Relazione della prova pratica

Laboratorio di Algoritmi

Giovanni Casano

Emanuele Castronovo

Progetto di Rete di Trasporti Minimale per Fantasilandia

Dipartimento di matematica e informatica
Università degli Studi di Palermo
Palermo
A.a. 2023/24

Indice

Capitolo 1

Introduzione

Nel paese di Fantasilandia ci sono n città ma nessuna strada. Il governo ha pianificato di costruire strade e ferrovie bidirezionali per collegare tutte le città attraverso un nuovo sistema di trasporto. Le strade verranno costruite solo tra città della stessa regione, mentre le ferrovie verranno utilizzate per collegare città situate in regioni diverse. Due città si trovano nella stessa regione se la distanza tra loro è al più r. L'obiettivo è minimizzare i costi di costruzione costruendo solo la minima estensione necessaria in modo che ci sia un percorso tra tutte le città del paese. Questo documento descrive la soluzione proposta per determinare il sistema di rete di trasporti minimale.

Capitolo 2

Descrizione del Problema

2.1 Input

L'input del problema è costituito da:

- Un intero n che rappresenta il numero di città, con $1 \leq n \leq 100$.
- Un intero r che determina se due città sono nella stessa regione, con $0 \leq r \leq 4000$.
- n righe, ciascuna contenente due interi x e y che rappresentano le coordinate delle città, con $0 \leq x, y \leq 1000$.

2.2 Output

L'output deve fornire tre valori separati da uno spazio:

1. Il numero di regioni presenti nel paese di Fantasilandia.
2. L'estensione minima delle strade, arrotondata all'intero più vicino.
3. L'estensione minima delle ferrovie.

Capitolo 3

Soluzione Proposta

La soluzione proposta dal nostro team si basa su diversi passaggi, descritti nei seguenti paragrafi.

3.1 Calcolo delle Distanze

Per ogni coppia di città, calcoliamo la distanza euclidea. La distanza tra due città con coordinate (x_1, y_1) e (x_2, y_2) è data dalla formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Questa distanza ci permette di determinare se due città appartengono alla stessa regione (se $d \leq r$) oppure a regioni diverse (se $d > r$).

3.2 Costruzione del Grafo

Costruiamo un grafo non orientato dove ogni nodo rappresenta una città e ogni arco rappresenta la distanza tra due città. Gli archi sono pesati in base alla distanza euclidea calcolata.

3.3 Lettura dei Dati dal File

Per costruire i vertici e gli archi pesati, è stato necessario leggere un file in input ("cities.txt") ed interpretare correttamente le righe in esso contenute. Il file contiene:

- La prima riga con due numeri: il numero di città n e il valore di r.
- Le successive n righe con le coordinate x e y delle città.

Il costruttore della classe 'Progetto' legge i dati dal file come segue:

- Apre il file e legge la prima riga per ottenere i valori di n e r utilizzando un 'istringstream'.
- Legge le successive righe per ottenere le coordinate delle città. Per ogni coppia di coordinate, cioè per ogni riga letta:
  - Calcola la distanza euclidea tra le città leggendo le righe che si trovano a seguire.
  - Aggiunge un arco pesato al grafo con questa distanza.

3.4 Algoritmo di Kruskal

Utilizziamo l'algoritmo di Kruskal per calcolare l'Albero Ricoprente Minimo (MST) del grafo. L'algoritmo di Kruskal è adatto per trovare l'MST in quanto:

- Ordina gli archi del grafo in ordine crescente di peso.
- Aggiunge gli archi all'MST, assicurando che non si formi alcun ciclo.
- Utilizza una struttura dati Union-Find per gestire le componenti connesse.

3.5 Struttura Union-Find

La struttura Union-Find (o Disjoint Set Union, DSU) con implementazione Quick-Union è utilizzata per:

- Unire due insiemi (o componenti connesse).
- Trovare il rappresentante di un insieme.

Questa struttura dati supporta efficientemente le operazioni di unione e di ricerca, essenziali per l'algoritmo di Kruskal. Inoltre, la struttura Union-Find è implementata considerando il rank degli alberi uniti, per mantenere l'altezza degli alberi bassa e ottimizzare ulteriormente le operazioni.

3.6 Determinazione delle Strade e delle Ferrovie

Durante la costruzione dell'MST:

- Sommiamo le lunghezze degli archi con peso inferiore o uguale a r per ottenere l'estensione totale delle strade necessarie per collegare le città della stessa regione.
- Gli archi con peso inferiore o uguale a r rappresentano le strade.
- Gli archi con peso maggiore di r rappresentano le ferrovie.

