ESAME 04/09/20

4) L'algoritmo proposto dal Prof. Milton dovrebbe determinare se un grafo non orientato G con n vertici contiene un ciclo hamiltoniano (un ciclo che passa per tutti i vertici esattamente una volta). Questo è un classico problema NP-Completo.

$T(n) = 3T\left(\frac{n}{2}\right) + n^2$

$a = 3 \quad b = 2 \quad f(n) = n^2 \qquad n^{\log_2 3} < n^2 \Rightarrow f(n) > g(n)$

3° CASO DEL TEOREMA MASTER

$f(n) = \Theta(n^{d+\varepsilon}) \Rightarrow n^2 = n^{\log_2 3 + \varepsilon}$

$\varepsilon = 2 - \log_2 3 > 0$ ✓

$\exists\, c < 1$ t.c. per n suff. grande : $a f\left(\frac{n}{b}\right) \leq c f(n)$

$3\left(\frac{n}{2}\right)^2 \leq c n^2 \Rightarrow \frac{3}{4} \cancel{n^2} \leq c \cancel{n^2} \Rightarrow c \geq \frac{3}{4}$

$T(n) = \Theta(n^2)$

L'affermazione del Prof. Milton NON È VEROSIMILE perché:

1) Risolvere il problema del ciclo hamiltoniano in tempo polinomiale contraddirebbe l'ipotesi $P \neq NP$
2) La complessità $O(n^2)$ è ordini di grandezza più efficiente di qualsiasi algoritmo noto per problemi NP-completi.
3) Non esiste alcuna tecnica nota che permette di risolvere problemi NP-completi con tale efficienza.

3) PROBLEMA RISOLTO DALL'ALGORITMO

L'algoritmo MyAlgorithm calcola il prodotto massimo dei pesi dei cammini dal vertice sorgente s a tutti gli altri vertici nel grafo orientato pesato G. In particolare:

- Trova il cammino da s a ogni altro vertice che massimizza il prodotto dei pesi degli archi
- Restituisce il valore 2^a (che corrisponde al peso del cammino massimo dal nodo sorgente a se stesso)

CORRETTEZZA DELL'ALGORITMO

1) INIZIALIZZAZIONE:
- Imposta $d[u] = +\infty$ per tutti i vertici tranne s
- Imposta $d[s] = 0$ (peso del cammino da s a se stesso)
- Inizializza la coda Q con tutti i vertici

2) FASE PRINCIPALE:
- Esegue $|V| - 1$ iterazioni (come Dijkstra)
- Ad ogni passo estrae il vertice u con $d[u]$ minimo
- Per ogni vicino v, aggiorna $d[v]$ come minimo tra:
  - Il valore corrente $d[v]$
  - $d[u] + \log_2 w(u,v)$ (equivalente a moltiplicare i pesi nel dominio originale)

3) FASE FINALE:
- Calcola il minimo ciclo che ritorna a s (se esiste)
- Restituisce 2^a (che è il peso del cammino da s a se stesso)

PERCHÉ FUNZIONA

La chiave è la trasformazione logaritmica:
- Massimizzare il prodotto dei pesi $\prod w(u,v)$ equivale a minimizzare la somma dei logaritmi $\sum \log_2 w(u,v)$
- L'algoritmo è essenzialmente l'algoritmo di Dijkstra applicato dei pesi

COMPLESSITÀ

Con implementazione della coda Q come array lineare:
1) INIZIALIZZAZIONE: $O(|V|)$
2) EXTRACTMIN: $O(|V|)$ per operazione, eseguita $|V|-1$ volte $\rightarrow O(|V|^2)$
3) AGGIORNAMENTO $d[v]$: $O(1)$ per operazione, al massimo $O(|E|)$ aggiornamenti $\rightarrow O(|E|)$
4) FASE FINALE: $O(|V|)$

Complessità totale: $O(|V|^2)$ (dominata dalle ExtractMin)

COSA RESTITUISCE CON 5 VERTICI COME SORGENTE

Se usiamo ogni vertice come sorgente ($a = 0$ per tutti):
- L'algoritmo restituirà sempre 1 (2^0) perché il peso del cammino da un nodo a se stesso è 1
- La fase finale con $(u,s) \in E$ serve a verificare cicli negativi (nel dominio logaritmico), ma non influisce sul valore restituito quando $a=0$.

