Arbres recouvrants

Calcul d'un arbre de coût minimal
recouvrant un graphe connexe

Applications conception de réseaux
(téléphonique, électrique, d'intercommunication,...)
et étude de leur fonctionnement

Algorithmes

de Prim	$O(n^2)$
(adapté aux matrices d'adjacence)	
de Kruskal	$O(a \log a)$
(adapté aux listes de successeurs	
et graphes contenant peu d'arêtes)	

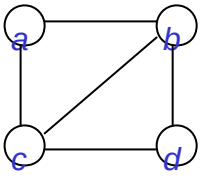
Graphes non orientés

$G=(S, A)$

S sommets card $S = n$

A arêtes card $A = a$

$A = \{\{s, t\}, \ldots\}$ $s \neq t, \ldots$ (pas de boucle)



chemin : ({a,b}, {b,c}, {c,d}, {d,b}, {b,a})
chemin simple : ({a,b}, {b,c}, {c,d})
cycle simple : ({a,b}, {b,d}, {d,c}, {c,a})

Chemin : $c = (\{s_0,s_1\}, \{s_1,s_2\}, \ldots, \{s_{k-1},s_k\})$ où les $\{s_{i-1},s_i\} \in A$

Chemin simple

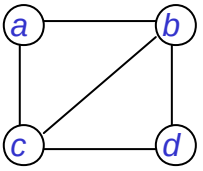
les sommets intermédiaires n'apparaissent qu'une seule fois

Cycle simple

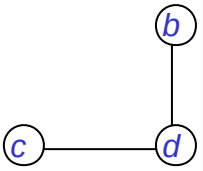
chemin simple avec $s_0 = s_k$

Sous-graphe

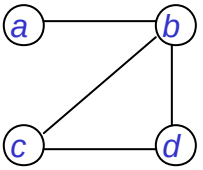
Graphe non orienté $G = (S, A)$



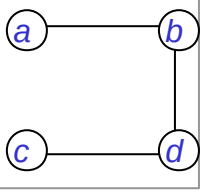
Sous-graphe
$G' \subseteq G$: G' graphe (S', A')
avec $S' \subseteq S$ et $A' \subseteq A$



Sous-graphe recouvrant
si $S' = S$



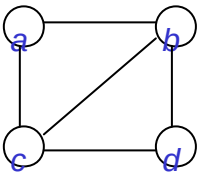
Arbre
graphe connexe sans cycle (simple)
Arbre recouvrant pour G
sous-graphe recouvrant qui est un arbre



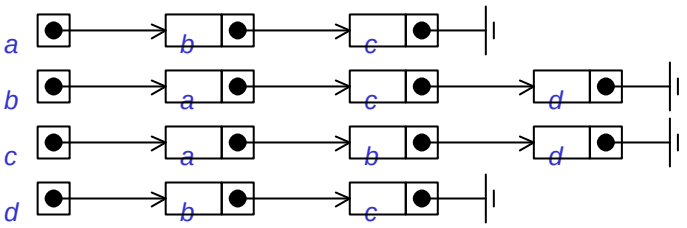
Représentations

Matrice d'adjacence
(symétrique)

	a	b	c	d
a	0	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	0



Listes de successeurs
(redondantes)

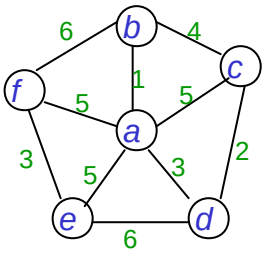


Problème ARCM

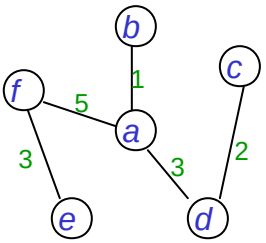
Graphe valué $G = (S, A, v)$ avec valuation $v : A \to \mathbf{R}$ **non-orienté et connexe**

