Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno `luigi.catuogno@uniparthenope.it`
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

[FdP]

H. M. Deitel, P. J. Deitel

C++ Fondamenti di programmazione

II ed. (2014) Maggioli Editore (Apogeo Education)

ISBN: 978-88-387-8571-9

3

Libri di testo

Tecniche avanzate e strutture dati elementari

[TAP]

H. M. Deitel, P. J. Deitel

C++ Tecniche avanzate di programmazione

II ed. (2011) Maggioli Editore (Apogeo Education)

ISBN: 978-88-387-8572-6

4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Strutture dati elementari

6

Liste a doppi puntatori

Esempio: scansione di una lista…

File: **doubleList.hpp**

```
 …  …
34  class doubleList2: public doubleList {
35  private:
36          doubleNode *current;
37  public:
38          doubleList2(): current(nullptr) {};
39          ~doubleList2() {};
40          int showCurrent();
41          doubleList2 *forward();
42          doubleList2 *backward();
43          doubleList2 *begin();
44          doubleList2 *end();
45          bool atFront();
46          bool atBack();
47          doubleList *remove(int&) override;
48          doubleList2 *insertNext(int);
49          doubleList2 *removeCurrent(int&);
50  };
```

Spostano **current** rispettivamente in posizione **front** e **back**;

Restituiscono **true** se **current** coincide rispettivamente con **front** e **back**;

Esempio: scansione di una lista…

File: **doubleList.cpp**

```
…    …
132  doubleList2 *doubleList2::removeCurrent(int &item) {
133
134          doubleNode *tmp,*prevNode,*nextNode;
135
136          if(isEmpty())
137                  throw listIsEmpty();
138
139          if(current==nullptr)
140                  throw currentIsNull();
141
142          tmp=current;
143          nextNode=current->next;
144          prevNode=current->prev;
…    …
```

è necessario che il nodo sia indicato inequivocabilmente. Questa eccezione, ricorda all'utente di scorrere la lista fino a raggiungere il nodo intermedio da rimuovere.

9

Esempio: scansione di una lista…

File: **doubleList.cpp**

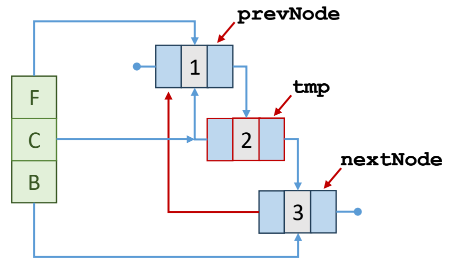
```
…    …
145          if(nextNode==nullptr) {
146                  back=prevNode;
147                  current=back;
148          } else {
149                  nextNode->prev=prevNode;
150                  current=nextNode;
151          }
152
153          if(prevNode==nullptr)
154                  front=nextNode;
155          else
156                  prevNode->next=nextNode;
157
158          item=tmp->data;
159          delete tmp;
160          return this;
161  }
```

10

Esempio: scansione di una lista...

```
     …   …
145          if(nextNode==nullptr) {
146                  back=prevNode;
147                  current=back;
148          } else {
149                  nextNode->prev=prevNode;
150                  current=nextNode;
151          }
152
153          if(prevNode==nullptr)
154                  front=nextNode;
155          else
156                  prevNode->next=nextNode;
157
158          item=tmp->data;
159          delete tmp;
160          return this;
161  }
```

Rimozione di un nodo intermedio...

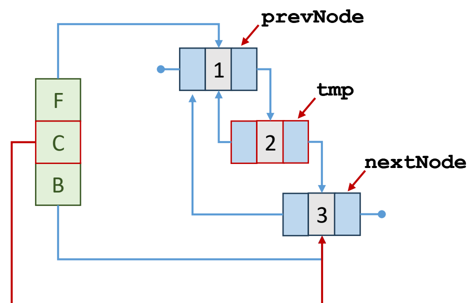


11

Esempio: scansione di una lista...

```
     …   …
145          if(nextNode==nullptr) {
146                  back=prevNode;
147                  current=back;
148          } else {
149                  nextNode->prev=prevNode;
150                  current=nextNode;
151          }
152
153          if(prevNode==nullptr)
154                  front=nextNode;
155          else
156                  prevNode->next=nextNode;
157
158          item=tmp->data;
159          delete tmp;
160          return this;
161  }
```

Rimozione di un nodo intermedio...

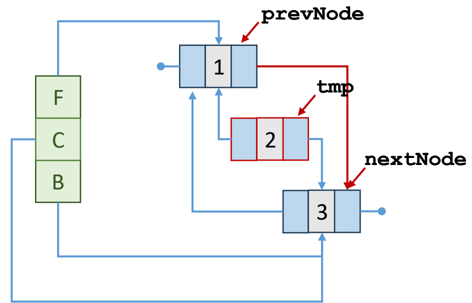


12

Esempio: scansione di una lista…

```
...                ...
145                if(nextNode==nullptr) {
146                        back=prevNode;
147                        current=back;
148                } else {
149                        nextNode->prev=prevNode;
150                        current=nextNode;
151                }
152
153                if(prevNode==nullptr)
154                        front=nextNode;
155                else
156                        prevNode->next=nextNode;
157
158                item=tmp->data;
159                delete tmp;
160                return this;
161        }
```

Rimozione di un nodo intermedio…

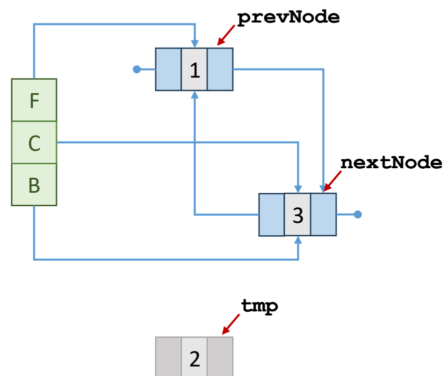


13

Esempio: scansione di una lista…

```
...                ...
145                if(nextNode==nullptr) {
146                        back=prevNode;
147                        current=back;
148                } else {
149                        nextNode->prev=prevNode;
150                        current=nextNode;
151                }
152
153                if(prevNode==nullptr)
154                        front=nextNode;
155                else
156                        prevNode->next=nextNode;
157
158                item=tmp->data;
159                delete tmp;
160                return this;
161        }
```

Rimozione di un nodo intermedio…

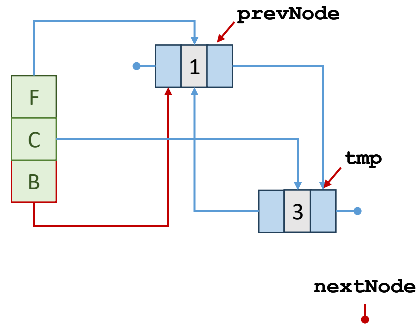


14

Esempio: scansione di una lista...

```
 …    …
145           if(nextNode==nullptr) {
146                   back=prevNode;
147                   current=back;
148           } else {
149                   nextNode->prev=prevNode;
150                   current=nextNode;
151           }
152
153           if(prevNode==nullptr)
154                   front=nextNode;
155           else
156                   prevNode->next=nextNode;
157
158           item=tmp->data;
159           delete tmp;
160           return this;
161   }
```

Rimozione di un nodo in coda...

