$NL = coNL$

Relazioni tra le classi

L'affermazione NL=coNL si può generalizzare dimostrando che per ogni s(n) ≥ log n

$$\text{NSPACE}(s(n)) = \text{coNSPACE}(s(n))$$

Come Neil Immerman e Róbert Szelepcsényi hanno dimostrato indipendentemente nel 1987.

Cosa che ha loro fruttato il premio Gödel dell' ACM SIGACT (Association for Computing Machinery Special Interest Group on Algorithms and Computation Theory) e dell' EATCS (European Association for Theoretical Computer Science) nel 1995.

Relazioni tra le classi

Sappiamo che
1. $L \subseteq NL = coNL \subseteq P \subseteq PSPACE$

Si può dimostrare che
2. $NL \subsetneq PSPACE$

quindi $NL \subsetneq P$ o $P \subsetneq PSPACE$

Prova che NL = coNL

Si prova facendo vedere che ¬PATH è in NL:

sappiamo che per ogni B in NL

$$B \leq_L PATH \Rightarrow \neg B \leq_L \neg PATH,$$

se ¬PATH è in NL si può costruire una TM che decide ¬B in spazio logaritmico e quindi

$$coNL \subseteq NL$$

d'altro canto se ¬PATH è in NL per definizione PATH è in coNL e quindi

$$NL \subseteq coNL$$

PATH e notPATH

PATH = {<G,s,t> | G è un grafo diretto con un cammino da s a t}

Quindi ¬PATH={<G,s,t> | G è un grafo diretto in cui non c’è un cammino da s a t}

¬PATH è in NL.

Si tratta di costruire una NTM per ¬PATH = {<G,s,t> | G è un grafo diretto in cui non c'è un cammino da s a t} con complessità di spazio logaritmico

Supponiamo in un primo momento di avere in input anche il numero, r , dei vertici raggiungibili da s.
Allora calcolando il numero dei vertici <u>diversi da t</u> e raggiungibili da s, possiamo concludere che se i due numeri sono uguali t non è raggiungibile da s.
Possiamo fare questo esaminando tutti i sottoinsiemi di r vertici e accettando se si trova un sottoinsieme che contiene vertici raggiungibili da s ma che non contiene t e rifiutando altrimenti.
Consideriamo una TM ausiliaria CHECKPATH(G;s;v; k) che accetta se c'è un cammino di lunghezza al più k da s a v, altrimenti rifiuta.

