Brute Force

T[1..n] : ez lesz a szöveg
P[1..m] : ez lesz a keresendő szöveg

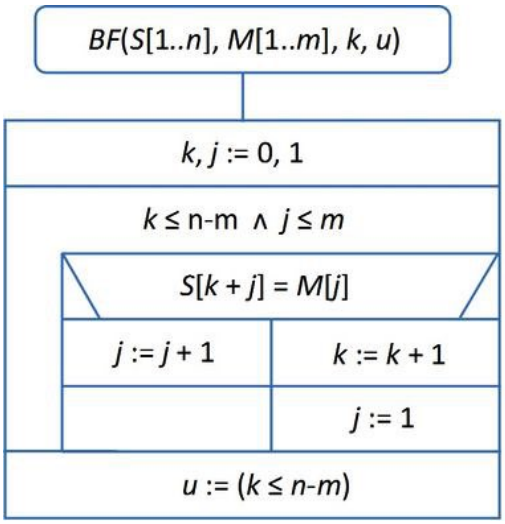
(X) ha eltérést találunk
(T) ha találatunk van
(V) ha kifutottunk a szövegből és túlindexeltünk

	1	2	3	4	5	6	7	8	9	10	11	
$T[1 \dots 11] =$	A	B	A	B	B	A	B	A	B	A	B	
$P[1 \dots 4] =$	B(X)	A	B	A								
		B	A	B	A(X)							
			B(X)	A	B	A						
				B	A(X)	B	A					
					B	A	B	A(T)				
						B(X)	A	B	A			
							B	A	B	A(T)		
								B(X)	A	B	A	
									B	A	B	A(V)

Ha n határozottan nagyobb mint m, akkor a műveletigények:

$$mT(n) \in \theta(n)$$
$$MT(n) \in \theta(n * m)$$

Stuktogram:



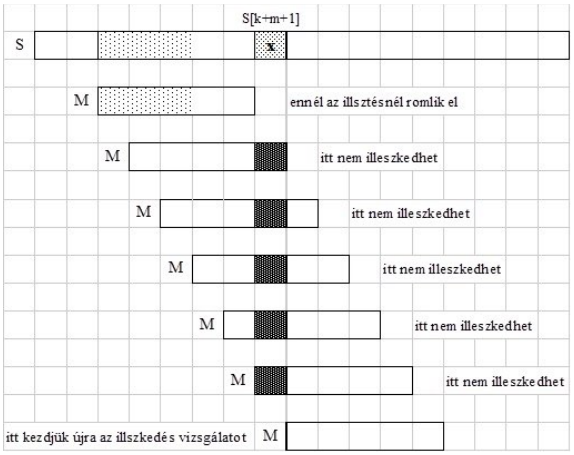
Quick-search

T[1..n] : ez lesz a szöveg
P[1..m] : ez lesz a keresendő szöveg

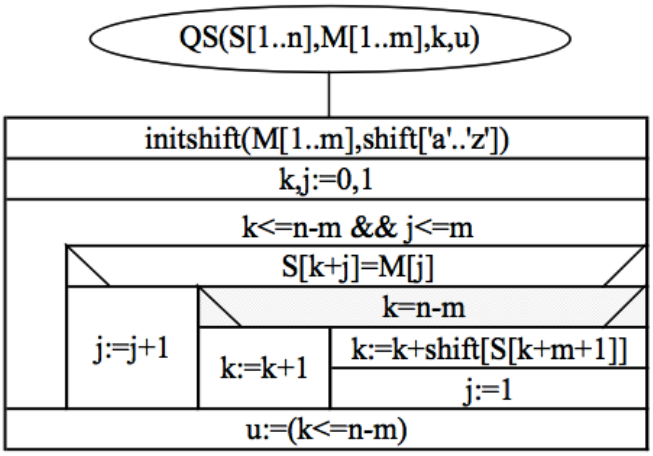
(X) ha eltérést találunk
(T) ha találatunk van

Az algoritmus két fontos lépésből áll.

1. Ha a szöveg, minta utáni első karaktere nem fordul elő a mintában, akkor átugorhatjuk teljesen, hiszen biztosan nem lesz egyezés, ameddig azt a kataktert vizsgáljuk.
2. Ha a szöveg, minta utáni első karaktere előfordul a mintában, akkor az előfordulások közül a jobbra lévőre ugrunk.