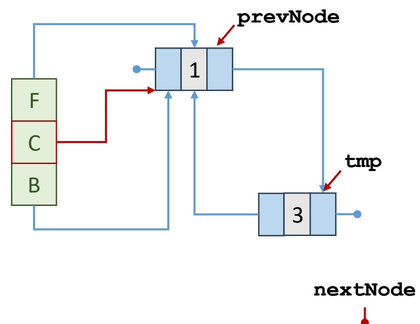


15

Esempio: scansione di una lista...

```
 …    …
145           if(nextNode==nullptr) {
146                   back=prevNode;
147                   current=back;
148           } else {
149                   nextNode->prev=prevNode;
150                   current=nextNode;
151           }
152
153           if(prevNode==nullptr)
154                   front=nextNode;
155           else
156                   prevNode->next=nextNode;
157
158           item=tmp->data;
159           delete tmp;
160           return this;
161   }
```

Rimozione di un nodo in coda...

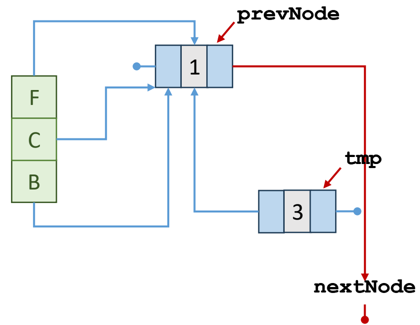


16

Esempio: scansione di una lista…

```
 …    …
145         if(nextNode==nullptr) {
146                 back=prevNode;
147                 current=back;
148         } else {
149                 nextNode->prev=prevNode;
150                 current=nextNode;
151         }
152
153         if(prevNode==nullptr)
154                 front=nextNode;
155         else
156                 prevNode->next=nextNode;
157
158         item=tmp->data;
159         delete tmp;
160         return this;
161 }
```

Rimozione di un nodo in coda…

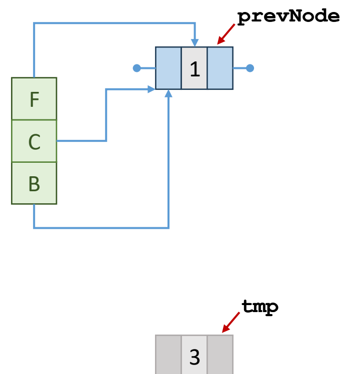


17

Esempio: scansione di una lista…

```
 …    …
145         if(nextNode==nullptr) {
146                 back=prevNode;
147                 current=back;
148         } else {
149                 nextNode->prev=prevNode;
150                 current=nextNode;
151         }
152
153         if(prevNode==nullptr)
154                 front=nextNode;
155         else
156                 prevNode->next=nextNode;
157
158         item=tmp->data;
159         delete tmp;
160         return this;
161 }
```

Rimozione di un nodo in coda…

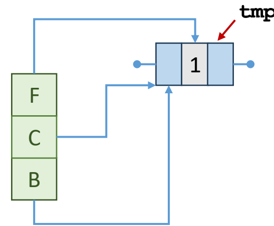


18

Esempio: scansione di una lista...

```
...
145          if(nextNode==nullptr) {
146                  back=prevNode;
147                  current=back;
148          } else {
149                  nextNode->prev=prevNode;
150                  current=nextNode;
151          }
152
153          if(prevNode==nullptr)
154                  front=nextNode;
155          else
156                  prevNode->next=nextNode;
157
158          item=tmp->data;
159          delete tmp;
160          return this;
161  }
```

Rimozione di un nodo singolo...

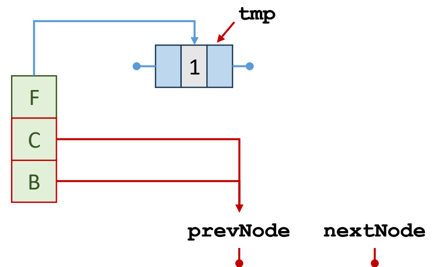


19

Esempio: scansione di una lista...

```
...
145          if(nextNode==nullptr) {
146                  back=prevNode;
147                  current=back;
148          } else {
149                  nextNode->prev=prevNode;
150                  current=nextNode;
151          }
152
153          if(prevNode==nullptr)
154                  front=nextNode;
155          else
156                  prevNode->next=nextNode;
157
158          item=tmp->data;
159          delete tmp;
160          return this;
161  }
```

Rimozione di un nodo singolo...

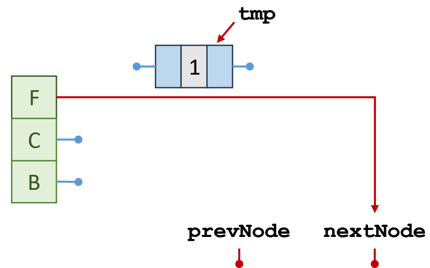


20

Esempio: scansione di una lista…

```
…
145         if(nextNode==nullptr) {
146                 back=prevNode;
147                 current=back;
148         } else {
149                 nextNode->prev=prevNode;
150                 current=nextNode;
151         }
152 
153         if(prevNode==nullptr)
154                 front=nextNode;
155         else
156                 prevNode->next=nextNode;
157 
158         item=tmp->data;
159         delete tmp;
160         return this;
161 }
```

Rimozione di un nodo singolo…

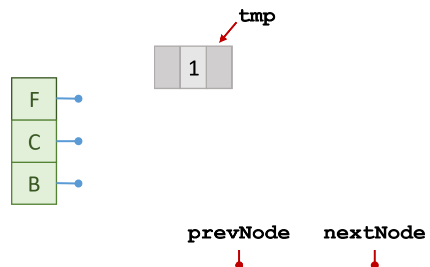


21

Esempio: scansione di una lista…

```
…
145         if(nextNode==nullptr) {
146                 back=prevNode;
147                 current=back;
148         } else {
149                 nextNode->prev=prevNode;
150                 current=nextNode;
151         }
152 
153         if(prevNode==nullptr)
154                 front=nextNode;
155         else
156                 prevNode->next=nextNode;
157 
158         item=tmp->data;
159         delete tmp;
160         return this;
161 }
```

Rimozione di un nodo singolo…



22

Esempio: scansione di una lista…

File: **doubleList.cpp**

```
…    …
108  doubleList2 *doubleList2::insertNext(int item) {
109
110          doubleNode *newNode,*nextNode;
111
112          if(isEmpty())
113                  throw listIsEmpty();
114
115          if(current==nullptr)
116                  throw currentIsNull();
117
118          newNode=new doubleNode(item);
119          nextNode=current->next;
120
…    …
```

23

Esempio: scansione di una lista…

File: **doubleList.cpp**

```
…    …
121          newNode->next=nextNode;
122          newNode->prev=current;
123          current->next=newNode;
124
125          if(current==back)
126                  back=newNode;
127          else
128                  nextNode->prev=newNode;
129
130          return this;
131  }
…    …
```

24

Esercizio: metodo **insertHere**…

Con riferimento alla classe **doubleList2**, scrivere il metodo…

```
doubleList2 *doubleList2::insertNext(int item)
```

…che inserisce il nuovo nodo *prima* di quello corrente. Si ricordi di gestire il caso in cui **current** è uguale a **front**;

25

Esercizio: metodi **search**, **count**

Con riferimento alla classe **doubleList2**, scrivere il metodo…

```
bool *doubleList2::search(int item)
```

…che restituisce **true** se l'item è presente nella lista e

```
int *doubleList2::count(int item)
```

…che restituisce il numero di occorrenze di **item** nella lista.