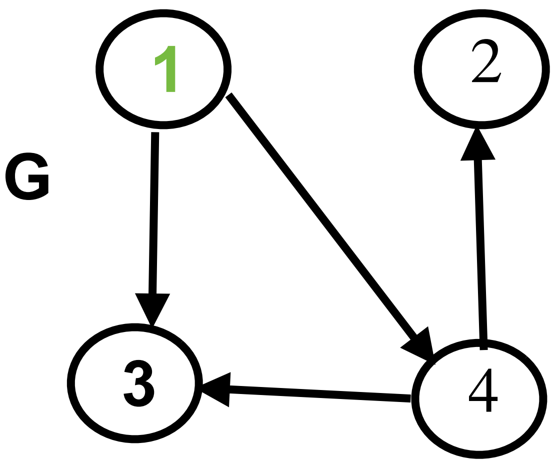
La NTM CHECKPATH

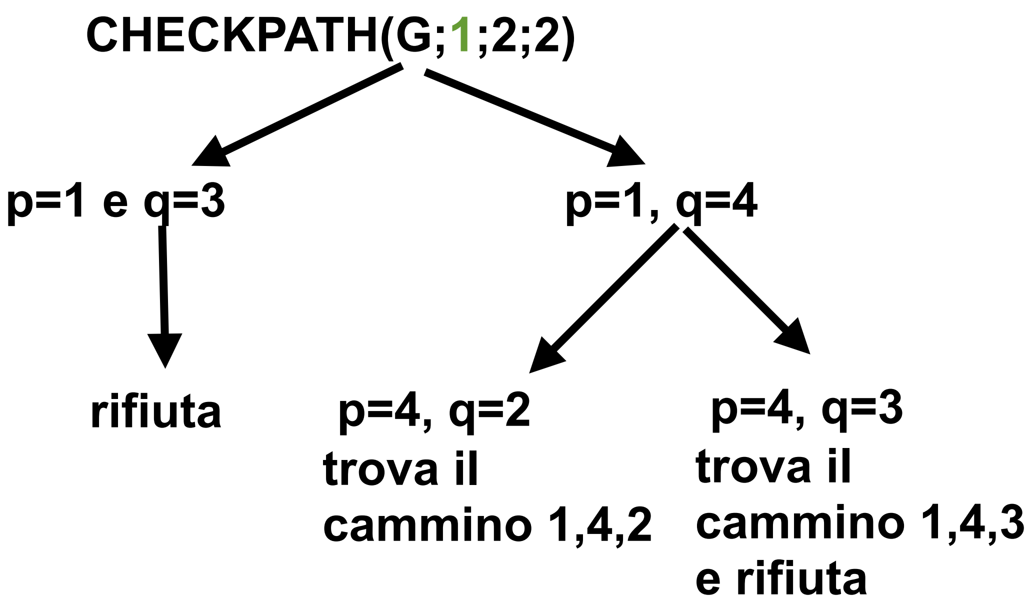
```
CHECKPATH(G;s;v; k)
 % accetta se c'è un cammino di lunghezza al più k
 da s a v, altrimenti rifiuta%
  if v = s then accetta return
  p ← s
  for i ← 1 to k do % costruisce non deterministicamente
 un cammino da s a v di lunghezza al più k %
          if ci sono vertici q tali che  (p,q) è in E then
          scegline uno non deterministicamente
          p←q
          else %se non ce ne sono%  q←p
   if p ≠ v then rifiuta % il  cammino non porta a v %
```

Lo spazio necessario è quello per i valori delle variabili v, p, k,q i cui valori sono al più |V| e quindi rappresentabili in log (|V|)

Esempio di esecuzione di CHECKPATH(G;s;v; k)

```
 CHECKPATH(G;s;v; k)
% accetta se c'è un cammino di lunghezza al più k
da s a v, altrimenti rifiuta%
 if v = s then accetta return
 p ← s
 for i ← 1 to k do % costruisce non deterministicamente
un cammino da s a v di lunghezza al più k %
        if ci sono vertici q tali che  (p,q) è in E then
        scegline uno non deterministicamente
        p←q
        else %se non ce ne sono%  q←p
  if p ≠ v then rifiuta % il  cammino non porta a v %
```





L'algoritmo per ¬PATH

NnotPATH:
input: G = (V,E),s,t,r
% r≥0 è il numero dei vertici raggiungibili da s %
output: sì, se scopre che t non è raggiungibile da s, no altrimenti
c←0
for all v ∈ (V − {t}) do % conta in c il numero dei vertici v, diversi da t, raggiungibili da s%
non deterministicamente assegna a guess un valore in {sì,no}
%i vertici per cui guess=sì costuiscono un possibile insieme di vertici raggiungibili da s%
if guess = sì then
non deterministicamente scegli un valore per k
%k è la lunghezza del cammino tra s e v%
CHECKPATH(G;s;v;k).
c←c+1
if c = r then return sì, else return no

Lo spazio necessario è quello per i valori delle variabili c, guess, v, k tutti rappresentabili in log (|V|)

Calcolare il numero dei vertici raggiungibili da s.

Un algoritmo non deterministico calcola un valore c, se ogni cammino che non rifiuta calcola il corretto valore di c.

Ci serviremo ancora della NTM CHECKPATH(G; s;v; k) che accetta, in spazio logaritmico, se in G c'è un cammino di lunghezza al più k da s a v, altrimenti rifiuta

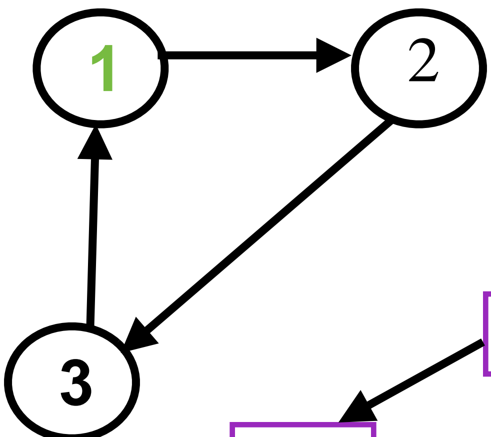
Calcolare r: primo tentativo

CalcolaR(G, s)
input: G = (V,E), s
output: il numero dei vertici raggiungibili da s, incluso s
c$\leftarrow$0
for all v $\in$ V do
  non deterministicamente assegna a guess un valore in {sì,no}
  if guess = sì then
    non deterministicamente scegli un valore per k
  %k è la lunghezza del cammino tra s e v%
    CHECKPATH(G;s;v;k).
    c$\leftarrow$c+1
return c