Stuktogram:



Példa:

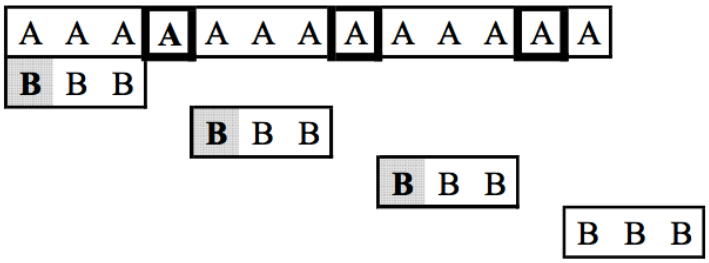
$T=$	A	D	A	B	A	D	C	A	D	A	B	C	A	B	A	D	A	C	A	D	A	D	A
$P=$	C_x	A	D	A																			
		C_x	A	D	A																		
				C_x	A	D	A																
					C_x	A	D	A															
$s=6$							C	A	D	A_T													
												C	A	D_X	A								
														C_X	A	D	A						
$s=17$																		C	A	D	A_T		
																				C_x	A	D	A

Műveletigény:

$$m\ddot{O}(n,m) = \Theta\left(\frac{n}{m+1}\right)$$

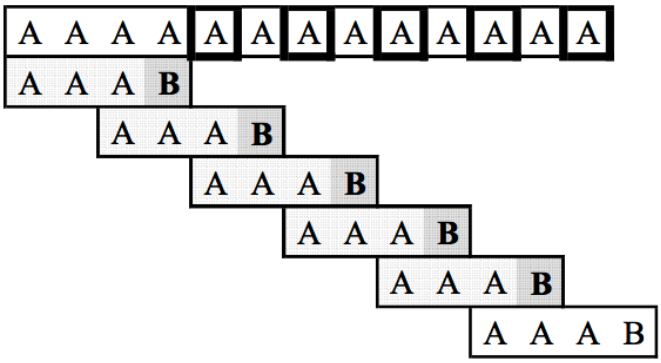
$$M\ddot{O}(n) = \Theta(n * m)$$

Legjobb eset: A minta olyan karakterekből áll, amelyek nem fordulnak elő a szövegben, így a minta első karakterénél már elromlik az illeszkedés, továbbá a minta utáni karakter sem fordul elő a mintában, így azt "átugorhatjuk".
Pl:



$$mÖ(n,m) = \Theta\left(\frac{n}{m+1}\right)$$

Legrosszabb eset: A minta végén romlik el az illeszkedés és csak kicsiket tudunk "ugrani".
Pl:



$$MÖ(n) = \Theta(n * m)$$

Knuth-Morris-Pratt

Amennyiben az illeszkedés elromlik, akkor egy hibás kezdetünk van, de ez a kezdet ismert, mivel az elromlás előtti karakterig egyezett a mintával. Ezt az információt használjuk fel, hogy elkerüljük az állandó visszalépést a szövegben a minta kezdetére

Látható, hogy a minta illeszkedő részének (M[1..5]) van egy olyan valódi kezdőszelete (valódi **prefix**e), amely egyezik ezen illeszkedő rész egy valódi végszeletével (valódi **szuffix**ével), azaz M [1..3] = M [3..5] ('ABA'='ABA').