26

Gli alberi

27

La struttura dati *albero*

- A differenza di stack, code e liste, l'*albero* è una struttura dati *bidimensionale,* particolarmente utile per rappresentare dati in cui sussistono relazioni gerarchiche
- La sua struttura *ricorsiva* rende la codifica degli algoritmi che la utilizzano, semplice, compatta ed elegante
- Gli alberi sono particolarmente efficienti per implementare collezioni di dati in cui si richiede di fare frequentemente operazioni di ricerca e ordinamento

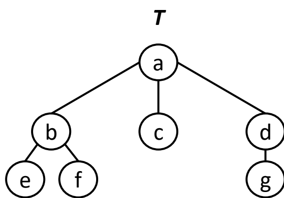
28

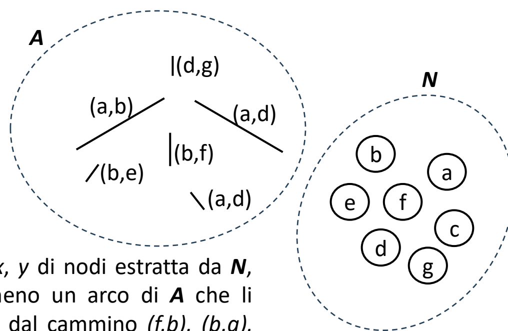
La struttura dati *albero*

- Formalmente, un albero è un *«grafo connesso, aciclico e non orientato»*
  - L'albero ***T*** è una coppia composta dagli insieme ***N*** dei *nodi* e ***A*** degli *archi.* Gli archi di ***A*** sono rappresentati da coppie di elementi di ***N***
  - Per cui, se *u* e *v* sono nodi di ***T***, l'elemento *(u,v)* di ***A*** è *l'arco che connette i nodi u e v*

29

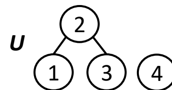
La struttura dati *albero*





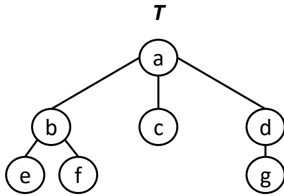
Il grafo **T** è *connesso*, poiché per qualsiasi coppia *x*, *y* di nodi estratta da ***N***, esiste una sequenza (cammino) composta da almeno un arco di ***A*** che li connette. Per esempio, i nodi *f* e *g* sono connessi dal cammino *(f,b), (b,a), (a,d), (d,g)*

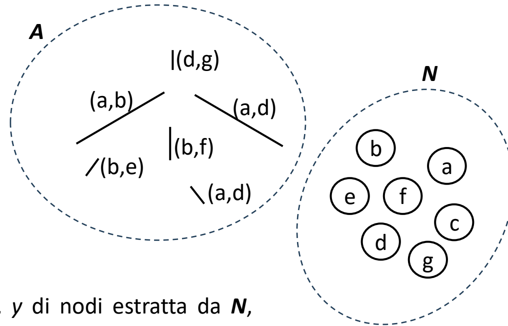
Il grafo ***U*** non è *connesso*, poiché non esistono cammini che colleghino il nodo *4* con gli altri.



30

La struttura dati *albero*

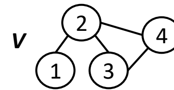




Il grafo ***T*** è *connesso*.

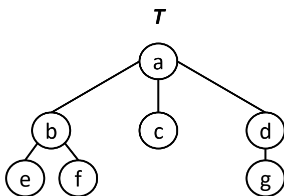
Il grafo ***T*** è *aciclico*, poiché per qualsiasi coppia *x*, *y* di nodi estratta da ***N***, esiste un solo cammino che connette *x* a *y*.

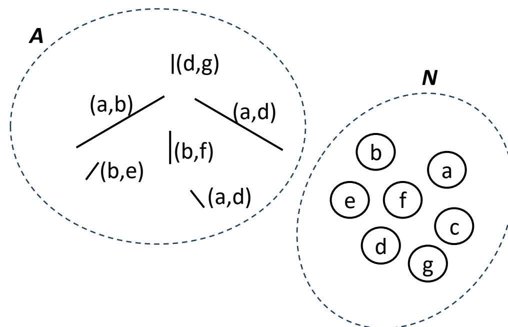
Il grafo ***V*** non è *aciclico*, poiché esistono due cammini che collegano *e.g.* il nodo *4* con il nodo 1.



31

La struttura dati *albero*



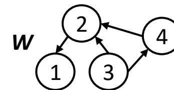


Il grafo ***T*** è *connesso*.

Il grafo ***T*** è *aciclico*.

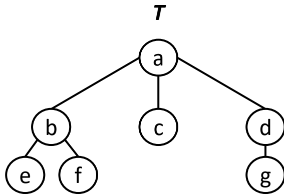
Il grafo ***T*** non è *orientato*, poiché, per qualsiasi coppia *x*, *y* di nodi in ***N***, il cammino che li connette *può essere percorso nei due sensi*.

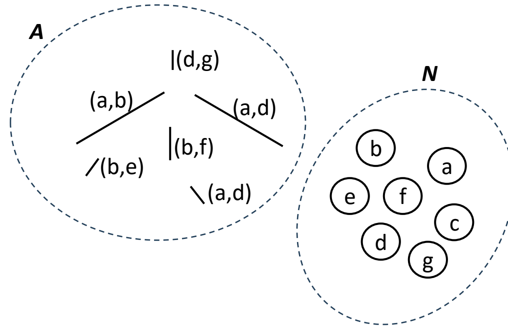
Il grafo ***W*** è *orientato*, poiché esistono cammini che collegano *e.g.* il nodo *3* con il nodo 1, ma nessuno che connetta il nodo 1 al nodo 3;



32

La struttura dati *albero*





Il grafo ***T*** è *connesso*.

Il grafo ***T*** è *aciclico*.

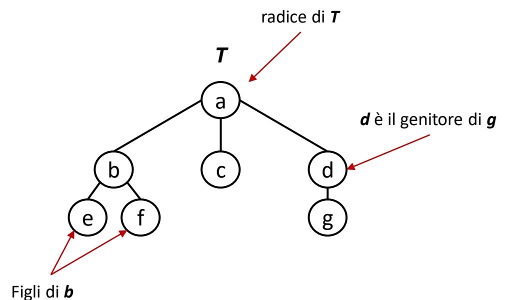
Il grafo ***T*** non è *orientato*.

Il grafo ***T*** è *un albero.*

33

La struttura dati *albero*

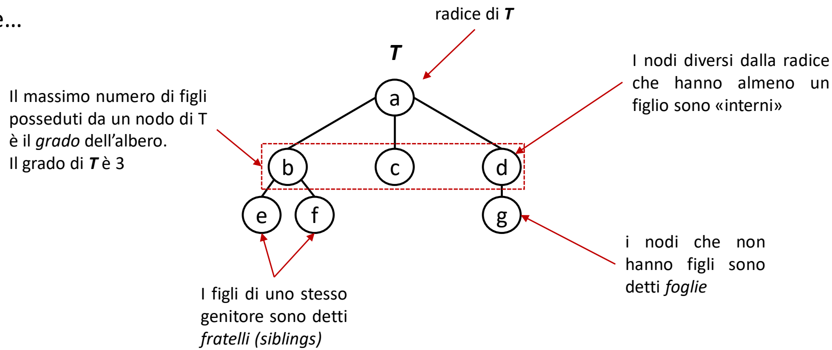
- In un albero *«radicato»*
  - C'è un nodo detto *radice (root)* che ha un ruolo *distinto* dagli altri;
  - ogni nodo *x* è connesso, mediante un arco, ad un certo numero di nodi *figli* (anche nessuno)
  - Tutti i nodi eccetto la radice, sono (sempre) connessi a un altro nodo detto *genitore (parent)*



