IN103
Résolution de problèmes algorithmiques
une approche fondée sur les structures de données

Alexandre Chapoutot

Année académique 2023-2024
https://perso.ensta-paris.fr/~chapoutot/teaching/in103/

ENSTA
IP PARIS

Plan du cours

1. Connectivité des (di)graphes
   - Composantes connexes
   - Composantes fortement connexes
2. (di)graphes pondérés
3. Arbre couvrant de poids minimum
4. Plus court chemin

Connectivité des (di)graphes

Notion de connexité

Exemple d'application : réseaux sociaux

Les liens entre les différents membres peuvent être modélisés par

- **un graphe dirigé** (digraphe), par exemple, pour Twitter ou Strava. Une personne peut suivre le fil d'une autre sans réciprocité.
- **un graphe non dirigé**, par exemple, pour Facebook. Deux personnes sont dans le même groupe d'amis.

Définition

Un (di)graphe est (fortement) connecté si entre toutes paires de sommets (s_1, s_2) il existe un (chemin) chaîne.

Vocabulaire, pour un graphe $G = (S, A)$

- une **composante** est un **sous-graphe maximal connecté** de G.
- Un **point d'articulation** de G est un sommet dont la suppression augmente le nombre de composantes connexes.
- Un **isthme** est une arête dont la suppression a le même effet.

Calcul des composantes connexes d'un graphe non orienté

Idée

Soit $G = (S, A)$ un graphe (non orienté), on souhaite calculer toutes les composantes connexes d'un graphe.

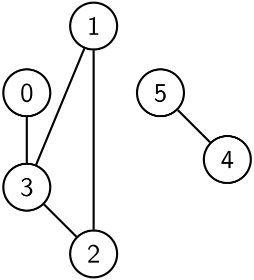
Solution facile avec l'utilisation d'une structure de données `union-find`.

Pseudo-code

```
Function cc(graph G) :
    uf_init (&dset, |S|);
    for Tous les sommets s de S do
        uf_add_element (&dset, s);
    end
    for Toutes les arêtes (u,v) de A do
        if not uf_are_connected(&dset, u, v) then
            uf_union (&dset, u, v);
        end
    end
```

Déroulement de l'algorithme

Graphe G



Structure de données `union-find`

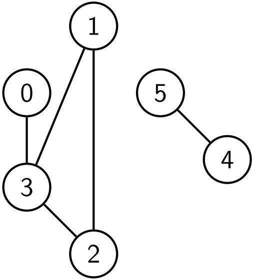
0	1	2	3	4	5
NULL	NULL	NULL	NULL	NULL	NULL

Principales étapes

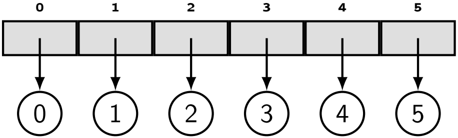
- Déclaration et initialisatione de la structure de données `union-find`

Déroulement de l'algorithme

Graphe G



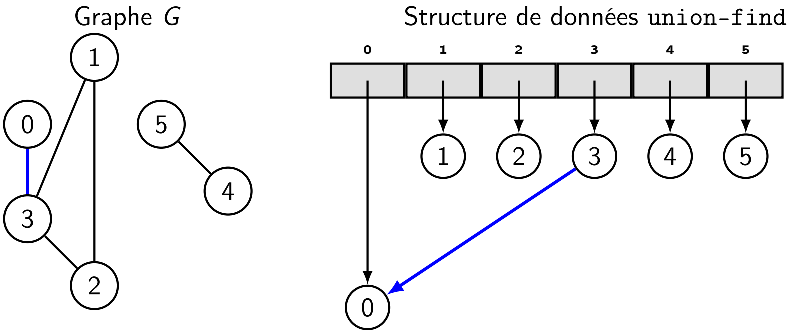
Structure de données `union-find`



Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)

Déroulement de l'algorithme

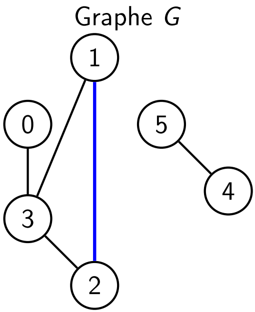


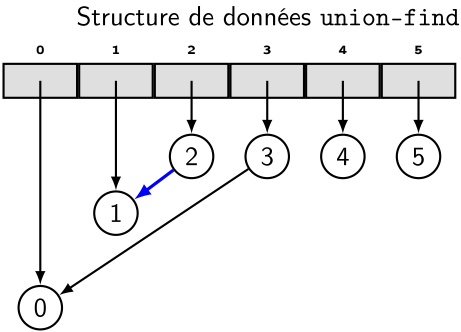
Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)
- Traitement des arcs de G et potentiels unions (**seconde boucle** `for`)

$$(0, 3), (1, 2), (1, 3), (2, 3), (5, 4)$$

Déroulement de l'algorithme





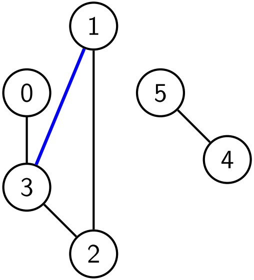
Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)
- Traitement des arcs de G et potentiels unions (**seconde boucle** `for`)

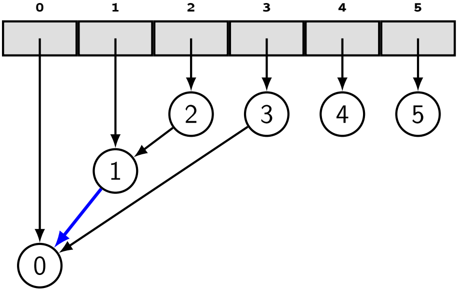
$$(0, 3),\ (1, 2),\ (1, 3),\ (2, 3),\ (5, 4)$$

Déroulement de l'algorithme

Graphe G



Structure de données `union-find`

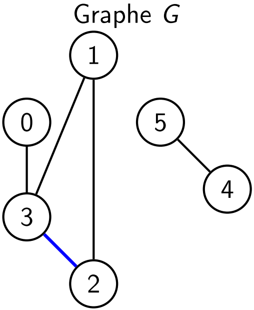


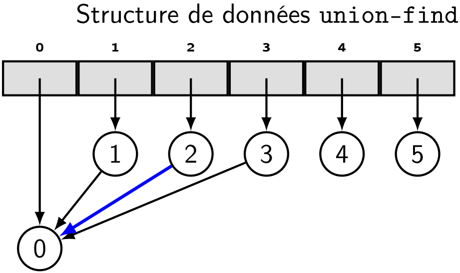
Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)
- Traitement des arcs de G et potentiels unions (**seconde boucle** `for`)

$$(0, 3),\ (1, 2),\ (1, 3),\ (2, 3),\ (5, 4)$$

Déroulement de l'algorithme





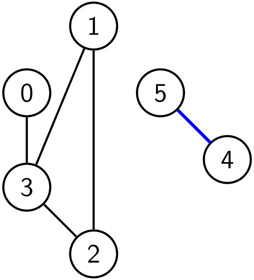
Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)
- Traitement des arcs de G et potentiels unions (**seconde boucle** `for`)

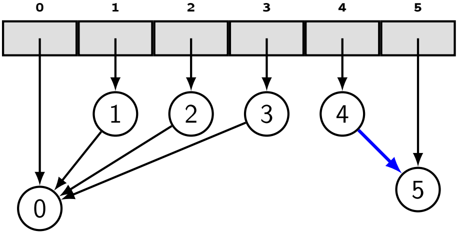
$$(0,3),\ (1,2),\ (1,3),\ (2,3),\ (5,4)$$

Déroulement de l'algorithme

Graphe G



Structure de données `union-find`



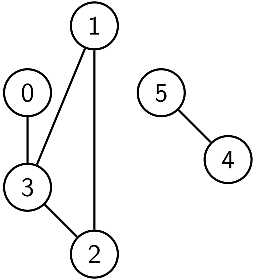
Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)
- Traitement des arcs de G et potentiels unions (**seconde boucle** `for`)

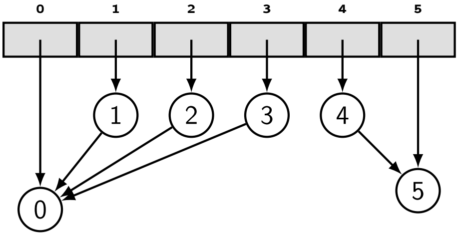
$$(0,3),\ (1,2),\ (1,3),\ (2,3),\ (5,4)$$

Déroulement de l'algorithme

Graphe G



Structure de données `union-find`



Principales étapes

- Déclaration et initialisatione de la structure de données `union-find`
- Ajouts de tous les sommets de G dans `union-find` (**première boucle** `for`)
- Traitement des arcs de G et potentiels unions (**seconde boucle** `for`)

$$(0,3),\ (1,2),\ (1,3),\ (2,3),\ (5,4)$$

- A la fin on a bien 2 sous-ensembles disjoints

Calcul des composantes fortement connexes d'un digraphe

Principe : Algorithme de Kosaraju-Sharir

Soit G un digraphe, l'algorithme opère en deux étapes :

- Effectuer un parcours en profondeur de G et enregistrer les dates de fin de visite
- Effectuer un parcours en profondeur sur le graphe transposé G^T de G, en suivant l'ordre décroissant des dates de fin données par la première étape.

