# Algoritmi e Strutture Dati
# La Struttura Dati Partizioni di A e le operazioni Union Find

P. Massazza[1]

[1]Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese Italy

# Outline

# Outline

# Partizioni

## Definizione [Partizione]

Una partizione di un insieme $A$ è una famiglia $\{A_1, \ldots, A_m\}$ t.c.

- $A_i \subseteq A$;
- $\forall i \neq j \; A_i \cap A_j = \emptyset$;
- $\bigcup_{i=1..m} A_i = A$

Ricordiamo che ogni partizione $P = \{A_1, \ldots, A_m\}$ identifica una relazione di equivalenza $R_P$:

$$\forall x, y \in A, \quad x R_P y \quad \text{sse} \quad \exists i, \; x, y \in A_i.$$

Possiamo quindi scegliere un elemento rappresentativo $a_i \in A_i$ per cui $[a_i] = A_i$, e rappresentare $P$ come tupla

$$P \equiv (a_1 \ldots, a_m).$$

# Partizioni

## Definizione [Partizione]

Una partizione di un insieme $A$ è una famiglia $\{A_1, \ldots, A_m\}$ t.c.

- $A_i \subseteq A$;
- $\forall i \neq j \ A_i \cap A_j = \emptyset$;
- $\bigcup_{i=1..m} A_i = A$

Ricordiamo che ogni partizione $P = \{A_1, \ldots, A_m\}$ identifica una relazione di equivalenza $R_P$:

$$\forall x, y \in A, \quad xR_Py \quad \text{sse} \quad \exists i, \ x, y \in A_i.$$

Possiamo quindi scegliere un elemento rappresentativo $a_i \in A_i$ per cui $[a_i] = A_i$, e rappresentare $P$ come tupla

$$P \equiv (a_1 \ldots, a_m).$$

# Algebra eterogenea Partizioni di A

## Esempio

Partizioni di $A = \{a_1, \ldots, a_m\}$, $\mathrm{PA} = \langle [A, P(A)], [\mathrm{Union}, \mathrm{Find}] \rangle$, dove

- $P(A)$ è l'insieme di tutte le partizioni su $A$;
- $\mathrm{Union} : A \times A \times P(A) \mapsto P(A)$,
  $$\mathrm{Union}(x, y, P) = (P \setminus \{[x], [y]\}) \cup \{[x] \cup [y]\};$$
- $\mathrm{Find} : A \times P(A) \mapsto A$,
  $$\mathrm{Find}(x, P) = y \quad \text{sse} \quad y = \mathrm{Rappr}([x]).$$

Si noti che $\mathrm{Find}(x_1, P) = \mathrm{Find}(x_2, P)$ sse $x_1 R_P x_2$.

# Algebra eterogenea Partizioni di A

## Esempio

Partizioni di $A = \{a_1, \ldots, a_m\}$, $\mathrm{PA} = \langle [A, P(A)], [\mathrm{Union}, \mathrm{Find}] \rangle$, dove

- $P(A)$ è l'insieme di tutte le partizioni su $A$;
- Union : $A \times A \times P(A) \mapsto P(A)$,
  $$\mathrm{Union}(x, y, P) = (P \setminus \{[x], [y]\}) \cup \{[x] \cup [y]\};$$
- Find : $A \times P(A) \mapsto A$,
  $$\mathrm{Find}(x, P) = y \quad \text{sse} \quad y = \mathrm{Rappr}([x]).$$

Si noti che $\mathrm{Find}(x_1, P) = \mathrm{Find}(x_2, P)$ sse $x_1 R_P x_2$.

# Algebra eterogenea Partizioni di A

## Esempio

Partizioni di $A = \{a_1, \ldots, a_m\}$, PA $= \langle [A, P(A)], [\text{Union}, \text{Find}] \rangle$, dove

- $P(A)$ è l'insieme di tutte le partizioni su $A$;
- Union : $A \times A \times P(A) \mapsto P(A)$,
  $$\text{Union}(x, y, P) = (P \setminus \{[x], [y]\}) \cup \{[x] \cup [y]\};$$
- Find : $A \times P(A) \mapsto A$,
  $$\text{Find}(x, P) = y \quad \text{sse} \quad y = \text{Rappr}([x]).$$

Si noti che $\text{Find}(x_1, P) = \text{Find}(x_2, P)$ sse $x_1 R_P x_2$.