A	B	A	B	A	B	A	C
A	B	A	B	A	C		
		A	B	A	B	A	C

$P[1 \ldots 8] =$

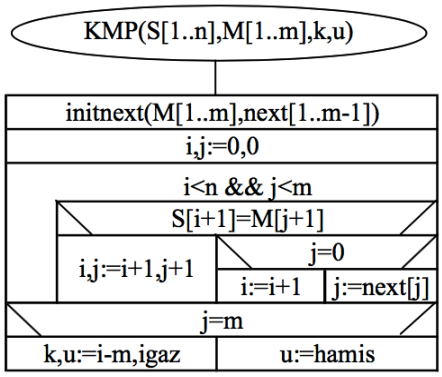
	1	2	3	4	5	6	7	8
	B	A	B	A	B	B	A	B

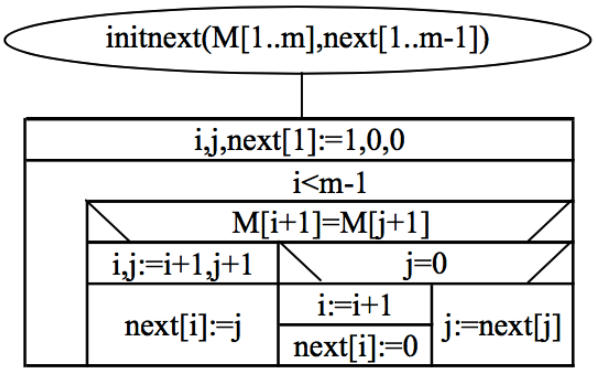
$s = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T[1...18]=	A	B	A	B	A	B	A	B	B	A	B	A	B	A	B	B	A	B
	B_X																	
		B	A	B	A	B	B_X											
				B	A	B	A	B	B	A	B_T							
									B	A	B	A	B	B_X				

$P =$	B	A	B	A	B	B	A	B
j	1	2	3	4	5	6	7	8
$next(j)$	0	0	1	2	3	1	2	3
			B	A	B	A		
				B	A	B		
					B	A		
						B	A	B

Stuktogram:





Műveletigény:

initNext() : Θ(m)

keresés: Θ(n)

Gráfalgoritmusok

A gráf egy kulcsokból és élekből álló halmaz, ahol a csúcsok halmaza véges, az élhalmaz pedig részhalmaza az él*él halmaznak. Képletekkel leírva:

$G = (V, E)$

$0 < |V| \leq +\infty \quad E \subseteq V * V$

$W: E \rightarrow R$

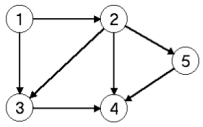
Ábrázolás:

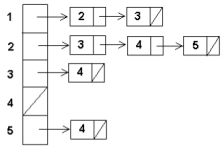
1. A csúcsmátrix (szomszédossági mátrix) esetében az ábrázolásnál fel kell tenni, hogy a csúcsok halmaza, $V = \{1, \ldots, n\}$ között fel vannak címkézve (ezek részhalmazai N-nek).

Először nézzük a súlyozatlan eseteket. Ilyenkor azt mondjuk, hogy ha c-vel jelöljük a mátrixot, akkor $c[i,j] = \begin{cases} 1, \text{ha (i,j)} \in E \\ 0, \text{ha (i,j)} \notin E \end{cases}$. Míg ha súlyozott, akkor a $c[i,j] = \begin{cases} w(i,j), \text{ha } (i,j) \in E \ (i \neq j) \\ 0, \text{ha } i = j \\ +\infty, \text{ha } i \neq j \text{ és } (i,j) \notin E \end{cases}$.

2. A megbeszélt memória igényen kívül, milyen előnye van az egyik ábrázolásnak a másikkal szemben?
   - Éllistás ábrázolás esetén gyorsan végignézhetjük egy adott csúcsból kiinduló éleket.
   - Szomszédsági-mátrix ábrázolás esetén gyorsan eldönthető, hogy egy (i,j) pár éle-e a gráfnak

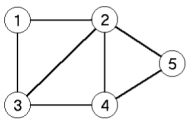
Irányított gráf esetén:

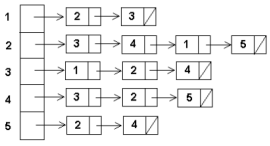




Helyfoglalás: $n + e$

Irányítatlan gráf esetén:





Helyfoglalás: $n + 2e$

Ha $u \rightarrow v$ csúcsok vannak, akkor:

1. faél: Ha egy mélységi fának egy éle van
2. visszaél: Ha egy mélységi fában a v csúcs az u őse
3. előreél: Ha egy mélységi fában a v csúcs nem közvetlenül az u leszármazottja
4. keresztél: Vagy egy mélységi fa két különböző csúcsa között van, vagy két különböző mélységi fát köz össze.

