U.F.R SCIENCES ET TECHNIQUES
Département d'Informatique
B.P. 1155
64013 PAU CEDEX
Téléphone secrétariat : 05.59.40.79.64
Télécopie : 05.59.40.76.54

V- PROBLEME DU CHEMIN OPTIMAL

I- POSITION DU PROBLEME
II- ALGORITHME DE DIJKSTRA
II- ALGORITHME DE BELLMAN

I- Position du problème

Il est tentant, lorsque l'on se rend d'un point à un autre, dans un réseau routier, par exemple, d'interroger le GPS, sur le trajet **optimal**.

Le cas où le nombre de trajets possibles entre le point de départ et le point d'arrivée est faible est **trivial**.

Il suffira de :

- calculer le coût de **chacun** des trajets,
- de **comparer** directement les coûts obtenus.

Cette solution **exhaustive** devient rapidement impraticable si le nombre de trajets possibles est **grand**.

Heureusement, il existe des algorithmes qui évitent d'avoir à calculer **tous** les trajets possibles.

1- Un problème moins simple qu'il n'y paraît

Soit par l'exemple un **réseau de communication.**

On peut le représente par un **graphe** orienté où:

- les **sommets** représentent les **nœuds** du réseau,
- les **arcs**, le **liens** du réseau.

Imaginez ce réseau composé de n=**20** nœuds et comportant un lien direct entre chaque nœud.

Un tel réseau comporte **380** liens.

Un simple **calcul combinatoire** montre :

- qu'entre deux noeuds **x** et **y**,
- le nombre de chemins possibles est supérieur à **6. 10^{12}**

La solution exhaustive prendrait environ **741** jours !

Cet exemple montre bien la nécessité de disposer d'algorithmes :

- évitant de calculer **tous** les trajets possibles,
- pour éviter l'**explosion combinatoire**.

On connaît des algorithmes, dont la complexité (temps de calcul) est, dans le pire des cas:

- en $O(n^3)$
- voire, souvent, en $O(n^2)$.

Ainsi, pour notre exemple, le nombre de chemins calculés sera, selon les cas :

- au pire de… 20^3 c'est à dire 8000,
- voire même de… 20^2 : 400

2- Cas particulier d'un problème plus général

Le problème sous-jacent est celui plus connu sous le nom du **chemin de valeur optimale** dans un graphe valué.

L'exemple le plus connu est sans doute celui du **réseau de communication**.

Dans un tel réseau, on imagine aisément la diversité du type de **coûts** pouvant être attribuées aux **liens** (arcs du graphe)

- **distances**, **temps** de parcours, **coûts** de transferts,

- **fiabilités** : valeurs comprises entre 0 et 1,

- **capacités** : débits maximum, par exemple , etc.

Le long d'un chemin, les **distances**, **temps**, **coûts** vont en général s'**additionner**.

Les **fiabilités** vont se **multiplier** et…

les **capacités** vont se **composer par minimum**.

On aura tendance à rechercher:

- un **minimum** dans le cas des **distances**, **temps**, **coûts : «**chemin le plus court»

- mais un **maximum** dans le cas des **fiabilités** ou des **capacités**. «chemin le plus long»

De façon plus générale, on imagine aisément toutes les applications d'un tel mécanisme:

- itinéraire routier,
- réseau de télécommunication,
- système de trafic d'un réseau ferroviaire,
- ordonnancement des tâches en production robotisée.

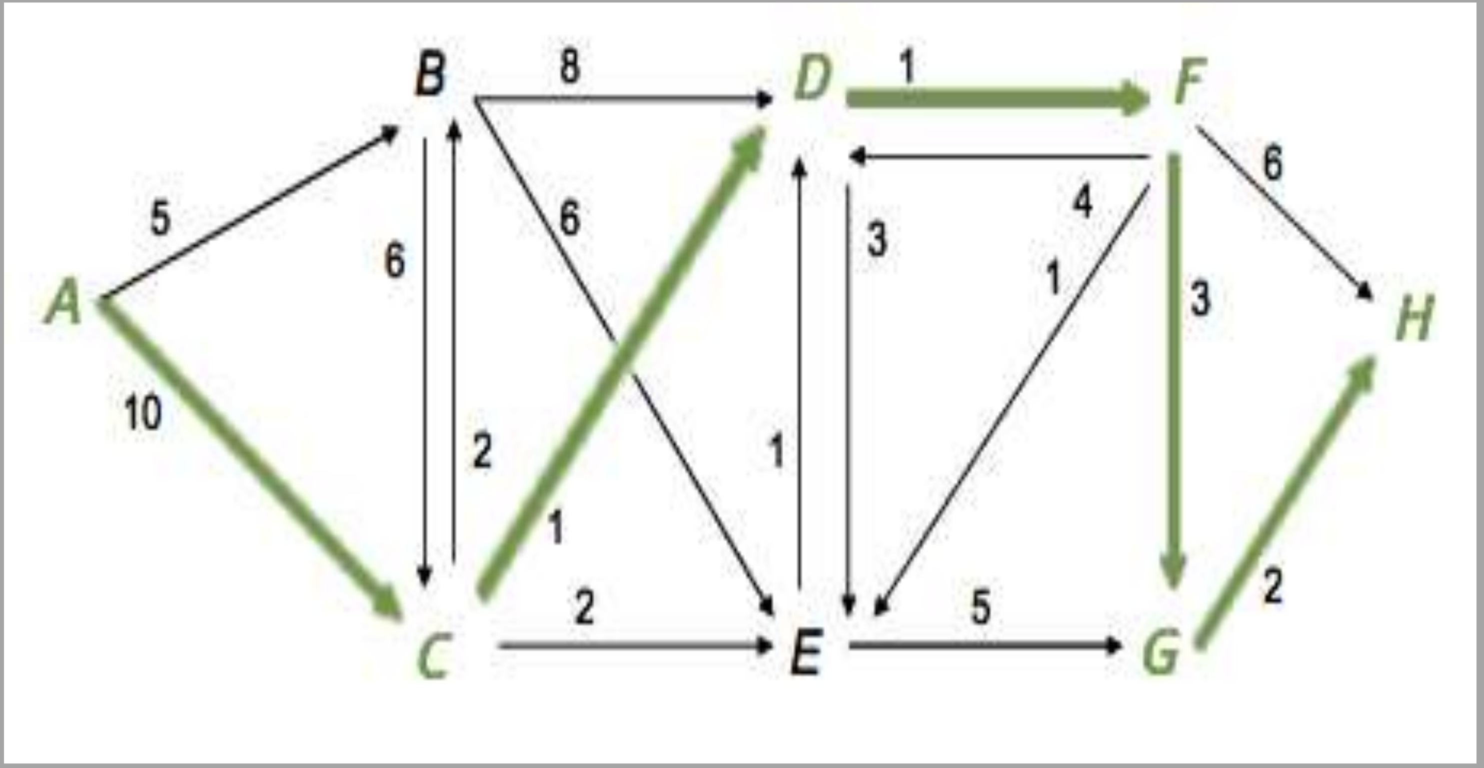
Position du problème

Dans ce qui va suivre, nous avons choisi de nous limiter au problème de recherche du **plus «court» chemin.**

Il est plus connu en théorie sous le nom de **«chemin de valeur additive minimale »**.

3- Mieux formaliser le problème

1- Le premier résultat cherché est bien sûr la **valeur minimale** du coût



Dans le graphe ci-dessus, le plus «court» chemin pour aller de A à H a pour coût **17**.

Mais, cela ne suffit pas !

2- le deuxième résultat est le tracé d'**un chemin** ayant ce coût minimum.

Ainsi, le plus court chemin de A à H a pour coût 17, et son tracé est :

A→ C→ D→ F→G→ H

On parle alors de **routage**.

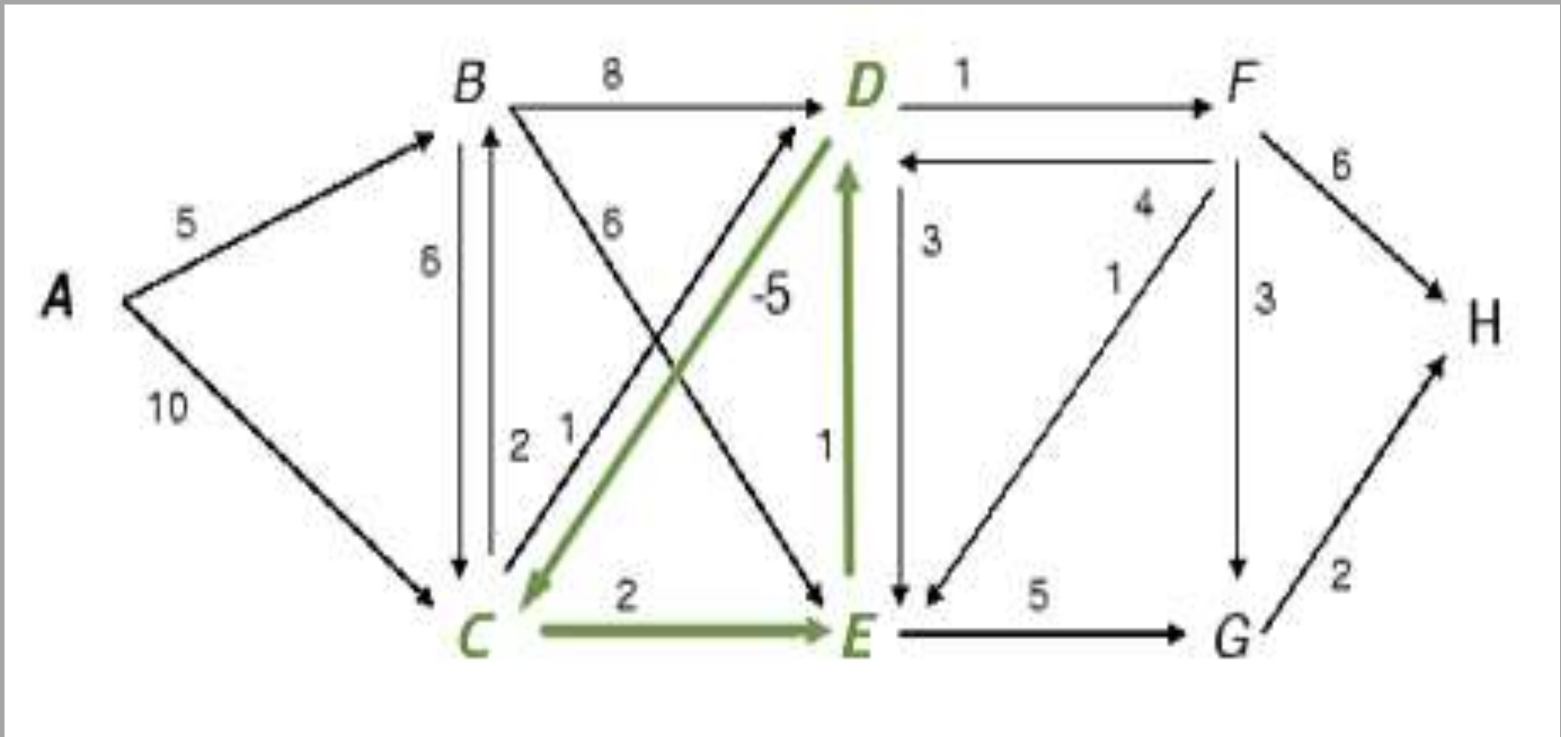
4- Condition d'existence d'une solution

Le problème étant ainsi précisé, peut-on affirmer qu'il y a toujours une solution ?

La question peut paraître **triviale**, mais elle ne l'est pas.

5-Preuve de non trivialité

Supposons que le graphe précédent possède des **circuits**.



Il peut exister des **chemins** empruntant de tels circuits.

Sur cette figure, par exemple, le chemin

[A → B → D →C →E→ D→ F→ G→H]

comprend le circuit [D →C →E→ D].

Lequel circuit a pour coût **-2**.

Si l'on cherche les chemins de valeur **minimale** entre A et H, le problème n'a **pas de solution**.

En effet, on peut trouver :

- un chemin de **coût aussi petit** que l'on veut,
- en «tournant» suffisamment de fois sur le circuit.

De tels circuits s'appellent **circuits absorbants**.

Leur présence éventuelle **doit être détectée**.

Sans quoi, la **terminaison** les algorithmes n’est pas garantie.

La question n'est pas seulement théorique : il existe des systèmes **concrets** :

- **ayant pour** modèle un graphe,
- avec des circuits **absorbants**.

C'est le cas pour de nombreux **problèmes de décision** traités en Recherche Opérationnelle: **ordonnancement**, **plannings** , **flot**,

6- Multiplicité de solutions

Dans un graphe, le nombre de chemins **élémentaires** de **x** vers **y** est **fini**.

Il existe **au moins un** plus court chemin élémentaire de **x** vers **y**.

7- Variantes du problème

Le problème de **plus court chemin** se décline sous l'une des trois formes suivantes :

-P_1 : « rechercher **un** plus court chemin entre **deux sommets** donnés **x** et **y** » ;

P_2 : « rechercher **les** plus courts chemins entre **un** sommet donné **s**, appelé **source,** et **tous les autres»**;

P_3 : « rechercher **les** plus courts chemins entre **tous les couples**(x,y) de sommets ».

A première vue, il semble que ces problèmes aillent du plus simple au plus complexe.

En effet, le nombre de chemins à calculer va en **croissant**.

En réalité, il n'en est rien !.

La résolution du problème $\mathbf{P_1}$ nécessite en général de résoudre le problème $\mathbf{P_2}$!

Par ailleurs, dans le cas général, le meilleur algorithme connu:

- pour résoudre le problème $\mathbf{P_3}$
- n'est pas plus complexe que le meilleur algorithme connu pour résoudre le problème $\mathbf{P_2}$.

II- Algorithme de Dijkstra

Le problème de **plus court chemin** est abordé ici sous sa forme $\mathbf{P_2}$.

Soit un graphe orienté G= (S,A) valué et un sommet **s** :
la source.

L'algorithme ne peut être appliqué que lorsque tous les arcs ont un coût **non négatif**.

L'algorithme de Dijkstra déploie une **stratégie** pour calculer **les** plus courts chemins :

- entre un sommet **s** : **source**

-et **tous les autres sommets** du graphe;

Cette stratégie évite le **calcul exhaustif** des chemins possibles

1- Idée

C’est un algorithme dit «glouton» qui :

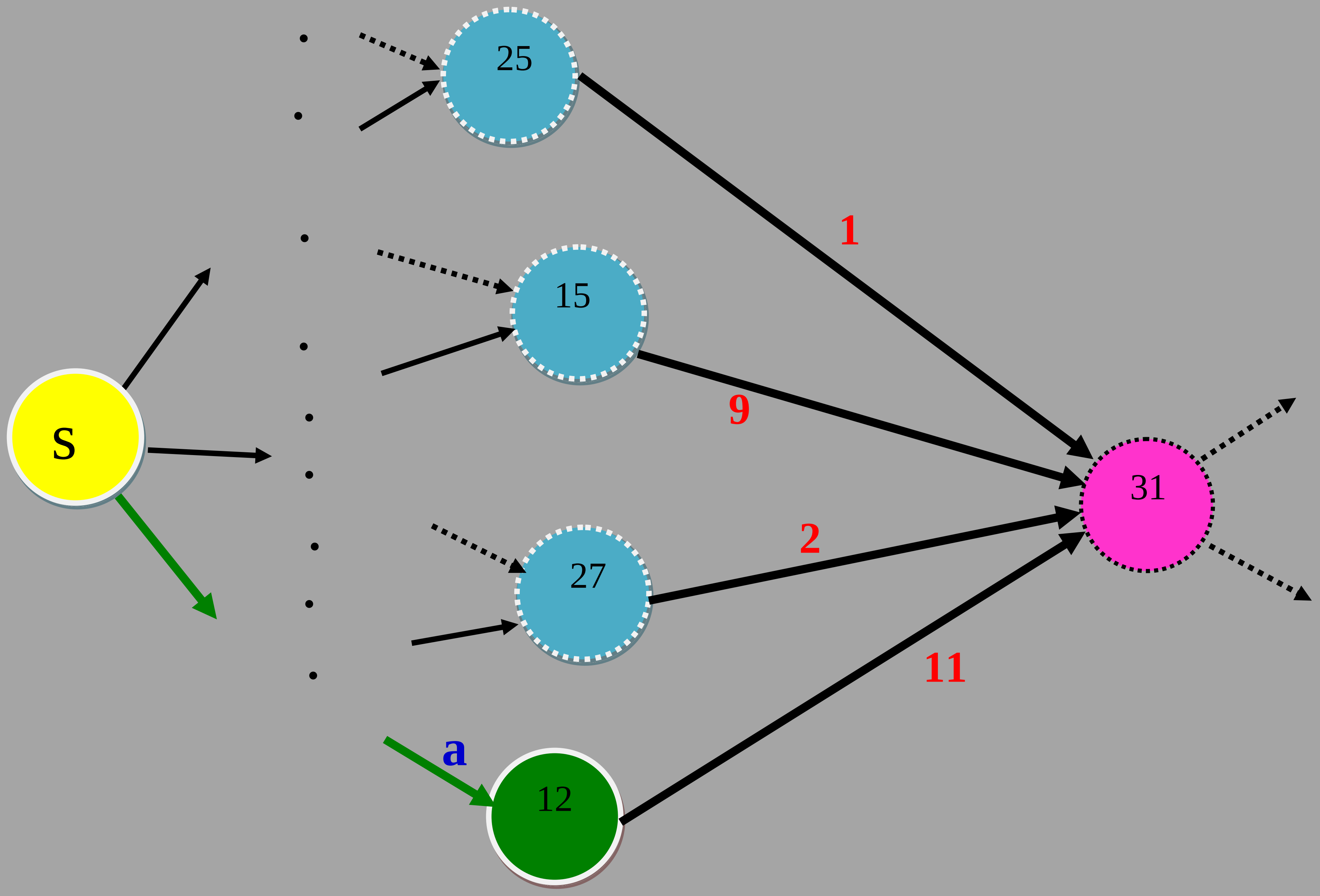
1-repose sur **l'exploration de la descendance** d’un sommet.

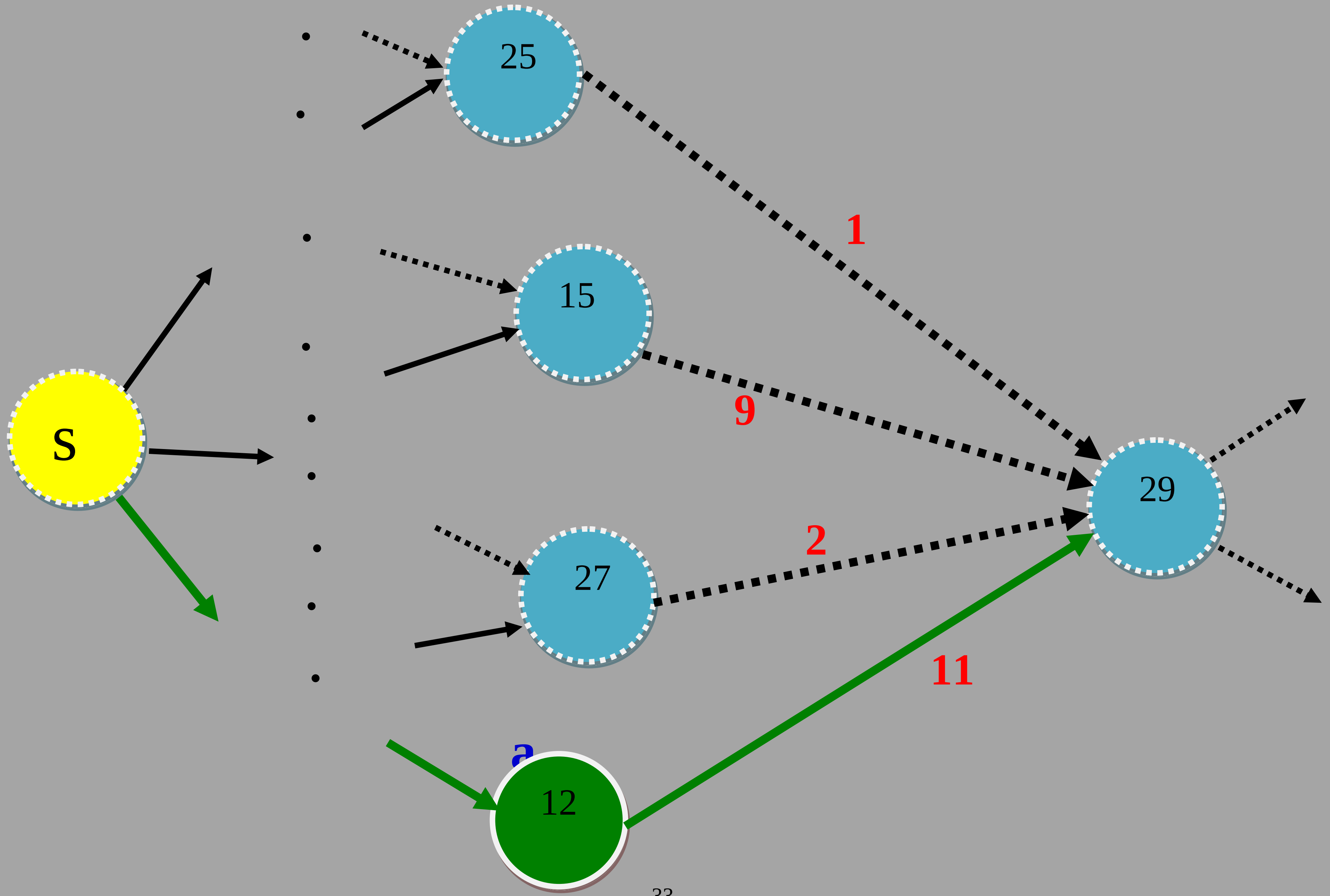
2-utilise, pour cela, la stratégie dite «**à partir du meilleur prédécesseur»**.

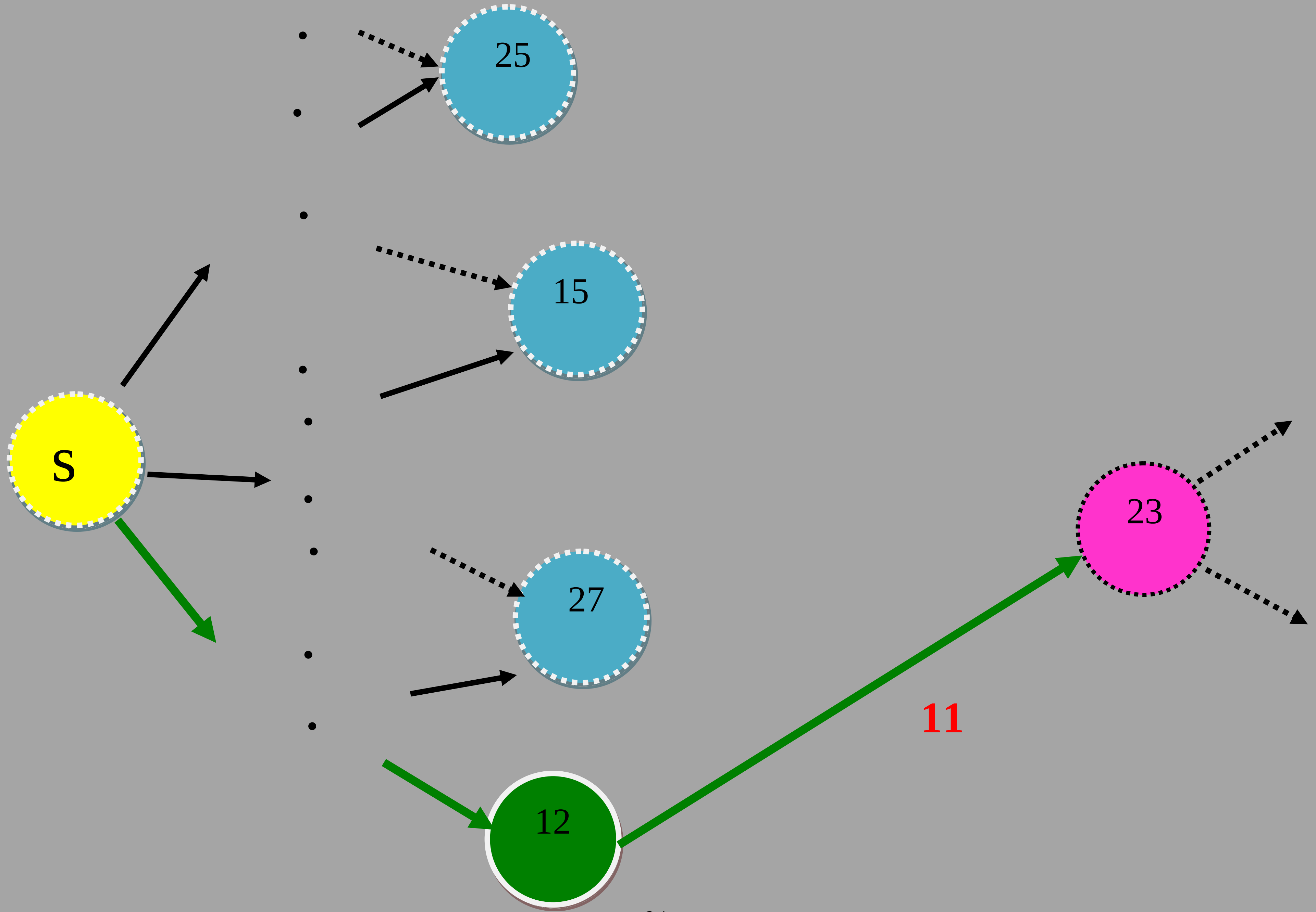
Le mode opératoire de cette stratégie agit en temps :

1-chercher le sommet le **plus proche** de s

2-explorer **ses successeurs** pour tester s'il est **le meilleur prédécesseur**.


25
15
27
12
31
S
1
9
2
11
a


25
1
15
9
S
29
27
2
11
a
12


25
15
S
27
23
11
12

Dans le graphe G, cela signifie atteindre :

- tout **descendant**
- **à partir du meilleur prédécesseur visité.**

« **meilleur** » signifie celui qui correspond à la distance la **plus courte**.

2-Principe

L'algorithme fonctionne en construisant un **arbre couvrant** éventuellement **partiellement**, le graphe G, soit :

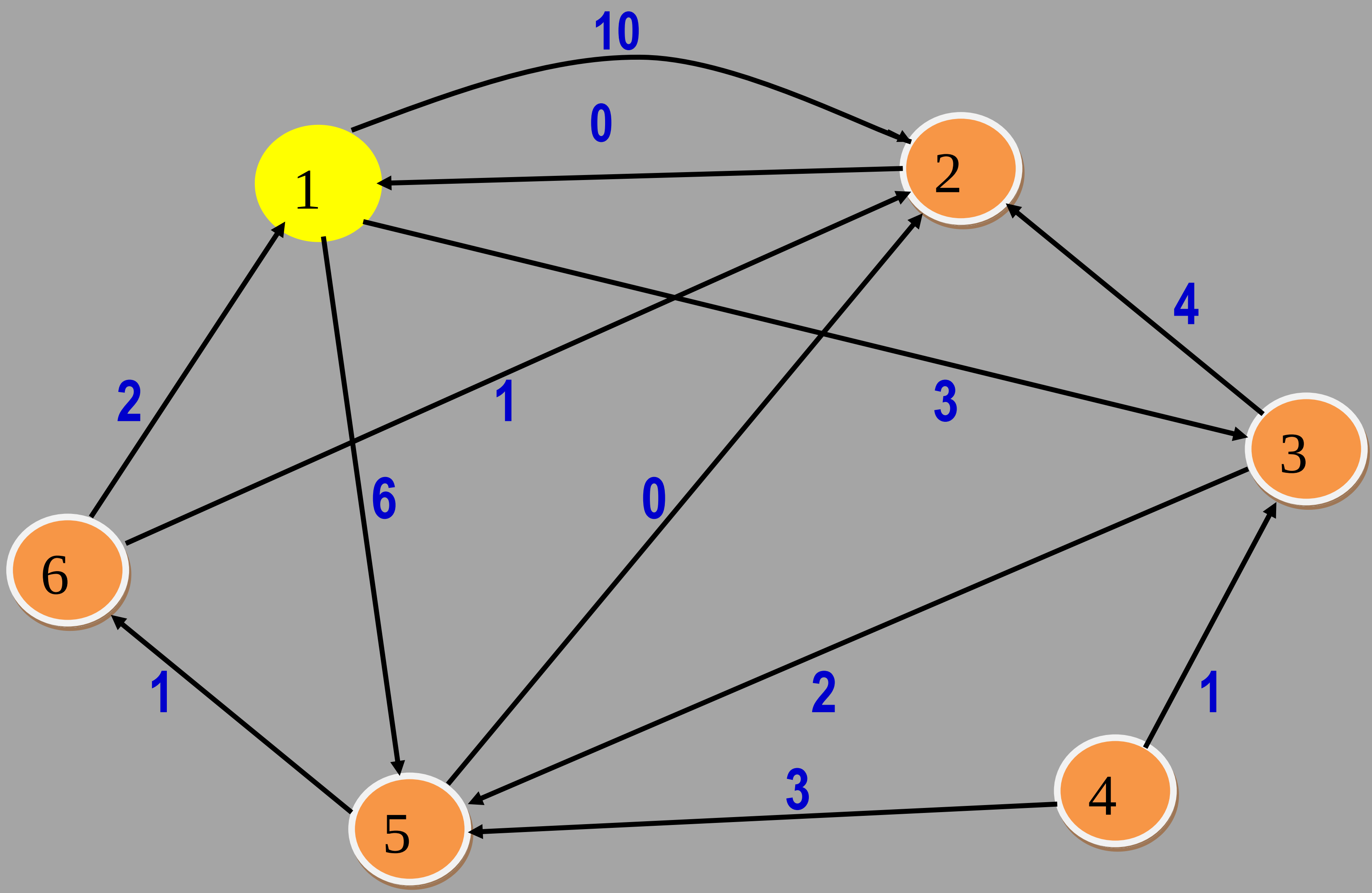
$$G' = (S', A')$$

La construction de G' part d'un sommet **source s**.