# Algebra eterogenea Partizioni di A

## Esempio

Partizioni di $A = \{a_1, \ldots, a_m\}$, $\text{PA} = \langle [A, P(A)], [\text{Union}, \text{Find}] \rangle$, dove

- $P(A)$ è l'insieme di tutte le partizioni su $A$;
- $\text{Union} : A \times A \times P(A) \mapsto P(A)$,
  $$\text{Union}(x, y, P) = (P \setminus \{[x], [y]\}) \cup \{[x] \cup [y]\};$$
- $\text{Find} : A \times P(A) \mapsto A$,
  $$\text{Find}(x, P) = y \quad \text{sse} \quad y = \text{Rappr}([x]).$$

Si noti che $\text{Find}(x_1, P) = \text{Find}(x_2, P)$ sse $x_1 R_P x_2$.

# Algebra eterogenea Partizioni di A

## Esempio

Partizioni di $A = \{a_1, \ldots, a_m\}$, PA $= \langle [A, P(A)], [\text{Union}, \text{Find}] \rangle$, dove

- $P(A)$ è l'insieme di tutte le partizioni su $A$;
- Union : $A \times A \times P(A) \mapsto P(A)$,
  $$\text{Union}(x, y, P) = (P \setminus \{[x], [y]\}) \cup \{[x] \cup [y]\};$$
- Find : $A \times P(A) \mapsto A$,
  $$\text{Find}(x, P) = y \quad \text{sse} \quad y = \text{Rappr}([x]).$$

Si noti che $\text{Find}(x_1, P) = \text{Find}(x_2, P)$ sse $x_1 R_P x_2$.

# Algebra eterogenea Partizioni di A

Studiamo possibili implementazioni per la struttura dati PA

- Liste concatenate
  - Vantaggi: semplicità
  - Svantaggi: costo medio di ciascuna operazione $O(|A|)$
- Foreste di alberi
  - Vantaggi: costo di ciascuna operazione $O(\log |A|)$ se si effettua il bilanciamento degli alberi
  - Svantaggi: nessuno

# Algebra eterogenea Partizioni di A

Studiamo possibili implementazioni per la struttura dati PA

- Liste concatenate
  - **Vantaggi:** semplicità
  - **Svantaggi:** costo medio di ciascuna operazione $O(|A|)$
- Foreste di alberi
  - **Vantaggi:** costo di ciascuna operazione $O(\log |A|)$ se si effettua il bilanciamento degli alberi
  - **Svantaggi:** nessuno

# Algebra eterogenea Partizioni di A

Studiamo possibili implementazioni per la struttura dati PA

- Liste concatenate

  **Vantaggi:** semplicità

  **Svantaggi:** costo medio di ciascuna operazione $O(|A|)$

- Foreste di alberi

  **Vantaggi:** costo di ciascuna operazione $O(\log |A|)$ se si effettua il bilanciamento degli alberi

  **Svantaggi:** nessuno

# Outline

# Partizioni e foreste di alberi

- Rappresentiamo ciascuna classe di equivalenza tramite un albero avente alla radice l'elemento rappresentativo.
- Rappresentiamo ciascun albero attraverso il vettore dei padri

**Esempio**

Dato $A = \{1, \ldots, 9\}$, $P = \{\{1, 3, \underline{7}\}, \{\underline{2}\}, \{4, 5, 6, \underline{8}, 9\}\}$ potrebbe essere rappresentata da

$$\text{padre} = [7, 2, 7, 8, 8, 8, 7, 8, 8]$$

Abbiamo quindi:

- $\text{Find}(5, P) = 8$
- $\text{Union}(1, 2, P) = \{\{1, 2, 3, \underline{7}\}, \{4, 5, 6, \underline{8}, 9\}\}$