Szélességi bejárás

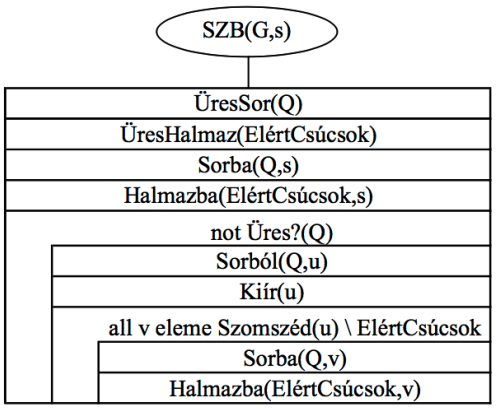
Az algoritmus elvét az előzőekben már láttuk, most foglaljuk össze röviden:

1. Először elérjük a kezdőcsúcsot.
2. Majd elérjük a kezdőcsúcstól 1 távolságra lévő csúcsokat (a kezdőcsúcs szomszédait)
3. Ezután elérjük s-től 2 távolságra lévő csúcsokat (a kezdőcsúcs szomszédainak a szomszédait), és így tovább.
4. Ha egy csúcsot már bejártunk, akkor a későbbi odajutásoktól el kell tekinteni

Az ADT szinten tárgyalt halmazt most a csúcsok színezésével valósítsuk meg, sőt a csúcsoknak ne csak két állapotát különböztessük meg, hanem az alábbi három állapotát.

1. Amikor egy csúcsot még nem értünk el legyen fehér színű. Induláskor a kezdőcsúcs kivételével minden csúcs ilyen. (u ∉ Q és u ∉ ElértCsúcsok)
2. Amikor egy csúcsot elérünk és bedobjuk a sorba, színezzük szürkére. A kezdőcsúcs induláskor ilyen. (u ∈ Q és u ∈ ElértCsúcsok)
3. Amikor egy csúcsot kivettünk a sorból és kiterjesztettük (elértük a szomszédait), a színe legyen fekete. (u ∉ Q és u ∈ ElértCsúcsok)

Stuktogram:

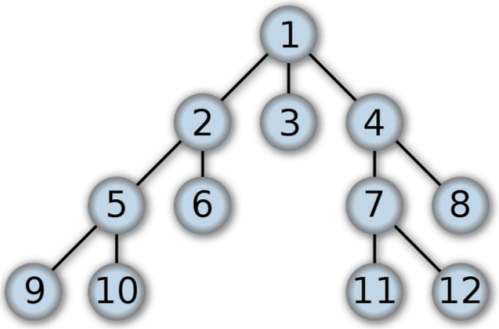


Műveletigény:

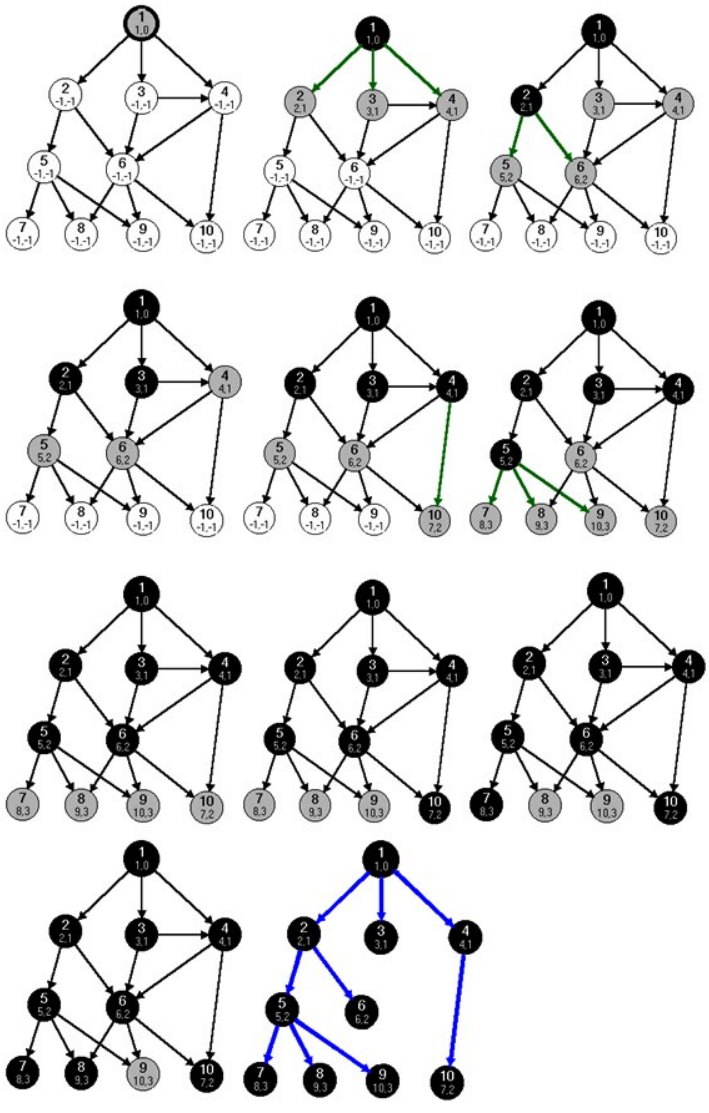
Beállítani fehérre: Θ(n)