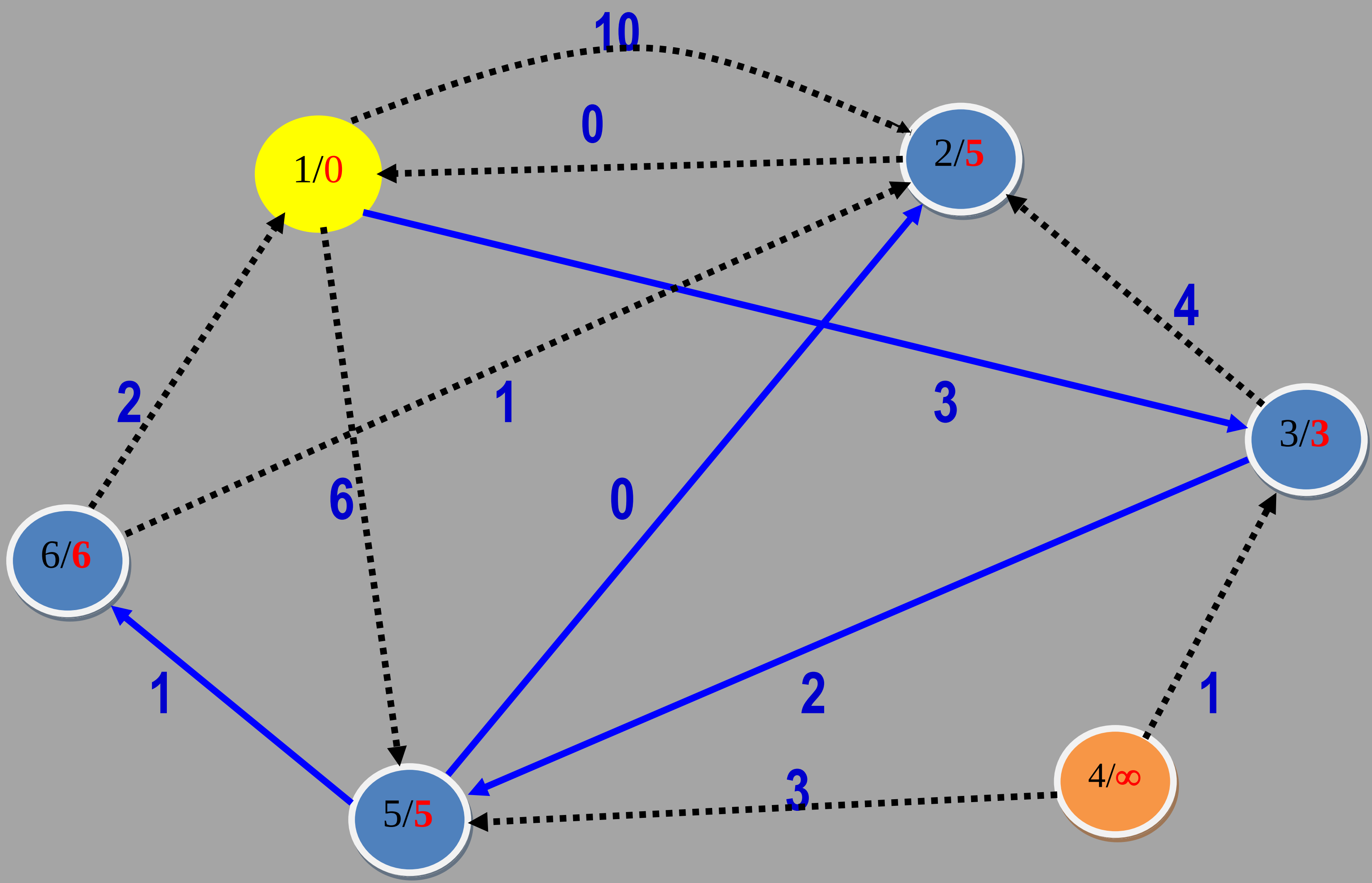
Elle se fait progressivement de telle sorte que la **distance** entre un sommet **x** de G’ et **s** :

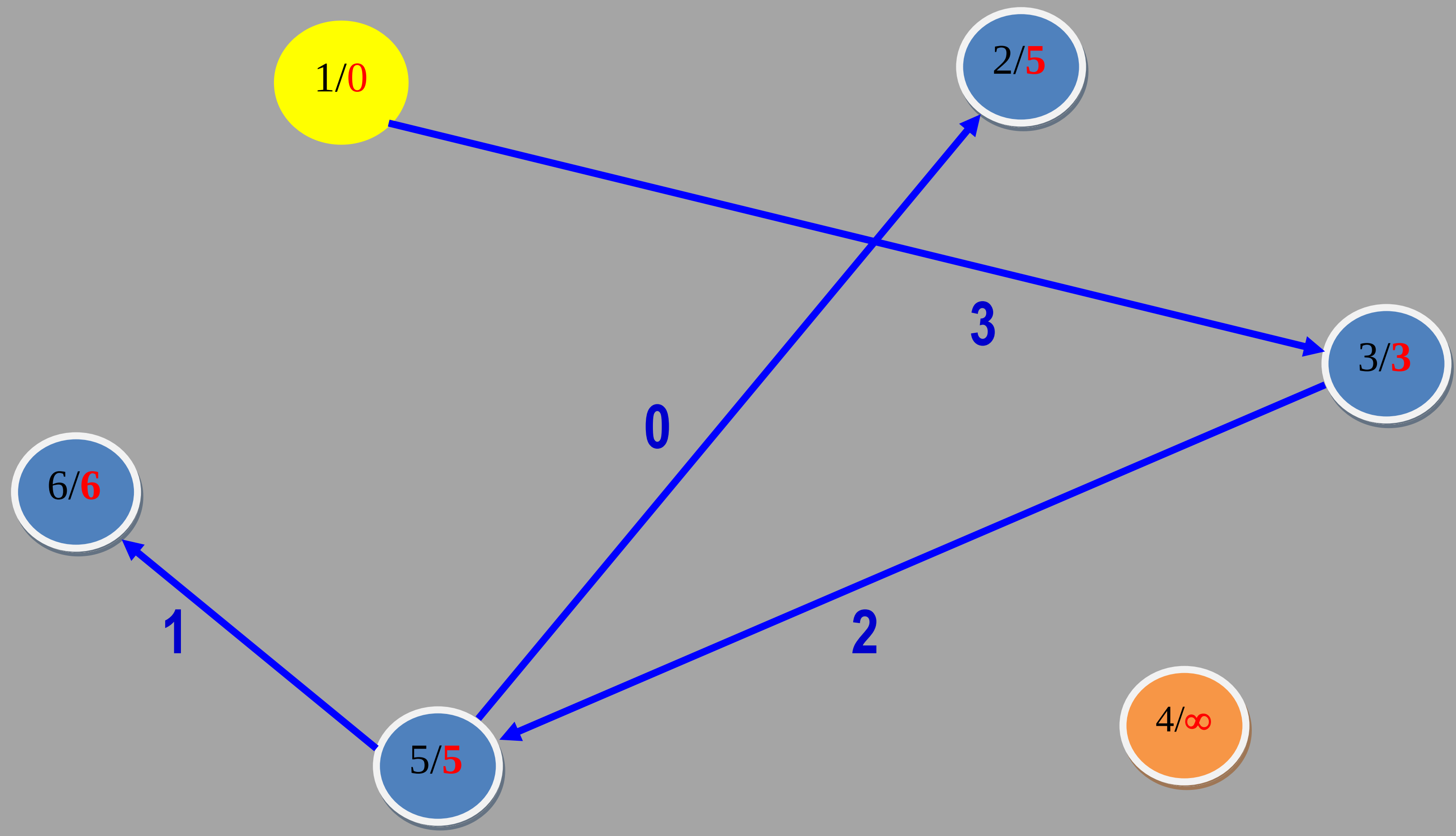
- soit **connue**,
- et soit **minimale** dans G.

Graphe G de source 1



Graphe G' de source 1




1/0
2/5
3/3
6/6
5/5
4/∞
3
0
1
2

3- procédure

3.1- Extension de la définition de coût(x ,y)

On étend la définition de **coût( )** pour qu'elle soit définie pour tout couple (x,y) de sommets de G :

coût : SOMMET x SOMMET $\rightarrow$ ENTIER

Pré_coût(x,y) $\Leftrightarrow$ $x \in S \wedge y \in S$

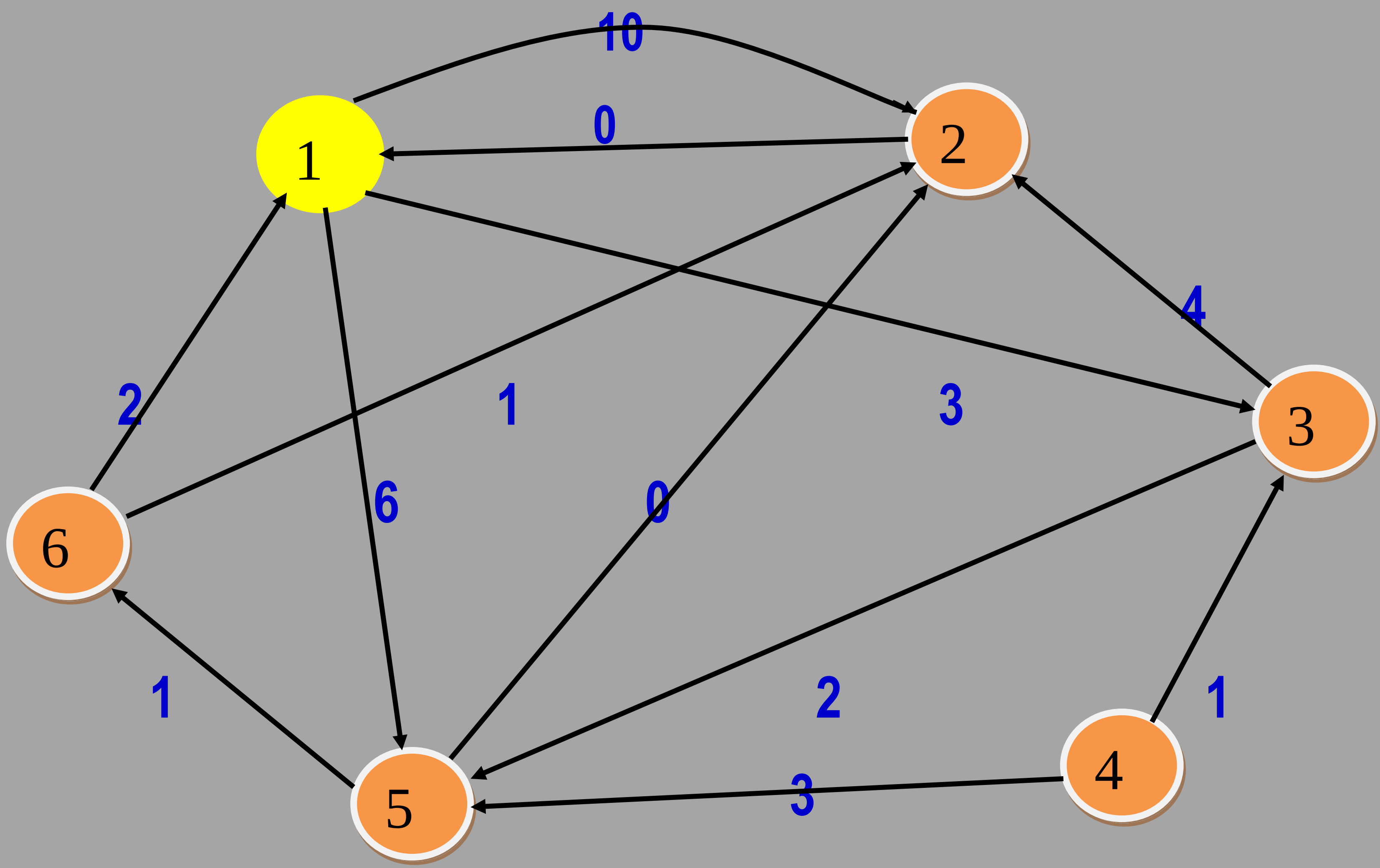
```
coût(x,y)

début

  si x = y alors coût :=0; / *graphe sans boucle*/
  sinon
          si ¬(x→y∈G) alors coût := ∝;    /*∝ signifie «le plus grand possible»*/
          sinon coût :=c(x,y);      /c(x,y) désigne le coût de l'arc x→y
        fin_si
  fin_si
fin   ;
```

Calcul de coût(1,x)

Graphe G de source 1



x	1	2	3	4	5	6
COUT(1,x)	0	10	3	∝	6	∝

3.2- Initialisation de l'algorithme

Initialement, on considère que:

- **G'** contient **seulement** le sommet **s, ** isolé,
- et la distance de **s** à lui-même vaut **0**.

```
initialiser()
  Début
          pour chaque sommet x ∈ S
/* on initialise la distance des sommets autres que s à coût(s,x) * /
               distance[x] ←coût(s,x) ;
/* au départ le seul prédécesseur connu est s lui-même*/
               prédecesseur[x]←s ;
     fin_pour
               distance[s]←0;     /*s étant à une distance 0 de lui-même*/
     S'←{s}   /* initialement, G' ne contient que s */
  Fin
```

Le tableau **distance** est tel que **distance[x]** désigne la distance optimale **connue à une étape** entre **s** et x.

Le tableau **prédécesseur** est tel que **prédécesseur[x]** désigne le meilleur prédécesseur **connu à une étape** de x.

Les tableaux **distance[]** et **prédécesseur[]** seront **mis à jour** à chaque étape de l'algorithme.

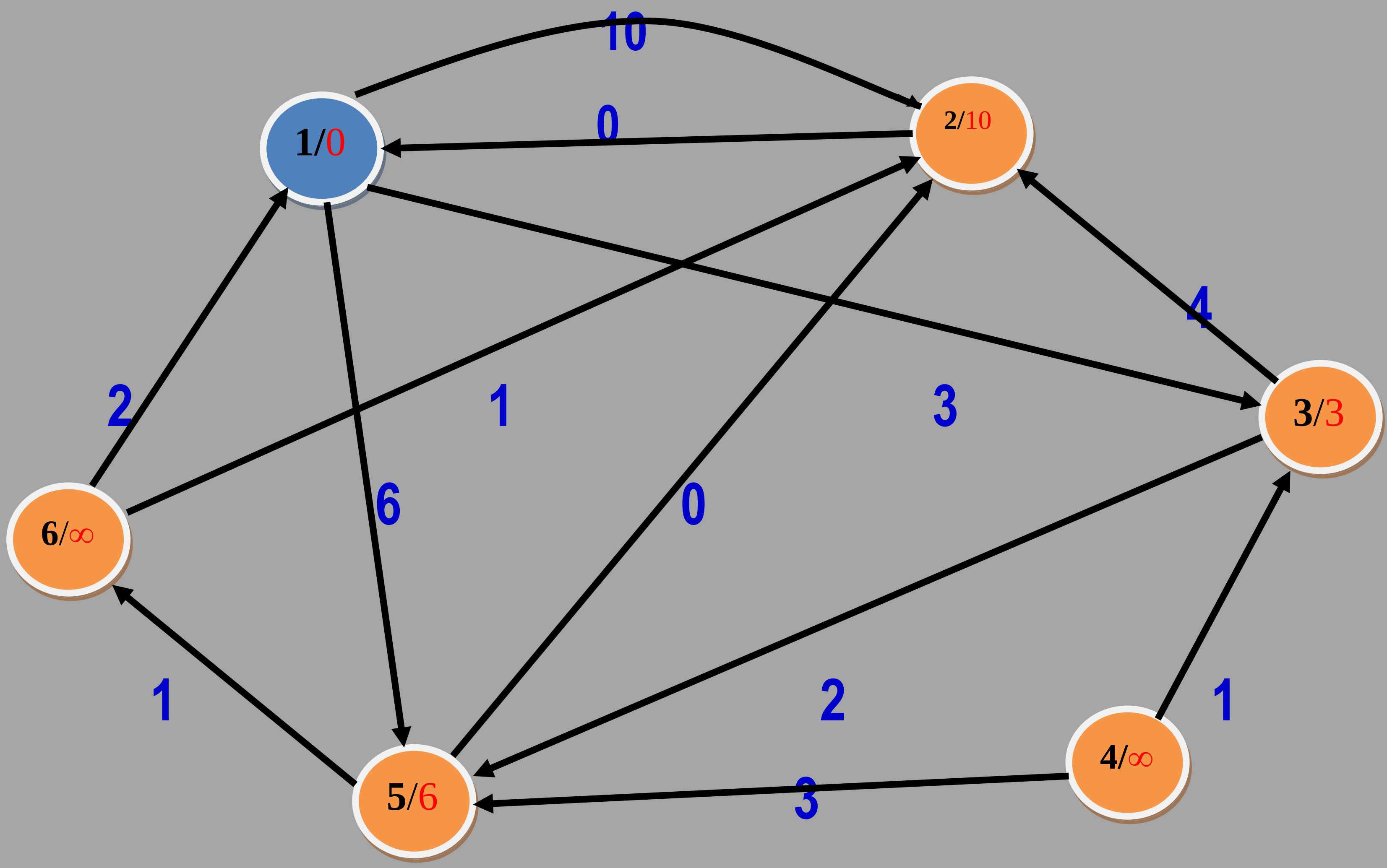
Application de initialiser()

distance[ ]

1	2	3	4	5	6
0	10	3	∞	6	∞

prédécesseur[ ]

1	2	3	4	5	6
1	1	1	1	1	1


1/0
2/10
3/3
4/∞
5/6
6/∞
10
0
4
2
1
3
6
0
1
2
1
3

3.3- Construction de G'

Des arcs de G sont ajoutés à G' étape par étape.

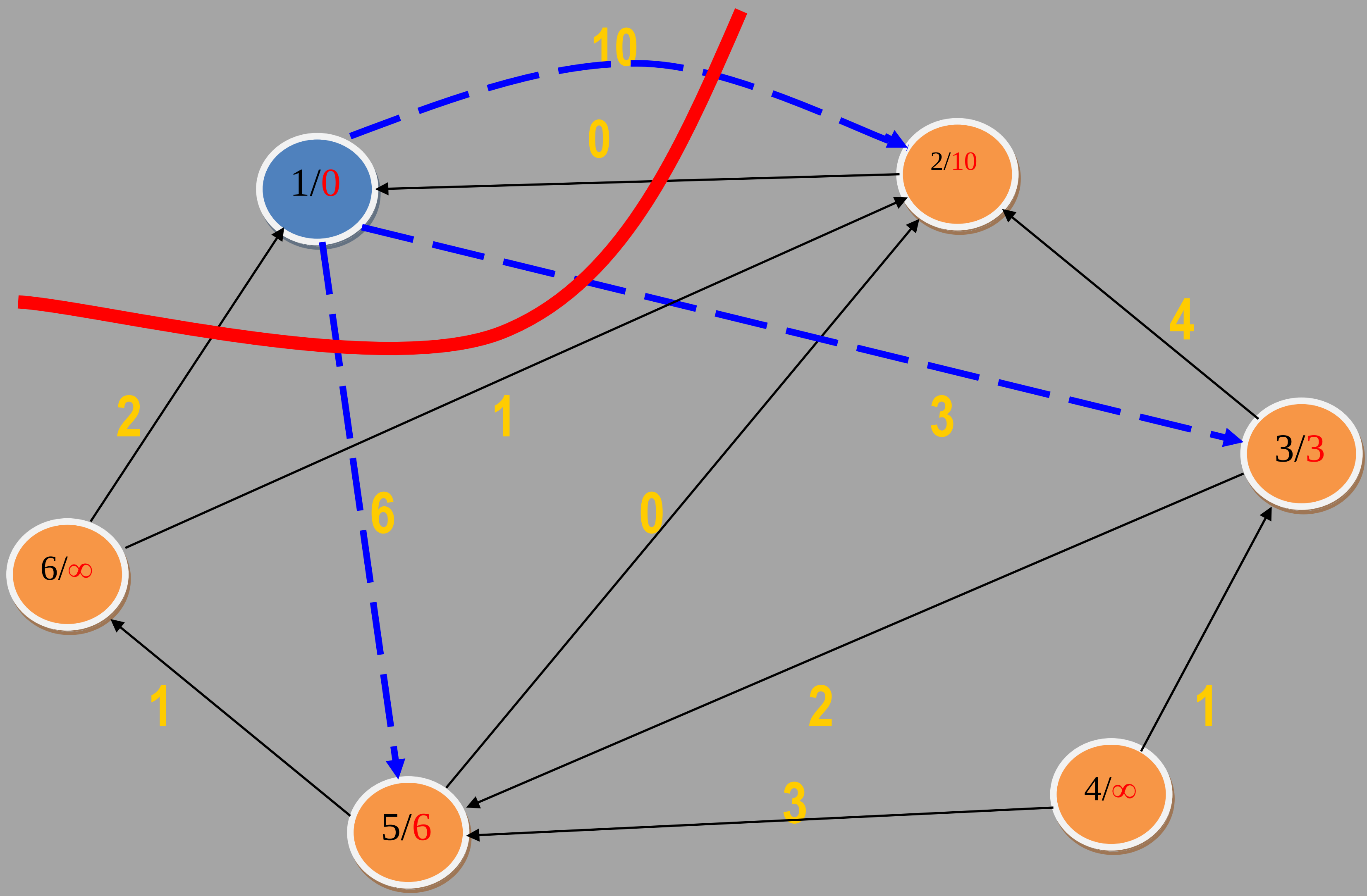
Chaque étape commence par identifier tous les arcs

$$a_i = (x'_i \rightarrow x_i)$$

tels que :

$$x'_i \in S' \quad \wedge \quad x_i \in S\text{-}S'$$

Identification des arcs a_i



Ensuite, parmi les *arcs* $\boldsymbol{a_i}$, elle choisit l'arc a :

$$\boldsymbol{a} = (x' \rightarrow x)$$

tel que x soit «*le plus proche*» de **s**:

$$\forall\ \mathbf{y} \in S\text{-}S' \quad \text{distance}[x] < \text{distance}[\mathbf{y}]$$

L'algorithme se termine quand l'arbre G' de racine **s** **couvre tous** les sommets de G **atteignables** en partant de **s**.

Choix du «plus proche»

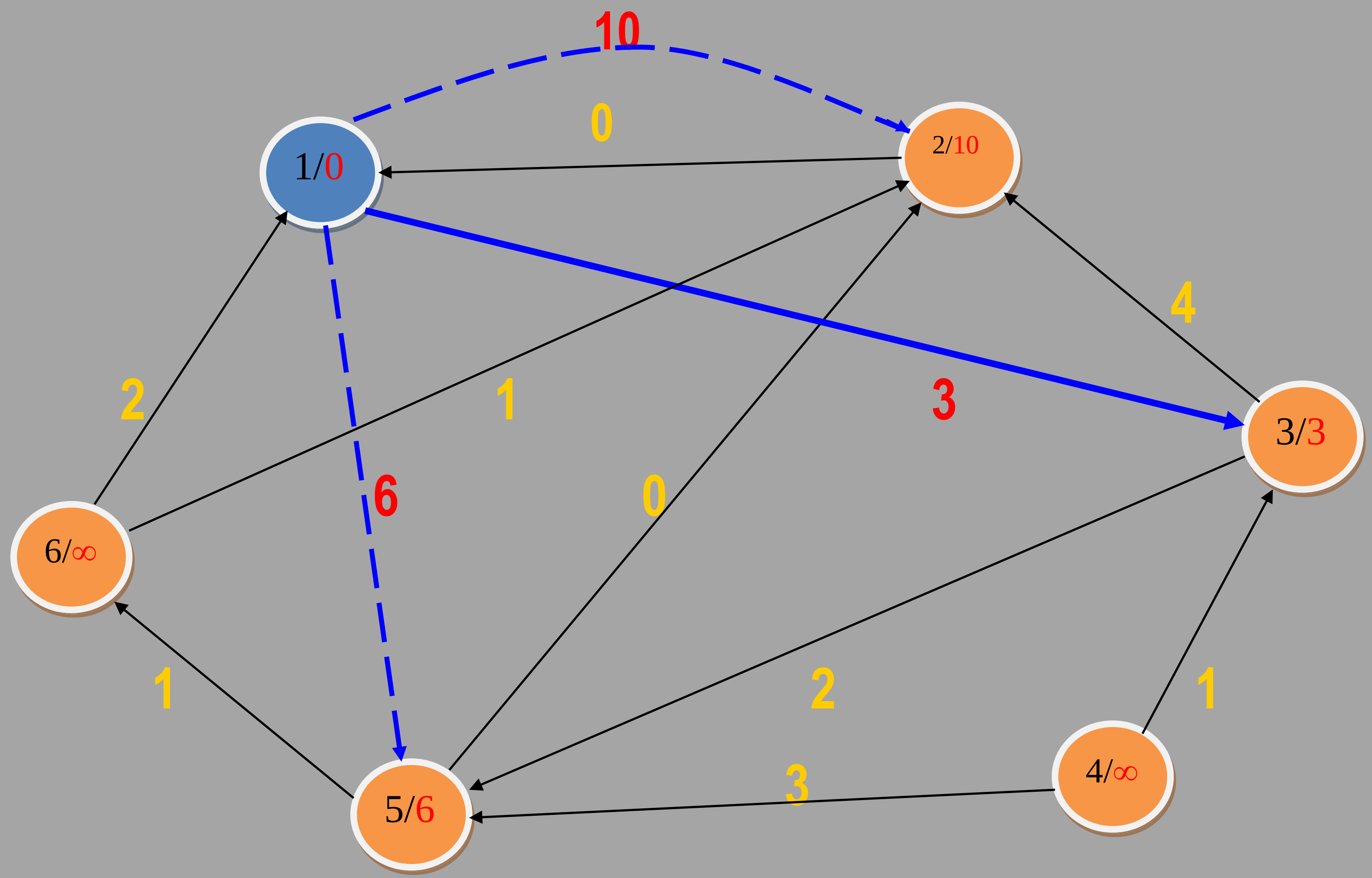
Choisir l'arc $a = x' \rightarrow x$ revient à choisir le sommet ***x*** tel que :

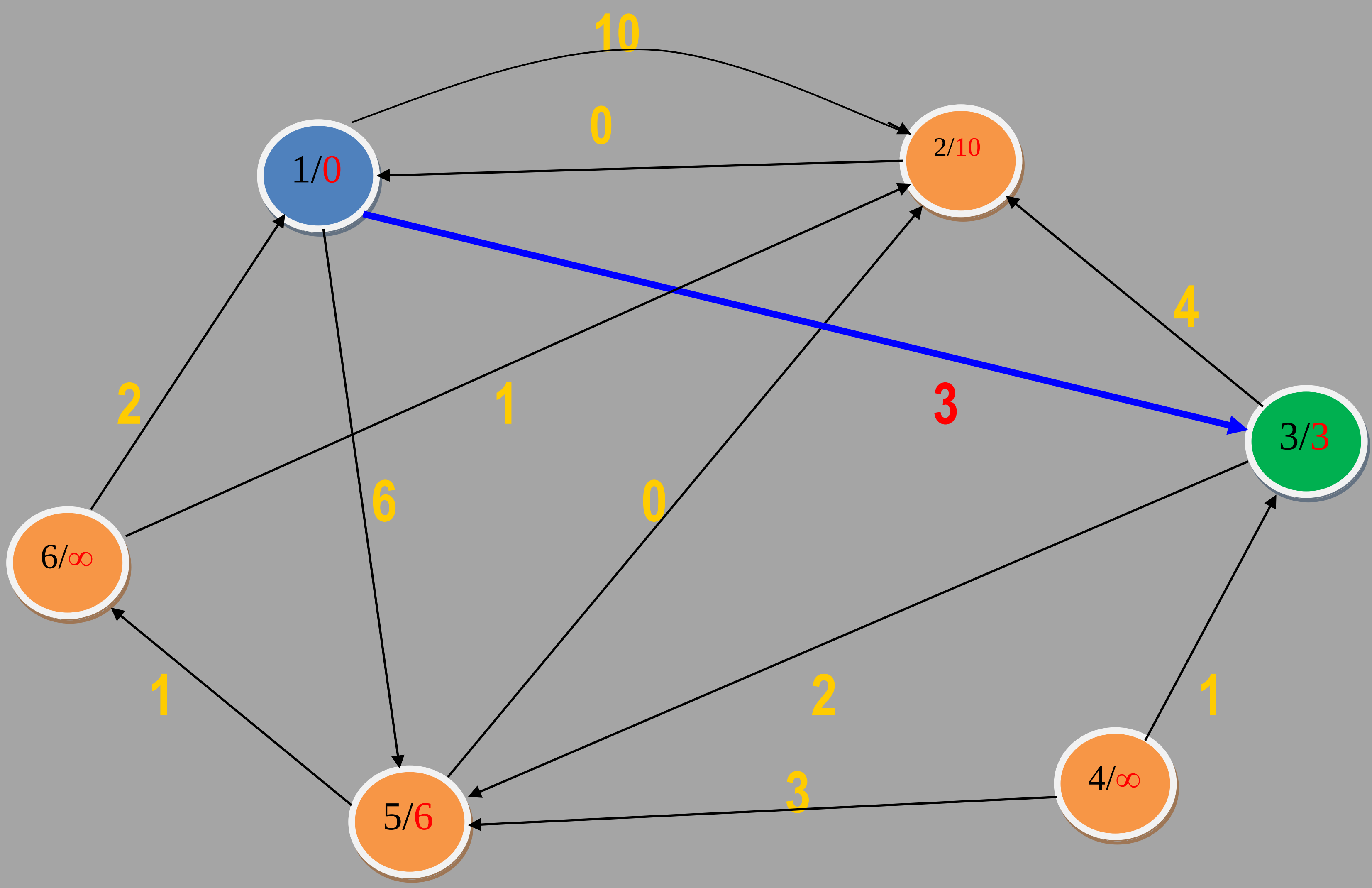
- $x \in S\text{-}S'$,
- $\forall\ y \in S'$ distance [x] < distance[y]

Pour cela, on définit la fonction **plus_proche().**

```
plus_proche():SOMMET
    Début
        choisir x∉S'
        Pour chacun des sommets y ∉S'
        Si distance[y]< distance[x]
            alors x←y ;
    fin_pour
  retourne x;
    Fin    ;
```

Choix de l'arc **a**




10
0
1/0
2/10
4
2
1
3
3/3
6
0
6/∞
1
2
1
3
4/∞
5/6

Test de «meilleur prédécesseur»

Est-on sûr que le chemin traversant l'arc **a** soit le «plus court chemin » entre **s** et x ?