# Partizioni e foreste di alberi

- Rappresentiamo ciascuna classe di equivalenza tramite un albero avente alla radice l'elemento rappresentativo.
- Rappresentiamo ciascun albero attraverso il vettore dei padri

## Esempio

Dato $A = \{1, \ldots, 9\}$, $P = \{\{1, 3, \underline{7}\}, \{\underline{2}\}, \{4, 5, 6, \underline{8}, 9\}\}$ potrebbe essere rappresentata da

$$\text{padre} = [7, 2, 7, 8, 8, 8, 7, 8, 8]$$

Abbiamo quindi:

- $\text{Find}(5, P) = 8$
- $\text{Union}(1, 2, P) = \{\{1, 2, 3, \underline{7}\}, \{4, 5, 6, \underline{8}, 9\}\}$

# Partizioni e foreste di alberi

- Rappresentiamo ciascuna classe di equivalenza tramite un albero avente alla radice l'elemento rappresentativo.
- Rappresentiamo ciascun albero attraverso il vettore dei padri

## Esempio

Dato $A = \{1, \ldots, 9\}$, $P = \{\{1, 3, \underline{7}\}, \{\underline{2}\}, \{4, 5, 6, \underline{8}, 9\}\}$ potrebbe essere rappresentata da

$$\text{padre} = [7, 2, 7, 8, 8, 8, 7, 8, 8]$$

Abbiamo quindi:

- $\text{Find}(5, P) = 8$
- $\text{Union}(1, 2, P) = \{\{1, 2, 3, \underline{7}\}, \{4, 5, 6, \underline{8}, 9\}\}$

# Partizioni e foreste di alberi

- Rappresentiamo ciascuna classe di equivalenza tramite un albero avente alla radice l'elemento rappresentativo.
- Rappresentiamo ciascun albero attraverso il vettore dei padri

## Esempio

Dato $A = \{1, \ldots, 9\}$, $P = \{\{1, 3, \underline{7}\}, \{\underline{2}\}, \{4, 5, 6, \underline{8}, 9\}\}$ potrebbe essere rappresentata da

$$\text{padre} = [7, 2, 7, 8, 8, 8, 7, 8, 8]$$

Abbiamo quindi:

- $\text{Find}(5, P) = 8$
- $\text{Union}(1, 2, P) = \{\{1, 2, 3, \underline{7}\}, \{4, 5, 6, \underline{8}, 9\}\}$

# Union e Find: implementazione

```
public class UnionFind{
 private int[] father;
 private count;
 public UnionFind(int n)
 {father=new int[n]; count=n;
  for(int i=0;i< n;i++) father[i]=i;}
 public int size(void){return count;}
 public int Find(int x)
 {while(father[x]!=x) x=father[x];
  return x;}
 public void Union(int x,int y)
 {int u=Find(x);
  int v=Find(y);
  if (u!=v) {father[v]=u;count--;}}
}
```

# Foreste con bilanciamento

L'implementazione mostrata non garantisce prestazioni $O(\log |A|)$ dato che potrebbe creare alberi degeneri.

## Esempio

Se father $= [0, 1, 2, \ldots, n-1]$ l'esecuzione del ciclo

```
for (i=0;i<p.size()-1;i++) p.Union(i+1,0)
```

porta all'albero degenere (lista)

$$\text{father} = [1, 2 \ldots, n-1, n-1]$$

Modifichiamo quindi l'implementazione di Union in modo che la scelta del rappresentante avvenga in funzione del numero di elementi associati. Usiamo a tal fine un vettore

$$\texttt{sz}[i] = k \quad \text{sse} \quad \texttt{father}[i] = i \wedge \sharp[i] = k.$$

# Foreste con bilanciamento

L'implementazione mostrata non garantisce prestazioni $O(\log |A|)$ dato che potrebbe creare alberi degeneri.