Coût d'un sous-graphe $G' = (S', A')$: $\sum (v(p,q) \mid (p,q) \in A')$

Problème : déterminer un ARCM, arbre recouvrant de coût minimal pour G



coût = 14



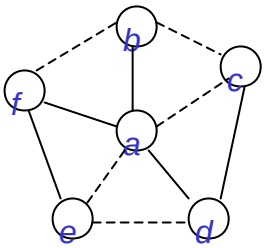
Arbre recouvrant

Graphe non-orienté et connexe $G = (S, A)$ card $S = n$
$T = (S, B)$ arbre recouvrant pour G

Propriétés

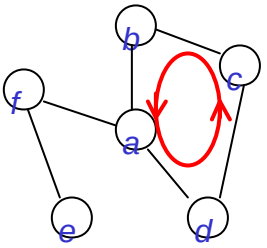
T possède n-1 arêtes : card $B = n$-1

si $\{p, q\} \in A$-B alors $H = (S, B + \{u,v\})$ possède un cycle



card $S = 6$

card $B = 5$

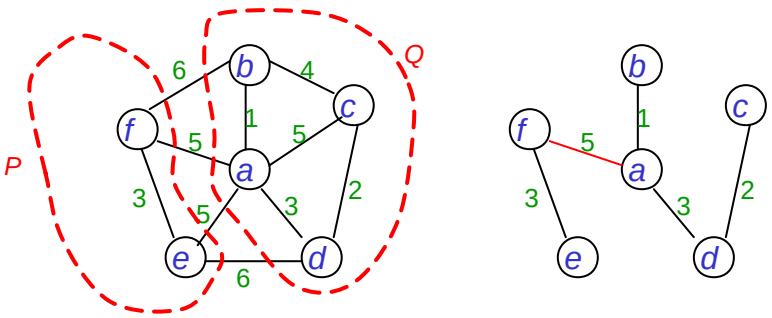


Coupure

Graphe valué $G = (S, A, v)$ non-orienté et connexe
$\{P, Q\}$ partition de S

Théorème

si $\{p, q\}$ une arête de coût minimal entre P et Q
il existe un ARCM qui contient $\{p, q\}$



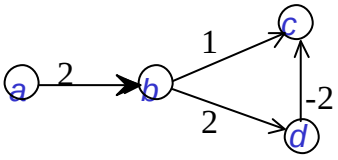
Preuve

Soit $\{p, q\}$ une arête de coût minimal entre P et Q
$p \in P \quad q \in Q$
Soit $T = (S, B)$ un ARCM pour G

si $\{p, q\} \in B$ terminé
sinon
$H = (S, B + \{p, q\})$ contient un cycle
ce cycle contient $\{u, v\} \in B,\ u \in P,\ v \in V$
soit $T' = (S, B - \{u, v\} + \{p, q\})$
T' est un arbre recouvrant et
coût $(T') \leq$ coût $(T) \Rightarrow$ coût (T') = coût (T)
donc $\{p, q\}$ contenu dans l'ARCM T'

Algorithmes gloutons

Traitement séquentiel des données
avec optimisation locale
Mais ne donne pas en général l'optimalité globale



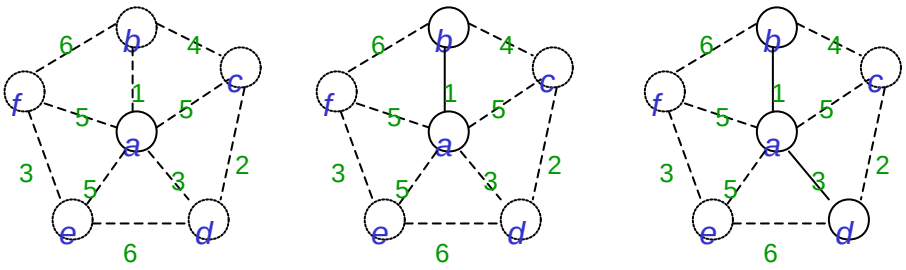
Plus court chemin de *a* à *c* ?

