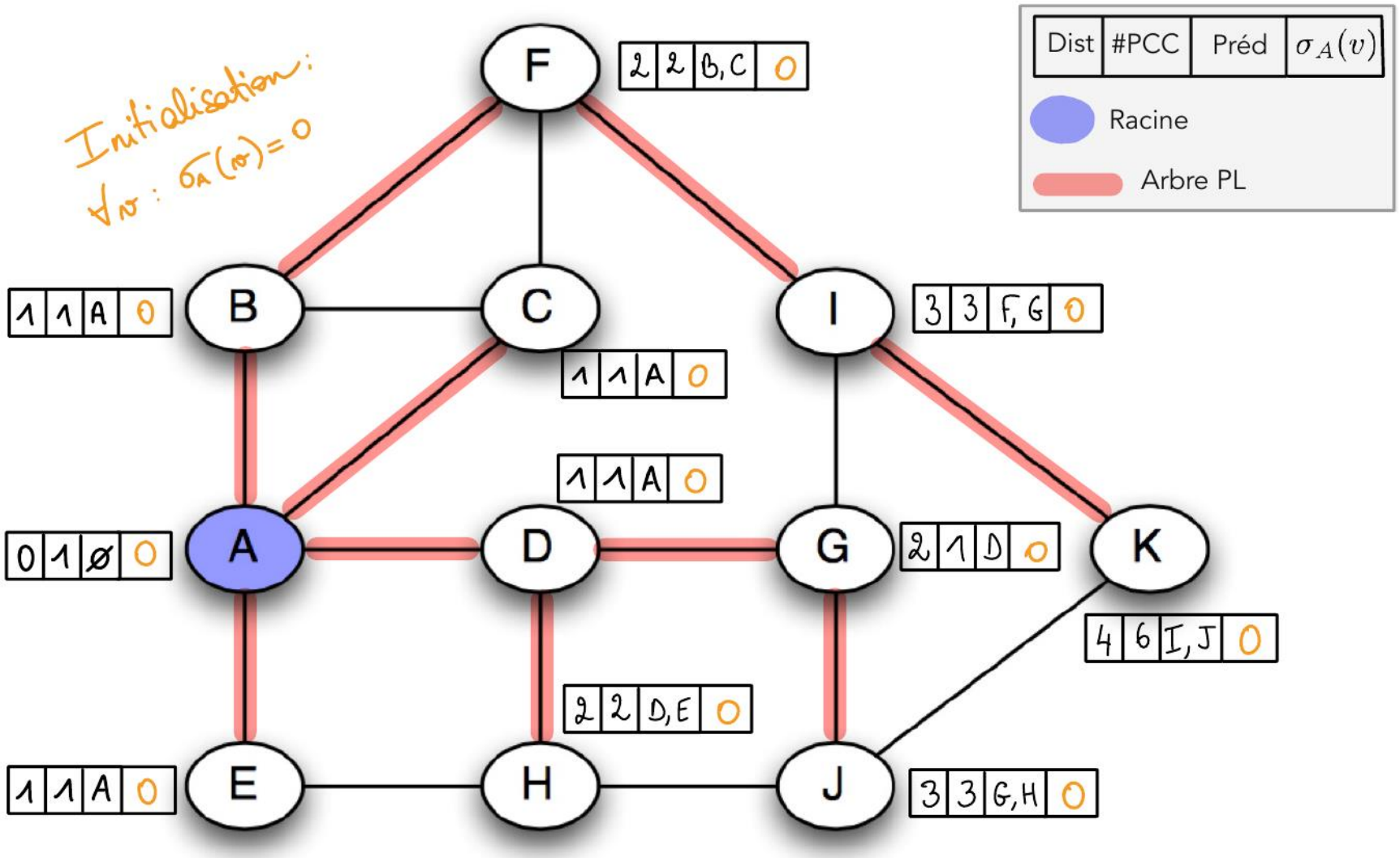
Un exemple


Dist
#PCC
Préd
Racine
Arbre PL
F
2 2 B,C
1 1 A
B
1 1 A
C
I
3 3 F,G
1 1 A
A
0 1 ∅
D
G
2 1 D
K
4 6 I,J
2 2 D,E
E
1 1 A
H
J
3 3 G,H

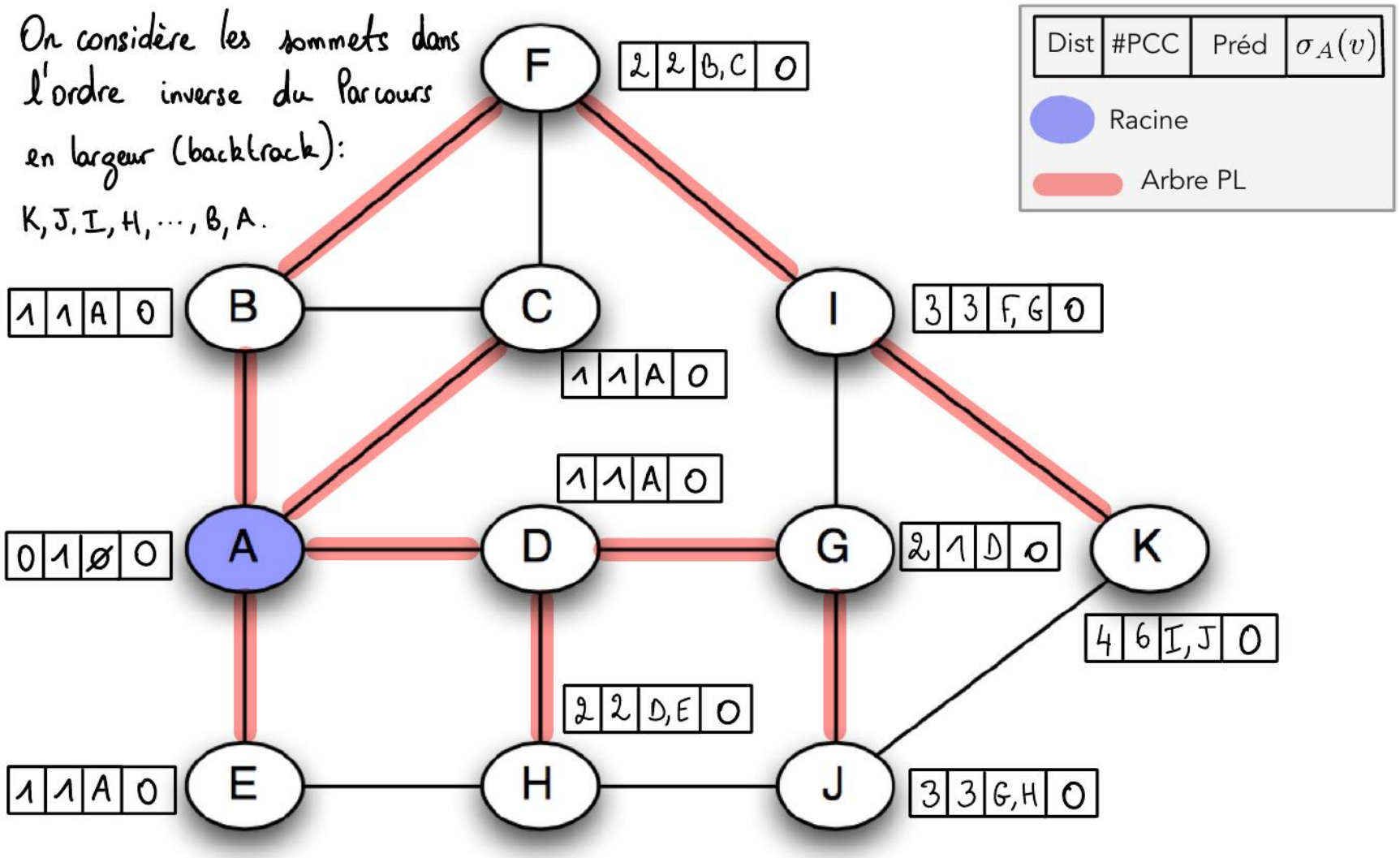
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C
B
1 1 A
C
1 1 A
I
3 3 F,G
A
0 1 ∅
1 1 A
D
2 2 D,E
G
2 1 D
K
4 6 I,J
E
1 1 A
H
J
3 3 G,H

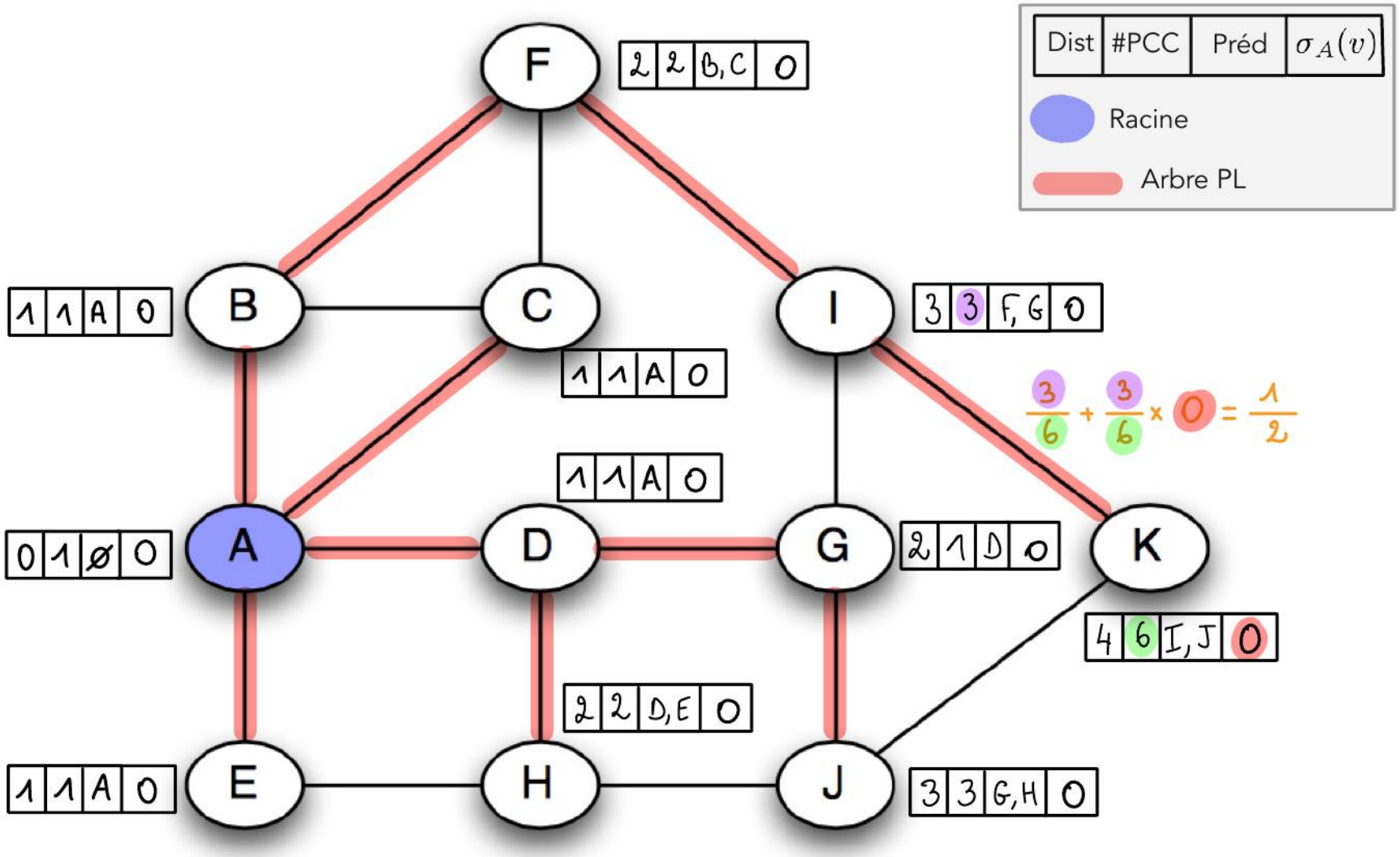
Un exemple


Initialisation :
$\forall v : \sigma_A(v) = 0$
Dist | #PCC | Préd | $\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 0
A
0 1 ∅ 0
D
1 1 A 0
G
2 1 D 0
K
4 6 I,J 0
E
1 1 A 0
H
2 2 D,E 0
J
3 3 G,H 0

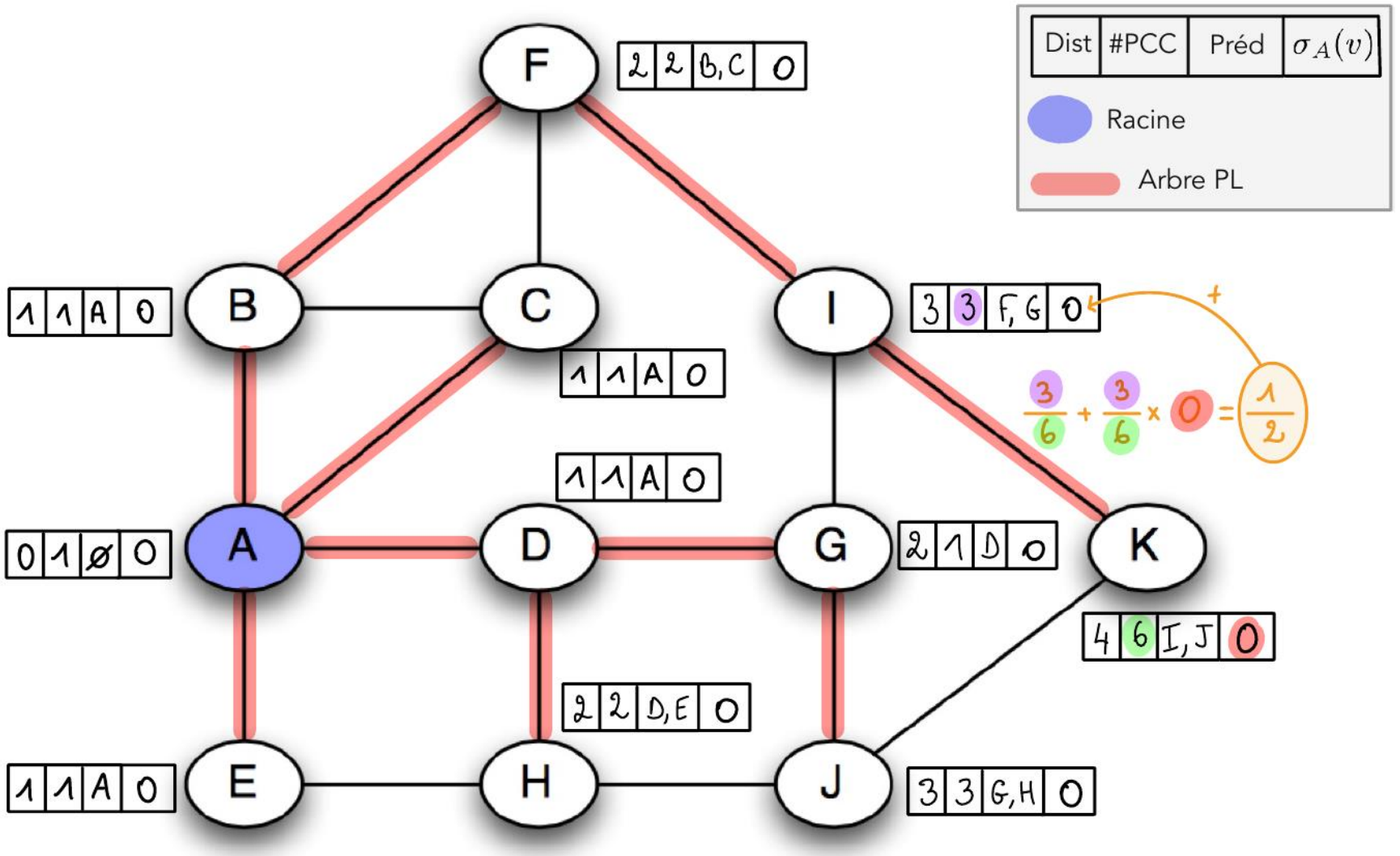
Un exemple


On considère les sommets dans l'ordre inverse du Parcours en largeur (backtrack):
K, J, I, H, ⋯, B, A.
Dist | #PCC | Préd | $\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 0
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 0
K
4 6 I,J 0
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 0

