Sommario

Elenco delle Figure

1. Tracce Proposte

Traccia 5:

A. La Fiat per la produzione della Nuova Panda ha installato nello stabilimento di Pomigliano d' Arco K catene di montaggio, con K > 1. Ogni catena di montaggio ha L stazioni di lavoro. Ogni stazione di lavoro ha un tempo di lavorazione differente T. Inoltre il telaio della Nuova Panda impiega un tempo iniziale I per entrare nella catena di montaggio ed un tempo differente U per ogni catena di montaggio per uscire. Si assuma che il tempo di migrazione ε ; tra due stazioni successive all' interno della stessa catena sia non trascurabile . Infine, per fornire all' impianto la massima flessibilità si preveda la possibilità che la Nuova Panda possa migrare da una catena di montaggio ad un' altra e tale operazione impieghi un tempo, differente per ogni catena, pari a S. Si risolva, mediante la programmazione dinamica, il problema di individuazione per ogni vettura del percorso pi rapido, scrivendo le relative equazioni di ricorrenza. Successivamente si implementi, un programma, utilizzando l' approccio summenzionato, che individui per ogni vettura il percorso più rapido. Si verifichi la correttezza dell' implementazione su una serie di configurazioni possibili dell' impianto.

B. Si costruisca, utilizzando, un albero Red-Black una coda di max priorità. Si ricorda che l' interfaccia pubblica della coda di priorit´a deve contenere i seguenti metodi (extract-max, insert, maximum, increase-key) più gli eventuali costruttori e distruttore.

2.Problema della Catena di Montaggio

2.1 Definizione del problema

Data la traccia A vengono analizzati i requisiti logici ed implementativi da essa richiesti.Si tratta evidentemente del noto problema della catena di montaggio,con diverse varianti rispetto alla tipologia affrontata durante il corso,di cui tenere necessariamente conto.Vengono introdotte innanzitutto K catene di montaggio,con K>1;la problematica non si limita dunque alle sole due catene di montaggio,ma ammette un numero superiore di catene con cui lavorare.Ogni catena di montaggio è inoltre composta da un numero di stazioni L,ognuna delle quali caratterizzata da un tempo di lavoro T differente rispetto cui lavorano l'automobile alla sua percorrenza,e dai rispettivi tempi di entrata I e tempi di uscita U.Viene poi consentito all'automobile di spostarsi tra le varie stazioni per portare a termine il suo percorso di completamento,pagando in ogni caso un tempo di transizione non trascurabile,sia che essa provenga da una stazione immediatamente precedente appartenente alla medesima catena,sia che essa provenga da una differente linea di assemblaggio. Definiamo in particolar modo come ε; il tempo di trasferimento dell'automobile tra due catene successive appartenenti alla stessa catena,e come S il tempo di trasferimento tra due stazioni di catene differenti.A tali condizioni imposte verranno dunque applicati i principi della programmazione dinamica.

2.2 PD per la Catena di Montaggio

La programmazione dinamica è un approccio algoritmico che si basa sulla divisione dei problemi in sottoproblemi,particolarmente utile quando essi risultano tra loro dipendenti.In tale tecnica infatti i sottoproblemi ottenuti vengono risolti una sola volta,memorizzando i risultati ottenuti ,evitando di ripetere eventualmente i calcoli.Viene in genere adottata per problemi di ottimizzazione,vale a dire problematiche che godono di diverse soluzioni ottime rispetto cui associare un indice di bontà per individuare quale sia la migliore soluzione desiderata.

Il suo utilizzo prevede quattro fasi di sviluppo fondamentali che sono:

1. caratterizzazione della struttura ottima di una soluzione.
2. definizione ricorsiva del valore di una soluzione ottima,rispettando la proprietà della sottostruttura ottima.
3. calcolo del valore di una soluzione ottima secondo uno schema bottom-up.
4. costruzione di una soluzione ottima a partire dalle informazioni calcolate precedentemente.