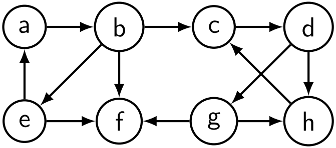
Les arbres produits par le deuxième parcours sont les composantes fortement connexes de G.

Complexité

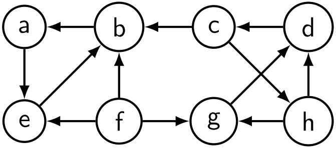
Pour un digraphe $G = (S, A)$, cet algorithme a une complexité linéaire en nombre de sommets et nombre d'arcs, c'est-à-dire, $\mathcal{O}(|S| + |A|)$

Déroulement de l'algorithme

Graphe G



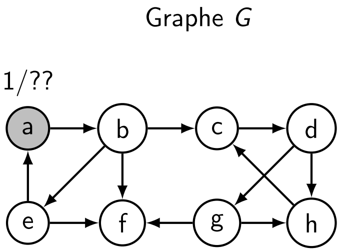
Graphe transposé G^T de G

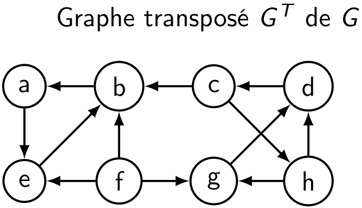


Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme



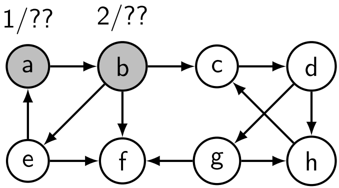


Déroulement de l'algorithme

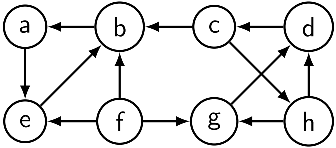
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



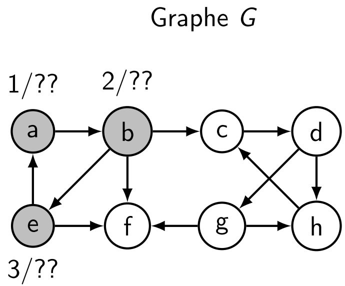
Graphe transposé G^T de G

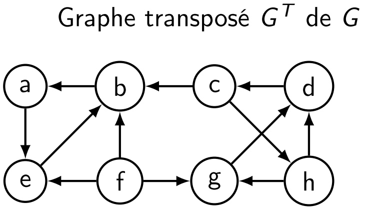


Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme



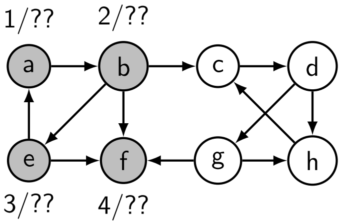


Déroulement de l'algorithme

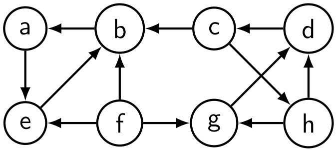
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G

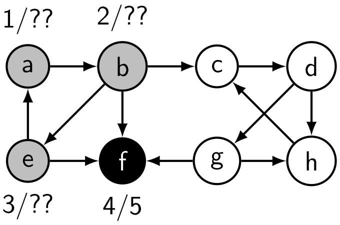


Déroulement de l'algorithme

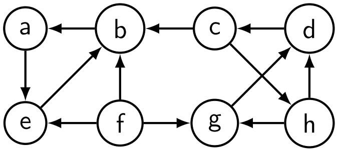
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



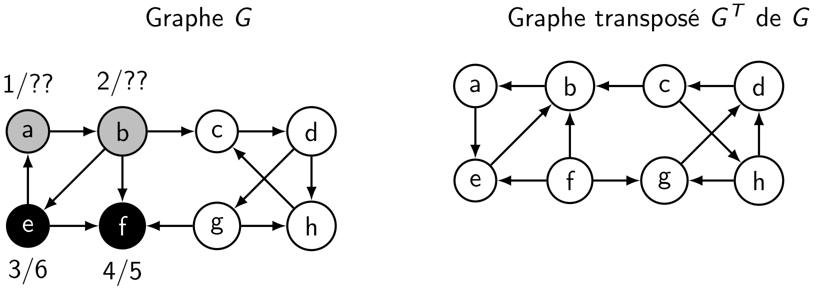
Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

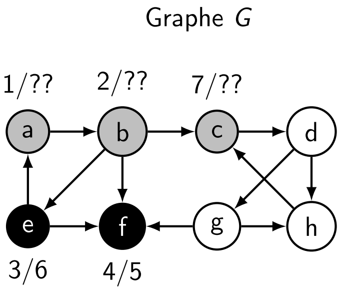


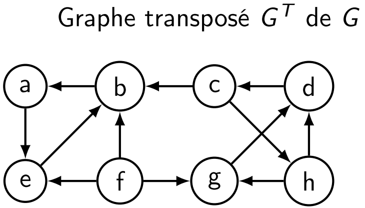
Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme



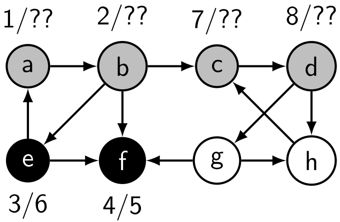


Déroulement de l'algorithme

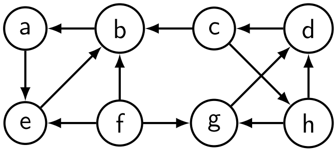
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G

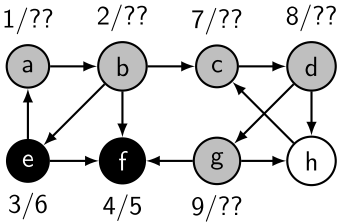


Déroulement de l'algorithme

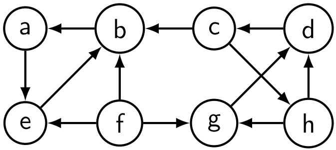
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



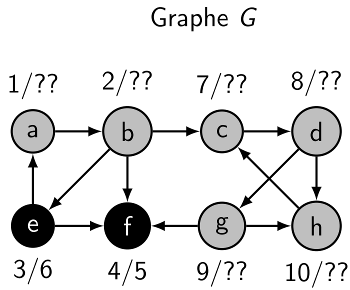
Graphe transposé G^T de G

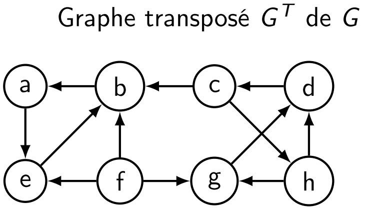


Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme



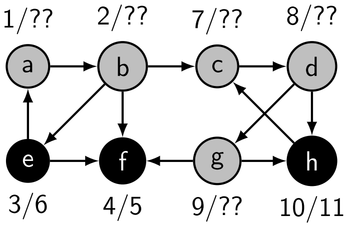


Déroulement de l'algorithme

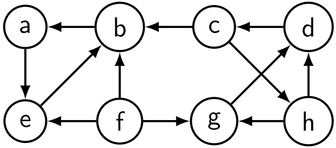
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



