Azzolini Riccardo 2019-05-15

# Algoritmi greedy

## 1 Sistema di indipendenza

Un **sistema di indipendenza** è una coppia $\langle E, F \rangle$ dove

- $E$ è un insieme finito;
- $F \subseteq 2^E$ è una famiglia di sottoinsiemi di $E$;
- $A \in F \wedge B \subseteq A \implies B \in F$, cioè per ogni $A$ appartenente alla famiglia $F$, anche tutti i sottoinsiemi di $A$ appartengono a $F$.

## 2 Problema di ottimizzazione

- *Input*: un sistema di indipendenza $\langle E, F \rangle$ e una funzione peso $w : E \to \mathbb{R}^+$ (che associa un peso reale non negativo a ogni elemento di $E$, e si può quindi estendere ai sottoinsiemi di $E$ semplicemente sommando i pesi degli elementi: $w(A) = \sum_{e \in A} w(e) \quad \forall A \subseteq E$).
- *Output*: $M \in F$ tale che $\forall A \in F, \quad w(A) \leq w(M)$, cioè il sottoinsieme di peso massimo (o, simmetricamente, minimo).

## 3 Procedura Greedy

Si può provare a risolvere un problema di ottimizzazione cercando a ogni passo di estendere una soluzione parziale mediante l'aggiunta di un elemento localmente ottimo (controllando che la soluzione così estesa appartenga ancora alla famiglia $F$).

```
Procedura Greedy(<E, F>, w) {
    set S = emptyset();
    while (!is_empty(E)) {
        m = max(E);
        E = E \ {m};
        if (member(S + {m}, F)) add(m, S);
    }
    return S;
```

```
}
```

1. La soluzione iniziale è l'insieme vuoto, che, per la definizione di sistema di indipendenza, appartiene sicuramente a $F$, dato che è sottoinsieme di tutti gli insiemi.
2. A ogni passo, si estrae l'elemento massimo corrente e si prova ad aggiungerlo alla soluzione. Se essa rimane valida, si ha una nuova soluzione, altrimenti l'elemento viene scartato.
3. La procedura termina dopo aver provato ad aggiungere tutti gli elementi di $E$. Siccome l'insieme $E$ è finito, `Greedy` termina e restituisce una soluzione in ogni caso, ma (in generale) non è garantito che essa sia la soluzione ottima.

Se gli elementi di E vengono ordinati all'inizio, la ricerca del massimo a ogni passaggio viene sostituita da una singola scansione dell'insieme ordinato:

```
Procedura Greedy(<E, F>, w) {
    set S = emptyset();
    sort(E);
    for (int i = 1; i <= #E; i++) {
        if (member(S + {E[i]}, F)) add(E[i], S);
    }
    return S;
}
```

### 3.1 Complessità

Se $n = \#E$, allora

- `sort(E)` ha costo $O(n \log n)$;
- il ciclo esegue $n$ iterazioni, ciascuna delle quali è composta principalmente da
  - `member(x, F)`, con costo $C(n)$, dove $C$ è una qualche funzione che dipende dall'implementazione della struttura dati usata per rappresentare $F$;
  - `add(y, S)`, di costo $O(1)$ (ad esempio, se l'insieme è rappresentato mediante una lista concatenata).

Complessivamente, quindi, il costo di `Greedy` (nella versione che ordina gli elementi di $E$) è $O(n \log n + nC(n))$.

# 4 Matroide

Un **matroide** è un sistema di indipendenza $\langle E, F\rangle$ tale che

$$A, B \in F,\ |B| = |A| + 1 \implies \exists b \in B \setminus A,\ A \cup \{b\} \in F$$

(cioè tale che si possa sicuramente "far crescere" l'insieme $A$ prendendo un nuovo elemento da $B$, se esiste un $B$ con un numero di elementi maggiore di 1 rispetto ad $A$).

## 4.1 Teorema di Rado

*Teorema*: Sia $\langle E, F\rangle$ un sistema di indipendenza. Allora, esso è un matroide se e solo se, per ogni funzione peso $w : E \to \mathbb{R}^+$, Greedy($\langle E, F\rangle, w$) fornisce la soluzione ottima.