D'après Dijkstra, la réponse est affirmative si *x* est le «meilleur prédécesseur» d'un sommet **y**:

$$\mathbf{y} \in S\text{-}S'.$$

Soit distance[**y**] la distance depuis **s** de **y** ; cela signifie que ***x*** assure d'atteindre **y** via une distance optimale.

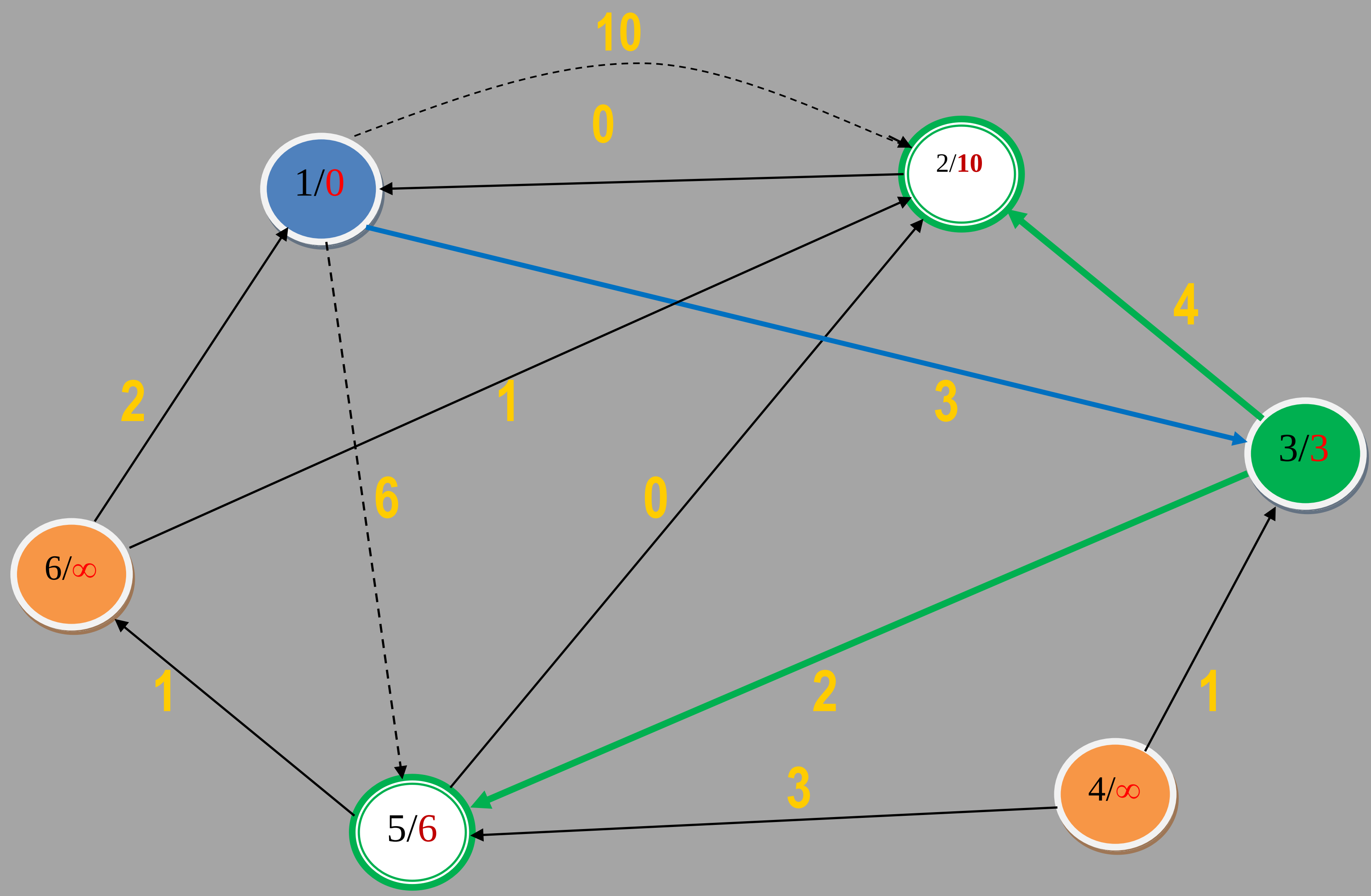
En d'autre terme :

$$\text{distance}[\boldsymbol{x}] + \text{coût}(\boldsymbol{x}, \mathbf{y}) \leq \text{distance}[\mathbf{y}]$$

Le test est réalisé en évaluant la fonction booléenne **meilleur_pred**(***x***, **y**).

```
meilleur_pred(x,y): BOOLEEN
 Début
    ¬ meilleur_pred(x,y) ;
     si distance[x]+coût(x,y)≤distance[y]
     alors  début
              prédécesseur[y]← x ;  /* le plus court chemin  atteignant y passe par x */
              meilleur_pred(x,y);
             fin
     fin_si
 Fin ;
```

meilleur_pred(3,2)



Mise à jour des distances

Si ***x*** est le meilleur prédécesseur de **y** alors on **met à jour** la distance entre ***s*** et **y**:

distance[y] := distance[**x**]+ coût(**x**, y)

Mise à jour des tableaux

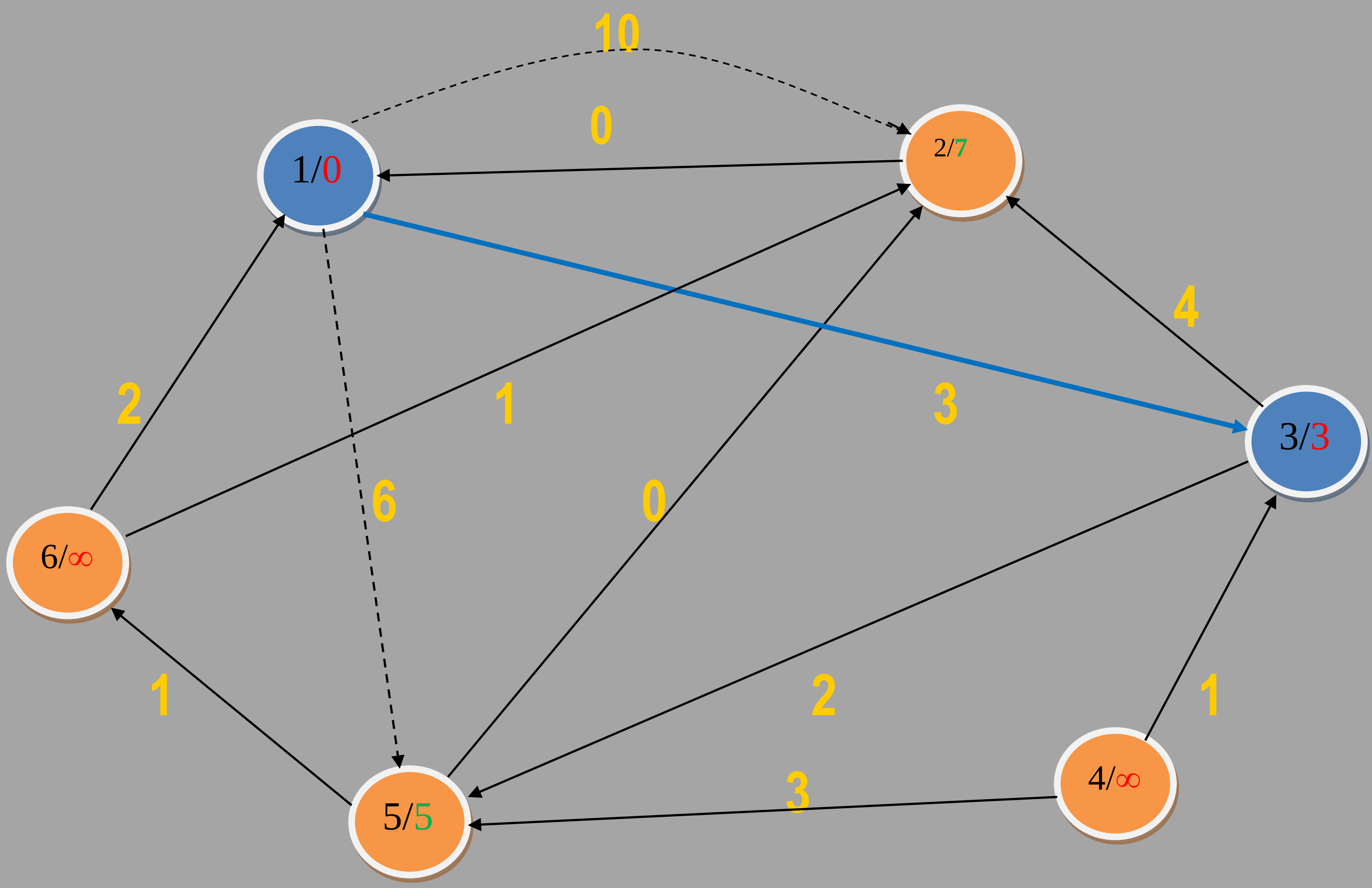
distance[ ]

1	2	3	4	5	6
0	7	3	∞	5	∞

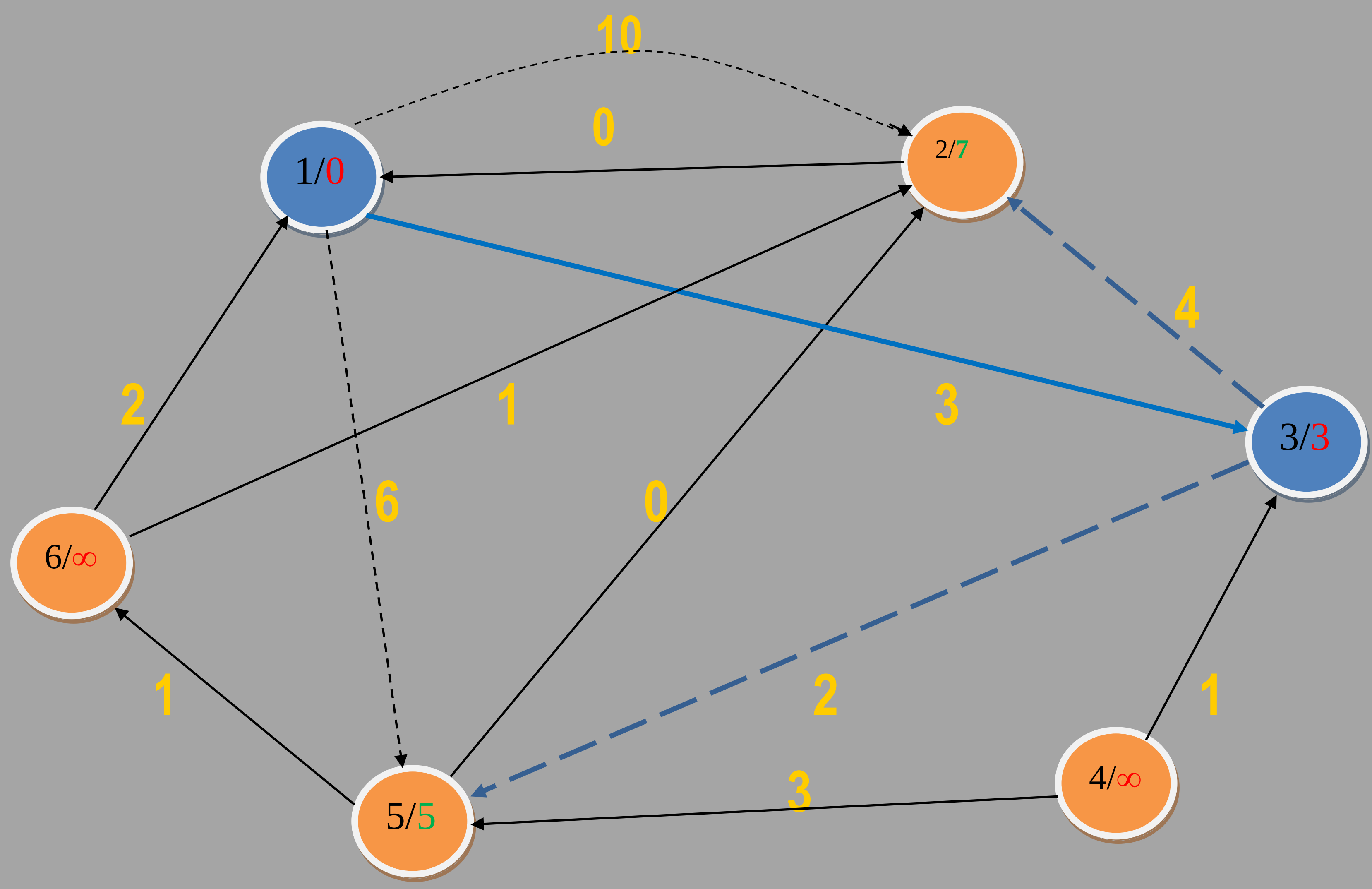
prédécesseur[ ]

1	2	3	4	5	6
1	3	1	1	3	1

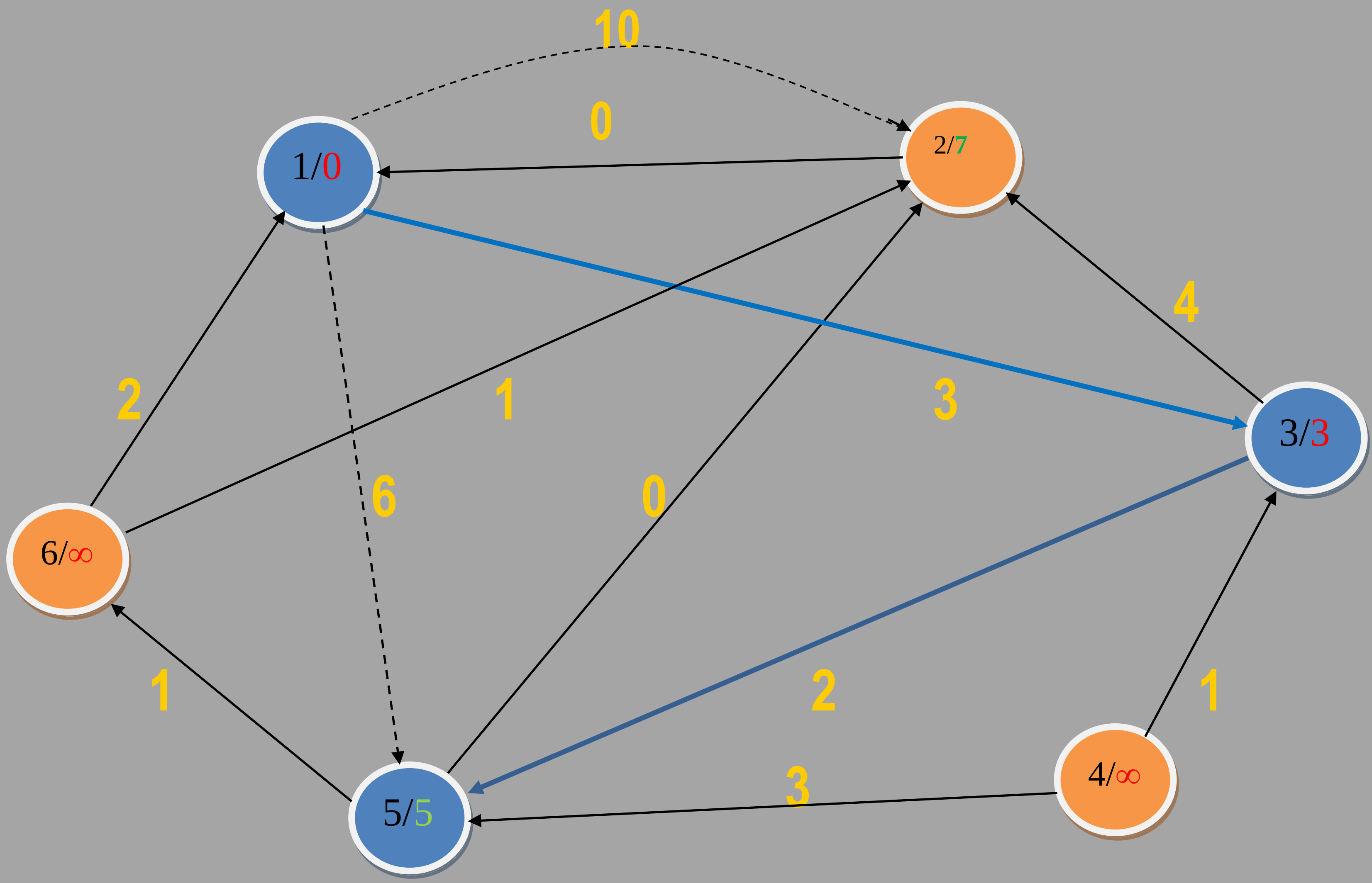
Mise à jour du graphe



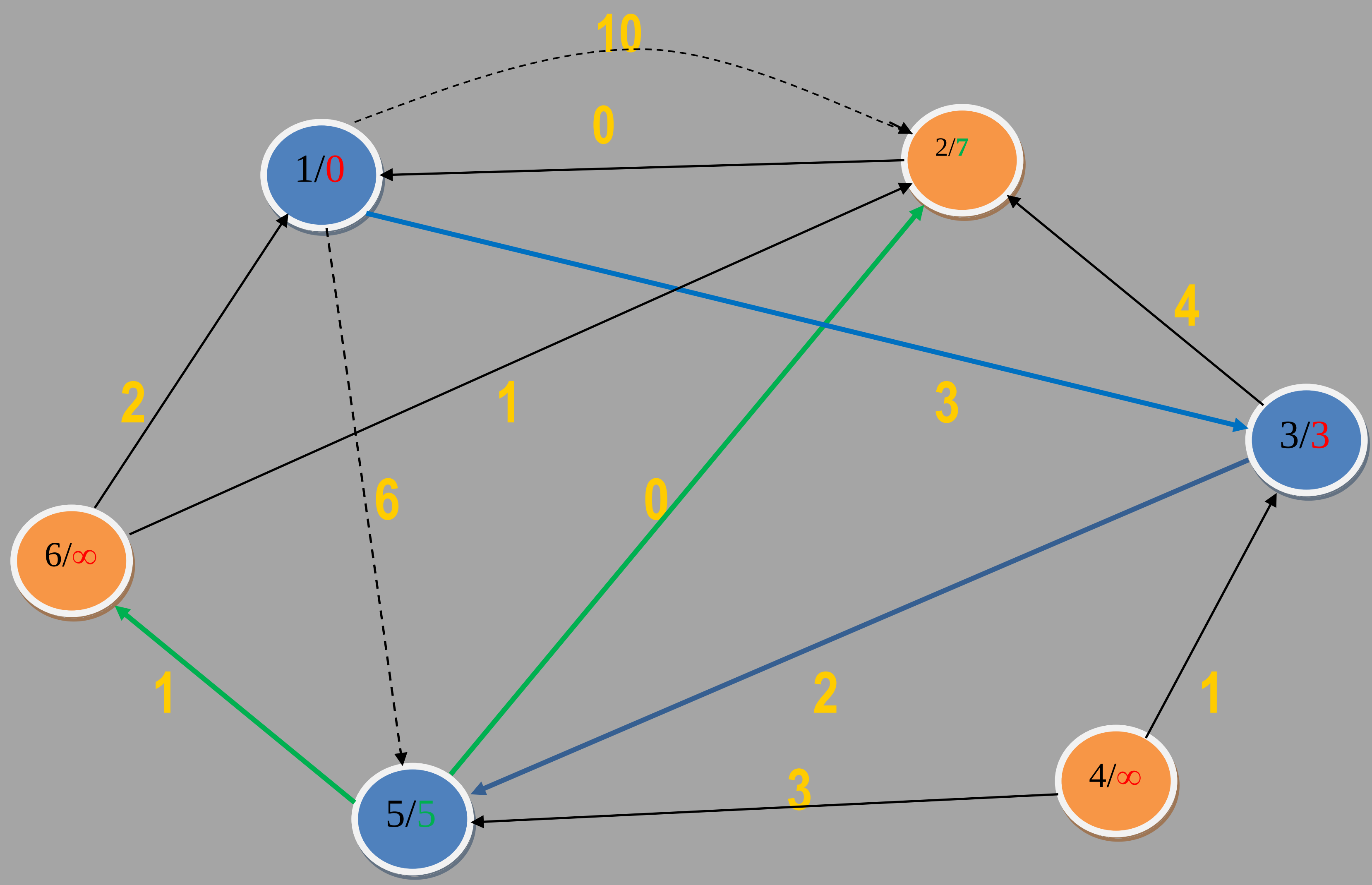
Chercher tous les arcs a_i



Trouver le sommet le plus proche



Meilleur prédécesseur ?



Mise à jour des tableaux

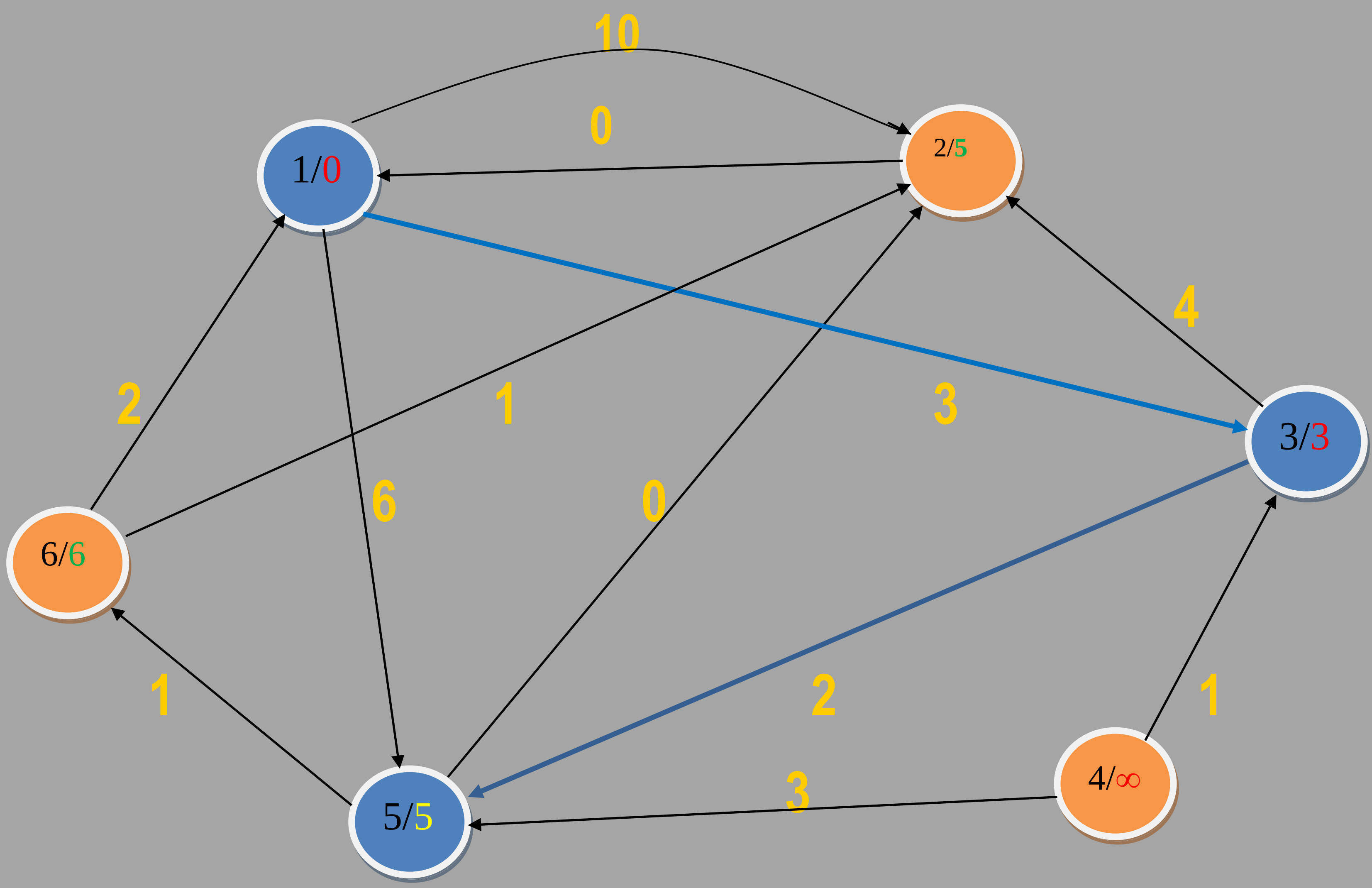
distance[ ]

1	2	3	4	5	6
0	5	3	∞	5	6

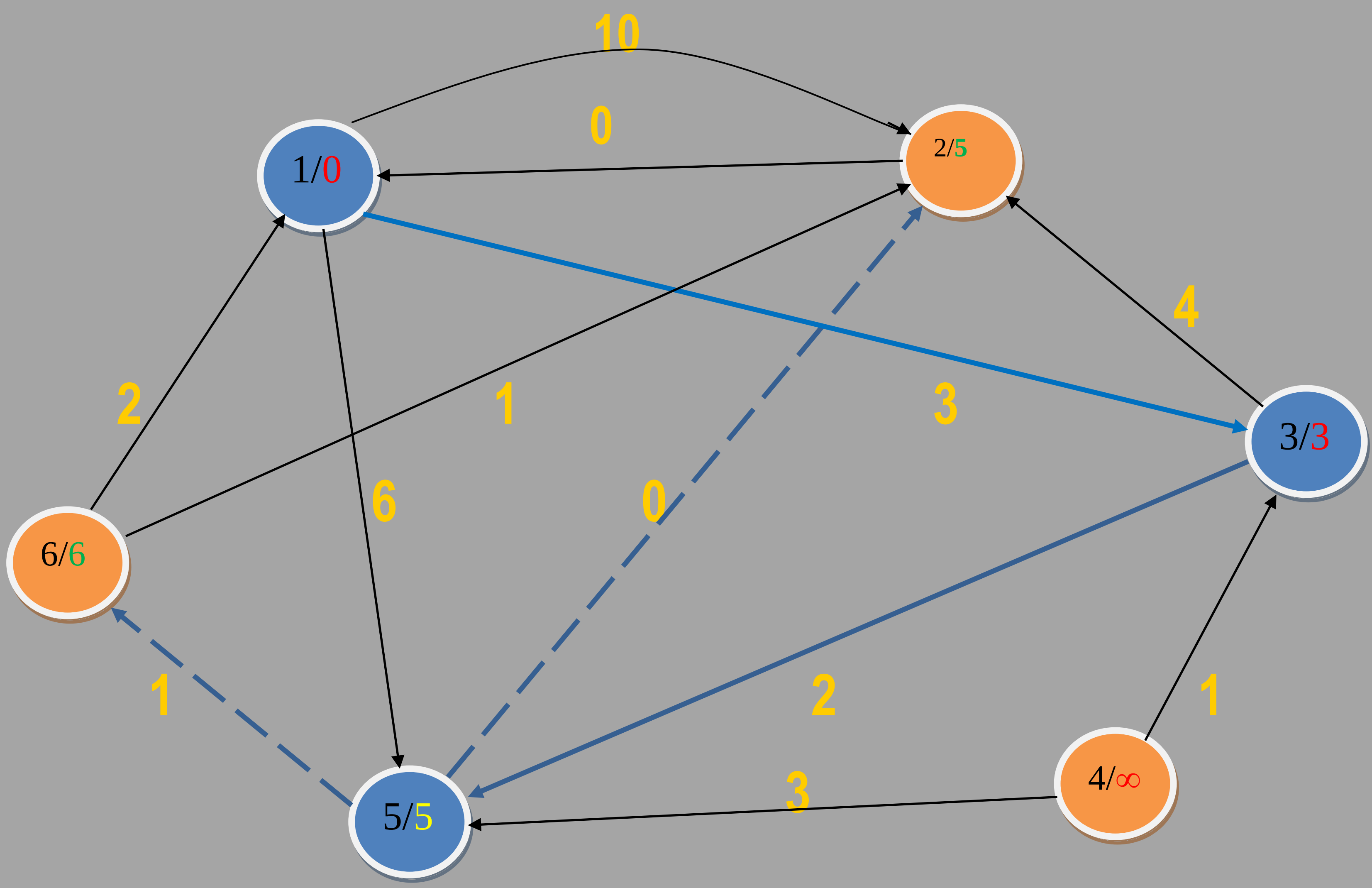
prédécesseur[ ]

