Tampereen yliopisto

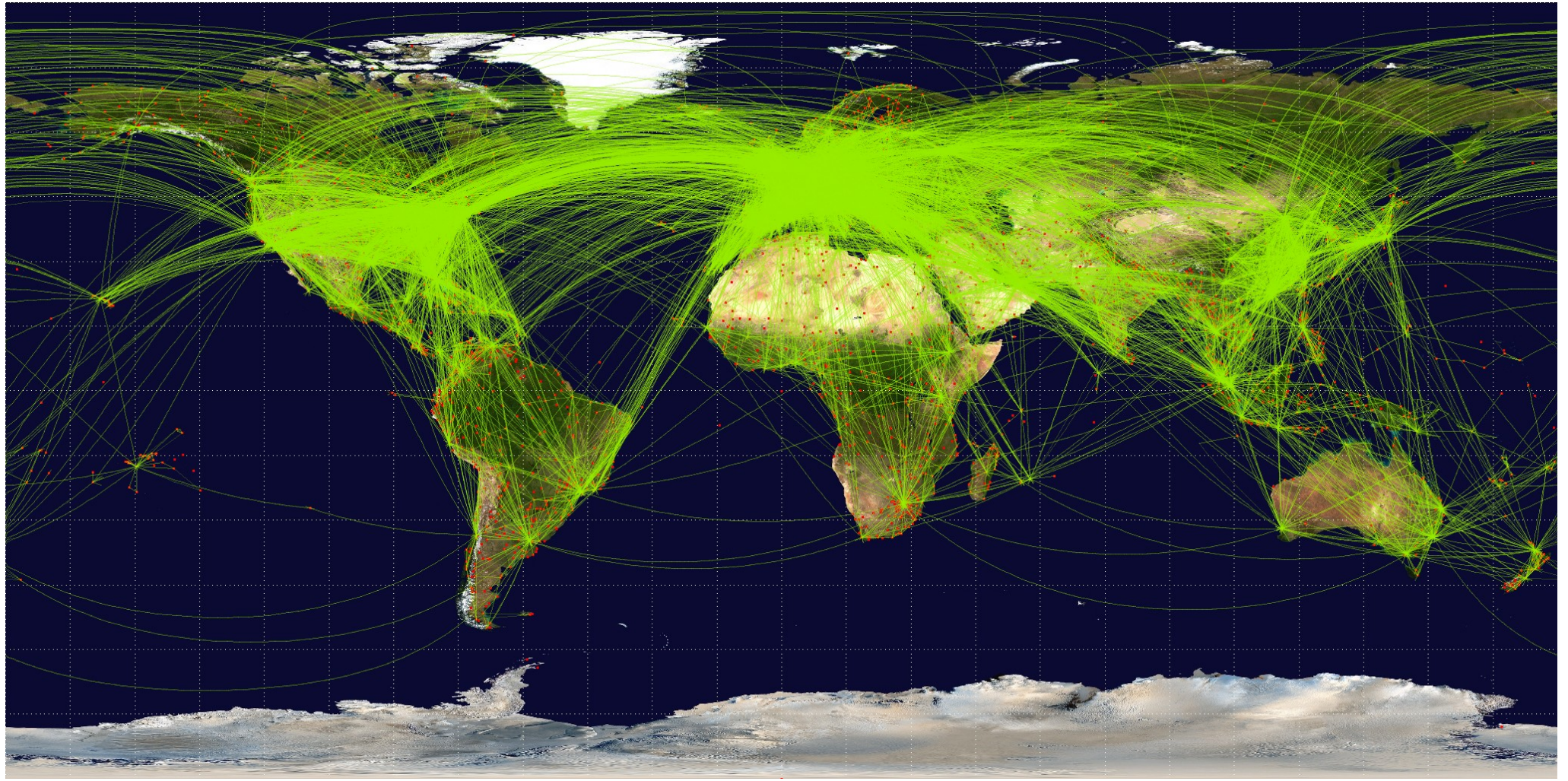
Graafit

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

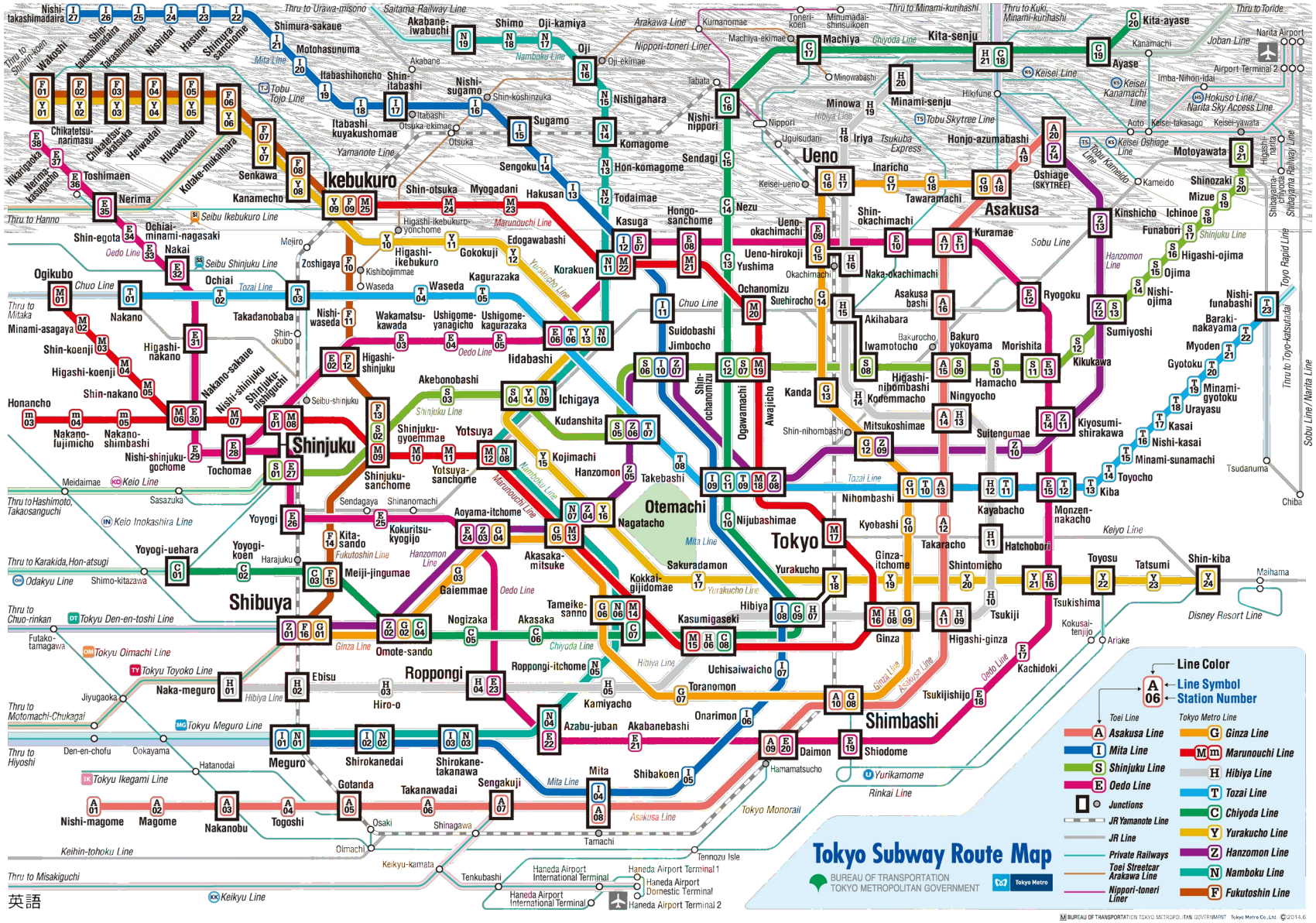
Tampereen yliopisto

Graafit



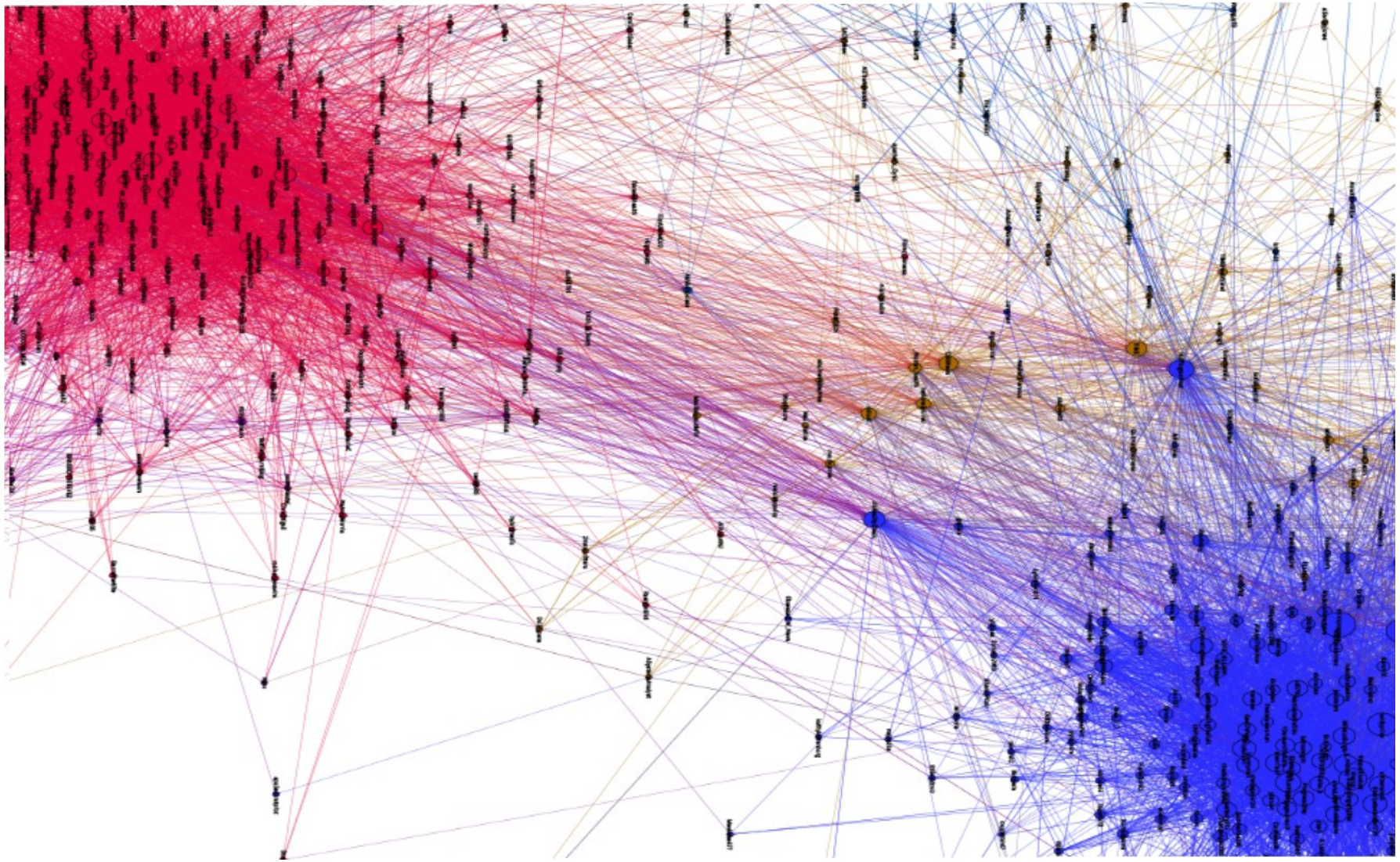
Tampereen yliopisto

Graafit



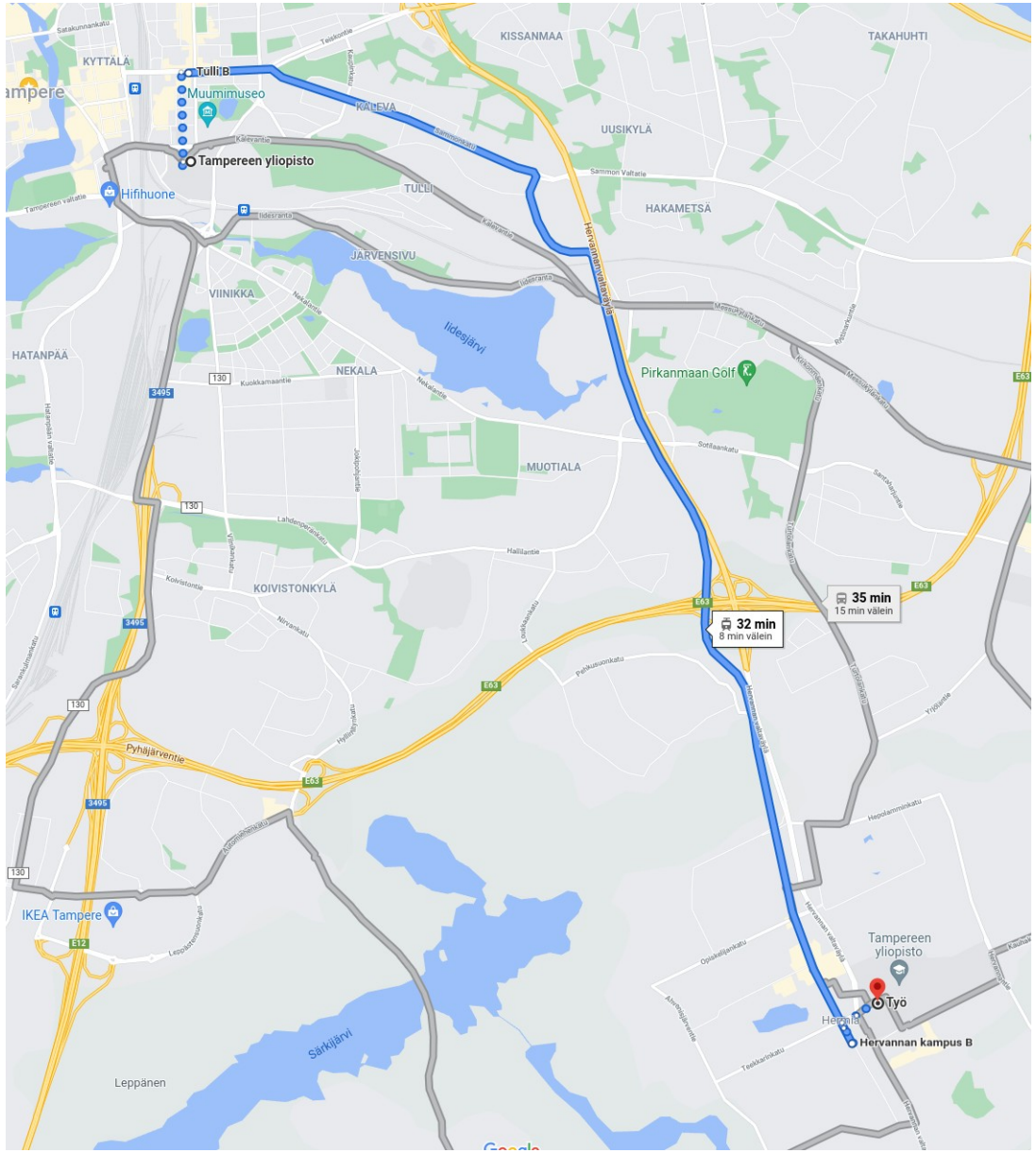
Tampereen yliopisto

