Parcours de graphes

Benjamin Monmege

Kézako ?

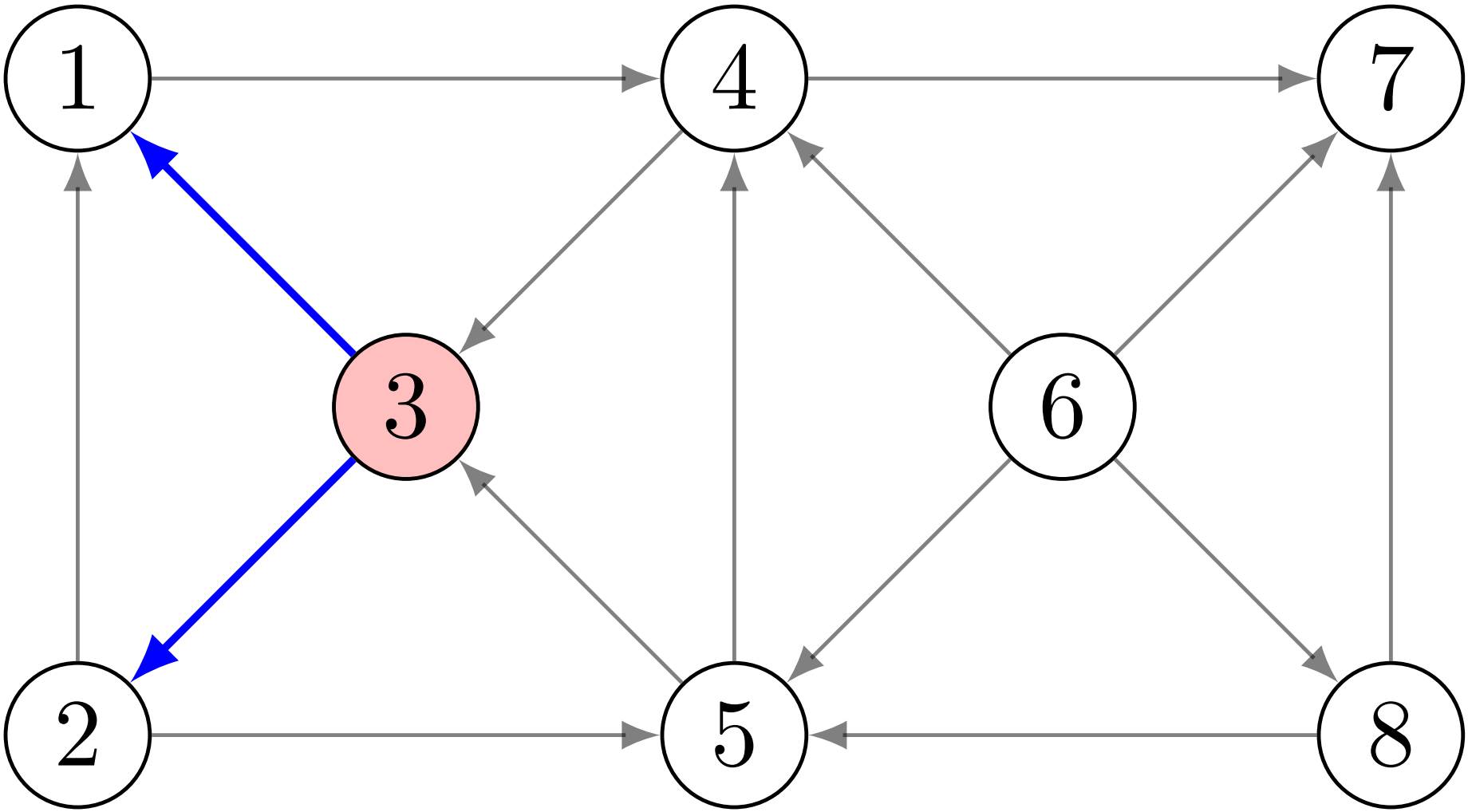
Un *algorithme de parcours de graphe* est un algorithme qui, partant d'un sommet s d'un graphe appelé *source*, explore tous les sommets ou tous les arcs accessibles depuis cette source, et aucun autre.

- chaque sommet u, hormis la source, est associé à un *sommet prédécesseur* $pred(u)$ tel que $(pred(u), u)$ est un arc du graphe
- il existe un entier ℓ tel que $pred^{\ell}(u) = s$
- ainsi, $pred^{\ell}(u), pred^{\ell-1}(u), \ldots, pred(u), u$ est un chemin de s à u

Parcours de graphe permet de calculer tous les sommets *accessibles* depuis la source.

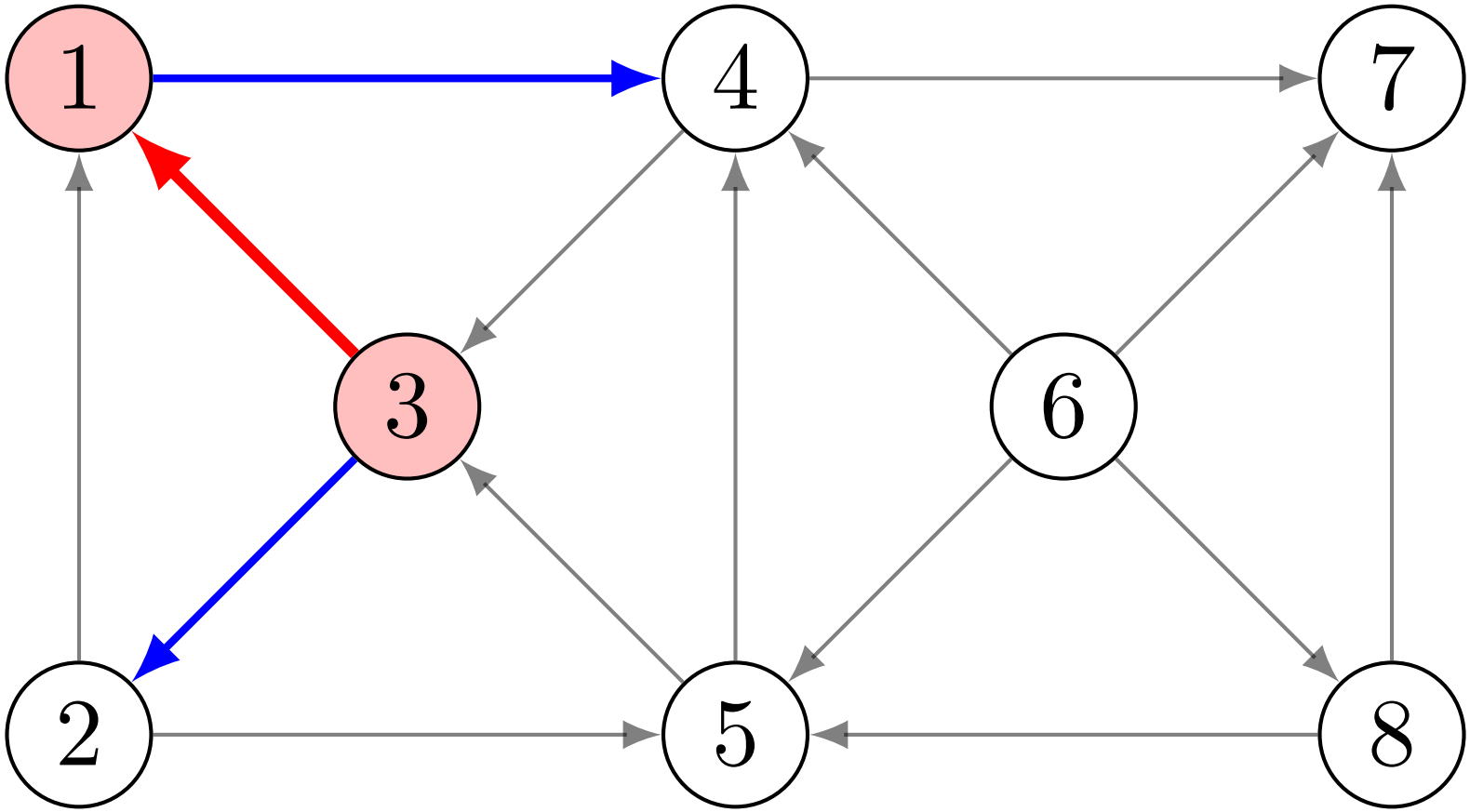
La fonction *pred* détermine une *arborescence*: un sous-ensemble d'arcs tel que chaque sommet possède un seul arc entrant sauf la source qui n'en a pas, et ne possédant pas de cycles.

Parcours générique : sommets parcourus et frontière



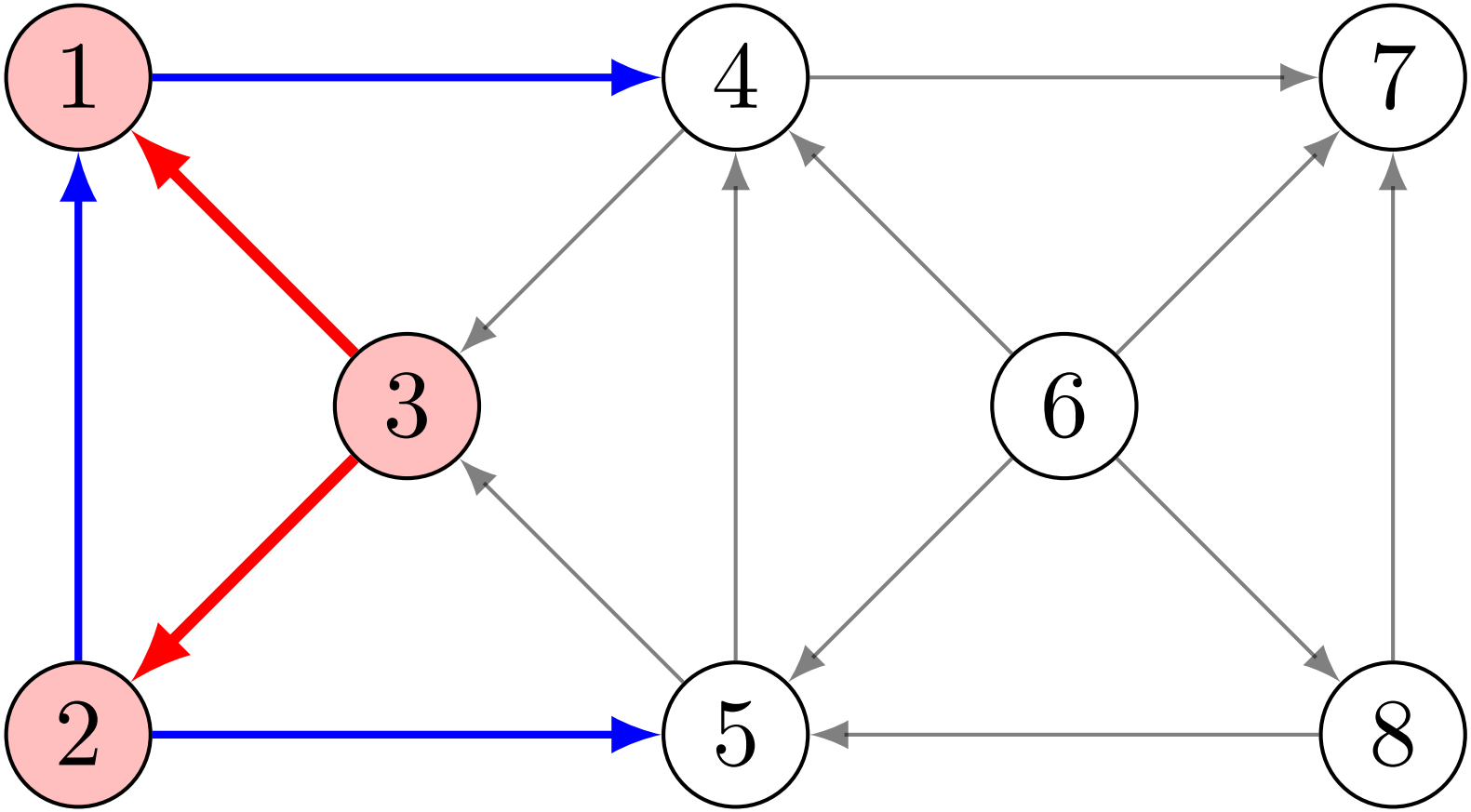
(a) Initialisation, en source 3

Parcours générique : sommets parcourus et frontière



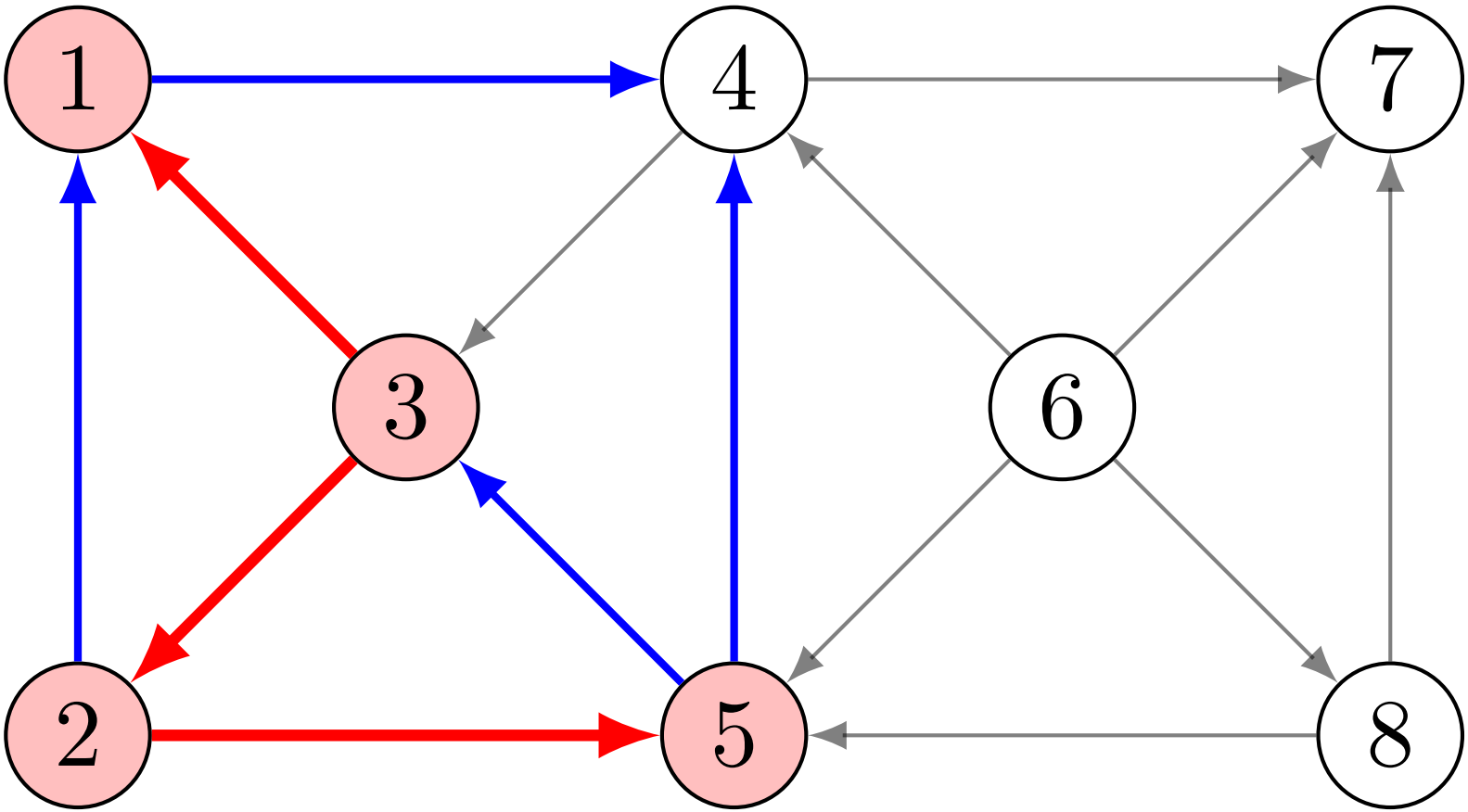
(b) avancée par (3,1) et exploration de 1

