Teoria della Computazione

Antonio Vivace - Sorgenti

Complessità

Definizioni

Cammino Hamiltoniano

Cammino che tocca tutti i vertici del grafo una ed una sola volta.

Si ha un *ciclo* hamiltoniano quando esiste un arco che collega l'ultimo vertice del cammino con il primo, realizzando così un ciclo che visita tutti i vertifici per poi ritornare al punto di partenza.

Cammino Euleriano

Cammino che tocca tutti gli archi di un grafo una ed una sola volta.

Ciclo euleriano: Cammino euleriano che inizia e finisce sullo stesso vertice.

Teorema di Eulero: un grafo connesso ha un ciclo di eulero se e solo se tutti i vertici hanno grado pari.

Il problema del ciclo euleriano è facile per il teorema di eulero.

Grado di un vertice: numero di archi uscenti + entranti.

Osservazione NP-equivalenza

I problemi NP-Completi sono riducibili l'uno all'altro in tempo polinomiale, e tutti i problemi in NP sono riducibili in tempo polinomiale a problemi NP-Completi. Dunque, dato un qualsiasi problema NP-Hard, tutti i problemi in NP sono riconducibili ad esso.

Riduzioni

SAT $\leq_p$ 3SAT.

Mapperemo una formula CNF f in una formula g 3CNF tale che g è soddisfacibile se e solo se f lo è. Mostreremo prima il caso in cui f sia una 4CNF. Sia C una clausola di di f. $C = u_1 \vee \neg u_2 \vee \neg u_3 \vee u_4$ Aggiungiamo una nuova variabile z ad f e sostituiamo C con congiunzione delle seguenti due clausole

- $C_1 \equiv u_1 \vee \neg u_2 \vee z$
- $C_2 \equiv \neg u_3 \vee u_4 \vee \neg z$
- e quindi $C := C_1 \wedge C_2$

Se C è vera, allora c'è un assegnamento per z che soddisfa sia C_1 che C_2 . Se C è falsa, qualsiasi assegnamento di z rende o C_1 o C_2 falsa.

Questa idea può essere applicata ad una clausola generica di dimensione 4 ed in generale di dimensione k (per k $>$ 3) in un equivalente coppia di clausole C1 di dimensione k-1 e C2 di dimensione 3 che dipendono dalle k variabili di C ed una variabile aggiuntiva z.

Applicare questa trasformazione comporta una trasformazione in tempo polinomiale di una formula CNF f in una equivalente formula 3CNF formula g.

3SAT $\leq_p$ IND

Ricordiamo che

- 3SAT = { ϕ : ϕ formula in 3-CNF soddisfacibile}
- IND = $\{(G,k)$: esiste un insieme indipendente del grafo $G = (V,E)$ di dimensione $k\}$

Ci serve una funzione tale che presa in input una formula 3CNF (3SAT) la trasformi in una coppia (G,k) dove G è un grafo con un indipendent set di dimensione k se solo se ϕ è soddisfacibile.

G contiene 3 vertici per clausola, uno per letterale della clausola (ogni sottografo è un *gadget*). Ogni gadget è connesso. Ora, colleghiamo ogni letterale con il suo negato in un altro gadget, se presente.

Fissiamo il parametro k uguale al numero di clausole di ϕ.

Questa trasformazione opera in tempo polinomiale, come si vede facilmente (al più quadratico rispetto al numero di vertici).

Mostriamo ora che la corrispondenza tra ϕ e (G,k) appena costruito è una riduzione corretta.

1) $IND \rightarrow 3SAT$. Sia S un insieme indipendente di dimensione k. Poichè in un triangolo tutti i vertici sono fra loro connessi, S non conterrà più di un vertice per ogni triangolo. Considerando i letterali in S come veri, si ottiene un assegnamento di verità consistente (se è incluso un letterale non lo è il suo opposto, dato che sono collegati da archi le assegnazioni opposte) e tutte le clausole sono soddisfatte (un vertice per triangolo, ovvero una clausola).