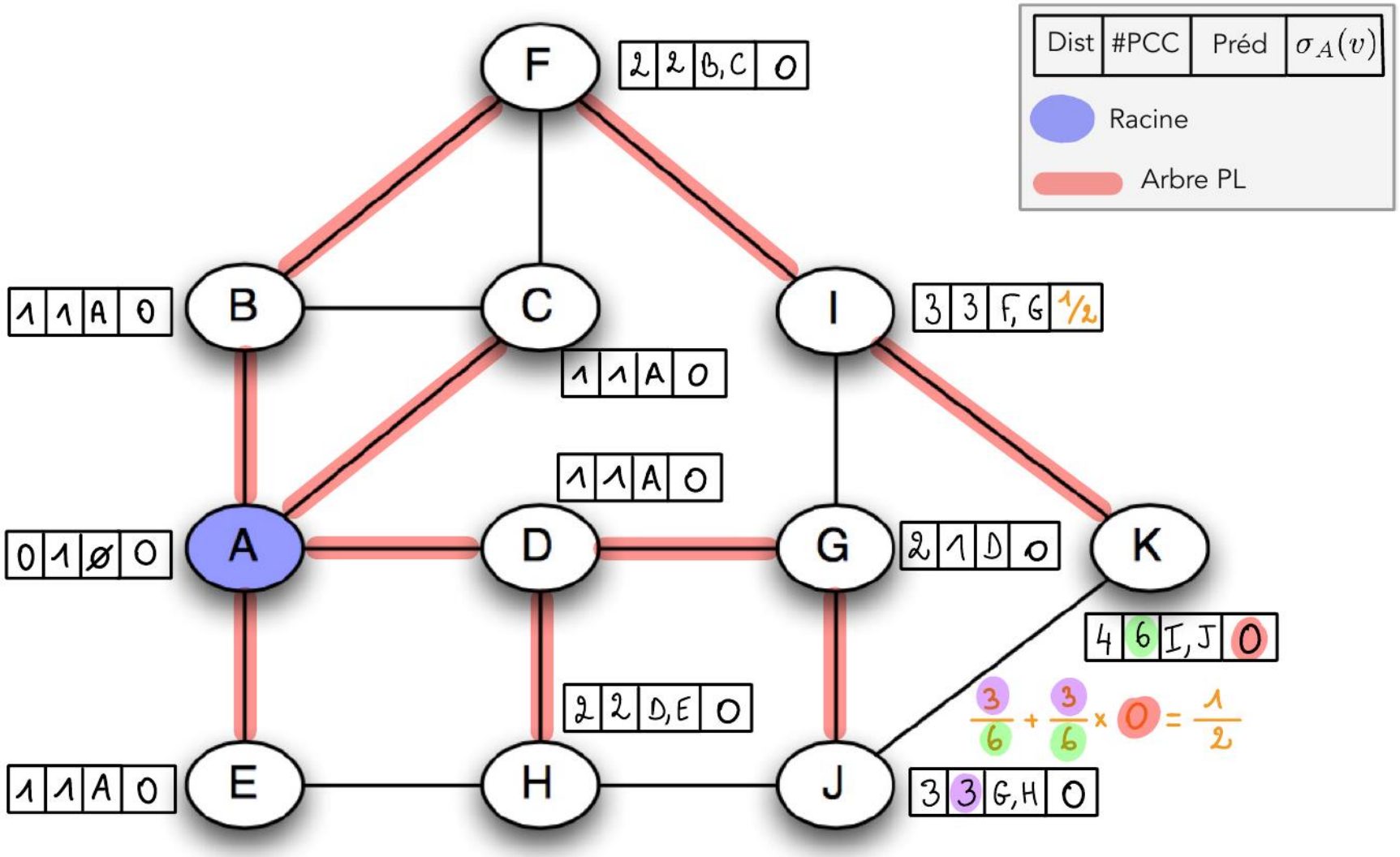
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 0
3/6 + 3/6 x 0 = 1/2
1 1 A 0
A
0 1 ø 0
D
G
2 1 D 0
K
4 6 I,J 0
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 0

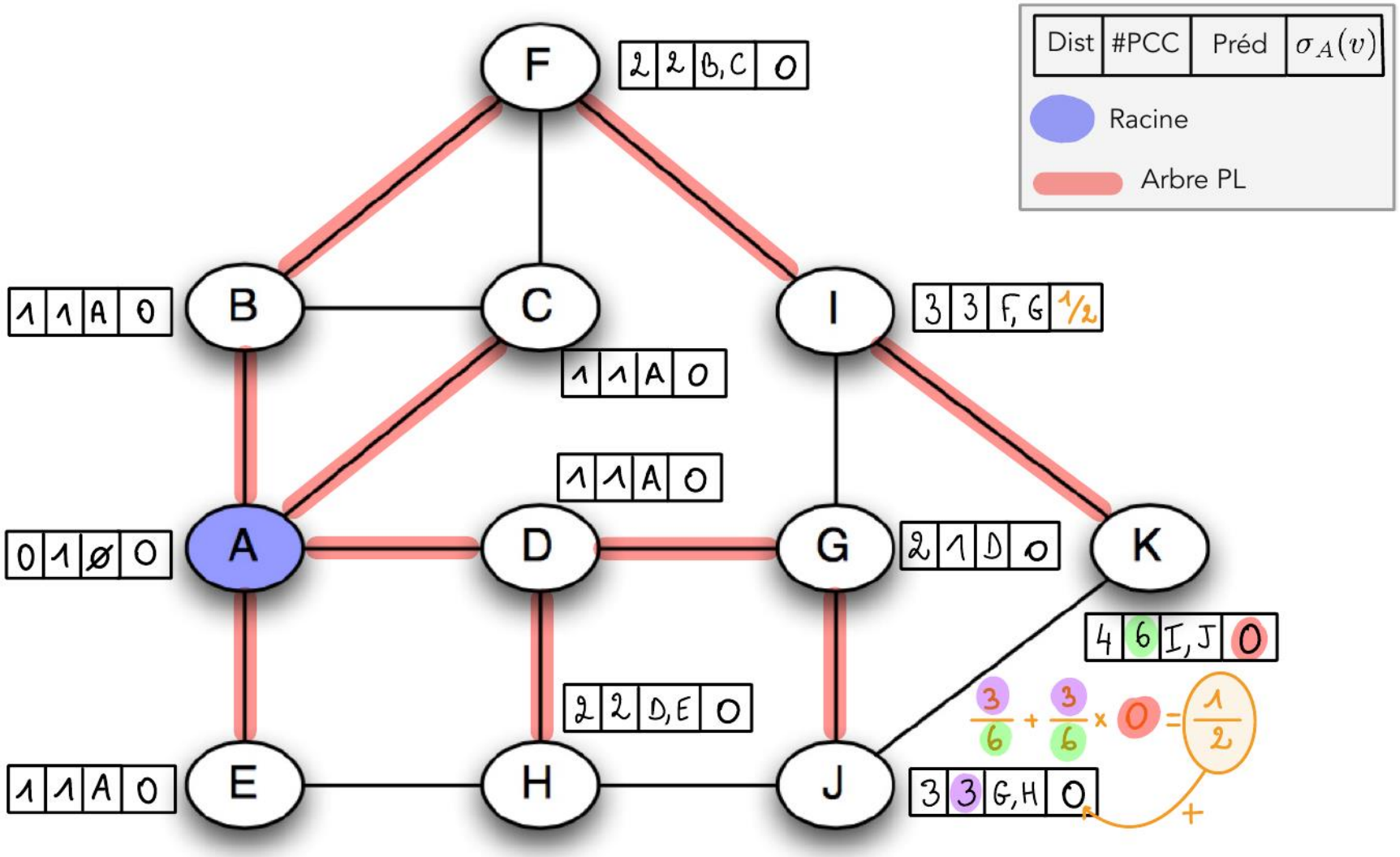
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 0
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 0
K
4 6 I,J 0
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 0

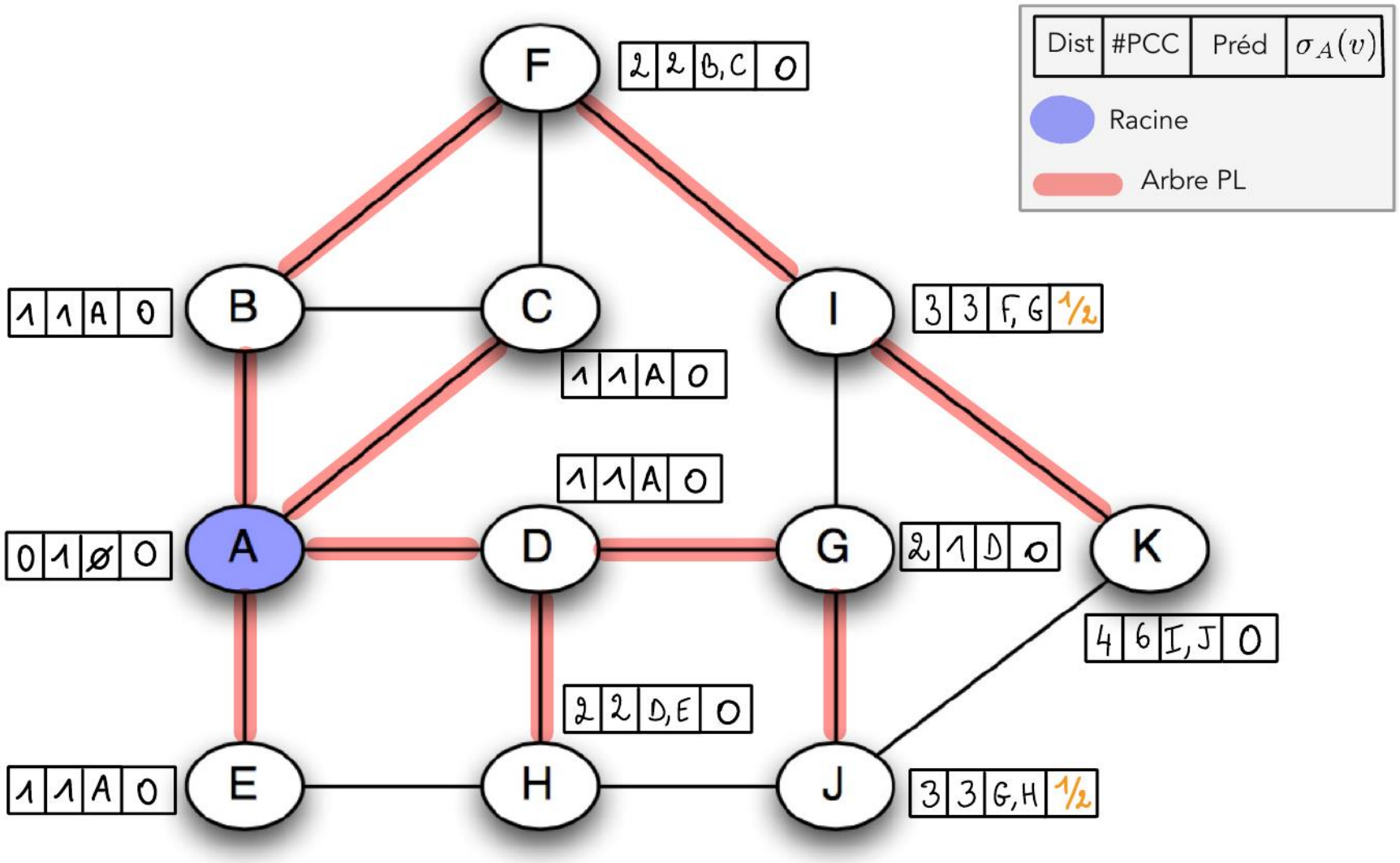
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ∅ 0
D
1 1 A 0
G
2 1 D 0
K
4 6 I,J 0
E
1 1 A 0
H
2 2 D,E 0
J
3 3 G,H 0
3/6 + 3/6 x 0 = 1/2

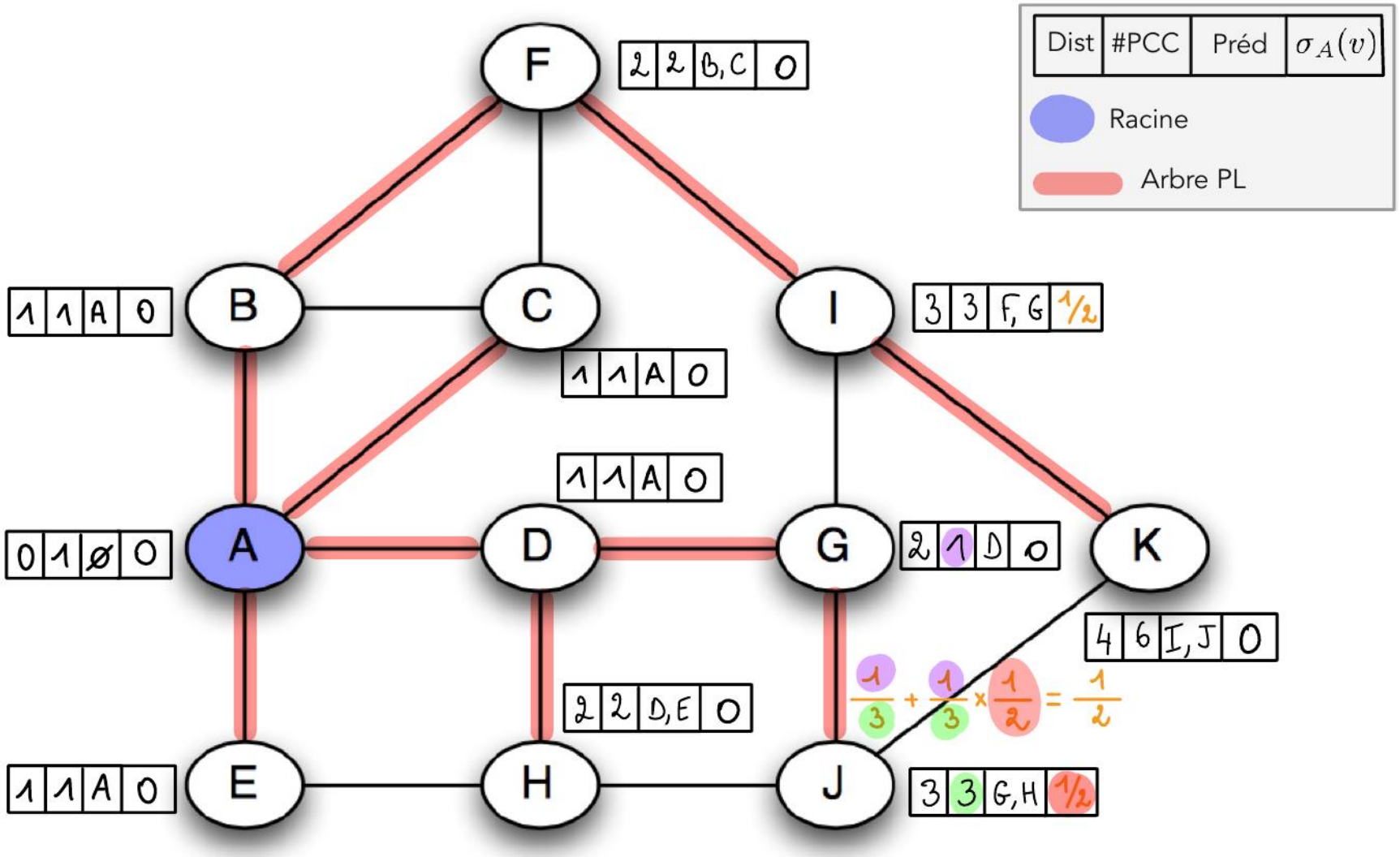
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
2 2 B,C 0
1 1 A 0
1 1 A 0
3 3 F,G 1/2
1 1 A 0
0 1 ∅ 0
2 1 D 0
4 6 I,J 0
2 2 D,E 0
1 1 A 0
3 3 G,H 0
3/6 + 3/6 × 0 = 1/2
+

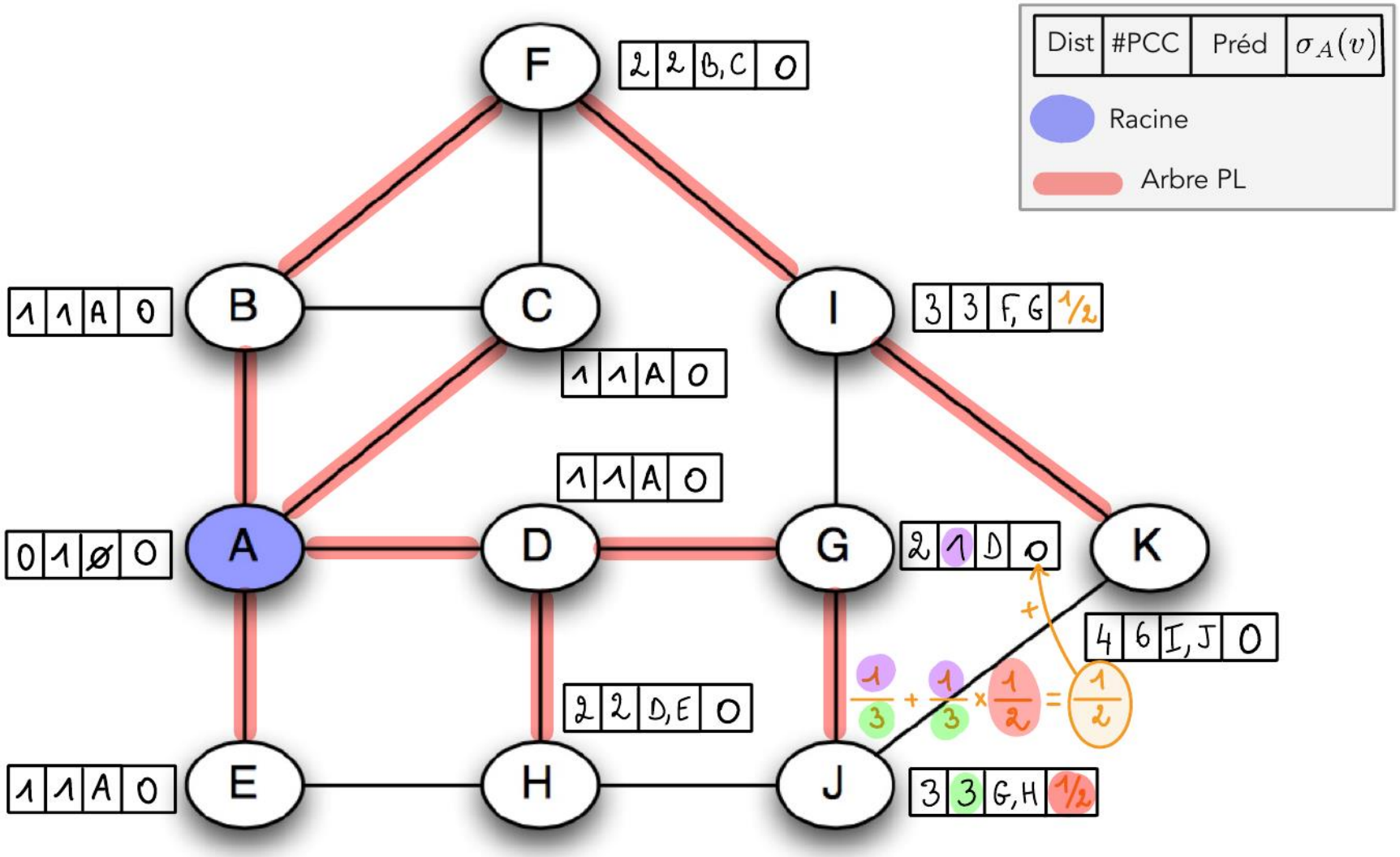
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 0
A
0 1 ∅ 0
D
2 1 D 0
G
K
4 6 I,J 0
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 1/2