Parcours générique : sommets parcourus et frontière



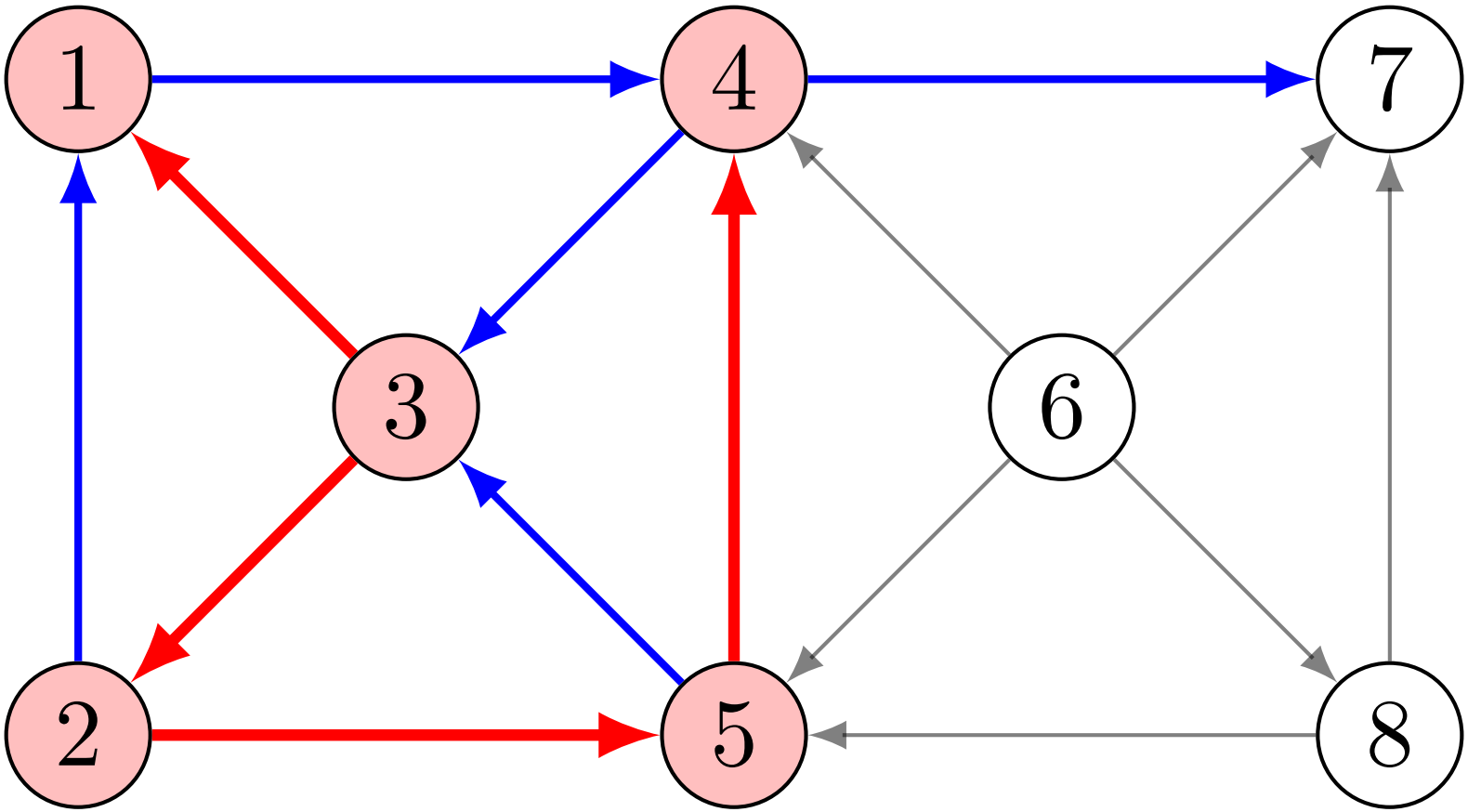
(c) avancée par (3,2) et exploration de 2

Parcours générique : sommets parcourus et frontière



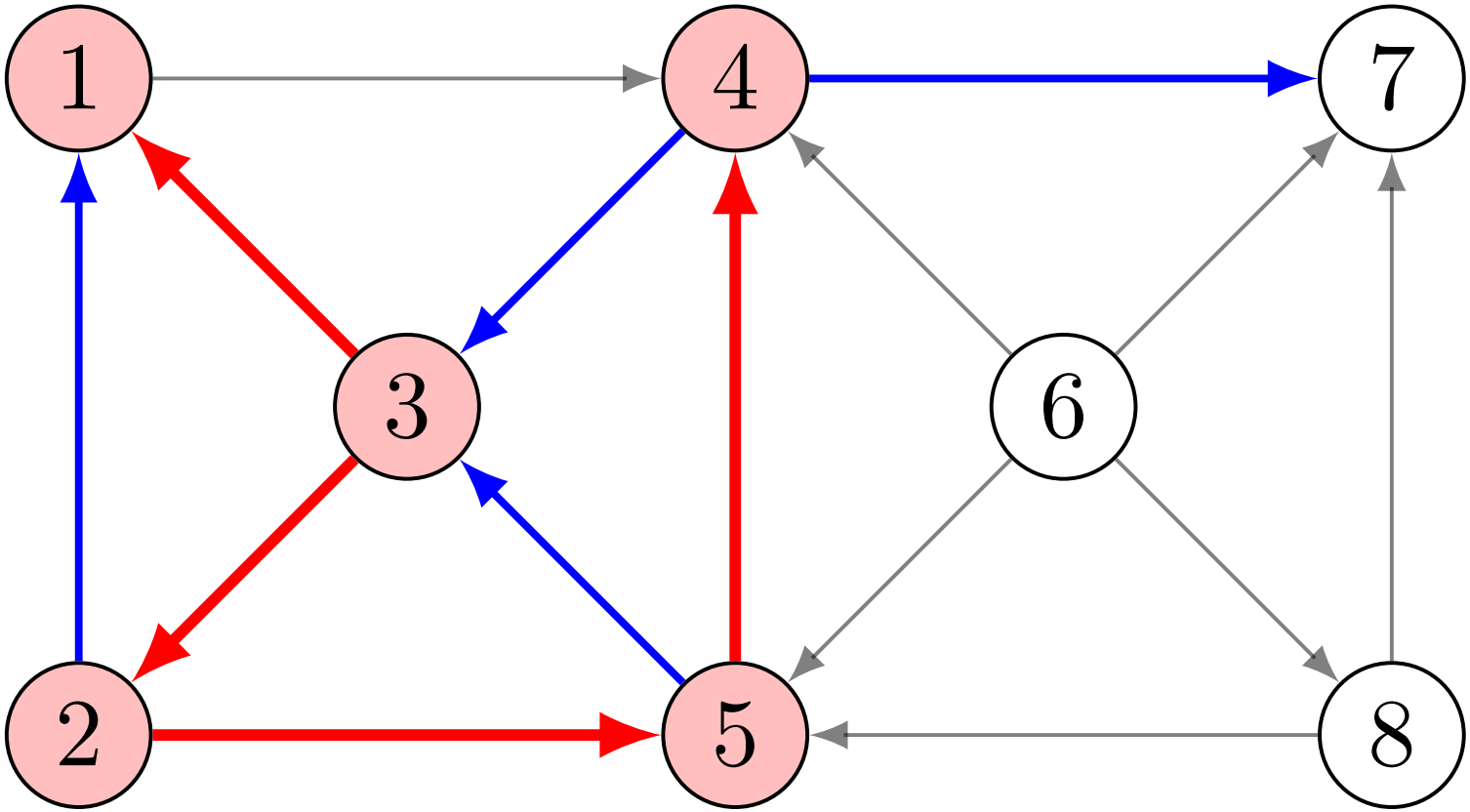
(d) avancée par (2,5), exploration de 5

Parcours générique : sommets parcourus et frontière



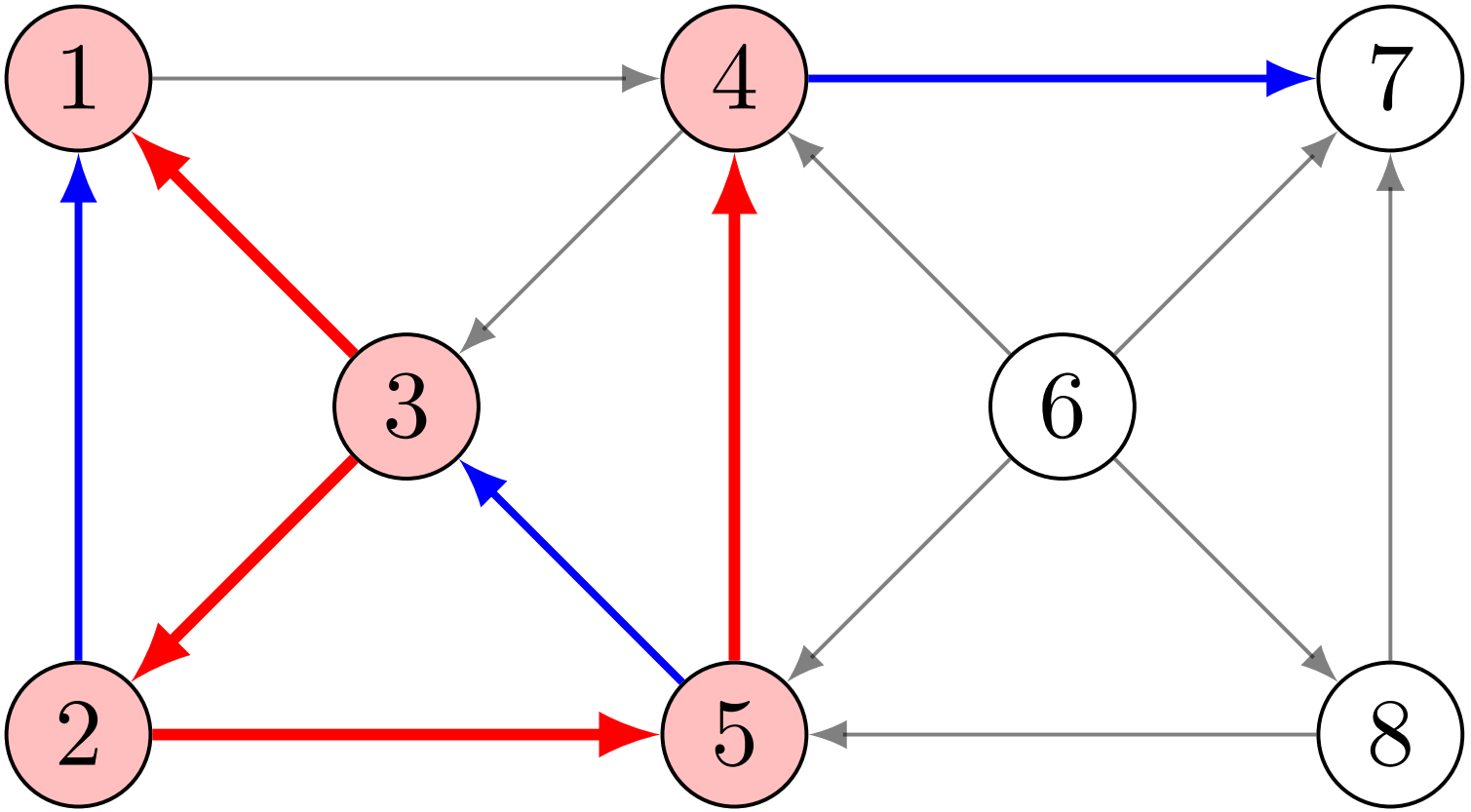
(e) avancée par (5,4) et exploration de 4

Parcours générique : sommets parcourus et frontière



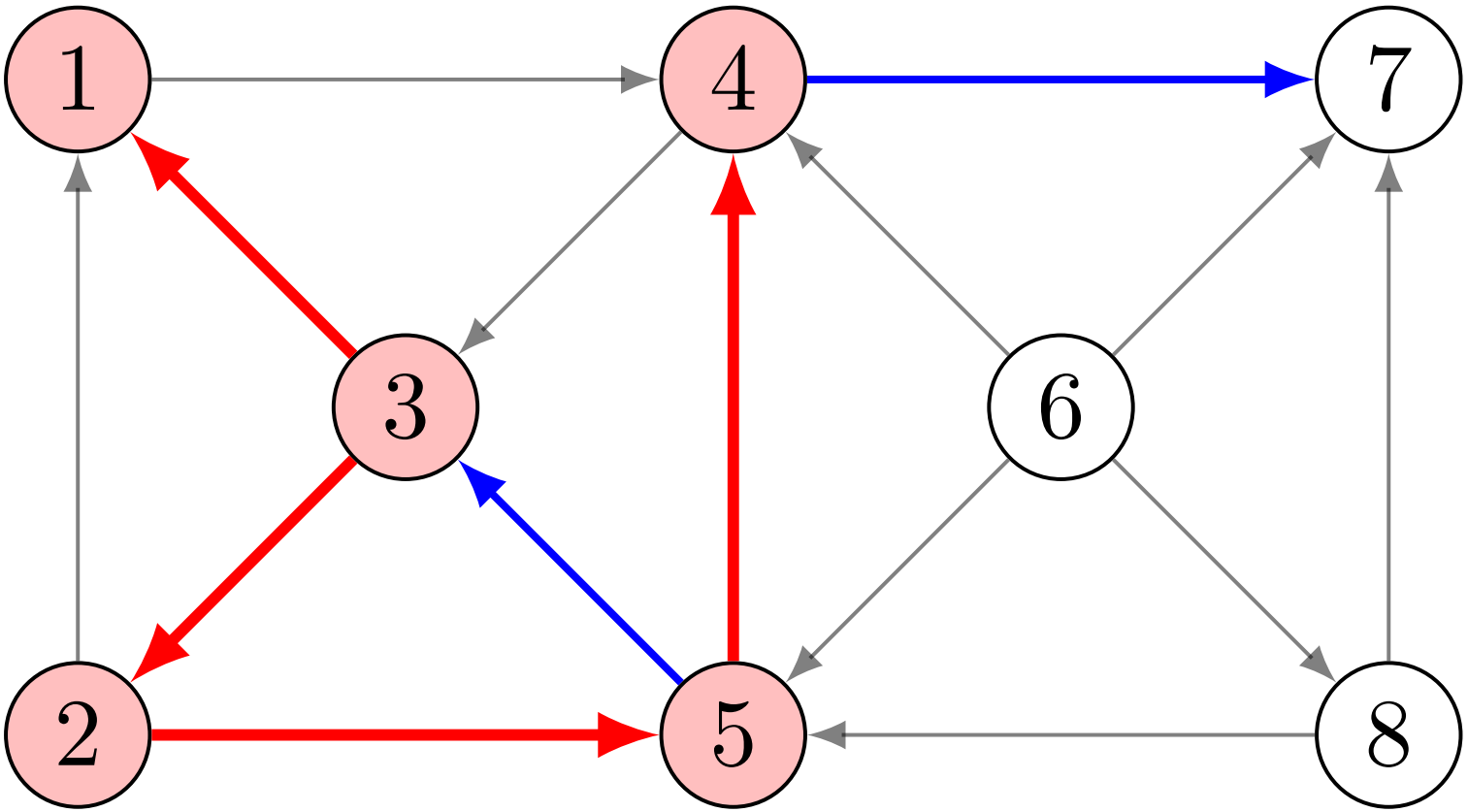
(f) avancée par (1,4), sommet 4 déjà visité