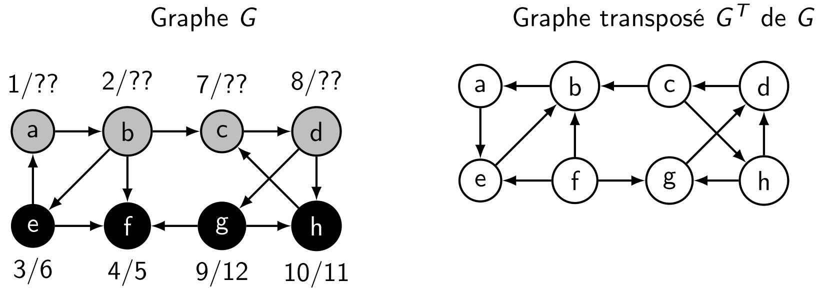
Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

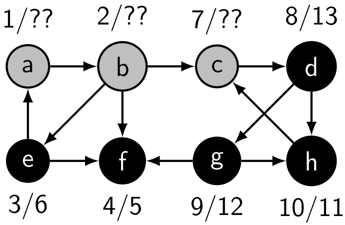


Déroulement de l'algorithme

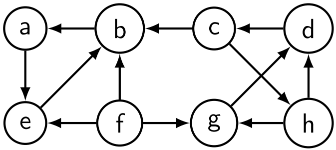
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G

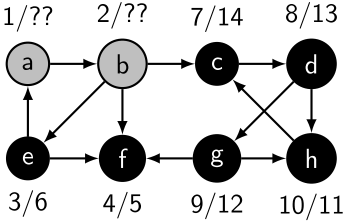


Déroulement de l'algorithme

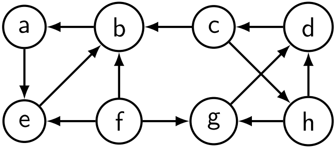
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



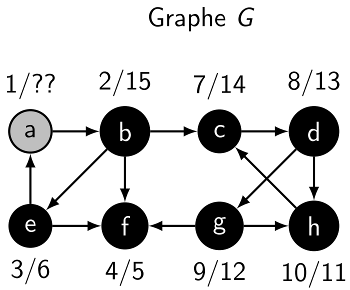
Graphe transposé G^T de G

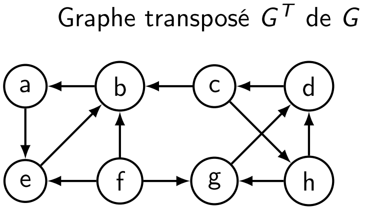


Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme



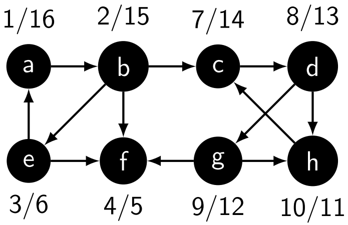


Déroulement de l'algorithme

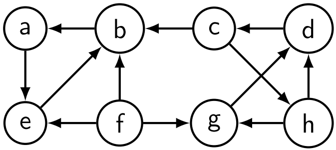
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G

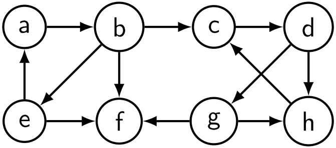


Déroulement de l'algorithme

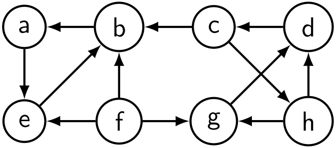
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

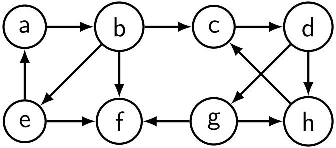
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

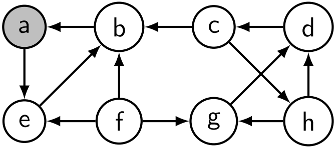
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



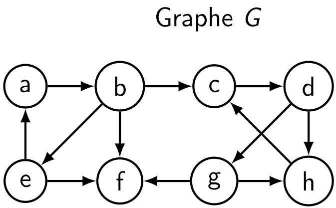
Déroulement de l'algorithme

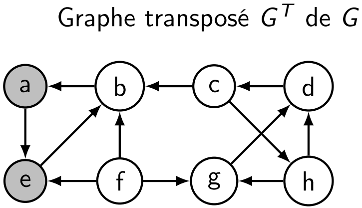
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme





Déroulement de l'algorithme

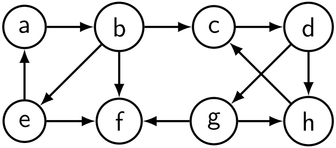
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

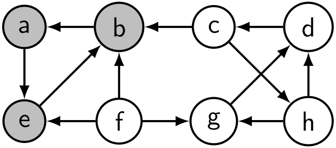
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

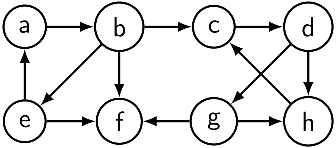
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

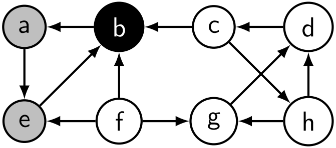
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



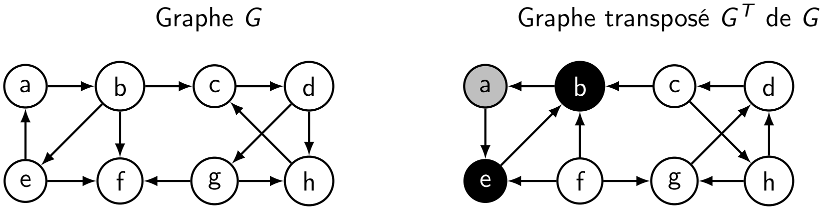
Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme



Déroulement de l'algorithme

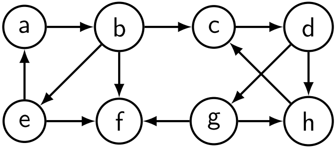
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

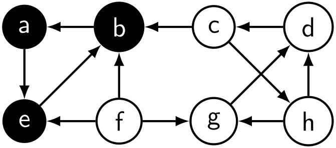
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

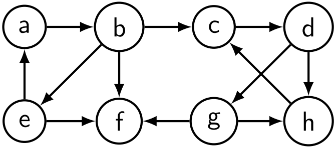
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, c, d, g, h, e, f

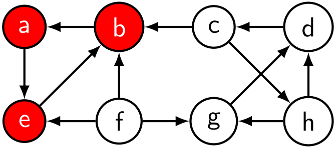
2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

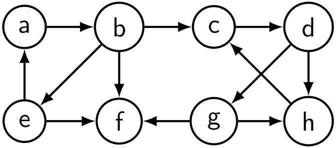
   a, b, c, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

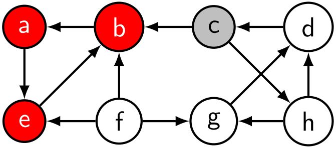
$$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

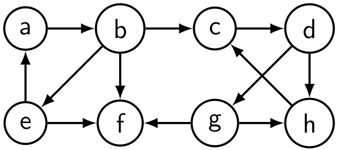
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

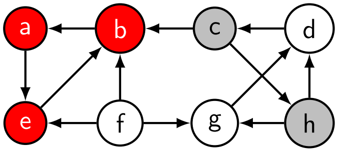
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

   $$\mathrm{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme





Déroulement de l'algorithme

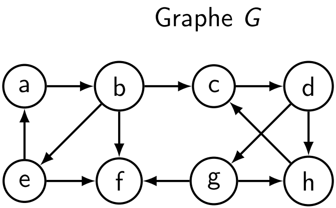
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

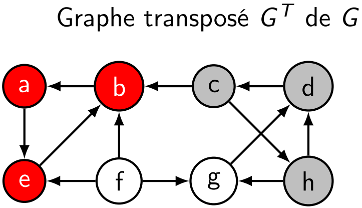
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

   $$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme





Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

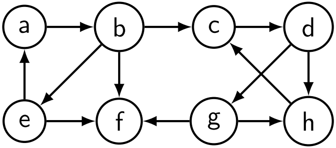
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

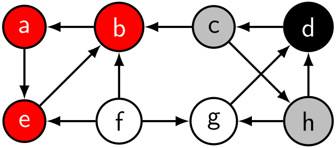
   $$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