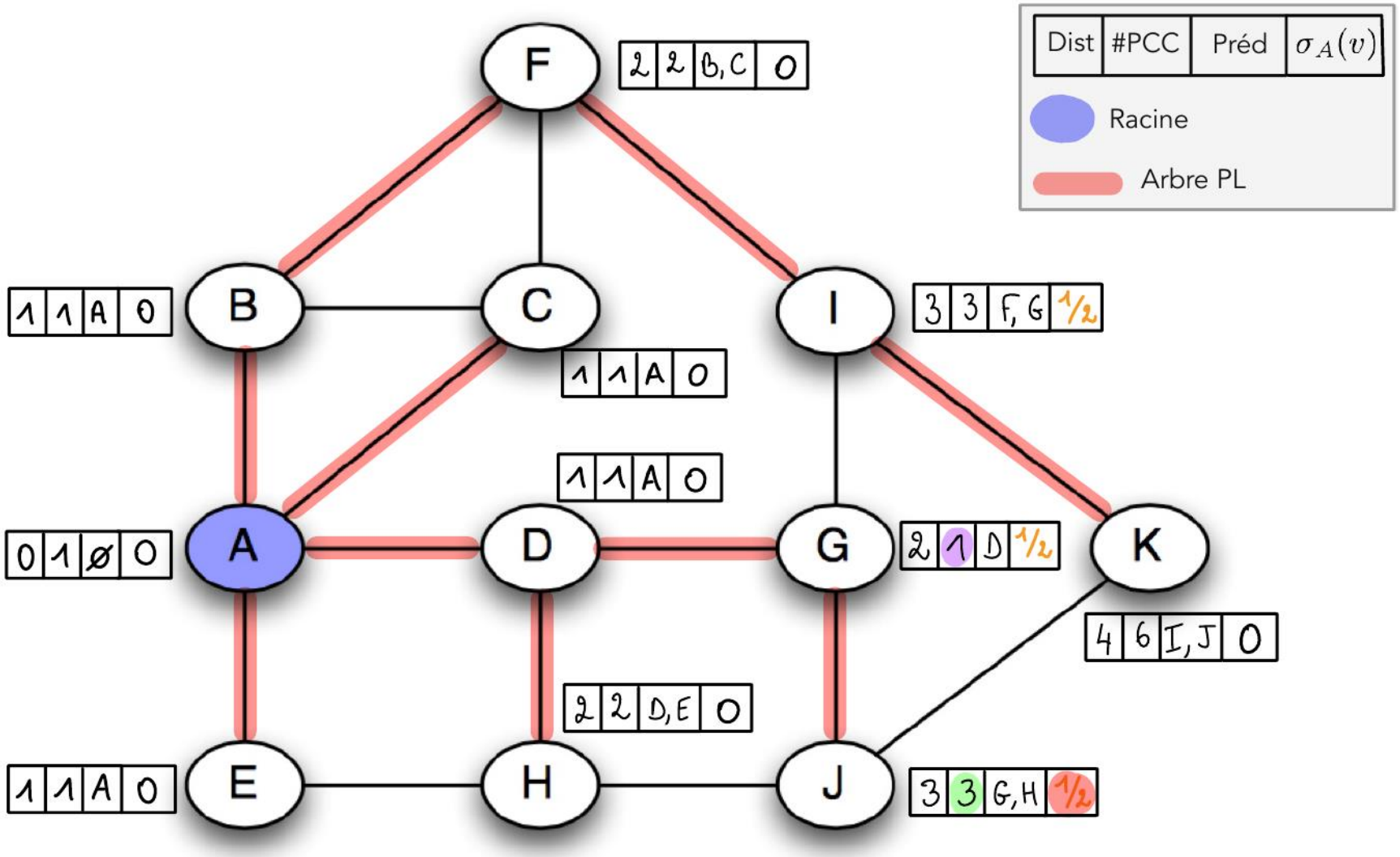
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ø 0
D
1 1 A 0
G
2 1 D 0
K
4 6 I,J 0
E
1 1 A 0
H
2 2 D,E 0
J
3 3 G,H 1/2
1/3 + 1/3 x 1/2 = 1/2

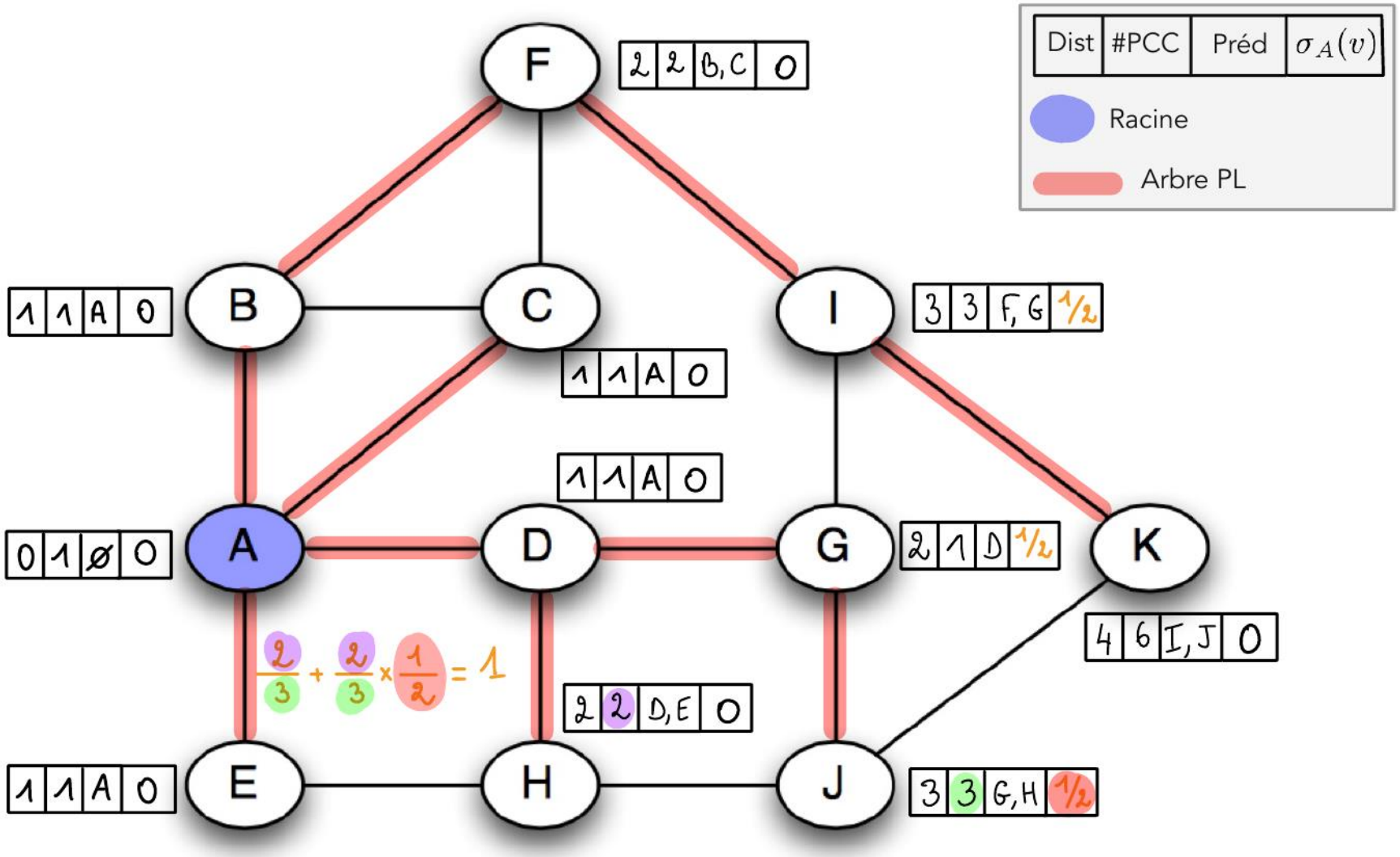
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 0
K
4 6 I,J 0
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 1/2
1/3 + 1/3 × 1/2 = 1/2
+

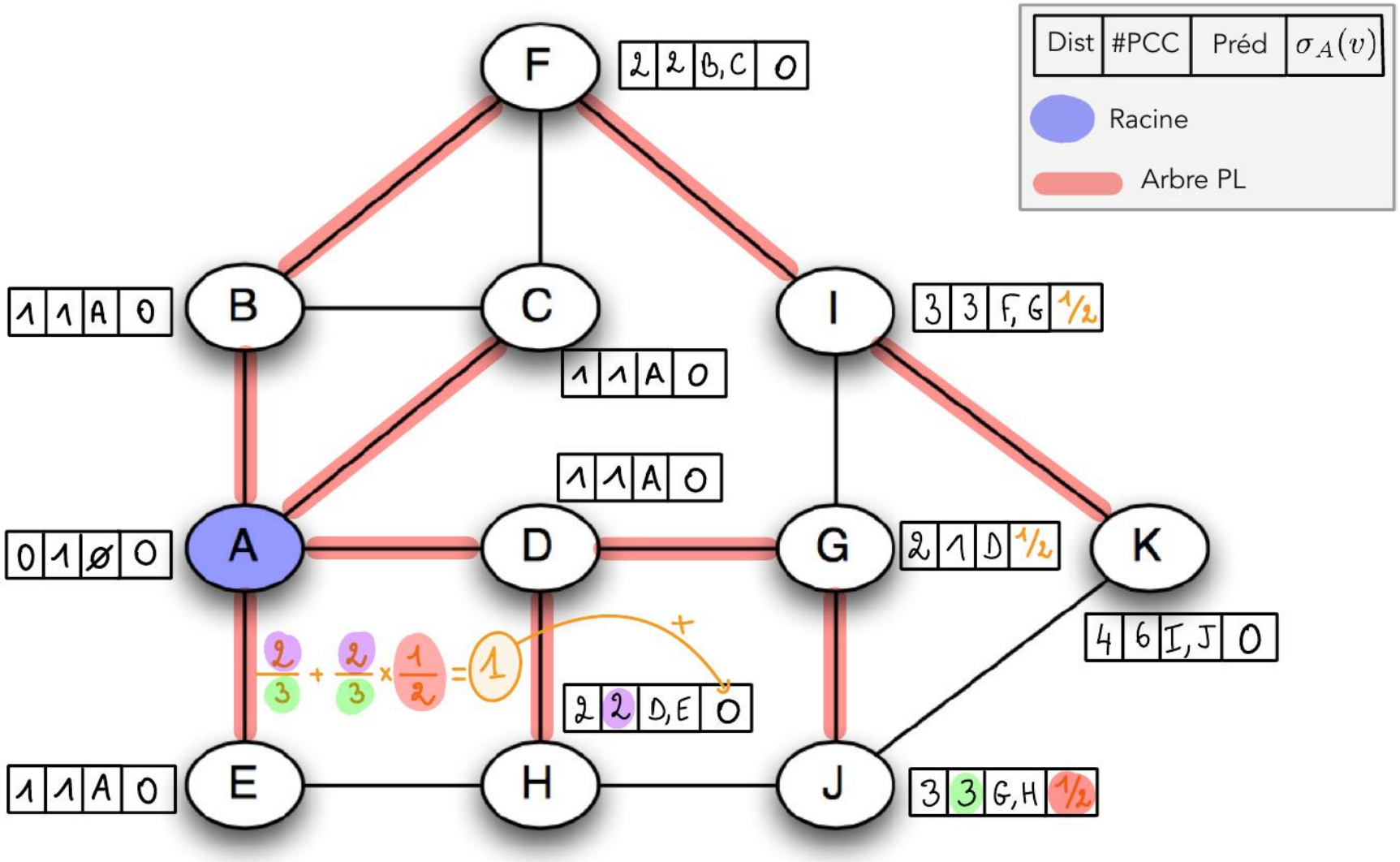
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 1/2
K
4 6 I,J 0
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 1/2

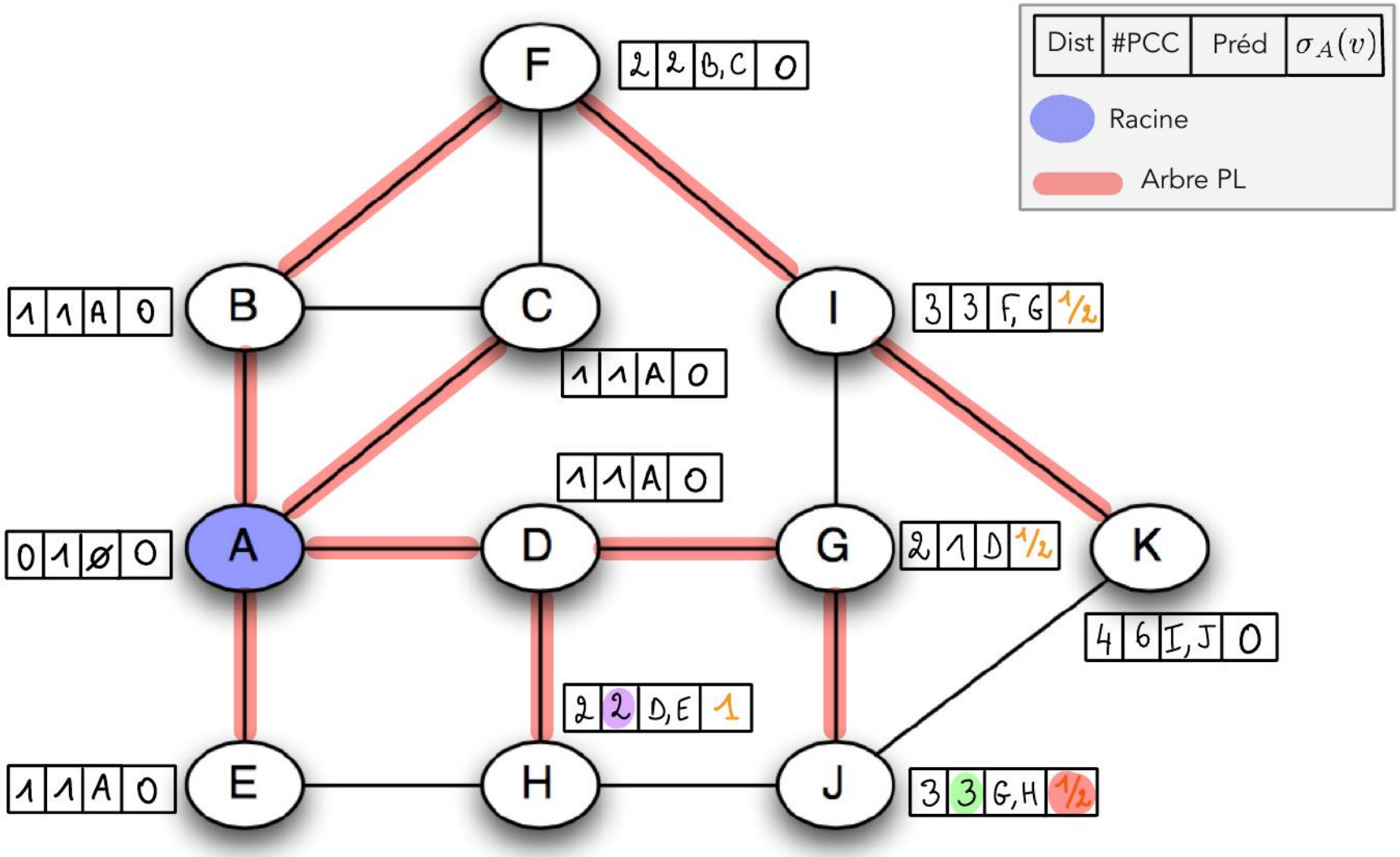
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 1/2
K
4 6 I,J 0
2/3 + 2/3 × 1/2 = 1
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 1/2