Enumeriamo tutti gli archi di peso maggiore di r per determinare il numero di regioni, che corrisponde al numero di ferrovie più uno. Tra questi archi, identifichiamo l'arco di costo minimo.

Capitolo 4

Correttezza della soluzione proposta

La correttezza della soluzione proposta si basa sulla correttezza dell'algoritmo di Kruskal, la quale a sua volta si basa su due proprietà fondamentali: la proprietà di taglio e la proprietà di ciclo.

4.1 Proprietà di Taglio

La proprietà di taglio afferma che, dato un insieme S di vertici in un grafo connesso e un arco (u, v) di peso minimo che attraversa il taglio $(S, V \setminus S)$, dove $u \in S$ e $v \in V \setminus S$, allora l'arco (u, v) appartiene all'albero ricoprente minimo (MST).

Dimostrazione: Consideriamo l'MST T e supponiamo per assurdo che l'arco (u, v) non appartenga a T. Aggiungendo (u, v) a T si forma un ciclo. Poiché (u, v) è l'arco di peso minimo che attraversa il taglio $(S, V \setminus S)$, sostituire l'arco più pesante nel ciclo con (u, v) produrrebbe un albero di costo minore, contraddicendo il fatto che T è l'MST. Quindi, (u, v) deve appartenere a T.

4.2 Proprietà di Ciclo

La proprietà di ciclo afferma che, dato un ciclo C in un grafo connesso e un arco (u, v) di peso massimo in C, rimuovendo (u, v) non si aumenta il costo dell'albero ricoprente minimo (MST).

Dimostrazione: Consideriamo l'MST T e supponiamo che T contenga un ciclo C. Rimuovendo l'arco (u, v) di peso massimo da C, il grafo rimane connesso e la somma dei pesi degli archi nel nuovo grafo è minore o uguale a quella di T. Poiché (u, v) è l'arco di peso massimo in C, nessun arco aggiunto successivamente potrà

ridurre ulteriormente il peso dell'MST, garantendo così che il nuovo albero è ancora l'MST.

4.3 Correttezza dell'Algoritmo di Kruskal

L'algoritmo di Kruskal applica queste proprietà come segue:

- Ordina tutti gli archi del grafo in ordine crescente di peso.
- Aggiunge gli archi uno alla volta all'MST, rispettando la proprietà di ciclo per evitare la formazione di cicli.
- Utilizza la proprietà di taglio per garantire che ogni arco aggiunto mantenga la correttezza dell'MST.

Poiché ogni passo dell'algoritmo di Kruskal è basato sulle proprietà di taglio e di ciclo, l'algoritmo trova correttamente l'albero ricoprente minimo del grafo.

Capitolo 5

Strutture Dati Utilizzate

In questo progetto sono state utilizzate diverse strutture dati per risolvere il problema della rete di trasporti minimale per Fantasilandia. Le principali strutture dati utilizzate sono `list`, `unordered_map`, e `vector`. Di seguito, descriviamo l'uso specifico di ciascuna di queste strutture e il motivo della loro scelta.

5.1 List

La struttura `list` è stata utilizzata per memorizzare gli archi del grafo e per costruire l'Albero Ricoprente Minimo (MST) utilizzando l'algoritmo di Kruskal. Le liste sono particolarmente adatte per operazioni di inserimento, rimozione e accesso sequenziale, che sono frequenti nel processo di gestione degli archi durante l'algoritmo.

- **Archiviazione degli archi**: Una lista di archi è utilizzata per memorizzare tutti gli archi del grafo, inclusi quelli che compongono l'MST.
- **Efficienza nelle operazioni**: La scelta di una lista per gli archi permette un'efficiente inserimento e rimozione durante la costruzione dell'MST, in quanto l'algoritmo di Kruskal richiede frequenti operazioni di aggiunta e controllo degli archi.

5.2 Unordered_Map

La struttura `unordered_map` è stata utilizzata per gestire le relazioni tra città e le loro distanze, nonché per indicizzare le città stesse. Le mappe non ordinate offrono un accesso mediamente costante, rendendo efficienti le operazioni di ricerca e aggiornamento.

- **Indicizzazione delle città**: Le città vengono indicizzate in modo efficiente usando una `unordered_map`, permettendo di mappare le coordinate delle città agli indici corrispondenti.
- **Memorizzazione delle distanze**: Le distanze tra le città vengono memorizzate in una `unordered_map`, consentendo un rapido accesso e aggiornamento delle distanze durante la costruzione del grafo.

5.3 Vector

La struttura `vector` è stata utilizzata nella classe `QuickUnionUF` per implementare la struttura dati Union-Find, essenziale per l'algoritmo di Kruskal. I `vector` offrono un accesso casuale costante e un'efficiente gestione della memoria dinamica.

