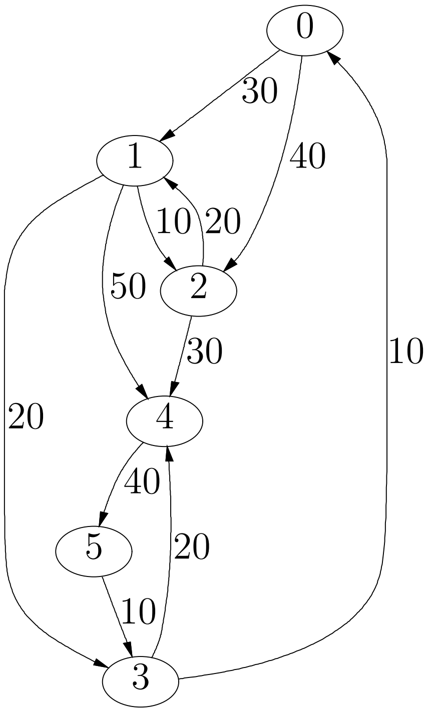
Algorithmique - Maitrise d'informatique - TD4
Dijkstra

1 Échauffement



▸ **Exercice 1.** Appliquer l'algorithme de Dijkstra au graphe de la figure précédente. Préciser pour chaque étape les valeurs attribuées à chaque sommet (poids $p[u]$ et sommet précédent $\pi[u]$).

▸ **Exercice 2.** Trouver pour ce graphe plusieurs arbres de plus court chemins.

2 Analyse

▸ **Exercice 3.** Montrer que si dans un graphe donné toutes les arêtes ont le même poid, alors l'algorithme de Dijkstra peut-être identique au parcours en largeur.

▸ **Exercice 4.** Pourquoi l'algorithme de Dijkstra n'est-il pas valable lorsque les arêtes peuvent avoir des poids négatifs ? Trouver un exemple.

▸ **Exercice 5.** Soit $G = (V, E)$ un graphe pondéré tel que $\forall (u, v) \in E, 0 < M(u, v) \leq 1$. On pourra considérer les poids d'un tel graphe comme des probabilités.

1. Proposer un algorithme permettant de trouver, étant donné un sommet source $s \in V$, des chemins de probabilité maximale de s à tout les autres sommets.

2. Montrer que votre algorithme est correct.

▸ **Exercice 6.** Soit $G = (V, E)$ un graphe pondéré et $s \in V$. Nous définissons la *capacité* c d'un chemin $p = v_1, \ldots v_n$ comme:

$$c(p) = min_{i \in 1 \ldots (n-1)} M(v_i, v_{i+1})$$

1. Adapter l'algorithme de Dijkstra à la recherche des chemins de capacité maximale.

2. Prouver votre algorithme.

Correction 1: Je rappelle Dijkstra:

```
Variables : D[1..n]
{initialisation}
C <- {2..n}
pour i de 2 a n faire D[i] <- M[1,i] (infini si existe pas)
                      (pere[i] <- 1 si M[1,i] non infini)

repete tant que C non vide
 v <- extrait dans C le min pour D
 pour tout w voisin de v faire
  D[w] <- min(D[w],D[v]+M[v,w])
  (et si le min a change pere[w] <- v)
fin repete
```

Correction 2:

Correction 3: D'abord essayer sur l'exemple.

POur comparer, onpeut dire que l'algorithme en largeur sort les sommets dans l'ordre croissant des profondeurs. La preuve de correction de Dijkstra dit qu'au moment de l'extraction, $D[v] = Distance(0, v) = Cte * prof$.

La liberte qui est laissée au moment où l'on extrait permet de conclure.

En remarque, on peut dire que le parcours en largeur est de complexite $O(n)$, et Dijkstra avec les poids est en $O((V + E).lnV)$, si la liste est un tas.