2) $IND \leftarrow 3SAT$ Dato un assegnamento che rende vera ϕ, per ogni gadget prentiamo un letterale che risulta vero nell'assegnamento. Tale insieme costituisce un Indipendent Set per considerazione analoghe a quelle già viste ed ha dimensione k per costruzione.

Vertex Cover $\leq_p$ Set Cover

Ci serve costruire una funzione polinomiale che mappi ogni istanza di Vertex Cover (G = (V,E), k) in un'istanza di Set Cover C' tale che (G,k) e (U, S, C', j) = f(G,k) diano le stesse risposte nei rispettivi problemi, per tutti i grafi e dimensioni.

Costruiamo un insieme per ogni vertice che contiene gli archi incidenti a quel vertice. Poniamo l'U da coprire uguale all'insieme degli archi (U = E), numeriamo i vertici da 1 a n e includiamo in S_i gli archi incidenti al vertice i. Il parametro j di Set Cover è il k di Vertex Cover.

1) $VertexCover \rightarrow SetCover$. Se G ha una copertura di al di più j vertici, per costruzione, a questa copertura corrisponde un sottoinsieme C' di sottoinsiemi di U. Poichè C è una copertura per G e U e l'insieme degli archi di G, ogni elemento di U deve essere incluso in almeno un sottoinsimeme di C' poichè per definizione di copertura ogni arco ha alcumeno uno dei due vertici in C.

2) $VertexCover \leftarrow SetCover$. Se C' è un set cover di dimensione al più j, ad ognun insieme è associato un vertice. Sia C l'insieme dei vertici corrispondenti agli insiemi in C'. C ha la stessa cardinalità di C'. Preso un qualsiasi arco in U, sicuramente C' contiene almeno un insieme che include l'arco e. Tale insieme corrisponde a un nodo che è estremo di e (contenendo i suoi incidenti), quindi C deve contenere almeno un estremo di e.

HAM $\leq_p$ TSP

Ricordiamo che l'input di TSP è un grafo pesato G(V, E, w) con w la funzione di peso. Esiste un cammino da u a u che visita ogni nodo di V in G esattamente una volta con costo $\leq k$?

Considerato G l'input per HAM, costruisco G' per TSP con gli stessi vertici di G e assegno peso 1 agli archi in G e peso 2 a tutti gli altri (grafo completo per TSP!)

Mostriamo ora che esiste un ciclo hamiltoniano se e solo se esiste un tour TSP

1) $HAM \leftarrow TSP$ con k numero di vertici di G', per costruizione se esiste un cammino di lunghezza $\leq$ k in TSP esso deve essere costituito soltanto da archi di peso 1. Tali archi sono anche in G quindi esiste il ciclo hamiltoniano.
2) $HAM \rightarrow TSP$ se esiste un ciclo hamiltoniano in G è sicuramente costituito da |V| archi (definzione). Percorrendo gli stessi archi in G' ottengo un cammino per TSP lungo |V| archi di peso 1 (costruzione). Essendo il limite di costo per la decisione del problema TSP, essa è affermativa.

Vertex Cover $\leq_p$ IND

todo

Complessità parametrica

Tempo = $f(k)n^c$

Vertex Cover con bounded search tree $O = (2^k|V|)$

Si dica quale delle seguenti affermazione è vera.

Un problema NP-Hard può non essere NP-completo

Vero.

Un problema in NP può non essere NP-completo

Vero.

Ci sono problemi in NP che si riducono in tempo polinomiale ad un problema in P

Falso.

Esistono problemi NP-Hard che non sono NP-Completi

Vero.

Esistono problemi in NP che non sono NP-Hard

Vero.

Un problema nella classe di complessità P è risolubile in tempo polinomiale da una MdT non deterministica

Vero.

Non esistono macchine di Turing che non terminano

Falso. Esistono Macchine di Turing che non terminano. Inoltre, non è possibile stabilire in generale se un algoritmo, dato un input finito, termini o continui la sua esecuzione all'infinito (Halting Problem).

Un problema in P non è nella classe NP

Falso. Tutti i problemi P sono anche in NP, in quanto le MdT non deterministiche sono più potenti e risolvono in tempo polinomiale anche i problemi risolti in tempo polinomiale dalle macchine deterministiche.