Graafit

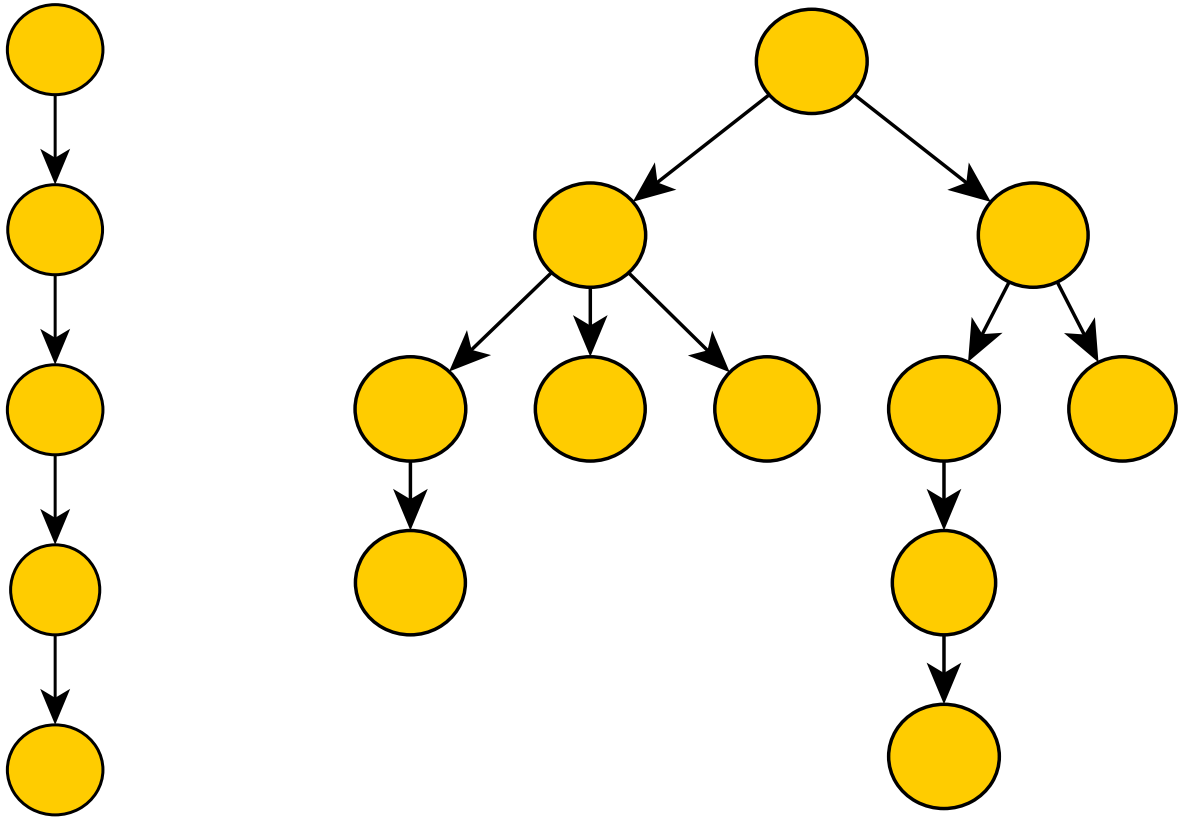


Tampereen yliopisto

Graafit

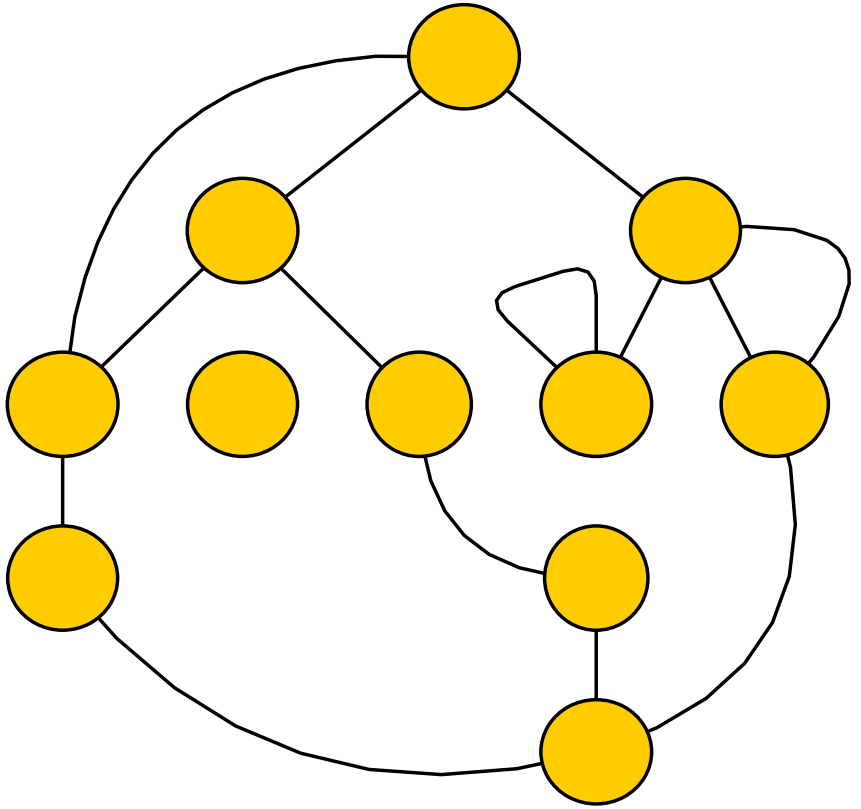


Lista-puu-graafi



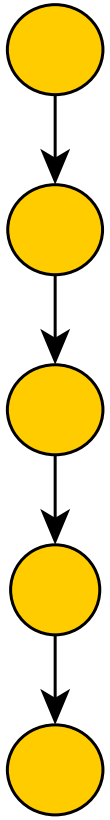
Tampereen yliopisto

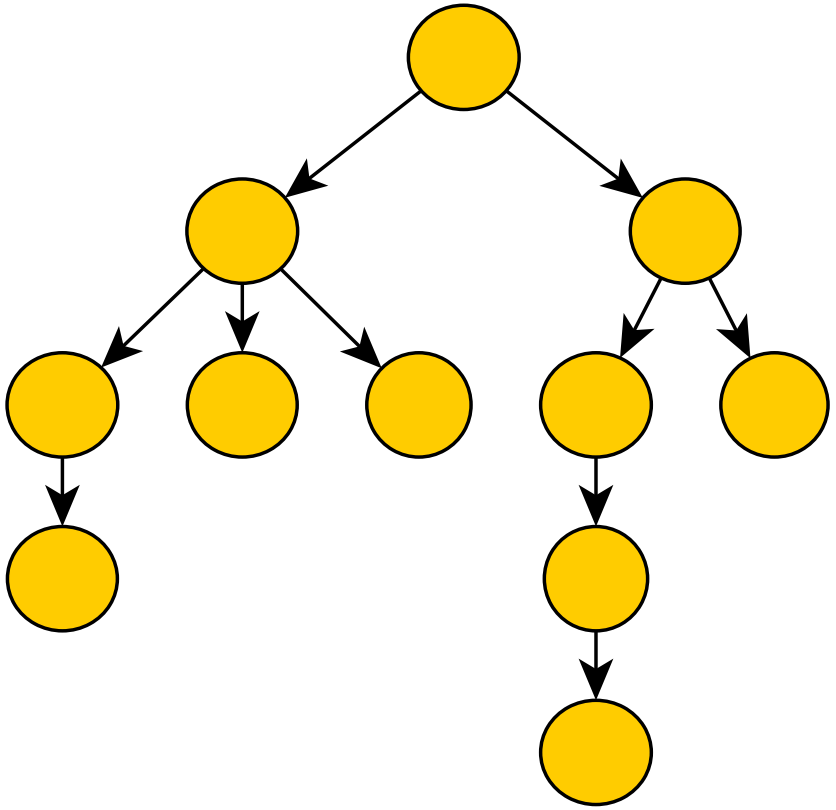
Lista-puu-graafi

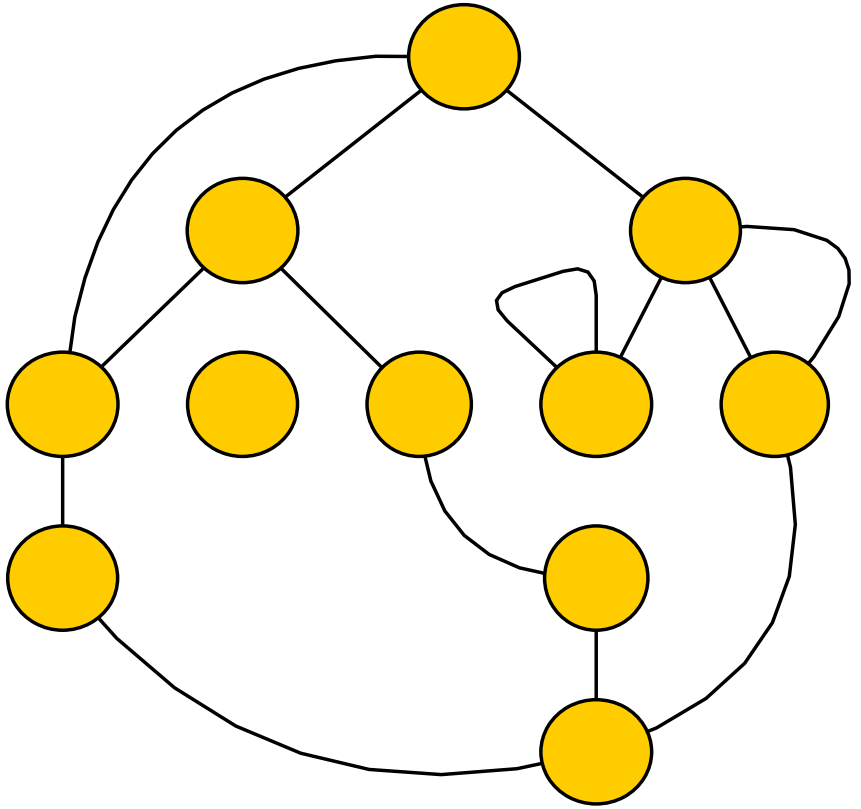