Optimalité pour
- rendeur de monnaie
- codage de Huffman
- algorithme de Dijkstra
- ARCM par algorithmes de Prim et de Kruskal

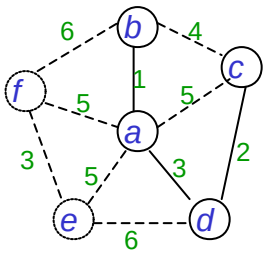
Méthode de Prim (1957)

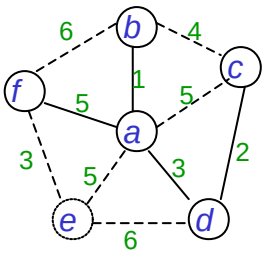
Calcul d'un ARCM :
faire grossir un sous-arbre jusqu'au recouvrement du graphe

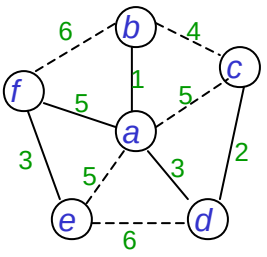
Exemple



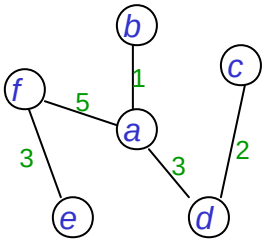
Exemple (suite)







ARCM
coût = 14



Algorithme de Prim

arbre PRIM(graphe ({1, 2, ..., n}, A, v)) **{**
    T ¬ {1} ;
    B ¬ ∅ ;
    `tant que` card $T < n$ **`faire {`**
        {p, q} ¬ arête de coût minimal
            telle que $p \in T$ et $q \notin T$;
        T ¬ T + {q} ;
        B ¬ B + {p, q} ;
    }
    `retour` (T, B) ;
}

Temps d'exécution : O(n^2) au moyen de deux tables indexées par les sommets

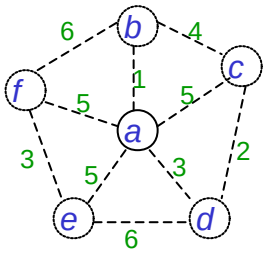
Implémentation

Tables **proche** et **coût** pour trouver l'arête {*p*, *q* }

$q \notin T$ **proche**[*q*] = $p \in T$
ssi $v(p, q) = \min \{ v(p', q) \mid p' \in T \}$

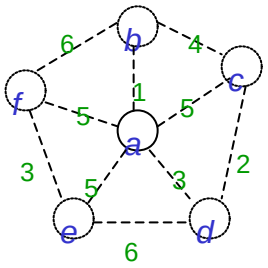
$q \notin T$ **coût**[*q*] = v(**proche**[*q*], *q*)

$q \in T$ **coût**[*q*] = +¥



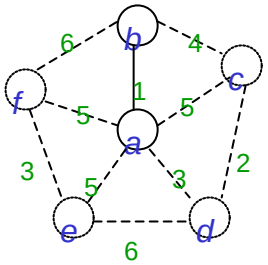
	a	*b*	*c*	*d*	*e*	*f*
proche		*a*	*a*	*a*	*a*	*a*
coût	¥	1	5	3	5	5

Une étape



	a	*b*	*c*	*d*	*e*	*f*
proche		*a*	*a*	*a*	*a*	*a*
coût	¥	1	5	3	5	5

ajout de *b* et {*a*, *b*}



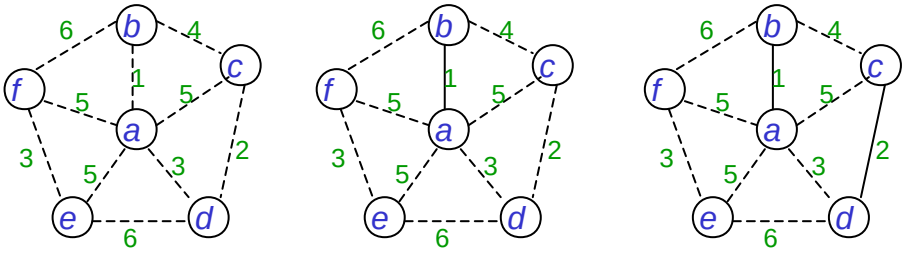
	a	*b*	*c*	*d*	*e*	*f*
proche		*a*	*b*	*a*	*a*	*a*
coût	¥	¥	4	3	5	5

