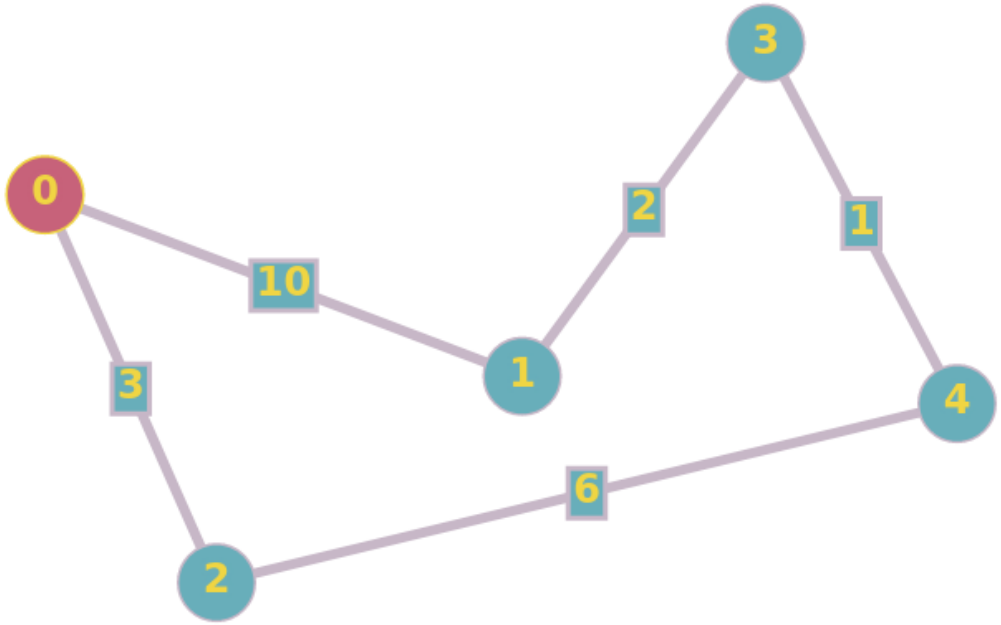
Algorithmique et optimisation

Loïc Demange
loic.demange@etud.univ-paris8.fr

26 novembre 2020

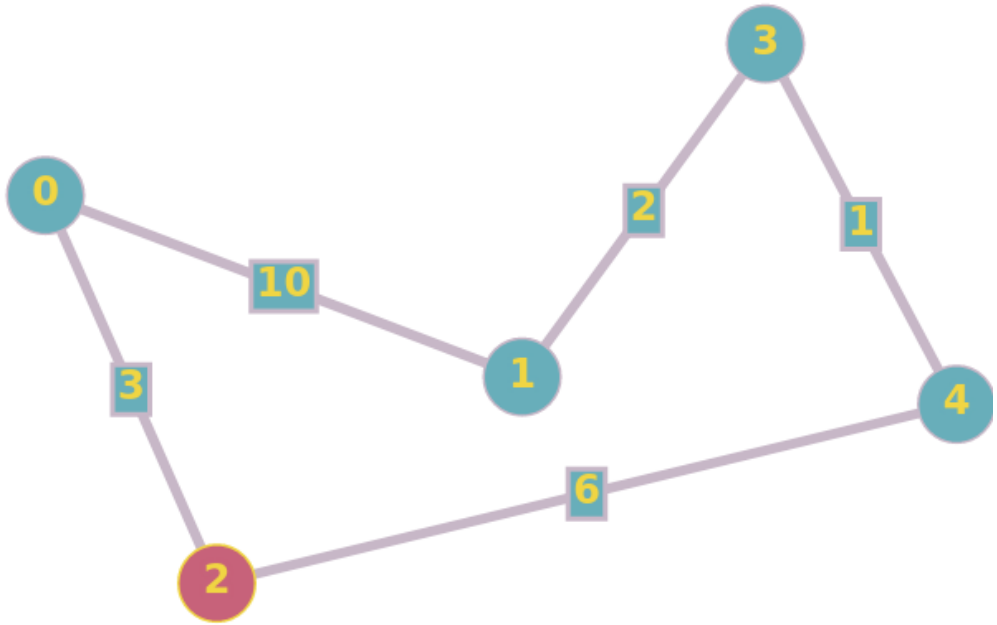
UNIVERSITÉ PARIS 8 VINCENNES-SAINT-DENIS