Applicare la tecnica della programmazione dinamica alla traccia A risulta evidentemente vantaggioso,calcolare infatti tutte le possibili soluzioni al fine di individuare il percorso con tempo minimo totale per assemblare un automobile richiederebbe un tempo esponenziale ti tipo $O(2^n)$,con n pari al numero delle stazioni.In particolar modo le condizioni su cui si basa il nostro problema sono le sequenti:

- Lo stabilimento ha un numero di stazioni n definito in input.
- Lo stabilimento ha un numero di catene m definito in input.
- Ogni stazione L è caratterizzata da un tempo di lavorazione T per l'automobile
  in particolar modo due stazioni corrispondenti L (p,j) ed L (q,j) ,con
  (p,q<= n) e (j<= m) ,effettuano la stessa operazione ma con due tempi di esecuzione T (p,j) ≠ / T (q,j) .
- Ogni catena K è caratterizzata da un tempo di entrata I e da un tempo di uscita U,differenti tra le varie catene. Es: I (p) ≠ / I (q)
- Le catene possono scambiarsi le auto da assemblare.
- Il passaggio dell'automobile da una catena immediatamente precedente a quella successiva di una stessa catena ammette un tempo tempo di trasferimento non trascurabile ε .
- Il passaggio dell'automobile da una catena immediatamente precedente a quella successiva di una catena differente ammette un tempo tempo di trasferimento pari ad S.

2.2.1 Notazioni:

Prima di analizzare la fase implementativa è utile introdurre alcune notazioni,necessarie per comprendere al meglio quanto espresso in seguito,in particolare:

- Si indica con n il numero di stazioni dello stabilimento.
- Si indica con m il numero delle catene dello stabilimento.
- Si indica con L (i,j) la j-sima stazione appartenente alla i-sima catena di montaggio.
- Si indica con T(i,j) il tempo di lavoro della catena j appartenente alla catena i.
- S indica con S(k,j) il tempo di trasferimento dalla stazione L (k,j-1)) alla stazione L (i,j) da una catena k se k ≠ / i.
- Si indica con ε (i,j) il tempo di trasferimento dalla stazione L (i,j-1) alla stazione L (i,j) dalla stessa catena i.
- Si indica con I (j) il tempo di entrata in una catena e con U (j) il tempo di uscita da una catena.
- Si indica con f (i,j) il tempo minimo che impiega l'auto dal punto iniziale sino alla stazione L (i,j) .
- Si indica con F' il tempo totale minimo che l'automobile impiega ad attraversare l'intero stabilimento.

2.3 Analisi dell'implementazione

Si è intrapreso un percorso di implementazione consono alle quattro fasi di sviluppo proprie della tecnica della programmazione dinamica precedentemente introdotte,utilizzando poi i risultati ottenuti per lo sviluppo del software in C++.

2.3.1 Caratterizzazione della struttura ottima di una soluzione

Prendiamo in considerazione il percorso con tempo totale minimo che l'automobile può seguire dal punto iniziale sino alla stazione L (1,j) :

- Se j=1 c'è un solo percorso da seguire,il tempo impiegato sarà dunque dato dalla somma tra il tempo di ingresso in quella catena ed il tempo di lavoro di quella stazione.
- Se j=2,3,...n allora ci sono due condizioni possibili da valutare:

1. L'auto proviene da una stazione immediatamente precedente della stessa catena,L (i,j-1) allora dovrà considerarsi il tempo di trasferimento ε .
2. L'auto proviene da una stazione appartenente ad una catena differente L (k,j-1) allora dovrà essere considerato il tempo di migrazione S.

Si noti che utilizzando questa struttura possiamo affermare che si tratta di una soluzione ottima che contiene al suo interno le soluzioni ottime dei suoi sottoproblemi. Scegliendo infatti di volta in volta il tempo minimo rispetto ai tempi di trasferimento siamo sicuri di giungere ad una soluzione ottima del problema finale.