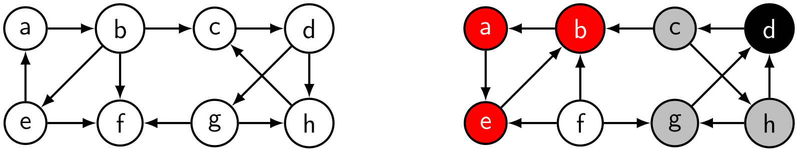
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

   $$\mathrm{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme

Graphe G — Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

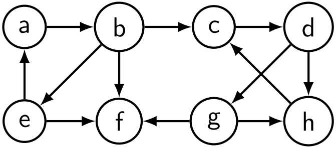
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

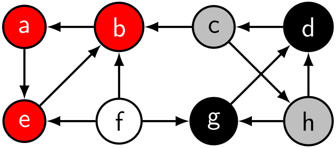
$$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

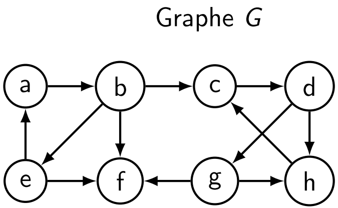
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

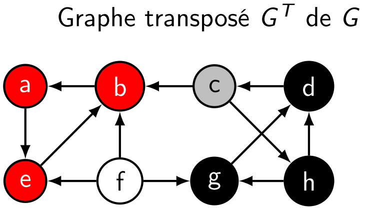
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

   $$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme





Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

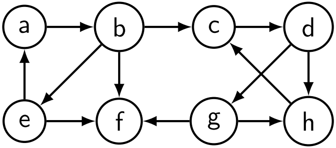
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

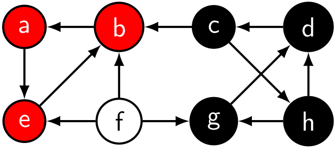
   $$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

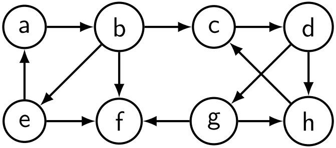
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

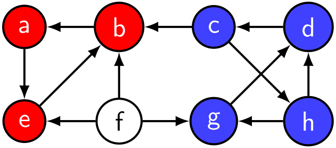
   $$\text{CFC}_1 = \{a, e, b\}$$

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

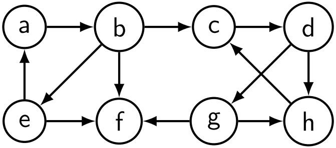
   a, b, **c**, d, g, h, e, f

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

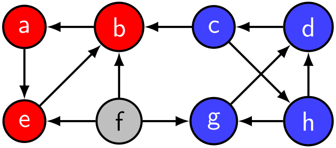
$$\mathrm{CFC}_1 = \{a, e, b\},\ \mathrm{CFC}_2 = \{c, h, d, g\}$$

Déroulement de l'algorithme

Graphe G



Graphe transposé G^T de G



Déroulement de l'algorithme

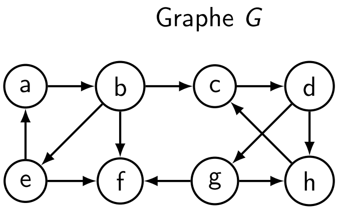
1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

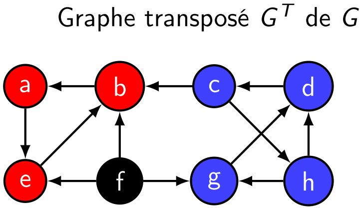
   a, b, **c**, d, g, h, e, **f**

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

$$\mathrm{CFC}_1 = \{a, e, b\},\ \mathrm{CFC}_2 = \{c, h, d, g\}$$

Déroulement de l'algorithme





Déroulement de l'algorithme

1. Parcours en profondeur de G et tri par ordre décroissant des dates de fin :

   a, b, **c**, d, g, h, e, **f**

2. Parcours en profondeur de G^T en suivant l'ordre des sommets donnés à l'étape 1

$$\mathrm{CFC}_1 = \{a, e, b\},\ \mathrm{CFC}_2 = \{c, h, d, g\}\ ,\ \mathrm{CFC}_3 = \{f\}$$

Quelques éléments de correction

Pour plus d'explications *cf* Cormen *et al.*, chapitre 22.5, les principaux résultats

- G et G^T ont les mêmes composantes fortement connexes.
- **Lemme 1** : Soient C et C' des CFC distinctes de $G = (S, A)$, soit $u, v \in C$, soit $u', v' \in C'$, et supposons qu'il y ait un chemin u vers u' dans G. Alors, il ne peut pas y avoir aussi un chemin v' vers v dans G.
- **Lemme 2** Soient C et C' des CFC distinctes de $G = (S, A)$. On suppose qu'il y a un arc $(u, v) \in A$, tel que $u \in C$ et $v \in C'$. Alors, $f(C) > f(C')$.
- **Corollaire** Soient C et C' des CFC distinctes de $G = (S, A)$. Supposons qu'il y ait un arc $(u, v) \in^T A$, tel que $u \in C$ et $v \in C'$. Alors, $f(C) < f(C')$.

Remarque

L'**algorithme de Tarjan** est un autre algorithme de calcul des CFC de complexité linaire mais qui réalise qu'un seul parcours en profondeur.

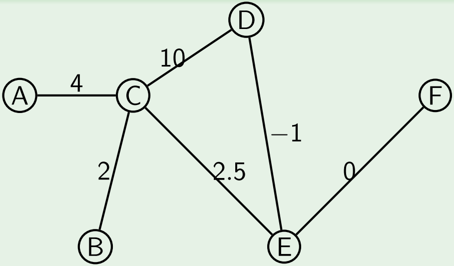
(di)graphes pondérés

Graphes pondérés

Un **graphe non orienté** (ou simplement **graphe**) **pondéré** est un triplet (S, A, w)

- S ensemble fini de **sommets** (*vertex/vertices*)
- A ensemble fini d'**arêtes** (*edges*) sous ensemble de $S \times S$
- w une fonction de $A \times A \mapsto \mathbb{R}$ (ou $\mathbb{Z}$ ou autre) qui associe à chaque arête un **poids**, une **capacité** ou une **valuation**.

Exemple



$$w(a, c) = 4$$
$$w(c, b) = 2$$
$$w(c, d) = 10$$
$$w(c, e) = 2.5$$
$$w(d, e) = -1$$
$$w(e, f) = 0$$

Mise en œuvre dans la bibliothèque `libin103`

Définition d'un graphe avec sommets identifiés avec des entiers

```
typedef struct integer_adjlist_elmt_ {
  int vertex;
  double weight;
} integer_adjlist_elmt_t;

typedef struct integer_adjlist_ {
  int vertex;
  generic_set_t adjacent;
} integer_adjlist_t;

typedef struct integer_graph_ {
  int vcount;
  int ecount;
  generic_list_t adjlists;
} integer_graph_t;
```

Point de vigilance !

Une liste d'adjacence est modélisée par

- une **liste chaînée** (générique) de structures
- dont un des champs est un **ensemble** (générique)
- dont les éléments sont des **structures**.

Remarque on considère des (di)graphes dont les arcs/arêtes peuvent avoir une valeur (`weight`).