Éllistánál: O(n + e)

Csúcsmátrixnál: O(n 2)



Szemléltetés:



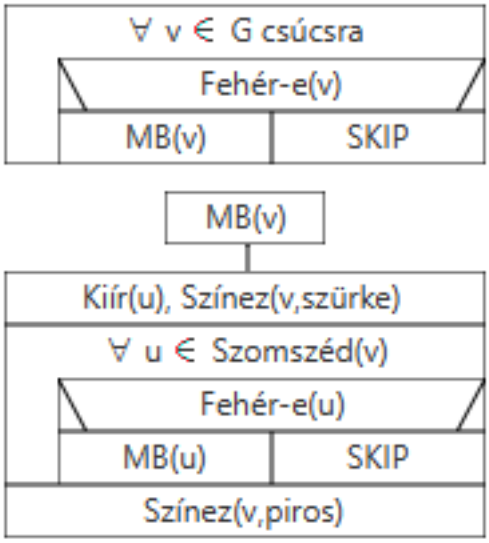
Mélységi bejárás

A DFS során feldolgozott (u, v) él:

1. faél akkor, és csak akkor, ha v színe még fehér
2. visszaél, akkor és csak akkor, ha v színe szürke
3. előreél, akkor és csak akkor, ha v színe fekete, és u.d < v.d
4. keresztél akkor, ha v színe fekete, és u.d > v.d.

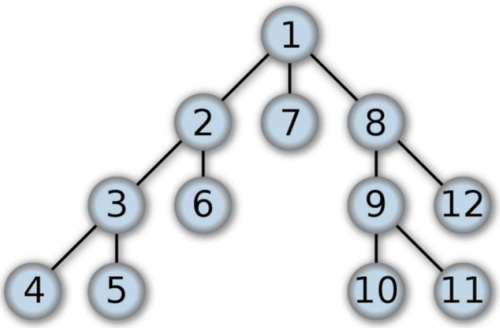
Ezen tétel segítségével bármely gráfban bármely élet fel tudunk címkézni.

Stuktogram:



Műveletigény:

$O(n+e)$

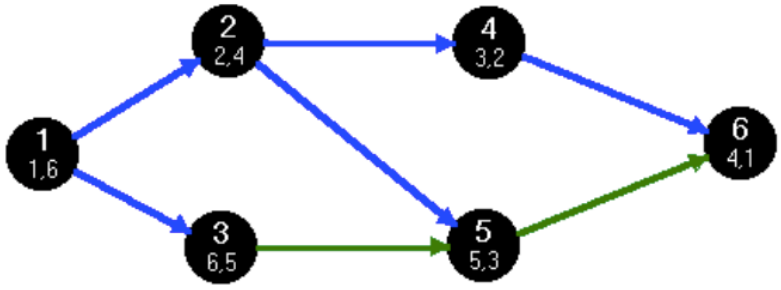


Topologikus rendezés

G-nek ∃ topologikus rendezése G DAG (Irányított Körmentes Gráf)

Mélységi bejárás után:

Majd a bejárás utolsó csúcsaként elhagyjuk az 1-es csúcsot is. A veremben van a gráf összes csúcsa a befejezésük szerint: V=[6,4,5,2,3,1.