Parcours générique : sommets parcourus et frontière



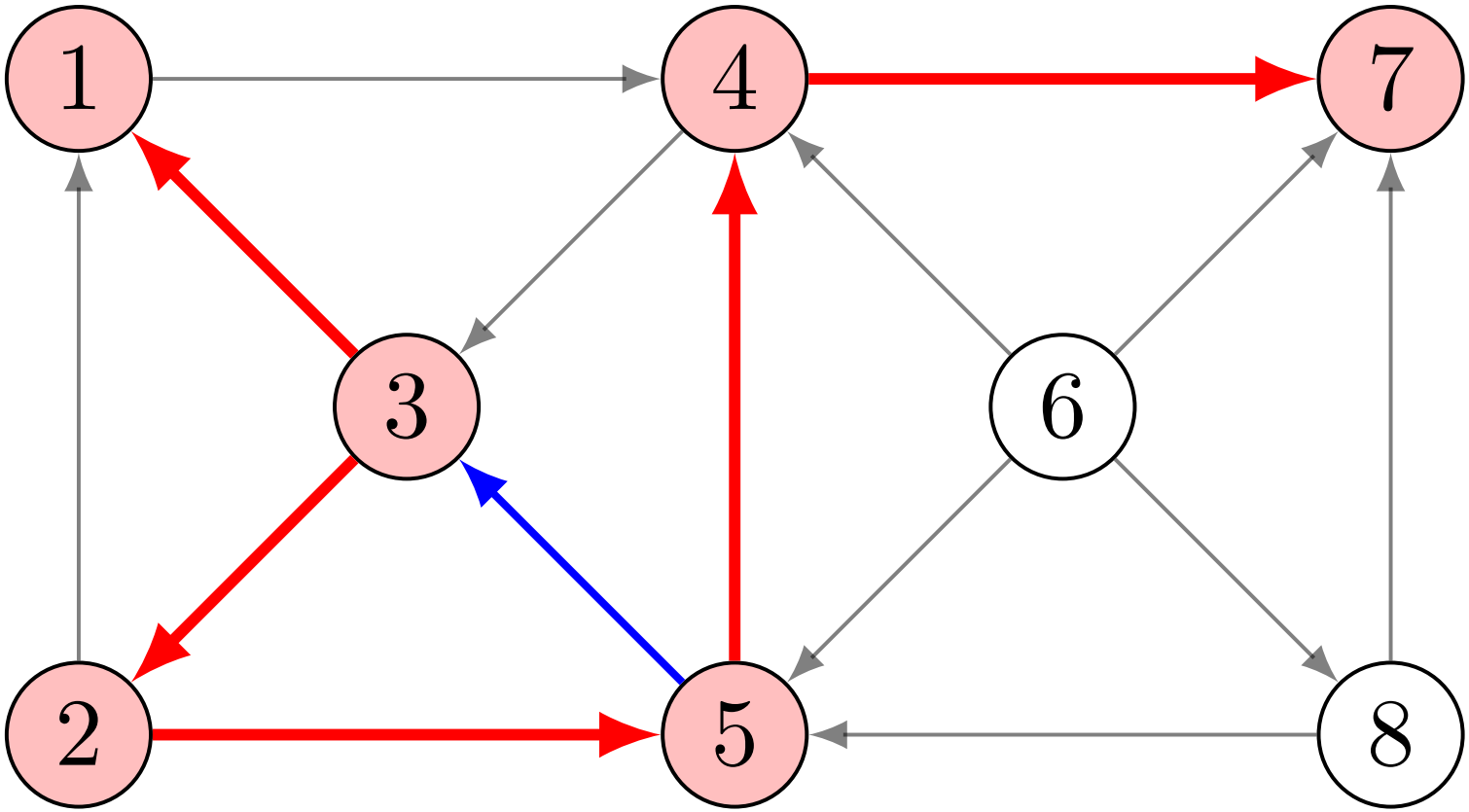
(g) avancée par (4,3), 3 déjà visité

Parcours générique : sommets parcourus et frontière



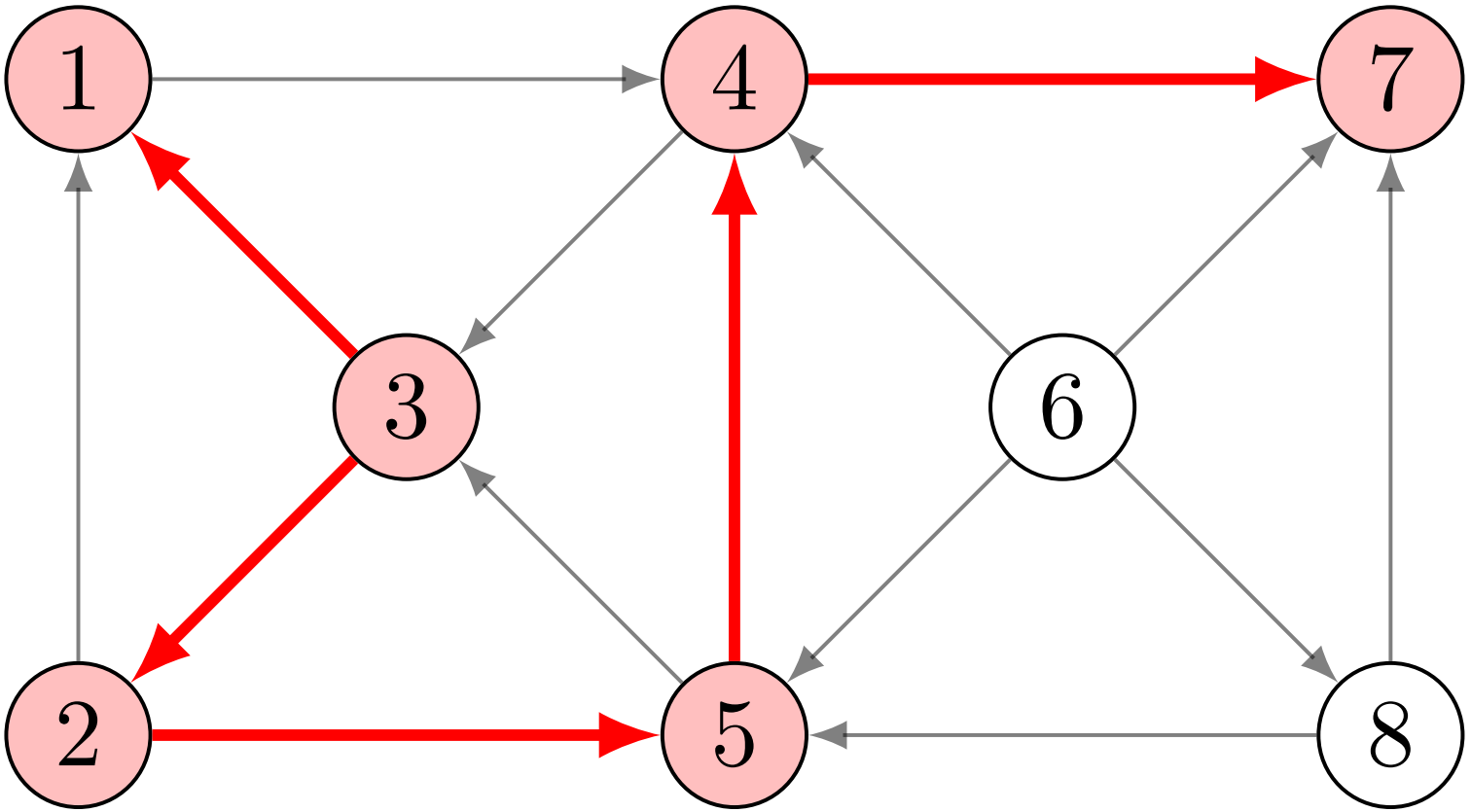
(h) avancée par (2,1), sommet 1 déjà visité

Parcours générique : sommets parcourus et frontière



(i) avancée par (4,7), exploration de 7

Parcours générique : sommets parcourus et frontière



(j) avancée par (5,3), 3 déjà visité.

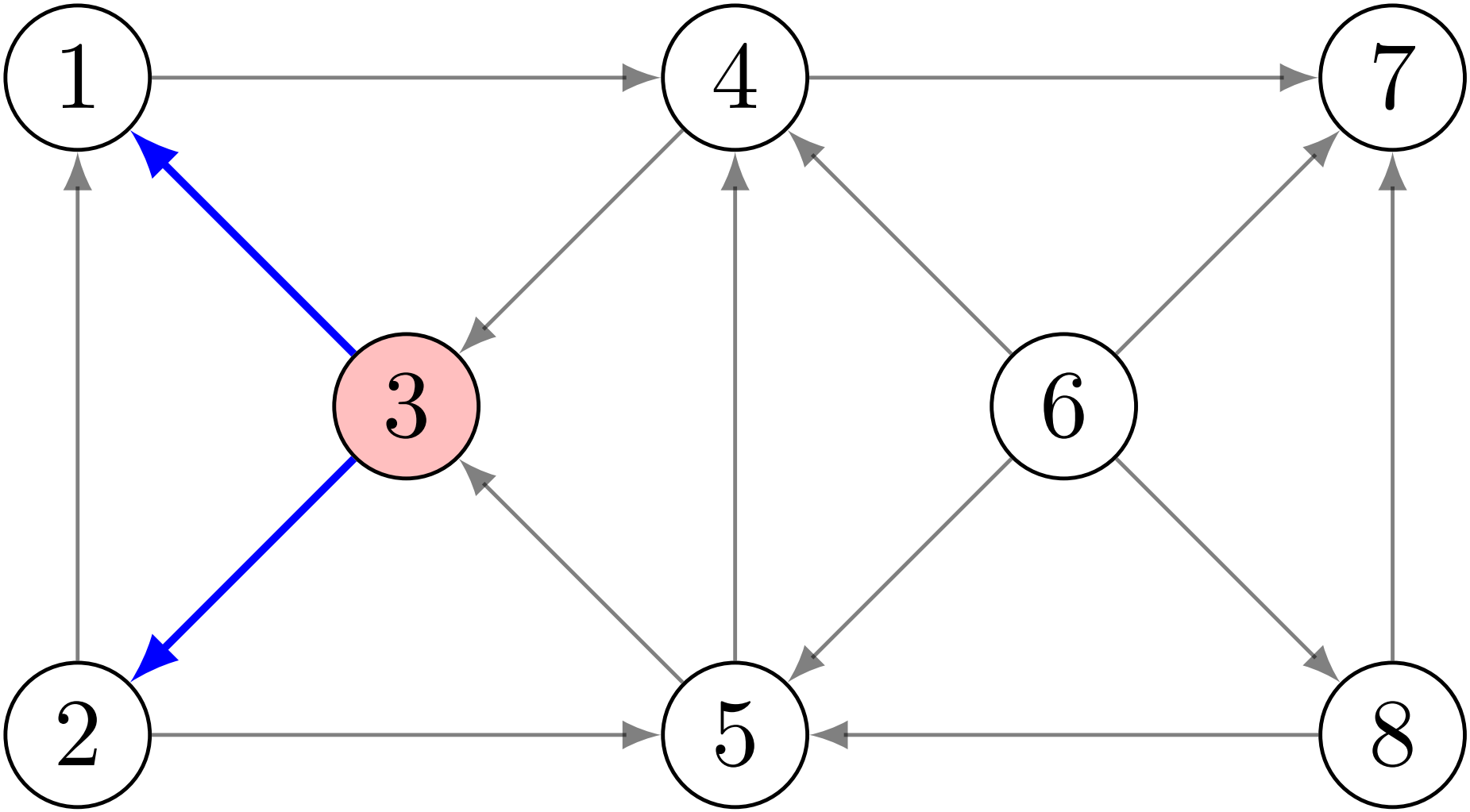
Structure de données insertion-extraction

type *collection* **de** t	référence à un multiensemble fini d'éléments de type t
collection_vide() : *collection*	$[\![\text{collection\_vide}()]\!] = \textbf{ref}\ \emptyset$
est_vide(*collection*) : *bool*	$[\![\text{est\_vide}(\text{c})]\!] =$ vrai si $!\,[\![c]\!] =$ $[\![\text{est\_vide}(\text{c})]\!] =$ faux sinon
insère(t, *collection*) : *void*	$[\![\text{insère}(\text{elt}, \text{c})]\!] = [\![\text{c}]\!] \leftarrow\!! [\![\text{c}]\!] \cup \{[\![\text{elt}]\!]\}$
extrais(*collection*) : t	$\text{extrais}(\text{c}) = e \in\! ! [\![\text{c}]\!], [\![\text{c}]\!] \leftarrow\!! [\![\text{c}]\!] \setminus \{e\}; e$ Précondition : $[\![\text{c}]\!] \neq \langle\rangle$