Tampereen yliopisto

Lista-puu-graafi





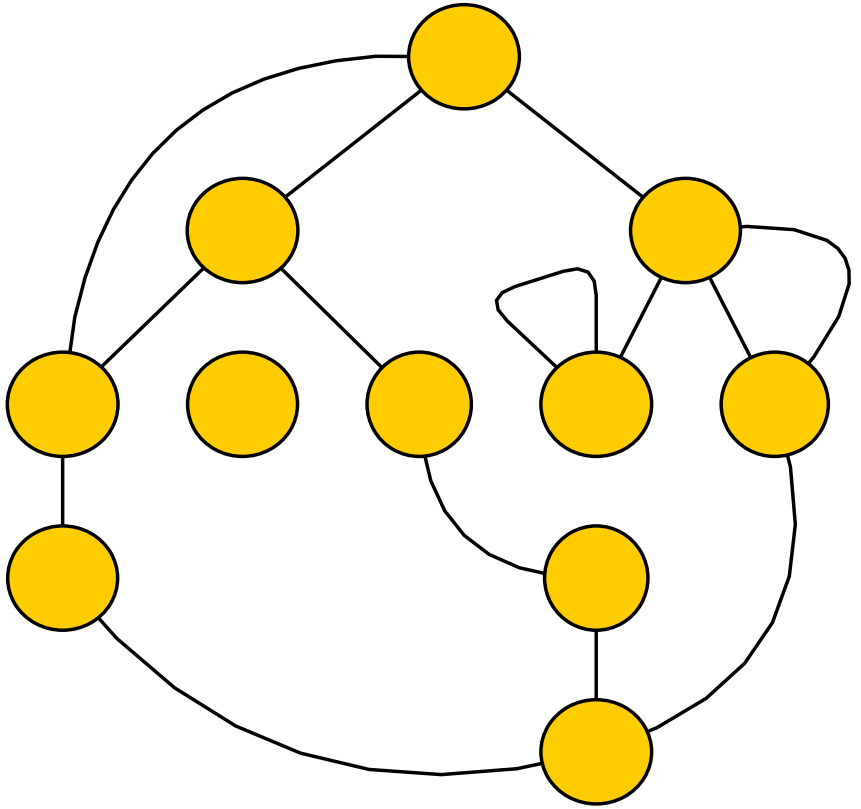


Termejä

- Solmut (node, vertex)
  - Sisältävät usein dataa
  - "Samanarvoisia"
- Kaaret (edge, arc)
  - Voivat myös sisältää dataa
  - Yhteys solmusta toiseen
- Suunnattu / suuntaamaton
- Monigraafi (multigraph)
- Kytketty/yhtenäinen graafi

