Relazione di

Laboratorio di Algoritmi e Strutture Dati

Edoardo Grassi

Settembre 2023

Indice

Premessa

Questa relazione fa parte di un progetto per il corso di Laboratorio di Algoritmi e Strutture ed è composto da due esercizi:

1. Confronto tra diverse strutture dati per insiemi disgiunti nell'ambito della ricerca di componenti connesse in grafi non diretti
2. Confronto tra *insertion sort* e *merge sort*

In questa relazione viene analizzato e discusso solo il primo esercizio, la cui implementazione è stata scritta in codice *Python*.

1 Introduzione al problema

Le strutture dati per insiemi disgiunti vengono utilizzate in algoritmi dove sia necessario raggruppare diversi elementi sotto un unico rappresentante; definito $\mathcal{S}$ l'insieme contenente tutti gli insiemi disgiunti, possiamo definire tre operazioni fondamentali su di esso:

- MAKE-SET(x): aggiunge a $\mathcal{S}$ un nuovo insieme $\mathcal{S}_i = \{x\}$ (x non deve esistere all'interno di alcun altro insieme, altrimenti non sarebbero disgiunti)
- FIND-SET(x): individua l'elemento rappresentante dell'insieme nel quale x è contenuto
- UNION(x, y): dati $x \in \mathcal{S}_x$ e $y \in \mathcal{S}_y$ ($x \neq y$ per quanto detto prima) $\mathcal{S} = (\mathcal{S} - \mathcal{S}_x - \mathcal{S}_y) \cup \mathcal{S}_\cup$ dove $\mathcal{S}_\cup = \mathcal{S}_x \cup \mathcal{S}_y$ il cui rappresentante è dato da uno degli elementi in $\mathcal{S}_x$ o $\mathcal{S}_y$

1.1 Ricerca di componenti connesse in grafi

Un possibile utilizzo di queste strutture dati è nella ricerca di componenti connesse in un **grafo non diretto** (o non orientato), ovvero quando i relativi archi non hanno direzione e pertanto collegano due nodi bidirezionalmente, e questo verrà utilizzato per confrontare diverse implementazioni della struttura dati.

Dato $G(V, E)$ un grafo non diretto, con V insieme dei nodi del grafo e E l'insieme degli archi, l'algoritmo è il seguente:

```
1. for v ∈ G.V do
2.     MAKE-SET(v)
3. for (u, v) ∈ G.E do
4.     if FIND-SET(u) ≠ FIND-SET(v) then
5.         UNION(u, v)
```

Nota sulla complessità computazionale
Essendo il grafo $G(V, E)$ non diretto si ha che

$$\max(|E|) = \frac{n(n-1)}{2} = \Theta(n^2) \text{ con } n = |V| \tag{1}$$

dunque il massimo numero di archi è minore rispetto al suo equivalente per un grafo diretto.

1.2 Implementazioni della struttura dati

Come riportato precedentemente, esistono varie implementazioni delle strutture dati per insiemi disgiunti che permettono di avere variazioni sui costi di complessità computazionale.

1.2.1 Liste concatenate

Questa implementazione consiste nel rappresentare gli insiemi disgiunti tramite liste concatenate. Ogni lista ha due puntatori, uno all'elemento iniziale della lista e uno all'elemento finale, permettendo di ottenere in tempo costante il rappresentante di un insieme, e di aggiungere elementi ad esso sempre in tempo costante. Ogni elemento della lista contiene, oltre ai dati associati, anche un puntatore al suo rappresentante, così da ottenere in tempo costante il rappresentante di qualsiasi elemento, e un puntatore al successivo elemento della lista, come in qualsiasi lista concatenata.

I costi computazionali delle operazioni base nei casi peggiori sono i seguenti:

- MAKE-SET(x) $\rightarrow O(1)$: dato che il costo di creazione di una nuova lista vuota è costante, e per quanto detto nel paragrafo precedente anche l'inserimento ha costo costante;
- FIND-SET(x) $\rightarrow O(1)$: per via del puntatore al rappresentante già presente all'interno di qualsiasi elemento;
- UNION(x, y) $\rightarrow O(n)$ con n numero di elementi totali nella struttura dati: infatti, supponendo di concatenare tutti gli elementi della lista contenente y alla lista contenente x, abbiamo il caso peggiore quando la lista di x contiene solo x stesso, e la lista di y contiene i rimanenti $n-1$ elementi; dato che l'unione prevede di cambiare il puntatore al rappresentante per ogni elemento inserito nella lista finale, questa operazione ha tempo lineare, giustificando il risultato.

Il costo computazionale nel caso peggiore tiene conto del fatto che per n elementi inseriti all'interno della struttura dati potranno essere eseguite massimo $n-1$ UNION il cui numero di operazioni interne tenderà man mano verso $n-1$, quindi si considera il costo pari a:

$$\Theta(n^2)^1 \tag{2}$$

Considerando il problema della ricerca di componenti connesse su un grafo non diretto $G(V, E)$, questo è composto da un ciclo *for* alla riga 1 che esegue $|V|$ operazioni di MAKE-SET, quindi crea $|V|$ elementi. Considerando (2), il costo totale è dato da:

$$\Theta(v^2) \tag{3}$$

con $v = |V|$. Se considero il caso con numero massimo di archi possibile (1), il risultato non varia.