Műveletigény:

$O(n+e)$

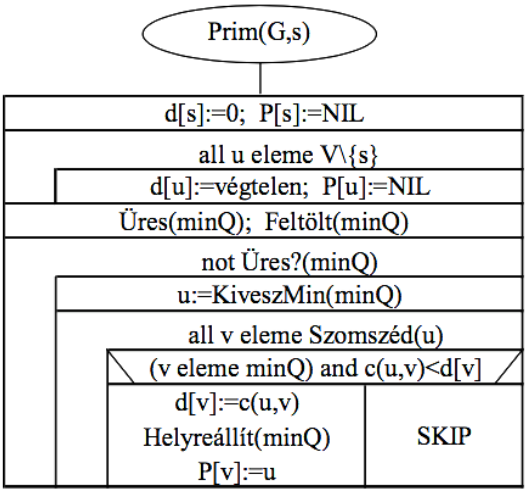
Ha leírom a veremben lévő sorrend szerint a csúcsokat egymás után, és behúzom az éleket, az élek irányítása csak jobbra fog mutatni (ha jól csináltad)

Prim algoritmus

A Prim algoritmus minden lépésben a kék szabályt alkalmazza egy s kezdőcsúcsból kiindulva. Az algoritmus működése során egyetlen kék fát tartunk nyilván, amely folyamatosan növekszik, míg végül minimális költségű feszítőfa nem lesz.

Kezdetben a kék fa egyetlen csúcsból áll, a kezdőcsúcsból, majd minden lépés során, a kék fát tekintve a kék szabályban szereplő X halmaznak, megkeressük az egyik legkisebb súlyú élt (mohó stratégia), amelynek egyik vége eleme a kék fának (X-ben van), a másik vége viszont nem (nem eleme X-nek). Az említett élt hozzá vesszük a kék fához, azaz az élt kékre színezzük, és az él X-en kívüli csúcsát hozzávesszük az X-hez.

Stuktogram:



Műveletigény:

Rendezetlen tömbb esetés: $O(n^2)$

Gráf:

a-b: 2
a-d: 4
b-c: 6
b-e: 3
b-d: 5
c-e: 6
c-f: 5
d-e: 1
e-f: 2

Algoritmus:

	c							π					
	a	b	c	d	e	f	Q	a	b	c	d	e	f
	0	∞	∞	∞	∞	∞	a,b,c,d,e,f	∅	∅	∅	∅	∅	∅
a		2		4			b,c,d,e,f		a		a		
b			6		3		c,d,e,f			b		b	
e				1		2	c,d,f				e		e
d							c,f						
f			5				c			f			
c							$<>$						
	0	2	5	1	3	2		∅	a	f	e	b	e

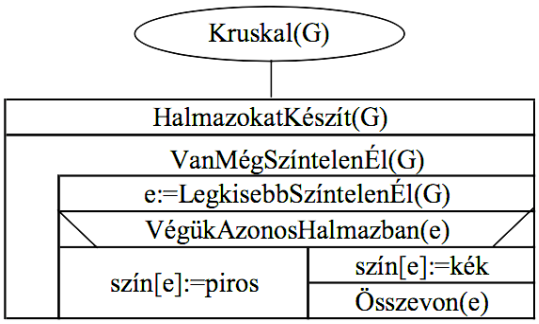
Elindultunk egy kiindulású csúcsból, majd végignézve a szomszédait, felírtuk, hogy az adott csúcsból egy szomszédos kulcs milyen költséggel érthető el, a Q sorból kivettük, majd felírtuk a szomszédos csúcs szülőjét, ahonnan elértük. Mindig figyeltük a költségeket, a szomszédok figyelembe vételénél ugyanis, ha egy csúcsnak több is volt, először azt vesszük figyelembe, amelyiknek kevesebb a költsége, hogy elérjük. Amelyik csúcsokat vettük, azok feszítőfát fognak alkotni. Ha nem tudtunk venni olyan csúcsot, amelyet már elértünk, de el tudnánk érni, akkor az adott sor üres marad.

Kruskal algoritmus

Az oszlopok mentén haladva, vettük az éleket, és ha a berajzolásával még nem keletkezett kör, akkor berajzoltuk a gráfba. Ha kihagytuk, mindig mentünk tovább. A lényeg, hogy addig menjünk, míg annyi élt rajzoltunk be, míg már többet nem tudunk berajzolni kör létrehozása nélkül.

