Sommario

Problemi di decisione, di ricerca e di ottimizzazione:

come sono legate le complessità delle diverse versioni dei problemi?

Decisione vs ricerca

Se disponiamo di un efficiente algoritmo per risolvere un problema di ricerca abbiamo risolto l'associato problema di decisione.

Potremmo dire che la decisione si riduce alla ricerca.

Quindi se un problema di decisione è NP-hard questo dà evidenza che anche il problema di ricerca associato sia "hard".

Uno dei motivi per cui si considerano problemi di decisione è che spesso le soluzioni algoritmiche per questi si possono utilizzare per i problemi di ricerca associati, con una complessità aggiuntiva polinomiale.

SAT

SAT ={ φ | φ è una formula booleana soddisfacibile}.

Problema SAT

input: φ, dove φ una formula booleana

Problema di decisione: esiste y in {0,1}* che soddisfa φ?

Risposta: Sì o no

Però la soddisfacibilità di una formula booleana è un tipico problema di ricerca (search): cerchiamo un assegnamento di valori di verità che renda vera la formula, tra tutti quelli possibili.

Problema di ricerca: trovare y in {0,1}*, se esiste, che soddisfa φ.

Risposta: un assegnamento che soddisfa φ o "φ non è soddisfacibile"

Decisione vs ricerca per SAT

Supponiamo di disporre di un algoritmo, un oracolo, per SAT, M_{SAT} tale che:

M_{SAT}

input: φ, dove φ una formula booleana

risposta sì se φ è soddisfacibile

altrimenti risposta no.

Possiamo usarlo come "subroutine" in un algoritmo che calcola un assegnamento che soddisfa φ.

IDEA:

Si esegue M_{SAT} su φ se la risposta è no, si termina con risposta "φ non è soddisfacibile", altrimenti

si assegna 0 alla prima variabile e si esegue M_{SAT} sulla formula ottenuta φ_0, se φ_0 non è soddisfacibile si pone la prima variabile a 1, e si controlla se φ_1 è soddisfacibile;

in entrambi i casi

si prosegue nello stesso modo per la seconda variabile, e così via fino ad esaurimento delle variabili.

Algoritmo di ricerca per SAT:esempio di esecuzione

Si esegue M_{SAT} su φ se la risposta è no, si termina con risposta "φ non è soddisfacibile", altrimenti

si assegna 0 alla prima variabile e si esegue M_{SAT} sulla formula ottenuta φ_1, se φ_1 non è soddisfacibile si pone la prima variabile a 1,

in entrambi i casi si prosegue nello stesso modo per la seconda variabile, e così via fino ad esaurimento delle variabili.

Esempio:

$\varphi(x1, x2, x3, x4) = (x1 \vee \neg x2) \wedge (x2 \vee x3 \vee \neg x1) \wedge (x4 \vee \neg x3)$

Si esegue M_{SAT} su
$\varphi(x1, x2, x3, x4) = (x1 \vee \neg x2) \wedge (x2 \vee x3 \vee \neg x1) \wedge (x4 \vee \neg x3)$, poichè è soddisfacibile, si costruisce
$\varphi(0, x2, x3, x4) = \varphi_0(x2, x3, x4) = (0 \vee \neg x2) \wedge (x2 \vee x3 \vee 1) \wedge (x4 \vee \neg x3)$
e si esegue M_{SAT} su φ_0
se la risposta è sì, allora si prosegue assegnando 0 a x2,
se la risposta è no, allora si considera
$\varphi(1, x2, x3, x4) = \varphi_1(x2, x3, x4) = (1 \vee \neg x2) \wedge (x2 \vee x3 \vee 0) \wedge (x4 \vee \neg x3)$
perchè visto che φ è soddisfacibile uno dei due valori è quello giusto!

L'algoritmo per la ricerca per SAT

φ è una formula sulle variabili X1,...,Xn

```
CSAT
Input φ
if MSAT(φ) = 0 then
        output "φ non è soddisfacibile"
        else
            for i = 1 to n do
                if MSAT(< φ(b1,...,bi-1,0>) = 1
                then  bi = 0
                else bi  = 1
  Output b1,...,bn
```

dove $\varphi(b_1,...,b_{i-1},0)$ è la formula ottenuta da φ sostituendo le prime i-1 variabili con $b_1,...,b_{i-1}$ in {0,1}

Complessità: L'algoritmo è polinomiale nella complessità t(n) di M_{SAT}. L'algoritmo M_{SAT} è "chiamato" n+1 volte su formule diverse, quindi si ha $O(nt(n) + n^2)$, tenendo conto del costo di sostituire ogni variabile con il valore scelto in φ.