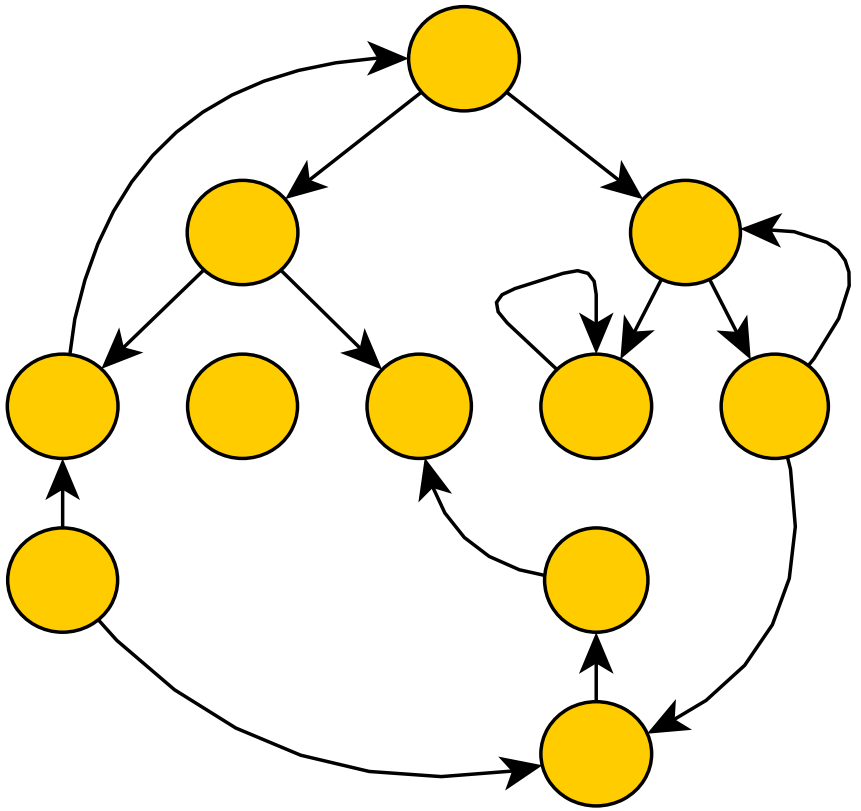
Termejä

- Solmut (node, vertex)
  - Sisältävät usein dataa
  - "Samanarvoisia"
- Kaaret (edge, arc)
  - Voivat myös sisältää dataa
  - Yhteys solmusta toiseen
- Suunnattu / suuntaamaton
- Monigraafi (multigraph)
- Kytketty/yhtenäinen graafi



Termejä

- Solmut (node, vertex)
  - Sisältävät usein dataa
  - "Samanarvoisia"
- Kaaret (edge, arc)
  - Voivat myös sisältää dataa
  - Yhteys solmusta toiseen
- Suunnattu / suuntaamaton
- Monigraafi (multigraph)
- Kytketty/yhtenäinen graafi



Tampereen yliopisto

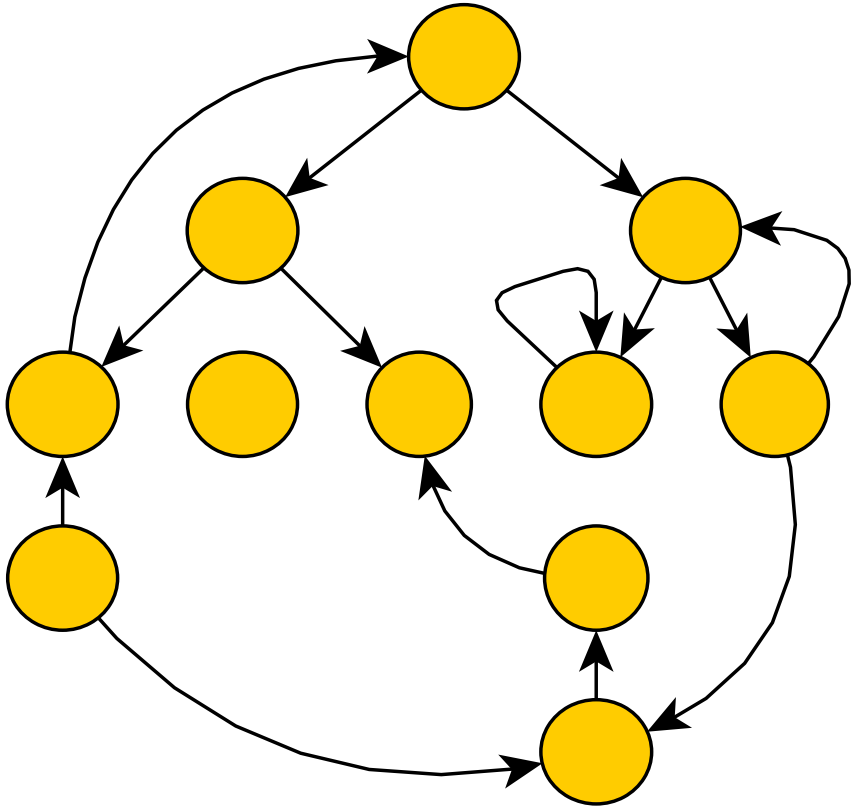
Graafihaut: Leveys-ensin-haku (BFS)

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

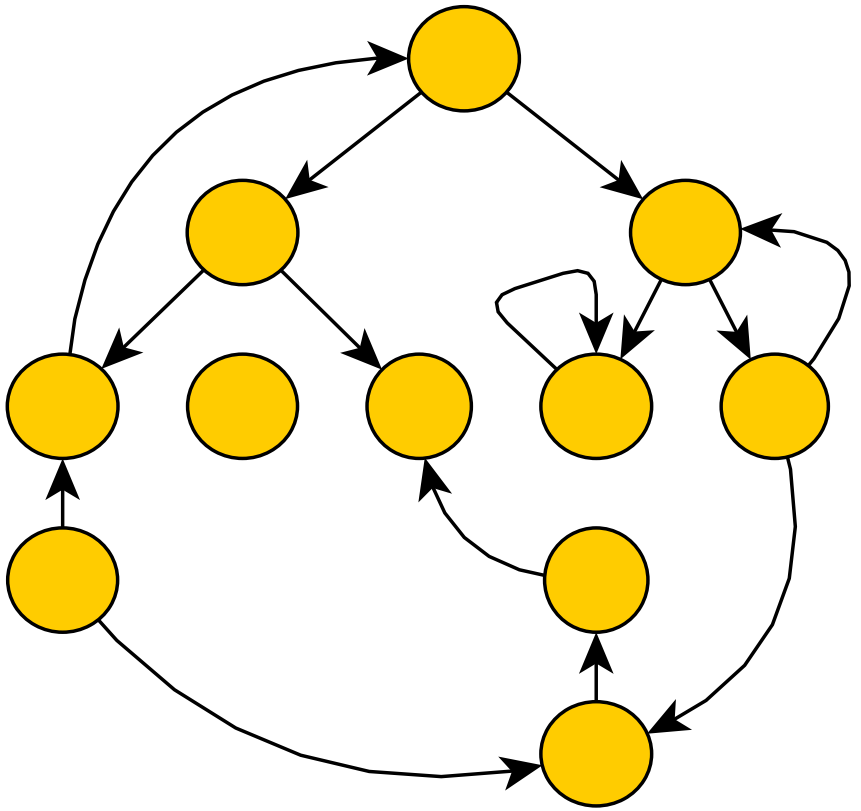
Graafialgoritmeista

- Askel
- Reitti
- Solmujen etäisyys δ(v1,v2)
- Solmun "väri" (vain apukeino)
  - Valkea: alkutila
  - Harmaa: löydetty, "kesken"
  - Musta: löydetty, loppuun käsitelty
- Aputietorakenne kesken oleville (harmaille) solmuille, "muistilista"