Temps O(1+card *A*(*b*))

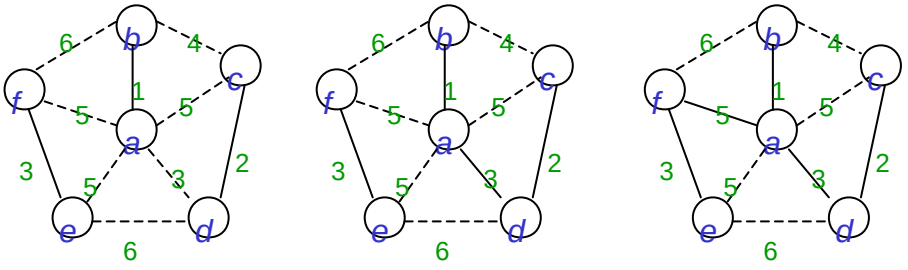
Méthode de Kruskal (1956)

Calcul d'un ARCM :
réunir deux sous-arbres disjoints par une arête de coût minimal

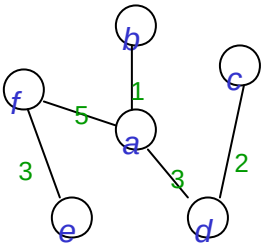
Exemple



Exemple (suite)



ARCM
coût = 14



Algorithme de Kruskal


```
arbre KRUSKAL( graphe ({1, 2, ..., n}, A, v) ) {
        Partition ← { {1}, {2}, ..., {n} } ;
        B ¬ ∅ ;
        tant que card Partition > 1 faire {
                {p, q} ¬ arête de coût minimal telle que
                         CLASSE(p) ≠ CLASSE(q) ;
                B ¬ B + {p, q} ;
                remplacer dans Partition CLASSE(p) et
                CLASSE(q) par leur UNION ;
        }
        retour ( ({1, 2, ..., n}, B ) ;
}
```

Temps d'exécution : O(card *A*.log card *A*) avec gestion efficace de la partition : type abstrait **CLASSE-UNION**

CLASSE / UNION

Gestion d'une partition de {1, 2, …, *n*} avec les **opérations principales**

- **CLASSE** : numéro de classe d'un élément
- **UNION** : union de classes disjointes

Implémentations possibles

1. table simple
2. arbres
3. idem + compression de chemins

Exemple n = 7

soit 1 ≡ 2 ; {1, 2} {3} {4} {5} {6} {7}
soit 5 ≡ 6 ; {1, 2} {3} {4} {5, 6} {7}
soit 3 ≡ 4 ; {1, 2} {3, 4} {5, 6} {7}
soit 1 ≡ 4 ; {1, 2, 3, 4} {5, 6} {7}
est-ce que 2 ≡ 3 ? oui car 2 et 3 dans la même classe

Par table simple

UMLV ©

	1	2	3	4	5	6	7
CLASSE	1	1	3	3	5	5	7

représente {1,2} {3,4} {5,6} {7}

UNION des classes (disjointes) de *p* et *q*

```
{       x ← CLASSE [p] ;  y ← CLASSE [q] ;
        pour k ← 1 à n faire
                si CLASSE [k] = y alors
                        CLASSE [k] ← x ;
}
```

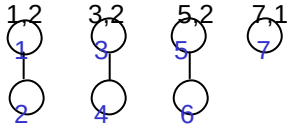
Temps
CLASSE : constant
UNION : O(*n*)

