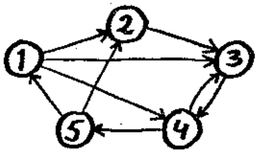
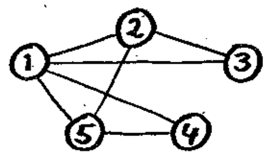
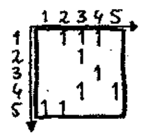
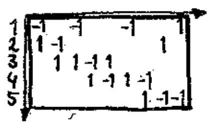
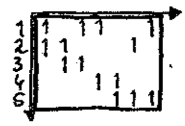
Representation av graf

Riktad graf	Oriktad graf
	

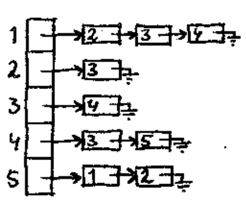
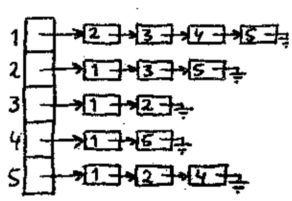
Som grannmatris (adjacency matrix):




Som kantmatris (incidence matrix):

Som grannlista (adjacency list):

Breddenförstsökning i graf

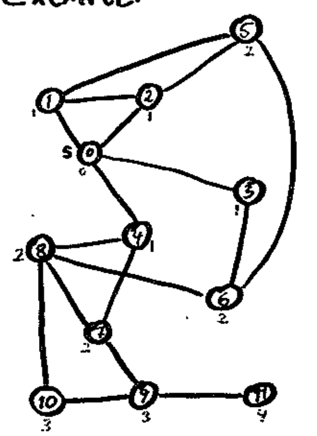
Breddenförstsökning (BFS) går igenom alla hörn som kan nås från ett speciellt starthörn s i följande ordning:
- först alla grannar till s
- sedan grannarna till grannarna till s
- sedan alla hörn på avstånd 3 från s
- sedan alla hörn på avstånd 4, osv

BFS(V, E, s) =

För varje hörn $u \in V$:
  $d[u] \leftarrow \infty$
$d[s] \leftarrow 0$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$ do
  $u \leftarrow$ Dequeue(Q)
  för varje granne v till u:
    if $d[v] = \infty$ then
      $d[v] \leftarrow d[u] + 1$
      Enqueue(Q, v)

Om något ska göras med varje hörn i grafen kan det göras med u här i algoritmen

Exempel:



Tidskomplexitet: $O(|V| + |E|)$

Djupetförstsökning i graf

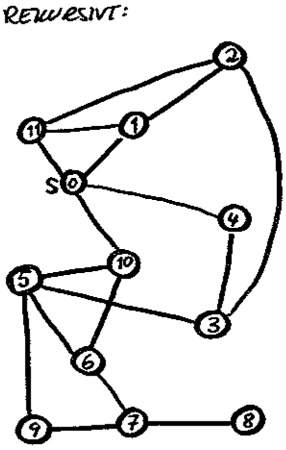
Djupetförstsökning (DFS) börjar liksom BFS i starthörnet s, men går sedan så långt det går i grafen (utan att besöka något redan tidigare besökt hörn). När det inte går längre backar man tillbaka ett steg i taget tills det går att fortsätta framåt igen. Detta implementeras enklast rekursivt:

DFS(V, E, s) =

För varje hörn $u \in V$:
  color[u] ← white
DFS_VISIT(V, E, s)

DFS_VISIT(V, E, u) =

  color[u] ← black
  gör något med u här
  för varje granne v till u:
    if color[v] = white then
      DFS_VISIT(V, E, v)



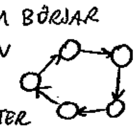
Tidskomplexitet: $O(|V| + |E|)$

Enkel tillämpning av sökning:

Avgör om en graf är en DAG

DAG = riktad acyklisk graf

Cykel = stig av riktade kanter som börjar och slutar i samma hörn



Idé: utvidga DFS så att man upptäcker kanter till förfäder (back edges).