2.3.2 Definizione ricorsiva del valore di una soluzione ottima

Utilizzando la nomenclatura introdotta ad inizio capitolo e la struttura ottima precedentemente caratterizzata possiamo individuare come il valore della soluzione ottima sarà restituito dalla seguente equazione di ricorrenza:

$$f(i,j) = \begin{cases} l(i)+T(i,1), & j=1 \\ \min(f(i,j-1)+T(i,j)+\varepsilon(i,j-1), f(k,j-1)+T(i,j)+S(i,j-1)), & j>1 \end{cases}$$

$$F' = \min(f(k,n)+U(k)) \text{ con } k=1....m$$

2.3.3 Calcolo del tempo minimo

Per ricostruire tale soluzione si è scelta una implementazione in C++ che ci permette di individuare la soluzione migliore utilizzando come principio fondamentale la precedente equazione.Si tratta dell'algoritmo *fastest_way()* presente nella sezione 'Codice' allegata di seguito. Sostanzialmente possiamo dire che il tempo totale minimo viene calcolato con il tempo di ingresso più il tempo di lavoro della prima stazione della catena corrente se il numero delle stazioni è pari ad uno. Altrimenti il tempo di percorso totale migliore è pari al minimo tra:

- il tempo di percorrenza migliore calcolato sino alla stazione precedente per la stessa catena più il tempo di lavoro della stazione corrente più il tempo di trasferimento dalla stazione precedente della stessa catena sino alla stazione corrente L (i,j) .
- il tempo di percorrenza migliore calcolato sino alla stazione immediatamente precedente di una qualsiasi catena k dello stabilimento più il tempo di lavoro della stazione corrente più il tempo di trasferimento dalla stazione precedente della diversa catena sino alla stazione corrente L (i,j) .

2.3.4 Calcolo del percorso più rapido

Anche in questo caso la parte implementativa in C++ si riferisce al metodo *print_solution()*, nella sezione 'Codice' allegata.L'obbiettivo principale di questo algoritmo è quello di ricostruire quelle che sono state di volta in volta le scelte effettuate relative agli indici delle catene attraversate nel calcolo della soluzione migliore attraverso fastest_way().

2.4 Implementazione in C++

L'implementazione e la risoluzione della problematica proposta dalla traccia avviene mediante il linguaggio e C++ di cui si è cercato di sfruttarne a pieno le potenzialità.In particolar modo si mantengono le caratteristiche del linguaggio di astrazione ed incapsulamento determinando come publici e privati i membri delle varie classi e concedendo all'utente finale di approcciarsi soltanto con una interfaccia dell'intera implementazione.Ogni classe è stata poi dotata di un costruttore di default e di un distruttore;in particolar modo laddove fosse necessario si è effettuato un overloading del costruttore di default per permettere l'inizializzazione degli oggetti mediante argomenti definiti in input.Sono state utilizzate poi le cosiddette funzioni di accesso per mantenenere l'integrità e la coerenza dei dati,utilizzando una 'set' per impostare le informazioni ed una get per ottenere le stesse.Si è fatto uso inoltre di alcuni oggetti appartenenti alla STL del C++ come ad esempio i vettori apparteneti alla classe vector e i rispettivi iteratori.Si èinfine cercato di suddividere al meglio il codice separando la definizione delle classi dalla loro reale implementazione.