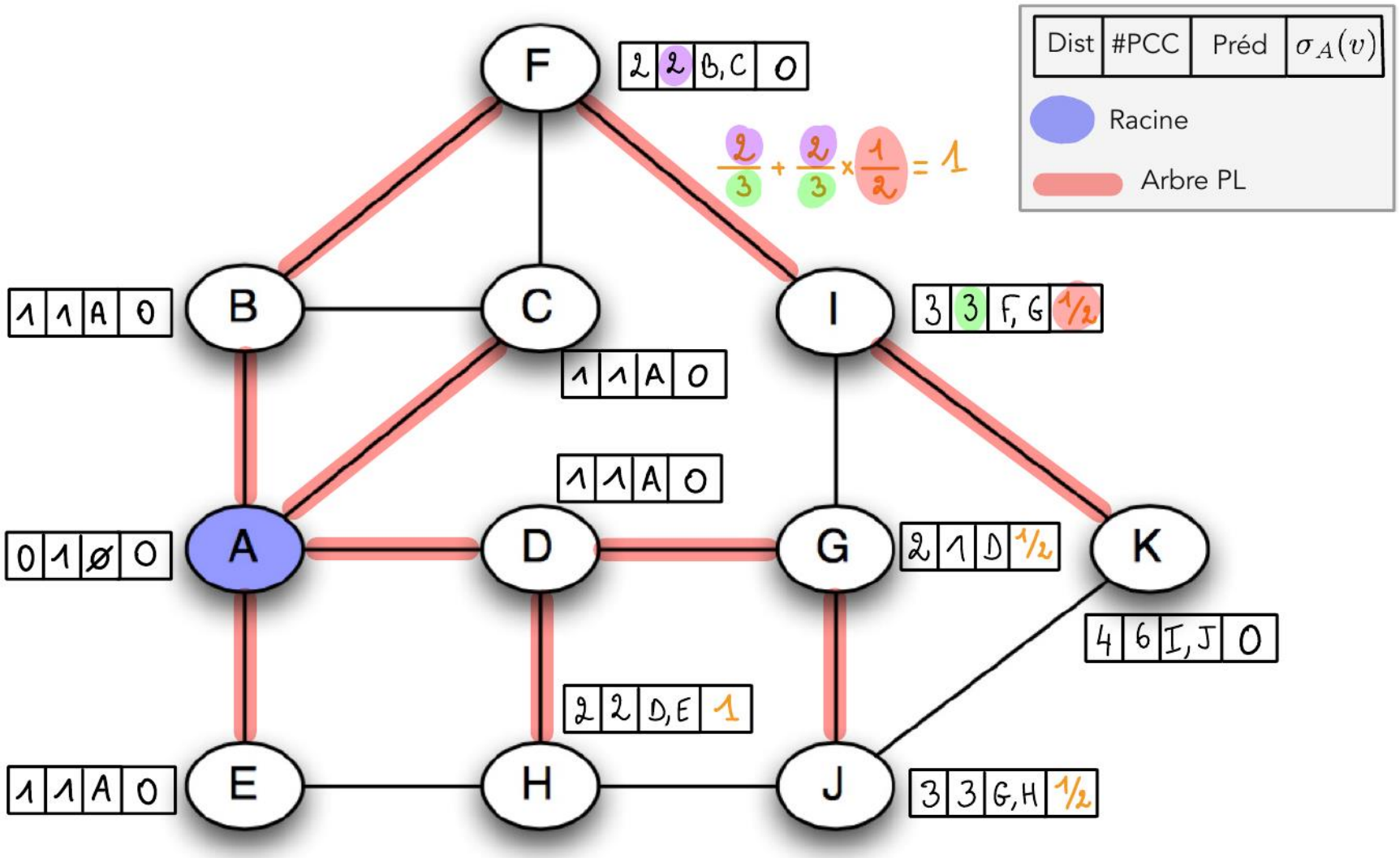
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ∅ 0
D
1 1 A 0
G
2 1 D 1/2
K
4 6 I,J 0
2/3 + 2/3 × 1/2 = 1
+
2 2 D,E 0
E
1 1 A 0
H
J
3 3 G,H 1/2

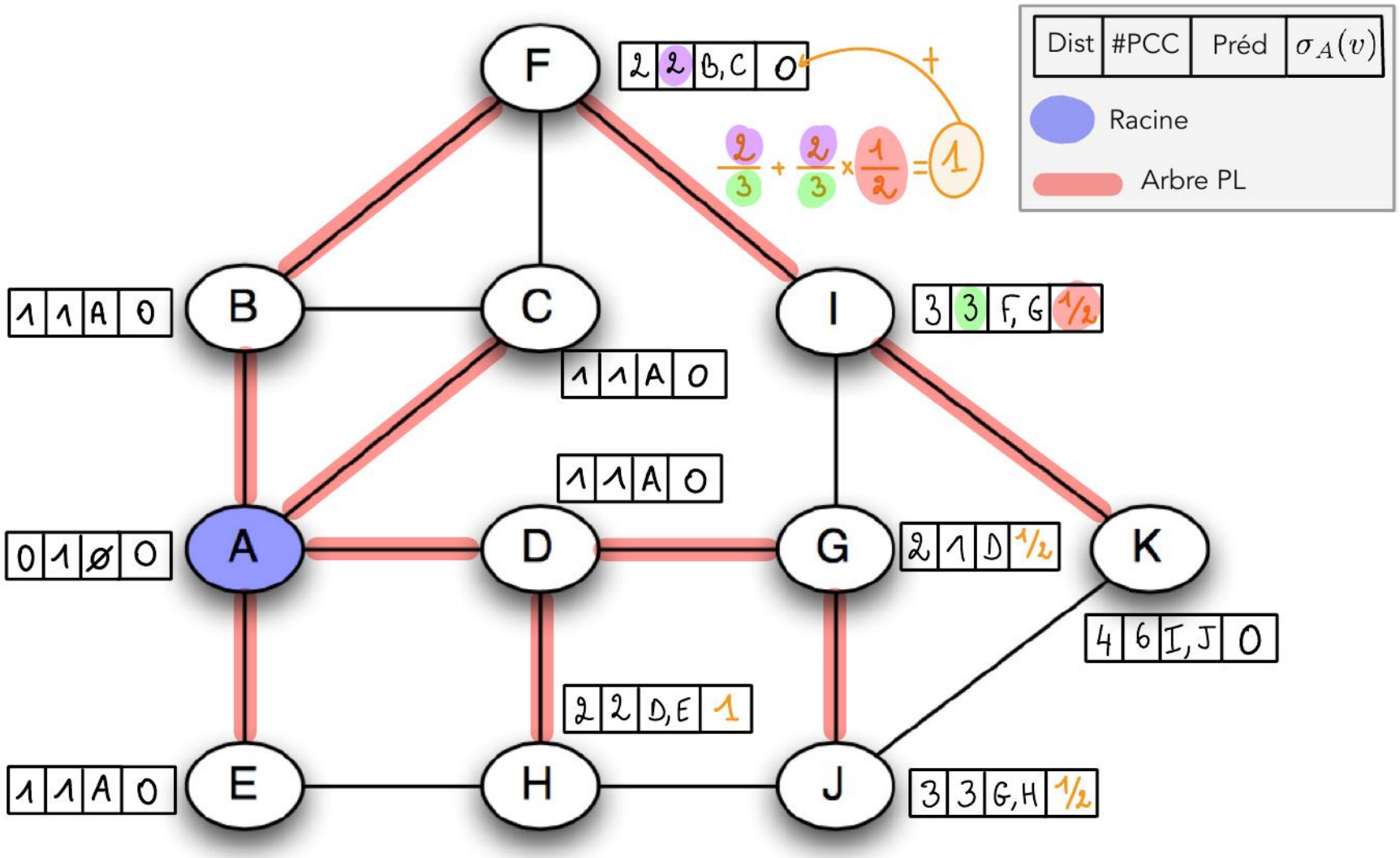
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 0
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ∅ 0
D
1 1 A 0
G
2 1 D 1/2
K
4 6 I,J 0
E
1 1 A 0
H
2 2 D,E 1
J
3 3 G,H 1/2

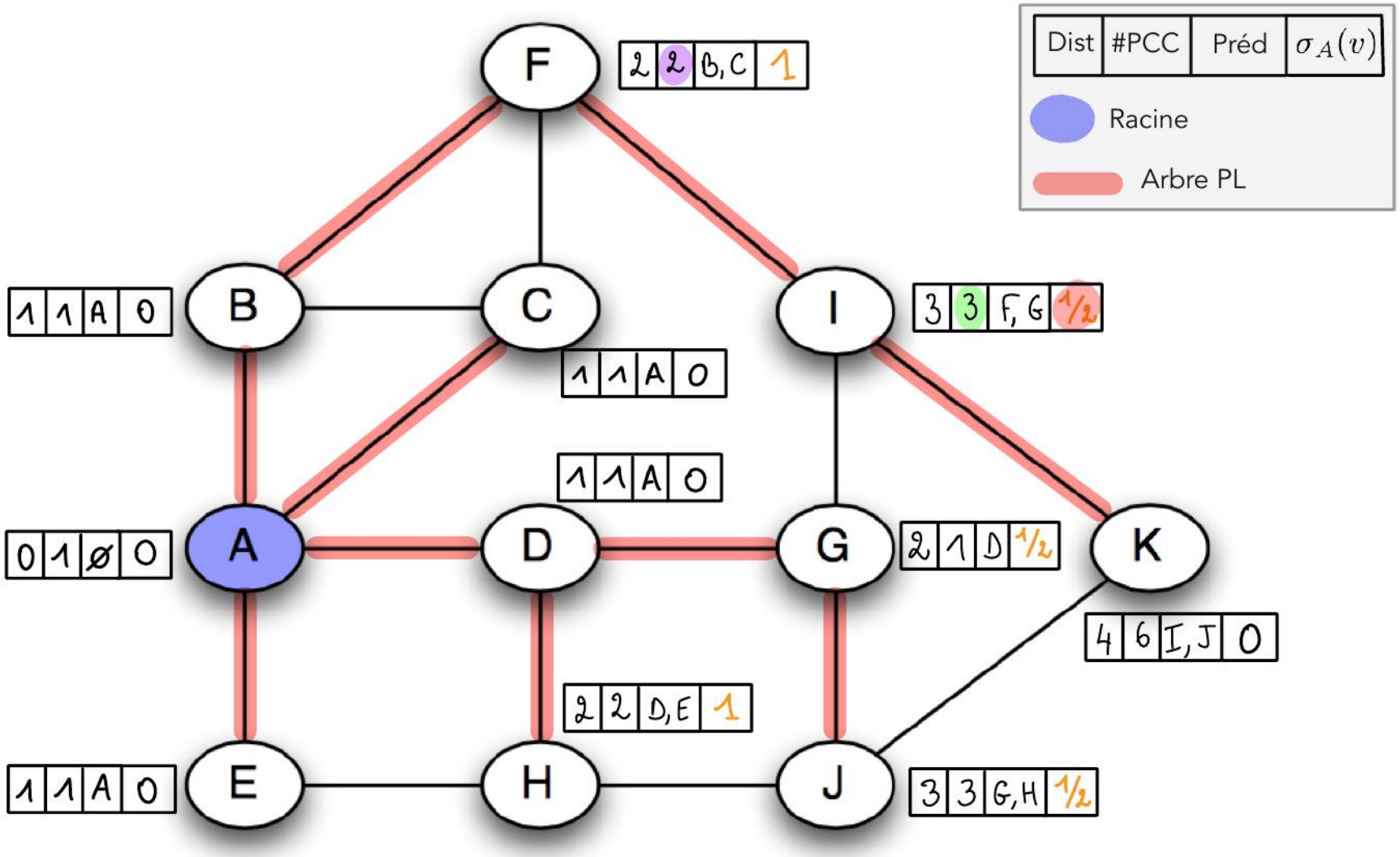
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
2 2 B,C 0
3 3 F,G 1/2
1 1 A 0
1 1 A 0
1 1 A 0
0 1 ∅ 0
2 1 D 1/2
4 6 I,J 0
2 2 D,E 1
1 1 A 0
3 3 G,H 1/2
$\frac{2}{3} + \frac{2}{3} \times \frac{1}{2} = 1$

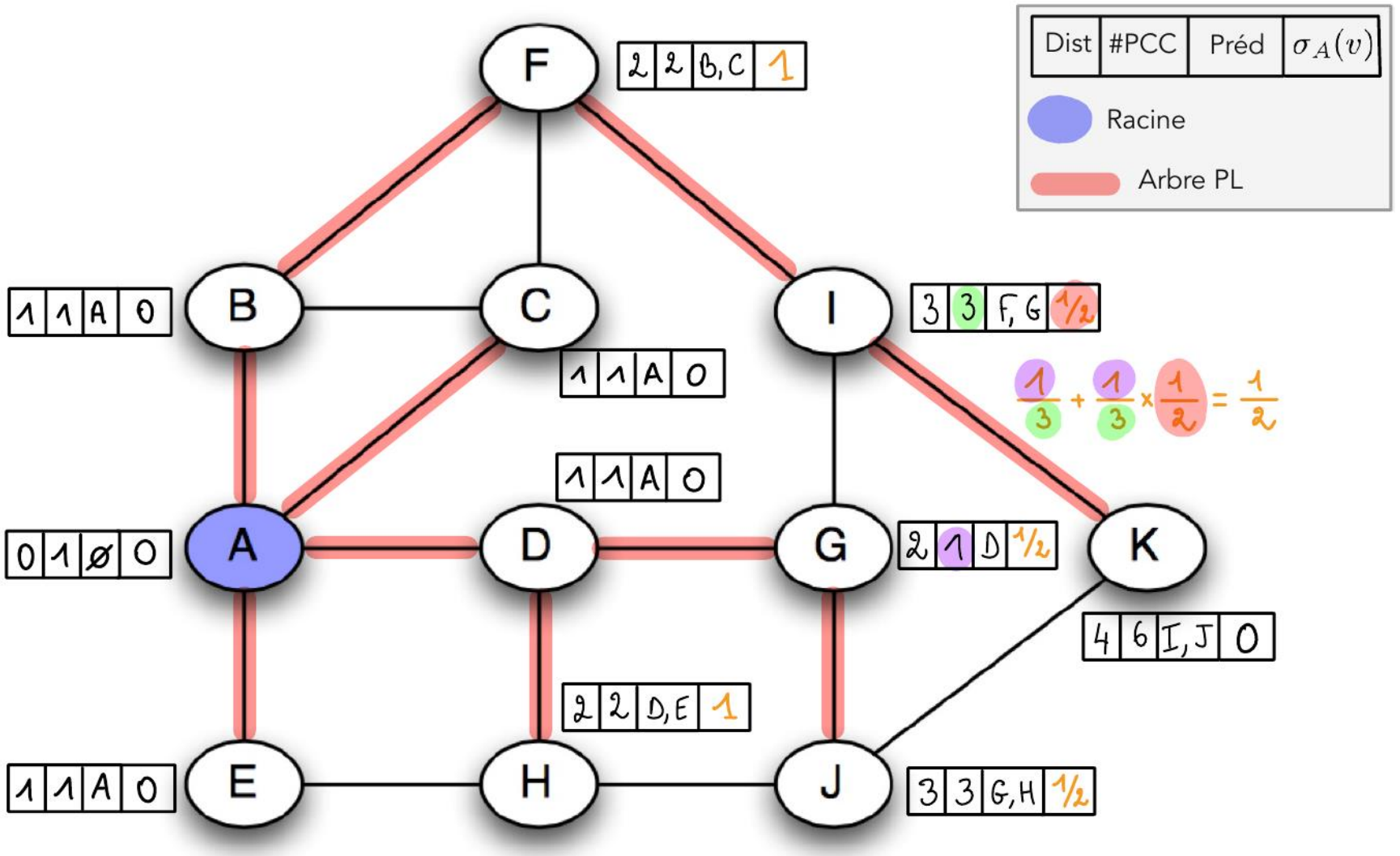
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
B
C
I
A
D
G
K
E
H
J
2 2 B,C 0
3 3 F,G 1/2
1 1 A 0
0 1 ∅ 0
2 1 D 1/2
4 6 I,J 0
2 2 D,E 1
3 3 G,H 1/2
+
2/3 + 2/3 × 1/2 = 1

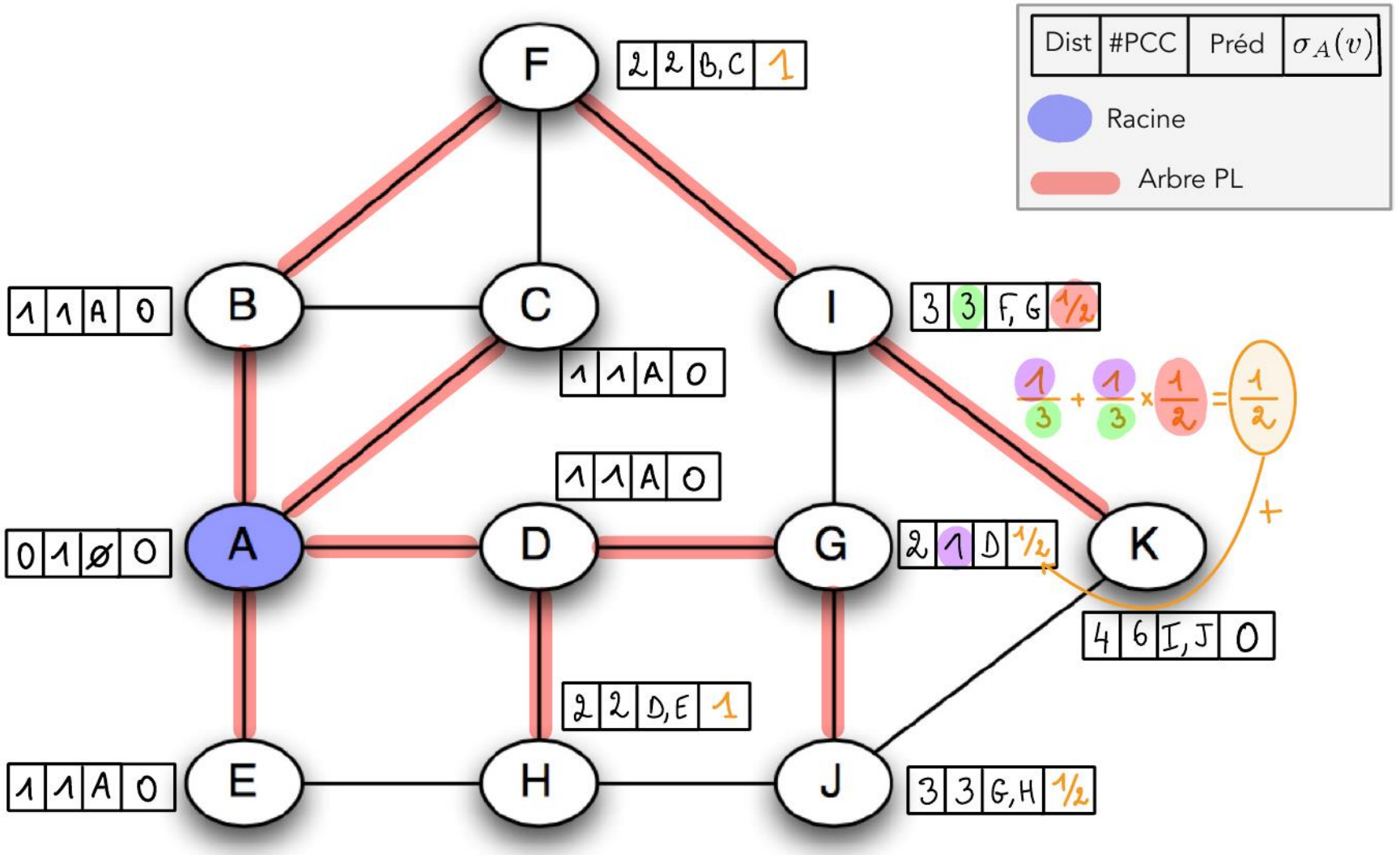
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ø 0
1 1 A 0
D
2 2 D,E 1
G
2 1 D 1/2
K
4 6 I,J 0
E
1 1 A 0
H
J
3 3 G,H 1/2

