Laboratorio di Algoritmi e Strutture Dati

Secondo esercizio, prima parte: insiemi disgiunti su liste concatenate (punti: 0)

Strutture dati astratte e loro implementazione

Implementare strutture dati astratte richiede nella maggioranza dei casi uno sforzo di **concretizzazione** non sempre banale. Inoltre le strutture dati sparse hanno la caratteristica di non essere visibili in maniera chiara durante l'implementazione e i test, che dunque richiedono l'uso di funzioni antagoniste di controllo più complesse di quelle viste fino adesso.

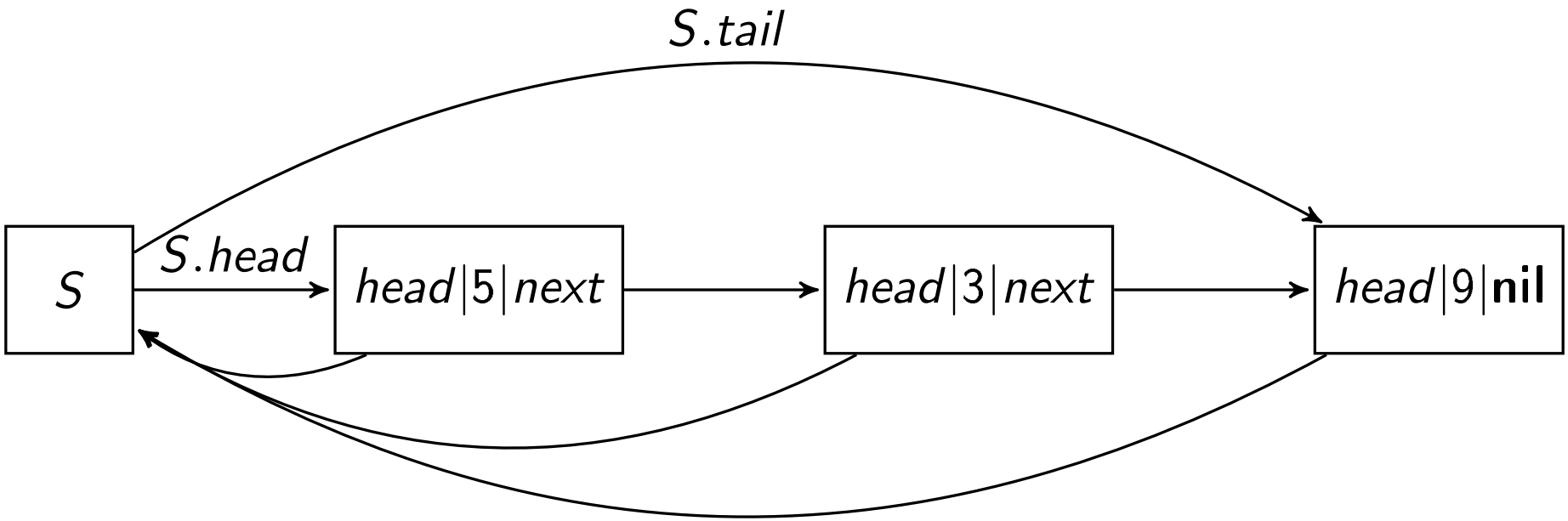
Una struttura dati è caratterizzata anche dalle funzioni che vi accedono, che ne costituiscono l'interfaccia verso l'esterno e ne determinano i comportamenti.

Insiemi disgiunti

Gli **insiemi disgiunti**, come visto nella teoria, sono una struttura dati che si interfaccia verso l'esterno solo attraverso *MakeSet*, *Union*, e *FindSet*. Come ripetutamente detto, non ha senso chiedersi quale sarebbe il comportamento di una struttura $\mathcal{S}$ per insiemi disgiunti di fronte alla richiesta di eseguire un'altra operazione (p.e., ordinare le chiavi di un insieme). In alcuni casi può essere utile fare questo tipo di esercizio, ma in generale le strutture sono ottimizzate per le operazioni per cui erano state progettate. Ricordiamo:

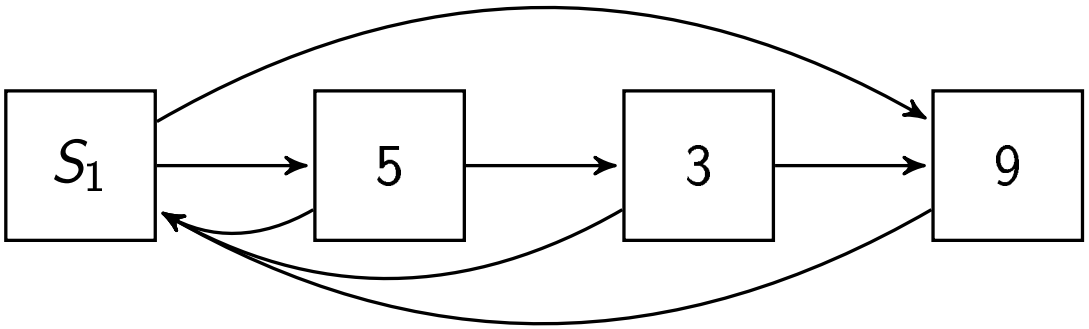
- *MakeSet*: costruisci un nuovo insieme singoletto che contiene solo l'elemento passato per parametro, e restituiscine l'indirizzo.
- *Union*: dati due insiemi, restiuisci un solo insieme che contiene gli elementi di entrambi.
- *FindSet*: dato un elemento, restituisci l'insieme che lo contiene.

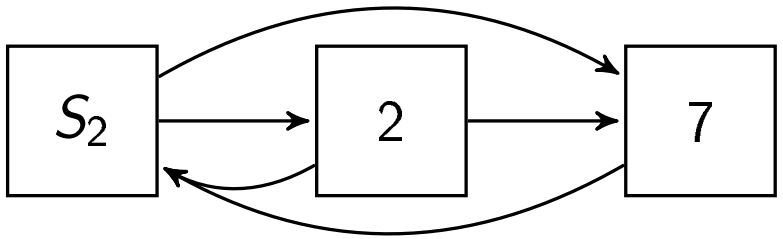
Insiemi disgiunti attraverso liste



Nella versione più semplice, un insieme disgiunto è una lista concatenata. Noi abbiamo implementato separando la loro testa dal corpo, anche se questo non è strettamente necessario da un punto di vista di efficienza o comodità; è necessario da un punto di vista di pulizia del codice e della semantica.

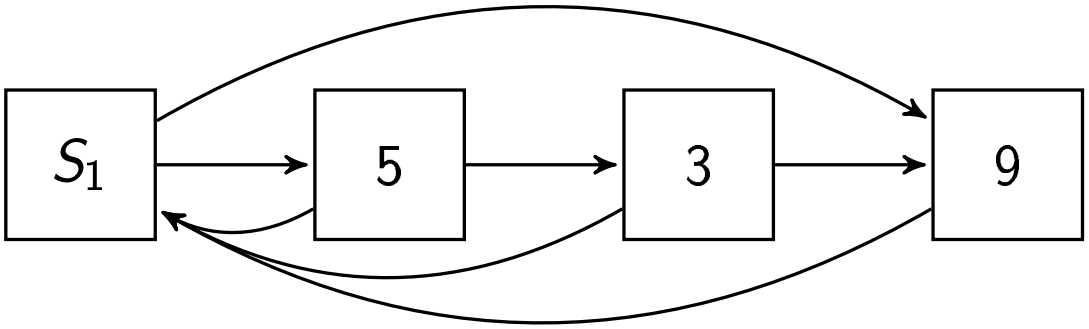
Insiemi disgiunti attraverso liste

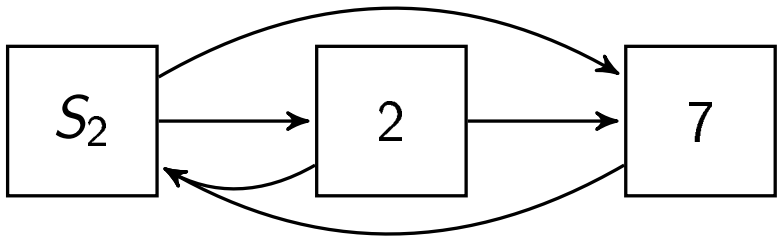




Quando ci sono più insiemi chiaramente ci sono più liste, tutte separate tra loro (in questo esempio $\mathcal{S} = \{S_1, S_2\}$).