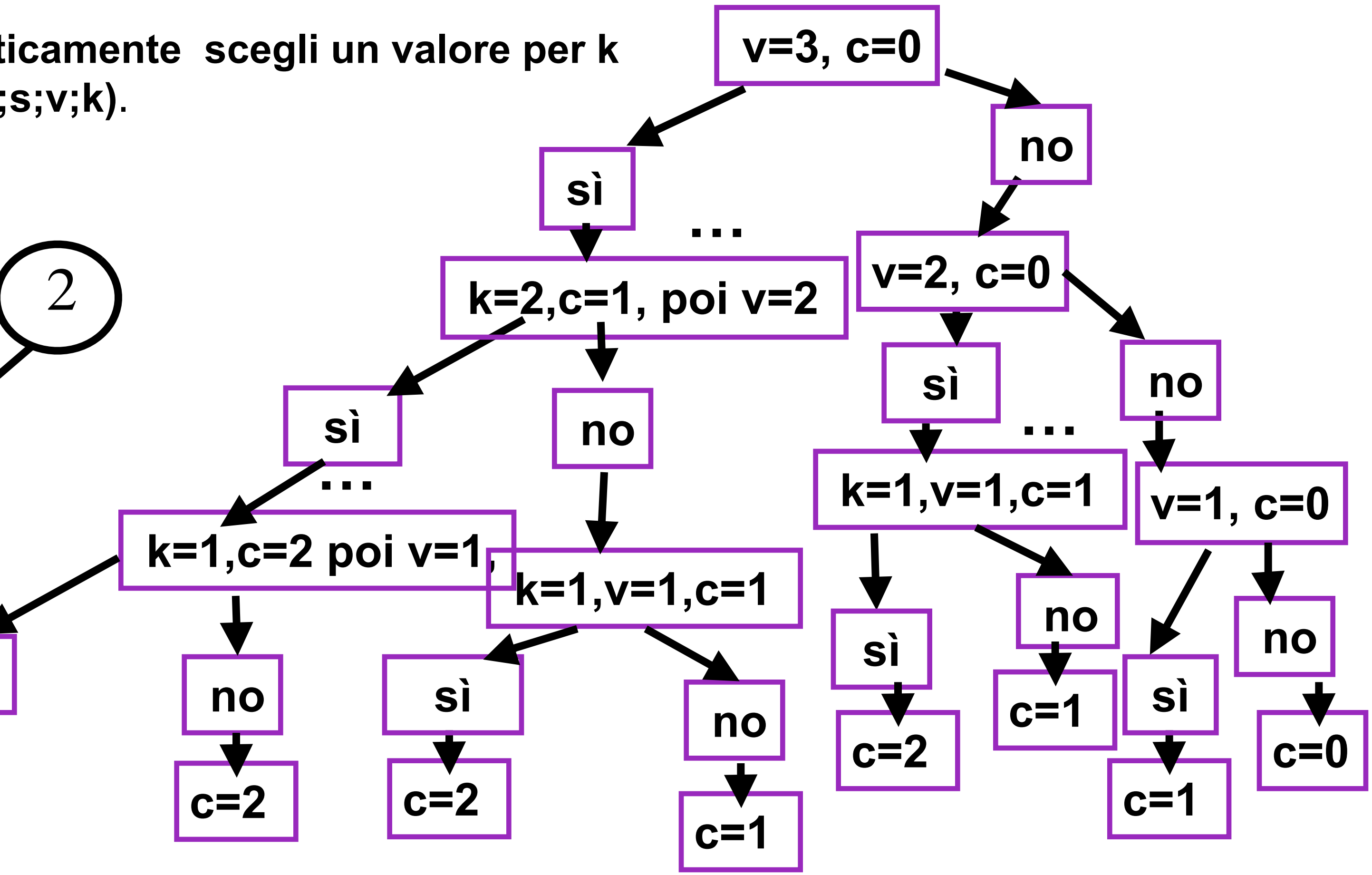
NON VA! Se CHECKPATH(G;s;v;k) rifiuta potrebbe restituire un valore minore di quello cercato.