34

La struttura dati *albero*

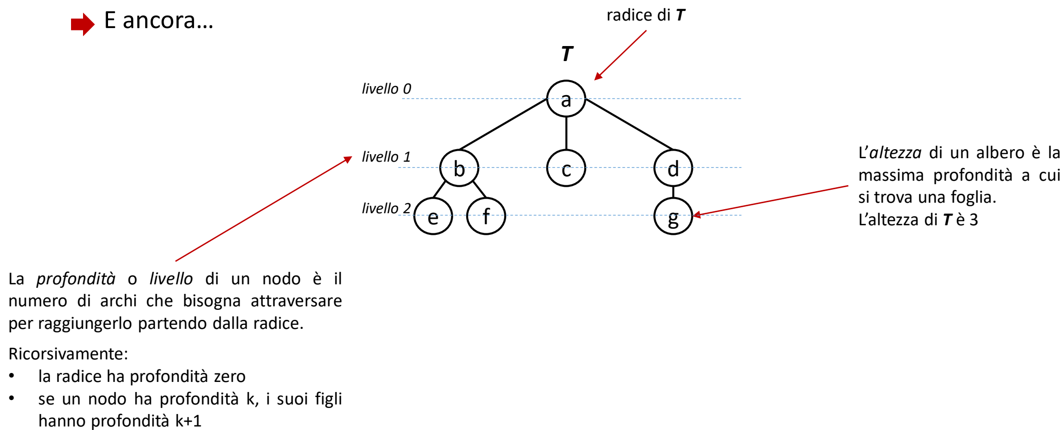
➡ Inoltre…



35

La struttura dati *albero*

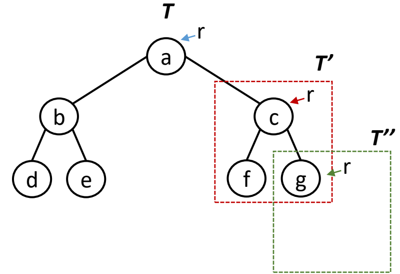
➡ E ancora…



36

La struttura dati *albero*

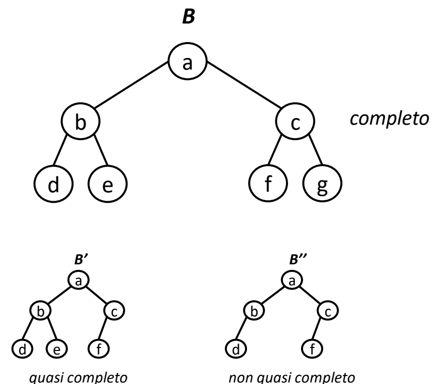
- E ancora…
  - L'albero ha una struttura intrinsecamente ricorsiva: ogni figlio *x* della della radice di ***T*** è a sua volta la radice del *sottoalbero* ***T'*** di ***T*** *radicato nel nodo x*
  - E così via.. Anche una foglia può essere considerato un sottoalbero con la sola radice che è anche nodo foglia.



37

La struttura dati *albero*

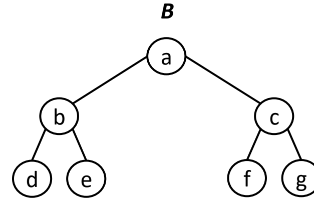
- E infine…
  - Un albero in cui tutti i nodi (eccetto le foglie) hanno *d* figli, si dice *d*-ario (*e.g. se d=2, l'albero è binario).*
  - Un albero *d*-ario in cui tutte le foglie sono sullo stesso livello si dice *«completo»*
  - Un albero *d*-ario si dice *«quasi completo»* se:
    - tutti i livelli, tranne al più l'ultimo sono completi
    - nell'ultimo livello possono mancare alcune foglie consecutive a partire dall'ultima foglia a destra



38

La struttura dati *albero*

- E infine…
  - Un *albero binario* completo* di altezza h, è composto da $2^{h+1} - 1$ nodi;
  - Un *albero binario* quasi completo* di n nodi ha una altezza $h = \lfloor \log n \rfloor$;
  - In un *albero binario completo*, in ciascun livello i ci sono 2^i nodi. Se l'albero è *quasi completo*, questo non vale per l'ultimo livello.



** d'ora in avanti si enunciano (per semplicità) le proprietà di un albero d-ario riferite al caso d=2. Tuttavia queste proprietà valgono per qualsiasi valore di d>1;*

39

Implementare gli alberi binari

40

Implementare gli alberi binari…

- Gli alberi (e quelli binari, in particolare) sono una struttura estremamente efficiente per «modellare» un gran numero di problemi.
- Ne sono state proposte innumerevoli varianti, ciascuna «specializzata» per specifiche applicazioni.
- La scelta della migliore implementazione dipende molto dalla natura dell'applicazione e delle proprietà degli alberi su cui questa fa maggiore uso.

41

Implementare gli alberi binari…

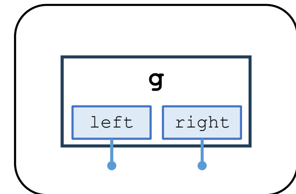
- Iniziamo con una implementazione «dinamica» che fa uso di nodi singoli e puntatori.
- Ciascun nodo dell'albero è un oggetto che contiene:
  - Un'*area dati*
  - due puntatori che lo connettono ai suoi (eventuali) nodi figlio.
  - Un *nodo di management* che contiene il puntatore alla radice dell'albero e altri *metadati*

42

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.hpp**

```
22  class binTreePtrNode {
23  public:
24          int data;
25          binTreePtrNode *left, *right;
26  
27          binTreePtrNode(int item): data(item), left(nullptr), right(nullptr) {};
28          ~binTreePtrNode() {};
29  
30          bool isLeaf() {
31                  return (left==nullptr) && (right==nullptr);
32          }
33          bool hasLeft() {
34                  return left!=nullptr;
35          }
36          bool hasRight() {
37                  return right!=nullptr;
38          }
39  };
```

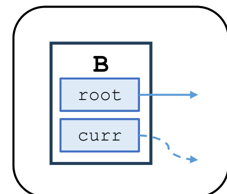


43

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.hpp**

```
…   …
42  class binTreePtr {
43  private:
44          binTreePtrNode *root;
45          binTreePtrNode *current;
46  
47          virtual void deleteSubtree(binTreePtrNode *);
48  public:
40          binTreePtr(): root(nullptr), current(nullptr) {};
50          virtual ~binTreePtr();
51  
52          bool isEmpty() {
53                  return root==nullptr;
54          }
55          bool isRoot() {
56                  return current==root;
57          }
…   …
```