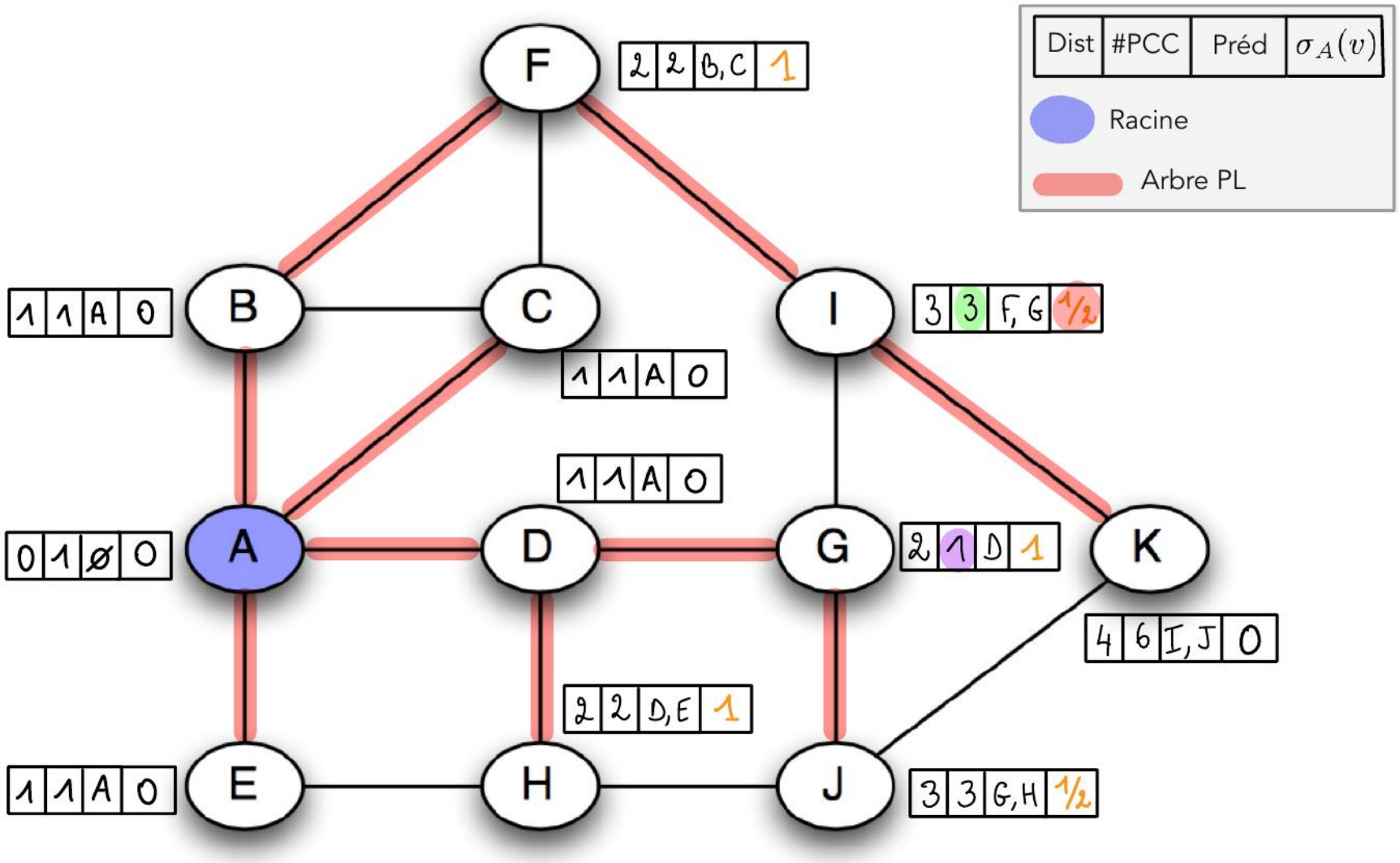
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1/3 + 1/3 x 1/2 = 1/2
A
0 1 ∅ 0
D
1 1 A 0
G
2 1 D 1/2
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 0
H
J
3 3 G,H 1/2

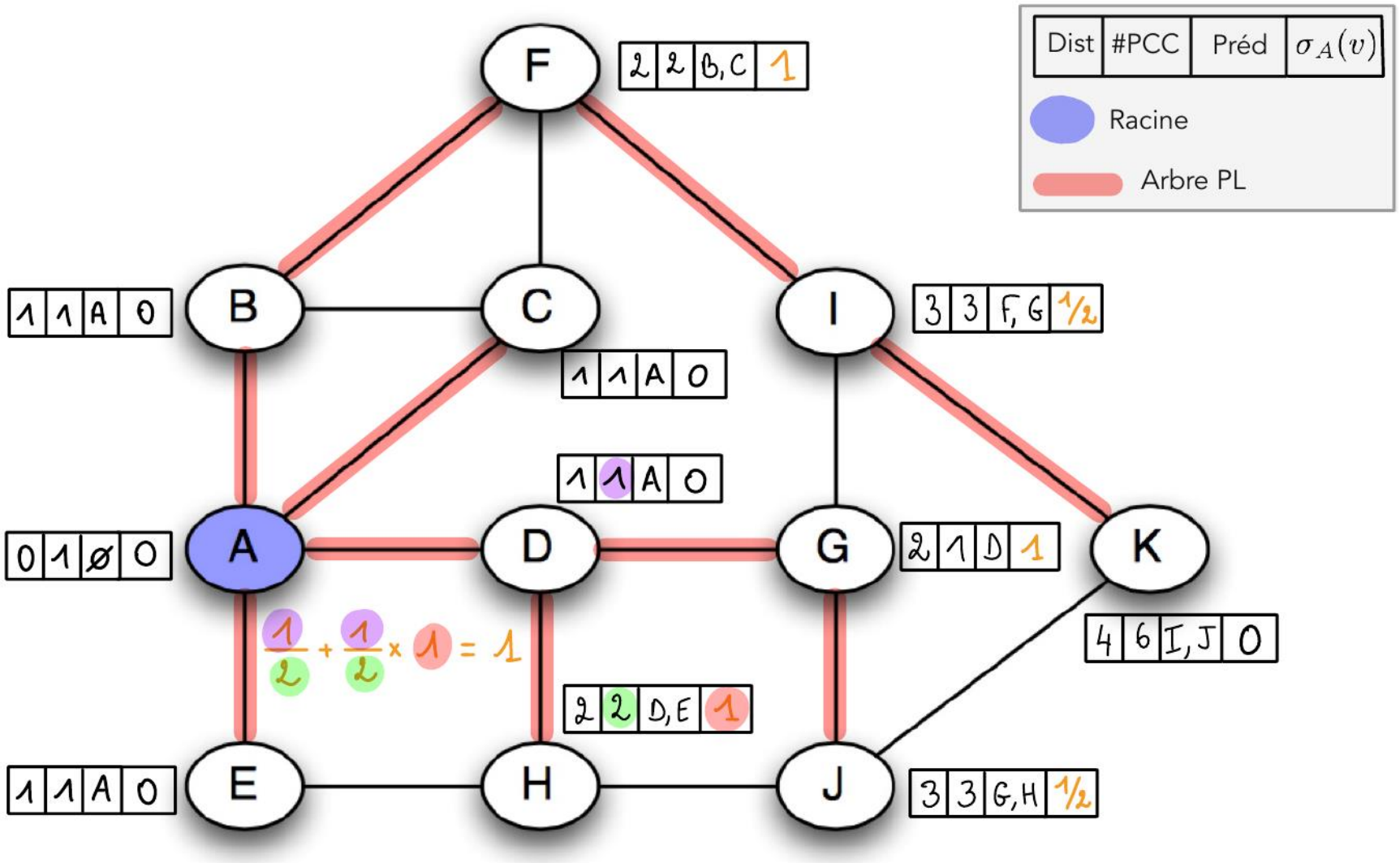
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ø 0
D
1 1 A 0
G
2 1 D 1/2
K
4 6 I,J 0
E
1 1 A 0
H
2 2 D,E 1
J
3 3 G,H 1/2
1/3 + 1/3 × 1/2 = 1/2
+

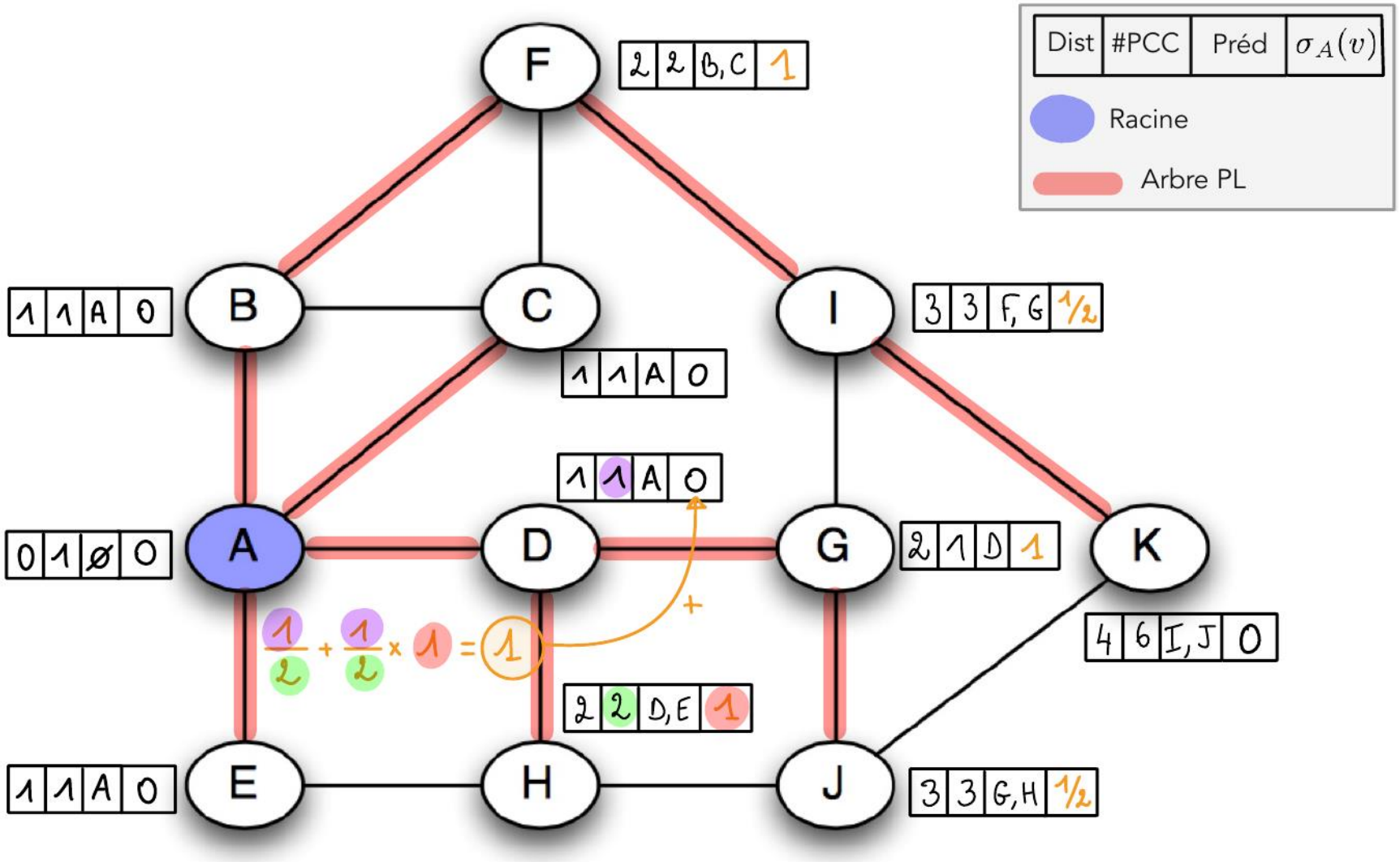
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 0
H
J
3 3 G,H 1/2

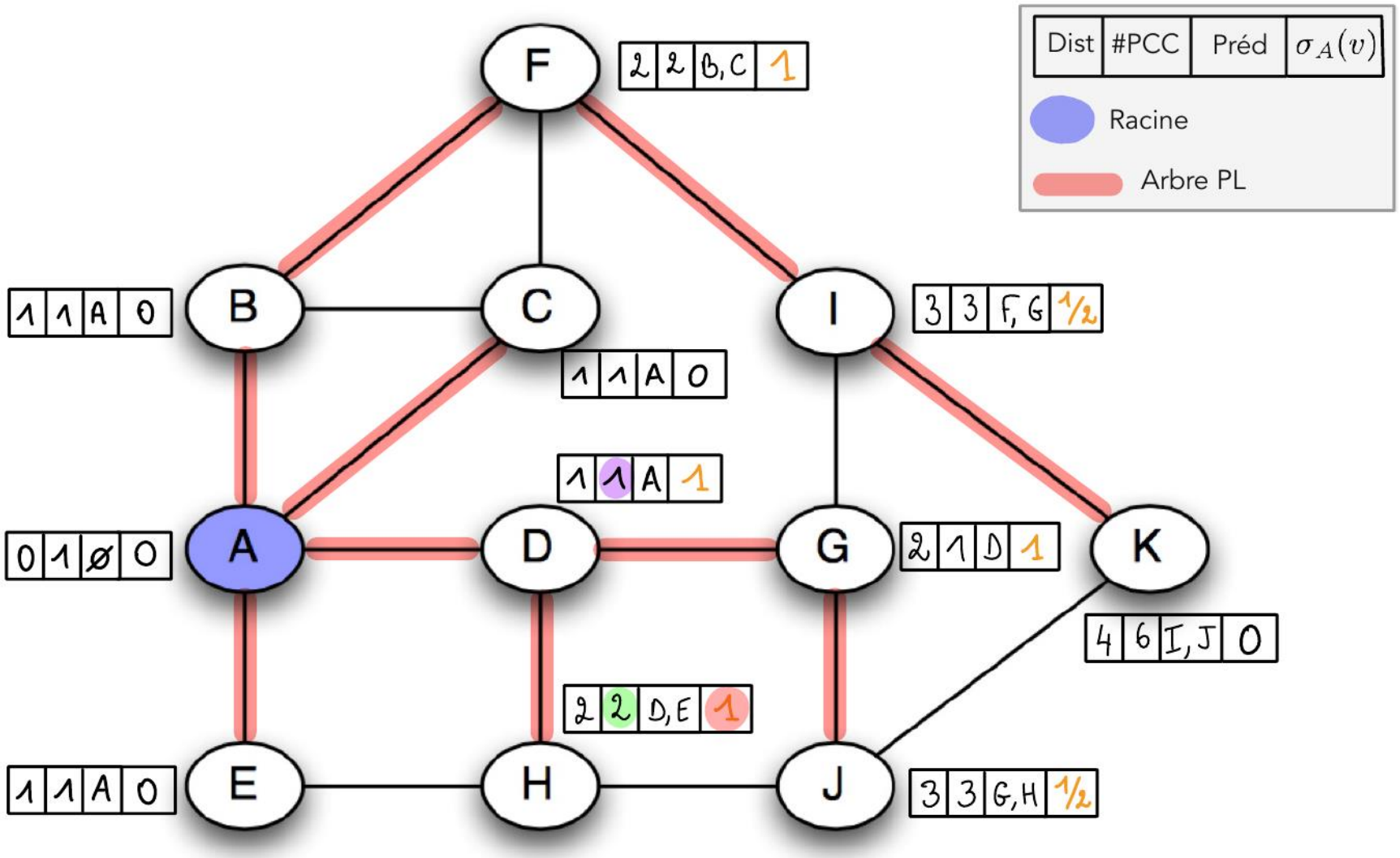
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 0
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
1/2 + 1/2 x 1 = 1
2 2 D,E 1
E
1 1 A 0
H
J
3 3 G,H 1/2