Esempio di esecuzione di calcolaR

CalcolaR(G, s)
input: G = (V,E), s
output: il numero dei vertici raggiungibili da s, incluso s
c $\leftarrow$ 0
for all v $\in$ V do
  non deterministicamente assegna a guess un valore in {sì,no}
  if guess = sì then
    non deterministicamente scegli un valore per k
    CHECKPATH(G;s;v;k).
    c $\leftarrow$ c+1
return c



G



Calcolare r

Bisogna forzare il calcolo in modo da trovare sempre <u>tutti</u> i vertici raggiungibili da s o rifiutare.

Il valore di r sarà calcolato calcolando $r_k = |A_k|$ dove A_k è l'insieme dei vertici raggiungibili da s con un cammino lungo al più k.
Questi insiemi possono essere calcolati induttivamente:

$A_0 = \{s\}$

$A_k = A_{k-1} \cup \{u \mid (v,u)$ è un arco di G e v è in $A_{k-1}\}$

Quindi per capire se v è un vertice raggiungibile in k passi da s, si devono esaminare tutti i vertici w in A_{k-1} e verificare se c'è un arco (w,v) o che w=v

Il valore cercato r è r_{n-1}, dove $n = |V|$.

Calcolare r_k

Poichè ogni A_k può avere |V| elementi non possiamo tenerne memoria.

L'idea è calcolare r_k conoscendo r_{k-1} e usando questa conoscenza per scartare i cammini computazionali in cui non tutti i vertici raggiungibili in k-1 passi sono stati considerati.

In conclusione:

1. se si trova almeno un elemento w in A_{k-1} e (w,v) è in E o w=v allora v va in A_k.

2. se siamo sicuri di aver calcolato tutto A_{k-1} e (w,v) non è in E e v≠w per ogni w in A_{k-1}, allora possiamo concludere che v non è un elemento di A_k.

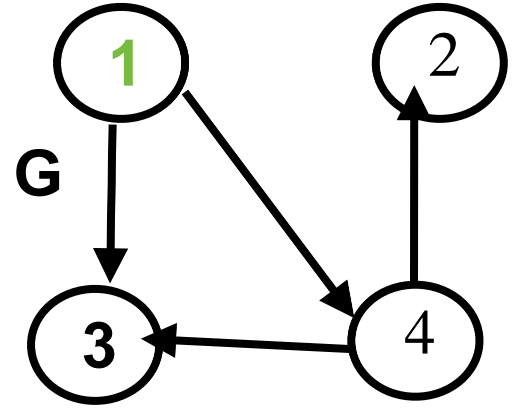
Si dovrà ripetere la procedura n-1 volte.

input: $G = (V,E)$, s, t, r_{k-1}
precondizione: r_{k-1} è il numero esatto di vertici raggiungibili da s in k-1 passi
output: il numero dei vertici raggiungibili da s, incluso s in al più k passi.

```
c←0
for all v ∈ V  do
        d ← 0; flag ←false % in d il numero trovato di vertici in A_{k-1}
        for all w ∈ V  do
            p ← s
            for i = 1 to k-1
                scegli non det q tale che (p,q) è in E,
                se non ce ne sono   q=p
                p ← q
            if p=w then
                    d←d+1
                    if (w,v) è in E o v=w  then   flag ← true
        if d < r_{k-1} then rifiuta
        if flag  then  c←c+1 %se flag è true, v è raggiungibile da un
    vertice w in A_{k-1} o è in A_{k-1}
return c
```

input: G = (V,E), s, t, r_{k-1}
precondizione: r_{k-1} è il numero esatto di vertici raggiungibili da s in k-1 passi
output: il numero dei vertici raggiungibili da s, incluso s in al più k passi.

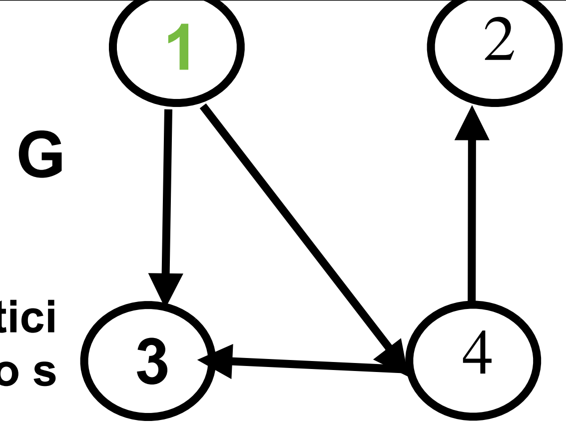
c ← 0
for all v ∈ V do
    d ← 0; flag ← false
    for all w ∈ V do
        p ← s
        for i= 1 to k-1
            scegli non det
            q tale che (p,q)
            è in E, o ancora p
            p ← q
        if p=w then
            d ← d+1
            if (w,v) è in E
             o v=w then
                flag ← true
    if d < r_{k-1} then rifiuta
    if flag then c ← c+1
return c