Parcours générique

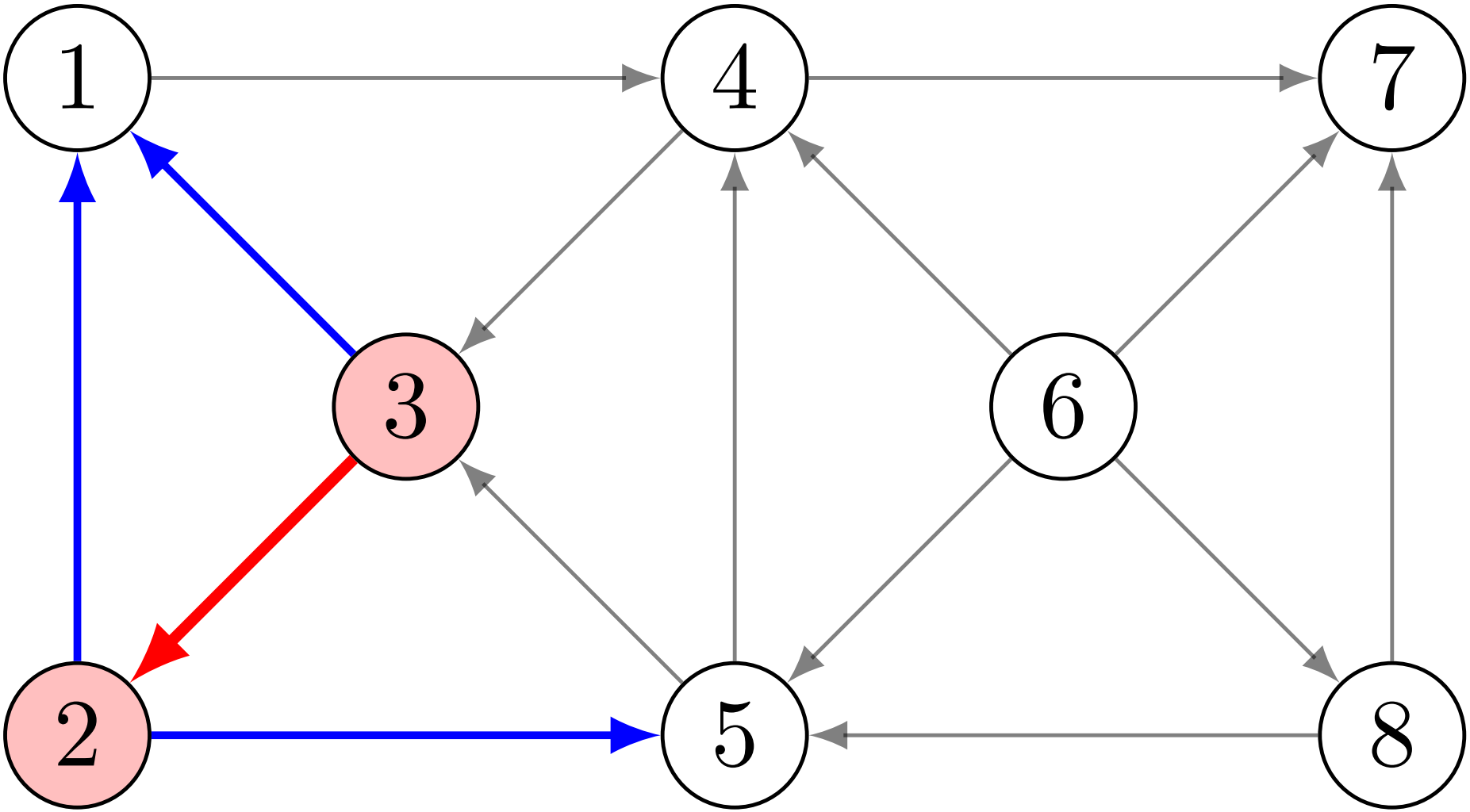
```
1   fonction parcours_générique(graphe g, sommet racine) :
                    tableau d'indices sommets d'(arc ou bien ⊥) :=
2     soit prédécesseur = tableau d'indices sommets(g) de (arc ou bien ⊥)
3     soit frontière := S.collection_vide()
4     soit parcouru := ref {racine}
5
6     soit fonction étends_frontière(sommet u) : void :=
7       pour tout a ∈ arcs_sortants(g, u) faire
8         S.insère(frontière, a)
9
10    soit fonction explore(arc a) : void :=
11      si tête(a) ∈ !parcouru alors retourner
12      parcouru ← !parcouru ∪ {tête(a)}
13      prédécesseur[tête(a)] ← a
14      étends_frontière(tête(a))
15
16    prédécesseur[racine] := ⊥
17    étends_frontière(racine)
18    tant que ¬ S.estVide(frontière) faire
19      explore(S.extrais(frontière))
```

Parcours en largeur : file



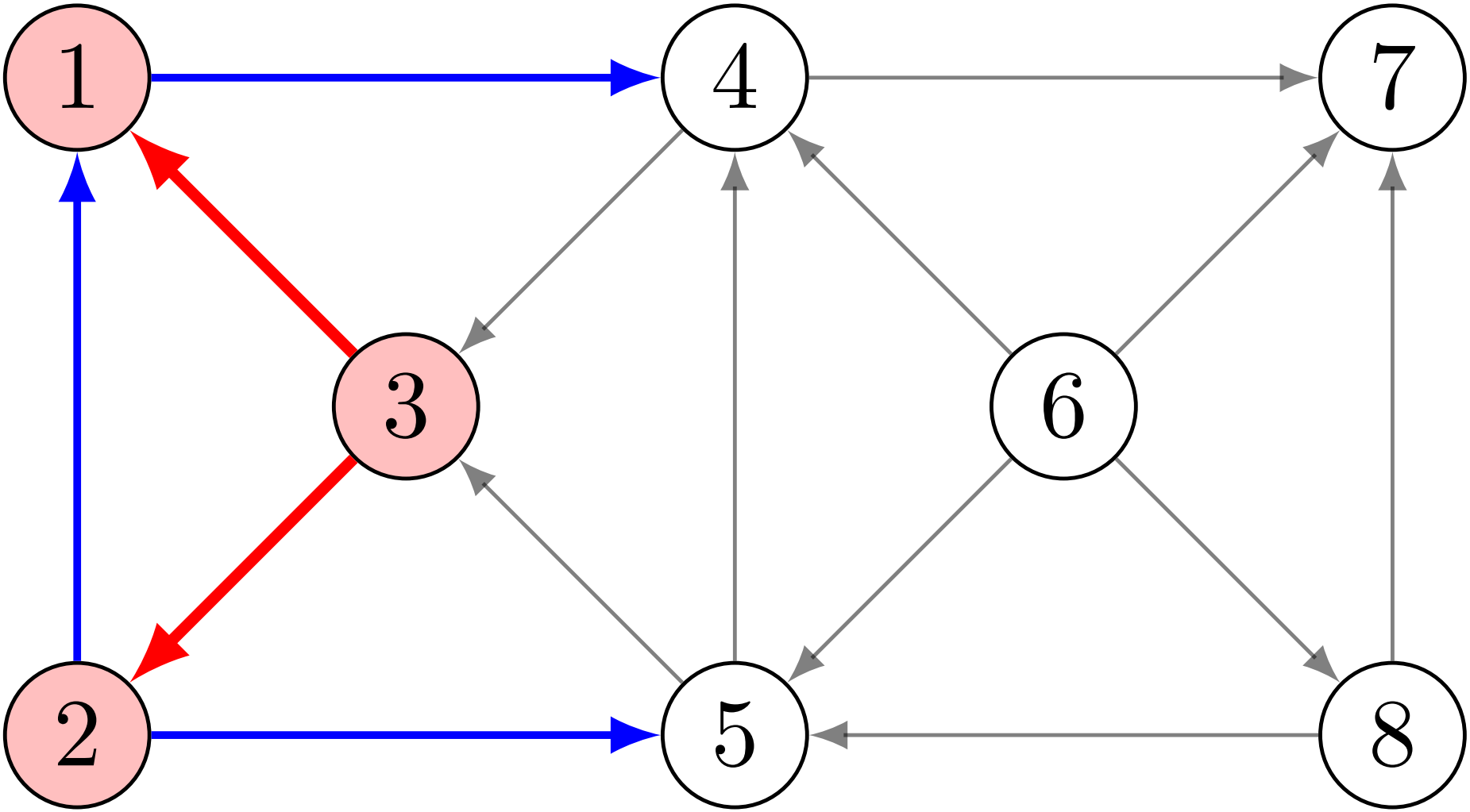
(a) Frontière : $\langle (3,1),(3,2) \rangle$

Parcours en largeur : file



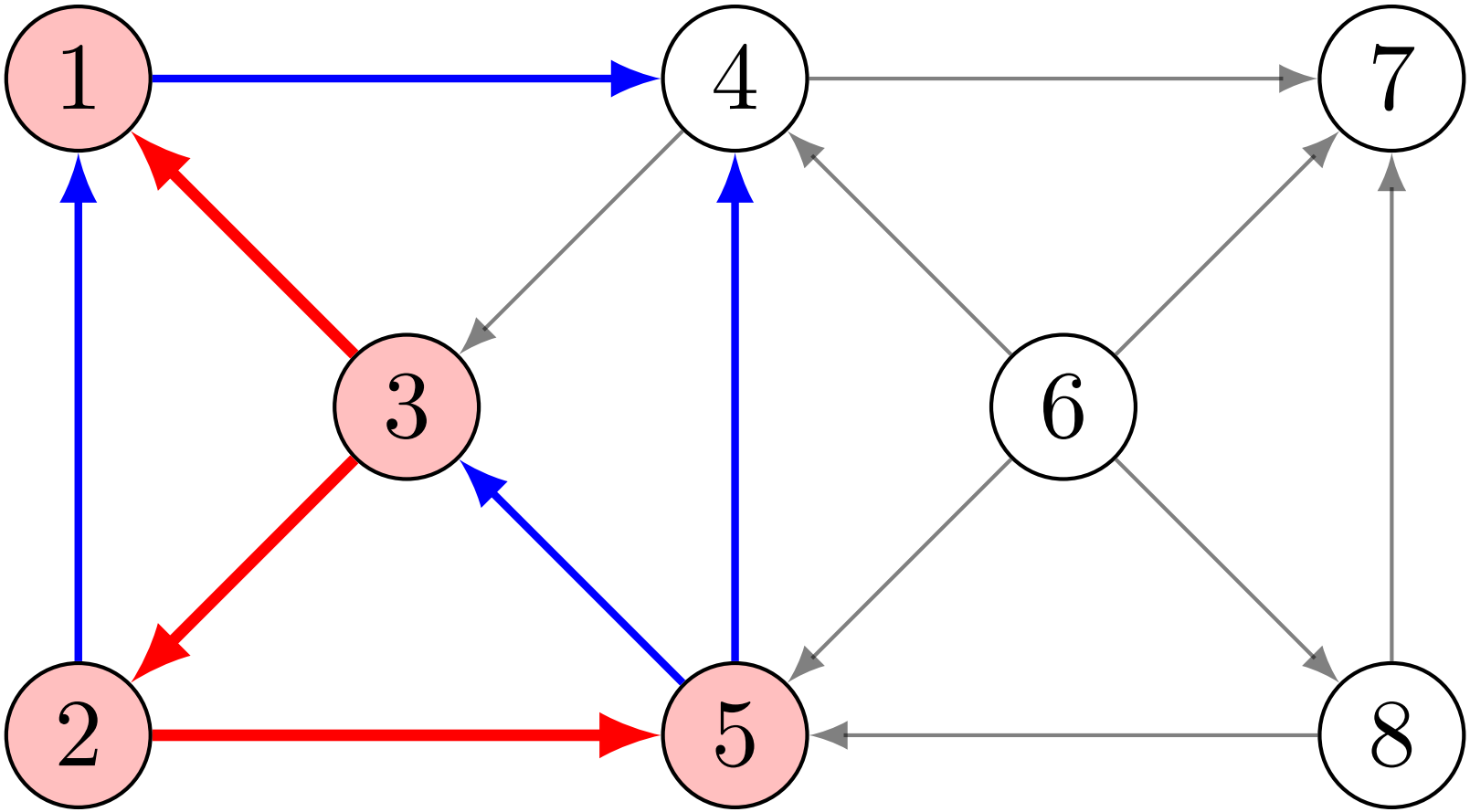
(b) Frontière : $\langle (2,1), (2,5), (3,1) \rangle$

Parcours en largeur : file



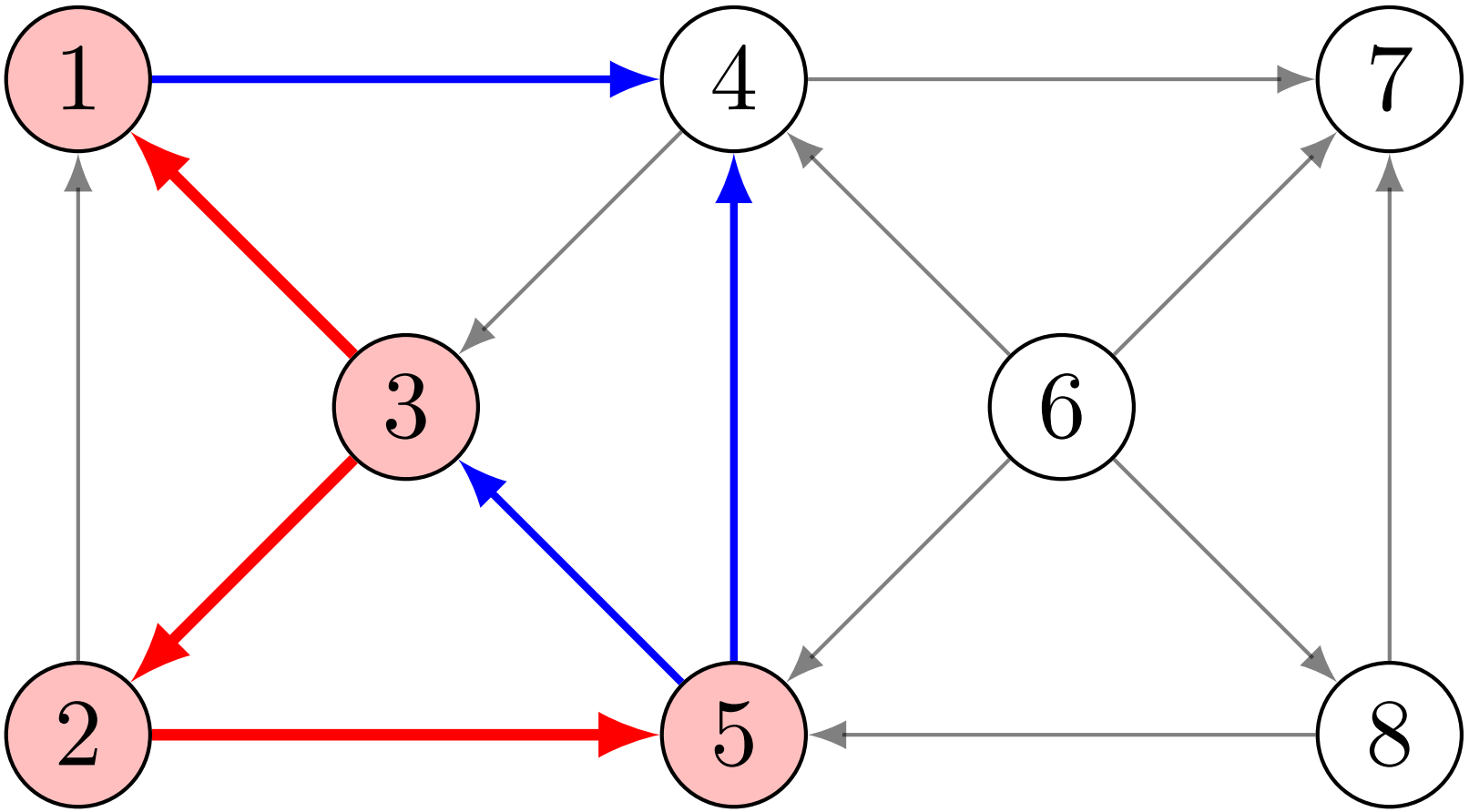
(c) Frontière : $\langle (1,4),(2,1),(2,5) \rangle$