2.4.1 Diagramma delle classi

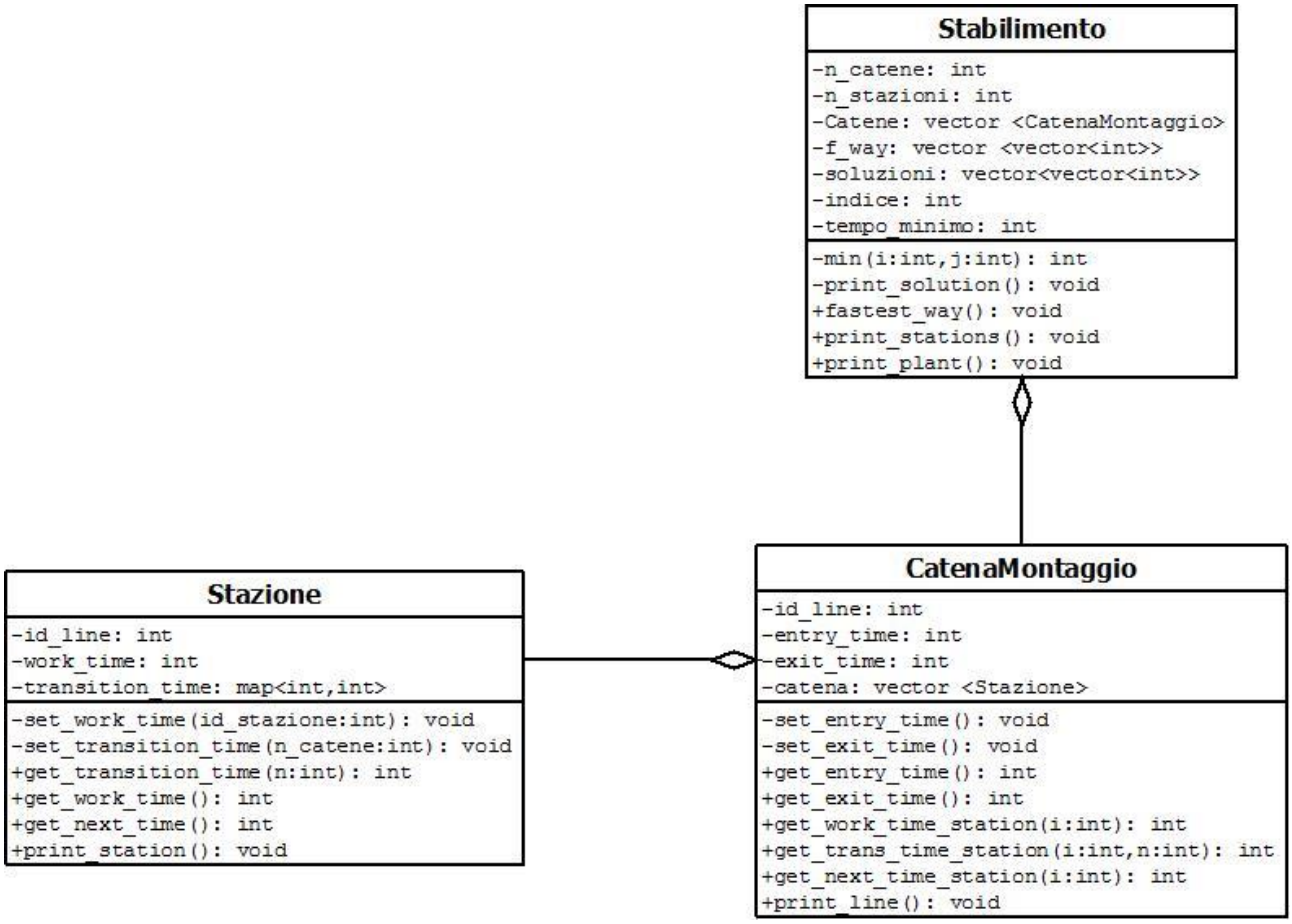

Figura 1:Diagramma UML Catena Di Montaggio

LEGENDA

- + : Membro Public
- - : Membro Private
- -<> : Operatore di composizione,è parte di

2.5 Definizione delle classi

- ***STAZIONE:*** è la classe più semplice implementata.Si istanzierarro da essi oggetti in grado di mantenere il loro tempo di trasferimento verso altre catene,oltre al loro tempo di lavoro.
- ***CATENAMONTAGGIO:*** è composta da una serie di stazioni di cui permette di ottenere i tempi di lavoro e di trasferimento oltre a mantenere i propri tempi di entrata ed uscita
- ***STABILIMENTO:*** La classe principale dell'implementazione.Contiene info riguardo il numero di catene e di stazioni,oltre alle strutture necessarie per il calcolo del tempo di percorrenza migliore.E' composta da una serie di catene di montaggio il cui numero di stazioni è quello definito in input.