## Esempio

Se father $= [0, 1, 2, \ldots, n-1]$ l'esecuzione del ciclo

```
for (i=0;i<p.size()-1;i++) p.Union(i+1,0)
```

porta all'albero degenere (lista)

$$\text{father} = [1, 2 \ldots, n-1, n-1]$$

Modifichiamo quindi l'implementazione di Union in modo che la scelta del rappresentante avvenga in funzione del numero di elementi associati. Usiamo a tal fine un vettore

$$\texttt{sz}[i] = k \quad \text{sse} \quad \texttt{father}[i] = i \wedge \sharp[i] = k.$$

# Union e Find: implementazione

```
public class UnionFindB{
 private int[] father;private int[]sz;
 private count;
 public UnionFind(int n)
 {father=new int[n];count=n;sz=new int[n];
  for(int i=0;i< n;i++){father[i]=i; sz[i]=1;}}
 public int size(void); // come in UnionFind
 public int Find(int x); // come in UnionFind
 public void Union(int x,int y)
 {int u=Find(x);int v=Find(y);
  if (u==v) return;
  if(sz[u]<sz[v])
   {father[u]=v;sz[v]+=sz[u];}
  else{father[v]=u;sz[u]+=sz[v];}
  count--;}}
```

# Foreste con bilanciamento

**Teorema**

Partendo dalla partizione identità $[0, 1 \ldots, n-1]$ ed eseguendo unicamente Union con bilanciamento, si ottengono alberi con $k$ nodi e altezza non superiore a $\lfloor \log_2 k \rfloor$.

**Dimostrazione** (per induzione su $k$)

($k = 1$) Vero: $0 \leq \lfloor \log_2 1 \rfloor = 0$;

($k > 1$) $T$ ha $k$ nodi ed è il risultato di Union di due alberi $T_1, T_2$ con $|T_1| \leq |T_2|$ . Allora vale:

- $|T_1| \leq \lfloor k/2 \rfloor$, $h(T_1) \leq \lfloor \log_2 k \rfloor - 1$ (ind.);
- $|T_2| \leq k - 1$, $h(T_2) \leq \lfloor \log_2(k-1) \rfloor \leq \lfloor \log_2 k \rfloor$.

Osservando che $h(T) = \max\{h(T_2), h(T_1) + 1\}$ si ottiene $h(T) \leq \lfloor \log_2 k \rfloor$.

# Foreste con bilanciamento

## Teorema

Partendo dalla partizione identità $[0, 1 \ldots, n-1]$ ed eseguendo unicamente Union con bilanciamento, si ottengono alberi con $k$ nodi e altezza non superiore a $\lfloor \log_2 k \rfloor$.

Dimostrazione (per induzione su $k$)

$(k = 1)$ Vero: $0 \leq \lfloor \log_2 1 \rfloor = 0$;

$(k > 1)$ $T$ ha $k$ nodi ed è il risultato di Union di due alberi $T_1, T_2$ con $|T_1| \leq |T_2|$ . Allora vale:

- $|T_1| \leq \lfloor k/2 \rfloor$, $h(T_1) \leq \lfloor \log_2 k \rfloor - 1$ (ind.);
- $|T_2| \leq k - 1$, $h(T_2) \leq \lfloor \log_2 (k-1) \rfloor \leq \lfloor \log_2 k \rfloor$.

Osservando che $h(T) = \max\{h(T_2), h(T_1) + 1\}$ si ottiene $h(T) \leq \lfloor \log_2 k \rfloor$.

# Foreste con bilanciamento

**Teorema**

Partendo dalla partizione identità $[0, 1 \ldots, n-1]$ ed eseguendo unicamente Union con bilanciamento, si ottengono alberi con $k$ nodi e altezza non superiore a $\lfloor \log_2 k \rfloor$.