API des graphes d'entiers

API insertion d'arête/arc

```
int integer_graph_ins_edge(integer_graph_t *g,
                           int v1, int v2, double w);
```

API bonus : liste des voisins d'un sommet (sans les poids !)

```
integer_list_t* integer_graph_adjlist (integer_graph_t* graph, int vertex);
```

Réécriture du parcours en largeur : boucle principale

```
while (integer_queue_size (\&queue) > 0) {
  int vertex; integer_queue_dequeue (\&queue, \&vertex);
  printf ("Vertex␣%d␣visited\n", vertex);

  integer_list_t* neighbors = integer_graph_adjlist (graph, vertex);

  integer_list_elmt_t* elem = integer_list_head (neighbors);
  for (; elem != NULL; elem = integer_list_next (elem)) {
    int n = integer_list_data (elem);
    if (!integer_set_is_member (\&set, n)) {
      integer_set_insert (\&set, n);
      integer_queue_enqueue (\&queue, n);
    }
  }
  free(neighbors);
}
```

API des parcours les graphes d'entiers

API parcours en profondeur

```
int integer_dfs(integer_graph_t *graph, int start,
                integer_list_t *ordered);

int integer_dfs_all(integer_graph_t *graph,
                    integer_list_t *ordered);
```

API parcours en largeur

```
int integer_bfs(integer_graph_t *graph, int start,
                integer_list_t *ordered);

int integer_bfs_all(integer_graph_t *graph,
                    integer_list_t *ordered);
```

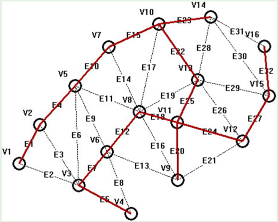
API tri topologique

```
int integer_ts (integer_graph_t* graph, integer_list_t* ordered);
```

Arbre couvrant de poids minimum

Introduction au problème

Exemple de problèmes



Différents algorithmes connus

- **Algorithme de Kruskal** (considéré dans ce cours) qui utilise
  - une structure de données `tas min`
  - une structure de données `union-find`

Complexité temporelle

Classe de complexité linéaire c'est-à-dire $\mathcal{O}(|A| \log(|S|))$ [1]

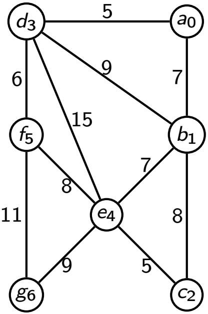
1. *cf* RO201 en 2A Math ou RO202 en 2A Info pour plus de détails

Algorithme de Kruskal

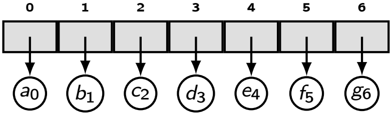
Pseudo-code

```
Function kruskal(graph G=(S,A)) :
    uf_init (&dset, |S|);
    heap_init (&heap_min);
    for /* Tous les sommets s de S */ do
        uf_make_set (&dset, s);
    end
    /* Trier les aretes dans l'ordre croissant des poids */
    for /* Toutes les aretes a de A */ do
        heap_insert (&heap_min, a);
    end
    while heap_size (&heap_min) > 0 do
        (u,v) = heap_extract (&heap_min);
        if not uf_are_connected(&dset, u, v) then
            list_ins_next (&list, NULL, (u,v));
            uf_union (&dset, u, v);
        end
    end
```

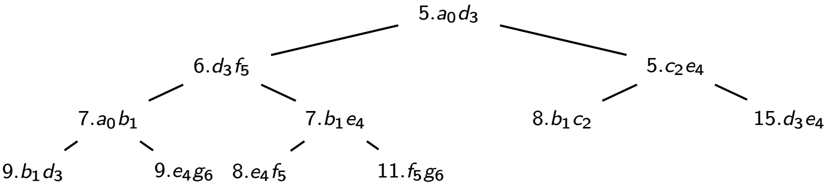
Déroulement de l'algorithme - après les deux boucles **for**



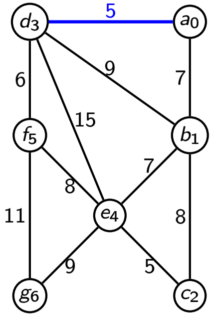
union-find des sommets



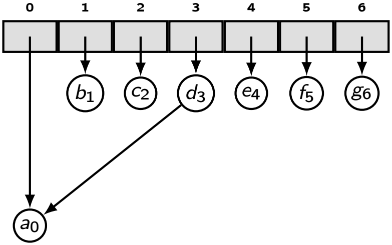
Tas min des arêtes



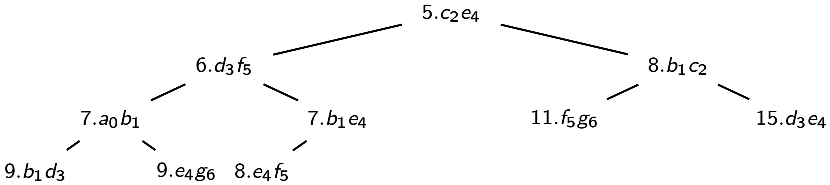
Déroulement de l'algorithme - boucle **while**



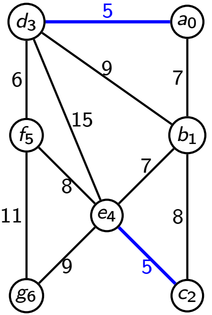
`union-find` **des sommets**



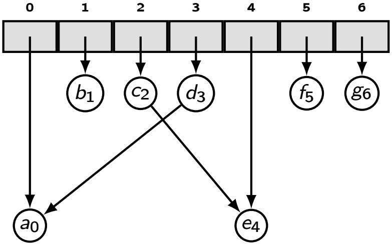
Tas min des arêtes (extraction de $5.a_0 d_3$)



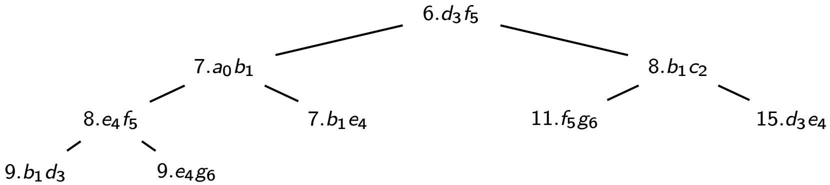
Déroulement de l'algorithme - boucle **while**



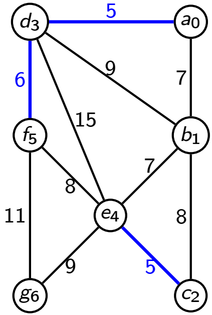
`union-find` **des sommets**



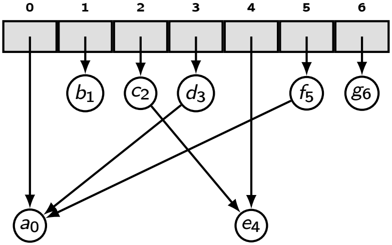
Tas min des arêtes (extraction de $5.c_2e_4$)



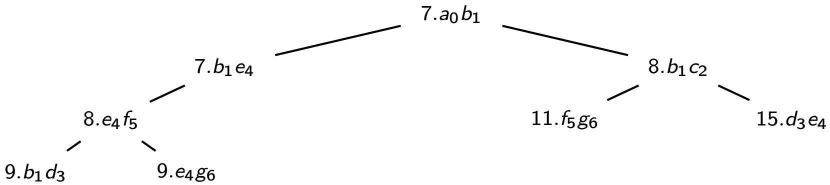
Déroulement de l'algorithme - boucle **while**



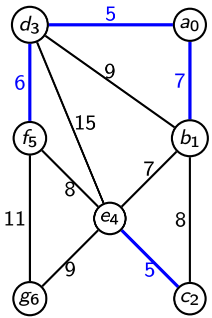
`union-find` **des sommets**



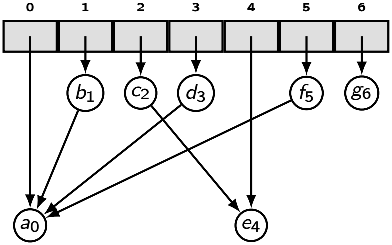
Tas min des arêtes (extraction de $6.d_3 f_5$)



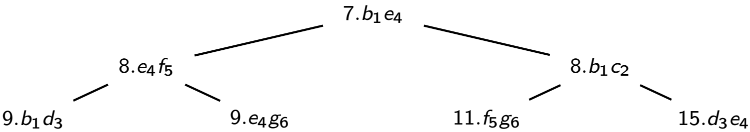
Déroulement de l'algorithme - boucle **while**



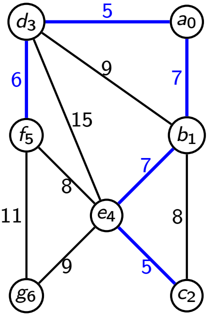
`union-find` **des sommets**



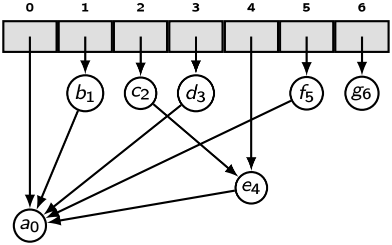
Tas min des arêtes (extraction de $7.a_0 b_1$)