2.6. Definizione dei metodi

A partire dalla definizione delle classi e dalle proprietà cui esse sono composte si è scelto di definire i seguenti metodi che soddisfano i requisiti proposti dalla traccia e le informazioni richieste.Vengono inoltre utilizzate le notazioni presenti nella 'Legenda' della sezione 'Diagramma delle classi' per differenziare i metodi appartenenti all'intefaccia publica da quelli appartenenti all'interfaccia privata.

2.6.1 Stazione

-:*set_transition_time(int n_catene) :* permette di settare il tempo di trasferimento verso le n catene dello stabilimento.

+ : *get_transition_time(int n)* : restituisce il tempo di trasferimento alla catena n definita in input.

+ : *get_work_time()*: restituisce il tempo di lavoro della stazione.

*+:get_next_time():*restituisce il tempo di trasferimento verso la stazione successiva della medesima catena della stazione corrente.

+ : *print_station():* stampa le informazioni riguardo la stazione

2.6.2 CatenaMontaggio

+ : *get_entry_time()* : restituisce il tempo di entrata della catena

+ : *get_exit_time()* : restituisce il tempo di uscita della catena

+ : *get_tran_time_station(int i,int n)* : resutisce il tempo di trasferimento della stazione i-sima verso la catena n-sima dello stabilimento

+ : *get_work_time_station(int i)* : restituisce il tempo di lavoro della stazione i-sima della catena

+:get_next_time_station(int i): restituisce il tempo di trasferimento alla stazione successiva della stessa catena della stazione i-sima

+ : *print_line()* : stampa le info riguardo la catena corrente

2.5.2 Stabilimento

+ : *fastest_way()* : variante dell'algoritmo trattato durante il corso che si basa sulla relazione di ricorrenza definita nella sezione 'Definizione ricorsiva del valore di una soluzione ottima'. Il suo scopo è quello di trovare il percorso migliore per l'automobile rispetto al numero di catene e stazioni definiti in input

- : *min()* : supporto all'algoritmo fastest_way,calcola effettivamente il minimo tra gli argomenti definiti nella relazione di ricorrenza,nel caso in cui j=2....n.

+ : print_stations() : Stampa i tempi di entrata e uscita delle varie catene oltre ai tempi di percorrenza compiuti dall'automobile rispetto al proprio percorso

- : *print_solutions()* : Ricostruisce e stampa il percorso migliore effettuato dall'automobile a partire dall'indice rispetto cui è stato individuato il tempo minimo totale in uscita.

+ : *print_plant()* : Stampa le informazioni di tutte le catene e relativamente di tutte le stazioni presenti all'interno dello stabilimento.

2.7 Test di Esecuzione

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/catena$ ./catena
**************************************************
*            ABATE ANTONIO 0124/874              *
*             Catena di Montaggio                *
**************************************************

[1]Inserire il numero di catene della fabbrica: 2
[2]Inserire il numero di stazioni della fabbrica: 6

---------------------CATENA 0--------------------
TE: 4

STAZIONE [0]
TL: 8
TT[0]: 6
TT[1]: 4
------------
STAZIONE [1]
TL: 6
TT[0]: 7
TT[1]: 3
------------
STAZIONE [2]
TL: 10
TT[0]: 2
TT[1]: 3
------------
STAZIONE [3]
TL: 8
TT[0]: 1
TT[1]: 10
------------
STAZIONE [4]
TL: 4
TT[0]: 7
TT[1]: 1
------------
STAZIONE [5]
TL: 7
------------

TX: 7
```

Figura 2: Test 1A,Catena di Montaggio

```
----------------------CATENA 1--------------------
TE: 3

STAZIONE [0]
TL: 2
TT[0]: 9
TT[1]: 8
------------
STAZIONE [1]
TL: 10
TT[0]: 3
TT[1]: 1
------------
STAZIONE [2]
TL: 3
TT[0]: 4
TT[1]: 8
------------
STAZIONE [3]
TL: 6
TT[0]: 10
TT[1]: 3
------------
STAZIONE [4]
TL: 3
TT[0]: 9
TT[1]: 10
------------
STAZIONE [5]
TL: 8
------------