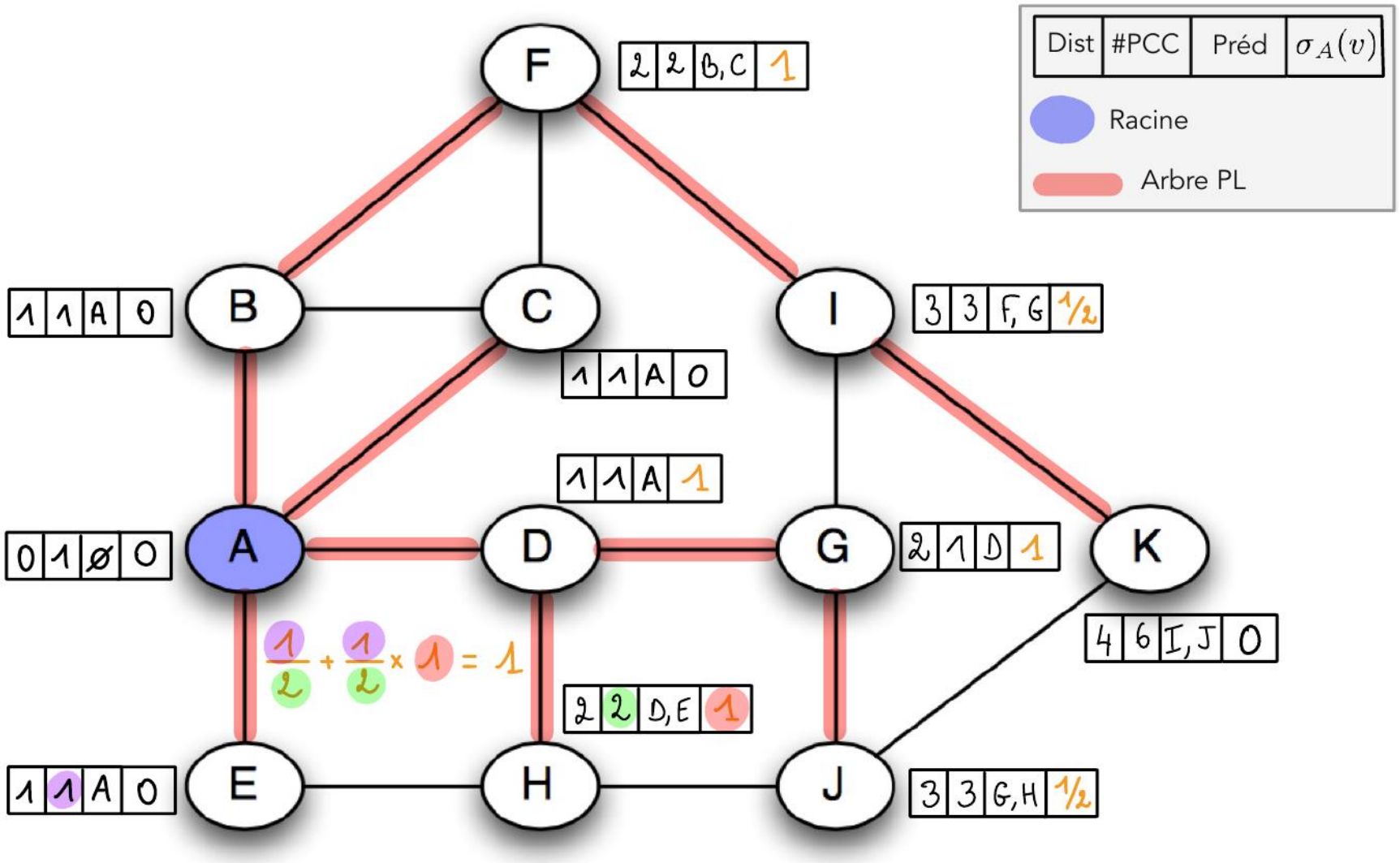
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ø 0
1 1 A 0
D
2 2 D,E 1
G
2 1 D 1
K
4 6 I,J 0
E
1 1 A 0
H
J
3 3 G,H 1/2
1/2 + 1/2 x 1 = 1
+

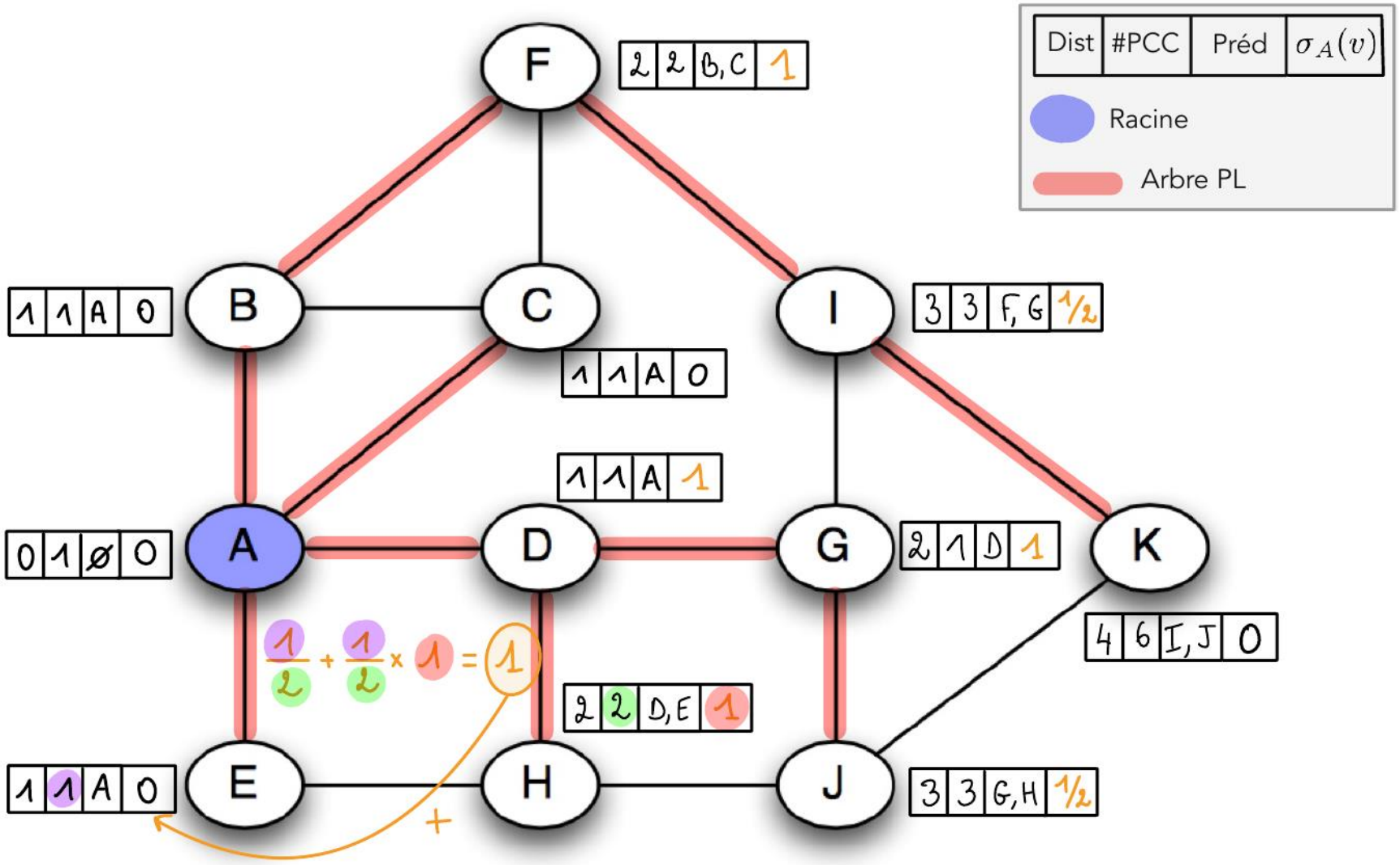
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 1
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 0
H
J
3 3 G,H 1/2

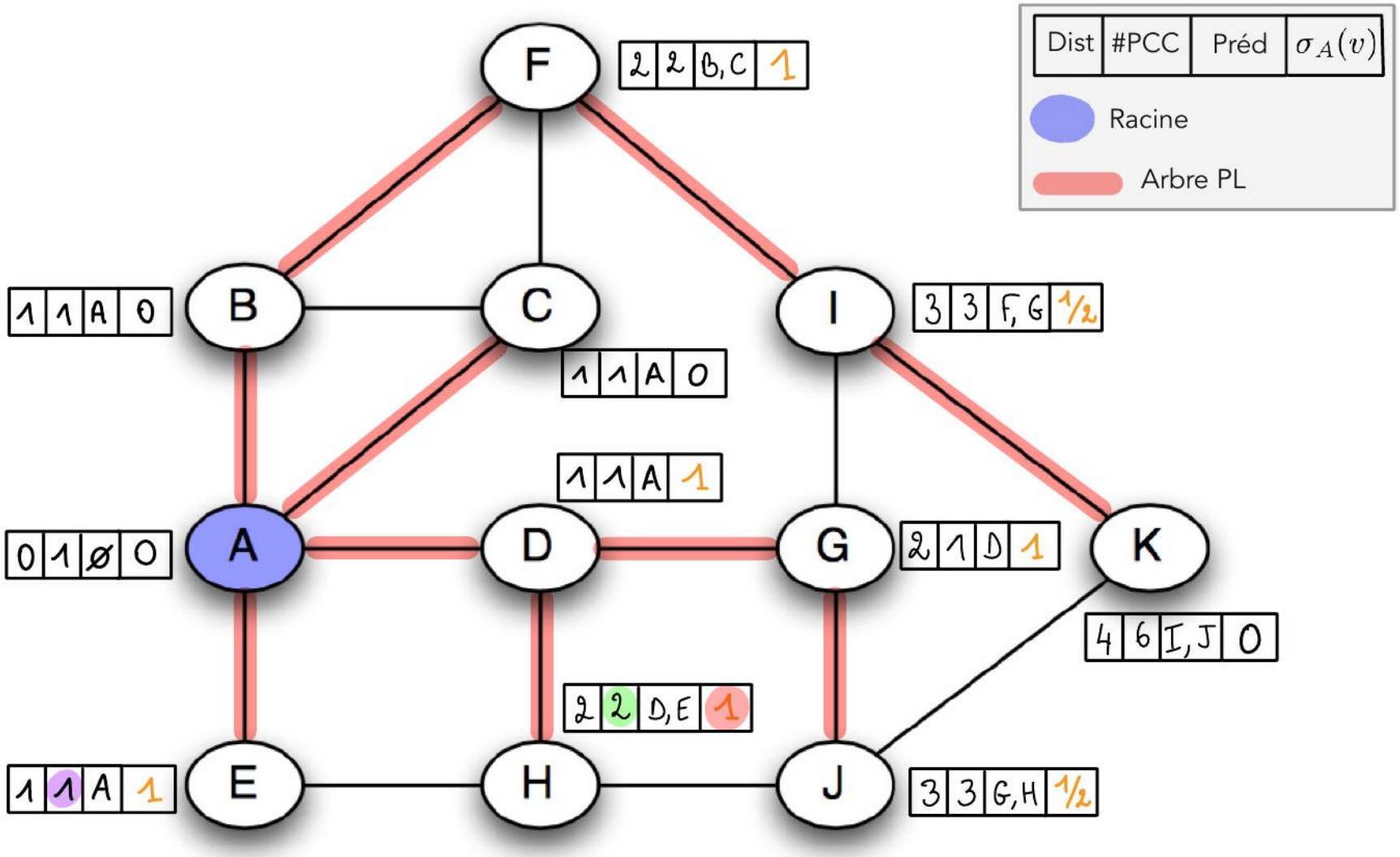
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 1
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
1/2 + 1/2 x 1 = 1
2 2 D,E 1
E
1 1 A 0
H
J
3 3 G,H 1/2

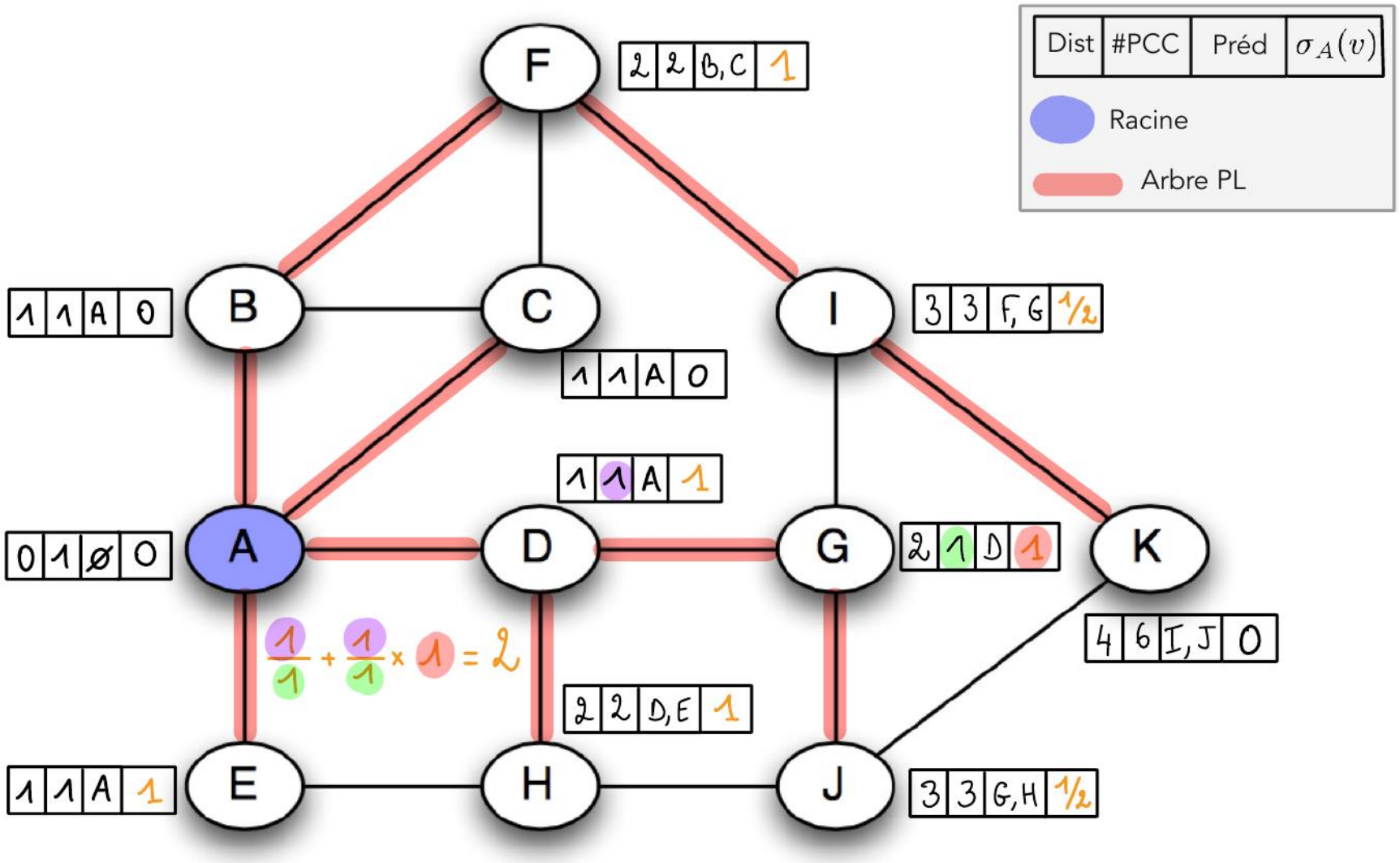
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 1
A
0 1 ø 0
D
G
2 1 D 1
K
4 6 I,J 0
1/2 + 1/2 x 1 = 1
2 2 D,E 1
E
1 1 A 0
H
J
3 3 G,H 1/2
+

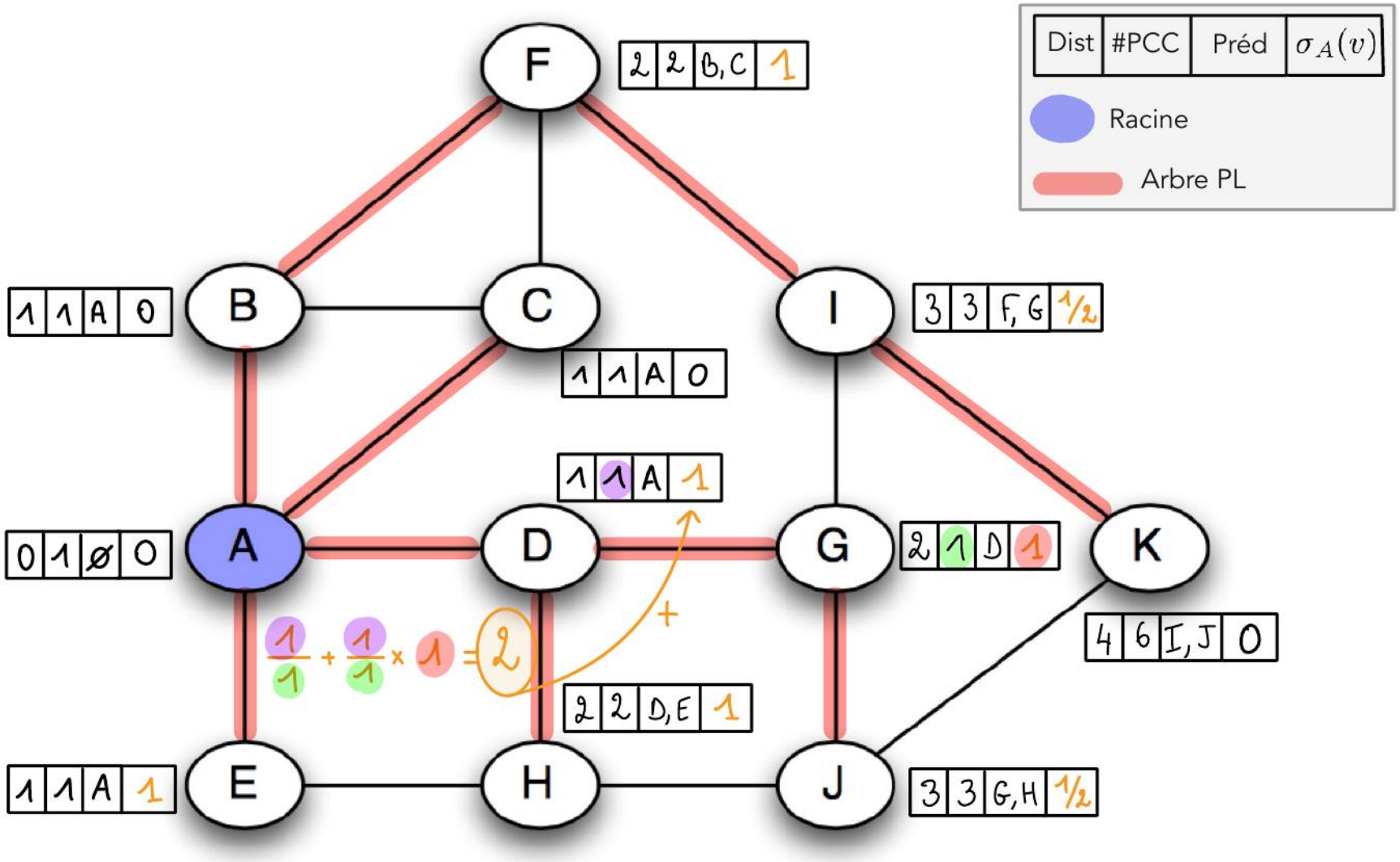
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 1
A
0 1 ø 0
D
G
2 1 D 1
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 1
H
J
3 3 G,H 1/2