1	2	3	4	5	6
1	5	1	1	3	5

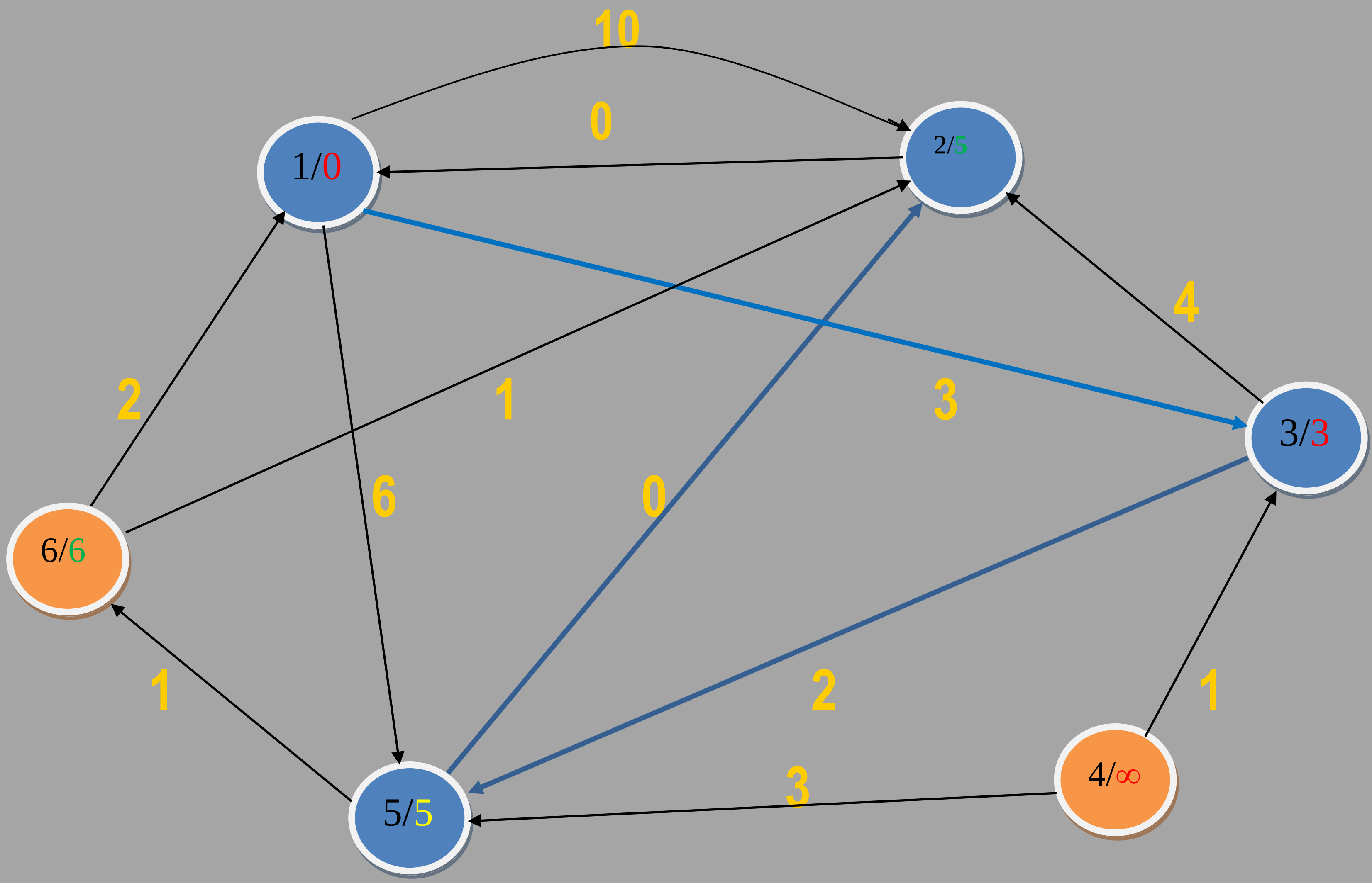
Mise à jour du graphe



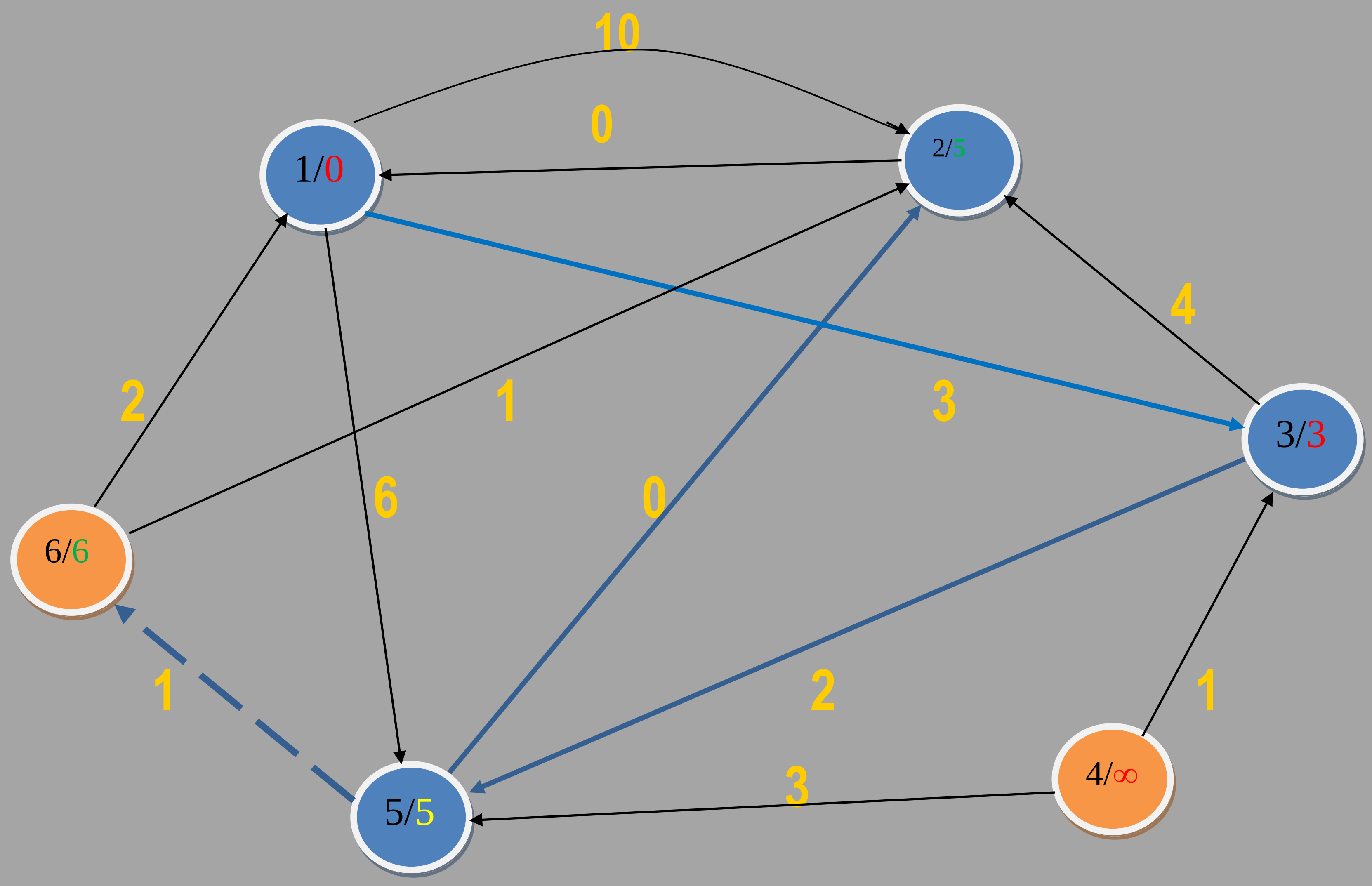
Chercher tous les arcs a_i



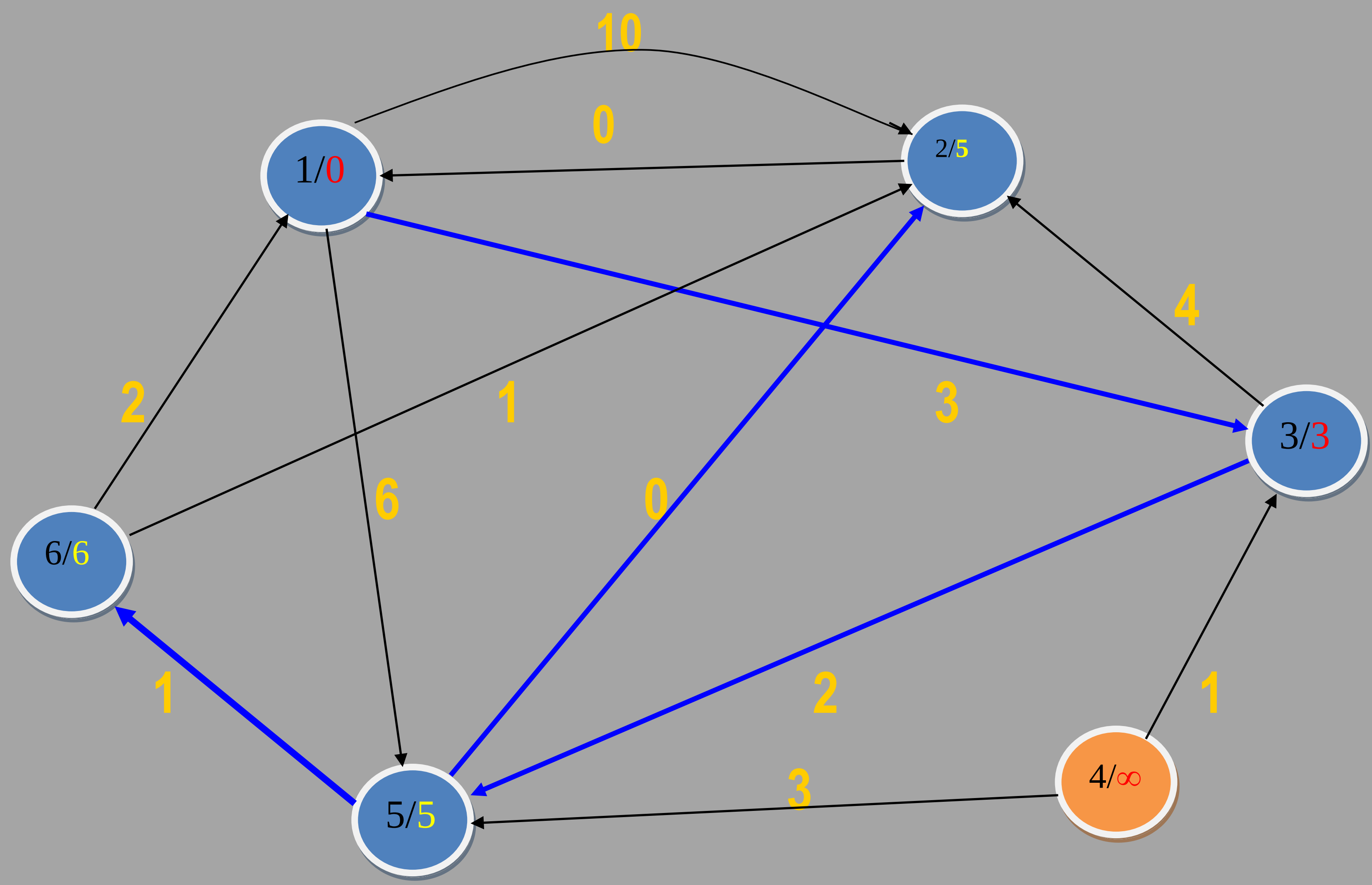
Trouver le sommet le plus proche



Chercher tous les arcs a_i



Trouver le sommet le plus proche



4- Spécification de l'algorithme

Entrées: G = (S,A) : GRAPHE, Coût : SOMMET x SOMMET $\rightarrow$ IR,

s: SOMMET.

Sortie: G' = (S',A') : GRAPHE.

Dijkstra(G,coût,s)

Début

initialiser() ;

*/*prédécesseur[s] ← s et distance[s]←0*/*

```
   Tant que S'≠ S
 /*l'algorithme se termine quand G' devient une arborescence couvrante de G */
           x ←plus_proche() ;
/* recherche d'un sommet x∉S',le plus proche de s */
          Si distance[x] = ∝ alors retour;
   /*car  les sommets qui restent ne sont pas atteignables  à partir de s*/

          S':=S'∪{x};    /*enrichissements successifs de S'*/
          pour i = 1, d°+(x,G)
              y ← ieme_succ(i,x,G) ;
/*si x est le meilleur prédécesseur de y ∉S', alors mettre à jour : distance[y] et prédécesseur[y]*/
              si y∉S' ∧ meilleur_pred(x,y)
              alors distance[y]←distance[x]+ coût(x,y);
              fin_si
        fin_pour
  fin_tant_que
 Fin
```

5- Variante P_1 du problème

Le plus court chemin depuis le sommet ***s*** jusqu'au sommet **x** peut être calculé itérativement.

Dans l'algorithme suivant, Liste_chemin est une liste représentant un plus court chemin de ***s*** à ***x*** :

```
chemin_optimal(G,s,x )  :LISTE
 Début
      Liste_chemin ←listeVide() ;
      y ←x ;
      Liste_chemin ←inserer(Liste_chemin,1,y);
   /* insère x au début de L */
      Tant_que y ≠ s faire
        y ←predecesseur[y] ,
      /* on continue à «remonter » le chemin */

  Liste_chemin←inserer(Liste_chemin,1,y);
  fin_tant_que
  retourner Liste_chemin ;
 Fin ;
```

6- Analyse de la complexité

La complexité de l'algorithme est en **$O(n^2)$** au pire.

Rappel:

Cette complexité est évaluée au pire:

- en nombre total d'opérations,
- en fonction du nombre **n** de sommets et du nombre d'arcs **m** du graphe G.

La procédure **initialiser()** réalise une boucle d'initialisation des tableaux :

distance[ ] et **prédécesseurs[ ]**

Elle comporte un nombre d'affectations d'ordre **O(n)**

Dans la boucle

Tant que $S' \neq S$

Il y a au plus **n-1** itérations.

Le nombre **n-1** correspond au cas où $S' = S$: la source **s** est racine de G.

Le coût de l’opération

x:= plus_proche()

est du même ordre que le nombre de comparaisons effectuées.

Au pire, ce nombre est en O(n) : tous les sommets sont considérés.

Le nombre d'itérations engagées par la boucle :

pour i = 1, **$d°^+(x,G)$**

est majoré par **n-1 :** chaque sommet a au plus n-1 successeurs.(graphe sans boucle, un seul arc incident extérieur vers les autres sommets)

Sa complexité est, au pire, d'ordre O(n)

La **complexité globale** de l'algorithme est donc, au pire:

$$(n-1)\ (\ O(n) + O(n))$$
$$=(n-1)\ O(n)$$
$$= O(n^2)$$

Cette complexité est également d'ordre $O(n^2)$, **dans le meilleur des cas**, pour une représentation par matrice d'adjacence.

III- ALGORITHME DE BELLMAN

Alors que l’algorithme de Dijkstra est valide uniquement pour les **plus courts** chemins.

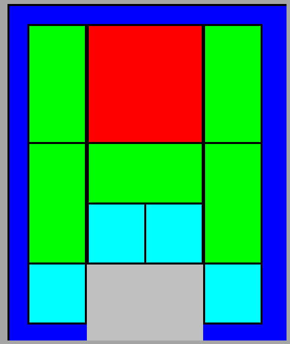
Celui de Bellman fonctionne aussi bien pour un **plus court** chemin que pour un **plus long** chemin.

L'algorithme de Bellman tolère la présence d'arcs de coût **négatif** contrairement à celui de Dijkstra.

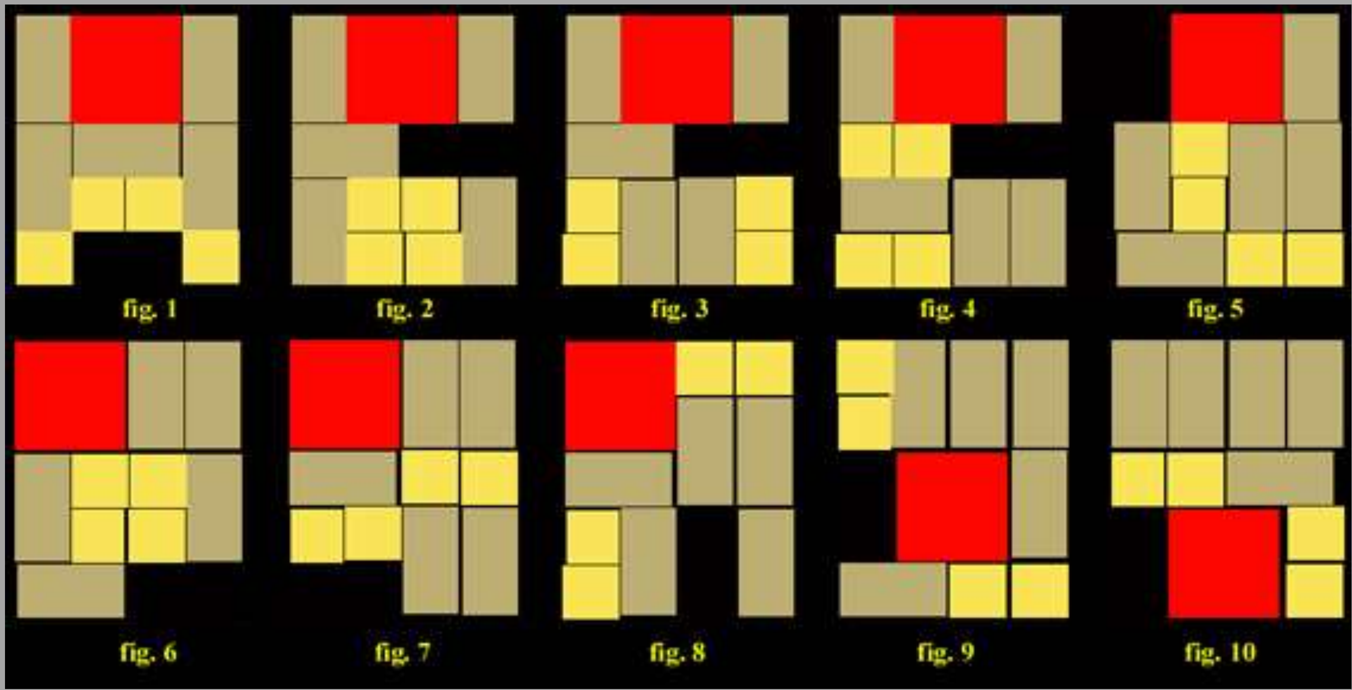
C'est un algorithme **dynamique** qui est souvent plus adapté que celui de Dijkstra pour résoudre des problèmes en RO.

Il est aussi appliqué à la navigation par GPS

Le célèbre taquin de l' «âne rouge»



Il n'existe pas de solution demandant **moins de 81** coups (ci-dessous 115 coups)



Il est aussi appliqué aux réseaux informatiques, pour déterminer le cheminement des messages : protocole de routage RIP.

La **complexité** de l'algorithme est, dans le pire des cas, en :

Θ(mn) pour un graphe courant,

- $\Theta(n^3)$ pour un graphe simple très dense.

1- Idée

L'idée de l'algorithme de Bellman pour la recherche de **chemin optimal** est similaire à celle de Dijkstra.

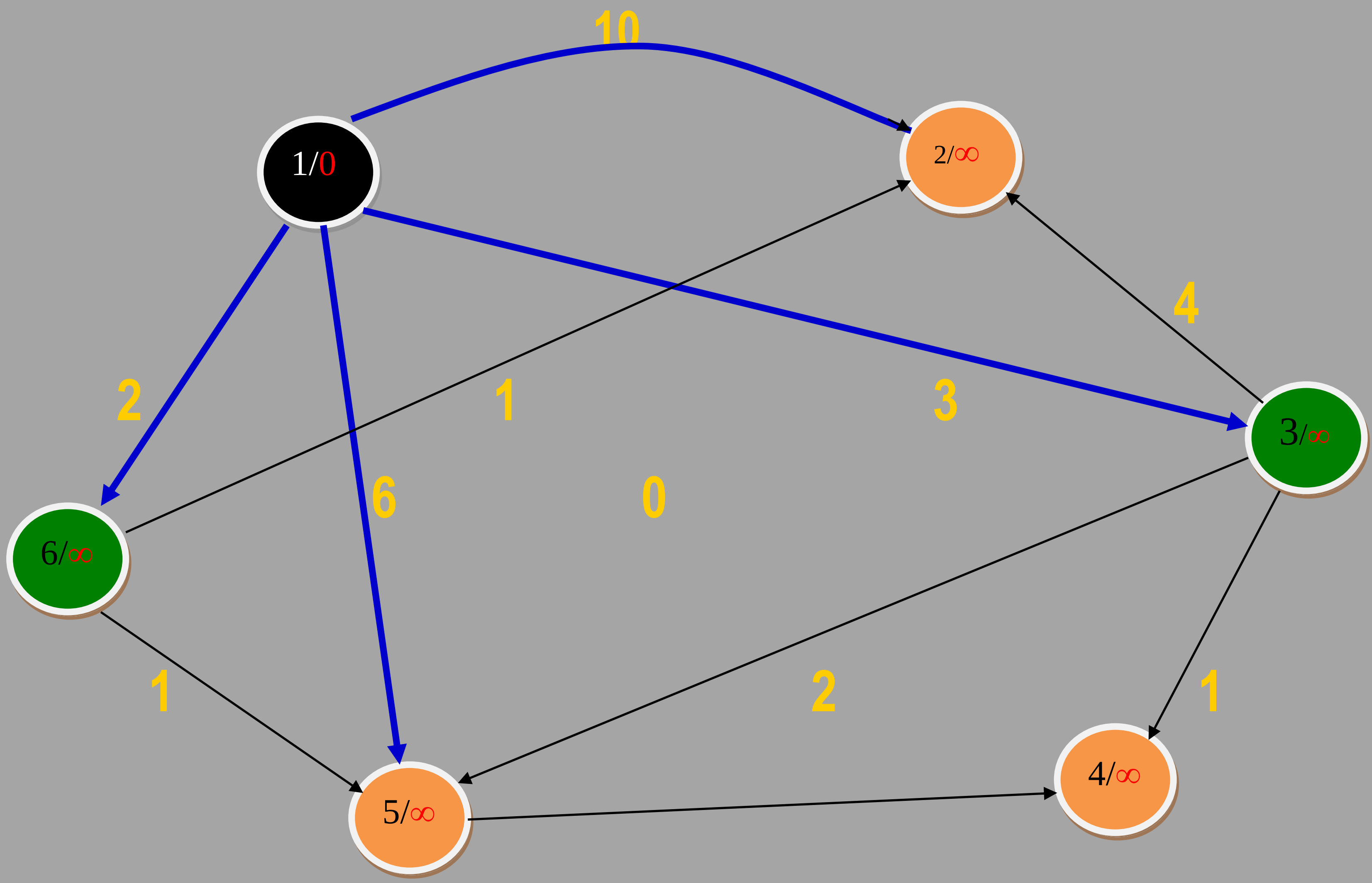
La différence réside dans le choix du sommet à chaque nouvelle itération.

Différence de stratégie !

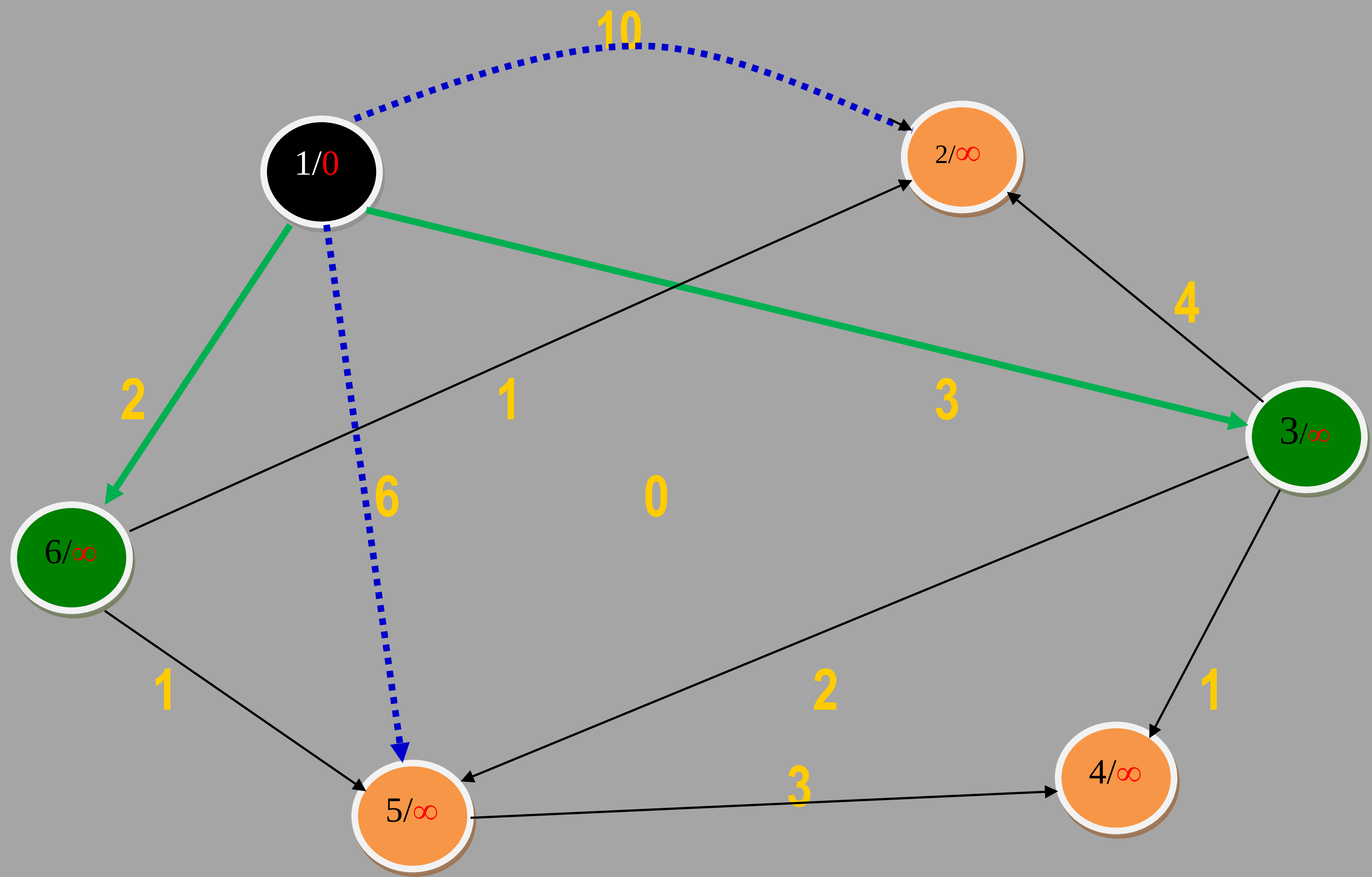
- Dans l'algorithme de Dijkstra, c'est le sommet **«le plus proche»** qui est choisi.

- Alors que dans celui de Bellman, on choisit un sommet dont **«tous les prédécesseurs** ont déjà été **visités»**.

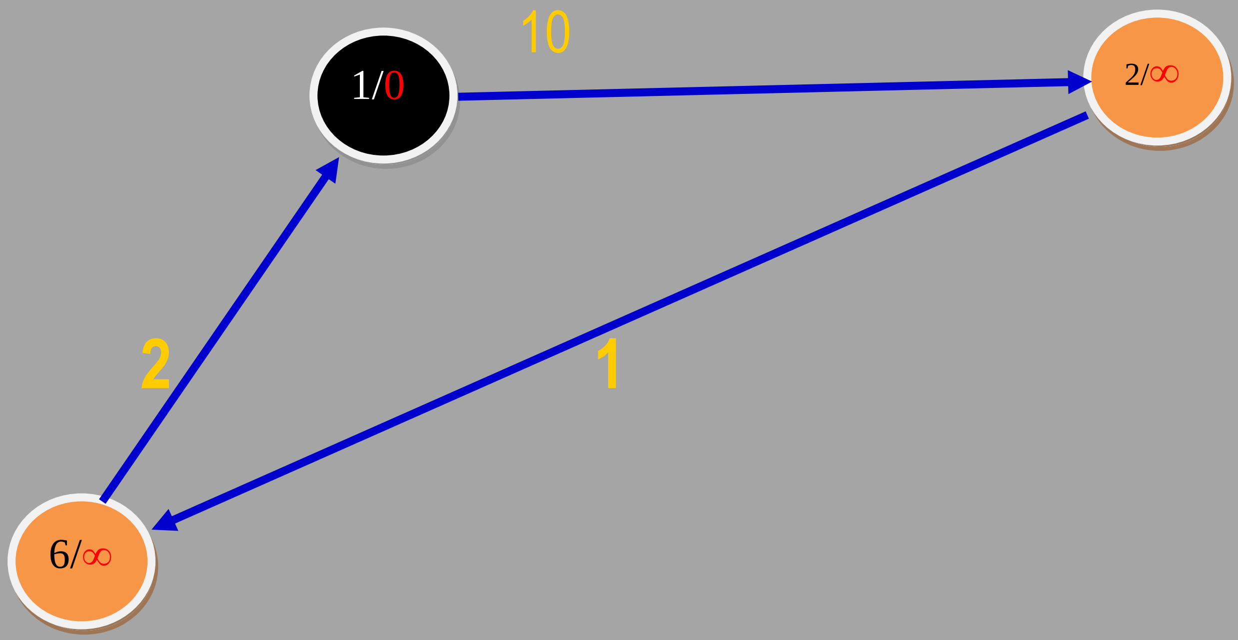
Illustration de la stratégie



Sélectionner un sommet v parmi {3 ,6}



Attention : nécessité de ne pas avoir de **circuit** !



2-Principe

L’algorithme consiste à construire un arbre G’ :

$$G' = (S', A')$$

G’ est arbre couvrant – en général partiellement- G.

La **racine** de G’ est le sommet de départ: s.

A chaque itération, l'algorithme met à jour deux tableaux :

Distance[x] : qui indique la «distance optimale» du sommet **x** depuis **s**.

Predécesseur[x] : qui indique le prédécesseur du sommet **x** en empruntant le «chemin le plus court» depuis **s**.