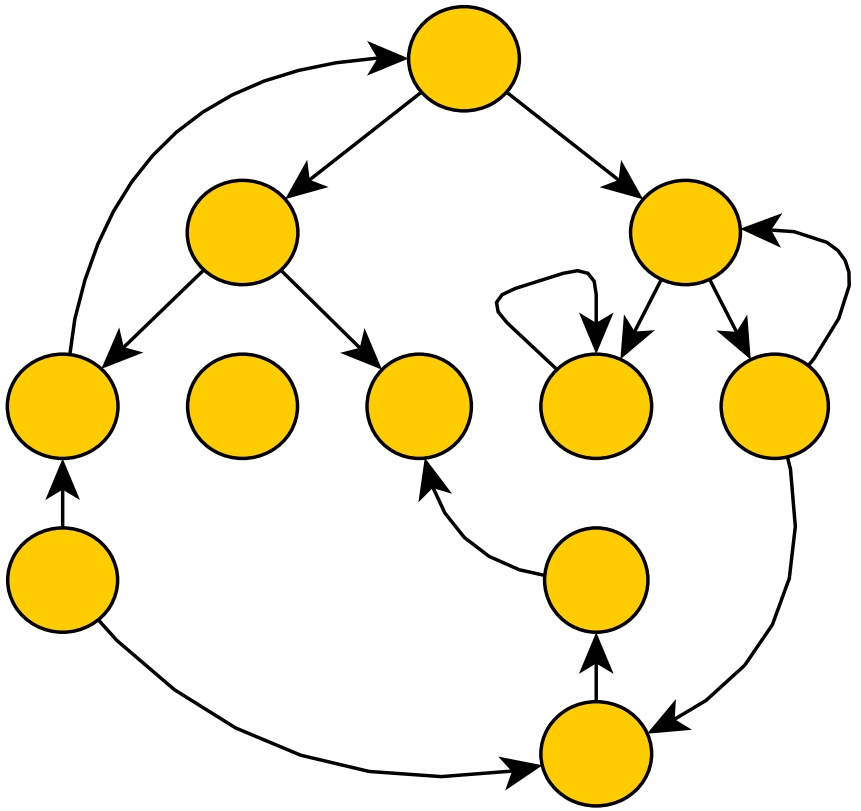
Graafialgoritmeista

- Algoritmit liittävät joka solmuun *v* lisädataa
  - *v*⟶*d:* etäisyys lähtösolmusta solmuun (∞ jos solmua ei (vielä) löydetty)
  - *v*⟶*π:* mistä solmusta saavuttiin tähän solmuun
  - *v*⟶*colour:* solmun väri
- Lisäksi merkitään
  - *v*⟶*Adj: v*:n naapurisolmut (mihin solmuihin *v*:stä lähtee kaaret)



Leveys-ensin-haku (BFS)

- Etsii kaikki solmut, joihin *pääsee* annetusta lähtösolmusta
- Laskee em. solmujen *etäisyydet* lähtösolmusta
- Laskee *lyhimmän polun* lähtösolmusta joka em. solmuun
- *Jono (queue) h*armaiden solmujen muistilistana
- (lisäksi tarvitaan solmujen lisäkentät)



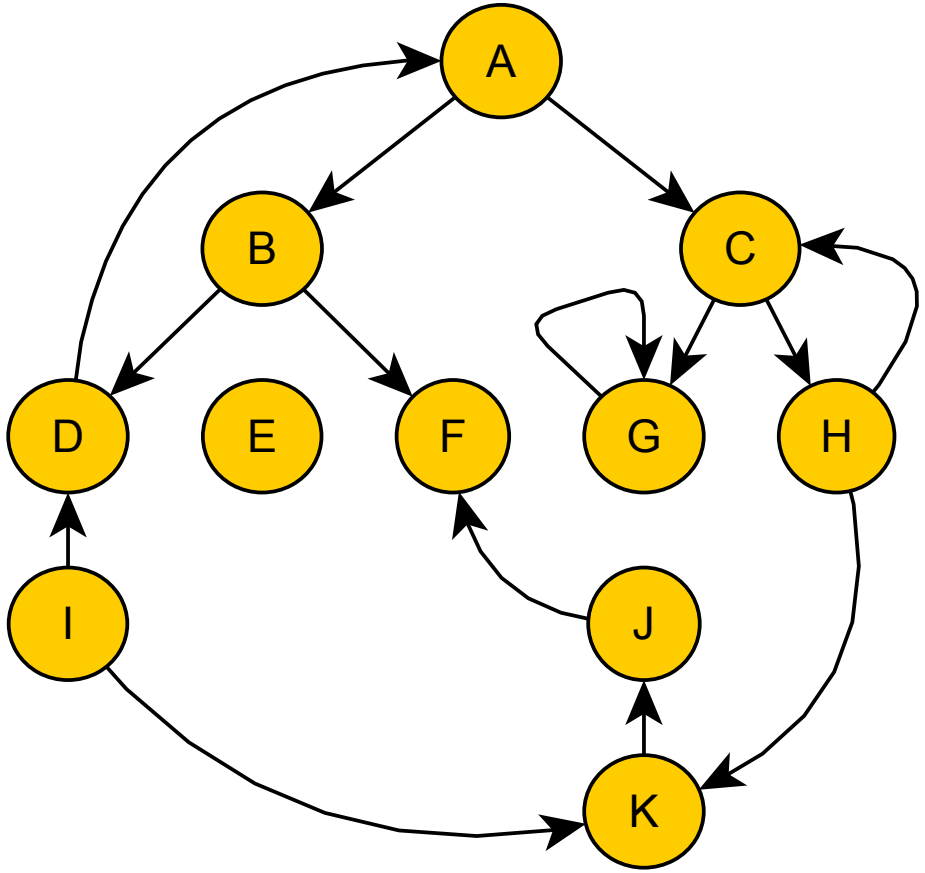
Leveys-ensin-haku (BFS)

```
Breath-first-search(s)                    (s on haun lähtösolmu)
1 ▷ (Kaikki solmut: colour := white, d := ∞, π := NIL)
2 ▷ (Q on muistilistana toimiva jonotietorakenne)
3 s→colour := gray                        (alkusolmu on kesken)
4 s→d := 0                                (etäisyys alkusolmuun 0)
5 Push(Q, s)                              (alkusolmu muistilistalle)
6 while Q ≠ empty do                      (kunnes muistilista on tyhjä)
7     u := Pop(Q)                         (vanhin alkio listasta)
8     for v in u→Adj do                   (käy läpi solmun naapurit)
9         if v→colour = white then        (jos uusi solmu...)
10            v→colour := gray            (...se on nyt kesken)
11            v→d := u→d + 1              (...ja yhtä kauempana)
12            v→π := u                    (...ja siihen tultiin u:sta)
13            Push(Q, v)                  (Lisätään v listalle)
14    u→colour := black                   (Nyt s on loppuun käsitelty)
```

Leveys-ensin-haku (BFS)

Breath-first-search(*s*)

```
1  ▷ (Kaikki solmut: ...)
2  ▷ (Q on ...)
3  s→colour := gray
4  s→d := 0
5  Push(Q, s)
6  while Q ≠ empty do
7      u := Pop(Q)
8      for v in u→Adj do
9          if v→colour = white then
10             v→colour := gray
11             v→d := u→d + 1
12             v→π := u
13             Push(Q, v)
14     u→colour := black
```



Tampereen yliopisto

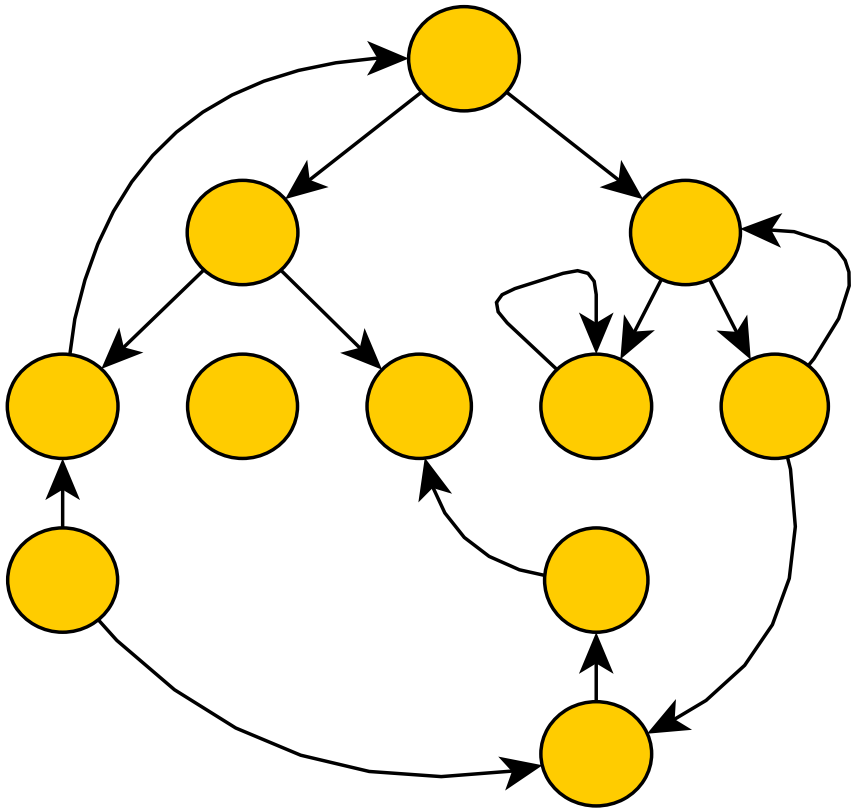
Graafihaut: Syvyys-ensin-haku (DFS)

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Syvyys-ensin-haku (DFS)

- Etsii myös kaikki solmut, joihin *pääsee* annetusta lähtösolmusta
- Laskee *jonkin polun* lähtösolmusta joka em. solmuun
- Selvittää, löytyykö poluista *silmukoita*
- *Pino (stack) h*armaiden solmujen muistilistana
- (muistinkulutus usein BFS:ää pienempi)