Possiamo dire che la versione decisionale e la versione ricerca di SAT sono polinomialmente riducibili l'uno all'altro, o più semplicemente che il problema della soddisfacibilità per formule booleane è autoriducibile.

II CLIQUE

Un clique per un grafo **è un sottoinsieme di vertici tutti connessi tra loro.**

input: <G,k>, dove G è un grafo non diretto e k un intero positivo

Problema di decisione: esiste un clique in G di k elementi?

Risposta: Sì o no

Problema di ricerca: determina un clique C di k vertici se esiste o rispondi no altrimenti

Risposta: un clique C o "non esiste"

Decisione vs ricerca per CLIQUE

Supponiamo di disporre di un algoritmo, un oracolo, per CLIQUE, M_{CLIQUE} tale che:

M_{CLIQUE}

input: <G=(V,E),k> dove G è un grafo non diretto e k un intero positivo

risposta sì se G ha un k-CLIQUE

altrimenti risposta no.

Possiamo usarlo come "subroutine" in un algoritmo che calcola un CLIQUE di k vertici:

L'idea si basa sulla seguente osservazione:

se elimino un vertice v da un k-clique può capitare che il grafo risultante abbia ancora un k-clique, perchè v faceva parte di un clique più grande di k, oppure il grafo non ha più un k-clique.

Se invece il vertice v non è in un k-clique la sua eliminazione da G restituisce un grafo che ha ancora un k-clique.

Quindi eliminando da G solo quei vertici che non incidono sull'esistenza di k-clique in G, alla fine ottengo il k-clique.

Algoritmo di ricerca per CLIQUE

Indichiamo con G\v il grafo ottenuto da G eliminando il vertice v da V insieme con gli archi incidenti su di esso.

```
   CCLIQUE
Input <G=(V,E),k> G ha n vertici numerati da 1  a n
if MCLIQUE <G=(V,E),k> = 0 then "G non ha un k-clique"
else
      for i = 1 to n do
          if MCLIQUE(<G\i,k) = 1 then G = G\i
      endFor
      Output G
```

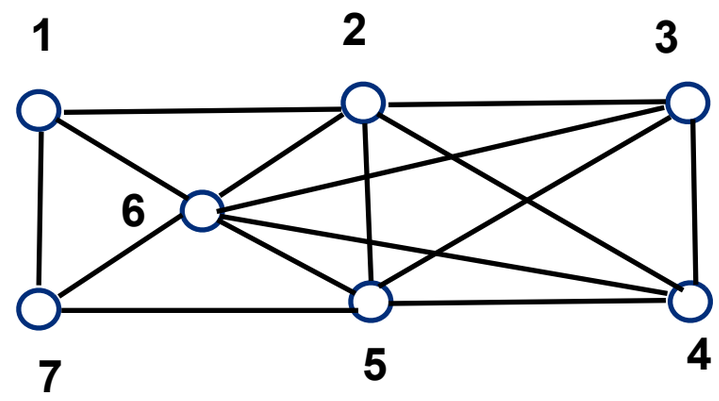
Esempio di esecuzione: CLIQUE

CCLIQUE

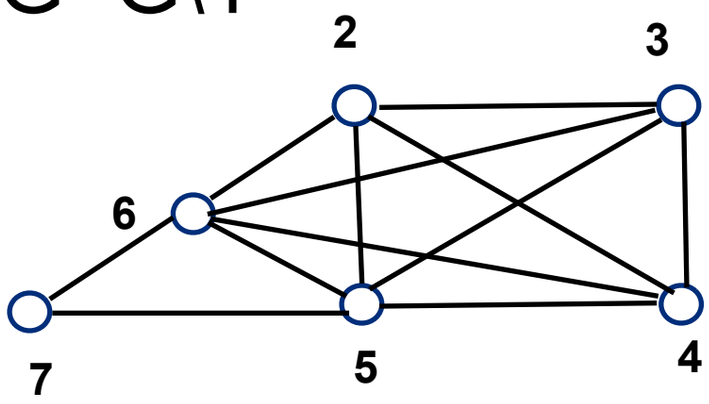
Input <G=(V,E),k> G ha n vertici numerati da 1 a n
if MCLIQUE <G=(V,E),k> = 0 then "G non ha un k-clique"
else
for i = 1 to n do
if MCLIQUE(<G\i,k) = 1 then G = G\i
endFor
Output G