Parcours en largeur : file



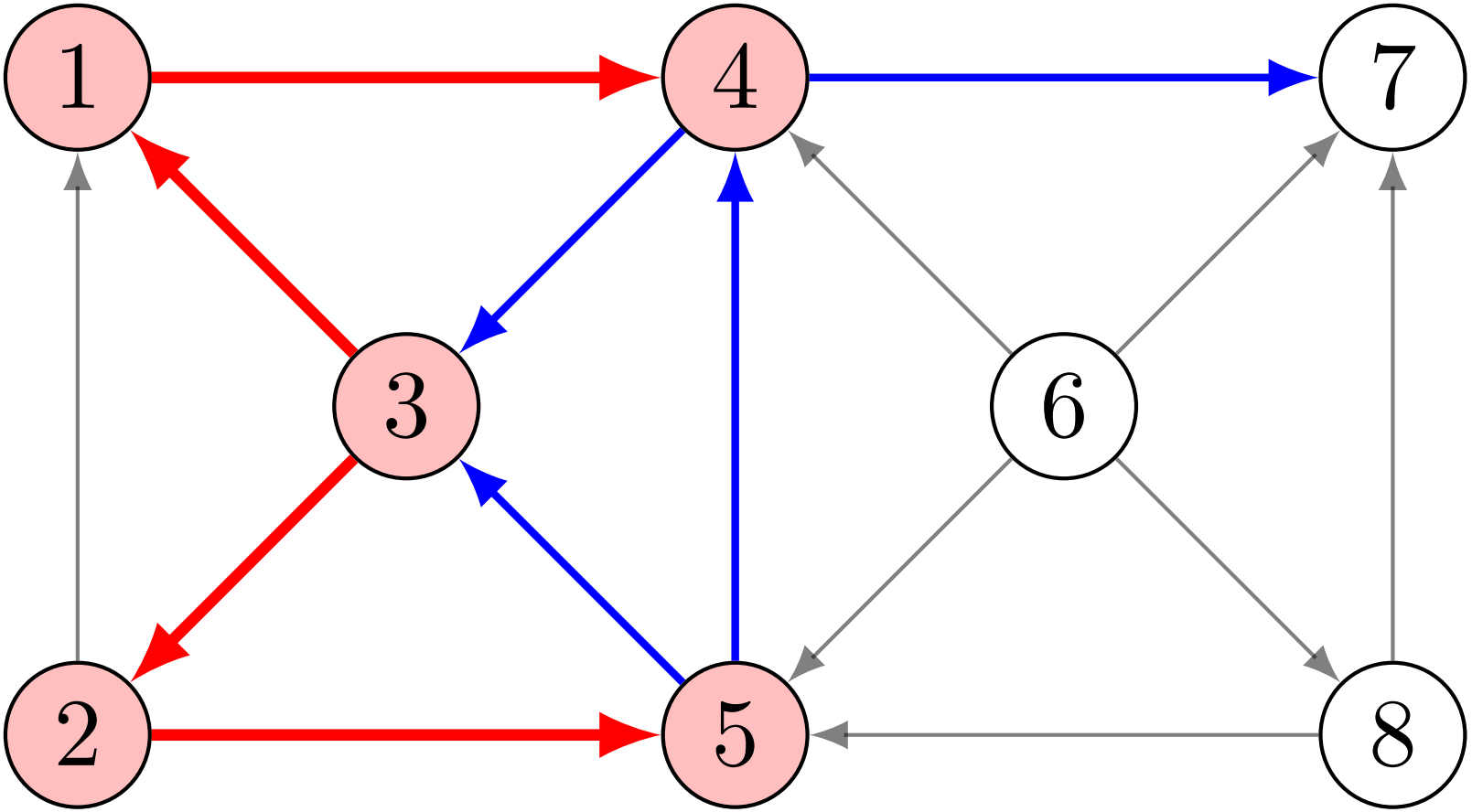
(d) Frontière : $\langle (5,3), (5,4), (1,4), (2,1) \rangle$

Parcours en largeur : file



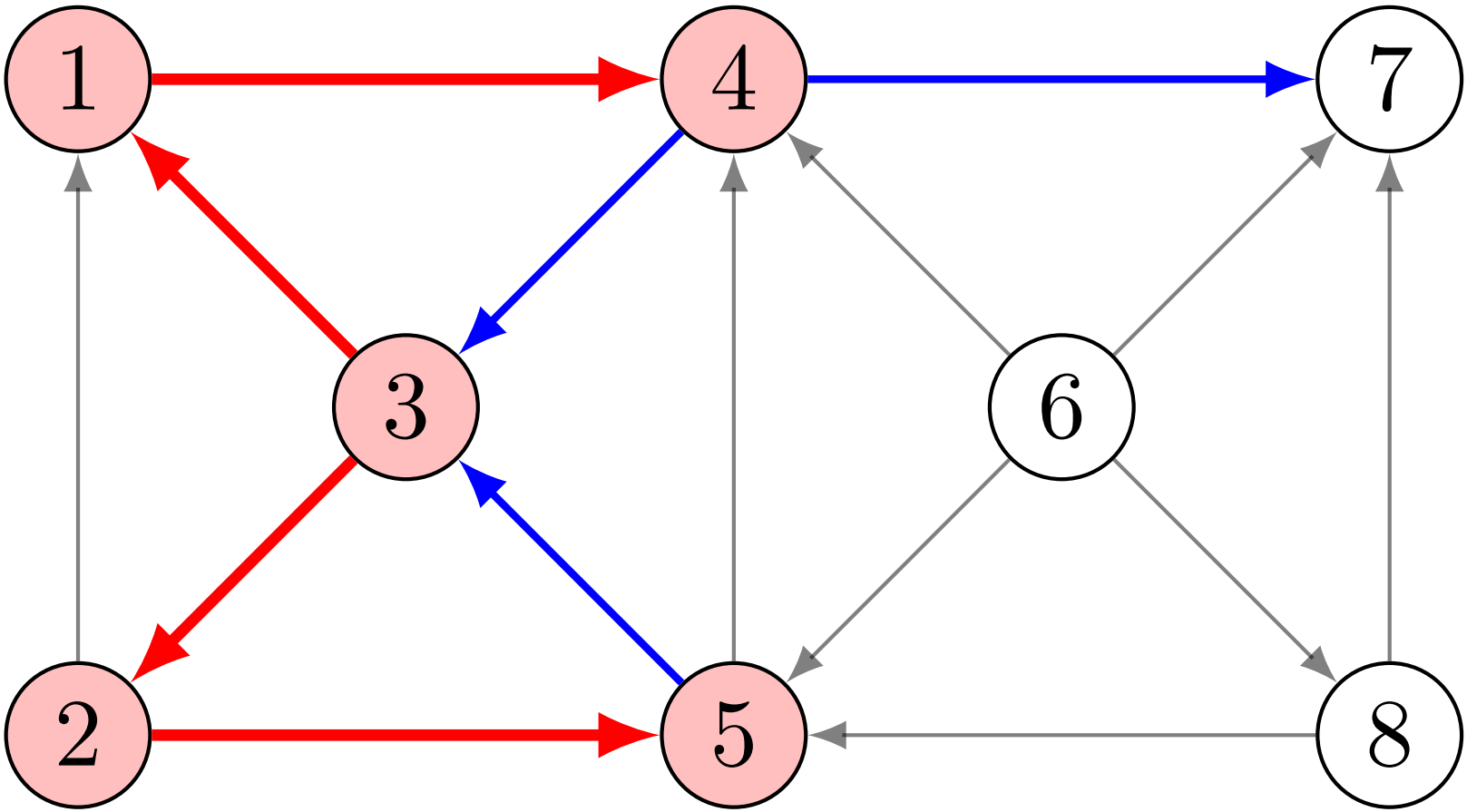
(e) Frontière : $\langle (5,3),(5,4),(1,4) \rangle$

Parcours en largeur : file



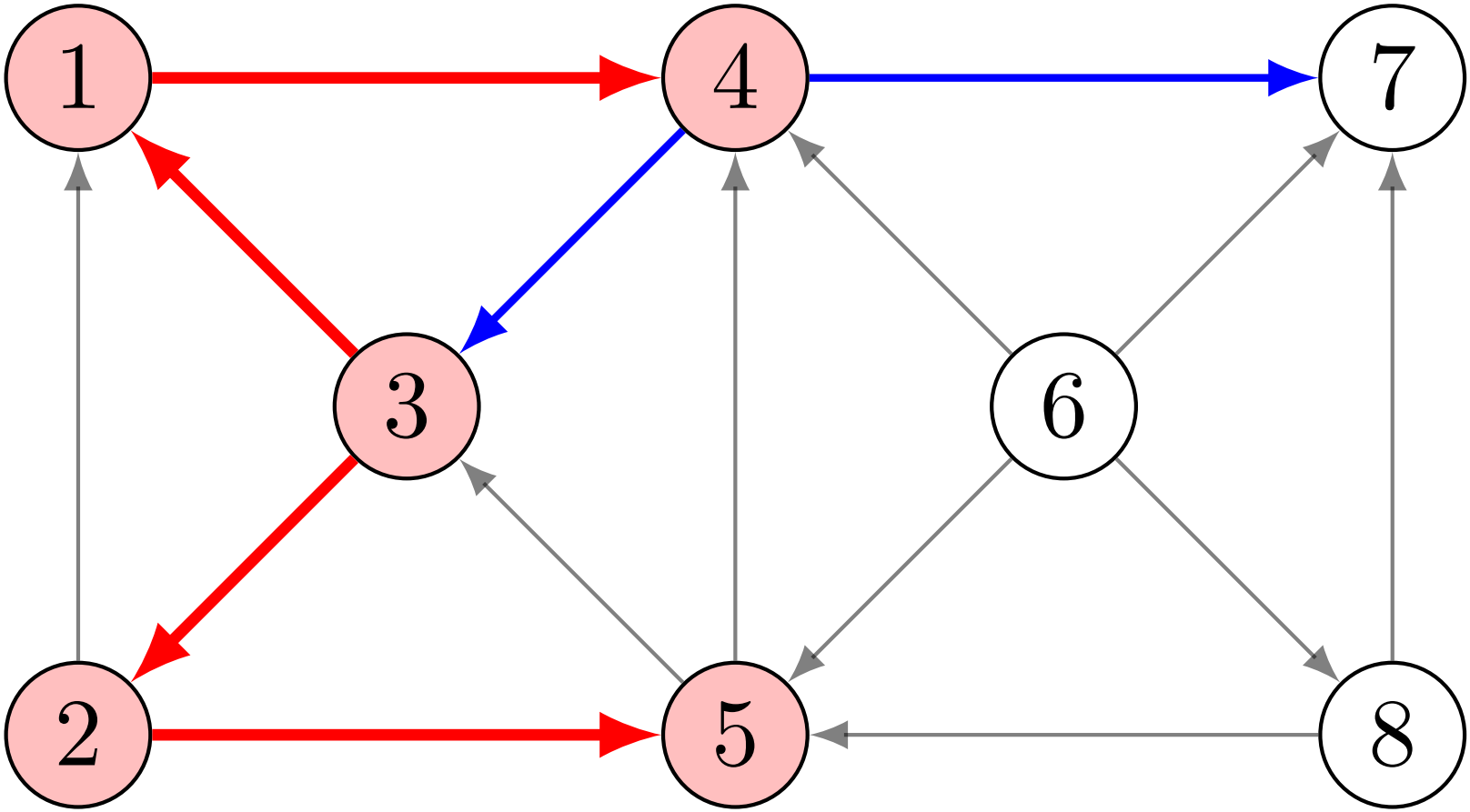
(f) Frontière : $\langle (4,3),(4,7),(5,3),(5,4) \rangle$

Parcours en largeur : file



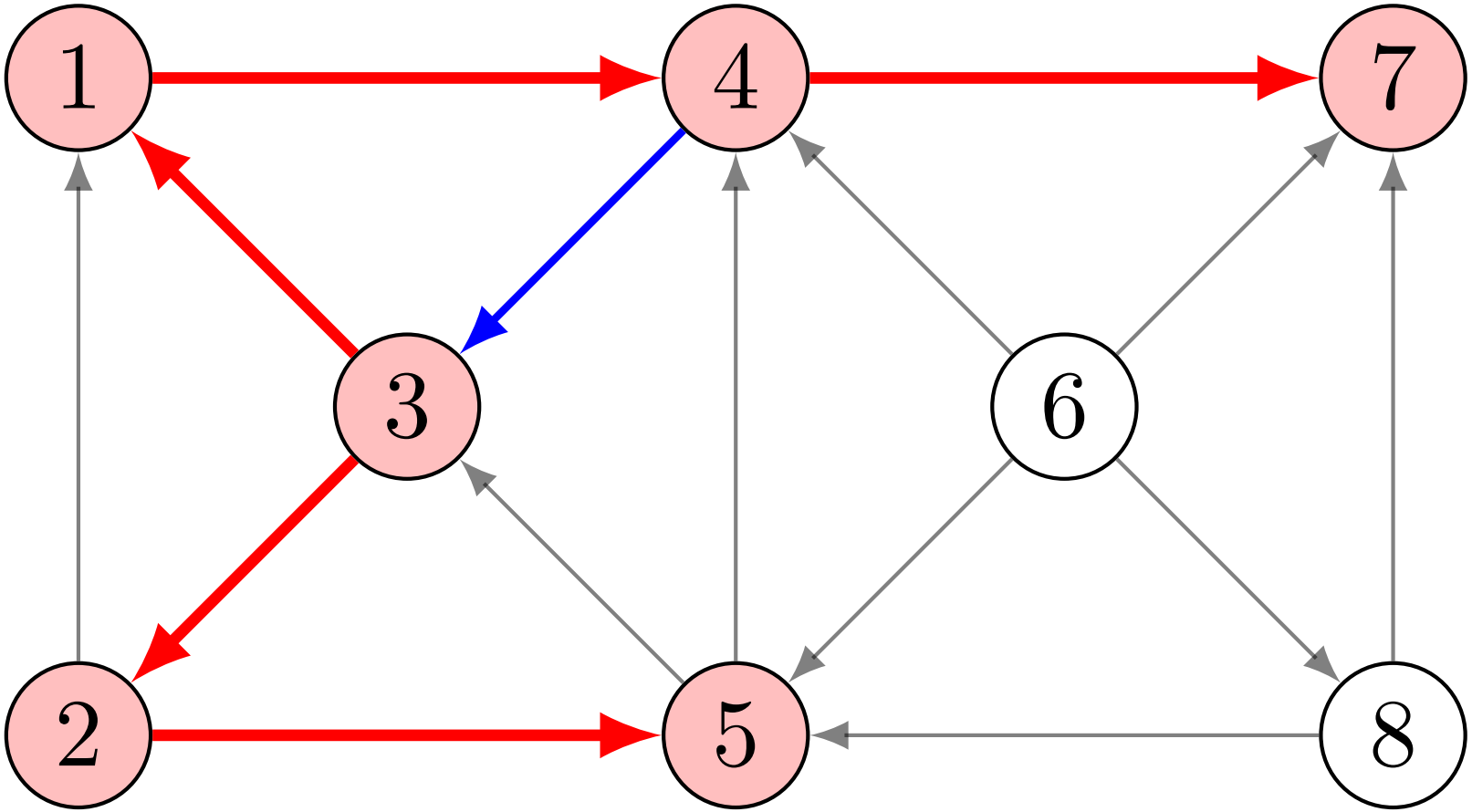
(g) Frontière : $\langle (4,3),(4,7),(5,3) \rangle$