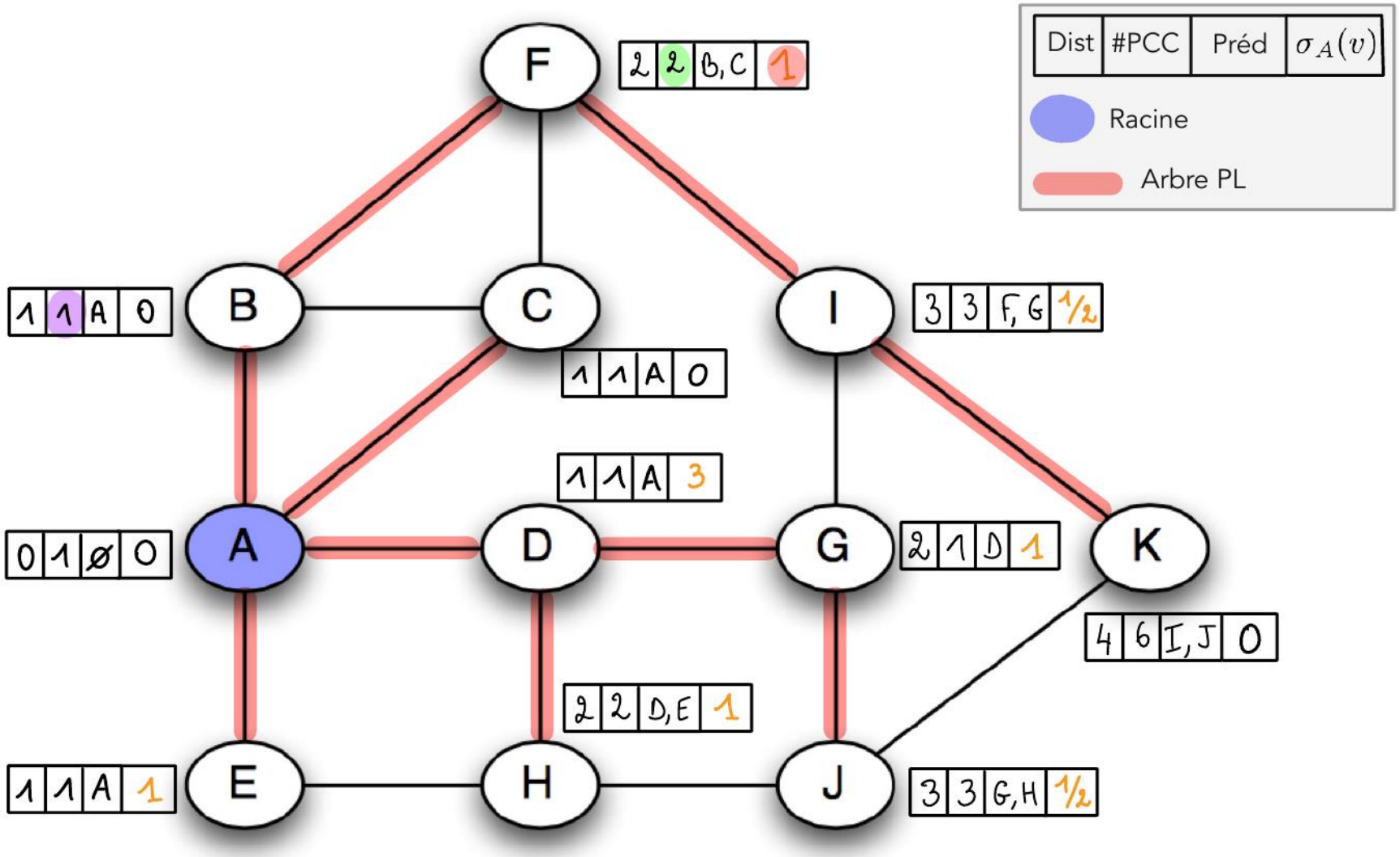
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 1
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
1/1 + 1/1 x 1 = 2
2 2 D,E 1
E
1 1 A 1
H
J
3 3 G,H 1/2

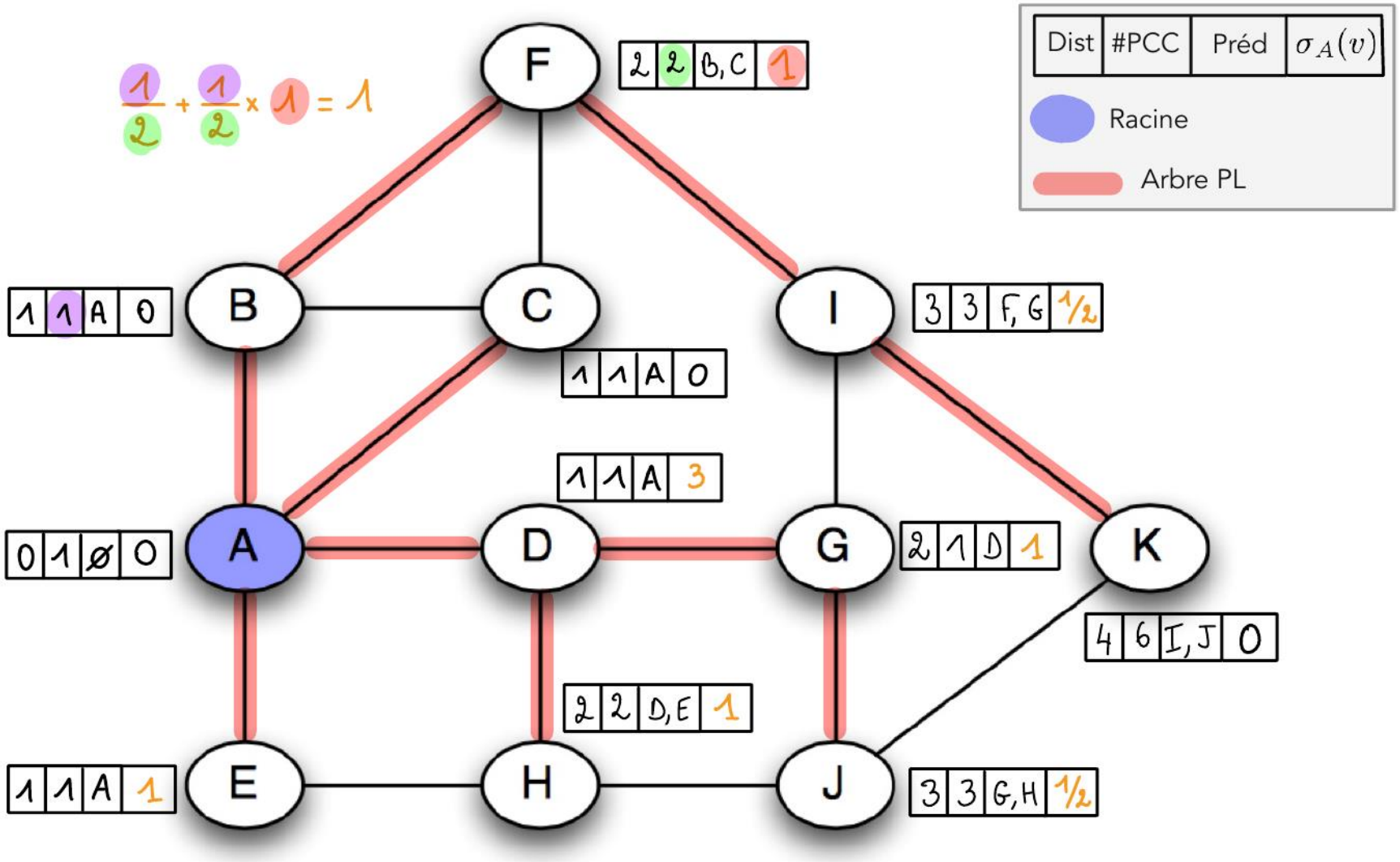
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ∅ 0
D
1 1 A 1
G
2 1 D 1
K
4 6 I,J 0
1/1 + 1/1 x 1 = 2
+
2 2 D,E 1
E
1 1 A 1
H
J
3 3 G,H 1/2

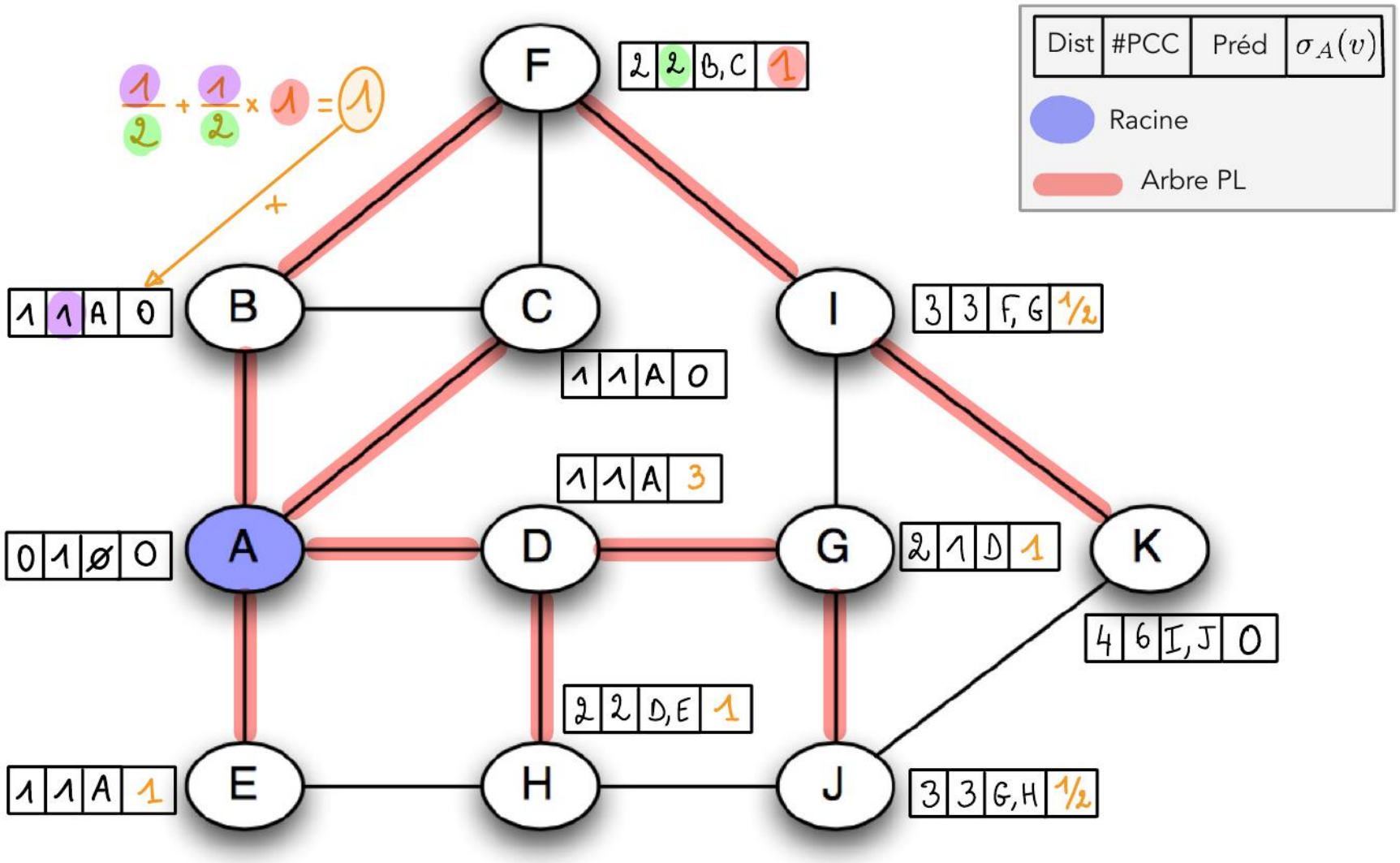
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 0
C
1 1 A 0
I
3 3 F,G 1/2
1 1 A 3
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 1
H
J
3 3 G,H 1/2