DFS(V, E) =

För varje hörn $u \in V$: color[u] ← white
För varje hörn $u \in V$: if color[u] = white then DFS_VISIT(u)

DFS_VISIT(u) =

  color[u] ← gray
  för varje granne v till u:
    if color[v] = gray then write "cykel"
    if color[v] = white then DFS_VISIT(v)
  color[u] ← black

Tidskomplexitet: $O(|V| + |E|)$

Minimalt spännande träd i viktad graf

Ett spännande träd för en graf G är en delgraf till G som är ett träd (sammanhängande, har inga cykler) och innehåller alla hörn i G.

Vikten för ett spännande träd är summan av dom ingående kanternas vikter.

Prims algoritm:

Indata: Graf $G=\langle V,E\rangle$, kantvikter $f: E\to\mathbb{R}$, starthörn s

Utdata: Ett minimalt spännande träd för G lagrat med faderspekare $\pi(u)$

Prim(V,E,f,s) =

```
Q ← V
för varje u∈Q:
   key[u] ← ∞
key[s] ← 0      (Q är en heap med s överst)
π[s] ← NIL
while Q ≠ ∅ do
   u ← HeapExtractMin(Q)
   för varje granne v till u:
      if v∈Q and f(u,v) < key(v) then
         π[v] ← u
         key[v] ← f(u,v)   (här måste v flyttas i heapen)
```

Tidskomplexitet: $\mathcal{O}(|V|\log|V| + |E|\log|V|) = \mathcal{O}(|E|\log|V|)$

Minimalt spännande träd, Kruskal

Kruskals algoritm:

Indata: Graf $G=\langle V,E\rangle$, kantvikter $f: E\to\mathbb{R}$

Utdata: Ett minimalt spännande träd för G lagrat som en kantmängd $A\subseteq E$.

Kruskal(V,E,f) =

```
A ← ∅
för varje u∈V
   MakeSet(u)
sortera kanterna i E efter stigande vikt
för varje kant (u,v)∈E i stigande viktsordning:
   if FindSet(u) ≠ FindSet(v) then
      A ← A ∪ {(u,v)}
      Union(u,v)
returnera A
```

Komplexitetsanalys:

MakeSet(u) tar tid $\mathcal{O}(1)$

FindSet och Union tar tid $\mathcal{O}(\log|V|)$

sorteringen av E tar tid $\mathcal{O}(|E|\log|E|)$

Totalt: $\mathcal{O}(|V|\cdot 1 + |E|\log|E| + |E|\log|V|) = \mathcal{O}(|E|\log|E|)$

om grafen är sammanhängande

Korrekthet för Prim och Kruskal

Idé: Visa att varje kant som läggs till i algoritmen är säker, dvs ingår i något MST.

Definitioner:

- Ett **snitt** (cut) är en delning av V i S och $V-S$.
- En kant **korsar** snittet om ena änden $\in S$ och andra $V-S$.

Sats: Givet $G=\langle V,E\rangle$, $f: E\to\mathbb{R}$, $A\subseteq E$, $S\subseteq V$. Om
- det finns ett MST som innehåller A,
- ingen kant i A korsar snittet $(S, V-S)$,
- (u,v) är det lättaste kant som korsar snittet

så är (u,v) säker att lägga till, dvs det finns ett MST som innehåller $A\cup\{(u,v)\}$.

Bevis: Låt T vara MST som innehåller A men inte (u,v). Konstruera T' som är ett MST och innehåller $A\cup\{(u,v)\}$:

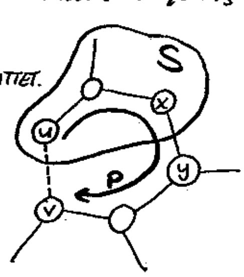
T innehåller stig p mellan u och v.
Det finns kant (x,y) i p som korsar snittet.
Låt $T' = T\cup\{(u,v)\} - \{(x,y)\}$.