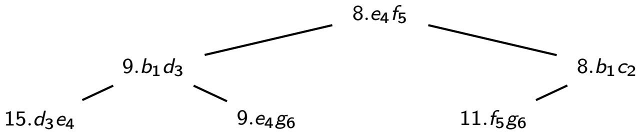
Déroulement de l'algorithme - boucle **while**



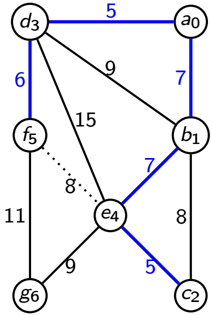
union-find des sommets



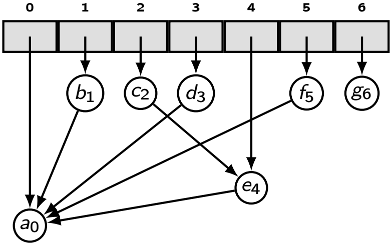
Tas min des arêtes (extraction de $7.b_1 e_4$)



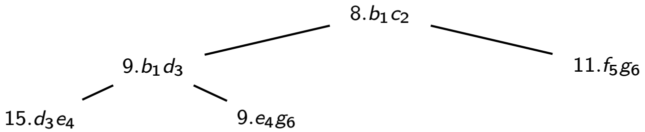
Déroulement de l'algorithme - boucle **while**



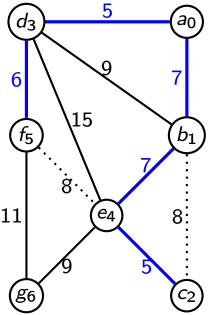
`union-find` **des sommets**



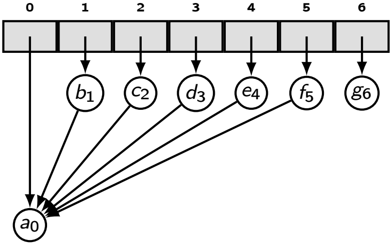
Tas min des arêtes (extraction de $8.e_4 f_5$)



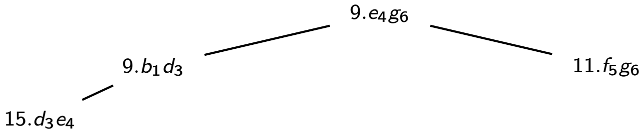
Déroulement de l'algorithme - boucle **while**



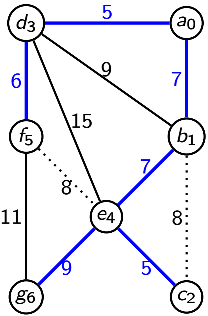
`union-find` **des sommets**



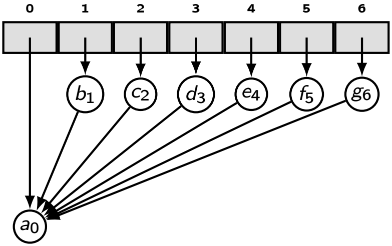
Tas min des arêtes (extraction de $8.b_1c_2$)



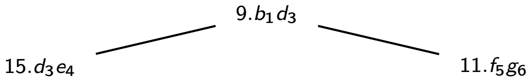
Déroulement de l'algorithme - boucle **while**



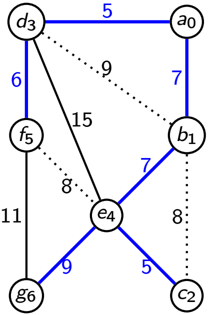
`union-find` **des sommets**



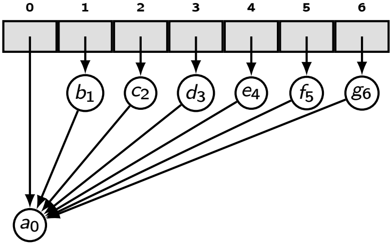
Tas min des arêtes (extraction de $9.e_4g_6$)



Déroulement de l'algorithme - boucle **while**



`union-find` **des sommets**

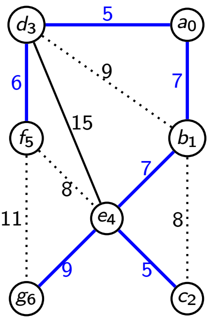


Tas min des arêtes (extraction de $9.b_1d_3$)

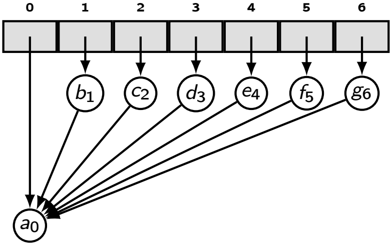
$11.f_5g_6$

$15.d_3e_4$

Déroulement de l'algorithme - boucle **while**



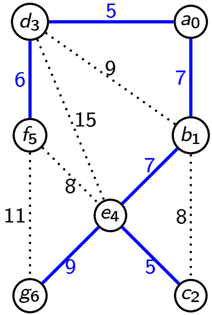
`union-find` **des sommets**



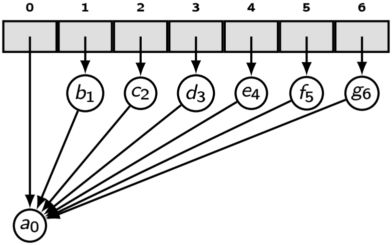
Tas min des arêtes (extraction de $11.f_5g_6$)

$15.d_3e_4$

Déroulement de l'algorithme - boucle **while**



`union-find` **des sommets**



Tas min des arêtes (extraction de $15.d_3e_4$ et fin)

Résultat final

- L'arbre couvrant est donné par la liste d'arêtes

$$(a_0 d_3), (c_2 e_4), (d_3 f_5), (a_0 b_1), (b_1 e_4), (e_4 g_6)$$

- C'est un **arbre non enraciné**, c'est-à-dire que la racine n'est pas définie

API des algorithmes sur les graphes d'entiers

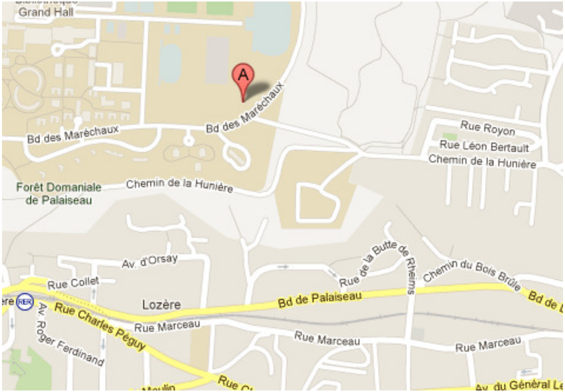
API algorithmes de Kruskal

```
typedef struct we_ {
  int source;
  int destination;
  double weight;
} we_t;

int integer_mst(integer_graph_t *graph, generic_list_t **span);
```

Plus court chemin

Motivation : assistance au déplacement



Objectif : trouver un chemin entre un point A et un point B

- qui est le plus court en distance
- ou qui le plus court en temps
- ou qui est le plus court en ...

Différentes classes de problèmes

- Plus court chemin à origine unique et poids positifs ou nuls :

  algorithme de Dijkstra

  considéré dans ce cours
- Plus court chemin à origine unique et poids quelconques :

  algorithme de Bellman-Ford [2]
- Plus court chemin entre toutes les paires de sommets et poids quelconques (modulo détails) :

  algorithme de Floyd-Warshall [2]

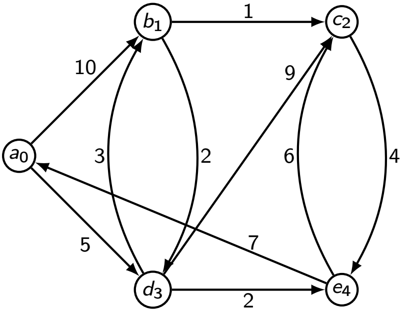
2. *cf* RO201 en 2A Math ou RO202 en 2A Info pour plus de détails

Algorithme de Dijkstra

Pseudo-code

```
Function shortest(graph G, start) :
    for /* Tous les sommets s de S */ do
        d[s]= ∞;
        π[s] = -1;
    end
    d[start] = 0;
    for /* Toutes les sommets s de S */ do
        heap_insert (&heap_min, (s, d[s],π[s]));
    end
    while heap_size (&heap_min) > 0 do
        u = heap_extract (&heap_min);
        list_ins_next (&list, list_tail(&list), u);
        for Tous les voisins v de u do
            if d[v] > d[u] + poids(u,v) then
                d[v] > d[u] + poids(u,v);
                π[v] = u;
            end
        end
    end
```