TX: 7

------------------TEMPI DI PERCORENZA-----------------
Stazione : [0]   [1]     [2]     [3]     [4]     [5]     [exit]

Catena  0: 12    20      36      38      43      57      64
Catena  1: 5     23      26      40      46      52      59

Tempo Migliore: 59
Catena di Provenienza: 1

MIGLIORE SOLUZIONE TROVATA
Stazione 5: Catena 1
Stazione 4: Catena 0
Stazione 3: Catena 0
Stazione 2: Catena 1
Stazione 1: Catena 0
Stazione 0: Catena 1
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/catena$
```

Figura 3: Test 1B,Catena di Montaggio

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/catena$ ./catena
**************************************************
*            ABATE ANTONIO 0124/874              *
*             Catena di Montaggio                *
**************************************************

[1]Inserire il numero di catene della fabbrica: 3
[2]Inserire il numero di stazioni della fabbrica: 3

----------------------CATENA 0--------------------
TE: 4

STAZIONE [0]
TL: 8
TT[0]: 6
TT[1]: 4
TT[2]: 6
------------
STAZIONE [1]
TL: 7
TT[0]: 3
TT[1]: 10
TT[2]: 2
------------
STAZIONE [2]
TL: 3
------------

TX: 7

----------------------CATENA 1--------------------
TE: 8

STAZIONE [0]
TL: 10
TT[0]: 4
TT[1]: 7
TT[2]: 1
------------
STAZIONE [1]
TL: 7
TT[0]: 3
TT[1]: 7
TT[2]: 2
------------
STAZIONE [2]
TL: 9
------------

TX: 1
```

Figura 4: Test 2A,Catena di Montaggio

```
----------------------CATENA 2---------------------
TE: 8

STAZIONE [0]
TL: 3
TT[0]: 1
TT[1]: 3
TT[2]: 4
------------
STAZIONE [1]
TL: 8
TT[0]: 6
TT[1]: 10
TT[2]: 3
------------
STAZIONE [2]
TL: 3
------------

TX: 10

------------------TEMPI DI PERCORENZA-----------------
Stazione : [0]   [1]     [2]     [exit]

Catena  0: 12    19      25      32
Catena  1: 18    21      37      38
Catena  2: 11    23      24      34

Tempo Migliore: 32
Catena di Provenienza: 0

MIGLIORE SOLUZIONE TROVATA
Stazione 2: Catena 0
Stazione 1: Catena 0
Stazione 0: Catena 2
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/catena$
```

Figura 5: Test 2B,Catena di Montaggio

2.7.1 Commenti

Nei test si indica con ‘TE’ il tempo di entrata nella catena i-sima e come ‘TX’ il relativo tempo di uscita.Il tempo di lavoro di una stazione è espresso con ‘TL’ mentre il tempo di trasferimento verso una data stazione per la stazione corrente è assiociato alla sigla ‘TT’ seguito dall’indice della stazione verso cui ci si sposta.Come si evince dai test il software rispetta i riequisiti richiesti dalla traccia applicando per varie configurazioni in principio di programmazione dinamica precedentemente prefissato.Stampando infatti a video quelle che sono le ricostruzioni dei percorsi migliori si può notare come l'algoritmo calcoli in modo efficiente il tempo minimo di percorrenza totale attraverso scelte dettate dal rispetto della proprietà di sottostruttura ottima,propria a tale tipo di problema.Si noti inoltre che,come da requisiti,non sono ammesse meno di una catena di montaggio e non meno ovviamente di una stazione.Condiserare i test 1A ed 1B,così come i test 2A e 2B come due viste della medesima esecuzione,in particolar modo il test 1 si riferisce ad uno stabilimento con 2 catene e 6 stazioni,ed il test 2 ad uno stabilimento con 3 catene e 3 stazioni.