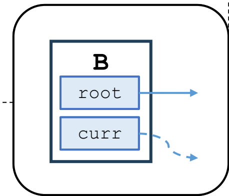


44

Esempio: albero binario *«a puntatori»*

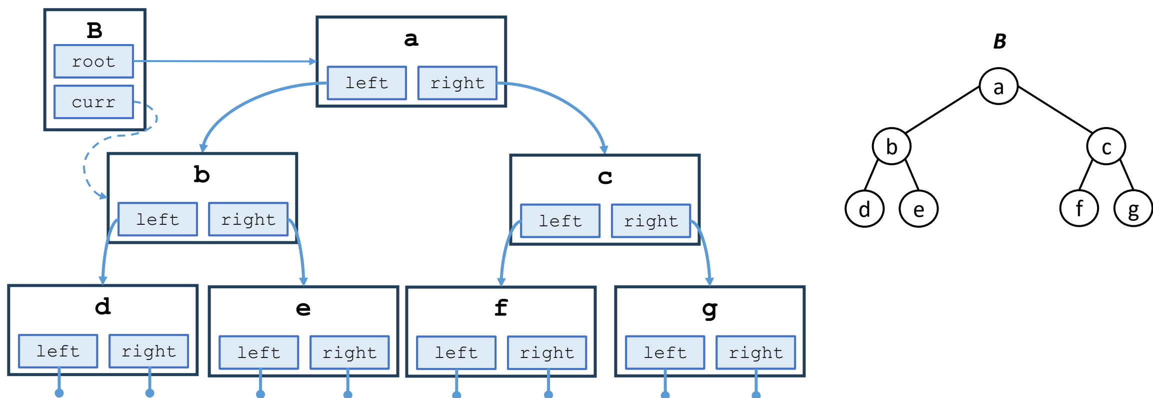
File: **binTreePtr.hpp**

```
...   ...
58          virtual binTreePtr *insertRoot(int);
59          virtual binTreePtr *insertRight(int);
60          virtual binTreePtr *insertLeft(int);
61
62          virtual binTreePtr *goRoot();
63          virtual binTreePtr *goLeft();
64          virtual binTreePtr *goRight();
65          int getData();
66
67          virtual void preorderVisit();
68          virtual void postorderVisit();
69          virtual void inorderVisit();
70    };
```



45

Esempio: albero binario *«a puntatori»*



46

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.hpp**

```
 4  #include<stdexcept>
 5  using std::runtime_error;
 6
 7  class nodeAlreadyExists: public runtime_error {
 8  public:
 9          nodeAlreadyExists(): runtime_error("Il nodo gia' esistente"){};
10  };
11
12  class treeIsEmpty: public runtime_error {
13  public:
14          treeIsEmpty(): runtime_error("Albero vuoto"){};
15  };
16
17  class missingChild: public runtime_error {
18  public:
19          missingChild(): runtime_error("Nodo figlio mancante"){};
20  };
```

47

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.cpp**

```
3  binTreePtr *binTreePtr::insertRoot(int item) {
4          if(!isEmpty())
5                  throw nodeAlreadyExists();
6          root=new binTreePtrNode(item);
7          current=root;
8          return this;
9  }
…  …
```

Inserisce la radice in un albero vuoto e assegna **item** al suo camppo **data**.

Inserire una radice dove già c'è è un errore

Il puntatore **current**, punta al nodo «corrente», quello a partire dal quale si effettuano le operazioni richieste. Inizialmente **current** e **root** coincidono.

48

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.cpp**

```
…   …
10  binTreePtr *binTreePtr::insertLeft(int item) {
11          if(isEmpty())
12                  throw treeIsEmpty();
13          if(current->left!=nullptr)
14                  throw nodeAlreadyExists();
15          current->left=new binTreePtrNode(item);
16          return this;
17  }
18  …
19
20
```

Aggiunge un nodo quale figlio sinistro del nodo corrente e assegna **item** al suo campo **data**.

Non si possono inserire nodi dove ce ne sono già o se l'albero è vuoto

Crea un nuovo nodo e ne assegna l'indirizzo al puntatore **left** di **current**. Si noti che **current** non cambia.

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.cpp**

```
…   …
10  binTreePtr *binTreePtr::insertLeft(int item) {
11          if(isEmpty())
12                  throw treeIsEmpty();
13          if(current->left!=nullptr)
14                  throw nodeAlreadyExists();
15          current->left=new binTreePtrNode(item);
16          return this;
17  }
18
19  binTreePtr *binTreePtr::insertRight(int item) {
20          if(isEmpty())
21                  throw treeIsEmpty();
22          if(current->right!=nullptr)
23                  throw nodeAlreadyExists();
24          current->right=new binTreePtrNode(item);
25          return this;
26  }
…   …
```

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.cpp**

```
29 binTreePtr *binTreePtr::goRoot() {
30         if (isEmpty())
31                 throw treeIsEmpty();
32         current=root;
33         return this;
34 }
35 binTreePtr *binTreePtr::goLeft() {
36         if (isEmpty())
37                 throw treeIsEmpty();
38         if (!current->hasLeft())
39                 throw missingChild();
40         current=current->left;
41         return this;
42 }
54 int binTreePtr::getData(){
55         if(isEmpty())
56                 throw treeIsEmpty();
57         return current->data; }
```

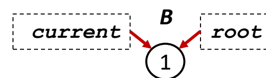
Imposta **root** come nodo *corrente.*

Il nuovo nodo *corrente* è il figlio sinistro di quello attuale

51

Esempio: creazione e utilizzo di un albero

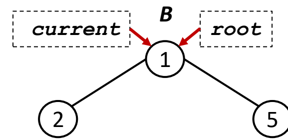
```
5  int main()
6  {
7          binTreePtr *B;
8          B=new binTreePtr();
9          B->insertRoot(1);
10
```



52

Esempio: creazione e utilizzo di un albero

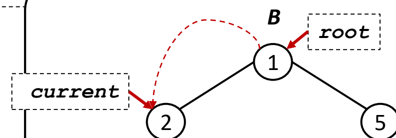
```
5   int main()
6   {
7           binTreePtr *B;
8           B=new binTreePtr();
9           B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
```



53

Esempio: creazione e utilizzo di un albero

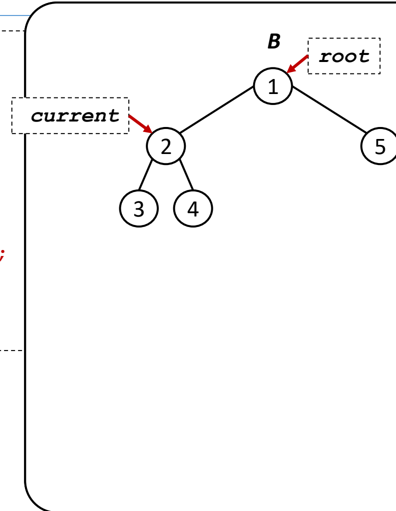
```
5   int main()
6   {
7           binTreePtr *B;
8           B=new binTreePtr();
9           B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
12          B->goLeft();
13
14
15
```



54

Esempio: creazione e utilizzo di un albero

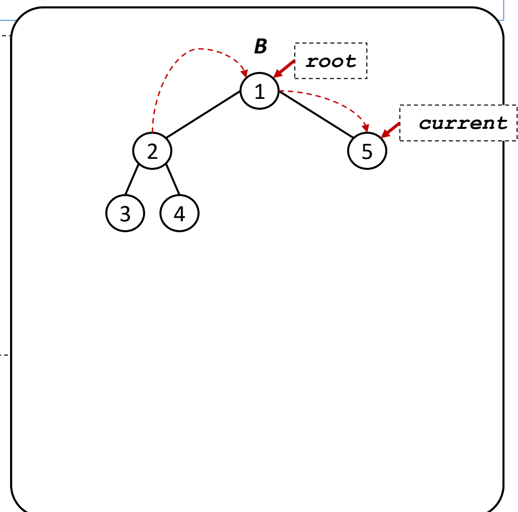
```
5   int main()
6   {
7           binTreePtr *B;
8           B=new binTreePtr();
9           B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
12          B->goLeft();
13          B->insertLeft(3)->insertRight(4);
14
15
```



55

Esempio: creazione e utilizzo di un albero

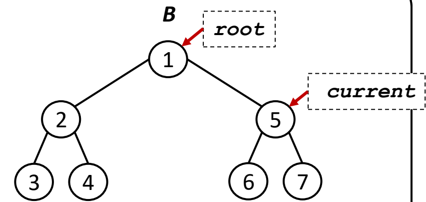
```
5   int main()
6   {
7           binTreePtr *B;
8           B=new binTreePtr();
9           B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
12          B->goLeft();
13          B->insertLeft(3)->insertRight(4);
14          B->goRoot()->goRight();
15
```