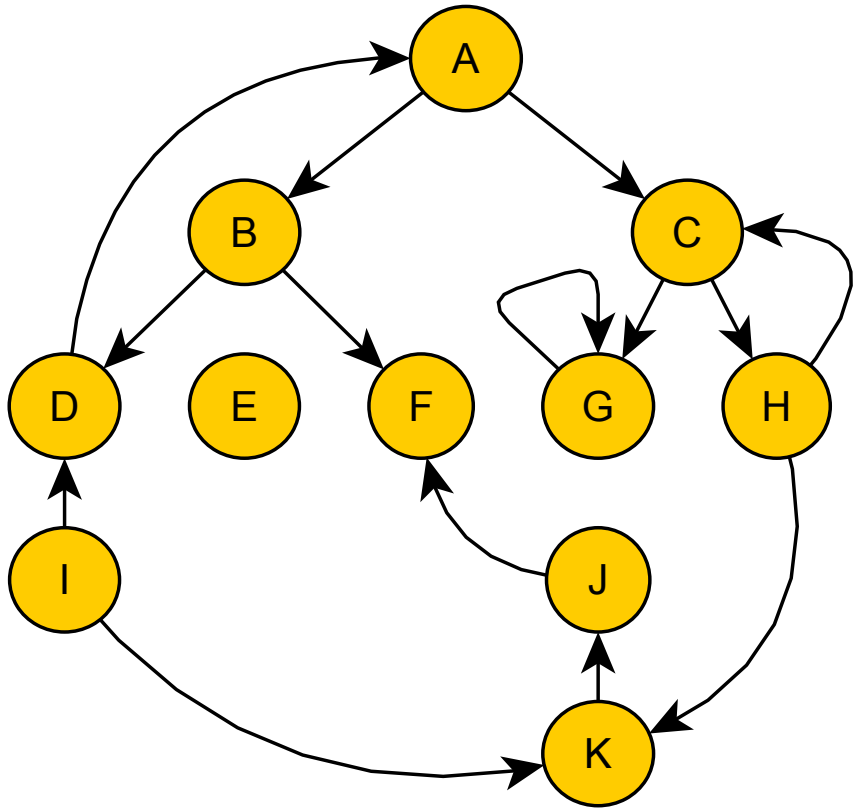


Syvyys-ensin-haku (DFS)

```
Depth-first-search(s)                    (s on haun lähtösolmu)
1 ▷ (Kaikki solmut: colour := white)
2 ▷ (S on muistilistana toimiva pinotietorakenne)
3 Push(S, s)                             (alkusolmu muistilistalle (valkoisena!))
4 while S ≠ empty do                     (kunnes muistilista on tyhjä)
5     u := Pop(S)                        (uusin alkio listasta)
6     if u→colour = white then           (jos kyse on uudesta alkiosta...)
7         u→colour = gray                (...niin se on nyt kesken...)
8         Push(S, u)                     (...ja laitetaan uudelleen listalle)
9         for v in u→Adj do              (käy läpi solmun naapurit)
10            if v→colour = white then   (jos uusi solmu...)
11                Push(S, v)             (...lisätään se listalle)
12            else if v→colour = gray then (päästiin takaisin, silmukka!)
13                ▷ (Käsittele silmukka)
14    else
15        u→colour := black              (Solmu tuli uudelleen, nyt valmis)
```

Syvyys-ensin-haku (DFS)

```
Depth-first-search(s)
1  ▷ (Kaikki solmut: colour := white)
2  ▷ (S on muistilistana toimiva pinotietorakenne)
3  Push(S, s)
4  while S ≠ empty do
5      u := Pop(S)
6      if u→colour = white then
7          u→colour = gray
8          Push(S, u)
9          for v in u→Adj do
10             if v→colour = white then
11                 Push(S, v)
12             else if v→colour = gray then
13                 ▷ (Käsittele silmukka)
14     else
15         u→colour := black
```



Rekursiivinen (DFS)

- Ihan aluksi kaikki solmut: *colour* := white
- Muistilista-pinona toimii funktioiden kutsupino!

```
Depth-first-search(s)                    (s on haun lähtösolmu)
1 s→colour := gray
2 for v in s→Adj do                      (käy läpi solmun naapurit)
3     if v→colour = white then           (jos uusi solmu...)
4             Depth-first-search(v)      (...käy se läpi rekursiivisesti)
5     else if v→colour = gray then       (päästiin takaisin, silmukka!)
6             ▷ (Käsittele silmukka)
7 s→colour := black                      (Aivan lopuksi väri mustaksi)
```

Tampereen yliopisto

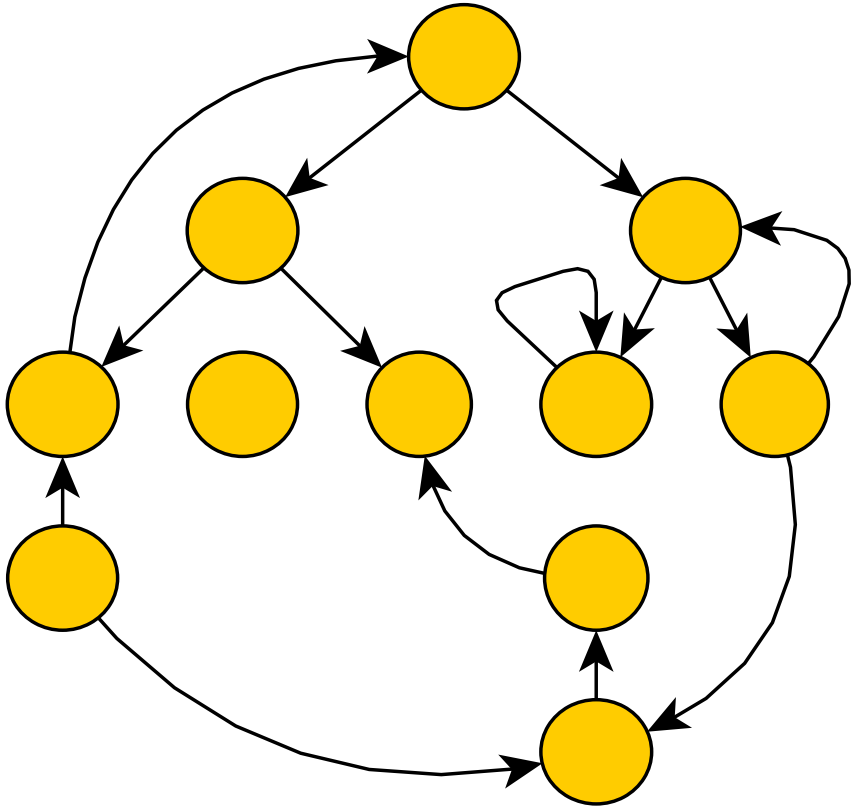
Graafien toteuttaminen

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Tampereen yliopisto

Graafien toteuttaminen



Tampereen yliopisto

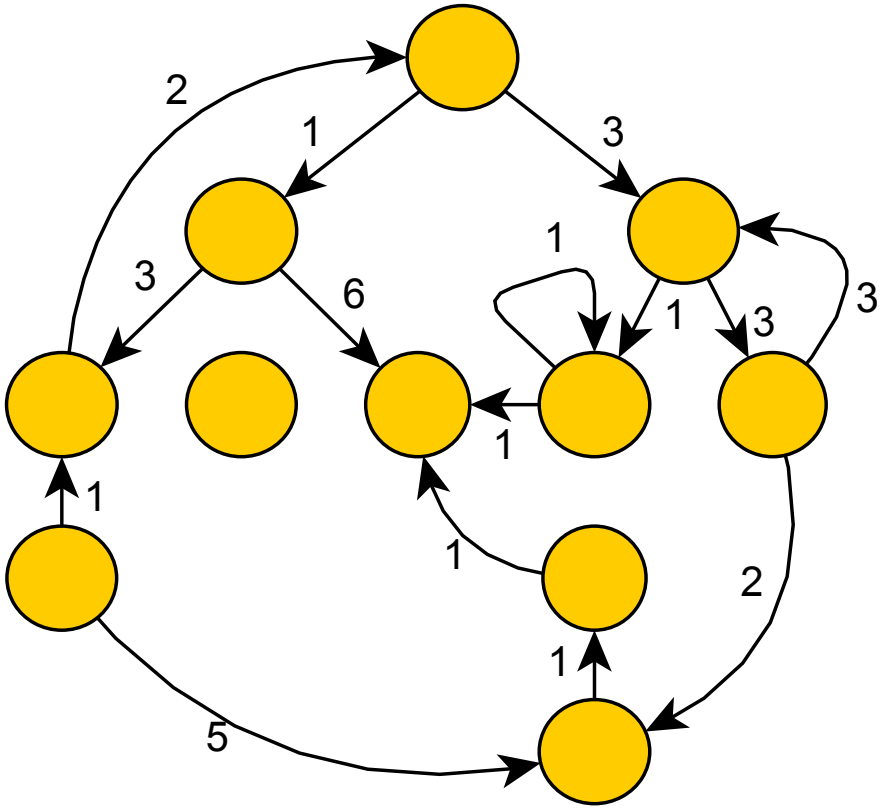
Painotetut graafit ja niiden toteuttaminen