Déroulement de l'algorithme



- Tableau des distances d

0	1	2	3	4
0	∞	∞	∞	∞

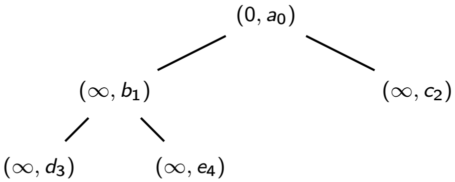
- Tableau des parents π

0	1	2	3	4
-1	-1	-1	-1	-1

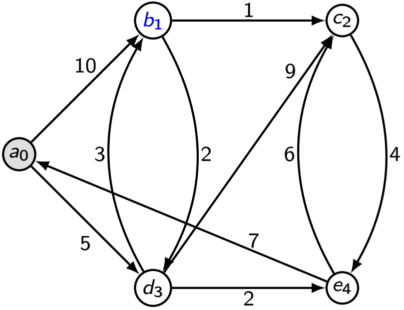
Étapes de l'algorithme

- Après les deux boucles **for**

- Tas-min des (distances, sommets)



Déroulement de l'algorithme



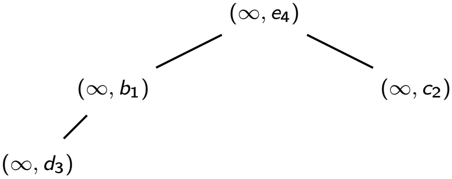
- Tableau des distances d

0	1	2	3	4
0	∞	∞	∞	∞

- Tableau des parents π

0	1	2	3	4
-1	-1	-1	-1	-1

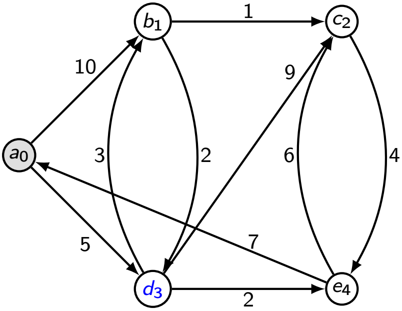
- Tas-min extraction de $(0, a_0)$



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, d_3
  - Mise à jour des distances

Déroulement de l'algorithme



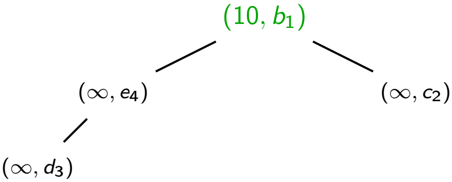
- Tableau des distances d

0	1	2	3	4
0	10	∞	∞	∞

- Tableau des parents π

0	1	2	3	4
-1	0	-1	-1	-1

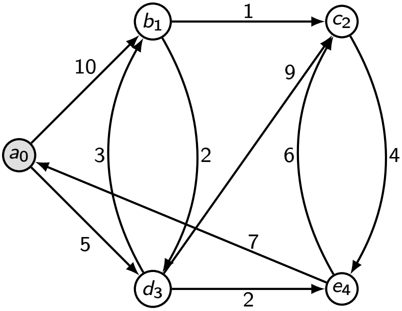
- Tas-min modification distance b_1



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, d_3
  - Mise à jour des distances
    - $d[1] > d[0] + w(0,1)$?

Déroulement de l'algorithme



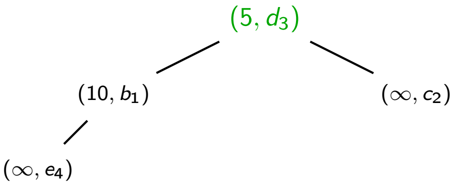
- Tableau des distances d

0	1	2	3	4
0	10	∞	5	∞

- Tableau des parents π

0	1	2	3	4
-1	0	-1	0	-1

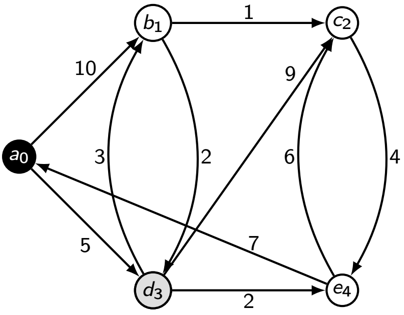
- Tas-min modification distance d_3



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, d_3
  - Mise à jour des distances
    - $d[1] > d[0] + w(0, 1)$?
    - $d[3] > d[0] + w(0, 3)$?

Déroulement de l'algorithme



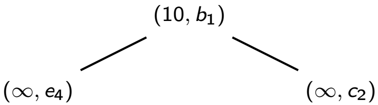
- Tableau des distances d

0	1	2	3	4
0	10	∞	5	∞

- Tableau des parents π

0	1	2	3	4
-1	0	-1	0	-1

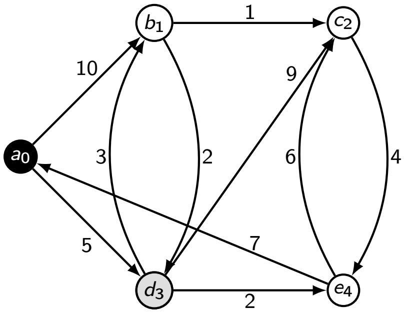
- Tas-min extraction de $(5, d_3)$



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, c_2, e_4
  - Mise à jour des distances

Déroulement de l'algorithme



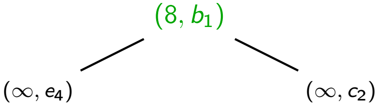
- Tableau des distances d

0	1	2	3	4
0	8	∞	5	∞

- Tableau des parents π

0	1	2	3	4
-1	3	-1	0	-1

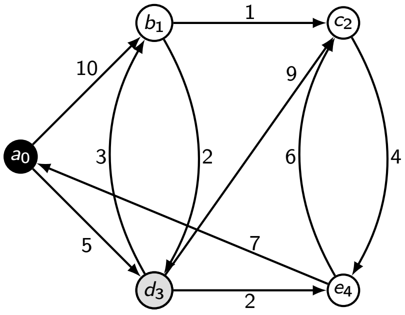
- Tas-min modification distance b_1



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, c_2, e_4
  - Mise à jour des distances
    - $d[1] > d[3] + w(3, 1)$?

Déroulement de l'algorithme



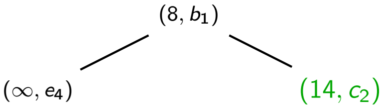
- Tableau des distances d

0	1	2	3	4
0	8	14	5	∞

- Tableau des parents π

0	1	2	3	4
-1	3	3	0	-1

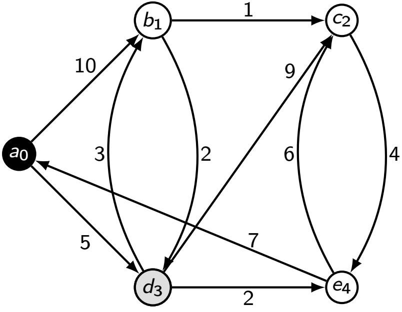
- Tas-min modification distance c_2



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, c_2, e_4
  - Mise à jour des distances
    - $d[1] > d[3] + w(3,1)$?
    - $d[2] > d[3] + w(3,2)$?

Déroulement de l'algorithme



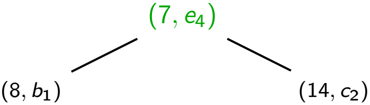
- Tableau des distances d

0	1	2	3	4
0	8	14	5	7

- Tableau des parents π

0	1	2	3	4
-1	3	3	0	3

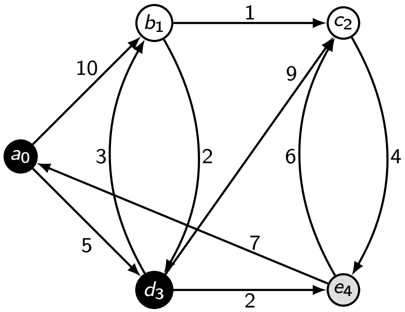
- Tas-min modification distance e_4



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : b_1, c_2, e_4
  - Mise à jour des distances
    - $d[1] > d[3] + w(3,1)$?
    - $d[2] > d[3] + w(3,2)$?
    - $d[4] > d[3] + w(3,4)$?