56

Esempio: creazione e utilizzo di un albero

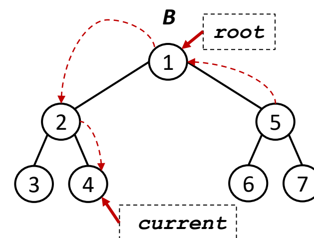
```
 5  int main()
 6  {
 7          binTreePtr *B;
 8          B=new binTreePtr();
 9          B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
12          B->goLeft();
13          B->insertLeft(3)->insertLeft(4);
14          B->goRoot()->goRight();
15          B->insertLeft(6);
16          B->insertRight(7);
```



57

Esempio: creazione e utilizzo di un albero

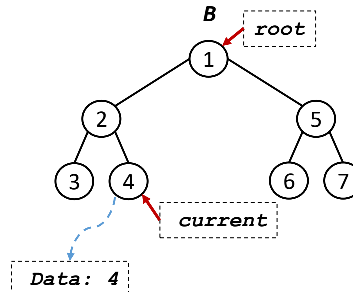
```
 5  int main()
 6  {
 7          binTreePtr *B;
 8          B=new binTreePtr();
 9          B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
12          B->goLeft();
13          B->insertLeft(3)->insertLeft(4);
14          B->goRoot()->goRight();
15          B->insertLeft(6);
16          B->insertRight(7);
17          B->goRoot()->goLeft()->goRight();
18
```



58

Esempio: creazione e utilizzo di un albero

```
 5  int main()
 6  {
 7          binTreePtr *B;
 8          B=new binTreePtr();
 9          B->insertRoot(1);
10          B->insertLeft(2);
11          B->insertRight(5);
12          B->goLeft();
13          B->insertLeft(3)->insertLeft(4);
14          B->goRoot()->goRight();
15          B->insertLeft(6);
16          B->insertRight(7);
17          B->goRoot()->goLeft()->goRight();
18          cout << "Data: " << B->getData();
19  }
```



59

Algoritmi sugli alberi binari

60

Algoritmi sugli alberi binari: *le visite*

- L'enumerazione dei dati contenuti in un albero richiede la *visita* di tutti i suoi nodi.
- A differenza di quanto avviene nelle strutture dati lineari, dove il percorso è unico, negli alberi è necessario seguire un cammino che si snodi tra tutte le biforcazioni e che tocchi almeno una vota ciascun nodo.

61

Algoritmi sugli alberi binari: *le visite*

Le principali strategie per effettuare la visita di un albero sono:

- Visite in profondità *(depth-first visit):*
  - L'algoritmo visita *«longitudinalmente»* i nodi lungo i cammini che vanno dalla radice a tutte le foglie
- Visite in ampiezza *(breadth-first visit)*
  - L'algoritmo visita tutti i nodi su uno stesso livello e poi procede con quelli del livello successivo, fino alle foglie.

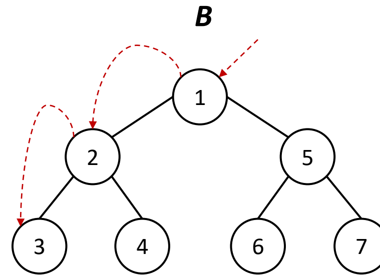
62

Esempio: una visita *depth-first*

Iniziamo la visita dalla radice, visualizziamo il suo campo **`data e`** poi:

passiamo al figlio sinistro, visualizziamo i dati e…

scendiamo ancora a sinistra, visualizziamo i dati e…



Visita: **1 2 3**

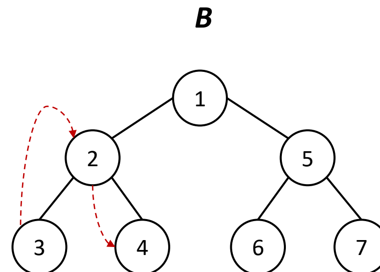
63

Esempio: una visita *depth-first*

Torniamo al nodo precedente

scendiamo dall'altra parte (a destra) e visualizziamo i dati…

Anche questo cammino è esaurito..



Visita: **1 2 3 4**

64

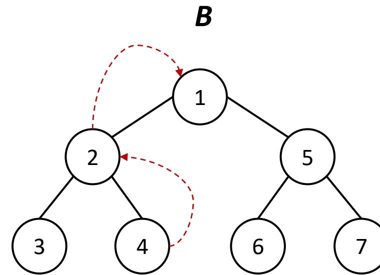
Esempio: una visita *depth-first*

Torniamo al nodo superiore: non ci sono altre strade qui…

Torniamo ancora su:

abbiamo completato la visita nel *sottoalbero* sinistro della radice,

Ora.. andiamo a destra…



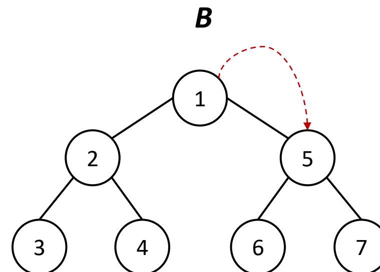
Visita: **1 2 3 4**

Esempio: una visita *depth-first*

Ora.. andiamo a destra:

Visualizziamo i dati del figlio destro della radice, poi..

scendiamo prima a sinistra:



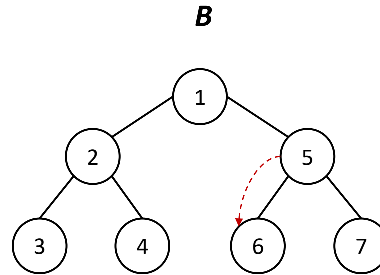
Visita: **1 2 3 4 5**

Esempio: una visita *depth-first*

Ora.. andiamo a destra:

Visualizziamo i dati del figlio destro della radice, poi..

scendiamo prima a sinistra e visualizziamo…



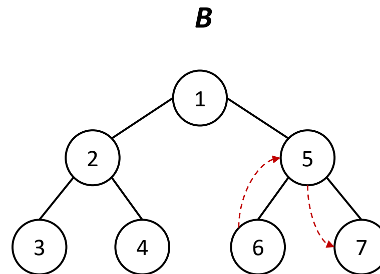
Visita: **1 2 3 4 5 6**

67

Esempio: una visita *depth-first*

Torniamo al livello superiore, scendiamo a destra…

Visualizziamo i dati



Visita: **1 2 3 4 5 6 7**

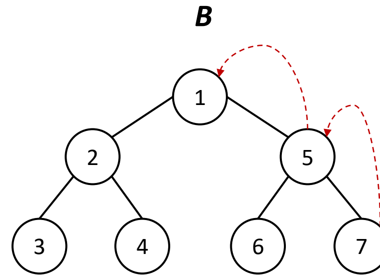
68

Esempio: una visita *depth-first*

Anche questo cammino è esaurito. Si torna a ritroso fino alla radice.

La visita è terminata.

B



Visita: **1 2 3 4 5 6 7**

69

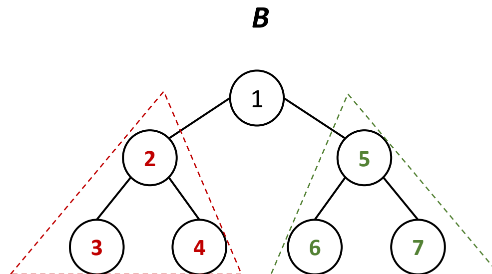
Esempio: una visita *depth-first*

Cosa abbiamo fatto?