Algorithme de Dijkstra



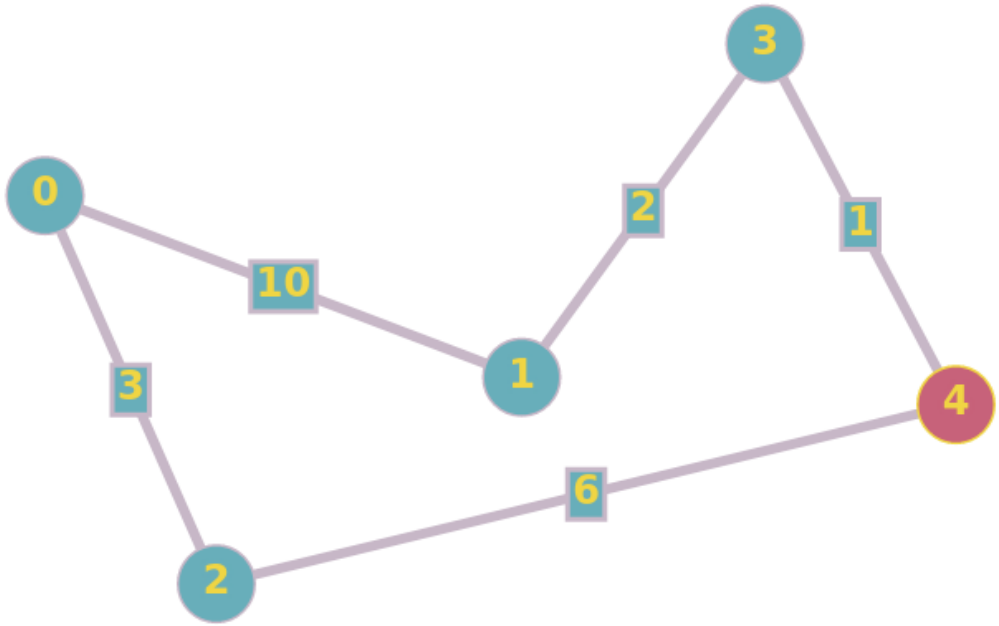
	0	1	2	3	4
0 (0)		10	$\underline{3}$	∞	∞

Algorithme de Dijkstra



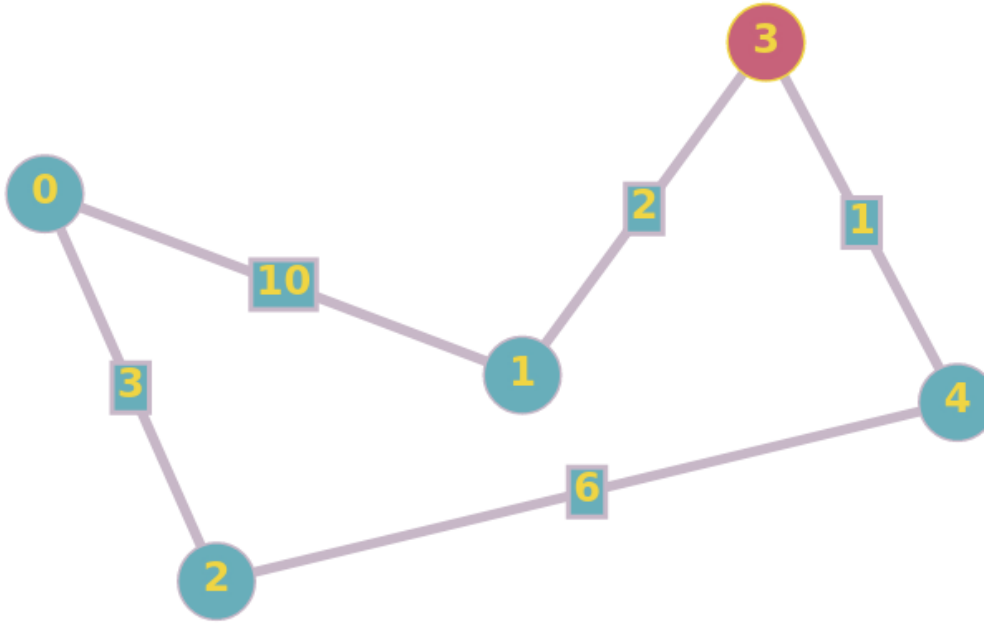
	0	1	2	3	4
0 (0)		10	$\underline{3}$	∞	∞
2 (3)		10	-	∞	$\underline{9}$