- **Union-Find**: Nella struttura Union-Find, i `vector` vengono utilizzati per memorizzare i nodi genitori e i ranghi dei nodi. Questo permette di implementare le operazioni di unione e ricerca in modo efficiente.
- **Operazioni efficienti**: L'uso dei `vector` permette di gestire in modo efficiente le operazioni di compressione dei percorsi e unione per rango, migliorando le prestazioni complessive dell'algoritmo di Kruskal.

5.4 Analisi delle Scelte

La scelta delle strutture dati è stata guidata dalla necessità di bilanciare l'efficienza delle operazioni di accesso, inserimento e gestione della memoria. Le `list` sono state preferite per la gestione degli archi a causa delle frequenti operazioni di inserimento e rimozione. Le `unordered_map` sono state scelte per le operazioni di ricerca e aggiornamento efficienti, cruciali per gestire le città e le loro distanze. I `vector` sono stati utilizzati nella struttura Union-Find per garantire operazioni efficienti di unione e ricerca.

In sintesi, l'uso combinato di `list`, `unordered_map` e `vector` ha permesso di implementare una soluzione efficiente e scalabile per il problema della rete di trasporti minimale in Fantasilandia.

Capitolo 6

Analisi del Costo Computazionale

L'analisi del costo computazionale del programma si divide in due parti: la fase di pre-elaborazione e la fase di elaborazione. Di seguito vengono dettagliate entrambe le fasi, considerando i costi temporali e spaziali associati.

6.1 Fase di Pre-elaborazione

La fase di pre-elaborazione consiste nel calcolo delle distanze tra tutte le coppie di città e nella costruzione delle strutture dati necessarie per l'elaborazione successiva. Questa fase è implementata principalmente nel costruttore della classe `Progetto`.

Calcolo delle Distanze Per ogni coppia di città, viene calcolata la distanza euclidea. Il numero di coppie di città è dato dal coefficiente binomiale $\binom{n}{2}$, che corrisponde a $\frac{n(n-1)}{2}$. Poiché la distanza tra le città (x_i, y_i) e (x_j, y_j) è la stessa della distanza tra (x_j, y_j) e (x_i, y_i), viene calcolata una sola volta per ciascuna coppia.

Costruzione delle Strutture Dati Per ogni distanza calcolata, le città devono essere indicizzate. L'indicizzazione della città può essere fatta solo se quest'ultima non è presente nella mappa utilizzata come struttura dati di ausilio. Se presente, la città è stata mappata durante il calcolo di una distanza calcolata precedentemente e non deve essere nuovamente mappata. Questa verifica viene fatta con il metodo `find`. L'assenza di una relazione d'ordine per le coordinate di un piano euclideo, obbliga l'utilizzo di una mappa non ordinata: cioè una mappa le cui chiavi sono disposte in un albero sbilanciato. La diretta conseguenza è il costo **lineare** del metodo `find`, il quale esplora l'albero delle chiavi. Per la natura del problema, l'indicizzazione delle città è essenziale per avere un costo temporale efficiente durante l'elaborazione. Pertanto, non è possibile fare a meno della struttura dati **mappa**.

Costi totali

- **Costo temporale**:

$$\begin{aligned}
&\mathcal{O}\left(\frac{n(n-1)}{2}\right) && \text{(calcolo delle distanze)}\\
&+\mathcal{O}\left(\frac{n(n-1)}{2}\cdot n\right) && \text{(metodi find)}\\
&+\mathcal{O}(n) && \text{(riempimento mappa)}\\
&+\mathcal{O}\left(\frac{n(n-1)}{2}\right) && \text{(riempimento lista)}\\
&\simeq \mathcal{O}(n^3)
\end{aligned}$$

- **Costo spaziale**: Il costo spaziale durante questa fase include:

$$\mathcal{O}(n)\text{ (mappa)}+\mathcal{O}(n)\text{ (vertici)}+\mathcal{O}\left(\frac{n(n-1)}{2}\right)\text{ (lista)}+\mathcal{O}\left(\frac{n(n-1)}{2}\right)\text{ (archi)}$$

  Una volta creata la lista degli archi, la memoria utilizzata dalla mappa e dai vertici viene liberata. Pertanto, il costo spaziale finale è dominato dalla lista degli archi:

$$\mathcal{O}\left(\frac{n(n-1)}{2}\right)\simeq\mathcal{O}(n^2)\text{ (lista degli archi)}$$

6.2 Fase di Elaborazione

La fase di elaborazione comprende l'esecuzione dell'algoritmo di Kruskal per trovare l'albero ricoprente minimo (MST) del grafo.