Per visitare l'albero ***B*** abbiamo:

a) Estratto i dati della radice
b) Aggiunto il risultato della visita del sottoalbero sinistro
c) Aggiunto il risultato della visita del sottoalbero destro

B



Visita: **1 2 3 4 5 6 7**

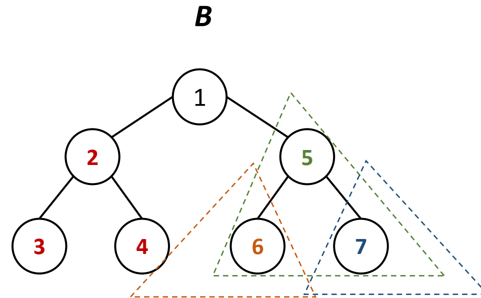
70

Esempio: una visita *depth-first*

Possiamo ragionare *ricorsivamente*:

Per visitare ciascun sottoalbero di ***B***:

a) estraiamo i dati della sua radice
b) aggiungiamo il risultato della visita del suo sottoalbero sinistro
c) aggiungiamo il risultato della visita del suo sottoalbero destro



Visita: **1 2 3 4 5 6 7**

71

Esempio: una visita *depth-first*

In *pseudocodice*. Sia *n* il puntatore a un nodo di ***B***, visitiamo il sottoalbero di ***B*** radicato in *n* :

```
visit(n) {
	if (n==nullptr)
		return;
	print n->data;
	visit (n->left);
	visit (n->right);
	return;
}
```

72

Esempio: una visita *depth-first*

In *pseudocodice.* Sia *n* il puntatore a un nodo di ***B**,* visitiamo il sottoalbero di ***B*** radicato in *n* :

```
visit(n) {
      if (n==nullptr)
            return;
      print n->data;
      visit (n->left);
      visit (n->right);
      return;
}
```

Base della ricorsione. Non si visitano sottoalberi vuoti.

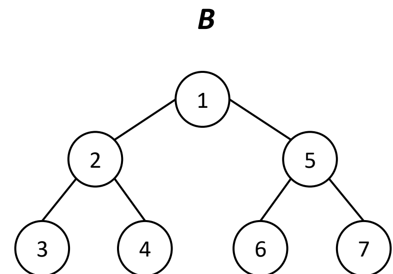
Quando la visualizzazione del dato della radice precede la visita dei sotto alberi la visita si dice *preorder*

Visita i sottoalberi destro e sinistro (nessun risultato là dove *n* non ha figli)

Alberi binari: visite *depth-first*

- Visita in ordine anticipato *(preorder traversal)*
  - *prima* elabora il dato del nodo e *dopo* ricorsivamente effettua la visita dei suoi sottoalberi sinistro e destro;
- Visita in ordine posticipato *(postorder traversal)*
  - *prima* effettua ricorsivamente la visita dei suoi sottoalberi sinistro e destro e *poi* elabora il dato del nodo;
- Visita in ordine *«simmetrico» (inorder traversal)*
  - *prima* effettua ricorsivamente la visita del sottoalbero sinistro, *poi* elabora il dato del nodo e *infine* effettua ricorsivamente la visita del sottoalbero destro;

B




```
Preorder:     1 2 3 4 5 6 7
Postorder:    3 4 2 6 7 5 1
Inorder:      3 2 4 1 6 5 7
```

Esempio: albero binario *«a puntatori»*

File: **binTreePtrDFV.cpp**

```
5   void do_preorderVisit(binTreePtrNode *n){
6           if(n==nullptr)
7                   return;
8           cout << n->data <<", ";
9           do_preorderVisit(n->left);
10          do_preorderVisit(n->right);
11  }
12
13  void binTreePtr::preorderVisit() {
14          if(isEmpty()) {
15                  cout << "Albero vuoto"<<endl;
16                  return;
17          }
18          cout<< "Lista (preorder): ";
19          do_preorderVisit(current);
20          cout<<endl;
21  }
```

75

Esempio: albero binario *«a puntatori»*

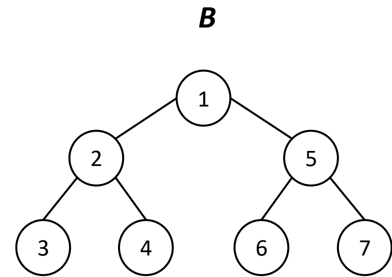
File: **binTreePtrDFV.cpp**

```
5   void do_inorderVisit(binTreePtrNode *n){
6           if(n==nullptr)
7                   return;
8           do_inorderVisit(n->left);
9           cout << n->data <<", ";
10          do_inorderVisit(n->right);
11  }
12
13  void binTreePtr::inorderVisit() {
14          if(isEmpty()) {
15                  cout << "Albero vuoto"<<endl;
16                  return;
17          }
18          cout<< "Lista (inorder): ";
19          do_inorderVisit(current);
20          cout<<endl;
21  }
```

76

Esempio: albero binario *«a puntatori»*

- Il metodo distruttore di un albero, deve de-allocare tutti i suoi nodi. Per fare questo deve effettuare una visita in cui la fase di «elaborazione» del nodo coincide con la sua distruzione. Occorre seguire le seguenti regole:
  - Tutti i nodi devono essere distrutti;
  - Un nodo può essere distrutto solo *dopo* che sono stati distrutti i suoi figli;



Quale visita effettua il distruttore?

Esempio: albero binario *«a puntatori»*

File: **binTreePtr.cpp**

```
 …  …
60  void binTreePtr::deleteSubtree(binTreePtrNode *r) {
61          if(r==nullptr)
62                  return;
63          deleteSubtree(r->left);
64          deleteSubtree(r->right);
65          delete r;
66          return;
67  }
68
69  binTreePtr::~binTreePtr() {
70          if(!isEmpty())
71                  deleteSubtree(root);
72  }
 …  …
```

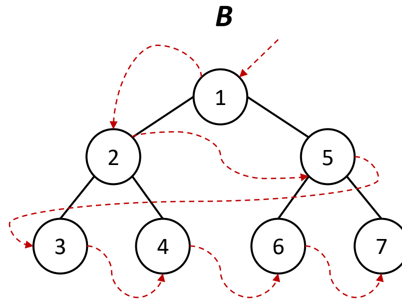
Esempio: una visita *breadth-first*

Nella visita per livelli *(breadth-first):*

Iniziamo la visita dalla radice, visualizziamo il suo campo **`data`** **`e`** poi si passa al livello successivo

al livello i, si enumera il contenuto dell'area dati di tutti i nodi (da sx verso dx), quindi…

si passa al livello $i + 1$ e si prosegue fino alle foglie…



Visita: **1 2 5 3 4 6 7**

79

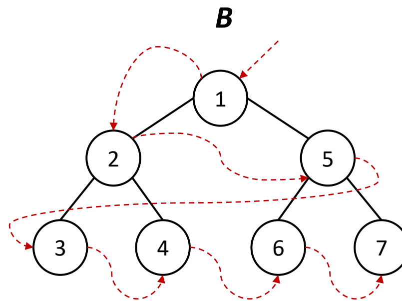
Esempio: una visita *breadth-first*

Per implementare una visita per livelli, utilizziamo una coda FIFO di supporto ***Q*** i cui item sono *puntatori ai nodi dell'albero.*

Una volta avviato su un nodo, l'algoritmo di visita

1) mette in coda i puntatori ai suoi figli (se non sono nulli),
2) visualizza il suo contenuto,
3) estrae il successivo nodo dalla coda e ricomincia da 1);