Algorithme de Dijkstra



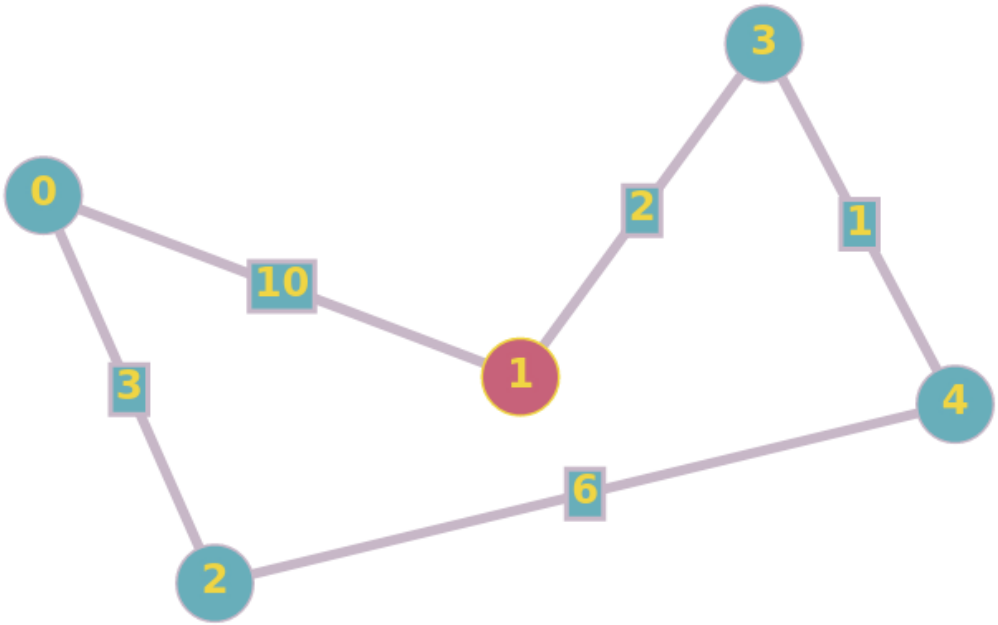
	0	1	2	3	4
0 (0)		10	$\underline{3}$	∞	∞
2 (3)		10	-	∞	$\underline{9}$
4 (9)		10	-	$\underline{10}$	-

Algorithme de Dijkstra



	0	1	2	3	4
0 (0)		10	$\underline{3}$	∞	∞
2 (3)		10	-	∞	$\underline{9}$
4 (9)		10	-	$\underline{10}$	-
3 (10)		$\underline{10}$	-	-	-

Algorithme de Dijkstra



	0	1	2	3	4
0 (0)		10	$\underline{3}$	∞	∞
2 (3)		10	-	∞	$\underline{9}$
4 (9)		10	-	$\underline{10}$	-
3 (10)		$\underline{10}$	-	-	-
1 (10)		-	-	-	-

Algorithme de Dijkstra

L'algorithme de Dijkstra se réalise sur un graphe pondéré, et permet de trouver le plus court chemin entre deux sommets.

Pour le réaliser, on effectue les tâches suivantes :

- On prend un sommet et on regarde les sommets adjaçents
- On note toutes les distances, et si une distance est inférieure à une distance déjà notée, on remplace
- On prend le sommet non parcouru avec la distance la plus proche, et on recommence, jusqu'à tant qu'il n'y ait plus de sommets non parcourus

Algorithme de Dijkstra

Algorithmiquement, on doit donc avoir accès à plusieurs choses.

- Le graphe, avec ses sommets et ses arêtes
- Le poids de chaque arête
- Les sommets adjaçents de chaque sommet

Algorithme de Dijkstra

Exercice :

- Écrire l'algorithme de Dijkstra

Indications Il faudra stocker les distances entre les sommets non adjaçents et ne pas recomptabiliser les sommets déjà visités. Enregistrer le prédécesseur de chaque sommet permet de refaire le parcours le plus court entre deux sommets.

Notons aussi la fonction **poids(a, b)** qui renvoie le poids de l'arête entre les sommets a et b.

Algorithme de Dijkstra

```
dijkstra(sommets, aretes, sommet_actuel)
{
    predecesseurs = {}
    deja_visites = {}
    Pour tout sommet dans sommets
        distance[sommet] = inf

    distance[sommet_actuel] = 0

    Tant que deja_visites != sommets
    {

    }
}
```

Algorithme de Dijkstra

```
dijkstra(sommets, aretes, sommet_actuel)
{
    predecesseurs = {}
    deja_visites = {}
    Pour tout sommet dans sommets
        distance[sommet] = inf

    distance[sommet_actuel] = 0

    Tant que deja_visites != sommets
    {
        Mettre sommet_actuel dans deja_visites
        Pour tout sommet_voisin de sommet_actuel et pas dans deja_visites
        {
            distance_actuelle = distance[sommet_actuel] +
                                poids(sommet_actuel, sommet_voisin)
            Si distance[sommet_voisin] > distance_actuelle
            {
                distance[sommet_voisin] = distance_actuelle
                predecesseurs[sommet_voisin] = sommet_actuel
            }
        }

    }
}
```

Algorithme de Dijkstra

```
dijkstra(sommets, aretes, sommet_actuel)
{
    predecesseurs = {}
    deja_visites = {}
    Pour tout sommet dans sommets
        distance[sommet] = inf

    distance[sommet_actuel] = 0

    Tant que deja_visites != sommets
    {
        Mettre sommet_actuel dans deja_visites
        Pour tout sommet_voisin de sommet_actuel et pas dans deja_visites
        {
            distance_actuelle = distance[sommet_actuel] + poids(sommet_actuel, sommet_voisin)
            Si distance[sommet_voisin] > distance_actuelle
            {
                distance[sommet_voisin] = distance_actuelle
                predecesseurs[sommet_voisin] = sommet_actuel
            }
        }

        distance_mini = inf
        Pour tout sommet dans sommets et pas dans deja_visites
        {
            Si distance[sommet] < distance_mini
            {
                distance_mini = distance[sommet]
                sommet_actuel = sommet
            }
        }
    }
}
```