Insiemi disgiunti attraverso liste

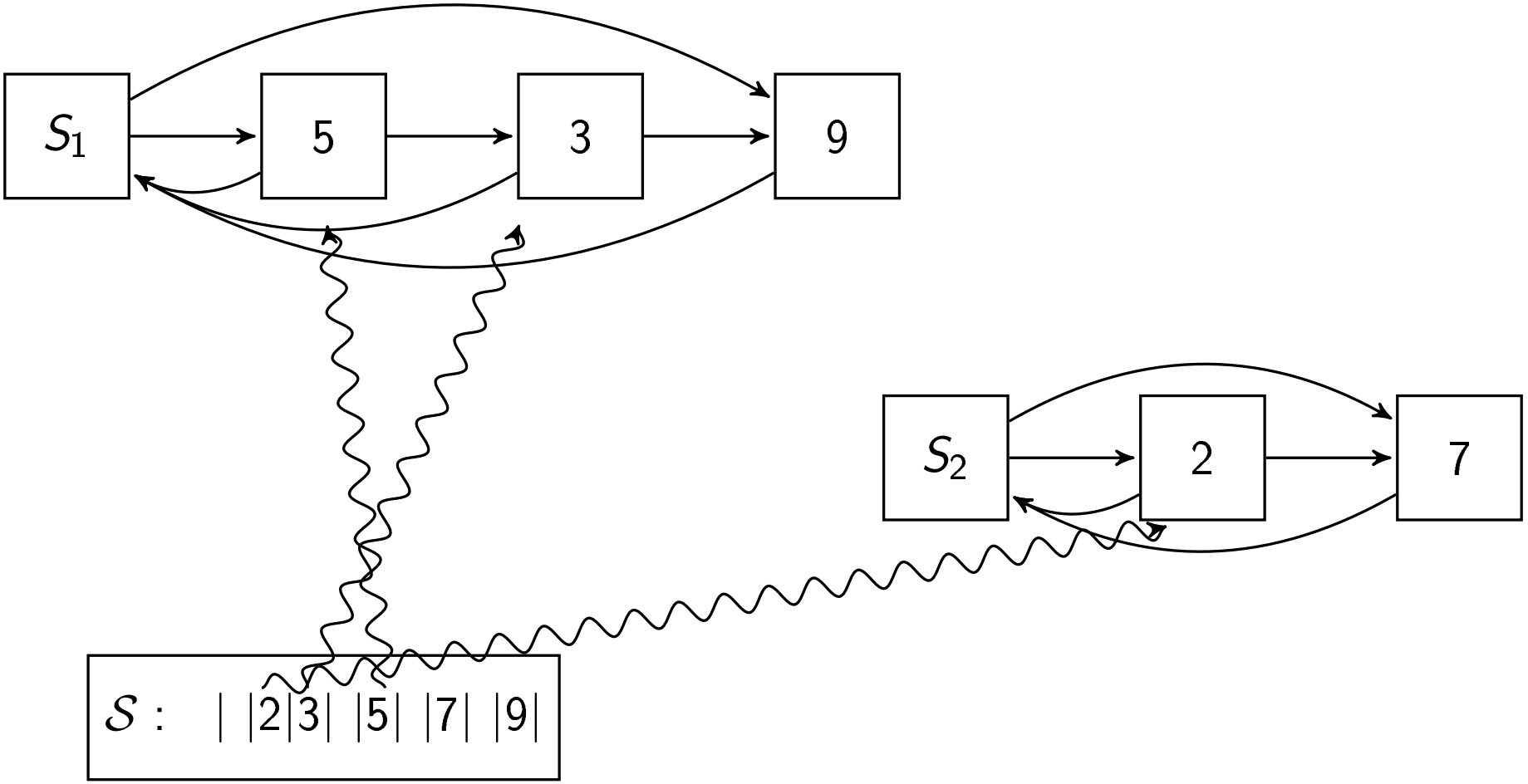




$\mathcal{S}$: |2|3|5|7|9|

Quando passiamo all'implementazione, però, facciamo un'ipotesi in più, e immaginiamo di conoscere gli elementi e li poterli mettere in un array.