Input:
k=4

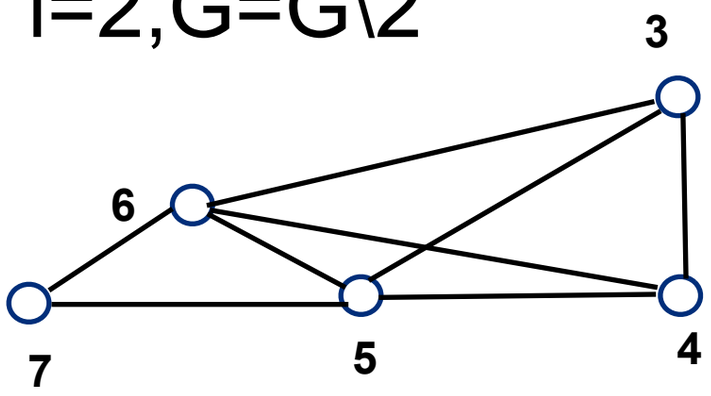
G=



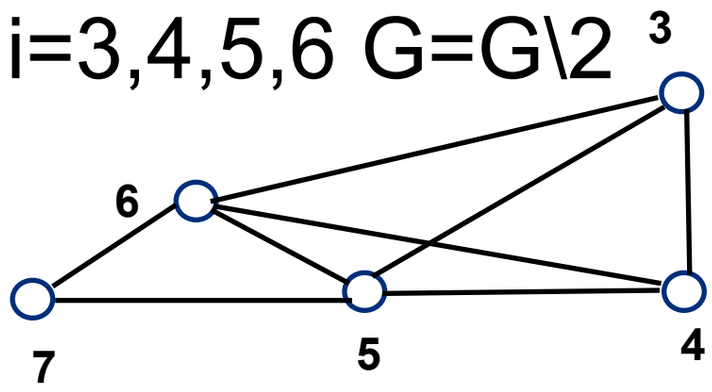
i=1,G=G\1



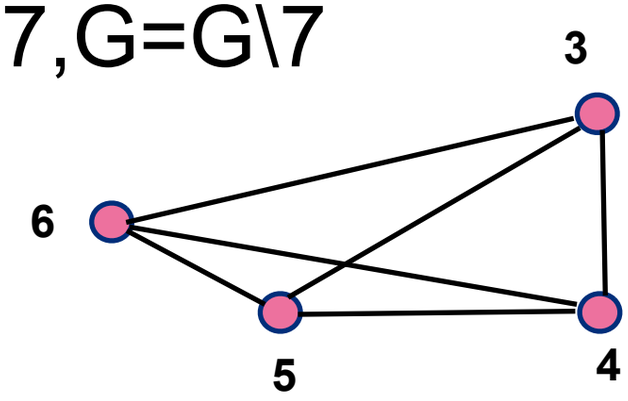
i=2,G=G\2



i=3,4,5,6 G=G\2



i=7,G=G\7



Correttezza ricerca per CLIQUE

```
    CCLIQUE
Input <G=(V,E),k> G ha n vertici numerati da 1  a n
if MCLIQUE <G=(V,E),k> = 0 then "G non ha un k-clique"
else
      for i = 1 to n do
          if MCLIQUE(<G\i,k) = 1 then G = G\i
       endFor
      Output G
```

Correttezza.
Invariante di ciclo: G ha un k-clique.
Inizialmente vero perchè altrimenti il ciclo non viene eseguito.
Conservazione:Se è vero all'i-sima esecuzione del ciclo è vero alla successiva perchè se $M_{CLIQUE}(<G\backslash i,k) = 1$ allora il nuovo grafo G su cui viene eseguito il ciclo è G\i, che ha un k-clique
se invece $M_{CLIQUE}(<G\backslash i,k) = 0$, G non cambia e ha un k-clique per ipotesi.
Conclusione: G ha k vertici ed è un k-clique.
G ha k vertici perchè si eliminano tutti i vertici la cui eliminazione non incide sulla presenza di un k-clique.

Complessità ricerca per CLIQUE

C_{CLIQUE}
Input <G=(V,E),k> G ha n vertici numerati da 1 a n
if M_{CLIQUE} <G=(V,E),k> = 0 then "G non ha un k-clique"
else
    for i = 1 to n do
        if M_{CLIQUE}(<G\i,k) = 1 then G = G\i
    endFor
    Output G

Complessità. Sia G un grafo con n vertici e m archi e $t_M(n,m)$ la complessità della procedura M_{CLIQUE}
Allora C_{CLIQUE} ha complessità $t_C(n,m) = O((n+1)t_M(n,m) + n*(m+n))$

Anche per il problema del CLIQUE possiamo parlare di autoriducibilità

Tecnica generale per l'autoriducibilità

Assumiamo di disporre dell'algoritmo che risolve il problema decisionale

scriviamo l'algoritmo per risolvere il problema di ricerca utilizzando quello per il problema di decisione

analizziamo la correttezza dell'algoritmo ottenuto

analizziamo la complessità dell'algoritmo ottenuto in funzione della complessità di quello per il problema di decisione associato.