Un problema in NP non può essere NP-Hard

Falso. Esistono problemi che sono NP ed NP-Hard (i cosidetti problemi NP-Completi).

Un problema NP-Hard è un problema che sta in NP

Falso. Non necessariamente. Esistono problemi NP-Hard che stanno anche in NP e si chiamano NP-Completi. Alcuni problemi NP-Hard non stanno in NP.

Un problema in NP si riduce in tempo polinomiale ad un problema NP-Hard

Vero. Tutti i problemi in NP sono riducibili in tempo polinomiale ad un problema NP-Hard (o NP-Completo).

Ci sono problemi in NP che non si riducono in tempo polinomiale a nessun problema in NP

Falso (?). Tutti i problemi in NP si riducono ad un problema NP-Completo. Non è detta la stessa cosa di due problemi in NP.

Un problema NP-Hard deve essere anche NP-completo

Falso.

Un problema in NP può non essere NP-completo

Vero.

Ci sono problemi in NP che non si riducono in tempo polinomiale a un problema in P

Vero (?), altrimenti non sarebbero problemi in NP.

Esistono problemi in P che non sono in NP

Falso.

Un problema in NP non può essere NP-Completo

Falso.

Un problema NP-difficile deve essere calcolabile da una macchina di Turing non deterministica in tempo polinomiale

Falso.

Esistono problemi NP-Hard che non sono NP-completi

Vero. Un problema è NP hard se tutti i problemi in NP si riducono polinomialmente ad esso. Non è necessario che esso stesso sia in NP. Se lo è, tuttavia, si chiama NP-Completo.

Esistono problemi in NP che non sono NP-Hard

Vero.

Un problema nella classe di complessità P è risolubile in tempo polinomiale da una MdT non deterministica

Vero.

Si dia la definizione di problema NP-Completo

Un problema è NP-Completo se e solo se:

1) È in NP, ovvero può essere risolto in tempo polinomiale da una macchina non deterministica o -equivalentemente- verificato in tempo polinomiale da una macchina deterministica.
2) È NP-Difficile, ovvero tutti i problemi in NP possono essere ridotti in tempo polinomiale (da una macchina di turing deterministica) ad esso.

Quindi, la classe NP-Completo contiene tutti i problemi di **decisione** in NP per cui qualsiasi problema in NP può essere ridotto ad essi in tempo polinomiale da una macchina di Turing deterministica.

Un problema NP-Completo si riduce in tempo polinomiale ad un problema NP-Hard

Vero. La classe dei problemi NP-Completi è NP-Equivalente, quindi ogni NP-C è riducibile ad un altro NP-C. Dato che NP-Completo $\rightarrow$ NP-Hard, l'affermazione è corretta.

Un problema NP-Hard può non essere in NP

Vero.

Ci sono problemi in NP che non si riducono in tempo polinomiale ad un problema in NP

Vero (?). L'unica garanzia è che ognuno di essi si deve poter ridurre in tempo polinomiale ad un problema NP-Hard.

Un problema NP-Hard sta in NP

Falso. Non necessariamente.

Un problema NP-Hard si riduce in tempo polinomiale ad un altro problema NP-Hard

Falso (?).

Se un problema in NP si risolve polinomialmente, allora P = NP

Falso. Se si risolvesse un problema NP-Hard in tempo polinomiale, allora P=NP.

Il minimo albero di copertura di un grafo completo pesato ha un costo maggiore del cammino Hamiltoniano di minimo costo

Falso. Un cammino Hamiltoniano di minimo costo però potrebbe avere costo maggiore di un minimo albero di copertura (è costretto a scegliere archi per non tornare mai su un vertice già coperto, cosa che invece può fare un albero di copertura). Ogni cammino può essere trasformato in albero, ma non viceversa. Quindi, un cammino di copertura minimo (Hamiltonian Path) potrebbe essere più costoso dell'albero.