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Painotetut graafit

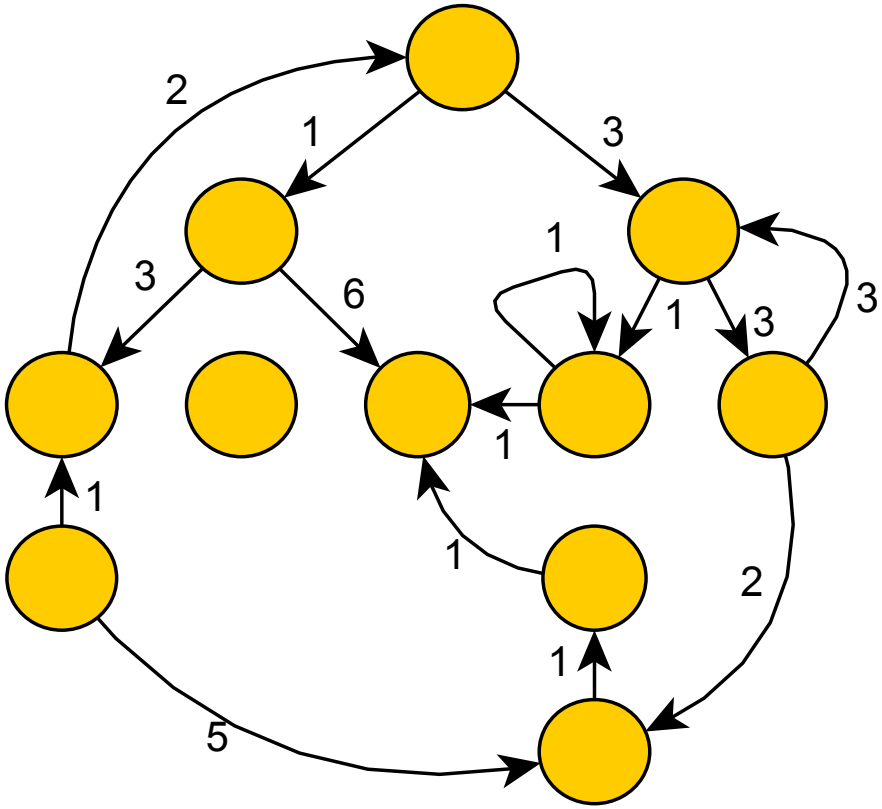
- *Paino/kustannus* joka kaaressa
- Etsitään *keveimpiä/halvimpia* polkuje solmujen välillä
- Leveys-ensin ei enää kelpaa! (siinä kaaren "hinta" on aina 1)
- Halvin reitti ≠ lyhin reitti (ehkä)
- (Kaaret voivat sisältää myös muuta dataa)
- Usein rajoitetaan paino/kustannus positiiviseksi



Tampereen yliopisto

Painotetun graafin toteuttaminen

- Kaaren tiedoksi solmussa ei riitä enää vain sen päätepiste



Tampereen yliopisto

Halvimmat reitit: Dijkstran algoritmi

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Tampereen yliopisto

Dijkstran algoritmi

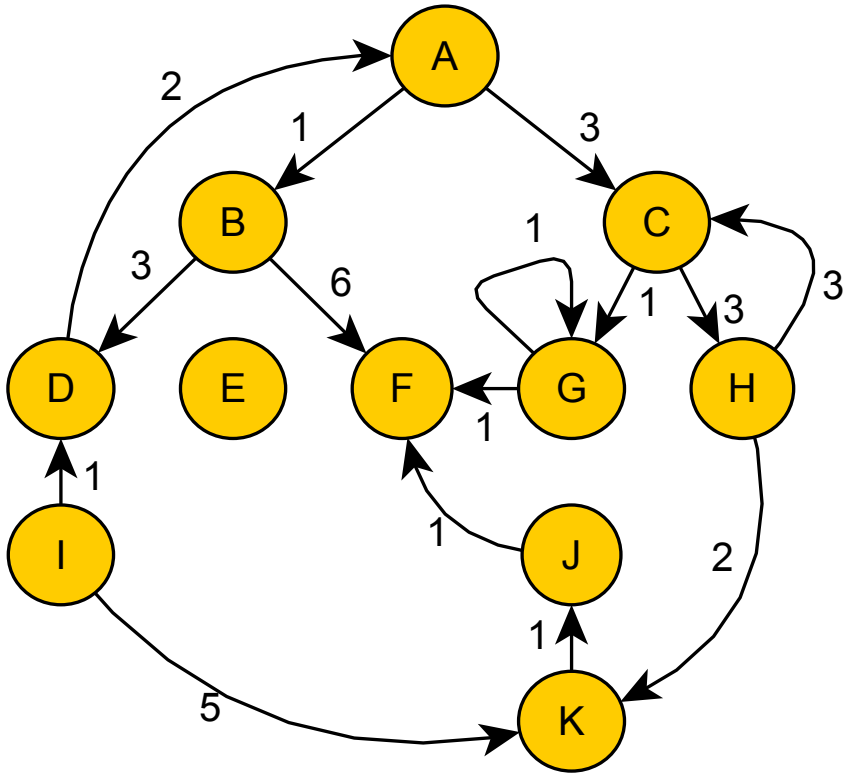
```
Dijkstra(s)                                   (s on haun lähtösolmu)
1  ▷ (Kaikki solmut: colour := white, d := ∞, π := NIL)
2  ▷ (Q on muistilistana toimiva prioriteettijono-tietorakenne)
3  s→colour := gray                           (alkusolmu on kesken)
4  s→d := 0                                   (etäisyys alkusolmuun 0)
5  Push(Q, s)                                 (alkusolmu muistilistalle)
6  while Q ≠ empty do                         (kunnes muistilista on tyhjä)
7     u := Extract-min(Q)                     (halvin alkio listasta)
8     for v in u→Adj do                       (käy läpi solmun naapurit)
9        Relax(u, v)                          (laske (tai korjaa) hinta)
10       if v→colour = white then             (jos uusi solmu...)
11          v→colour := gray                  (...se on nyt kesken)
12          Push(Q, v, v→d)                   (...ja laitetaan jonoon)
13       else
14          ▷ Jos hinta pieneni, korjaa prioriteetti Q:ssa!
15    u→colour := black                        (Nyt s on loppuun käsitelty)

16 Relax(u, v)                                (Laske/korjaa hinta u:n kautta solmuun v)
17 if v→d > u→d + cost(u, v) then             (Jos hinta v:hen u:n kautta on entistä halvempi)
18    v→d := u→d + cost(u, v)                 (...päivitä hinta)
19    v→π := u                                (...ja se, että tultiin u:sta)
```

Dijkstran algoritmi

```
Dijkstra(s)
 1 ▷ (Alusta kaikki solmut)
 2 ▷ (Q on prioriteettijono)
 3 s→colour := gray
 4 s→d := 0
 5 Push(Q, s)
 6 while Q ≠ empty do
 7     u := Extract-min(Q)
 8     for v in u→Adj do
 9         Relax(u, v)
10         if v→colour = white then
11             v→colour := gray
12             Push(Q, v, v→d)
13         else
14             ▷ Jos hinta pieneni, korjaa prioriteetti Q:ssa!
15     u→colour := black

16 Relax(u, v)
17 if v→d > u→d + cost(u, v) then
18     v→d := u→d + cost(u, v)
19     v→π := u
```



Prioriteettijono & Dijkstra

- Dijkstran toteutus edellyttää *minimi*-prioriteettijonon
  - `std::priority_queue` on *maksimi*-prioriteettijono
    - Tallenna prioriteetiksi -d!
    - (Tai mukauta prioriteettijonoa)
- Prioriteettia pystyttävä päivittämään
  - `std::priority_queue` ei tarjoa tätä
    - Lisää solmu aina uudelleen (prioriteetti aina parempi), jätä myöhemmät kopiot huomiotta
  - Tai käytä jotain muuta prioriteettijonona (esim. `std::set<std::pair<int, Node*>>`)