Dimostrazione (per induzione su $k$)

- $(k = 1)$ Vero: $0 \leq \lfloor \log_2 1 \rfloor = 0$;
- $(k > 1)$ $T$ ha $k$ nodi ed è il risultato di Union di due alberi $T_1, T_2$ con $|T_1| \leq |T_2|$ . Allora vale:
  - $|T_1| \leq \lfloor k/2 \rfloor$, $h(T_1) \leq \lfloor \log_2 k \rfloor - 1$ (ind.);
  - $|T_2| \leq k - 1$, $h(T_2) \leq \lfloor \log_2 (k-1) \rfloor \leq \lfloor \log_2 k \rfloor$.

  Osservando che $h(T) = \max\{h(T_2), h(T_1) + 1\}$ si ottiene $h(T) \leq \lfloor \log_2 k \rfloor$.

# Compressione dei cammini

Se si utilizza la Union con bilanciamento, l'esecuzione di $O(n)$ operazioni Union,Find ha costo $O(n \log n)$. Possiamo migliorare tale risultato ricorrendo alla compressione dei cammini.

**Idea:** Durante l'esecuzione di una Find facciamo in modo che tutti i nodi incontrati risalendo verso la radice diventino figli di questa.

```
public int Find(int x)
{Stack<Integer> s=new Stack<Integer>();int v;
 while(father[x]!=x)
  {s.Push(x);x=padre[x];}
 while(!Is.isEmpty())
  {v=s.Top();s.Pop();
   father[v]=x;}
 return x;
}
```

# Compressione dei cammini

Se si utilizza la Union con bilanciamento, l'esecuzione di $O(n)$ operazioni Union,Find ha costo $O(n \log n)$. Possiamo migliorare tale risultato ricorrendo alla compressione dei cammini.

**Idea:** Durante l'esecuzione di una Find facciamo in modo che tutti i nodi incontrati risalendo verso la radice diventino figli di questa.

```
public int Find(int x)
{Stack<Integer> s=new Stack<Integer>();int v;
 while(father[x]!=x)
  {s.Push(x);x=padre[x];}
 while(!Is.isEmpty())
  {v=s.Top();s.Pop();
   father[v]=x;}
 return x;
}
```

# Compressione dei cammini

Usando la compressione dei cammini la complessità di Union e Find viene a dipendere da

$$G(n) = \min\{k \in \mathbb{N} \mid n \leq F(k)\}$$

dove $F(k)$ (caso particolare della funzione di Ackermann) viene definita dall'eq. di ricorrenza

$$F(k) = 2^{F(k-1)}, \; F(0) = 1.$$

Nota: $G(n)$ cresce lentissimamente, $G(n) \leq 5$ per $n \leq 2^{65536}$.

**Teorema [Tarjan]**

L'esecuzione di $O(n)$ operazioni Union e Find (con bilanciamento e compressione dei cammini) ha costo

$$O(n \cdot G(n)).$$

# Compressione dei cammini

Usando la compressione dei cammini la complessità di Union e Find viene a dipendere da

$$G(n) = \min\{k \in \mathbb{N} \mid n \leq F(k)\}$$

dove $F(k)$ (caso particolare della funzione di Ackermann) viene definita dall'eq. di ricorrenza

$$F(k) = 2^{F(k-1)}, \; F(0) = 1.$$

Nota: $G(n)$ cresce lentissimamente, $G(n) \leq 5$ per $n \leq 2^{65536}$.

**Teorema [Tarjan]**

L'esecuzione di $O(n)$ operazioni Union e Find (con bilanciamento e compressione dei cammini) ha costo

$$O(n \cdot G(n)).$$

# Compressione dei cammini

Usando la compressione dei cammini la complessità di Union e Find viene a dipendere da

$$G(n) = \min\{k \in \mathbb{N} \mid n \leq F(k)\}$$

dove $F(k)$ (caso particolare della funzione di Ackermann) viene definita dall'eq. di ricorrenza

$$F(k) = 2^{F(k-1)}, \;\; F(0) = 1.$$

Nota: $G(n)$ cresce lentissimamente, $G(n) \leq 5$ per $n \leq 2^{65536}$.

**Teorema [Tarjan]**

L'esecuzione di $O(n)$ operazioni Union e Find (con bilanciamento e compressione dei cammini) ha costo

$$O(n \cdot G(n)).$$