Dare un esempio di problema indecidibile

Un classico problema indicidibile è il *Problema di corrispondenza Post*. Date due sequenze di parole $A =< v_1, v_2, ..v_n >$ e $B =< w_1, w_2, ..w_m >$, ci chiediamo se esiste una sequenza di $n \geq 1$ (compresi tra i 1 e m) indici tale che

$v_{i1}.v_{i2}....v_in = w_{i1}.w_{i2}....w_{in}$

In altre parole, è possibile comporre con due dizionari diversi una stessa frase?

Approssimazione

Algoritmo Double Tree 2-approssimante per TSP metrico

todo

Algoritmo di Christofides 3/2-approssimante per TSP metrico

todo

4) Sia A un algoritmo 5/2-approssimante per Vertex-Cover (VC). Sia dato il grafo G = (V,E) dove E = {(1,2),(1,3),(1,4),(4,5),(3,4)}. Dire qual è la massima dimensione di una soluzione restituita da A sul grafo G.

L'ottimo è di dimensione 2, ad esempio {1, 4}. La più grande (peggiore) soluzione che può dare A è 5/2 * 2 = 5.

Pattern Matching

Definizioni

Definizione ricorsiva di Bordo

Sia ε la stringa vuota, σ simbolo singolo dell'alfabeto σ.

- $B(\varepsilon) = \varepsilon$
- $B(\sigma) = \varepsilon$
- $B(X\sigma) = B(B(X)\sigma)$ se $X[|B(X)|+1] \neq \sigma$

Algoritmi

PME con Automi a stati finiti

Input del problema: testo T di lunghezza n, pattern P di lunghezza m.

Fase 1, Preprocessing

Definiamo la funzione di transione $\delta : \{0..m\} \times \Sigma \to \{0..m\}$:

- $\delta(j, \sigma) = j+1 \leftrightarrow P[j+1] = \sigma \wedge j < m$
- altrimenti, $\delta(j, \sigma) = k$, con $k = |B(P[i, j]\sigma)|$

Complessità: $\mathcal{O}(m|\Sigma|)$

Fase 2, Scansione del testo

Definita δ, partiamo dallo stato 0 e percorriamo l'automa consumando un simbolo di T alla volta. Ogni volta che si arriva allo stato $= m = |P|$ significa che c'è un occorrenza di P in T in posizione *numero transizione* - m + 1.

Complessità: $\mathcal{O}(n)$

PM Esatto con Knuth-Morris-Pratt (KMP)

Fase 1, Calcolo della funzione di fallimento φ

Dato un pattern P di lunghezza m, la funzione

$\varphi : \{0, 1..m\} \to \{-1, 0, 1..m-1\}$

È definita come segue:

- $\varphi(0) = -1$
- $\forall j \geq 1 \wedge j \leq m, \varphi(j) = |B(P[1, j])|$

  (lunghezza del bordo di P fino a j).

Calcolo di φ per induzione

Complessità: $\mathcal{O}(m)$

- $\varphi(0) = -1$
- $\varphi(1) = 0$
- Per ogni $j > 1$, fino a $j = m$:

  $k = \varphi(j-1)$

  A) Se $P[k+1] = P[j]$ allora $\varphi(j) = k+1$
  B) altrimenti, $k = \varphi(k)$ e torna ad A

todo: esempio

```
prefix (P)
begin
F(0) = -1
F(1) = 0

for j = 2 to m do:
    k = F(j-1)
    while k >= 0 and P[k+1] != P[j]
        k = F(k)
    F(j) = k + 1
end
```

Fase 2, Scansione del testo

Complessità: $\mathcal{O}(n)$

Viene confrontato, simbolo per simbolo, P con una finestra W di T (inizialmente W si posiziona su i=1 di T), da sinistra verso destra. Se vengono confrontati tutti i simboli con successo, esiste un'occorrenza di P in T, alla posizione i.