1-nitialisation

Au départ l'arbre G'(S',$\emptyset$) ne contient que le seul sommet s.

$$S' \leftarrow \{s\};$$

Les tableaux **Distance[ ]** et **Prédecesseur[ ]** sont tels que :

Pour tout $u \in S$

  Distance$[u] \leftarrow +\propto;$

  Predecesseur$[u] \leftarrow nil$

FinPour;

Distance$[s] \leftarrow 0;$

distance[ ]

1	2	3	.	.	n
0	∞	∞	∞	∞	∞

prédécesseur[ ]

1	2	3	.	.	n
-	-	-	-	-	-

2-Choix du sommet v

Choisir un sommet **v** :

-non encore exploré : $\mathbf{v} \in S\text{-}S'$

- dont les tous les prédécesseurs sont dans S'

$$\mathbf{v} \in S\text{-}S' \mid \mathbf{PredG}(\mathbf{v}) \subseteq S'$$

Marquer ce sommet:

$$S' \leftarrow S' \cup \{\mathbf{v}\};$$

Mise à jour des tableaux

Pour tout arc(u,**v**) | u $\in$ S'

Si Distance[**v**] > Distance[u] + cout(u,**v**)

alors

Distance[**v**] $\leftarrow$ Distance[u] + cout(u,**v**);

Predecesseur[**v**] $\leftarrow$ u;

FinSi;

FinPour

3 - Procédure de BELLMAN

<u>Entrées</u>:

G = (S, A, cout) : graphe orienté valué

s : sommet source.

<u>Sorties</u>:

Distance(x) : indique la «distance optimale» du sommet

x depuis **s**.

Predécesseur(x) : indique le prédécesseur du sommet

x si on emprunte le «chemin le plus court» depuis **s**.

Variables intermédiaires:

X et **X'** : deux sous-ensembles de sommets de G,

u et **v** : deux sommets de G,

PredG(x) : ensemble des **prédécesseurs** de **x** dans G.

Bellman(G, s)

Début

/* Initialisation*/

S' ← {s}; X← S-S' ;

Pour tout u∈S

Distance[u] ← ∝;

Predecesseur[u] ← nil

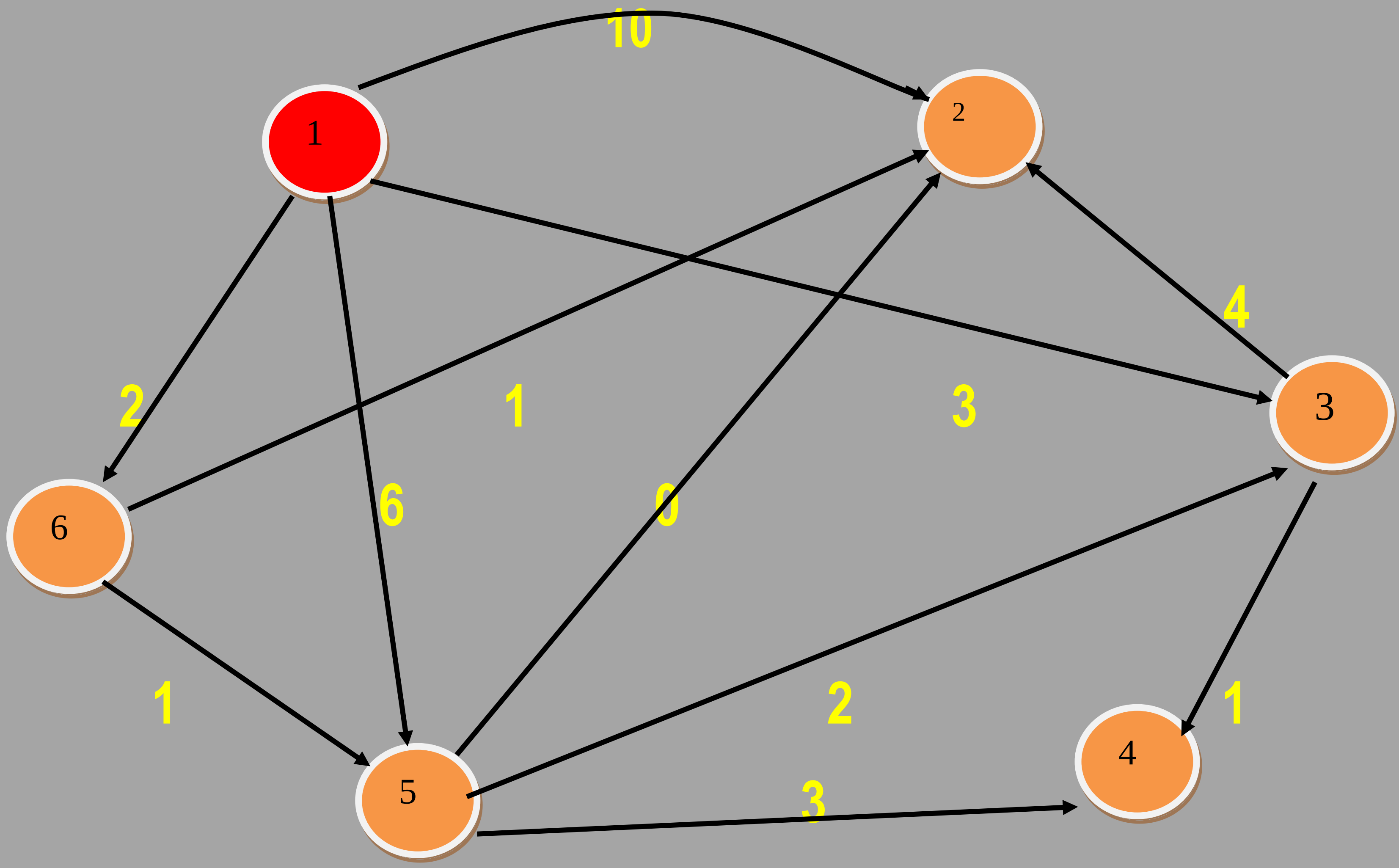
FinPour;

Distance[s]← 0;

/*Fin initialisation*/

```
/*relaxation de l'arc (u,v) */
 Tant que ∃v∈X | PredG(v)⊆ S'
      S'←S'∪{v};   X← X-{v};
      Pour tout(u,v) | u∈S'
         Si Distance[v]>Distance[u]+cout(u,v)
         Alors
               Distance[v]←Distance[u]+cout(u,v);
               Predecesseur[v]←u;
         FinSi;
      FinPour
 FinTq
Fin
```

Application n°1



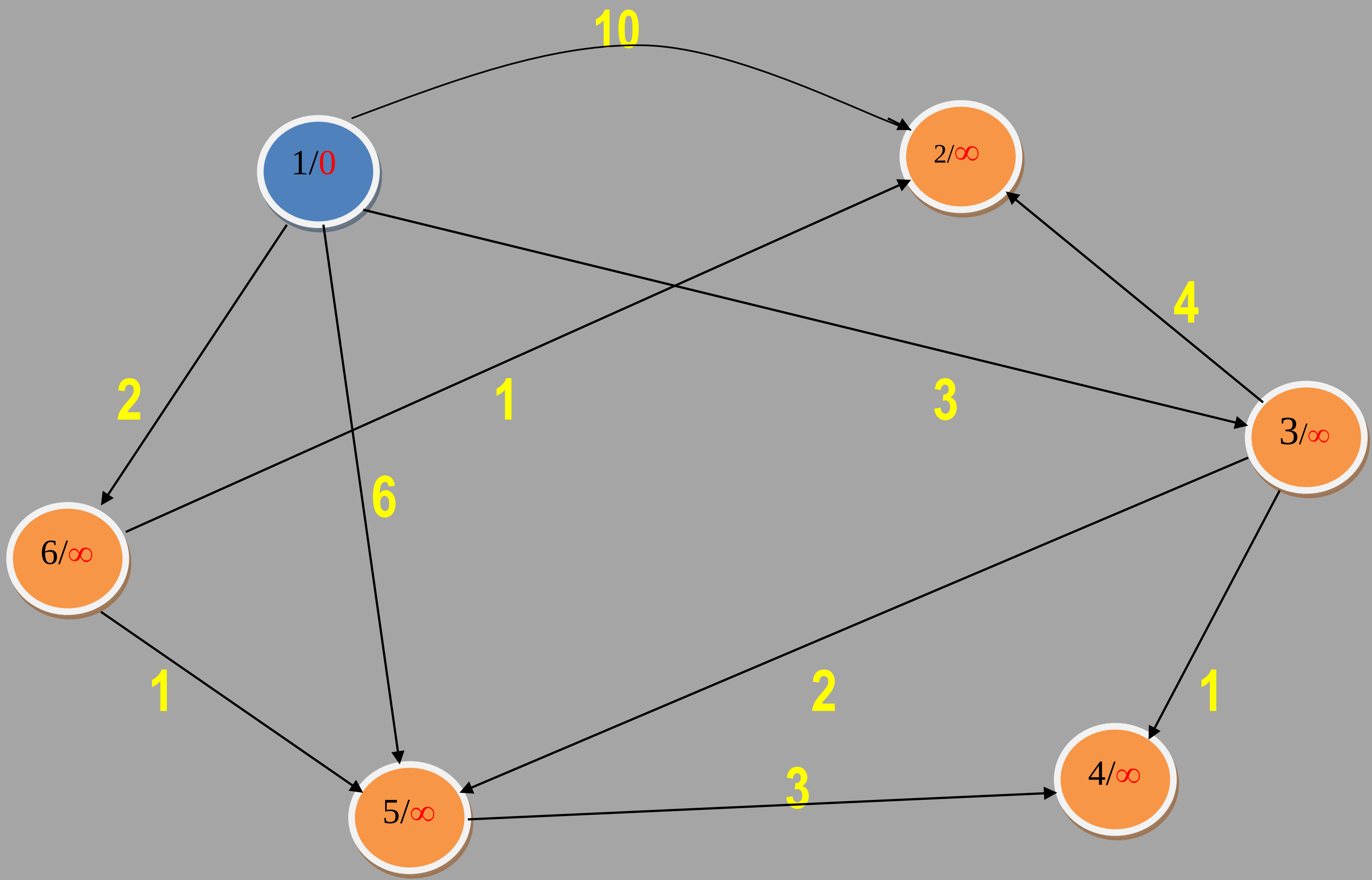
Au départ s$\rightarrow$1

Distance :

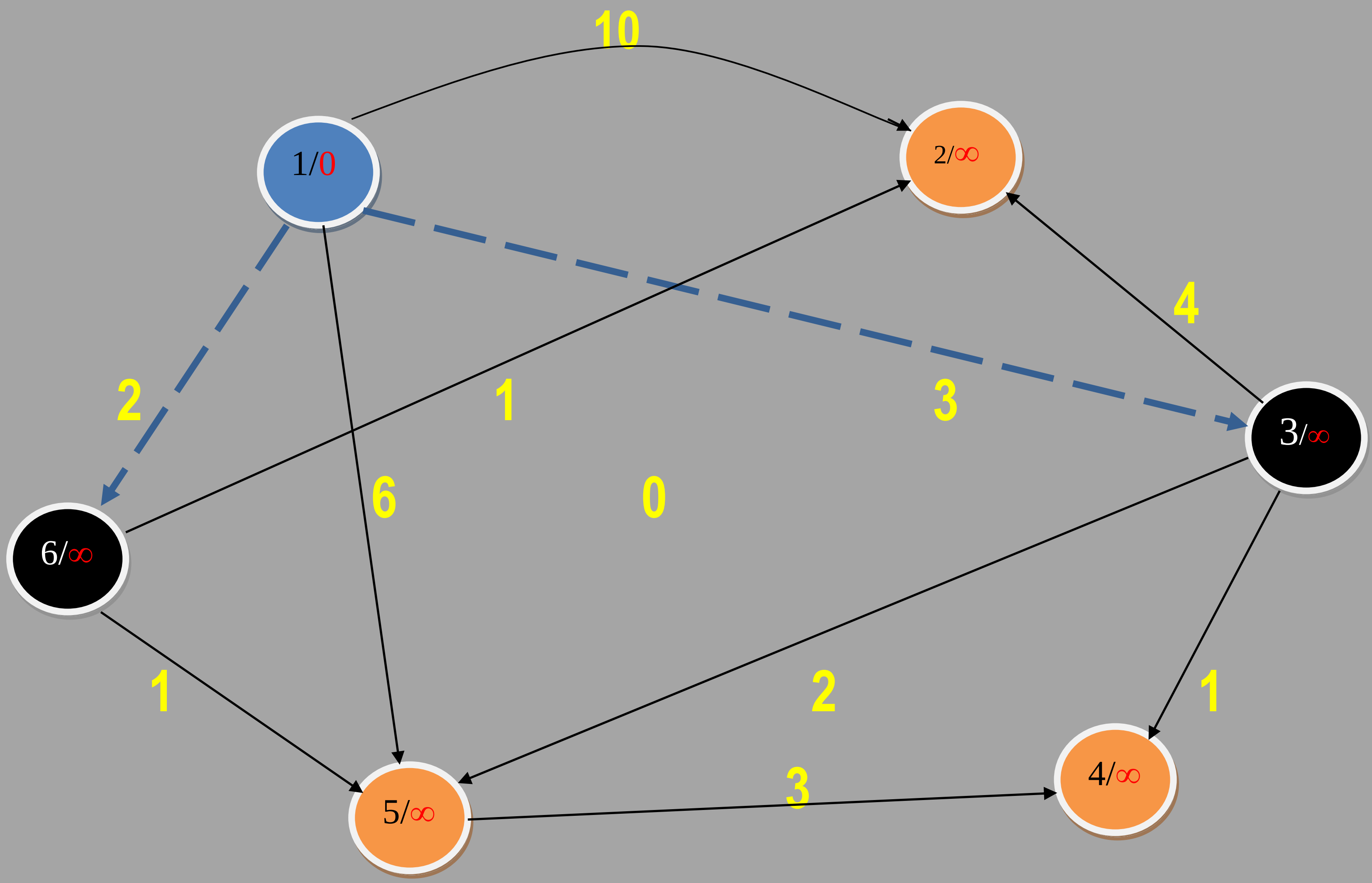
1	2	3	4	5	6
0	+∝	+∝	+∝	+∝	+∝

Predecesseur :

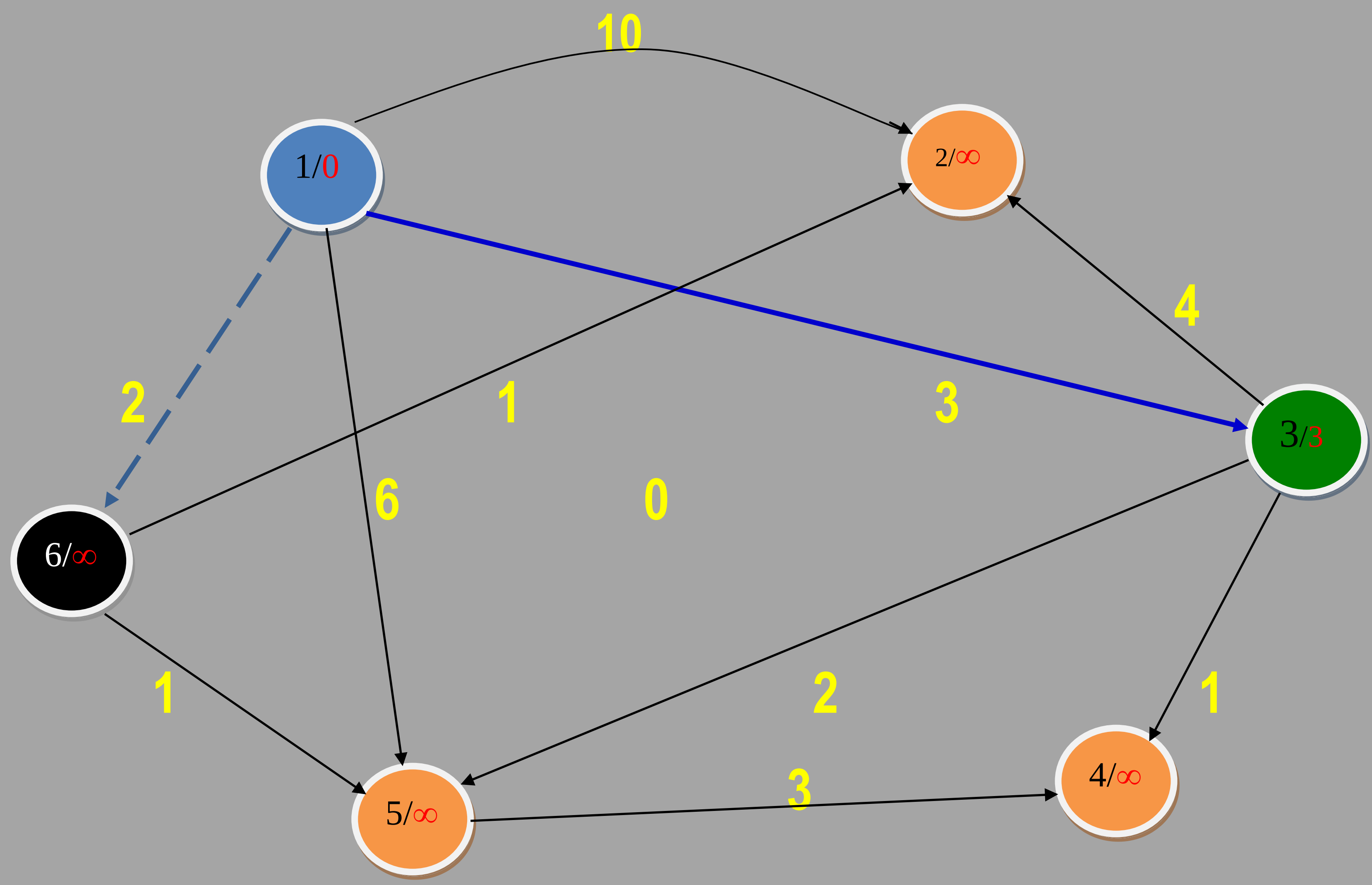
1	2	3	4	5	6
nil	nil	nil	nil	nil	nil


10
1/0
2/∞
4
2
1
3
3/∞
6/∞
6
1
2
1
3
4/∞
5/∞

Sélection de **v** parmi les sommets {3 ,6}



Choix du sommet 3



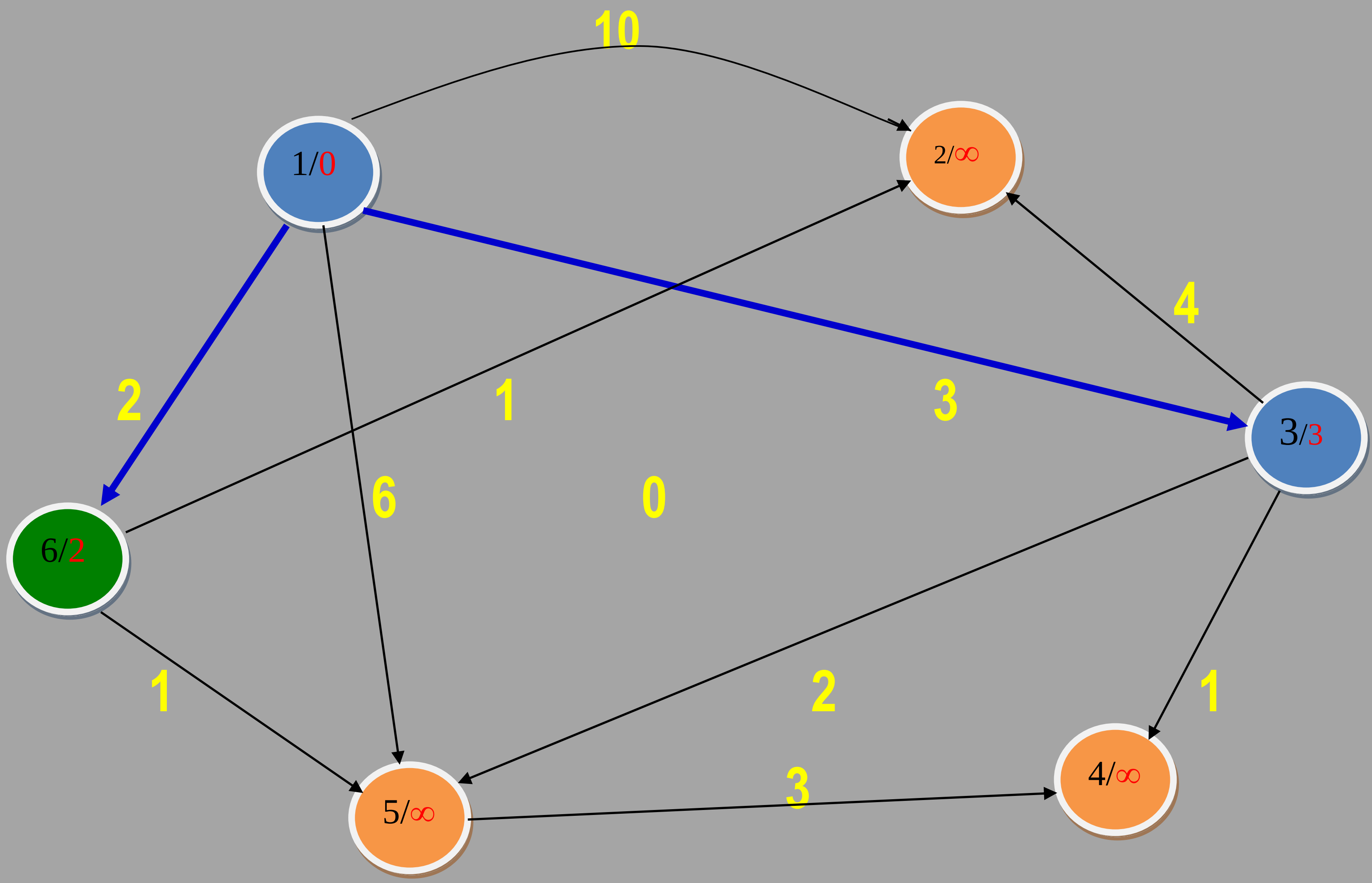
Distance :

1	2	3	4	5	6
0	+∝	3	+∝	+∝	+∝

Predecesseur :

1	2	3	4	5	6
nil	nil	1	nil	nil	nil

Choix du sommet 6



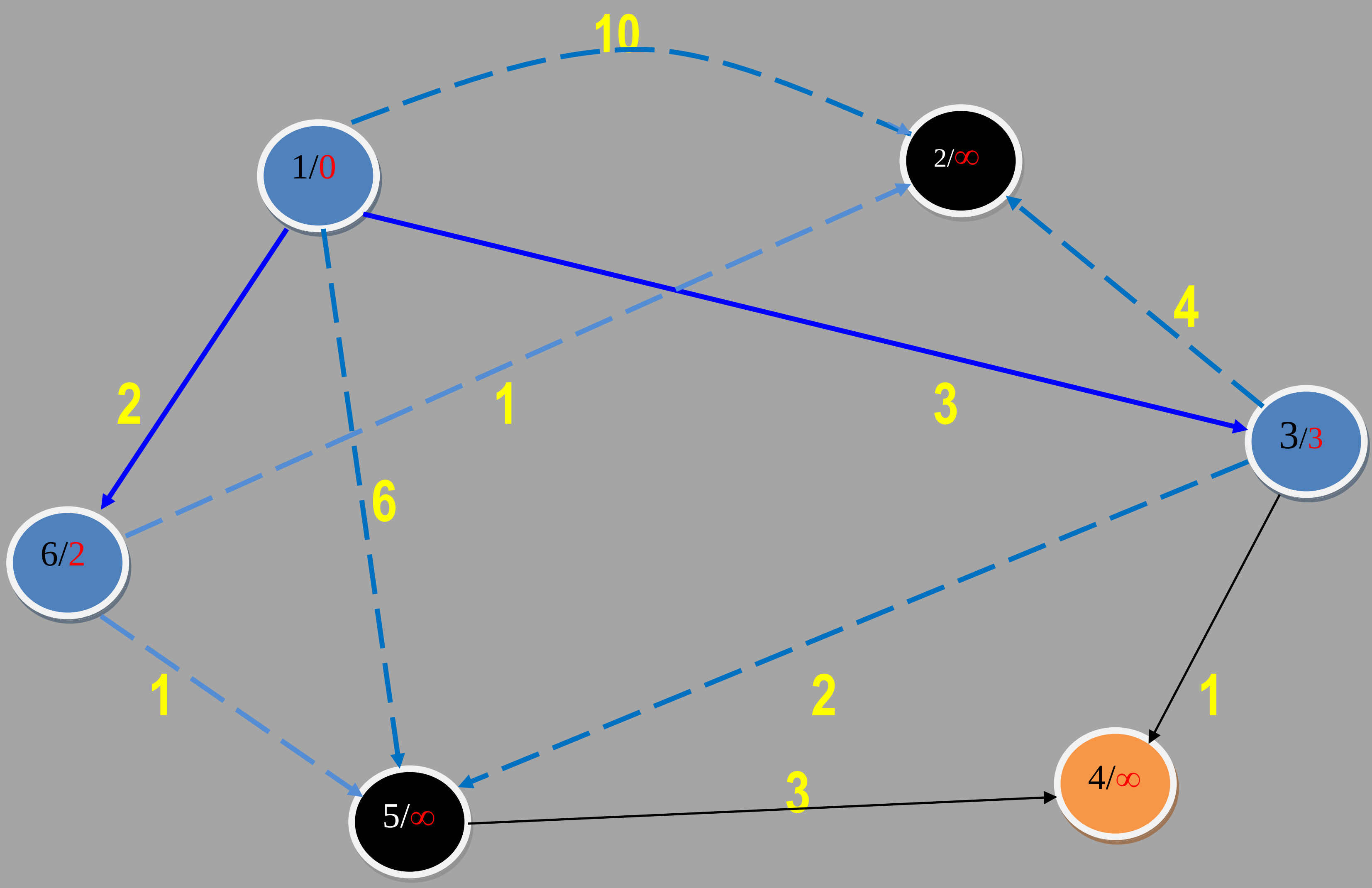
Distance :

1	2	3	4	5	6
0	+∝	3	+∝	+∝	2

Predecesseur :

1	2	3	4	5	6
nil	nil	1	nil	nil	1

Sélection de v parmi les sommets {2 ,5}



Choix du sommet 2

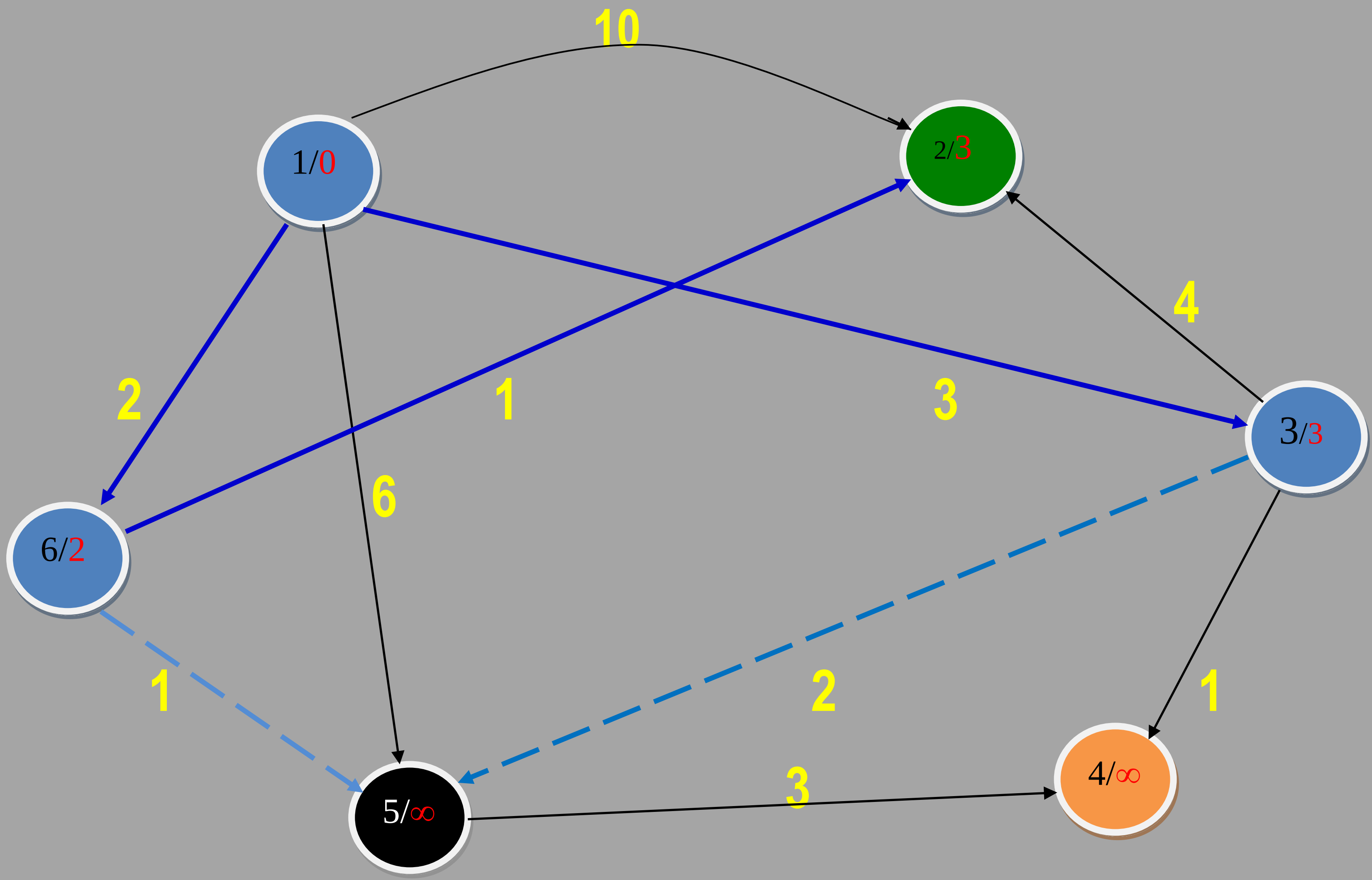
Distance :

1	2	3	4	5	6
0	3	3	$+\infty$	$+\infty$	2

Predecesseur :