Tampereen yliopisto

Halvimmat reitit: A* ja heuristiikat

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Tampereen yliopisto

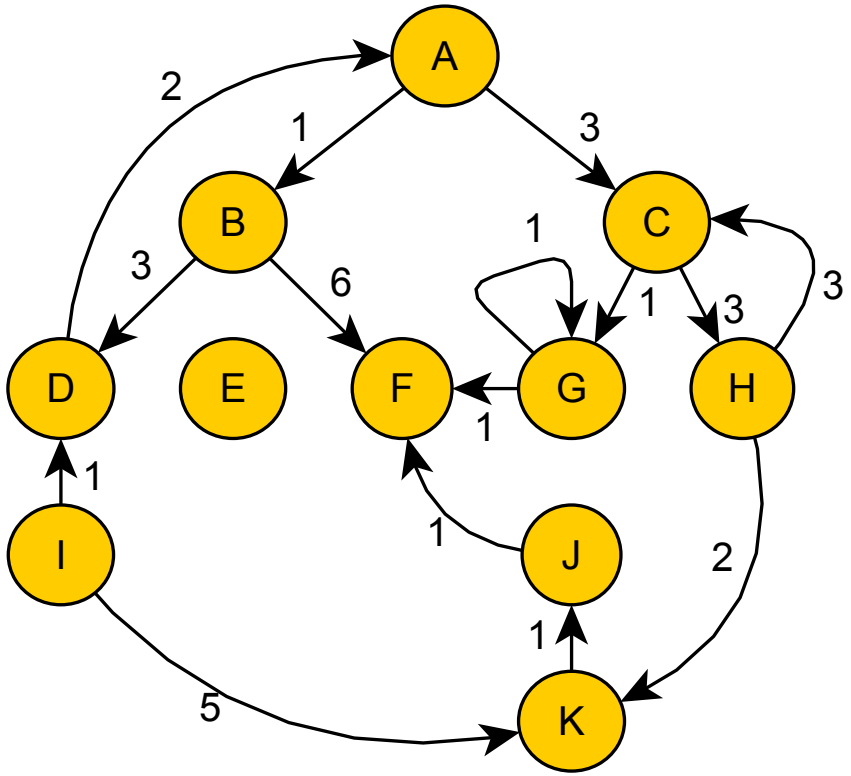
A*-algoritmi

```
A*(s, g)                                   (s on lähtösolmu, g maali)
 1 ▷ (Kaikki solmut: colour := white, d := ∞, de := ∞, π := NIL)
 2 ▷ (Q on muistilistana toimiva prioriteettijono-tietorakenne)
 3 s→colour := gray                        (alkusolmu on kesken)
 4 s→d := 0                                (etäisyys alkusolmuun 0)
 5 Push(Q, s)                              (alkusolmu muistilistalle)
 6 while Q ≠ empty do                      (kunnes muistilista on tyhjä)
 7    u := Extract-min(Q)                  (halvin alkio listasta)
 8    if u = g then  ▷ Maali löytynyt, lopeta!
 9    for v in u→Adj do                    (käy läpi solmun naapurit)
10       Relax-A*(u, v, g)                 (laske (tai korjaa) hinta)
11       if v→colour = white then          (jos uusi solmu...)
12          v→colour := gray               (...se on nyt kesken)
13          Push(Q, v, v→de)               (...ja laitetaan jonoon)
14       else
15          ▷ Jos hinta pieneni, korjaa prioriteetti Q:ssa!
16    u→colour := black                    (Nyt s on loppuun käsitelty)
17 Relax-A*(u, v, g)                      (Laske/korjaa hinta+arvio u:n kautta)
18 if v→d > u→d + cost(u, v) then         (Jos hinta v:hen u:n kautta on entistä halvempi)
19    v→d := u→d + cost(u, v)             (...päivitä hinta)
20    v→de := v→d + min-est(v, g)         (...päivitä minimiarvio kokonaishinnasta)
21    v→π := u                            (...ja se, että tultiin u:sta)
```

Tampereen yliopisto

A*-algoritmi

```
A*(s, g)
 1 ▷ (Alusta kaikki solmut)
 2 ▷ (Q on prioriteettijono)
 3 s→colour := gray
 4 s→d := 0
 5 Push(Q, s)
 6 while Q ≠ empty do
 7     u := Extract-min(Q)
 8     if u = g then  ▷ Maali löytynyt, lopeta!
 9     for v in u→Adj do
10         Relax-A*(u, v, g)
11         if v→colour = white then
12             v→colour := gray
13             Push(Q, v, v→de)
14         else
15             ▷ Jos hinta pieneni, korjaa prioriteetti Q:ssa!
16     u→colour := black
17 Relax-A*(u, v, g)
18 if v→d > u→d + cost(u, v) then
19     v→d := u→d + cost(u, v)
20     v→de := v→d + min-est(v, g)
21     v→π := u
```



Tampereen yliopisto

Graafihakujen tehokkuus ja yhteenveto

COMP.CS.300 Tietorakenteet ja algoritmit 1

Matti Rintala (matti.rintala@tuni.fi)

Leveys-ensin-haku (BFS)

```
Breath-first-search(s)                    (s on haun lähtösolmu)
1 ▷ (Kaikki solmut: colour := white, d := ∞, π := NIL)
2 ▷ (Q on muistilistana toimiva jonotietorakenne)
3 s→colour := gray                        (alkusolmu on kesken)
4 s→d := 0                                (etäisyys alkusolmuun 0)
5 Push(Q, s)                              (alkusolmu muistilistalle)
6 while Q ≠ empty do                      (kunnes muistilista on tyhjä)
7     u := Pop(Q)                         (vanhin alkio listasta)
8     for v in u→Adj do                   (käy läpi solmun naapurit)
9         if v→colour = white then        (jos uusi solmu...)
10            v→colour := gray            (...se on nyt kesken)
11            v→d := u→d + 1              (...ja yhtä kauempana)
12            v→π := u                    (...ja siihen tultiin u:sta)
13            Push(Q, v)                  (Lisätään v listalle)
14    u→colour := black                   (Nyt s on loppuun käsitelty)
```

Syvyys-ensin-haku (DFS)

```
Depth-first-search(s)                    (s on haun lähtösolmu)
1 ▷ (Kaikki solmut: colour := white)
2 ▷ (S on muistilistana toimiva pinotietorakenne)
3 Push(S, s)                             (alkusolmu muistilistalle (valkoisena!))
4 while S ≠ empty do                     (kunnes muistilista on tyhjä)
5     u := Pop(S)                        (uusin alkio listasta)
6     if u→colour = white then           (jos kyse on uudesta alkiosta...)
7         u→colour = gray                (...niin se on nyt kesken...)
8         Push(S, u)                     (...ja laitetaan uudelleen listalle)
9         for v in u→Adj do              (käy läpi solmun naapurit)
10            if v→colour = white then   (jos uusi solmu...)
11                Push(S, v)             (...lisätään se listalle)
12            else if v→colour = gray then (päästiin takaisin, silmukka!)
13                ▷ (Käsittele silmukka)
14    else
15        u→colour := black              (Solmu tuli uudelleen, nyt valmis)
```

Tampereen yliopisto

Dijkstran algoritmi

```
Dijkstra(s)                                   (s on haun lähtösolmu)
1  ▷ (Kaikki solmut: colour := white, d := ∞, π := NIL)
2  ▷ (Q on muistilistana toimiva prioriteettijono-tietorakenne)
3  s→colour := gray                           (alkusolmu on kesken)
4  s→d := 0                                   (etäisyys alkusolmuun 0)
5  Push(Q, s)                                 (alkusolmu muistilistalle)
6  while Q ≠ empty do                         (kunnes muistilista on tyhjä)
7     u := Extract-min(Q)                     (halvin alkio listasta)
8     for v in u→Adj do                       (käy läpi solmun naapurit)
9        Relax(u, v)                          (laske (tai korjaa) hinta)
10       if v→colour = white then             (jos uusi solmu...)
11          v→colour := gray                  (...se on nyt kesken)
12          Push(Q, v, v→d)                   (...ja laitetaan jonoon)
13       else
14          ▷ Jos hinta pieneni, korjaa prioriteetti Q:ssa!
15    u→colour := black                        (Nyt s on loppuun käsitelty)

16 Relax(u, v)                                (Laske/korjaa hinta u:n kautta solmuun v)
17 if v→d > u→d + cost(u, v) then             (Jos hinta v:hen u:n kautta on entistä halvempi)
18    v→d := u→d + cost(u, v)                 (...päivitä hinta)
19    v→π := u                                (...ja se, että tultiin u:sta)
```

Tampereen yliopisto

A*-algoritmi

```
A*(s, g)                                   (s on lähtösolmu, g maali)
 1 ▷ (Kaikki solmut: colour := white, d := ∞, de := ∞, π := NIL)
 2 ▷ (Q on muistilistana toimiva prioriteettijono-tietorakenne)
 3 s→colour := gray                        (alkusolmu on kesken)
 4 s→d := 0                                (etäisyys alkusolmuun 0)
 5 Push(Q, s)                              (alkusolmu muistilistalle)
 6 while Q ≠ empty do                      (kunnes muistilista on tyhjä)
 7     u := Extract-min(Q)                 (halvin alkio listasta)
 8     for v in u→Adj do                   (käy läpi solmun naapurit)
 9       Relax-A*(u, v, g)                 (laske (tai korjaa) hinta)
10       if v→colour = white then          (jos uusi solmu...)
11           v→colour := gray              (...se on nyt kesken)
12           Push(Q, v, v→de)              (...ja laitetaan jonoon)
13       else
14           ▷ Jos hinta pieneni, korjaa prioriteetti Q:ssa!
15     u→colour := black                   (Nyt s on loppuun käsitelty)

16 Relax-A*(u, v, g)                       (Laske/korjaa hinta+arvio u:n kautta)
17 if v→d > u→d + cost(u, v) then          (Jos hinta v:hen u:n kautta on entistä halvempi)
18     v→d := u→d + cost(u, v)             (...päivitä hinta)
19     v→de := v→d + min-est(v, g)         (...päivitä minimiarvio kokonaishinnasta)
20     v→π := u                            (...ja se, että tultiin u:sta)
```

Algoritmi(*par*)

```
1 koodia                    (kommentti)
2 koodia
```

Tampereen yliopisto

(kommentti)

Tampereen yliopisto