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, Dec_{SAT} tale che:

Dec_{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 Dec_{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 Dec_{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

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

Si esegue Dec_{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 Dec_{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 il problema di ricerca

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

$Search_{SAT}$
Input φ
if $Dec_{SAT}(\varphi) = 0$ then
output "φ non è soddisfacibile"
else
for i = 1 to n do
if $Dec_{SAT}(< \varphi(b_1,...,b_{i-1},0>) = 1$
then $b_i = 0$
else $b_i = 1$
Output $b_1,...,b_n$

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} e l'i-sima con 0

Complessità: L'algoritmo è polinomiale nella complessità t(n) di Dec_{SAT}. L'algoritmo Dec_{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.

Decisione e ricerca per SAT

Quindi possiamo dire che la ricerca si riduce alla decisione, perché utilizziamo l'algoritmo di decisione per risolvere il problema di ricerca.

Quindi 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 (polinomialmente) autoriducibile.

In altri termini: un problema è autoriducibile se una soluzione algoritmica efficiente del problema di decisione associato può essere usata per risolvere efficientemente il problema di ricerca.

Il 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, DEC_{CLIQUE} tale che:

DEC_{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.

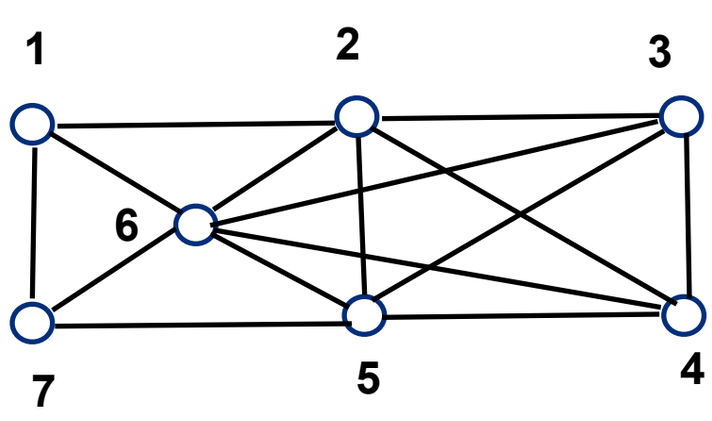
Decisione vs ricerca per CLIQUE

L'idea si basa sulla seguente osservazione:

se si elimina 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 perché il vertice v non è in un k-clique.

Esempio, sia k=4. I vertici 2,3,4,5 formano un 4-clique.
se si elimina 6 il grafo ha ancora un 4-clique, perché 2,3,4,5,6 formano un 5-clique, se elimina 1 di nuovo c'è ancora il 4-clique che non lo comprende.

G =



Si può usare l'algoritmo che decide la presenza di un k-clique per eliminare da G solo quei vertici che non incidono sull'esistenza di k-clique in G.

Così procedendo alla fine rimarranno solo i vertici del 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.

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

Complessità ricerca per CLIQUE

Search$_{CLIQUE}$
Input <G=(V,E),k> G ha n vertici numerati da 1 a n
if DEC$_{CLIQUE}$ <G=(V,E),k> = 0 then "G non ha un k-clique"
else
for i = 1 to n do
if DEC$_{CLIQUE}$(<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 DEC$_{CLIQUE}$
Allora Search$_{CLIQUE}$ 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.

Si può dimostrare che tutti i problemi NP-completi sono autoriducibili.

L'autoriducibilità: attenzione!

Non per tutti i problemi in NP è possibile costruire un algoritmo per la ricerca efficiente da quello per il problema di decisione associato.

Prendiamo

COMPOSITES={x | x=p*q per interi p,q>1}, che è il complemento di

PRIMES={x | x è un numero primo}

che è in P, ma il problema di ricerca associato consiste nella ricerca di un fattore per x.

Il problema della fattorizzazione

COMPOSITES

input: <x> dove x è un intero

Problema di decisione: esiste un divisore non banale di x?

Risposta: Sì o no

Problema di ricerca: determina un divisore non banale di x se esiste o rispondi no altrimenti

Risposta: un divisore non banale di x o “x è primo”

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.

Ott_{CLIQUE}
Input <G=(V,E)> G ha n vertici
low = 1
high = n
while low ≤ high do
m = (low+high)/2
if DEC_{CLIQUE} <G=(V,E),m> = 1 then m'= m; low = m + 1 else high = m - 1
Output $Search_{CLIQUE}$ <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, $t_{DEC}(n,m)$ la complessità della procedura DEC_{CLIQUE} e $t_{Search}(n,m)$ la complessità di $Search_{CLIQUE}$
Allora Ott_{CLIQUE} ha complessità $t_O(n,m) = O(\lg n^* t_{DEC}(n,m) + t_{Search}(n,m))$

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. Sia n il numero dei vertici.
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 costo 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.

Ott_{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 DEC_{TSP} <G=(V,E),m> = 1 then m'= m; high = m - 1 else low = m + 1
Output $Search_{TSP}$ <G=(V,E), m' >

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

Complessità: Siano n e m il numero dei vertici rispettivamente degli archi di un grafo, $t_{DEC}(n,m)$ la complessità della procedura DEC_{TSP} e $t_{search}(n,m)$ la complessità di $Search_{TSP}$
Allora Ott_{TSP} ha complessità $t_O(n,m) = O(lg ((c_{max} - c_{min})n)* t_{DEC}(n,m) + t_{search}(n,m))$