T' är uppenbart ett spännande träd

(u,v) är den lättaste korsande kanten

$\Rightarrow f(u,v)\le f(x,y) \Rightarrow |T'|\le|T|$

$\Rightarrow T'$ är MST.



Algoritm för grafproblemet "kortaste stig"

Dijkstras algoritm:

Indata: $G=\langle V,E\rangle$, $f:E\to\mathbb{N}$, $s\in V$, $t\in V$

Utdata: Längden av den kortaste stigen i G från s till t

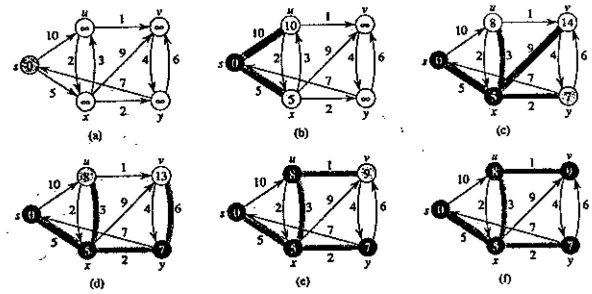
Märk varje hörn med det hittills kortaste kända avståndet från s.

Upprätta en mängd S med dom hörn till vilka den optimala kortaste stigen är känd.

Algoritm:

1. För varje hörn $u\in V$:
   om $(s,u)\in E$ märk u med $f(s,u)$
   annars märk u med ∞
2. märk s med 0 och låt $S=\{s\}$
3. så länge $t\notin S$:
   utvidga S med det hörn som är märkt med det kortaste avståndet och uppdatera hörnmärkningen
4. returnera avståndet som t är märkt med

Exempel på Dijkstras algoritm:



Analys:

S utvidgas $|V|$ gånger (högst).
Vid varje utvidgning letar man upp det hörn som är märkt med kortaste avståndet: $O(|V|)$
Uppdatering av hörnmärkningen görs högst en gång för varje kant i grafen: $O(|E|)$
Initiering av S och märkningen tar tid $O(|V|+|E|)$
Totalt: $O(|V|^2+|E|+|V|+|E|) = O(|V|^2)$
(eftersom $|E|\in O(|V|^2)$)

Korrekthet för Dijkstras algoritm

Låt $\delta(s,v)$ vara det kortaste avståndet från s till v.
Låt $d[v]$ vara hörnet v-s märkning i ett läge i algoritmen.

Bevisskiss:

Notera att $d[v]\geq\delta(s,v)$ alltid gäller för alla hörn.

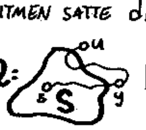
Induktion över S:

Basfall: $S=\{s\}$, $d[s]=0$, $\delta(s,s)=0$ **OK!**

Induktionssteg: Visa att om $d[v]=\delta(s,v)$ för alla $v\in S$ när u just ska läggas till S så är $d[u]=\delta(s,u)$.

Fall 1:  Kortaste stigen från s till u går helt inuti S utom sista kanten (x,u).

Antagandet $\Rightarrow d[x]=\delta(s,x)$
Algoritmen satte $d[u]=d[x]+f(x,u)=\delta(s,x)+f(x,u)=\delta(s,u)$
OK!

Fall 2:  Låt y vara första hörnet utanför S i kortaste stigen från s till u.

Fall 1 $\Rightarrow d[y]=\delta(s,y)\leq\delta(s,u)$
Algoritmen lägger till u före $y \Rightarrow d[u]\leq d[y]$
Vi har nu: $d[y]\leq\delta(s,u)\leq d[u]\leq d[y]$
$\Rightarrow d[y]=\delta(s,u)=d[u]$ **OK!**

Alla hörn som kan nås från s kommer med i S. Övriga har $d[v]=\infty$