# 5 Matroide grafico

Dati un grafo non orientato $G = \langle V, E\rangle$, e la famiglia di sottoinsiemi di lati

$$F = \{X \subseteq E \mid X \text{ non forma cicli in } G\}$$

$\langle E, F\rangle$ è un sistema di indipendenza (perché se i lati in $X$ non formano cicli, allora nessun sottoinsieme di lati di $X$ può formare cicli), e in particolare un matroide, il **matroide grafico**.

# 6 Minimum Spanning Tree

*Problema*: Calcolo dell'**albero di copertura di peso minimo** (**Minimum Spanning Tree**, MST).

- *Input*: un grafo pesato (cioè nel quale a ogni lato è associato un peso), non orientato e connesso $G = \langle V, E\rangle$ e una funzione peso $w : E \to \mathbb{R}^+$.
- *Output*: $T = \langle V', E'\rangle$, con $V' = V$ ed $E' \subseteq E$, tale che
  - $T$ è un albero di copertura per $G$;
  - il peso $w(T) = \sum_{l \in E'} w(l)$ è minimo.

Poiché si tratta di un problema di ottimizzazione, e la soluzione da calcolare è un albero, cioè un grafo (connesso) senza cicli, si possono sfruttare il matroide grafico e la procedura Greedy.

## 6.1 Algoritmo di Kruskal

L'**algoritmo di Kruskal** è semplicemente una specializzazione della procedura Greedy per il calcolo dell'MST.

```
Procedura Kruskal(<V, E>, w) {
    set S = emptyset();
    partition P = {{v1}, {v2}, ..., {vn}};
    while (#P != 1) {
        (u, v) = min(E);
        delete((u, v), E);
        x = find(u, P);
        y = find(v, P);
        if (x != y) {
            union(x, y, P);
            add((u, v), S);
        }
    }
}
```

- Si usa una partizione di $V$ per tenere traccia delle componenti connesse dell'albero di copertura che si sta costruendo. Inizialmente si ha la partizione identità, dato che nessun vertice è connesso ad altri dall'albero. L'obiettivo è una partizione con una sola classe, che indica che l'albero collega effettivamente tutti i vertici.
- Per ottimizzare la ricerca del lato di peso minimo, si può rappresentare l'insieme $E$ con una lista ordinata, oppure sotto forma di *heap rovesciato* (uno heap con i valori minori più in alto, detto anche *orientato al minimo*).
- Se il lato di peso minimo collega due vertici della stessa componente connessa, (ovvero `find` restituisce per entrambi lo stesso rappresentante), significa che c'è già un percorso tra di essi nell'albero, quindi tale creerebbe un ciclo e viene allora scartato. Altrimenti, si aggiunge il lato all'albero, unendo due componenti connesse che prima erano separate (`union`).

  Il controllo che non si creino cicli corrisponde al controllo che la nuova soluzione rimanga nella famiglia $F$.

### 6.1.1 Complessità

Siano $n = \#V$ e $m = \#E$ (con $m = \Omega(n)$ perché il grafo in input è connesso).

- La creazione della partizione `P` ha costo $\Theta(n)$.
- Il numero di iterazioni del ciclo è $O(m)$:

  - $n - 1$ nel caso migliore, perché l'albero di copertura ha $n - 1$ lati;
  - $m$ nel caso peggiore, perché al massimo è necessario provare ad aggiungere all'albero tutti i lati esistenti.

- `add` ha costo $O(1)$.
- `union` e `find` hanno costo $O(\log n)$.
- `min` ha costo $O(1)$, se $E$ viene rappresentato con una lista ordinata o uno heap rovesciato.
- La costruzione della rappresentazione di $E$ e l'operazione `delete` hanno un costo che dipende dalla struttura dati utilizzata:
  - Se si sceglie una lista ordinata, si ha un costo iniziale di ordinamento di $O(m \log m)$ (con un algoritmo ottimale), e poi ogni `delete` ha costo $O(1)$.
  - Se, invece, si utilizza uno heap rovesciato, la costruzione (bottom-up) ha costo $\Theta(m)$, ma in compenso `delete` ha costo $O(\log m)$.

  Il costo totale della costruzione e di tutte le cancellazioni è quindi $O(m \log m)$ per entrambe le strutture.

Il costo complessivo è quindi $O(m \log m)$.