Nel caso di grafi, spesso la tecnica consiste nel considerare grafi più piccoli di quello in input ottenuti eliminando vertici o archi e applicando a questi l'algoritmo per il problema di decisione associato.

Decisione vs ottimalità

Spesso i problemi di ricerca si presentano "naturalmente" come problemi di ottimizzazione.

Se un problema di decisione è NP-hard questo dà evidenza che anche il problema di ottimizzazione associato sia "hard".

Anche in questo caso gli algoritmi per i problemi di decisione si possono utilizzare per i problemi di ottimizzazione, con una complessità aggiuntiva polinomiale.

Si utilizza la ricerca binaria.

Esempio del MAX-CLIQUE

Dato un grafo G, si vuole determinare un clique massimale nel grafo.

L'idea: Si basa su quella della ricerca binaria.
Il limite superiore alla dimensione di un Clique è il numero dei vertici del grafo in input.
Si verifica se c'è un clique di dimensione m = n/2, poi per capire se è massimale si cerca un clique di dimensione maggiore nell'intervallo [m+1,n], altrimenti, cioè se un clique di dimensione m non c'è nel grafo, dobbiamo cercarne uno di dimensione minore nell'intervallo [1,m-1]. Reiterando questo ragionamento si ottiene il valore cercato.

Esempio del MAX-CLIQUE

Dato un grafo G, si vuole determinare un clique massimale nel grafo.

O_{CLIQUE}
Input <G=(V,E)> G ha n vertici
low = 1
high = n
while low ≤ high do
    m = (low+high)/2
    if M_{CLIQUE} <G=(V,E),m> = 1 then m'= m; low = m + 1 else high = m - 1
Output C_{CLIQUE} <G=(V,E), m' >

l'output di $C_{CLIQUE}(G,k)$ è un clique di k vertici di G

Complessità: Siano n e m il numero dei vertici rispettivamente degli archi di un grafo, $t_M(n,m)$ la complessità della procedura M_{CLIQUE} e $t_C(n,m)$ la complessità di C_{CLIQUE}
Allora O_{CLIQUE} ha complessità $t_O(n,m) = O(\lg n * t_M(n,m) + t_C(n,m))$

Esempio del MIN-TSP

Dato un grafo G, si vuole determinare un ciclo hamiltoniano di costo minimo nel grafo.

Idea: Si basa sulla ricerca binaria.
Un limite superiore al costo di un ciclo hamiltoniano, c_{max}, è n volte il costo dell'arco di costo massimo e quello inferiore, c_{min}, è n volte il costo dell'arco di costo minimo.

Si verifica se c'è un ciclo hamiltoniano di costo m pari a circa la metà tra minimo e massimo.
Se si trova un ciclo hamiltoniano di costo m, per capire se è minimale si cerca un ciclo hamiltoniano di costo minore nell'intervallo $[c_{min}, m-1]$, altrimenti, cioè se un ciclo hamiltoniano di dimensione m non c'è nel grafo, dobbiamo cercarne uno nell'intervallo $[m+1, c_{max}]$. Come nel caso del clique reiterando il ragionamento si ottiene l'algoritmo desiderato

Esempio del MIN-TSP

Dato un grafo G, si vuole determinare un ciclo hamiltoniano di costo minimo nel grafo.

O_{TSP}
Input <G=(V,E)> G ha n vertici
low = c_{min}*n
high = c_{max}*n;
while low ≤ high do
    m = (low+high)/2
    if M_{TSP} <G=(V,E),m> = 1 then m'= m; high = m - 1 else low = m + 1
Output C_{TSP} <G=(V,E), m' >

l'output di $C_{TSP}(G,k)$ è un ciclo hamiltoniano di costo k

Complessità: Siano n e m il numero dei vertici rispettivamente degli archi di un grafo, $t_M(n,m)$ la complessità della procedura M_{TSP} e $t_C(n,m)$ la complessità di C_{TSP} Allora O_{TSP} ha complessità $t_O(n,m) = O(m + lg\ n^* t_M(n,m) + t_C(n,m))$