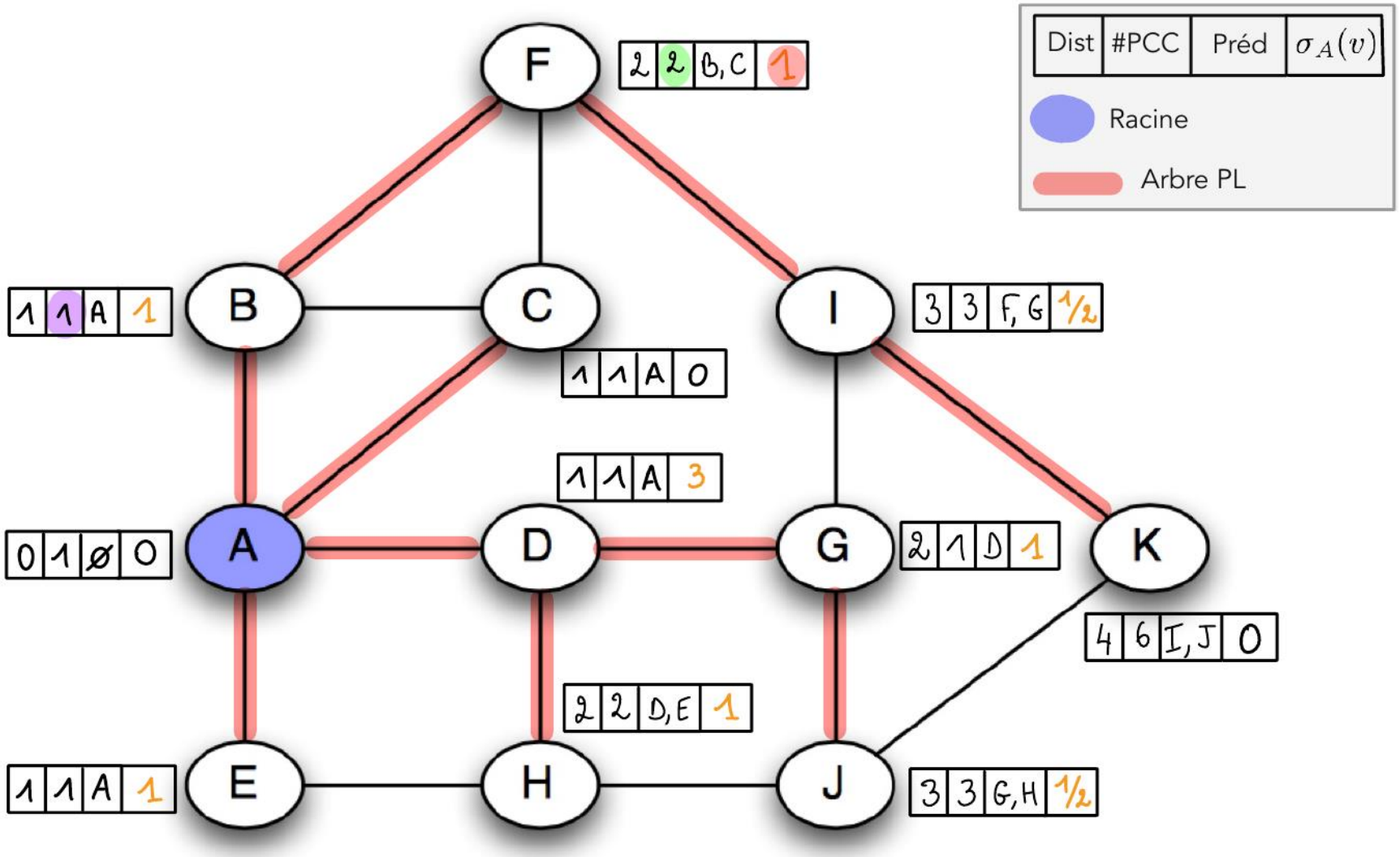
Un exemple



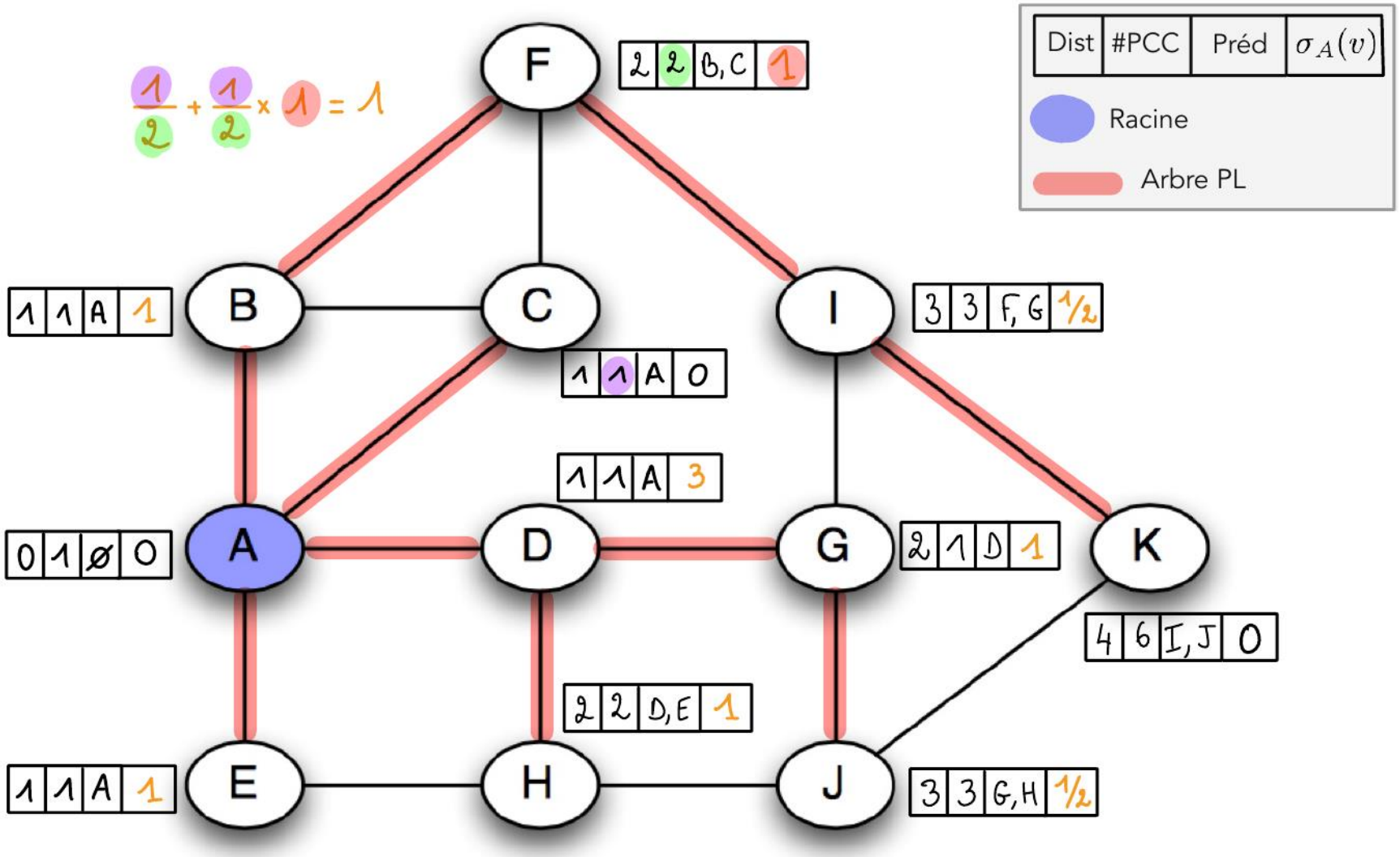
Un exemple



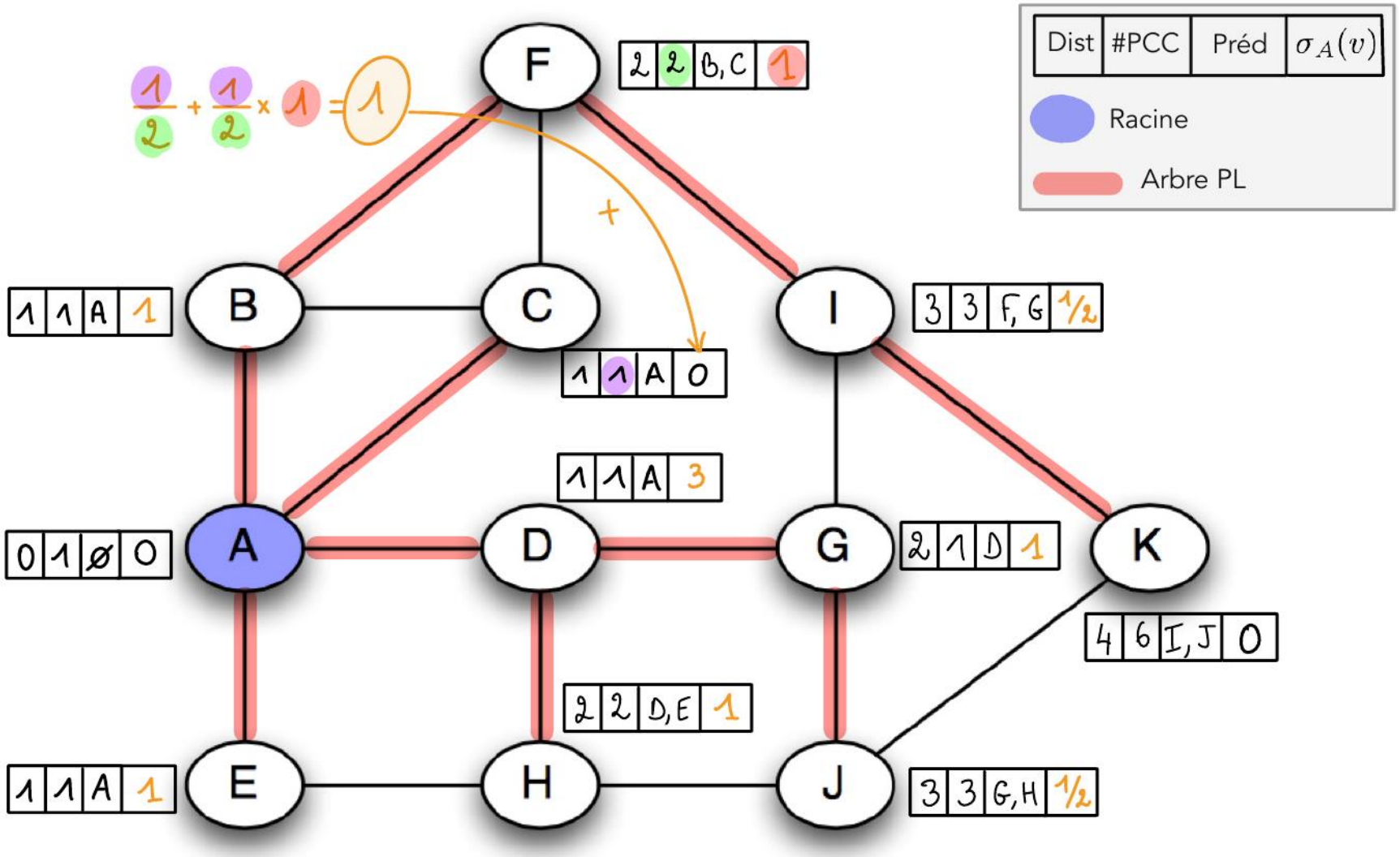
Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 1
C
1 1 A 0
I
3 3 F,G 1/2
A
0 1 ø 0
D
1 1 A 3
G
2 1 D 1
K
4 6 I,J 0
E
1 1 A 1
H
2 2 D,E 1
J
3 3 G,H 1/2

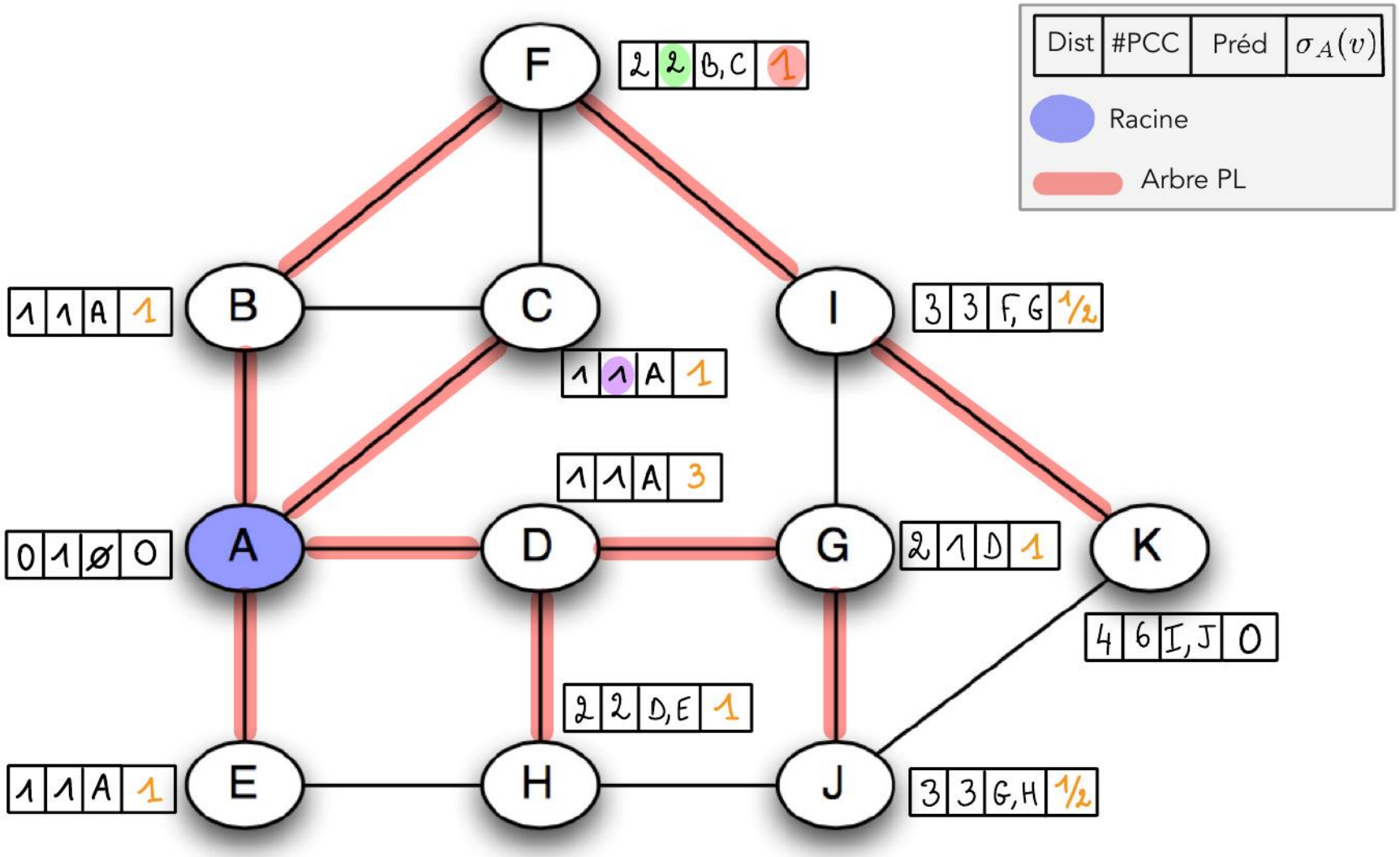
Un exemple



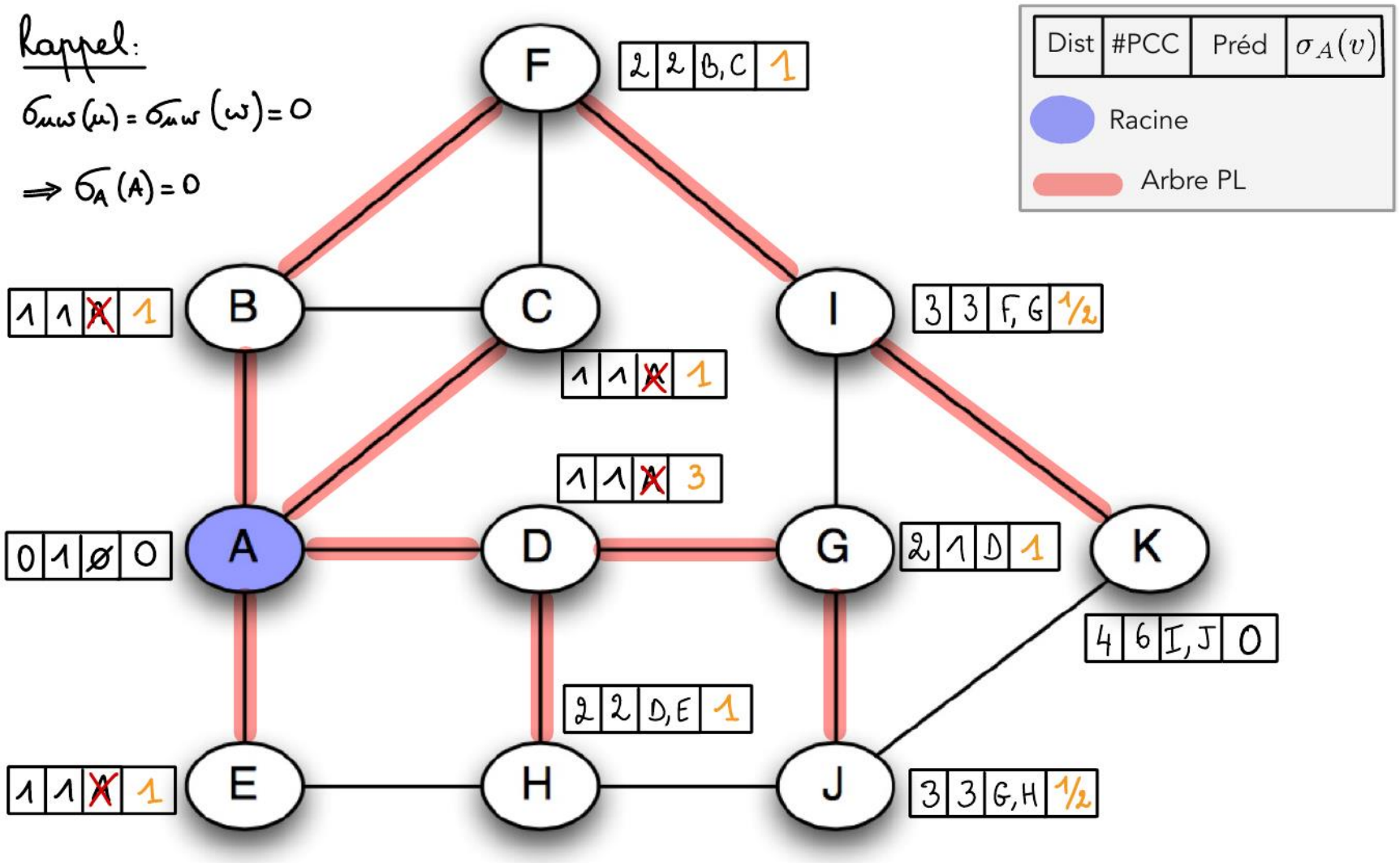
Un exemple



Un exemple


Dist
#PCC
Préd
$\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 1
C
1 1 A 1
I
3 3 F,G 1/2
1 1 A 3
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 1
H
J
3 3 G,H 1/2

Un exemple


Rappel :
$\sigma_{sw}(u) = \sigma_{sw}(w) = 0$
$\Rightarrow \sigma_A(A) = 0$
Dist | #PCC | Préd | $\sigma_A(v)$
Racine
Arbre PL
F
2 2 B,C 1
B
1 1 A 1
C
1 1 A 1
I
3 3 F,G 1/2
1 1 A 3
A
0 1 ∅ 0
D
G
2 1 D 1
K
4 6 I,J 0
2 2 D,E 1
E
1 1 A 1
H
J
3 3 G,H 1/2

Algorithme de Brandes (2001)

Algorithme : Centralité d'Intermédiarité (Brandes, 2001)
Données : Un graphe non pondéré $G = (V, E)$ donné par liste de voisins
Résultat : Une fonction : $C_B : V \to \mathbb{R}$ (la centralité d'intermédiarité de chaque sommet).

```
1  début
2      pour tous les v ∈ V faire
3          C_B(v) ⟵ 0 ; // la centralité d'intermédiarité.
4      pour tous les r ∈ V faire
5          AT ⟵ file vide ; // sommets à traiter
6          S ⟵ pile vide ; // pour obtenir les sommets par distance non-croissante
7          P ⟵ liste de files ;
8          pour tous les v ∈ V faire
9              ℓ(v) ⟵ −1 ; // la distance (niveau) à la racine r, −1 signifie non visité.
10             σ(v) ⟵ 0 ; // le nombre de PCC pour atteindre v depuis r.
11             P(v) ⟵ file vide ; // prédécesseurs de v sur les PCC de r à v.
12             δ(v) ⟵ 0 ; // centralité d'intermédiarité de v pour les PCC débutant en r
13         ℓ(r) ⟵ 0 ; // la racine
14         σ(r) ⟵ 1 ; // la racine
15         Enfiler r dans AT ;
16         tant que AT ≠ ∅ faire
17             Prendre v le premier sommet de AT, le défiler de AT, l'empiler dans S ;
18             pour tous les x ∈ Vois(v) faire
19                 si ℓ(x) = −1 alors
20                     Enfiler x dans AT ;
21                     ℓ(x) ⟵ ℓ(v) + 1 ;
22                 si ℓ(x) = ℓ(v) + 1 alors
23                     σ(x) ⟵ σ(x) + σ(v) ;
24                     si v ≠ r alors
25                         Enfiler v dans P(x) ;
26         tant que S ≠ ∅ faire
27             Prendre v le premier sommet de S, le dépiler de S ;
28             pour tous les p ∈ P(v) faire
29                 δ(p) ⟵ δ(p) + (σ(p)/σ(v)) · (1 + δ(v)) ;
30             si v ≠ r alors
31                 C_B(v) ⟵ C_B(v) + δ(v)
```

Parcours en largeur
Comptage # de PCC

Backtrack

Centralité : corrélation

- Généralement, les différents paramètres de centralité sont positivement corrélés.
- Quand ils ne le sont pas, c'est révélateur de configurations particulières

	Faible degré	Faible proximité	Faible intermédiarité
Degré élevé		Au sein d'une partie très dense mais loin du reste du graphe	De nombreux plus courts chemins, les communications évitent ce nœud
Proximité élevée	Relié à un sommet important du graphe		De nombreux plus courts chemins, proche du nœud « ego »
Intermédiarité élevée	Le nœud se trouve sur un « pont »	Très rare, à la frontière d'une communauté avec très peu de chemins sortants	