Al primo mismatch, W viene spostata verso destra alla posizione p = i + j + $\varphi(j-1)-1$. I primi k = $\varphi(j-1)$ simboli di P non vengono più confrontati e si parte da i+j-1.

```
KMP (P,T,F)
begin
    m = |P|
    n = |T|
    j = 0
    for q = 1 to n do
        while j >= 0 and P[j+1] != T[q] then
            j = F(j)
        j = j + 1
        if j = m then
            return q-m+1
end
```

PM Esatto con Baeza-Yates e Gonnet (BYG, paradigma SHIFT-AND)

Complessità $O(|\Sigma|+m)$

Fase 1, Preprocessing

$B_\sigma = B_\sigma[i] = 1 \leftrightarrow P[i] = \sigma$, altrimenti 0

Dato il pattern P di lunghezza m, la tabella B è l'insieme delle parole B_σ, per ognuno dei $\sigma \in \Sigma$

Calcolo di B:

- Inizializzazione: tutta B a 0, M a `1000..`
- Per ognuno dei caratteri in P:

  $B_\sigma = M$ OR $B\sigma$

  SHIFT RIGHT di M

Fase 2, Scansione del testo

Definizione di parola D_j. Dato un indice j compreso tra 0 e n=|T|, D è una parola di m=|P| bit tale che:

$d_i = D_j[i] \leftrightarrow P[1,i] = suff(T[1,j])$

Se $d_m \leftrightarrow$ P ha un'occorrenza in T che finisce in j.

- Inizializzazione di una maschera M = `00..1` di m bit tutti uguali a 0 tranne l'ultimo che è 1
- Inizia dalla parola $D_0 = 00.00$
- per j tra 1 ed ed n calcola D_j come segue:
  - D_j = 1RSHIFT(D_{j-1}) AND $B_{T[j]}$
- ogni volta che D_j and M è diverso da 0..0 viene c'è un matching di P in T a j-m+1

PM Approssimato con Wu-Manber (paradigma SHIFT-AND)

todo

Definire la parola D_j^k per la ricerca approssimata di una stringa S in un testo T tramite algoritmo di Wu-Manber. Fare un esempio esplicativo

Dato

- $0 \leq j \leq n = |T|$
- $0 \leq h \leq k$

D_j^h è una parola di $m = |P|$ bit ($D_j^k = d_1 d_2 .. d_m$) tale che:

- $d_i = D_j^h[i] = 1 \leftrightarrow P[1,i] = \text{suff}_h(T[1,j])$
- altrimenti $d_i = D_j^h[i] = 0$

In cui $P[1,i] = \text{suff}_h(T[1,j])$ se $P[1,i]$ occorre come suffisso di $T[1,j]$ con al più h errori.

Esempio:

- T = `absd aabb abc`
- P = `bbab` (m = 4)

$D_8^1 =$ `1110`

7) Data la BWT B = `gg$ccaacac` di un testo T definito su di un alfabeto {a, c, g, t}, specificare senza ricostruire T (e motivando la risposta) quanti sono i suffissi che iniziano con il simbolo `c` e in che posizione stanno nell'ordinamento lessicografico di tutti i suffissi di T. Determinare inoltre l'FM Index e ricostruire il testo T.

1) Riordiniamo B

B = `gg$ccaacac`

F = `$aaaccccgg`

Ci sono 4 suffissi che iniziano con il simbolo c, in posizione 5, 6, 7 ed 8.

Motivazione: la BWT B di un testo T è la permutazione dei simboli di T tale che B[i] è il simbolo che precede l'i-esimo suffisso in ordine lessicografico.

2) Determiniamo l'FM Index

L'FM Index è composto da due funzioni: C ed Occ:

$C(\sigma) : \Sigma \to N$

$C(\sigma)$ = numero di simboli di B lessicograficamente minori di σ

$Occ : \Sigma \times \{1,2,3...|B|+1\} \to N$

$Occ(\sigma, i)$ = numero di simboli uguali a σ in $B[1, i-1]$

j = C(sigma) + Occ(sigma,j) + 1

8) Descrivere l'algoritmo di ricerca esatta di un pattern P in una stringa S basato sulla BWT (Burrows-Wheeler Transform) di S.

- Spazio di B: $\mathcal{O}(n \log |\Sigma|)$
- B può essere calcolata da S in tempo $\mathcal{O}(n)$