Il caso di k=2,
r_1=3 c=0,

<u>v=3, ,d=0 flag = false</u>
per w = 1 p=1,
le possibilità sono q=1,3 o 4,
se q = 3, p=q≠w e se q = 4, p=q≠w quindi
in questi casi d non si incrementa
la scelta q = 1, invece comporta
d=1, perché p=q=w
Supponiamo di aver fatto questa scelta.
per w = 2 p=1,
nessuna scelta porta a 2
per w = 3 c'è l'arco (1,3), supponendo di aver
fatto questa scelta allora d=2,
poi w=v quindi flag = true
per w = 4 c'è l'arco (1,4) e supponendo di aver
fatto questa scelta
d=3, poi c'è l'arco (4,3)
d= r_1=3, quindi c= 1. Il vertice 3 va in A_2.

input: G = (V,E), s, t, r_{k-1}
precondizione: r_{k-1} è il numero esatto di vertici raggiungibili da s in k-1 passi
output: il numero dei vertici raggiungibili da s, incluso s in al più k passi.



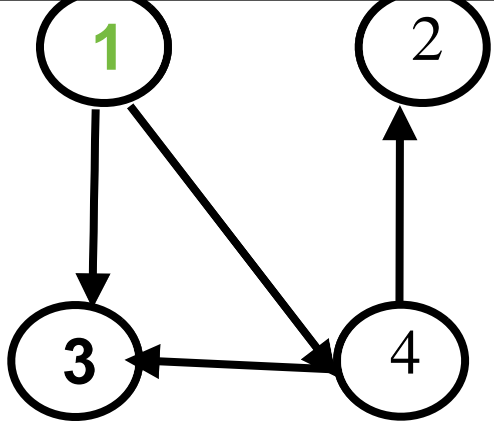

```
c←0
for all v ∈ V  do
        d ← 0; flag ←false
        for all w ∈ V  do
            p ← s
            for i = 1 to k-1
                scegli non det
                q tale che (p,q)
                è in E, o ancora p
                p ← q
            if p=w then
                d←d+1
                if (w,v) è in E
                 o v=w  then
                   flag ← true
        if d < r_{k-1} then rifiuta
        if flag  then  c←c+1
return c
```

Il caso di k=2,
r_1=3 c=0,

v=2, ,d=0 flag = false
per w = 1 p=1,
d=1
con w = 2 p=1,
le possibilità sono q=1,3 o 4,
per w = 3 c'è l'arco (1,3) d=2,
per w = 4 c'è l'arco (1,4),
quindi d=3
poiché (4,2) è un arco flag va a true
c=2. Il vertice 2 va in A_2.

input: G = (V,E), s, t, r_{k-1}
precondizione: r_{k-1} è il numero esatto di vertici raggiungibili da s in k-1 passi
output: il numero dei vertici raggiungibili da s, incluso s in al più k passi.

G




```
c ← 0
for all v ∈ V  do
        d ← 0; flag ← false
        for all w ∈ V  do
            p ← s
            for i = 1 to k-1
               scegli non det
               q tale che (p,q)
               è in E, o ancora p
                p ← q
            if p=w then
                  d ← d+1
                  if (w,v) è in E
                   o v=w  then
                     flag ← true
       if d < r_{k-1} then rifiuta
        if flag  then  c ← c+1
return c
```

Il caso di k=2, r_1=3 c=0,

<u>v=4, ,d=0 flag = false</u>
per w = 1 p=1,
d=1
per w = 2 p=1,
le possibilità sono q=1,3 o 4,
per w = 3 c'è l'arco (1,3) d=2,
per w = 4 c'è l'arco (1,4) d=3
d= r_1=3, quindi c= 3
Il vertice 4 va in A_2.

<u>v=1, ,d=0 flag = false</u>
per w = 1 p=1,
d=1, v=w quindi flag va a true
per w = 2 p=1,
le possibilità sono q=1,3 o 4,
per w = 3 c'è l'arco (1,3) d=2,
per w = 4 c'è l'arco (1,4), quindi d=3
c=4.
Il vertice 1 va in A_2.