1	2	3	4	5	6
nil	6	1	nil	nil	1

Choix du sommet 2



Choix du sommet 5

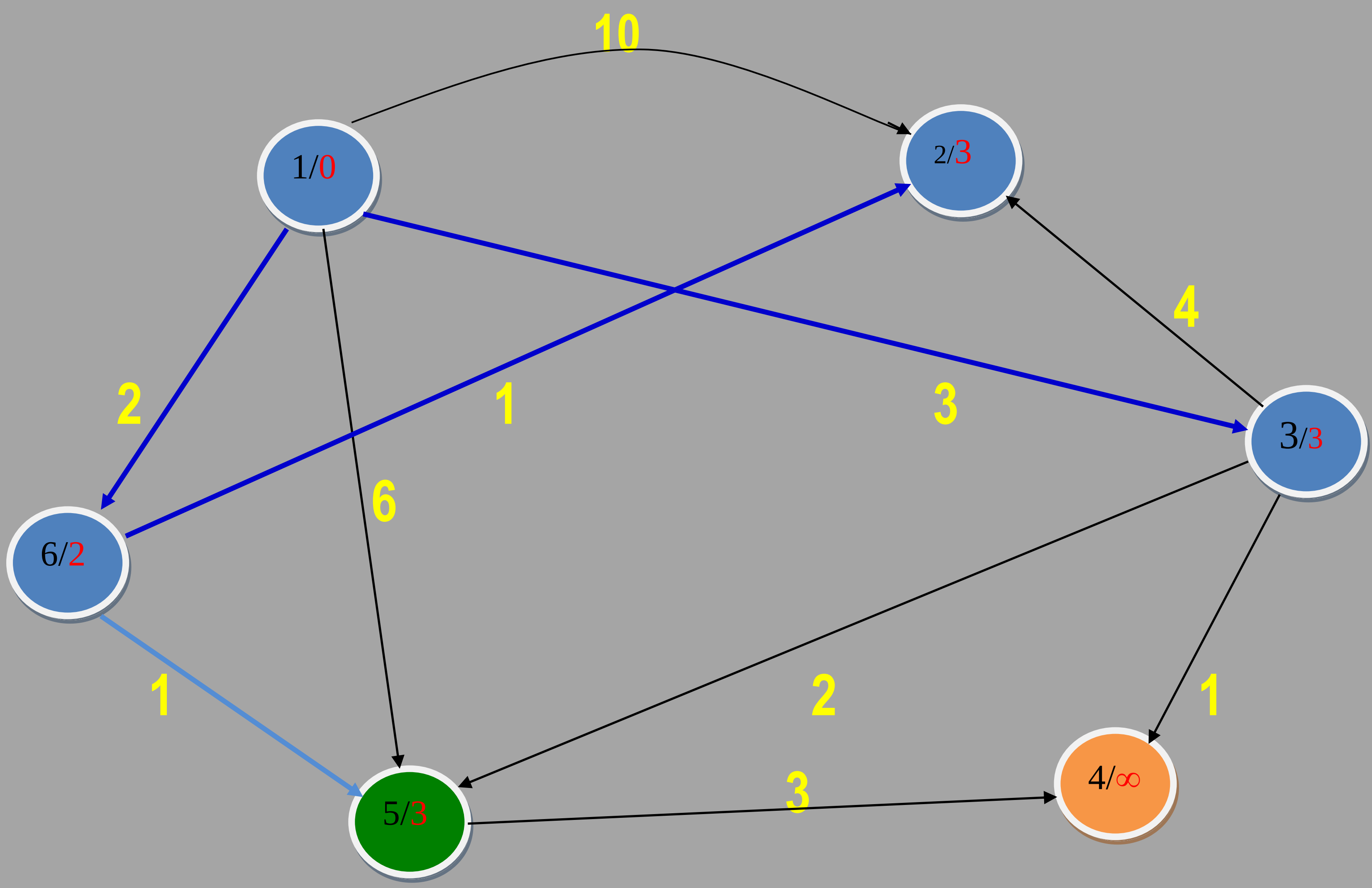
Distance :

1	2	3	4	5	6
0	3	3	$+\infty$	3	2

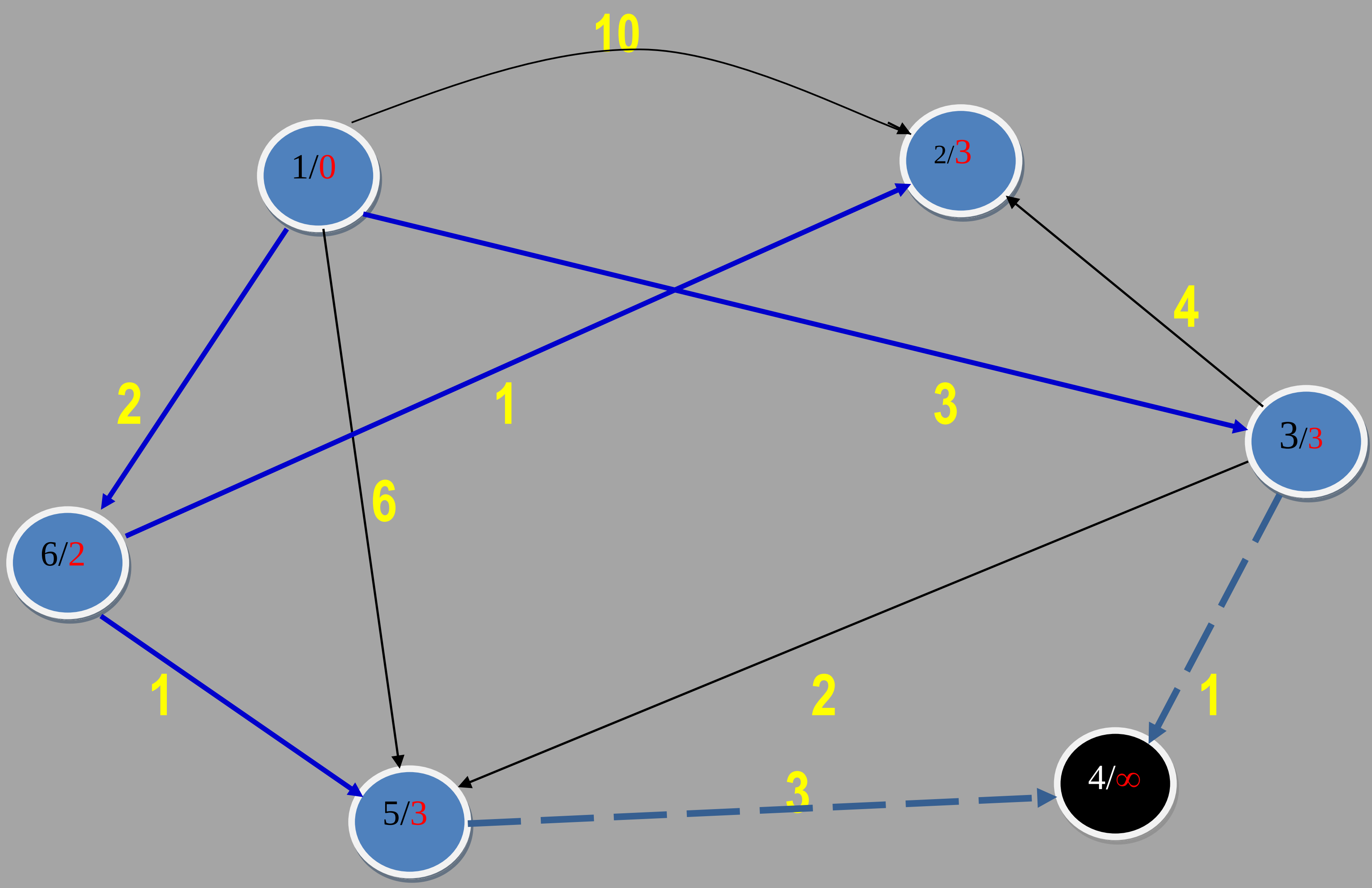
Predecesseur :

1	2	3	4	5	6
nil	6	1	nil	6	1

Choix du sommet 5



Sélection de **v** parmi {4 }



Choix du sommet 4

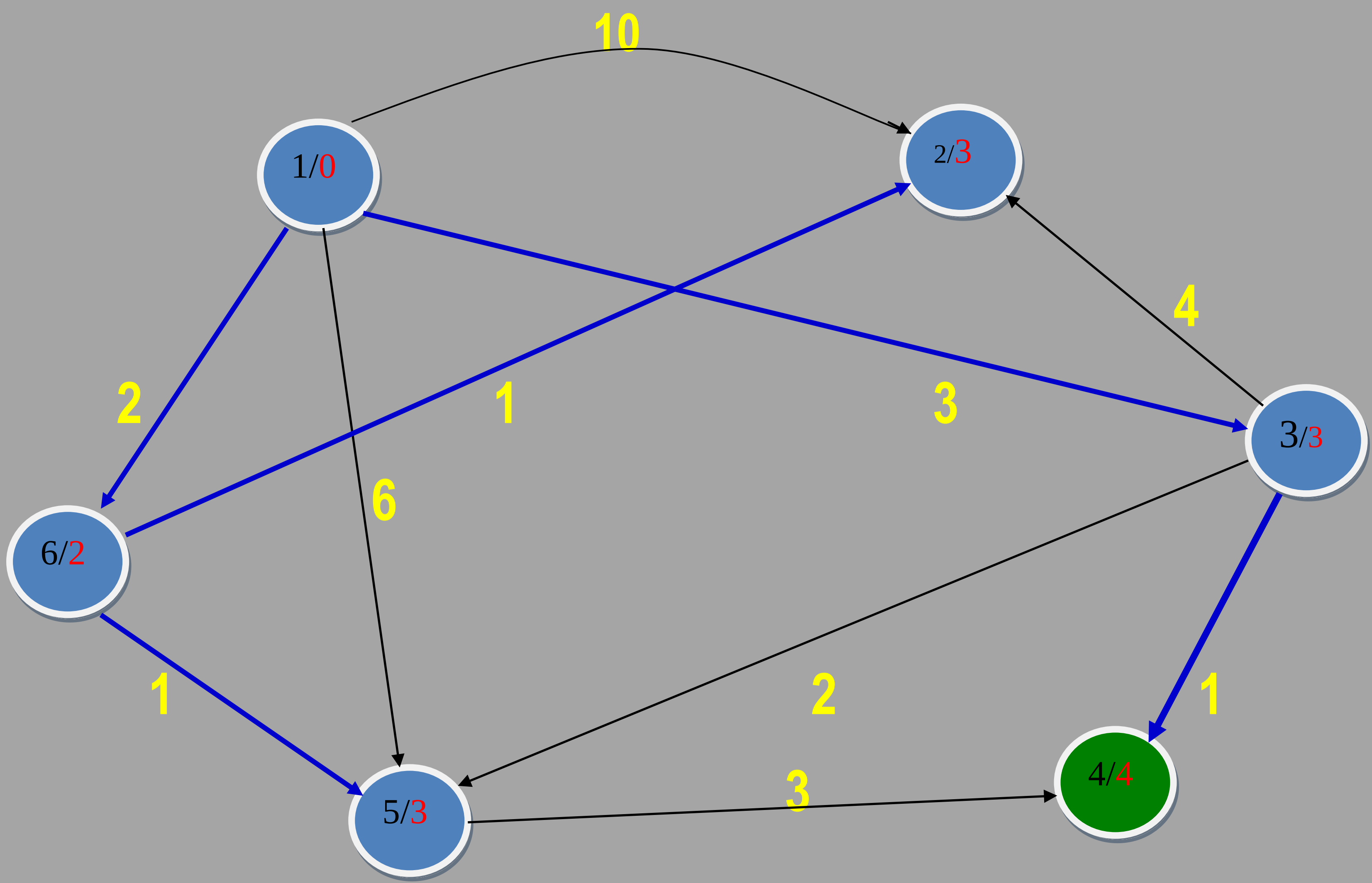
Distance :

1	2	3	4	5	6
0	3	3	4	3	2

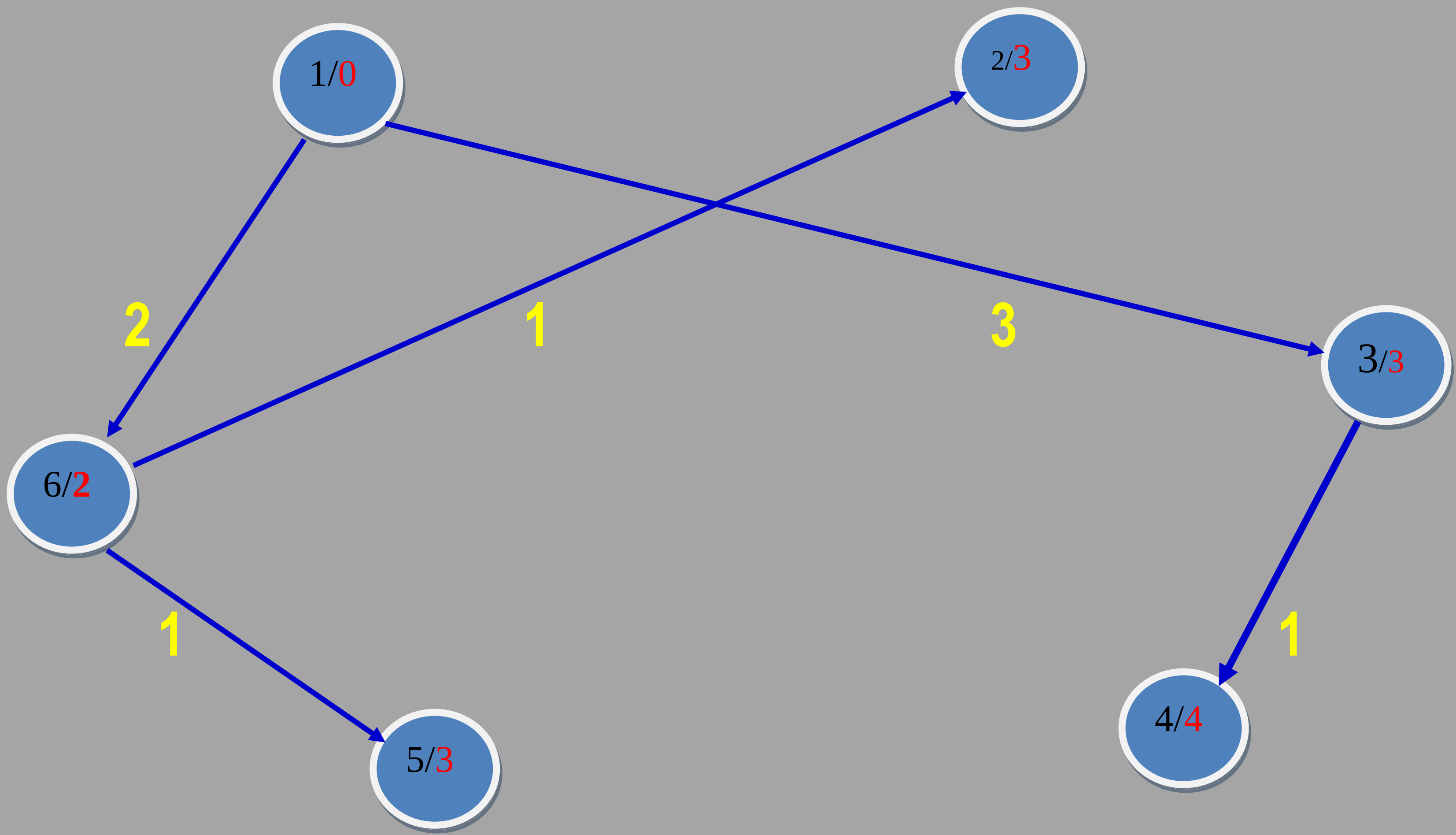
Predecesseur :