ESAME 04/09/20

1) MIN_HEAP

Per ogni nodo i diverso dalla radice, si ha la seguente proprietà: $A[parent(i)] \leq A[i]$

Si può dimostrare per induzione e per transitività dell'operazione $\leq$ che la proprietà del min_heap garantisce che il minimo elemento di un min_heap si trova nella radice e che il sottoalbero di un nodo contiene valori non minori del valore contenuto nel nodo stesso.

```
intersect_min_heaps(H1, H2) {
    Ordina H1 e H2 in maniera crescente
    Crea un nuovo heap intersection
    i = j = 0
    while (i < H1.length AND j < H2.length) {
        if( H1[i] == H2[j]) {
            intersection.append(H1[i])
            i++
            j++
        }
        else if (H1[i] < H2[j])
            i++
        else
            j++
    }
    n = intersection.length
    For (i = floor(n/2) DOWN TO 1)
        min-heapify(intersection, i)
    return intersection
}
```

```
min_heapify(Heap A, int i){
    l=left(i)
    r=right(i)
    if(l ≤ A.heapsize AND A[l] < A[i]){
        minimo=l
    }
    else
        minimo =i
    if(r ≤ A.heapsize AND A[r] < A[minimo])
        minimo = r
    if(i ≠ minimo){
        Scambio A[i] e A[minimo]
        min_heapify(A, minimo)
    }
}
```

ANALISI COMPLESSITÀ

Ordinamento H_1 : $O(m_1 \log m_1)$

Ordinamento H_2 : $O(m_2 \log m_2)$

Intersezione array ordinati: $O(m_1 + m_2)$

Costruzione del nuovo heap: $O(m)$, $m = \min(m_1, m_2)$

Complessità totale: $O(m_1 \log m_1 + m_2 \log m_2 + (m_1 + m_2) + m) = O(m_1 \log m_1 + m_2 \log m_2)$

L'approccio è efficiente perché:

1) Sfrutta la proprietà dei Min_Heap che permette l'estrazione ordinata in tempo $O(m \log m)$
2) L'intersezione di array ordinati è un'operazione lineare
3) La ricostruzione dell'heap è ottimale $O(m)$
4) Evita di fare ricerche multiple negli heap che costerebbero $O(m_1 * m_2)$ nel caso ingenuo.

ESAME 04/09/20

2) Quicksort_LB(A, p, q)
```
if p<q then
    r = ⌊(p+q)/2⌋   //scegliamo il pivot come elemento mediano
    QuickSelect (A, p, q, r)   // Ora A[r] e' il suo valore finale
    QuickSort_LB (A, p, r-1)   // Ordina la parte sinistra
    Quicksort_LB (A, r+1, q)   // Ordina la parte destra
```

Ad ogni livello partizioniamo l'array in due parti approssimativamente uguali

$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$

$a=2 \quad b=2 \quad d=\log_2 2 = 1 \quad f(n) = \log n$

$f(n) < g(n)$

1° CASO DEL TEOREMA MASTER

$f(n) = O(n^{d-\varepsilon}) \Rightarrow \log n = n^{1-\varepsilon}$ per ogni $\varepsilon > 0$

$T(n) = \Theta(n)$

ESISTENZA DI UN QUICKSELECT CON COMPLESSITÀ $O(\log n)$

Lower bound teorico:

- Il problema della selezione richiede $\Omega(n)$ operazioni nel caso peggiore
- Per trovare l'elemento in posizione r, e' necessario esaminare almeno una frazione costante degli elementi

Contraddizione con la complessità:

- Un algoritmo $O(\log n)$ implicherebbe che possiamo trovare un elemento in tempo logaritmico senza leggere tutto l'input
- Questo viola il lower bound per problemi di selezione basati su confronti.