Insiemi disgiunti attraverso liste



L'array contiene gli elementi che però sono ipotizzati essere numeri naturali. Dunque li usiamo come indici, e il contenuto della casella dell'array è un puntatore all'elemento denotato dal suo nome.

Insiemi disgiunti attraverso liste: *MakeSet*

L'operazione di *MakeSet* deve quindi partire dall'ipotesi che esista un array $\mathcal{S}$ di oggetti che sono puntatori a elementi di una lista. Abbiamo tre strutture dati concrete in gioco: gli elementi dell'array (puntatori a oggetti di tipo nodo), i nodi delle liste (triple fatte da una chiave intera, e due puntatori: *next*, e *head*), e teste delle liste (coppie di puntatori a oggetti di tipo nodo: *head* e *tail*).

Ogni esperimento parte da immaginare che gli elementi che entreranno nella struttura sono naturali fino ad un certo numero deciso all'inizio. L'array $\mathcal{S}$ si inizializza mettendo tutti i puntatori a **nil**. L'operazione di *MakeSet*, per esempio della chiave 4, alloca una nuova lista S, un nuovo nodo con chiave 4, collega S a 4 come *head* e come *tail*, e inizializza la posizione 4 di $\mathcal{S}$ con l'indirizzo del nodo appena creato.

Insiemi disgiunti attraverso liste: *Union*

Nel caso dell'operazione di *Union*, invece, dobbiamo eseguire le seguenti operazioni. Attraverso le chiavi (che fungono da indici in $\mathcal{S}$), risaliamo alle caselle di memoria che le contengono, e poi seguendo *head*.*head* arriviamo alla chiave che rappresenta l'insieme che contiene ognuna delle due (nel nostro esempio, partendo da 3 e 2 rispettivamente, arriveremmo a 5 e 2 stesso). Se queste sono uguali, non facciamo nulla. Altrimenti: si sceglie arbitrariamente una delle due e si segue *head*.*tail*; il *next* del nodo che si ottiene si fa puntare all'altro; si scorre tutta la lista del secondo rimpiazzando tutti i puntatori *head* al valore corretto; si libera la casella del nome della lista della seconda chiave.

Insiemi disgiunti attraverso liste: *FindSet*

Per effettuare *FindSet*, invece, si parte da una chiave (indice) in $\mathcal{S}$, si segue il suo puntatore, si segue *head*.*head*, e si restituisce la chiave ottenuta in questo modo.

Insiemi disgiunti attraverso liste: funzioni antagoniste e di controllo

Come verifichiamo che la nostra implementazione sia corretta? Poichè non ha senso 'stampare a schermo' una struttura dati complessa per esaminarla, il modo migliore è quello di avere una conferma indiretta della sua correttezza. In altre parole, è opportuno costruire una (serie di) funzione (funzioni) che compiano una serie di operazioni di *MakeSet*, *Union*, e *FindSet* con chiavi costanti decise a tavolino (hardcoded) delle quali si deve conoscere il risultato. Per esempio, si fanno un certo numero di insiemi, si uniscono in una certa sequenza, poi si effettua un'ultima operazione di *FindSet*, e si deve poter prevedere il valore della chiave risultato. Costruiamo una batteria di test fatta in questo modo, e la eseguiamo, seguendo, in un certo senso, il comportamento atteso su carta.

Conclusione

Nella prima parte dei questo esercizio abbiamo costruito e testato una prima versione delle strutture dati per insiemi disgiunti; non le abbiamo ancora inserite in un contesto di esperimento, cosa che faremo nella seconda parte.