Parcours en largeur : file



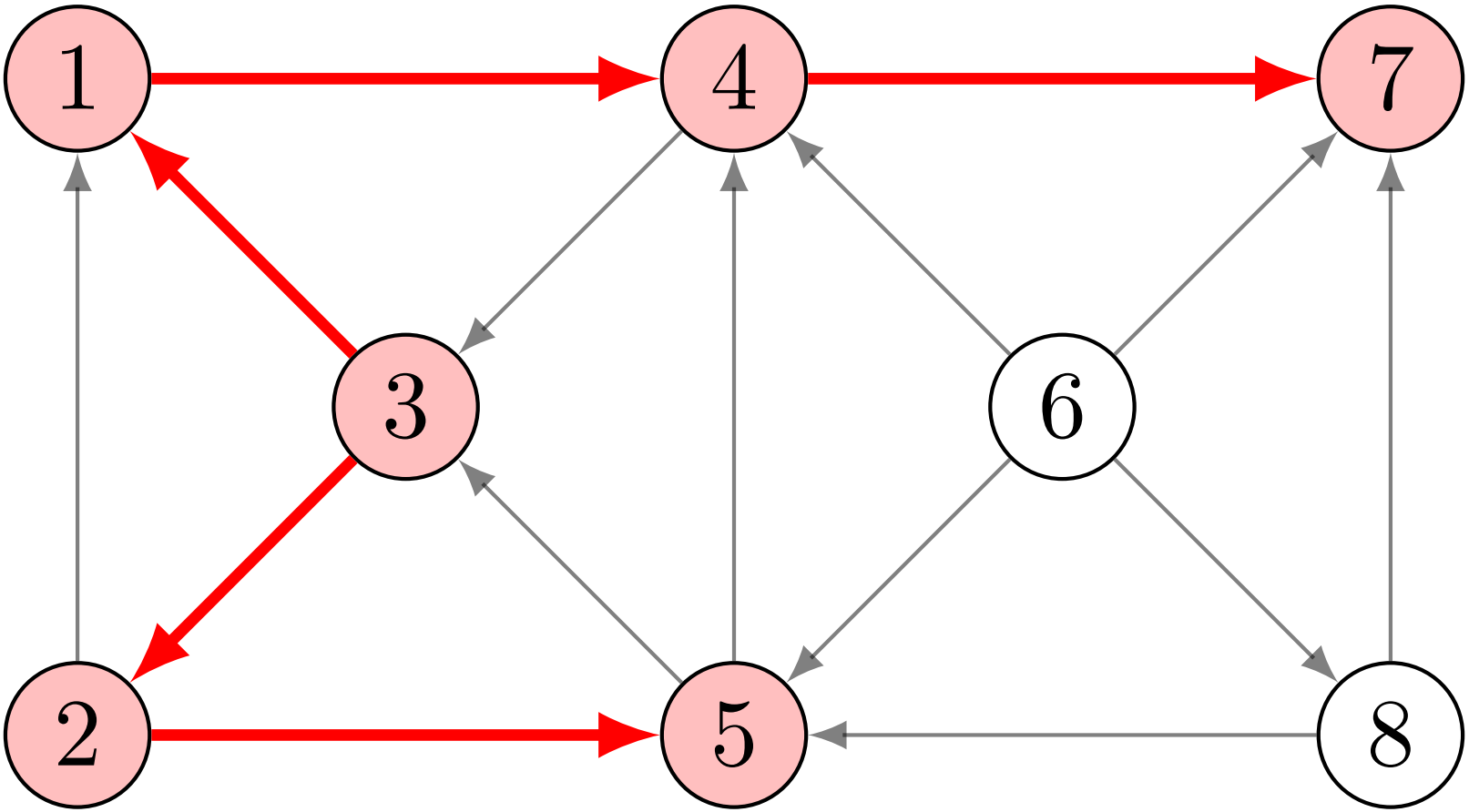
(h) Frontière : $\langle (4,3),(4,7) \rangle$

Parcours en largeur : file



(i) Frontière : $\langle (4, 3) \rangle$

Parcours en largeur : file

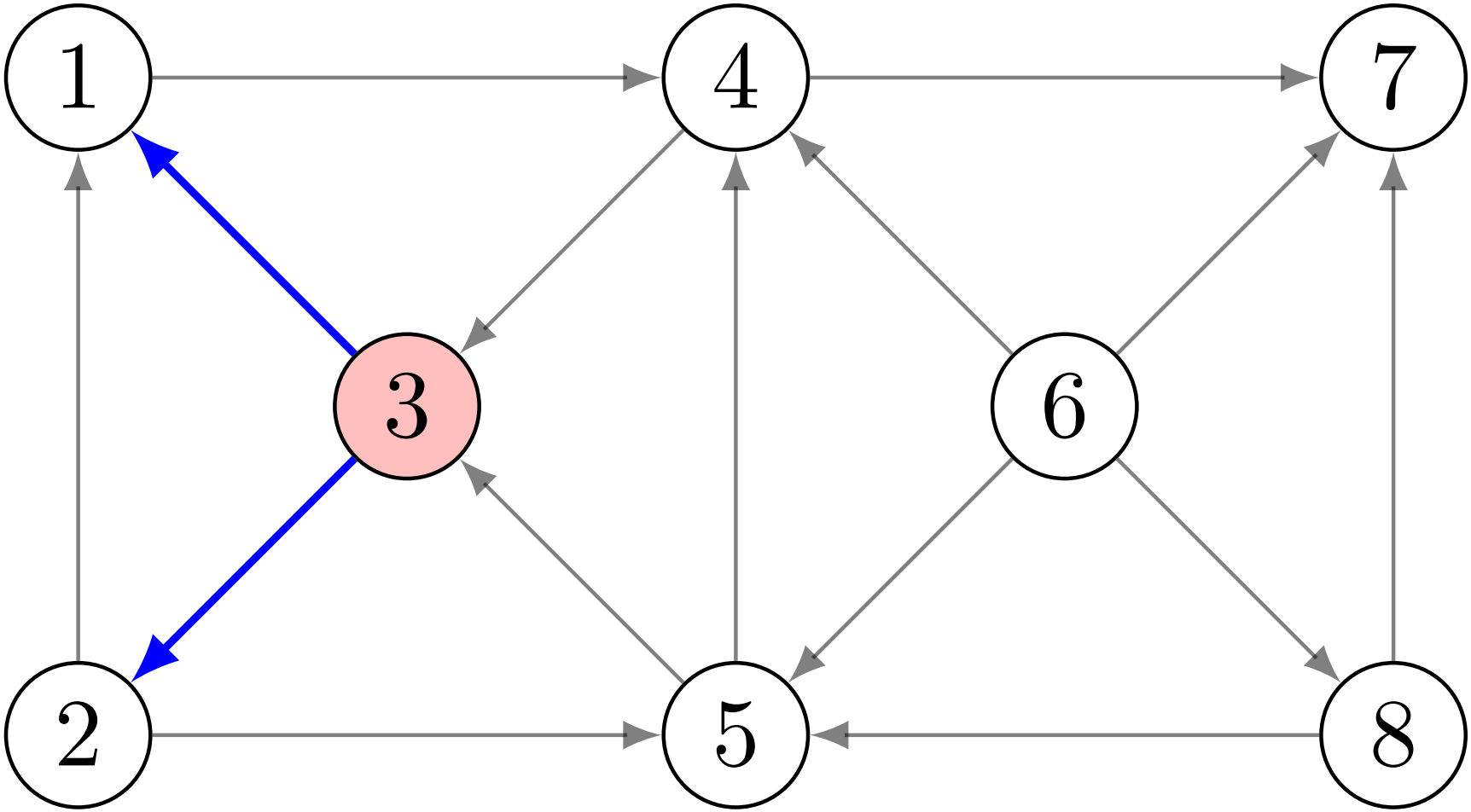


(j) Frontière : $\langle \rangle$

Parcours en largeur : pseudo-code

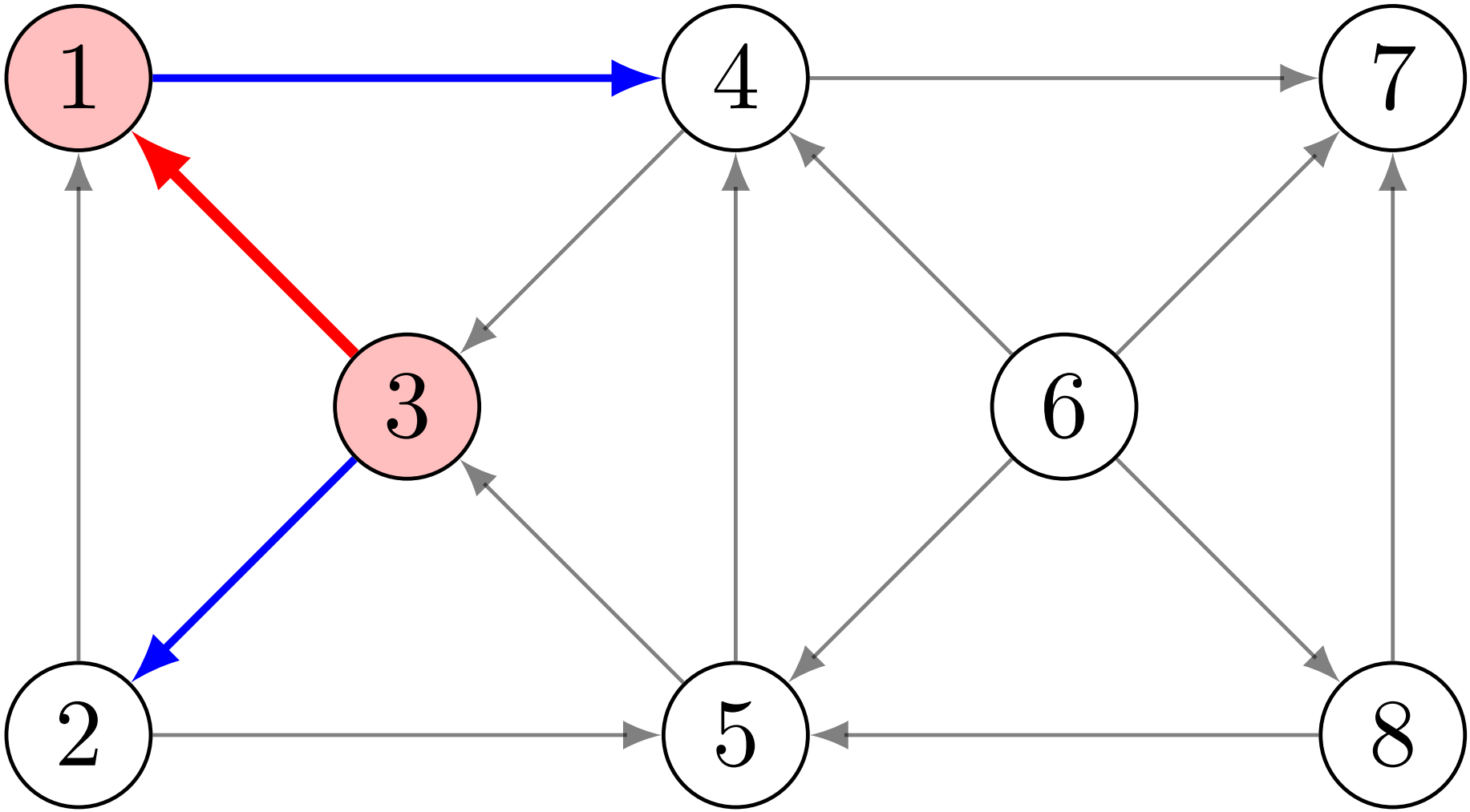
```
fonction parcours_en_largeur(M, s)
   n := nombre_sommets(M)
   H := graphe_vide(n) #  graphe sans arcs
   F := file d'attente vide
   couleur := tableau de longueur n rempli de « blanc »
   couleur[s] := rouge
   insérer(F, s)
   Tant que F ≠ ∅ faire
      u := extraire(F)
      Pour v de 0 à n-1 faire
         Si (M[u,v] = 1 et couleur[v] = blanc ) alors
            couleur[v] := rouge
            H[u,v] := 1
            insérer(F, v)
         FinSi
      FinPour
   FinTantQue
   retourner(H)          #  graphe des chemins minimaux
```