1	2	3	4	5	6
nil	6	1	3	6	1

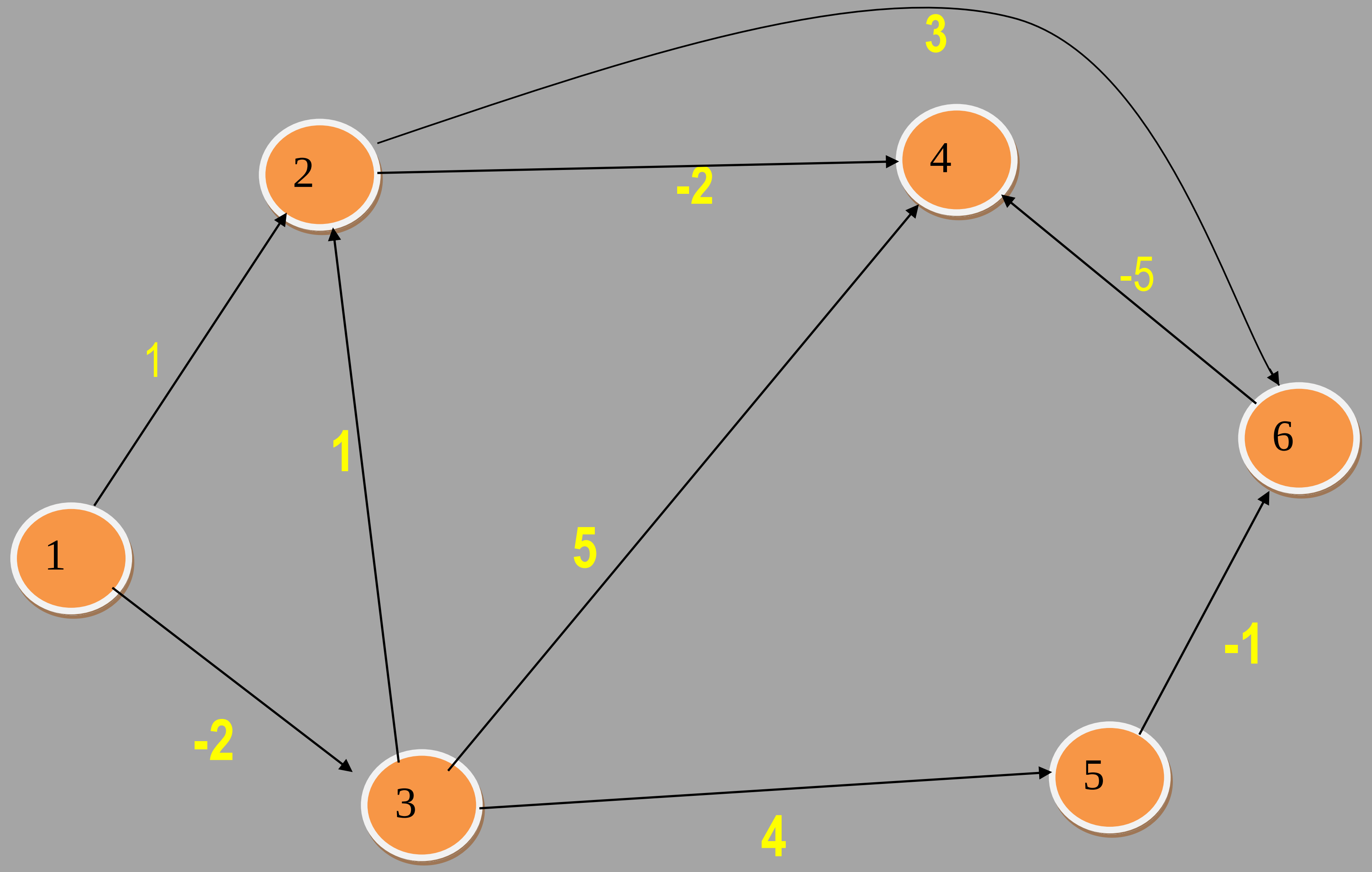
Sélection du sommet 4



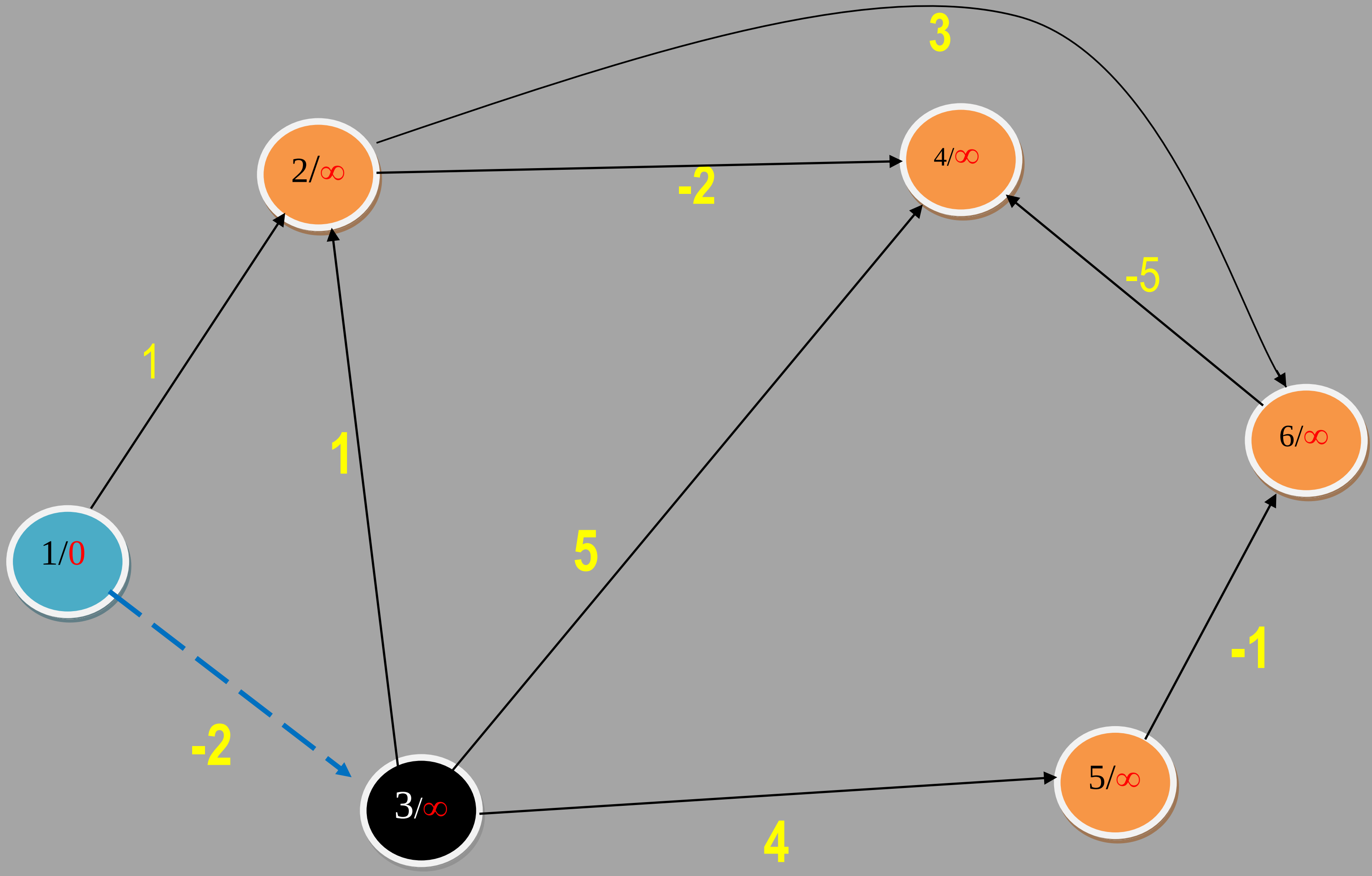
Arbre de Bellman



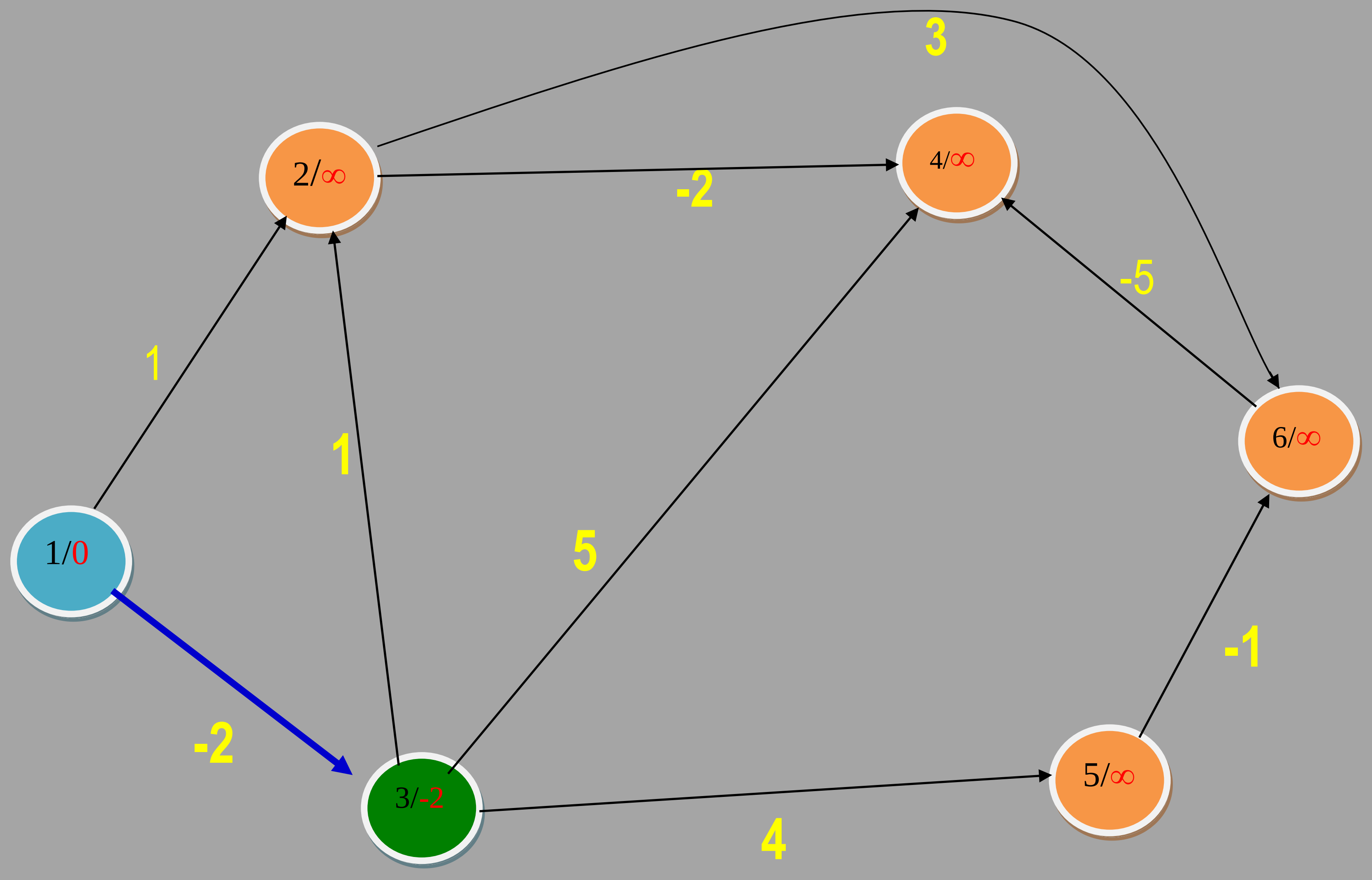
Application n°2



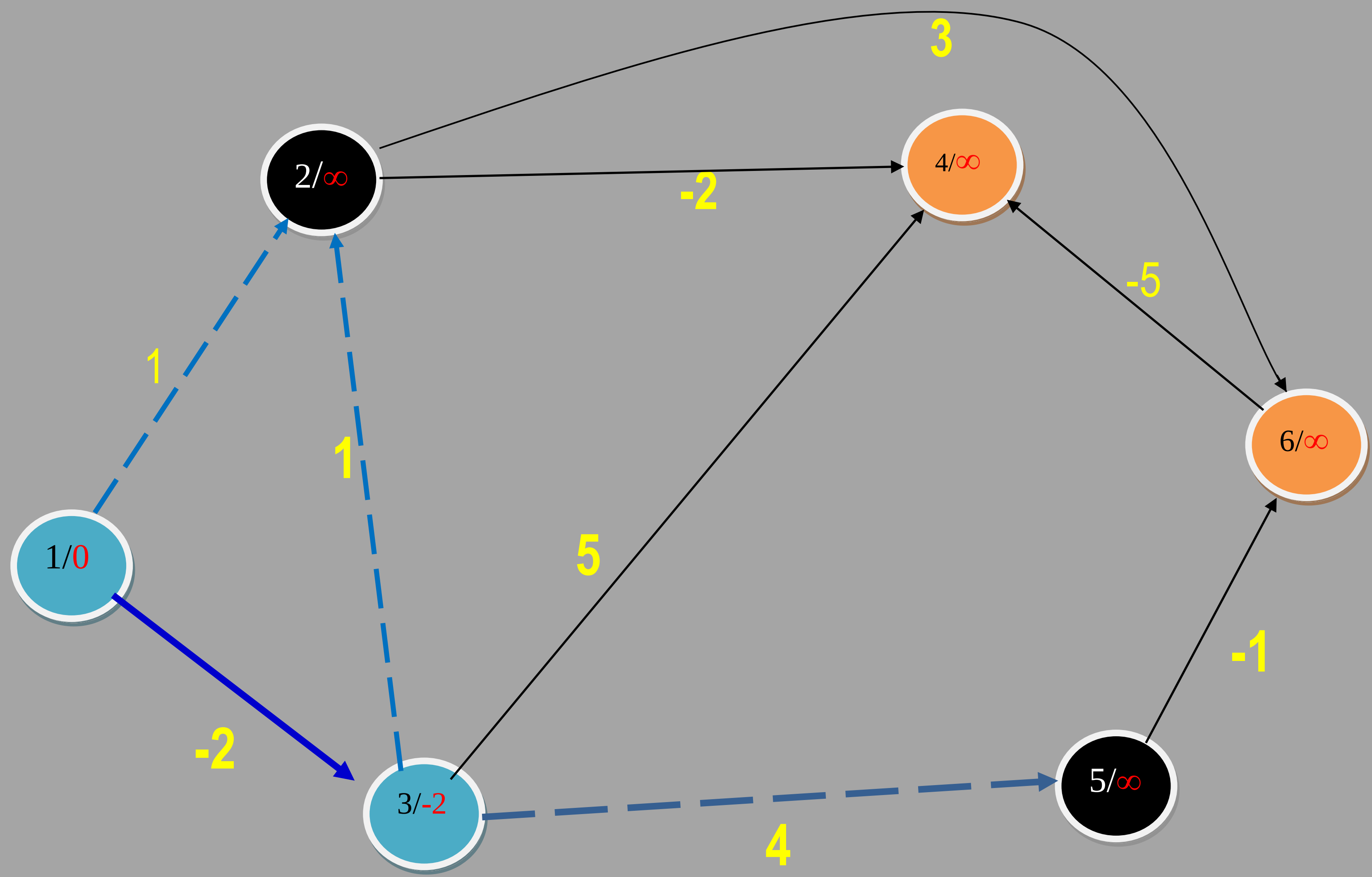
Au départ sommet s $\rightarrow$1



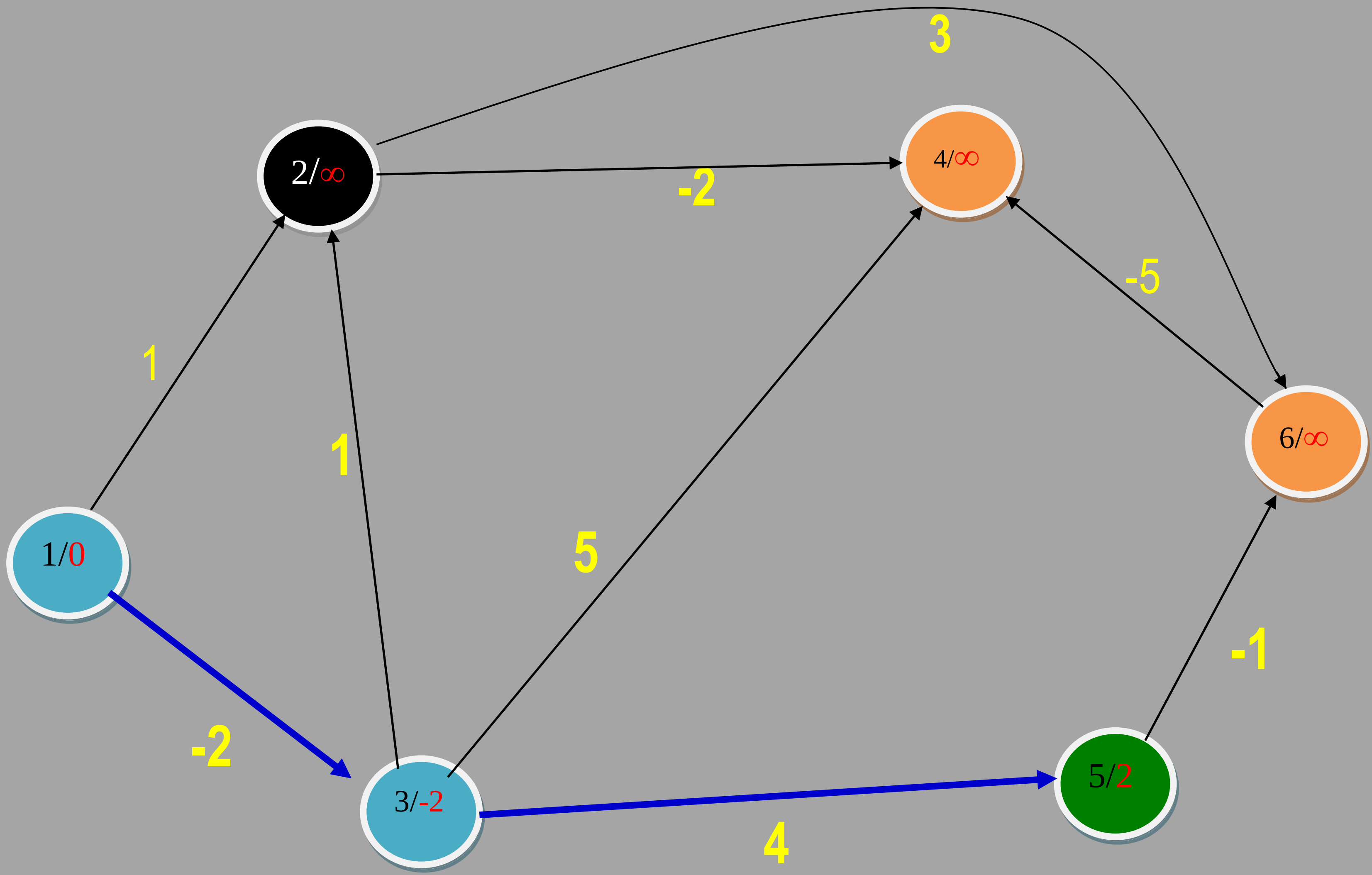
Sélection de v dans {3}



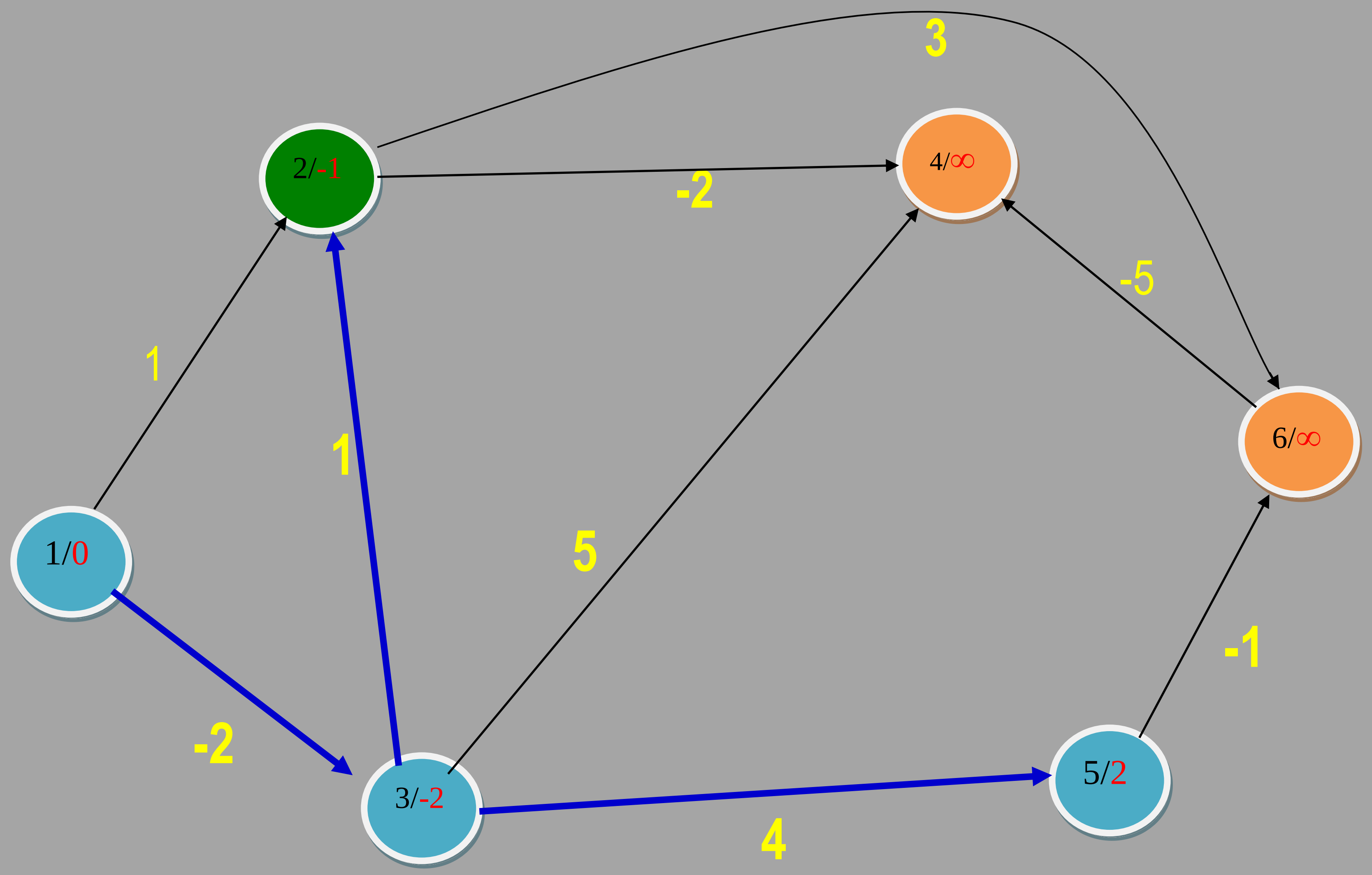
Sélection de v dans {5,2}



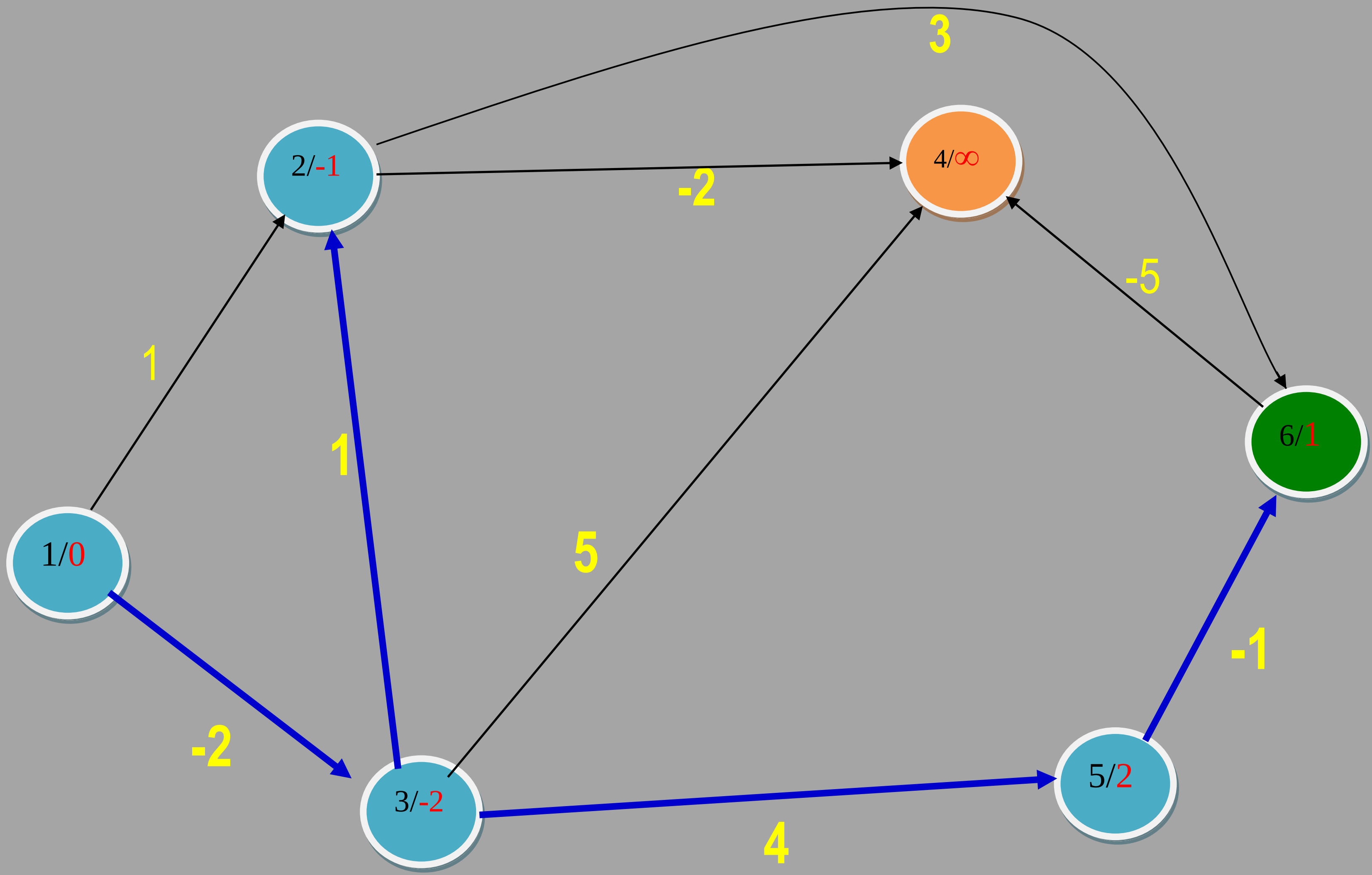
Choix du sommet 5



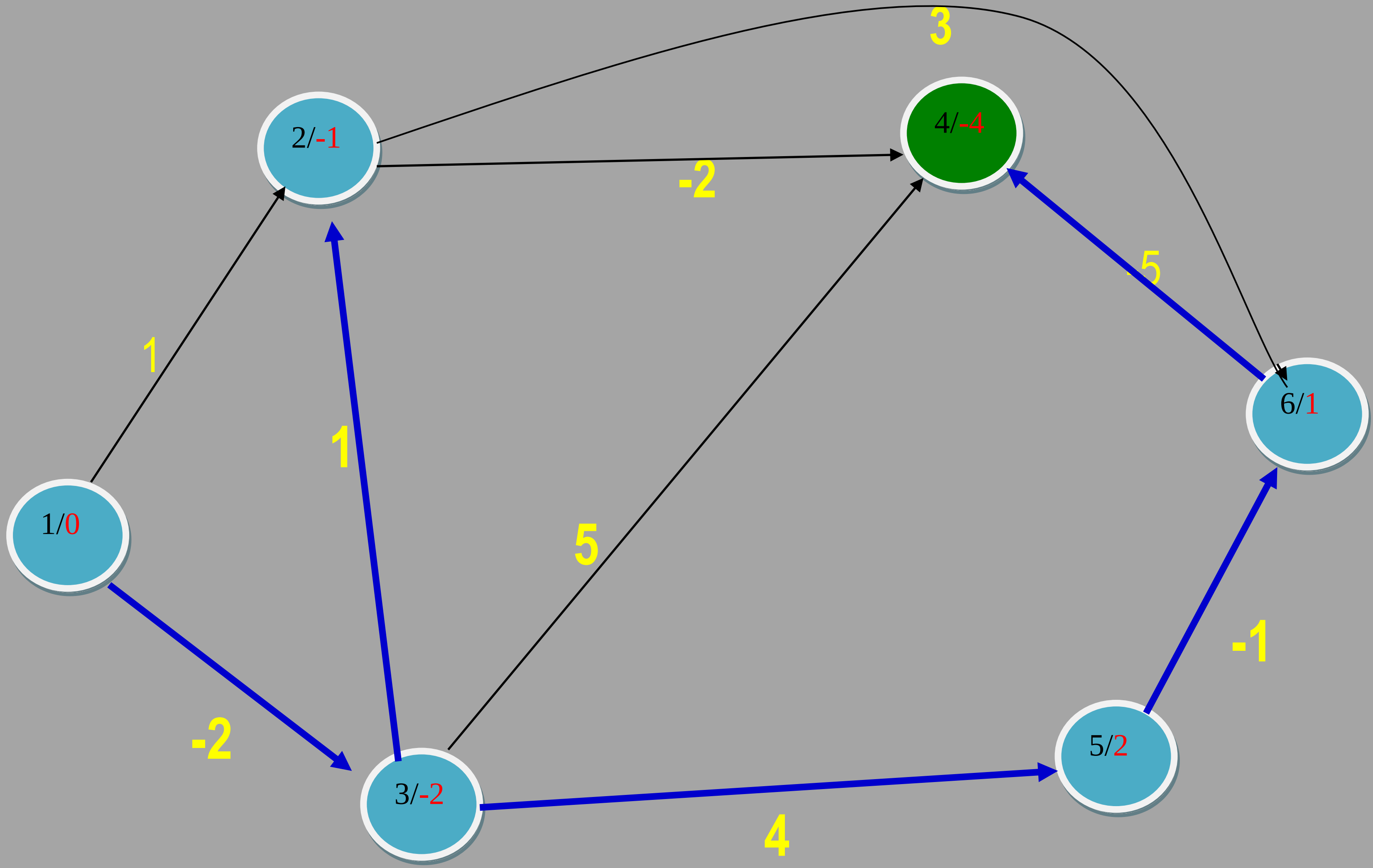
Choix du sommet 2



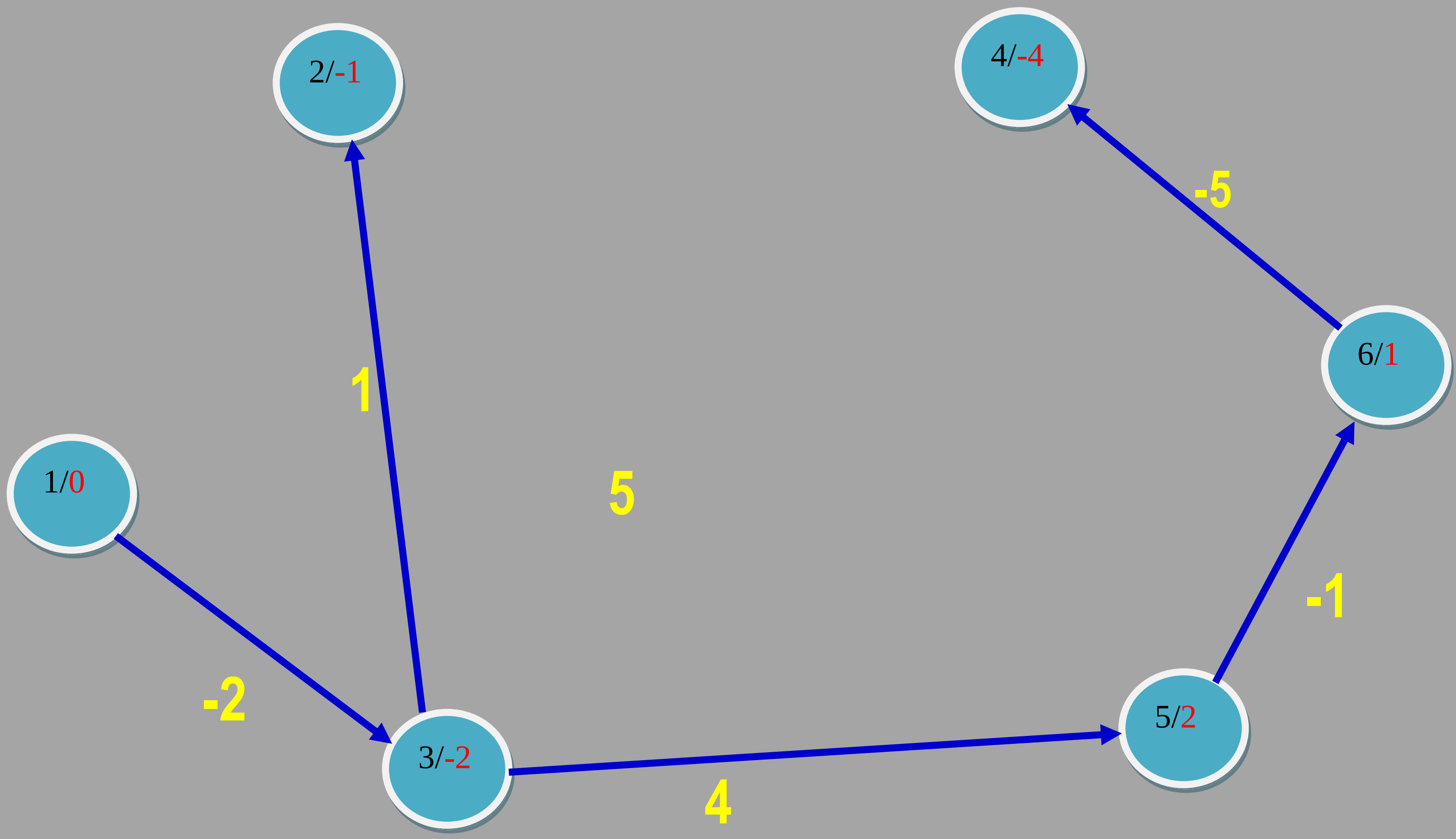
Sélection de v dans {6}



Sélection de v dans {4}



Arbre de Bellman



3 - Procédure de BELLMAN-Duale

Entrées:

G = (S, A, cout) : graphe orienté valué

t : un sommet cible.

Sorties:

L[x] : indique la « distance optimale » du sommet **x** ***vers*** **t**.

Suc[x] : indique le successeur du sommet **x** si on emprunte le «chemin le plus court» vers t.

Variables intermédiaires:

X et **X'** : deux sous-ensembles de sommets,

u et ***v*** : deux sommets,

SucG (u): sous-ensemble des successeurs de u dans G.

Bellman_Duale(G, t)

Début

/*initialisation */

X'←{t}; X ←S-X' ;

Pour tout u∈S **Faire**

L(u) ← ∝;

Suc(u)← nil

FinPour;

L(t)← 0;

/*relaxation de l'arc (u,v) */

Tant que $\exists u \in X$ | **SucG**(u)$\subseteq$ X' **Faire**

X'$\leftarrow$X'$\cup$\{u\};

X$\leftarrow$X-\{u\};

Pour tout(u,v) | v$\in$X' **Faire**

Si **L**(u)> **L**(v)+cout(u,v)

Alors

L(u)$\leftarrow$ **L**(v)+cout(u,v);

Suc(u)$\leftarrow$v

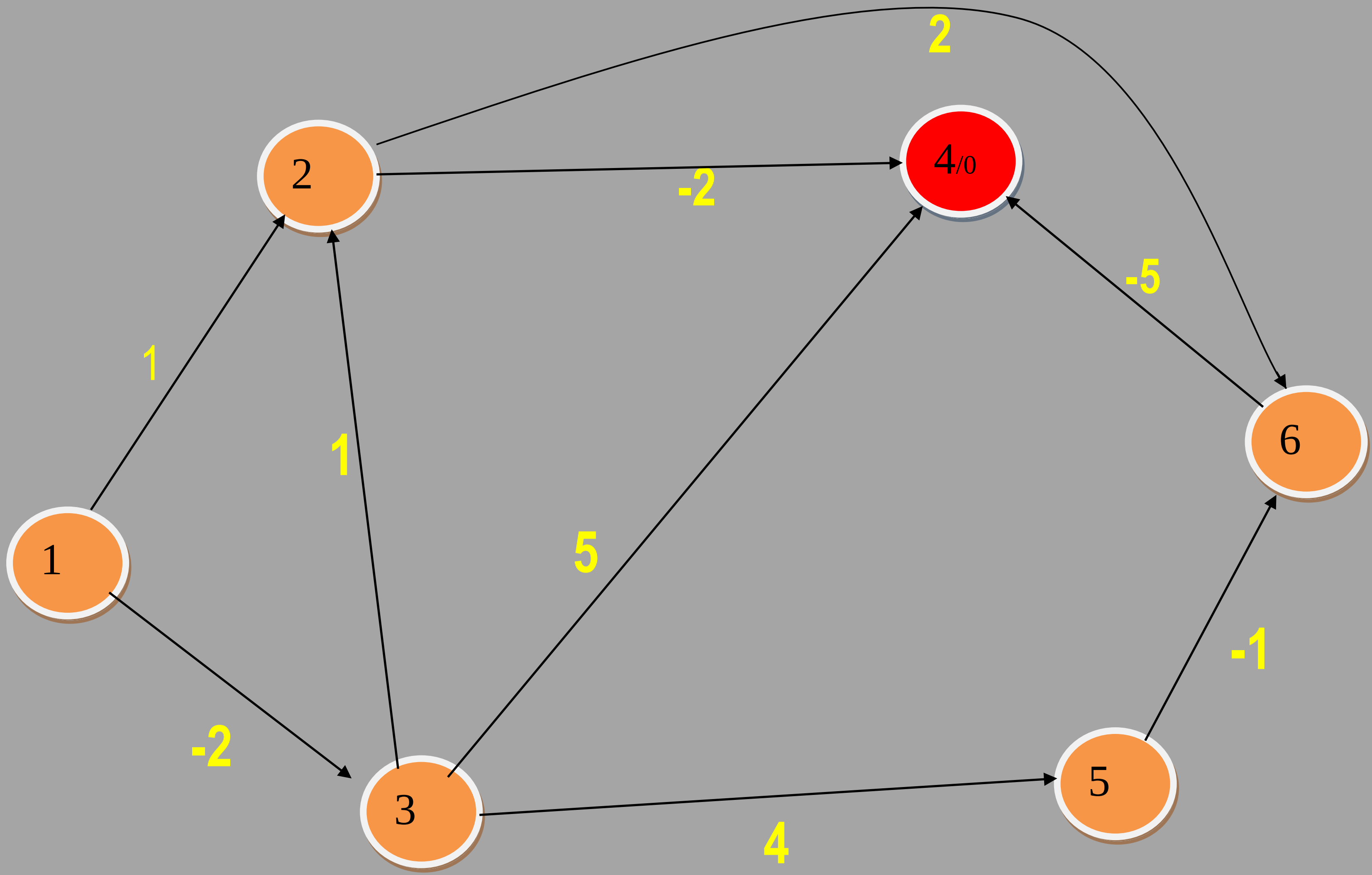
FinSi;

FinPour

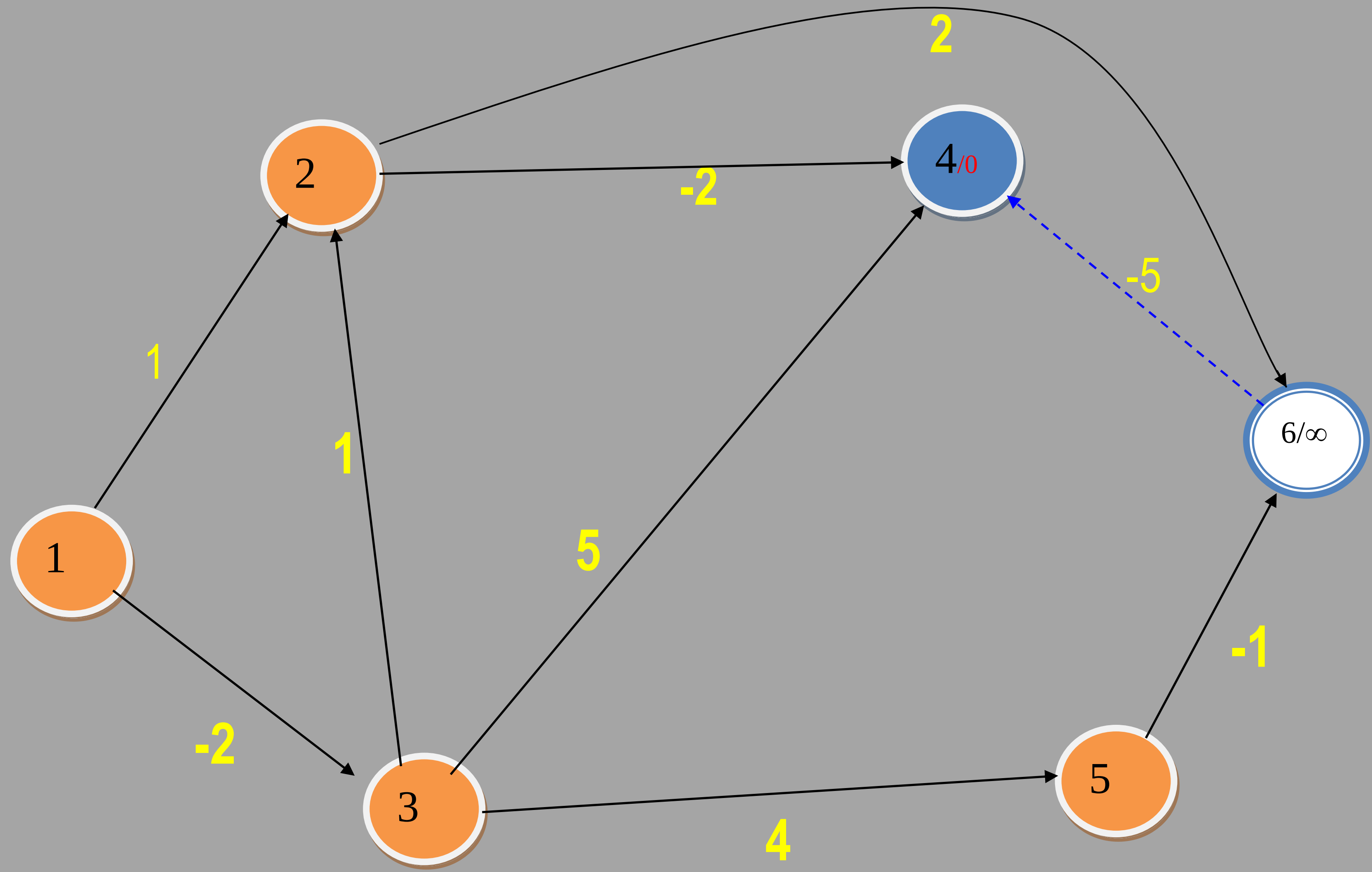
FinTq

Fin

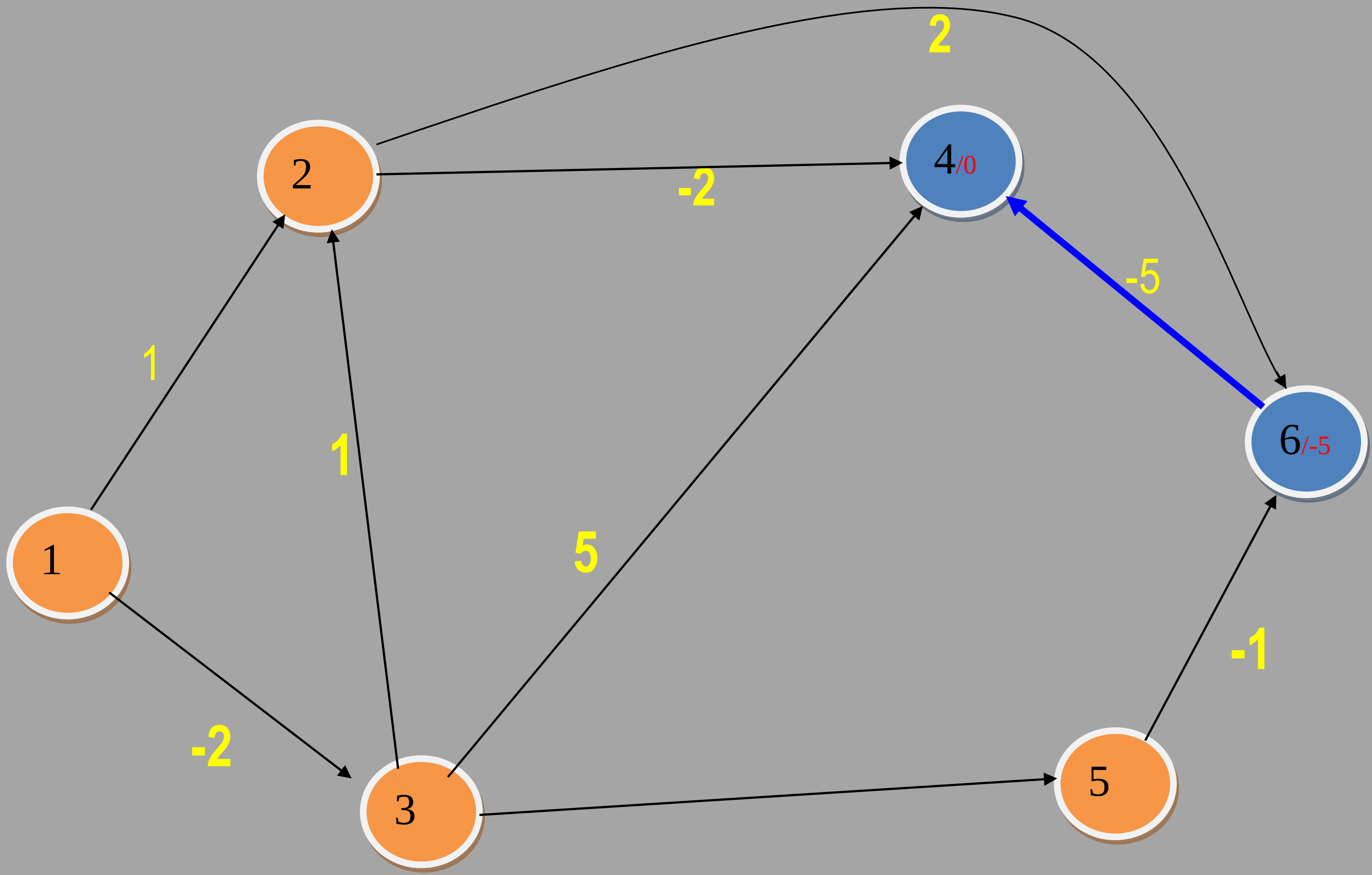
Application n°3 : routage vers le sommet 4