Ordinamento degli Archi Gli archi vengono ordinati per peso, il che ha una complessità temporale di $\mathcal{O}(E\log E)$, dove E è il numero di archi. Nel nostro caso, $E=\frac{n(n-1)}{2}\simeq n^2$, quindi il costo è:

$$\mathcal{O}(n^2\log n^2)\simeq\mathcal{O}(n^2\log n)$$

Inizializzazione delle Strutture Union-Find Ogni città viene inizializzata come una singola struttura Union-Find, con un costo di $\mathcal{O}(1)$.

Unione e Ricerca degli Archi Gli archi vengono processati in ordine crescente di peso. Per ogni arco, viene eseguito due volte il metodo `uf.connected` per verificare se un nodo è presente nella Union-Find dell'altro e viceversa. Grazie all'implementazione con compressione dei percorsi, cioè al *balancing by size*, questo metodo ha un costo di $\mathcal{O}(\log n)$. Se i nodi non risiedono nella Union-Find viene eseguita l'operazione di unione (`uf.unite`), la quale ha un costo di $\mathcal{O}(1)$ considerando che la struttura Union-Find è di tipo ***quick union***.

Costi totali

- **Costo temporale**:

$$\begin{aligned}
&\mathcal{O}(n^2 \log n) && \text{(ordinamento degli archi)}\\
&+ \mathcal{O}(n) && \text{(strutture Union-Find)}\\
&+ \mathcal{O}(n^2 \cdot 2 \log n) && \text{(\texttt{uf.connected} eseguiti)}\\
&+ \mathcal{O}(n^2) && \text{(\texttt{uf.unite} eseguiti)}\\
&\simeq \mathcal{O}(n^2 \log n)
\end{aligned}$$

- **Costo spaziale**: il costo spaziale della fase di elaborazione è dato dallo spazio per le strutture Union-Find, che è $\mathcal{O}(n)$ più l'MST costruito $\mathcal{O}(n-1)$.

6.3 Riepilogo dei Costi

Costo Temporale Totale

$$\mathcal{O}(n^3) \text{ (pre-elaborazione)} + \mathcal{O}(n^2 \log n) \text{ (elaborazione)} \simeq \mathcal{O}(n^3)$$

Costo Spaziale Totale

$$\mathcal{O}(n^2) \text{ (lista degli archi e strutture Union-Find)} \quad +$$

$$+ \quad \mathcal{O}(n) \quad \text{(strutture Union-Find)} \quad +$$

$$+ \quad \mathcal{O}(n-1) \quad \text{(MST)}$$

Capitolo 7

Esempio di Calcolo

Consideriamo un esempio con $n = 4$ e $r = 20$. Le coordinate delle città sono:

- Città 1: $(0, 0)$
- Città 2: $(40, 30)$
- Città 3: $(30, 30)$
- Città 4: $(10, 10)$

Calcoliamo le distanze euclidee:

- Distanza tra Città 1 e Città 2: $\sqrt{(40-0)^2 + (30-0)^2} = 50$
- Distanza tra Città 1 e Città 3: $\sqrt{(30-0)^2 + (30-0)^2} = 42.43$
- Distanza tra Città 1 e Città 4: $\sqrt{(10-0)^2 + (10-0)^2} = 14.14$
- Distanza tra Città 2 e Città 3: $\sqrt{(30-40)^2 + (30-30)^2} = 10$
- Distanza tra Città 2 e Città 4: $\sqrt{(10-40)^2 + (10-30)^2} = 36.06$
- Distanza tra Città 3 e Città 4: $\sqrt{(10-30)^2 + (10-30)^2} = 28.28$

Costruiamo il grafo con gli archi pesati dalle distanze calcolate e applichiamo l'algoritmo di Kruskal per trovare l'MST (pesi arrotondati all'intero più piccolo):

- Aggiungiamo l'arco (2, 3) con peso 10 (strada).
- Aggiungiamo l'arco (1, 4) con peso 14 (strada).
- Aggiungiamo l'arco (3, 4) con peso 28 (ferrovia).

Le strade hanno un'estensione totale di 24 (arrotondato) e le ferrovie di 28.

Capitolo 8

Conclusione

Il metodo proposto permette di minimizzare i costi di costruzione del sistema di trasporti in Fantasilandia, garantendo al contempo la connessione di tutte le città. Le distanze euclidee sono utilizzate per determinare la suddivisione in regioni e la costruzione dell'Albero Ricoprente Minimo assicura che il percorso tra tutte le città sia ottimizzato. Questo approccio bilancia l'uso di strade e ferrovie in modo efficiente, riducendo al minimo i costi totali di costruzione.