Parcours en profondeur : pile



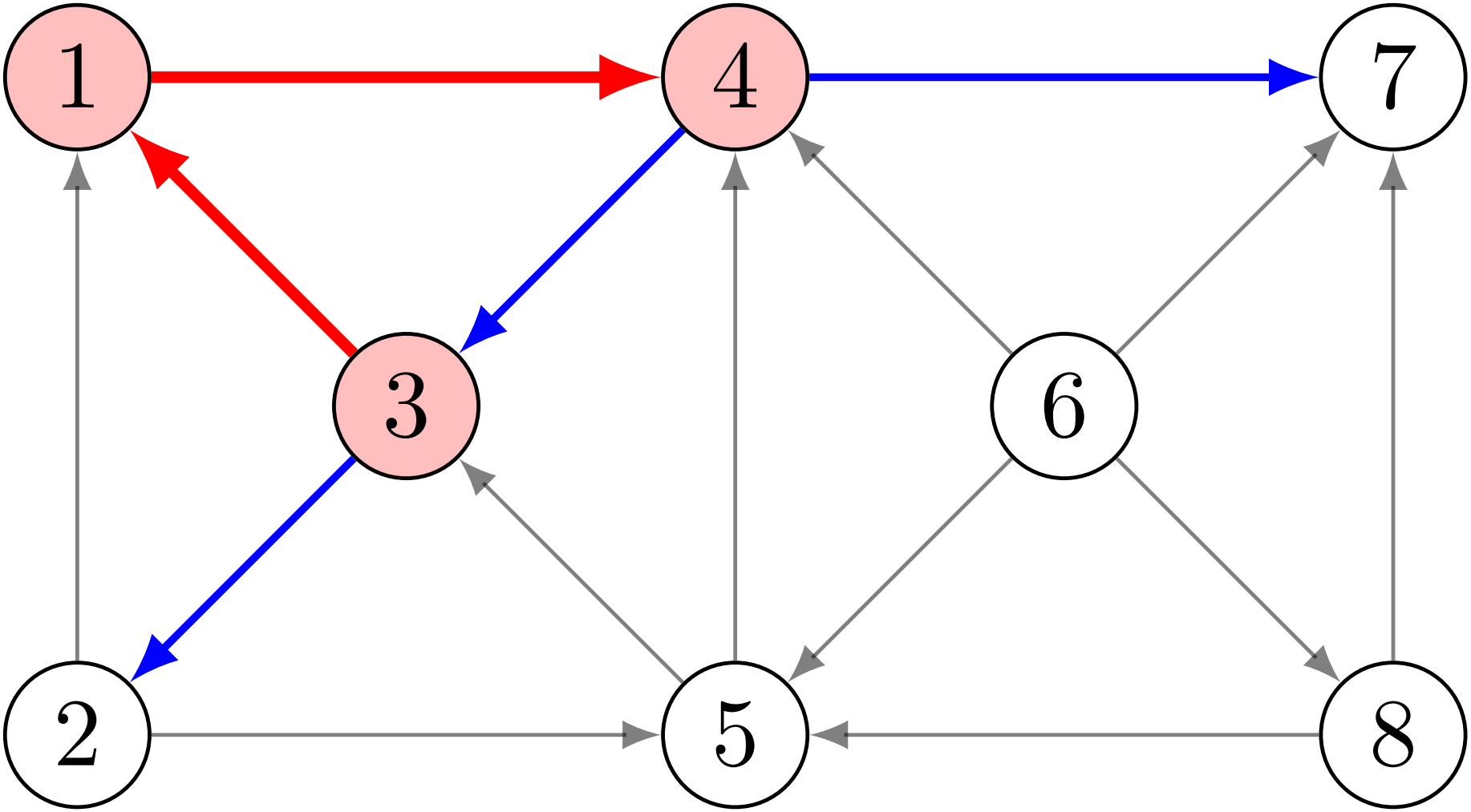
(a) Frontière : $\langle (3,1),(3,2) \rangle$

Parcours en profondeur : pile



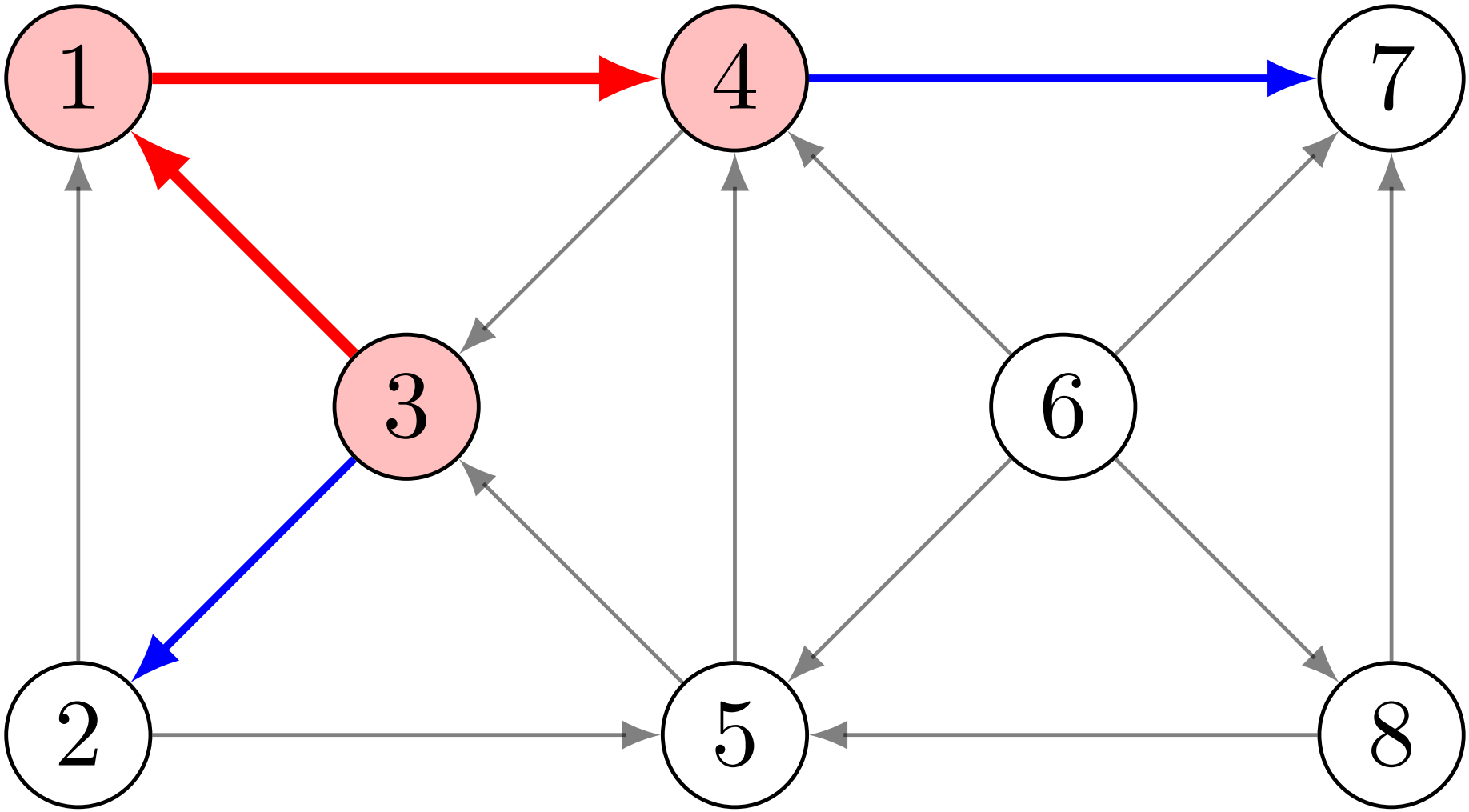
(b) Frontière : $\langle (1,4),(3,2) \rangle$

Parcours en profondeur : pile



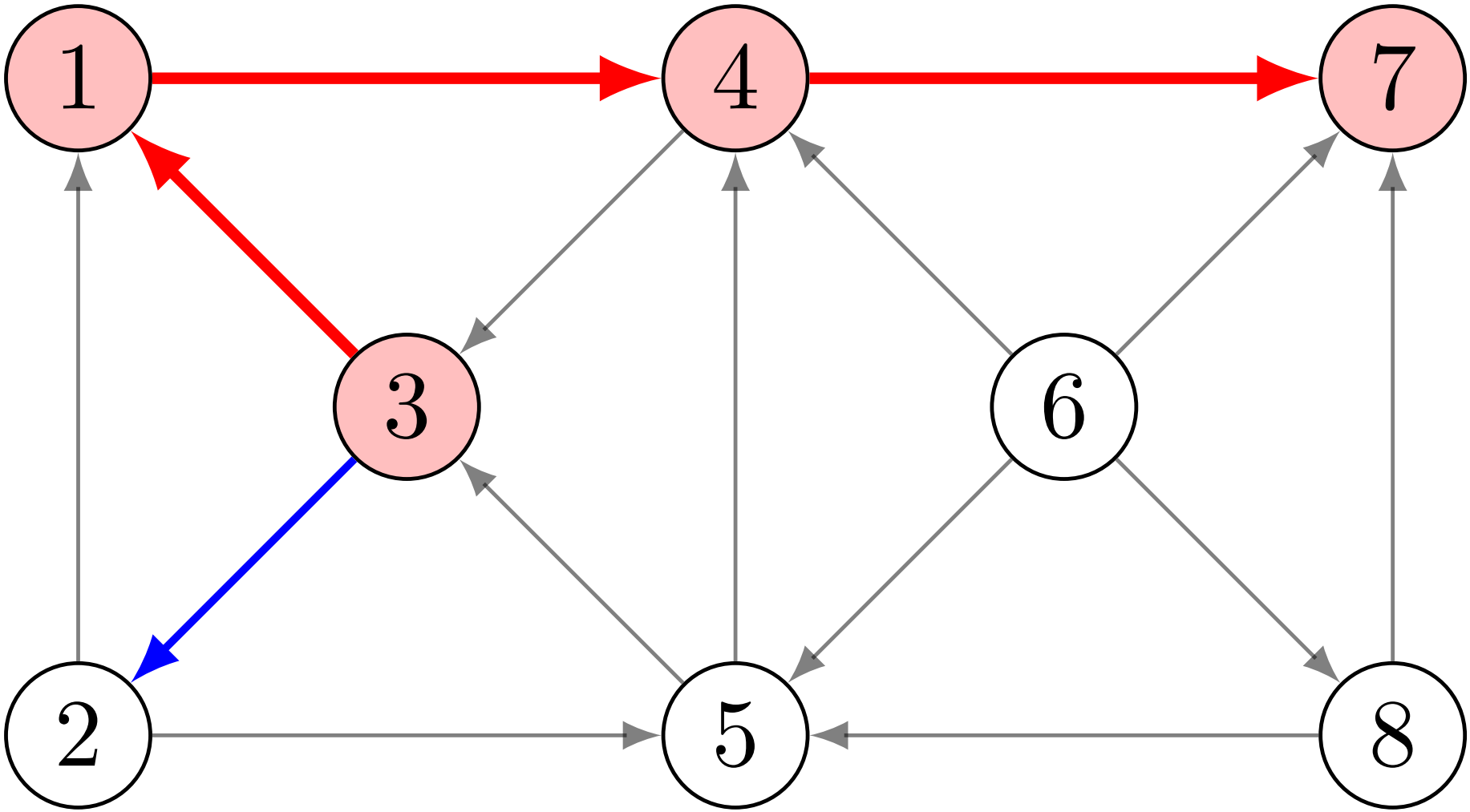
(c) Frontière : $\langle (4,3),(4,7),(3,2) \rangle$

Parcours en profondeur : pile



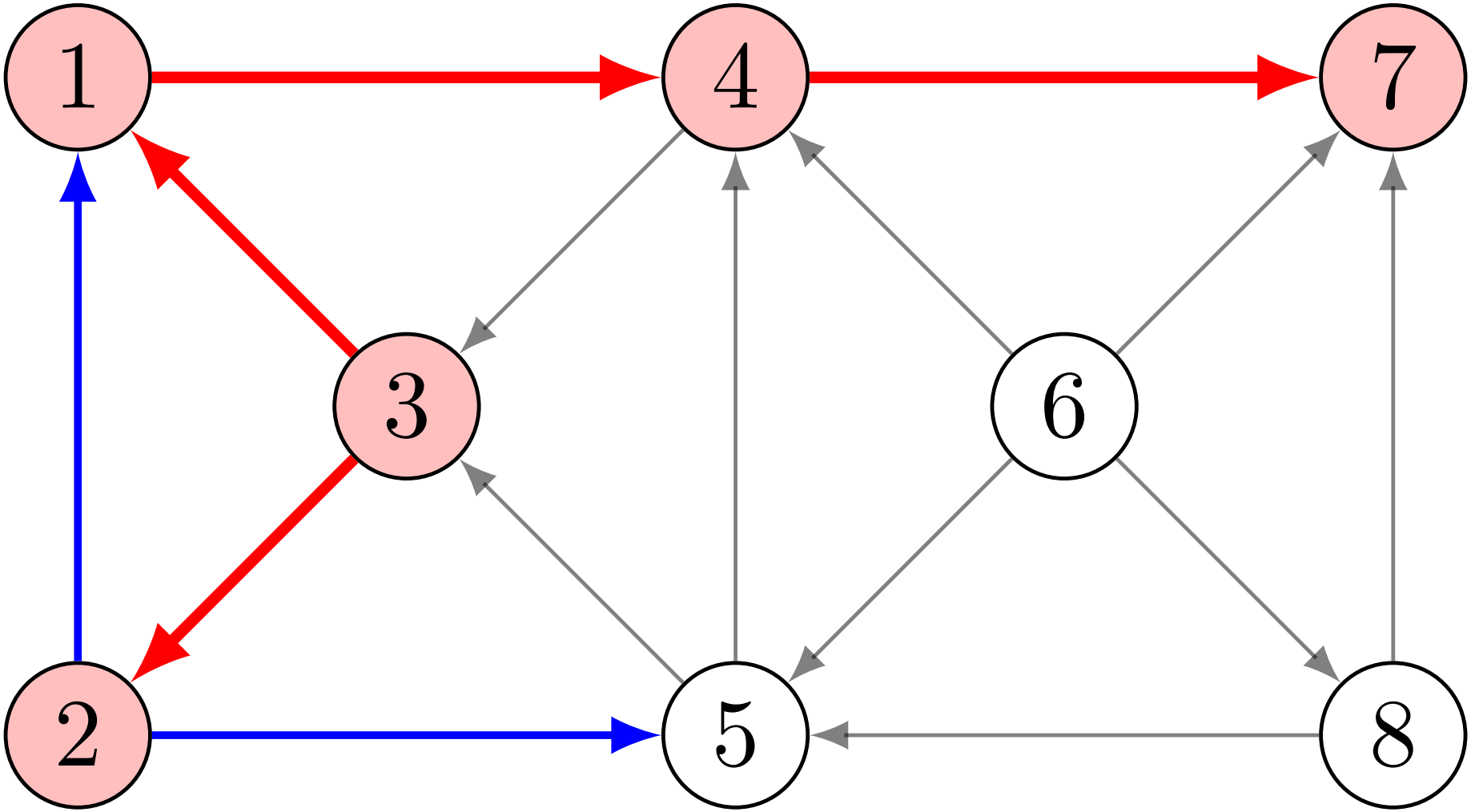
(d) Frontière : $\langle (4,7),(3,2) \rangle$