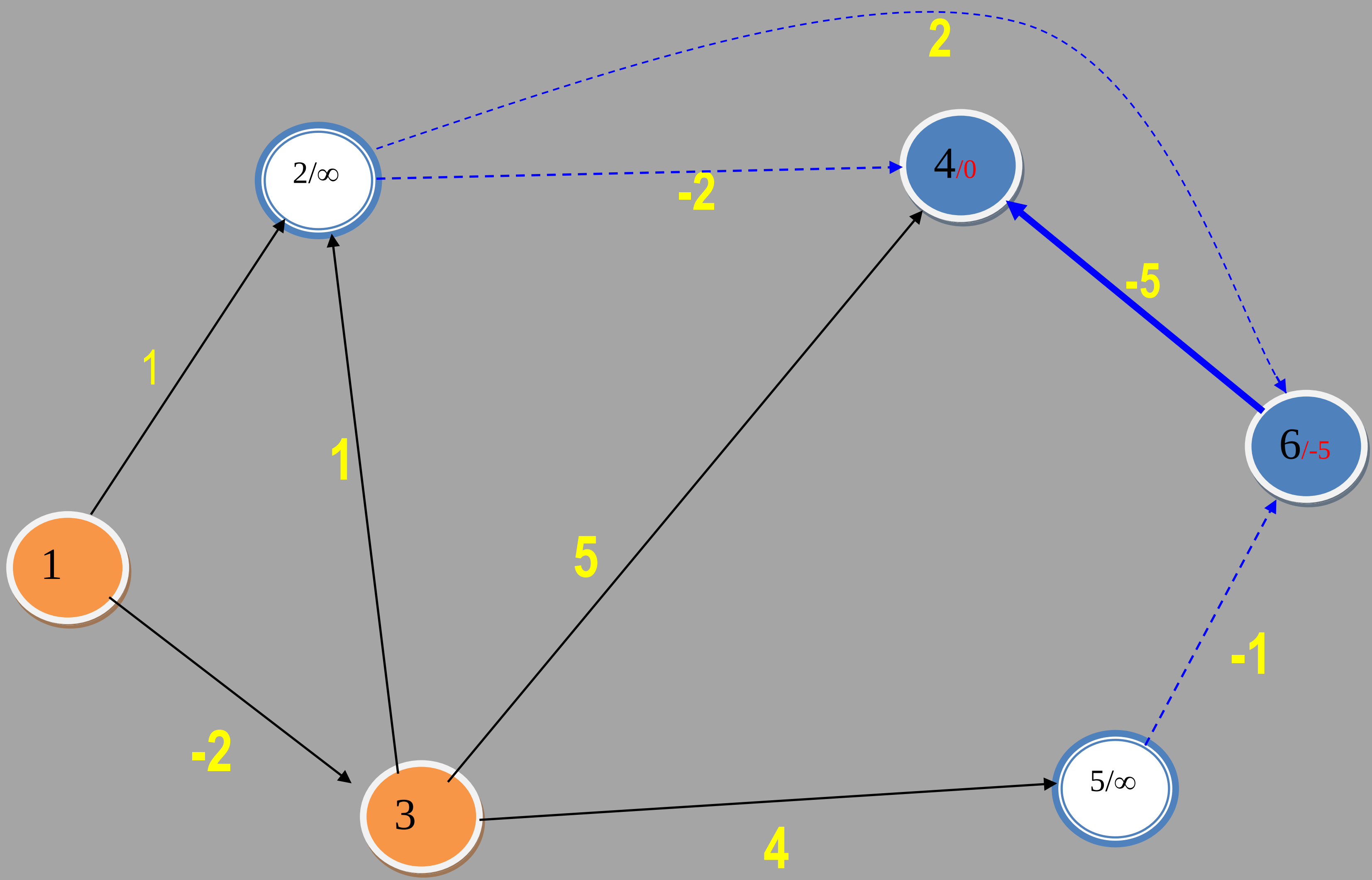
{6} vers {4}



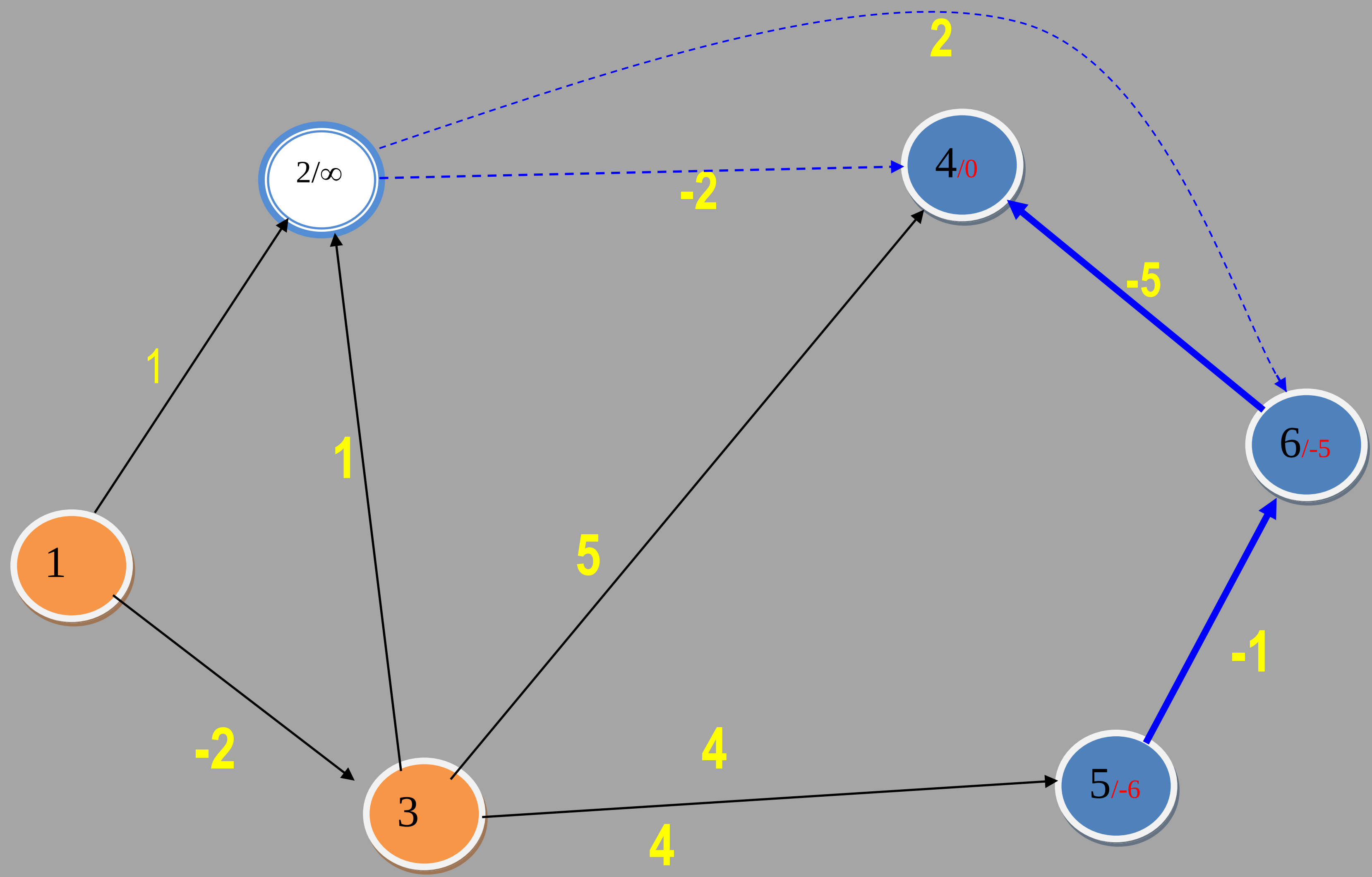
6 vers 4



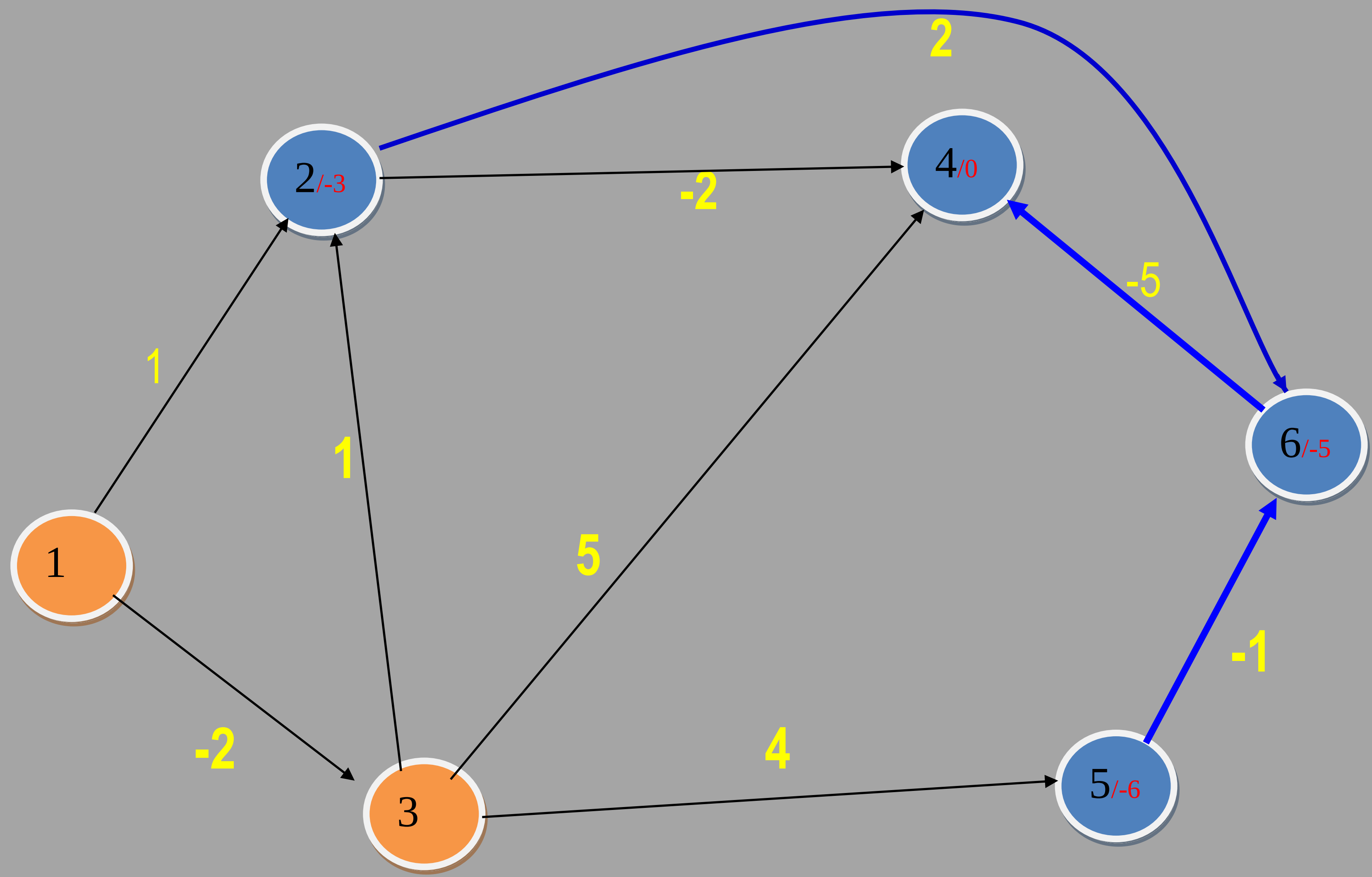
{2,5} vers {4,6}



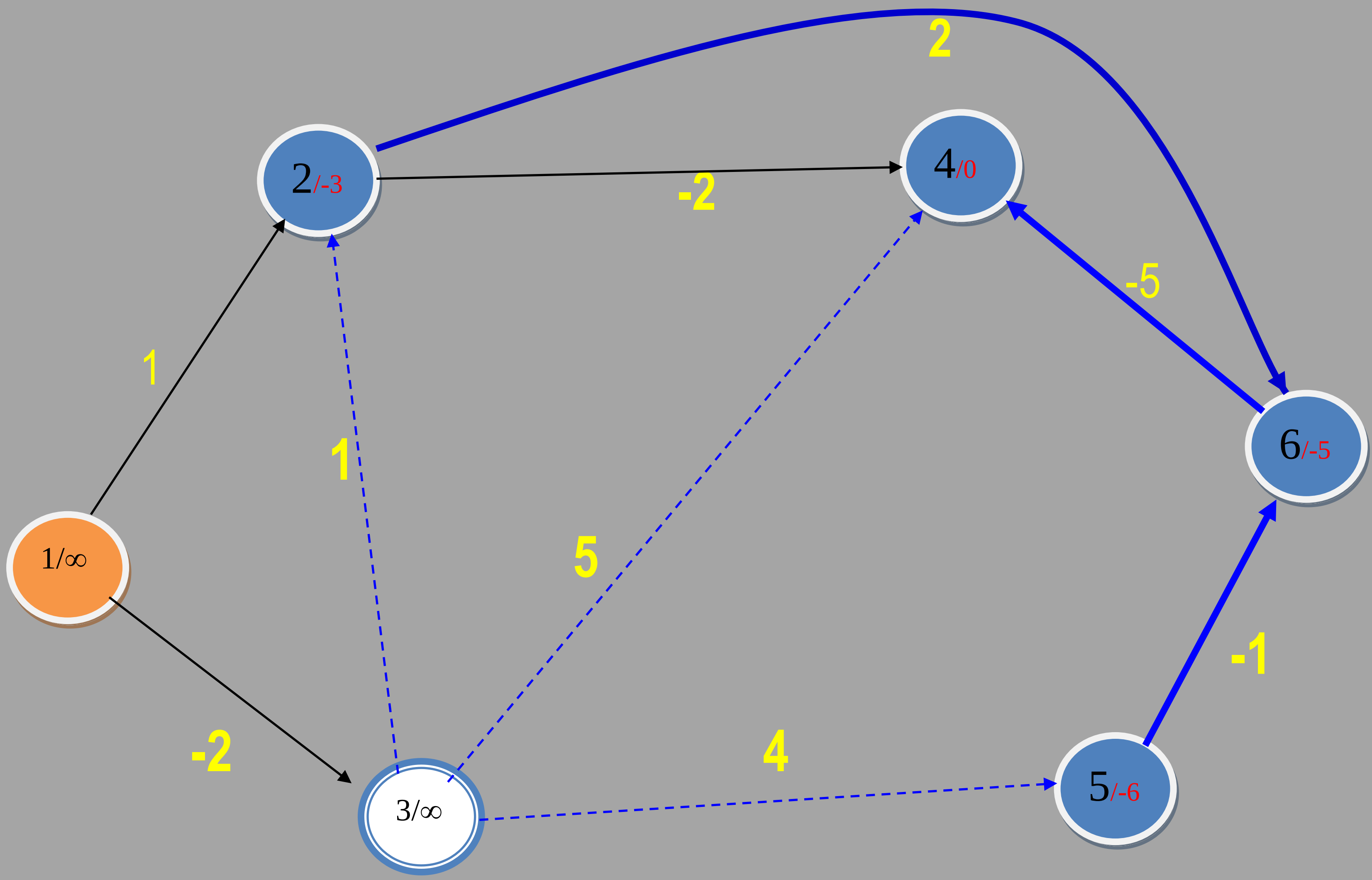
5 vers 6



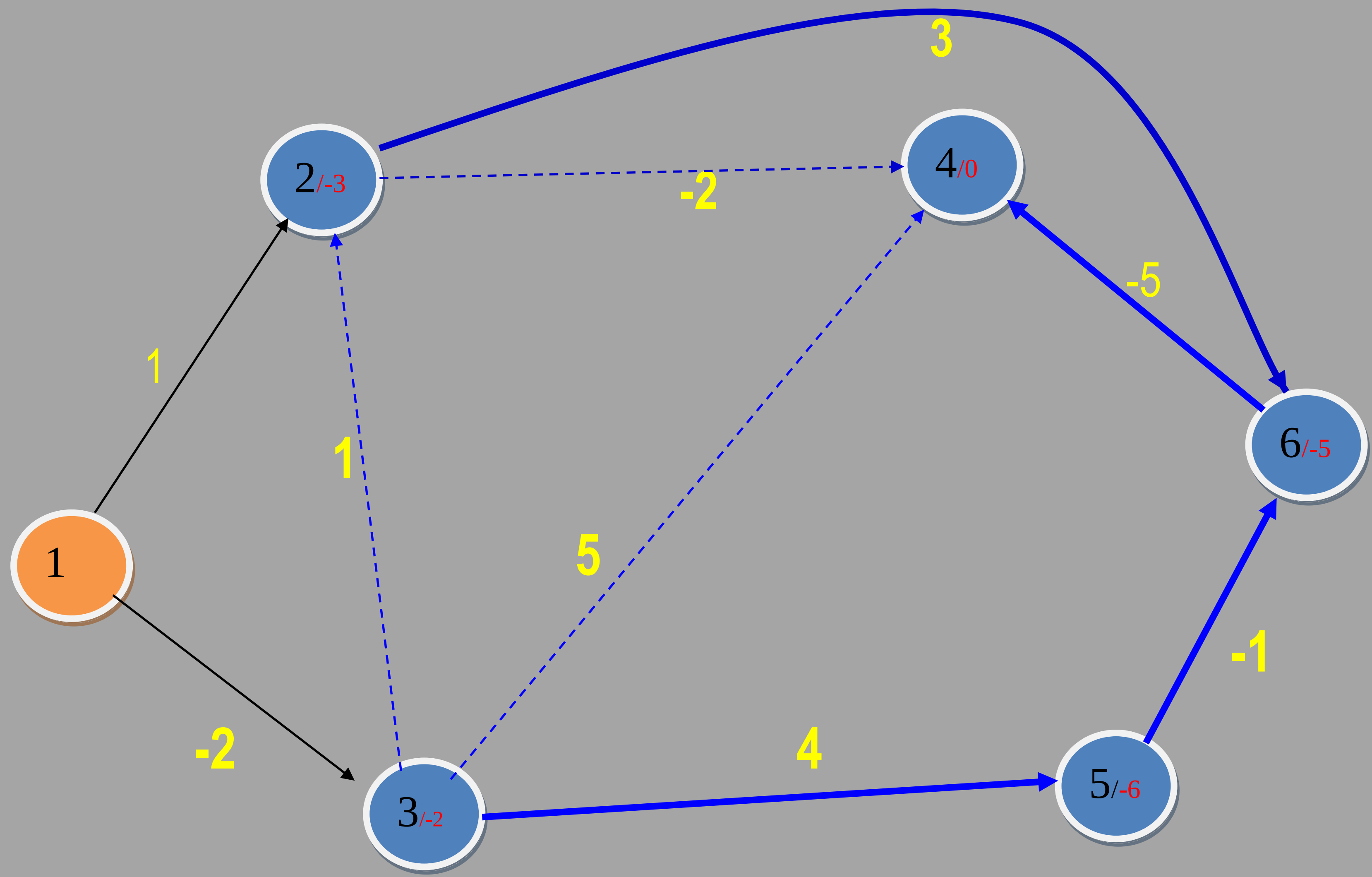
2 vers 6



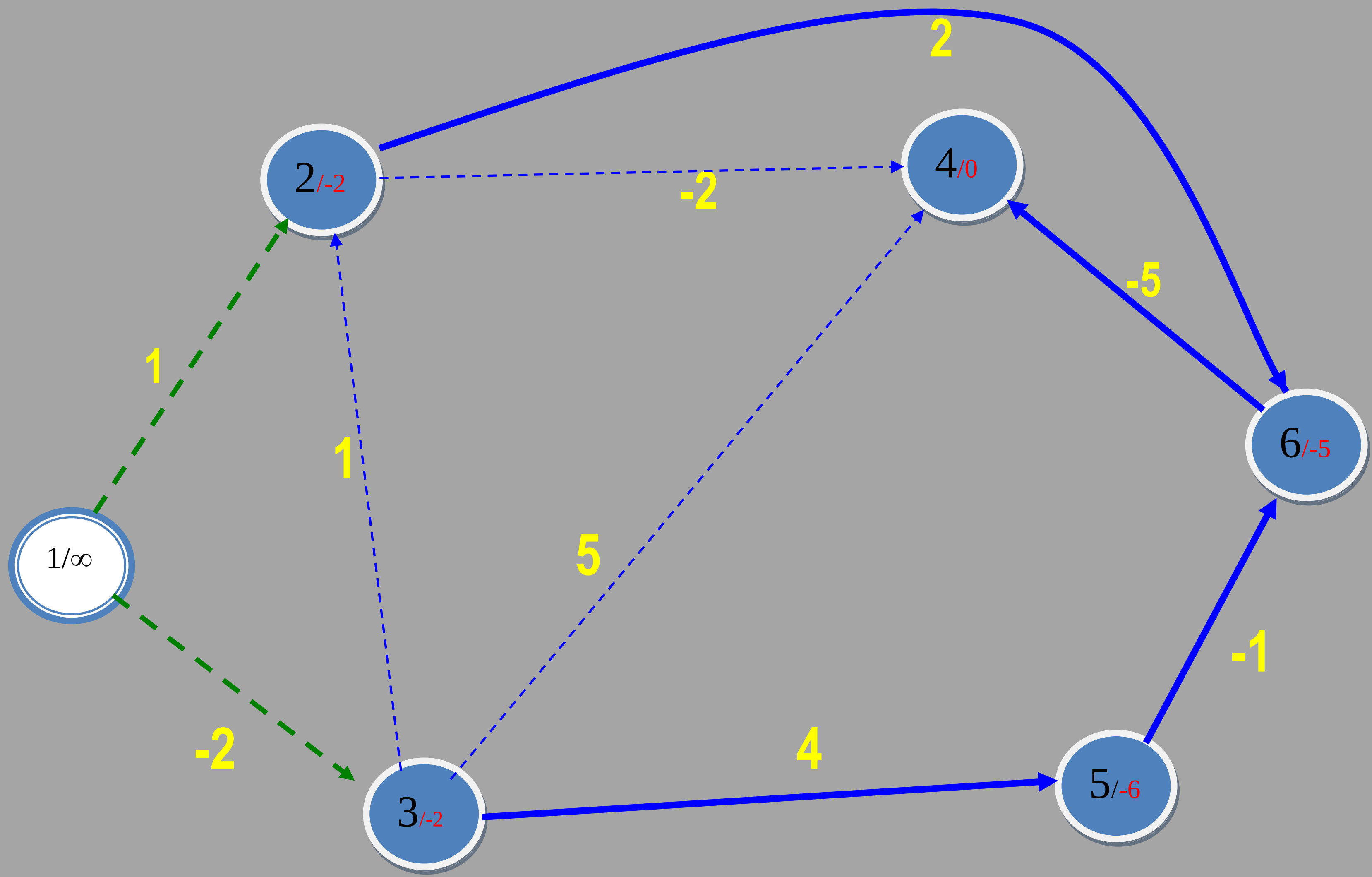
{3} vers {2,4,5}



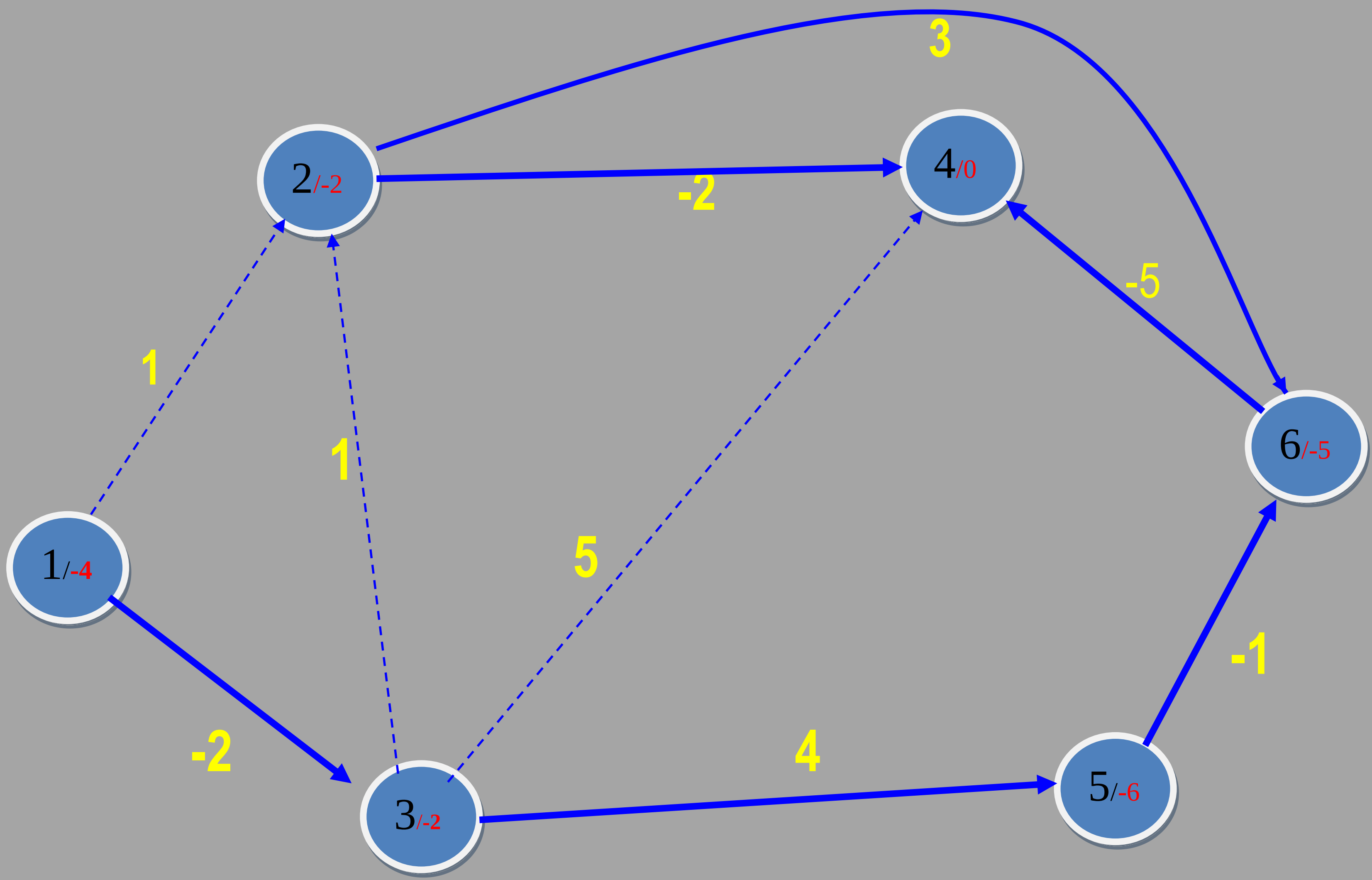
3 vers 5



{1} vers { 2,3}



1 vers 3



Arbre dual de Bellman