[1] Thomas H. Cormen et al. "Introduction to Algorithms". In: Fourth. Cambridge, Massachusetts: The MIT Press, 2022, p. 689

1.2.2 Liste concatenate con euristica dell'unione pesata

Questa implementazione riprende quella delle normali liste concatenate e cerca di ridurre il costo dell'operazione di UNION conservando nelle liste un parametro che indica il numero di elementi contenuti in esse e questo viene usato per unire la lista più corta a quella più lunga, riducendo il numero di operazioni. Inoltre, la gestione di un contatore di elementi all'interno di ogni lista ha costo costante, quindi i costi per le rimanenti operazioni continuano ad essere i medesimi per le liste concatenate.

In generale, data una sequenza di m operazioni di MAKE-SET, FIND-SET and UNION, n delle quali sono operazioni di MAKE-SET (i.e., n è il numero di elementi totali all'interno della struttura dati), il costo computazionale è dato da:

$$O(m + n \log_2 n)^2 \tag{4}$$

Analizzando il costo della struttura dati nell'algoritmo di ricerca di componenti connessi su $G(V, E)$ grafo non diretto, si osserva che: n, il numero di MAKE-SET è dato dal numero di iterazioni del ciclo *for* alla riga 1; m è invece dato dalla somma di n con il numero di FIND-SET e UNION del ciclo *for* alla riga 3, moltiplicate per il numero di iterazioni del ciclo stesso:

$$\begin{aligned} n &= |V| \\ m &= n + |E| \cdot (2 + 1) = |V| + 3|E| \end{aligned} \tag{5}$$

Quindi, per (4), e indicando con $v = |V|$ e $e = |E|$, il costo complessivo è dato da:

$$\begin{aligned} O(m + n \log_2 n) &= O((v + 3e) + v \log_2 v) \\ &= O(v + 3e + v \log_2 v) \\ &= O(3e + v \log_2 v) \\ &= O(e + v \log_2 v) \end{aligned} \tag{6}$$

Considerando il caso peggiore in cui il numero di archi è massimo, per (1) si ha:

$$\begin{aligned} e &= \Theta(v^2) \\ O(e + v \log_2 v) &= O(v^2 + v \log_2 v) = O(v^2) \end{aligned} \tag{7}$$

quindi si ha un costo quadratico.

[2] Thomas H. Cormen et al. "Introduction to Algorithms". In: Fourth. Cambridge, Massachusetts: The MIT Press, 2022, pp. 689–690

1.2.3 Foreste di insiemi disgiunti con compressione dei cammini

Questa implementazione prevede di rappresentare ogni elemento di ogni insieme disgiunto come nodo di un albero, quindi è composto dal dato che contiene e da un puntatore al nodo padre. Ogni insieme disgiunto è quindi rappresentato da un albero la cui radice diventa automaticamente il rappresentante dell'insieme. Nella versione base, ovvero senza compressione dei cammini, i costi delle operazioni base sono i seguenti:

- MAKE-SET$(x) \to O(1)$: prevede di creare un nuovo nodo senza padre;
- FIND-SET$(x) \to O(h_x)$ con h_x altezza dell'albero in cui è contenuto x: questo perchè per individuare il rappresentante del nodo è necessario risalire tutto l'albero fino a giungere alla radice;
- UNION$(x,\ y) \to O(h_y)$ supponendo di porre x come padre del rappresentante di y: infatti, per unire due alberi (o i due insiemi) basta porre come padre di uno dei due rappresentanti l'altro elemento, e automaticamente il rappresentante degli elementi dell'albero unito diviene il rappresentante dell'albero espanso.

La compressione del cammino cerca di attenuare i tempi dell'operazione di FIND-SET (che viene utilizzata anche in UNION per individuare il rappresentante sul quale cambiare padre), andando a ridurre i cammini dai singoli nodi verso la radice dell'albero. Ciò viene fatto nell'operazione di FIND-SET che diventa una funzione ricorsiva: man mano che si risale l'albero i nodi attraversati vengono salvati e una volta raggiunta la radice i padri dei nodi salvati vengono aggiornati con la radice individuata.

Con la compressione dei cammini, la struttura dati ha un costo computazionale pari a:

$$\Theta(n + f \cdot (1 + \log_{2+\frac{f}{n}} n)^3 \tag{8}$$

con f il numero di operazioni FIND-SET e n il numero di operazioni UNION. Nel caso della ricerca di componenti connesse n è dato dal numero delle iterazioni del ciclo *for* alla riga 1, mentre f dipende dal numero di iterazioni del ciclo *for* alla riga 3, e considerando anche la FIND-SET implicita in UNION, abbiamo:

$$\begin{aligned} n &= |V| \\ f &= 3|E| \end{aligned} \tag{9}$$

Quindi, per (8), il costo totale è dato da:

$$\begin{aligned} \Theta(n + f \cdot (1 + \log_{2+\frac{f}{n}} n)) &= \Theta(v + 3e(1 + \log_{2+\frac{3e}{v}} v)) \\ &= \Theta(v + 3e + 3e \log_{2+\frac{3e}{v}} v) \\ &= O(v + 3e \log_{2+\frac{3e}{v}} v) \end{aligned} \tag{10}$$

[3]Thomas H. Cormen et al. "Introduction to Algorithms". In: Fourth. Cambridge, Massachusetts: The MIT Press, 2022, p. 696

con $v = |V|$ e $e = |E|$.
Nel caso peggiore, con il numero massimo di archi, si ha:

$$\begin{aligned} e &= \Theta(v^2) \\ O(v + 3e \log_{2+\frac{3e}{v}} v) &= O(v + 3v^2 \log_{2+\frac{3v^2}{v}} v) \\ &= O(v + v^2 \log_2 2 + 3vv) \\ &= O(v^2 \log_{2+3v} v) \end{aligned} \tag{11}$$

2 Documentazione del codice

Il codice per testare e confrontare le varie implementazioni della struttura dati è contenuta nel modulo `disjoint_sets`