Parcours en profondeur : pile



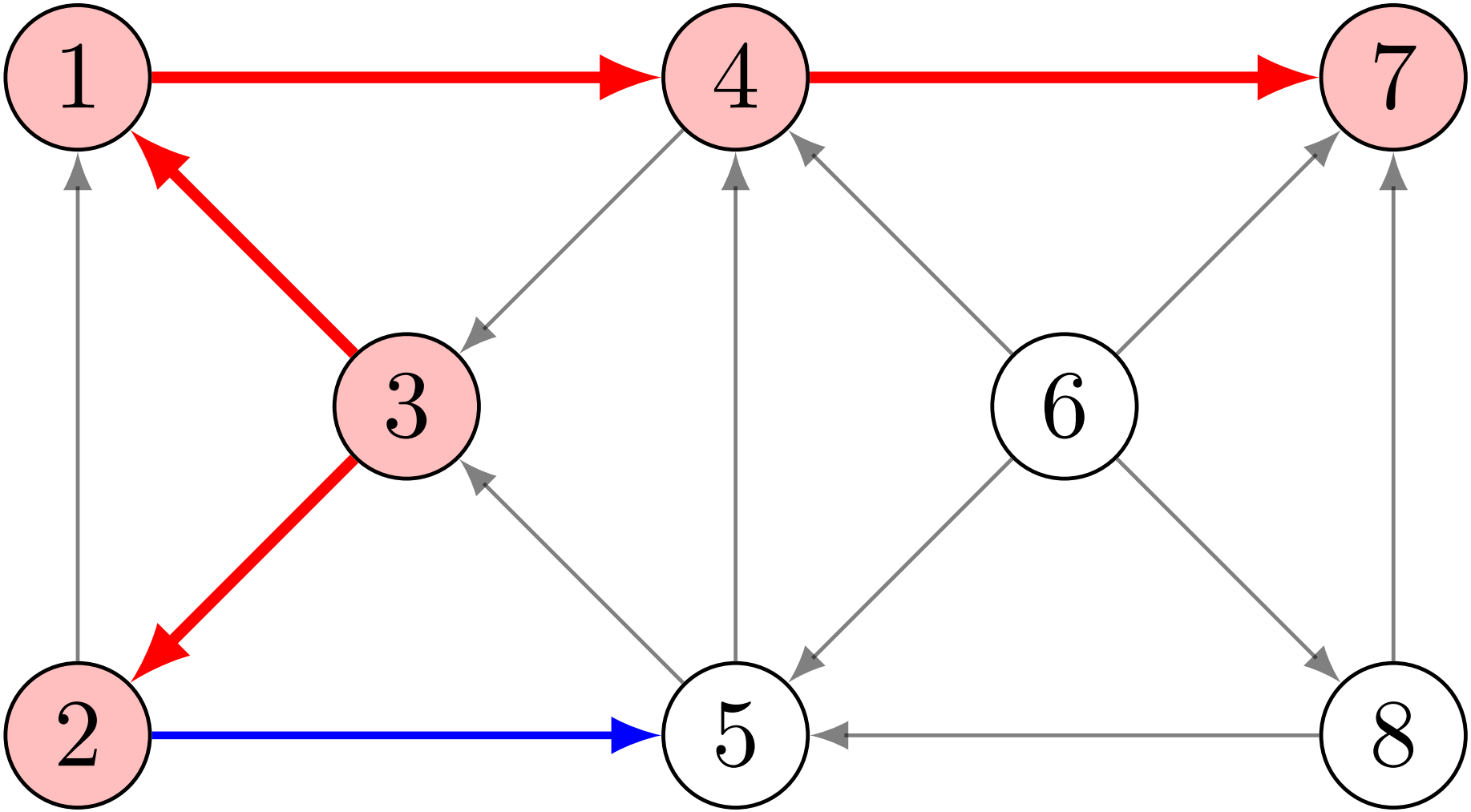
(e) Frontière : $\langle (3, 2) \rangle$

Parcours en profondeur : pile



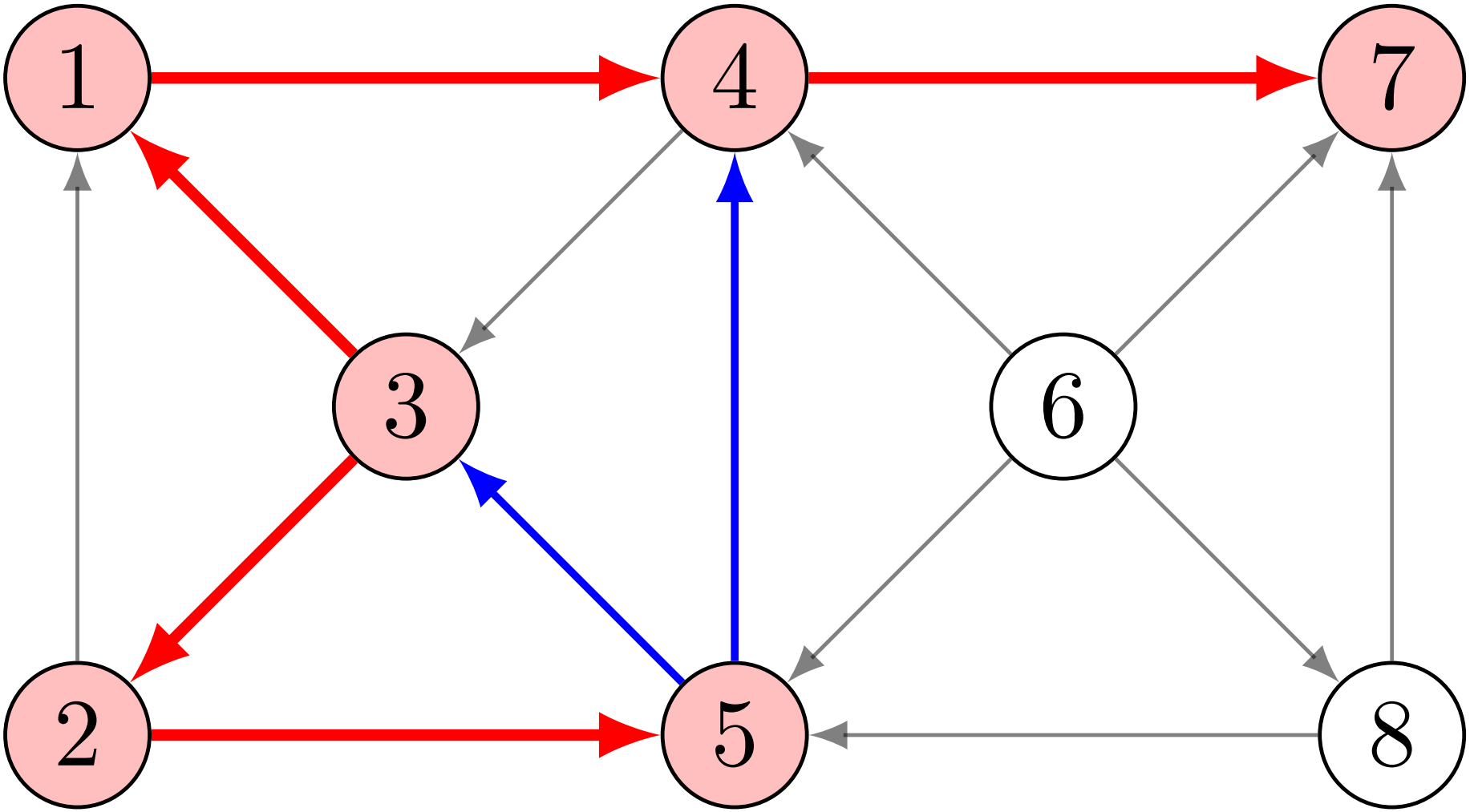
(f) Frontière : $\langle (2,1), (2,5) \rangle$

Parcours en profondeur : pile



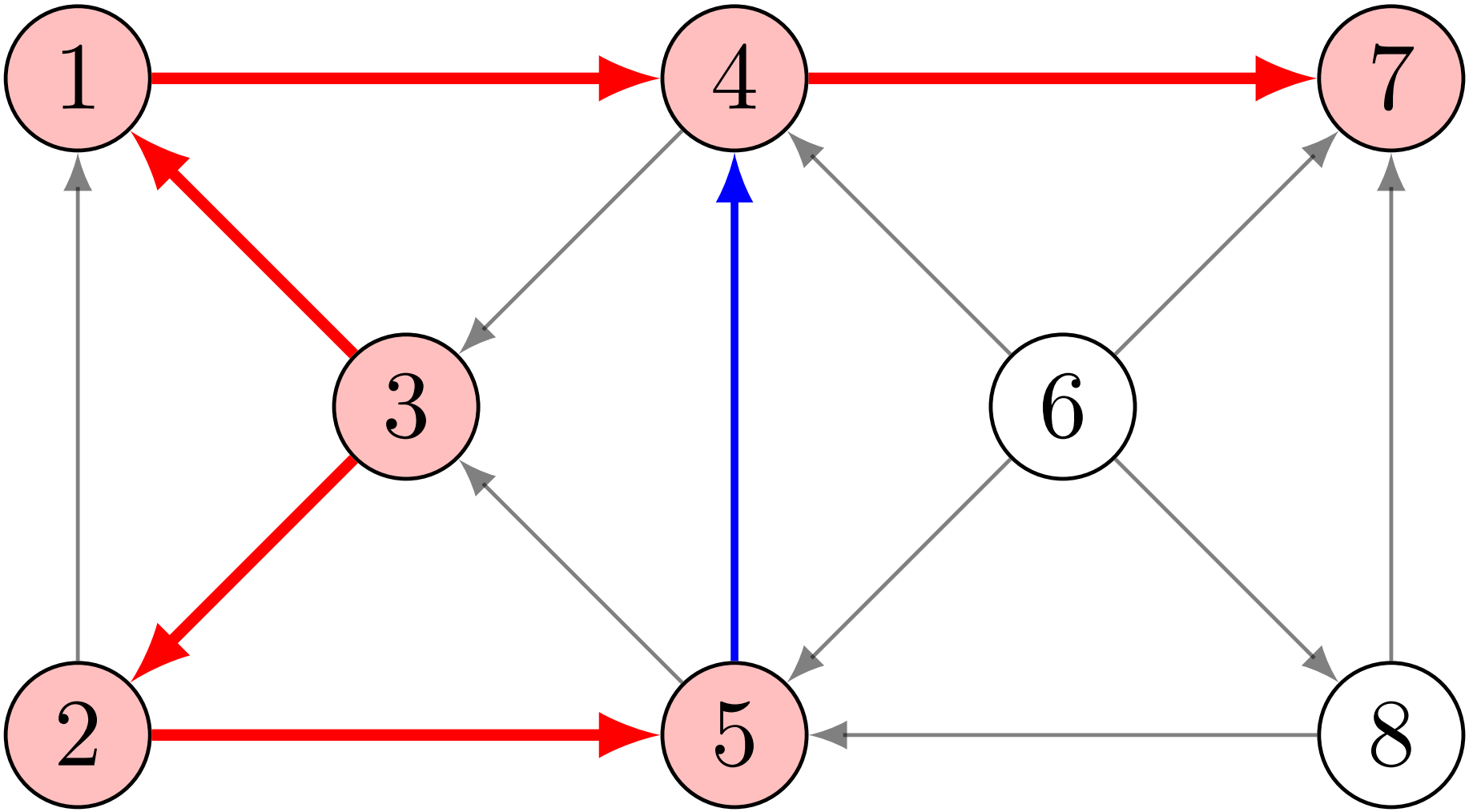
(g) Frontière : $\langle (2, 5) \rangle$

Parcours en profondeur : pile



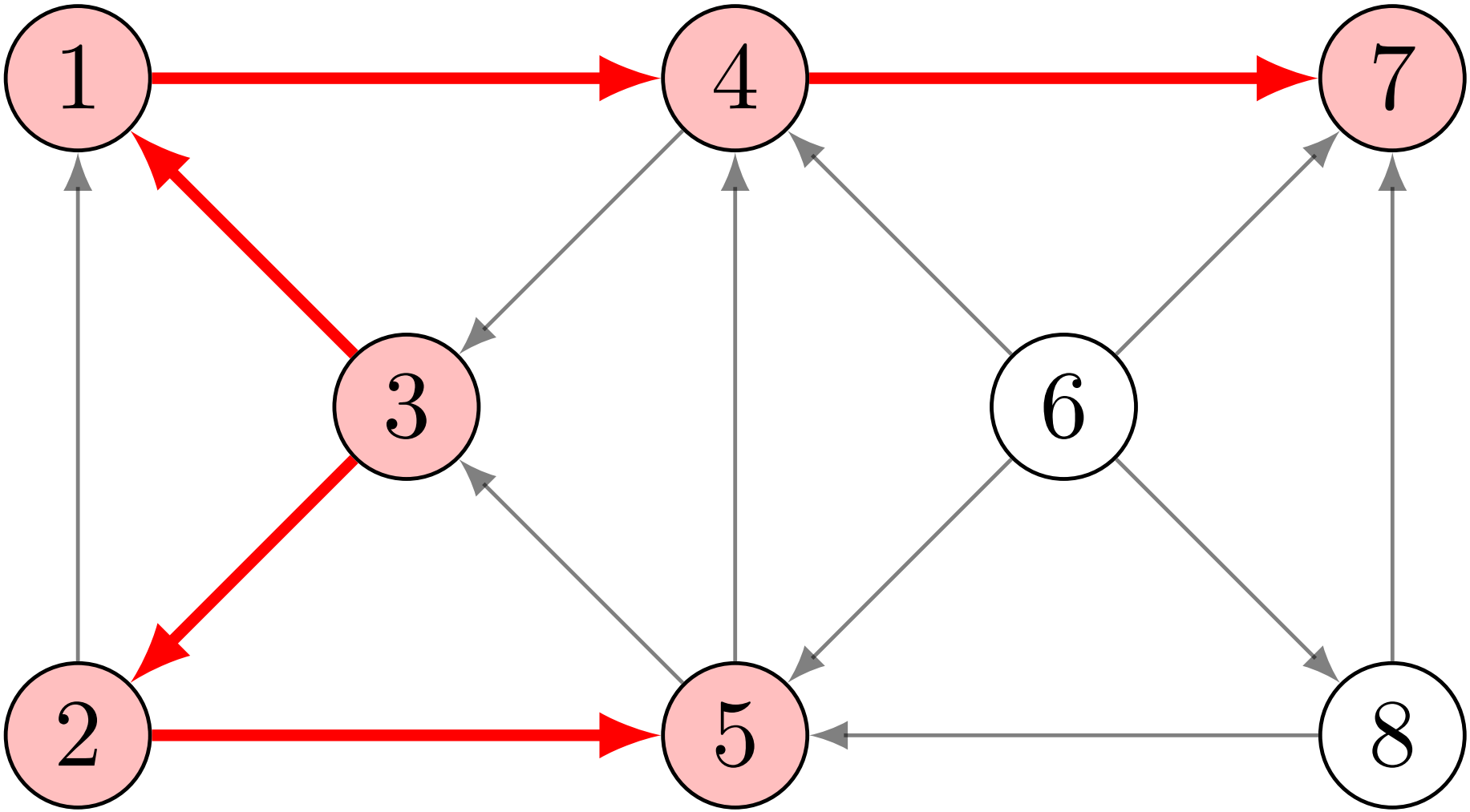
(h) Frontière : $\langle (5,3),(5,4) \rangle$

Parcours en profondeur : pile



(i) Frontière : $\langle (5, 4) \rangle$

Parcours en profondeur : pile



(j) Frontière : $\langle\rangle$

Parcours en profondeur : écriture récursive

```
1   fonction parcours_en_profondeur(graphe g, sommet s) : tableau d'indice sommet d'(arc ou bien ⊥)
2     soit predecesseur := tableau d'indice sommets(g) de (arc ou bien ⊥)
3     soit parcouru := {s}
4     predecesseur[s] ← ⊥
5     soit fonction explore(arc e) : void =
6       soit v := tete(e)
7       si v ∉ parcouru alors
8         parcouru ← parcouru ∪ {v}; predecesseur[v] ← e
9         pour tout arc ∈ arc_sortants(g, v) faire explore(arc)
10    pour tout arc ∈ arc_sortants(g, s) faire explore(arc)
11    retourner predecesseur
```