Déroulement de l'algorithme



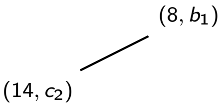
- Tableau des distances d

0	1	2	3	4
0	8	14	5	7

- Tableau des parents π

0	1	2	3	4
-1	3	3	0	3

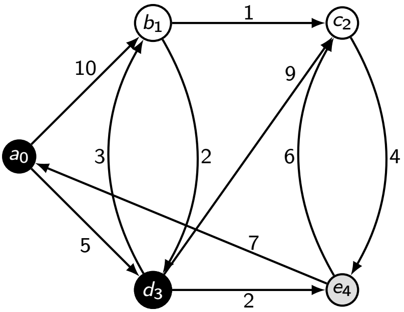
- Tas-min extraction de $(7, e_4)$



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : a_0, c_2
  - Mise à jour des distances

Déroulement de l'algorithme



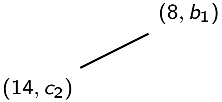
- Tableau des distances d

0	1	2	3	4
0	8	14	5	7

- Tableau des parents π

0	1	2	3	4
-1	3	3	0	3

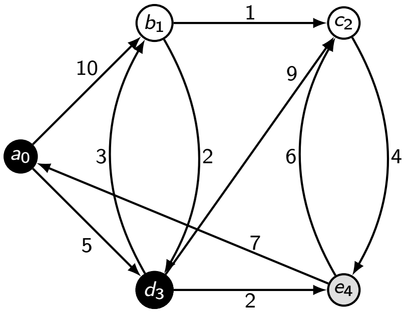
- Tas-min pas de changement distance a_0



Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : a_0, c_2
  - Mise à jour des distances
    - $d[0] > d[4] + w(4, 0)$?

Déroulement de l'algorithme



- Tableau des distances d

0	1	2	3	4
0	8	13	5	7

- Tableau des parents π

0	1	2	3	4
-1	3	4	0	3

- Tas-min modification distance c_2

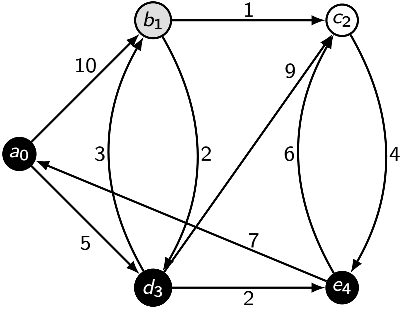
$(8, b_1)$

$(13, c_2)$

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : a_0, c_2
  - Mise à jour des distances
    - $d[0] > d[4] + w(4, 0)$?
    - $d[2] > d[4] + w(4, 2)$?

Déroulement de l'algorithme



- Tas-min extraction de $(8, b_1)$

$(13, c_2)$

- Tableau des distances d

0	1	2	3	4
0	8	13	5	7

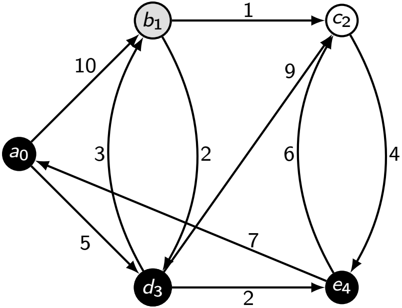
- Tableau des parents π

0	1	2	3	4
-1	3	4	0	3

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : c_2, d_3
  - Mise à jour des distances

Déroulement de l'algorithme



- Tas-min modification distance c_2

$(9, c_2)$

- Tableau des distances d

0	1	2	3	4
0	8	9	5	7

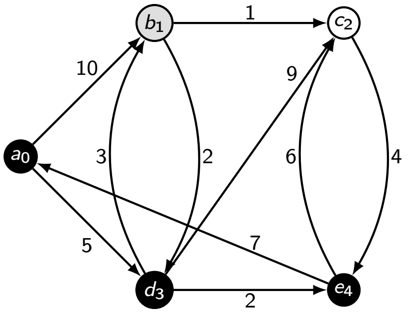
- Tableau des parents π

0	1	2	3	4
-1	3	1	0	3

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : c_2, d_3
  - Mise à jour des distances
    - $d[2] > d[1] + w(1,2)$?

Déroulement de l'algorithme



- Tableau des distances d

0	1	2	3	4
0	8	9	5	7

- Tableau des parents π

0	1	2	3	4
-1	3	1	0	3

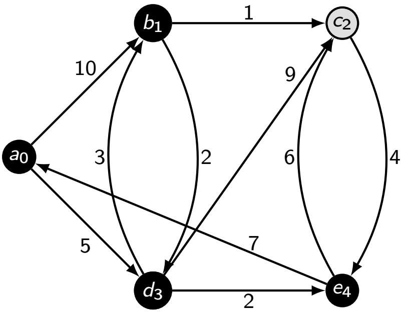
- Tas-min pas de changement distance d_3

$(9, c_2)$

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisins : c_2, d_3
  - Mise à jour des distances
    - $d[2] > d[1] + w(1,2)$?
    - $d[3] > d[1] + w(1,3)$?

Déroulement de l'algorithme



- Tas-min extraction de $(9, c_2)$

- Tableau des distances d

0	1	2	3	4
0	8	9	5	7

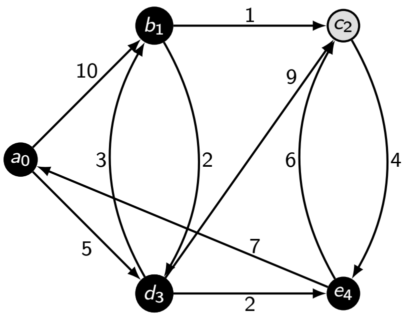
- Tableau des parents π

0	1	2	3	4
-1	3	1	0	3

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisin : e_4
  - Mise à jour des distances

Déroulement de l'algorithme



- Tableau des distances d

0	1	2	3	4
0	8	9	5	7

- Tableau des parents π

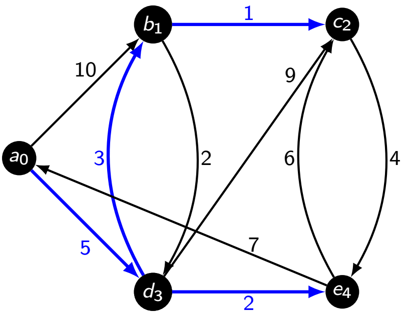
0	1	2	3	4
-1	3	1	0	3

- Tas-min pas de changement distance e_4

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
  - Examen voisin : e_4
  - Mise à jour des distances
    - $d[4] > d[2] + w(2,4)$?

Déroulement de l'algorithme



- Tableau des distances d

0	1	2	3	4
0	8	9	5	7

- Tableau des parents π

0	1	2	3	4
-1	3	1	0	3

Étapes de l'algorithme

- Après les deux boucles **for**
- Parcours du graphe (**while**)
- Fin

Au final, on a calculé tous les chemins depuis a_0 vers les autres sommets de coût minimal, représentés en bleu (pour simplicité), mais donnés par le tableau π.

API des algorithmes sur les graphes d'entiers

API algorithmes Dijkstra

```
typedef struct ed_ {
  int vertex;
  double distance;
  int parent;
} ed_t;

int integer_shortest(integer_graph_t *graph, int start,
                     generic_list_t **paths);
```

MST vs Plus court chemin

	MST	Plus court chemin
Définition	Un arbre qui traverse tous les sommets, tout en minimisant le poids total.	Le chemin dont la distance accumulées est la plus faible.
Objectif	Connexion de tous les nœuds entre eux avec un poids total minimal.	Trouver l'itinéraire le plus efficace entre deux sommets.
Applications	La conception de réseaux pour la pose de câbles (minimise le coût)	La connexion entre les deux points peut être choisie très efficacement par le chemin le plus court et cela aide également les gens à naviguer vers l'endroit final avec une distance minimale.

Conclusion

En IN103, nous avons vu beaucoup beaucoup de choses

- des structures de données :
  - listes, piles, files, ensembles, arbres, tas, union-find, graphe
- des algorithmes pour résoudre des problèmes classiques :
  - les itérateurs, les parcours d'arbres ou de graphes, couverture d'ensemble, plus court chemin, arbre couvrant de poids minimal, les composantes (fortement) connexes, coloration de graphe

Remarque ;-)

Tout ceci n'est (encore) qu'un petit aperçu des possibilités infinies de la science informatique.