Correction 4: Le choix du suivant n'est plus optimal. J'ai donné en TD 2 reponses: la premiere c'est que si il existe un cycle de poids négatif, alors la définition de plus cours chemin n'existe pas, de fait Dijkstra pne peut donner un résultat correct (il n'y en a pas).

On peut compliquer la question: et si on autorise les pods négatifs, pourvu qu'il n'y ait pas de cycle de poids négatifs? Dans ce cas on peut considérer l'exemple suivant: $V = \{a, b, c, d, e\}$ et $w(a, b) = 10, w(a, c) = 1, w(b, d) = -15, w(d, c) = 1, w(d, e) = 1$ On voit alors que Dijkstra s'arete avact d'avoir propager une information ...

On peut alors proposer de modifier l'algo de la facon suivante: "Si un sommet x qui n'est [plus dans C, voit sont D modifié alors le remettre dans C. Je n'ai pas montré la correction.

Correction 5:

1. C'est en fait une modification de Dijkstra:

    ```
    Variable Prob[1..n]
    { Initialisation }
    C <- {2 .. n}
    P[1]=1
    ```

```
pour i de 2 a n faire
  Prob[i] <- M[1,i] ( 0 si n'existe pas )
  (pere[i]=1 si M[1,i] non zero)
fin pour
tant que C non vide faire
  v <- extrait dans C le max pour Prob
  pour tout voisin w de v faire
    Prob[w] <- max(Prob[w],Prob[v]*M[v,w])
    (et si P[w] a change pere[w] <- v)
  fin pour
fin tant que
```

2. La preuve est la même que celle de Dijkstra (Ce qui permet de la refaire): On suppose que l'algorithme n'est pas correct. On choisit u le premier sommet extrait de C tel que `Prob[u]<>p(s,u)` au moment de l'extraction (p(s,u) est l'analogue de $\delta(s,u)$). On a forcement $u \neq s$ car dès le départ $Prob[s] = 1 = p(s,s)$. Il existe un chemin de s à u car sinon $p(s,u) = 0 = p(u)$. Il existe donc un chemin de proba. max. p. Sur p, on considère y le premier sommet tel que, au moment où l'on retire u de C, $y \in C$ et x son prédécesseur dans p. On doit avoir $p(s,y) = Prob(y)$ au moment où l'on retire u de C. En effet, par hypothèse à ce moment x n'est plus dans C et donc $Prob(x) = p(s,x)$ (sinon u n'est pas le premier). A ce moment on recalcule $Prob(y) = Prob(x) * M(x,y) = p(s,x) * M(x,y) = p(s,y)$. Donc $p(s,y) = Prob(y)$. Puisque y apparaît avant u dans p et que toute les probas sont ≤ 1 on a: $p(s,y) \geq p(s,u)$ et donc:

$$\begin{array}{rcl} prob[y] & = & p(s,y) \\ & \geq & p(s,u) \\ & \geq & prob[u] \end{array}$$

Mais puisque u et y sont tous les deux dans C au moment de l'extraction de u on a $prob[u] \geq prob[y]$ et donc $p(s,u) = prob[u]$ à ce moment: Contradiction.

Donc l'algorithme est correct.

Correction 6:

1. C'est en fait une modification de Dijkstra:

```
Variable Cap[2..n] { Cap[1] n'a pas vraiment de sens }
{ Initialisation }
C <- {2 .. n}
pour i de 2 a n faire
  Cap[i] <- M[1,i] ( 0 si n'existe pas )
  (pere[i]=1 si M[1,i] non zero)
fin pour
```

```
tant que C non vide faire
  v <- extrait dans C le max pour Cap
  pour tout voisin w de v faire
    Cap[w] <- max(Cap[w],min(Cap[v],M[v,w]))
    (et si Cap[w] a change pere[w] <- v)
  fin pour
fin tant que
```

2. Une fois de plus la preuve est une adaptation de celle de Dijkstra. Si ils ont trop luttés pour la précédente on peut leur refaire sinon on leur laisse en exercice