Ha van két komponens, amelyek csak egy éllel vannak összekötve, akkor vegyünk egy vágást, amely kettészedi ezeket. Ez az egy él tehát könnyű él lesz, mert ennek a legalacsonyabb költségű, ha lenne ennél olcsóbb, az a rendezésben is előrébb lett volna.

Stuktogram:



Műveletigény:

$O(e \log n)$

Gráf:

a-b: 2
a-d: 4
b-d: 3
d-e: 1
b-e: 3
b-c: 4
c-e: 6
c-f: 5
e-f: 5

	1	2	3	4	5	6	7	8	9
	$d-e$ 1	$a-b$ 2	$e-f$ 2	$b-e$ 3	$a-d$ 5	$d-b$ 5	$c-f$ 5	$b-c$ 6	$e-c$ 6
a b c d e f	a b c d-e f	a-b c d-e f	a-b c d-e-f	a-b c | d-e-f	a-b c | d-e-f	a-b c | d-e-f	a-b c | | d-e-f		

Dijkstra algoritmus

Elindultunk a startcsúcsból, majd vettük a szomszédait, és leírtuk az elérési költségeit. A soron lévő csúcsot kivettük a Q sorból, és utána leírtuk az adott csúcs szülőjét, hogy onnan tudjuk elérni. Itt viszont figyeljük az elérési költségeket, ezért 3 a c elérési költsége d- ből,viszont még mindig kevesebb, mintha a-ból közvetlen érnénk el. Az épp soron következő kulcsoknál figyelni kell az költségeket, mert mindig az aktuálisan legalacsonyabbat kell figyelembe venni. Ha egy már járt csúcsot el tudnánk érni máshonnan is, de magasabb költséggel, akkor azt nem írjuk bele a táblázatba, figyelmen kívül hagyjuk.

	d						π				
	a	b	c	d	e	Q	a	b	c	d	e
	0	∞	∞	∞	∞	a,b,c,d,e	∅	∅	∅	∅	∅
a		2	4	1		b,c,d,e		a	a	a	
d			3			b,c,e			d		
b			2			c,e			b		
c						e					
e						< >					
	0	2	2	1	0		∅	a	d	a	∅
	a	b	c	d	e		a	b	c	d	e

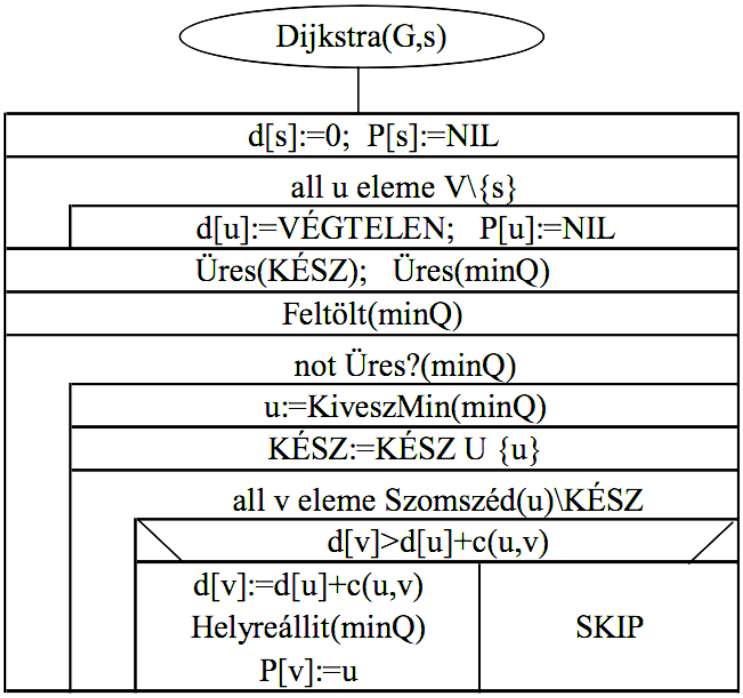
Gráf:

$a \to d: 1$
$a \to b: 2$
$a \to d: 4$
$b \to c: 0$
$d \to c: 2$
$d \to b: 1$
$c \to d: 1$
$e \to d: 3$
$e \to c: 2$

Műveletigény:

Rendezetlen tömb esetés: $O(n^2)$

Stuktogram:



Bellmann-Ford algoritmus

Sorban haladtunk a csúcsok mentén ábécé sorrendben. Megnéztük, hogy melyik csúcsokat érhetjük el belőle, és beírtuk a táblázatba először is, hogy mekkora az elérés költsége. Majd ezt beírjuk a sorba, kivesszük a sorból azt a csúcsot, amelyiknek a gyermekeit keressük, és leírjuk az adott csúcs szülőjét. Ha sokadik elérésre jutunk el egy csúcsra, akkor az élek költségeit összeadjuk. Ha olyan csúcsba jutunk el, ahol már voltunk, akkor összehasonlítjuk a költségeiket, és az alacsonyabbat vesszük, ugyanezt vesszük a π értékeinél is, azaz hogy az adott csúcsnak melyik a szülője. Az egész algoritmust addig ismételgetjük, míg üres nem lesz a Q sor.

d	a	b	c	d	Q	π	a	b	c	d	(*)
	0	∞	∞	∞	$< a >$		Ø	Ø	Ø	Ø	INIT
a		3	1		$< b, c >$			a	a		0. menet
b				1	$< c, d >$					b	1. menet
c		0			$< d, b >$			c			
d			~~5~~		$< b >$						2. menet
b				-2	$< d >$					b	
d			~~2~~		$< >$						3. menet
∑	0	0	1	-2			Ø	c	a	b	

A gráfunk: $(start) \to a: 0, a \to c: 1, c \to b: -1, b \to d: -2$, a csúcsokban az értékek: $a: 0, c: 1, b: 0, d: -2$.

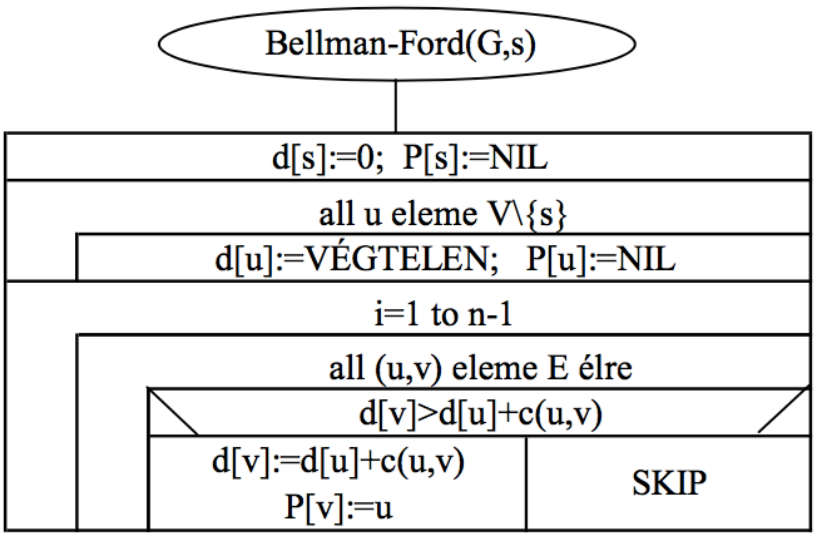
Műveletigény:

$$O(n * e)$$

Gráf:

$a \to b: 3$
$a \to c: 1$
$b \to d: -2$
$c \to b: -1$
$d \to c: 4$

Stuktogram:



Ennek az algoritmusnak az a legnagyobb hátránya, hogy igen magas futási ideje is lehet. De mi van, ha van olyan gráfunk, amelyben vannak negatív élek, de nincs benne negatív kör? A Dijkstra algoritmus a maga alacsonyabb műveletigényével ezt hatékonyabban tudta megoldani. De a kérdés az, hogy ha nincs egyáltalán kör a gráfban, ki lehet-e ezt használni? A válasz: igen, mégpedig úgy, hogy a gráf csúcsait topologikusan rendezzük és utána a csúcsokat kiterjesszük.

Floyd-Warshall algoritmus

Dijkstra: Legrövidebb út ***egy*** csúcsból kiindulva.
Bellmann-Ford: Legrövidebb út ***egy*** csúcsból kiindulva, ***negatív*** súlyokat is megengedve.
Floyd-Warshall: Legrövidebb út ***több*** csúcsból kiindulva, ***negatív*** súlyokat is megengedve.

Műveletigény:

$\theta(n^3)$

????