La visita termina quando la coda si svuota…



Visita: **1 2 5 3 4 6 7**

80

Esempio: una visita *breadth-first*

In *pseudocodice.* Sia *n* il puntatore a un nodo di ***B,*** visitiamo il sottoalbero di ***B*** radicato in *n* :

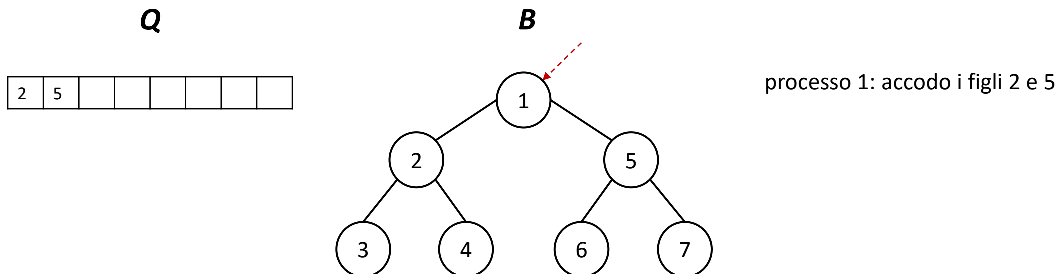
```
BFVisit(node *n) {
        queue Q;
        Q.enqueue(n);
        while(!Q.isEmpty()) {
                n=Q.dequeue():
                if(n->left!=nullptr)
                        Q.enqueue(n->left);
                if(n->right!=nullptr)
                        Q.enqueue(n->right);
                print n->data;
        }
}
```

81

Esempio: una visita *breadth-first*

Q

B

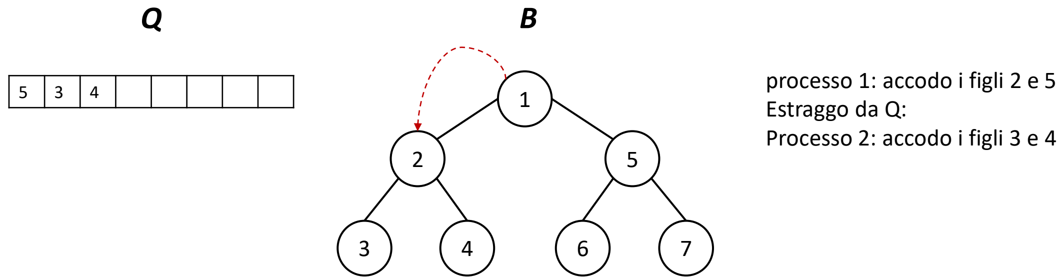


processo 1: accodo i figli 2 e 5

Visita: **1**

82

Esempio: una visita *breadth-first*

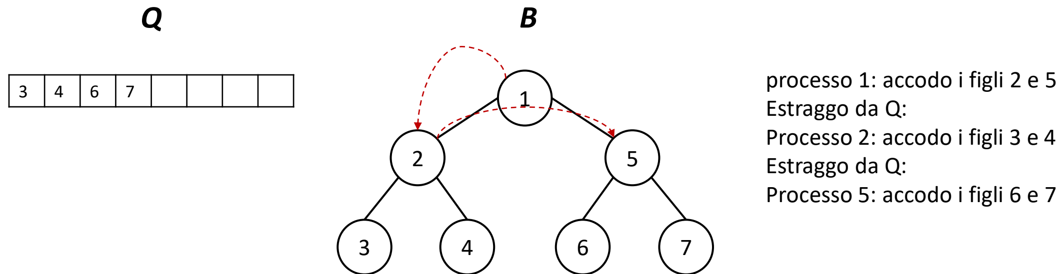


processo 1: accodo i figli 2 e 5
Estraggo da Q:
Processo 2: accodo i figli 3 e 4

Visita: **1 2**

83

Esempio: una visita *breadth-first*

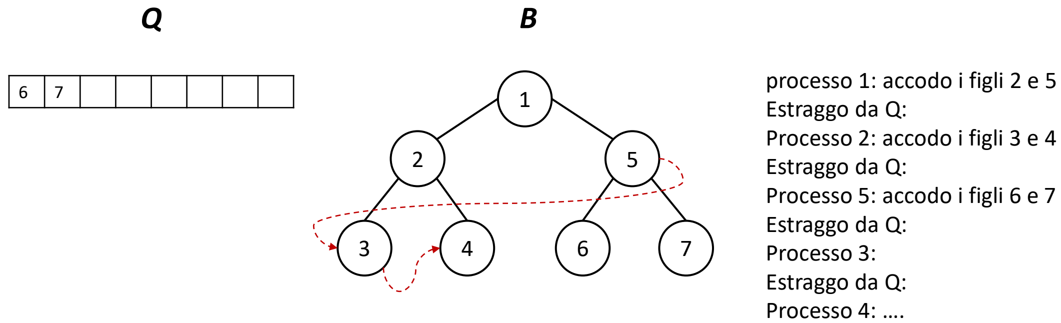


processo 1: accodo i figli 2 e 5
Estraggo da Q:
Processo 2: accodo i figli 3 e 4
Estraggo da Q:
Processo 5: accodo i figli 6 e 7

Visita: **1 2 5**

84

Esempio: una visita *breadth-first*

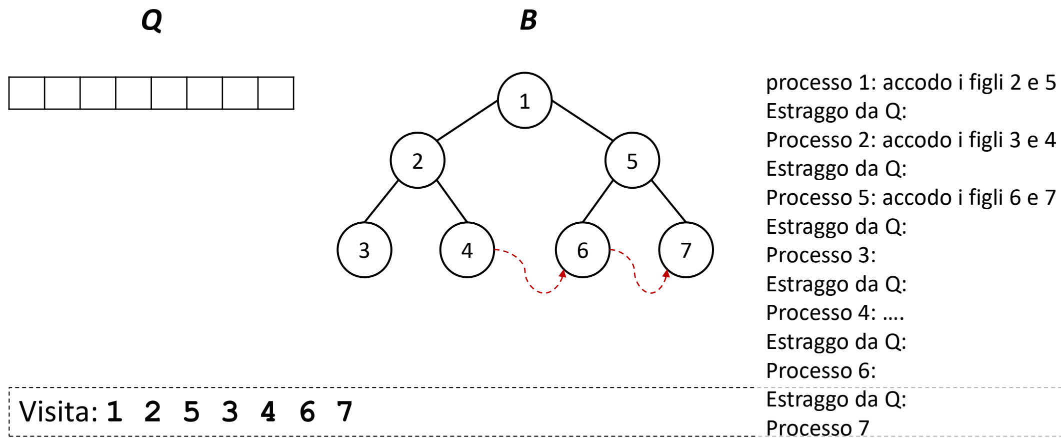


processo 1: accodo i figli 2 e 5
Estraggo da Q:
Processo 2: accodo i figli 3 e 4
Estraggo da Q:
Processo 5: accodo i figli 6 e 7
Estraggo da Q:
Processo 3:
Estraggo da Q:
Processo 4: ….

Visita: **1 2 5 3 4** …

85

Esempio: una visita *breadth-first*



processo 1: accodo i figli 2 e 5
Estraggo da Q:
Processo 2: accodo i figli 3 e 4
Estraggo da Q:
Processo 5: accodo i figli 6 e 7
Estraggo da Q:
Processo 3:
Estraggo da Q:
Processo 4: ….
Estraggo da Q:
Processo 6:
Estraggo da Q:
Processo 7

Visita: **1 2 5 3 4 6 7**

86