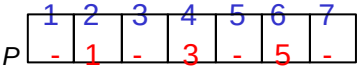
	1	2	3	4	5	6	7
CLASSE	1	1	1	1	5	5	7

représente {1,2,3,4} {5,6} {7}

Par arbres

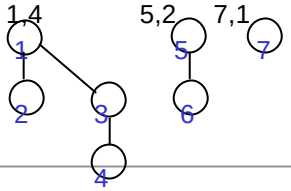
UMLV ©

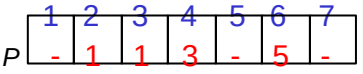
partition	{1,2}	{3,4}	{5,6}	{7}
CLASSE,TAILLE	1,2	3,2	5,2	7,1
arbres				




```
CLASSE(i ){
        k ← i ;
        tant que P[k] défini faire k ← P[k] ;
        retour ( CLASSE[k] ) ;
}
```

Temps
CLASSE : O(*n*)
UNION : constant

partition	{1,2,3,4}	{5,6}	{7}
CLASSE,TAILLE	1,4	5,2	7,1
arbres			

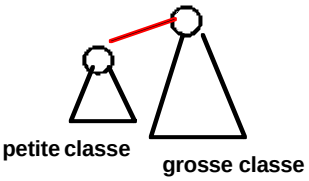


Union des arbres

Éviter des arbres filiformes pour réduire le temps de calcul de **`CLASSE`**`(`*i*`)`

Stratégie pour `UNION` : toujours mettre le petit arbre enfant de la racine du gros

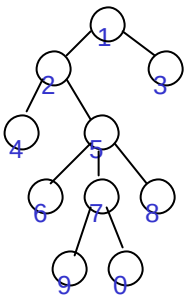
Temps
`CLASSE` : O(log*n*)
`UNION` : constant

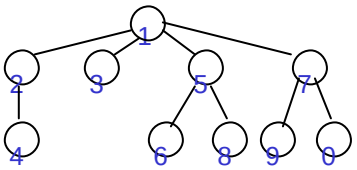


Preuve
niveau(*i*) augmente de 1 quand
union de *P* et *Q*, card *P* ≤ card *Q* et *i* ∈ *P*
i.e., quand la taille de la classe de *i* double au moins
Ceci ne peut arriver que $\lfloor \log_2 n \rfloor$ fois au plus

Compression de chemins

Idée : réduire le temps de calcul de **`CLASSE`**`(`*i*`)`
en « aplatissant » l'arbre à chaque calcul





après calcul
de **`CLASSE`**`(7)`

Temps de *n* calculs de **`CLASSE`** : O(*n* α (*n*))

où α(*n*) est le plus petit entier *k* tel que $n \leq 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ (*k* fois)

Preuve [Aho, Hopcroft, Ullman, 1974]

Ackermann

fonction A : **N** x **N** ® **N** définie par

$A(0, y) = 1$ $\quad y \geq 0$

$A(1, 0) = 2$

$A(x, 0) = x + 2$ $\quad x \geq 2$

$A(x, y) = A(A(x\text{-}1, y), y\text{-}1)$ $\quad x, y \geq 1$

Propriétés

$y = 0$ $\quad A(x, 0) = x+2$ $\quad x \geq 2$

$y = 1$ $\quad A(x, 1) = 2.x$ $\quad x \geq 1$

car $\quad A(1,1) = A(A(0,1),0) = A(1,0) = 2$

$\quad A(x,1) = A(A(x\text{-}1,1),0) = A(x\text{-}1,1)+2$, . . .

$y = 2$ $\quad A(x, 2) = 2^x$ $\quad x \geq 1$

car $\quad A(1,2) = A(A(0,2),1) = A(1,1) = 2$

$\quad A(x,2) = A(A(x\text{-}1,2),1) = 2.A(x\text{-}1,2)$, . . .

$y = 3$ $\quad A(x,3) = 2^{2^{\nearrow 2}}$ $\quad$ « tour de 2 en x exemplaires »

$\alpha\,(n)$ = plus petit k tel que $n \leq A(k, 3)$

$A(4,4)$ = « tour de 2 en 65536 exemplaires »

si n raisonnable, $\alpha(n) \leq 4$