2.7.2 Elenco dei sorgenti

1. main.cpp
2. Header.h
3. Stazione.h
4. Stazione.cpp
5. CatenaMontaggio.h
6. CatenaMontaggio.cpp
7. Stabilimento.h
8. Stabilimento.cpp

2.7.3 Note compilazione

- Compilatore: GCC
- Versione: 4.9.2-10
- Ambiete di sviluppo: Debian Linux 8.2
- Editor: Sublime Text

Per la compilazione in ambiente Linux posizionarsi nella cartella in cui sono presenti i file sorgenti e da terminale digitare:

- g++ -o CatenaMontaggio main.cpp Stazione.cpp Stazione.h CatenaMontaggio.cpp CatenaMontaggio.h Stabilimento.cpp Stabilimento.h Header.h

Per l'esecuzione posizionarsi nella cartella in cui è stata effettuata la compilazionee da terminale digitare:

- ./CatenaMontaggio

Coda di max priorità

3.1 Definizione del problema

La seconda traccia richiede esplicitamente di implementare una coda di priorità massima,una struttura dati che serve a mantenere un insieme di elementi ad ognuno dei quali è associata una chiave.In base a quest'ultima infatti si può individuare la precedenza di un elemento nell'insieme rispetto ad un altro.Il caso trattato descrive una coda di priorità massima che supporta le seguenti di inserimento,restituzione ed estrazione del massimo,oltre all'incremento della chiave di un elemento.Gestiste nella maniera opportuna si può ridurre il tempo delle operazioni richieste sino ad una complessità minima,sfruttando come struttra di supporto una estensione degli alberi binari di ricerca quali gli alberi binari Red-Black.

3.2 Analisi dell'implementazione

Anche per la seconda traccia l'implementazione in C++ discende dai concetti teorici.In particolar modo si dovrà tenere conto per essa delle definizioni,proprietà ed operazioni fondamentali degli Alberi Red-Black,oltre alle funzionalità legate ad una coda di massima priorità.Tali aspetti verranno esplicitati di seguito,esplicitando anche gli elementi del codice che da essi derivano.

3.2.1 Alberi binari Red-Black

Gli alberi binari red-black come detto sono una struttura dati gerarchica che estendono,in termini di complessità minima delle oprazioni di modifica,quella che è la funzionalità degli alberi binari di ricerca.Rispetto

a questi ultimi infatti ne mantengono le medesime proprietà,ma limitano i costi conservando una ulteriore informazione,quella del colore di un singolo nodo,necessaria per mantenere la struttura correttamente bilanciata al seguito delle operazioni di modifica.Possiamo definire dunque che il nodo della struttura *Albero Binario Red-Black* mantiene le seguenti informazioni:

- Padre
- Figlio destro
- Figlio sinistro
- chiave,o *key*
- dati

In base alle informazioni contenute in ogni elemento e sia *z* un nodo in albero binario red-black possiamo definire che tale struttura mantiene sempre le seguenti proprietà e caratteristiche,ereditate dagli semplici alberi binari:

- Se *y* è u nodo del sottoalbero sinistro di *z*, allora *key [y]* < *key* [z] .
- Se *y* è un nodo del sottoalbero destro di *z*,allora *key [y]* > *key [z]* .

Verranno inoltre conservate le condizioni proprie degli *Alberi Red-Black* fini al mantenimento della struttura bilanciata,quali:

- Ogni nodo può essere o *rosso* o *nero.*
- Il nodo radice è *nero.*
- I nodi sentinella *nil* sono neri.
- Lo stesso percorso da un nodo ad una foglia contiene lo stesso numero di nodi neri.
- I figli di un nodo rosso sono neri.

Particolare attenzione va prestata al nodo sentinella definito come nodo *nil*,utilizzato per non trattare diversamente i puntatori ai nodi dai puntatori NULL.Infatti invece di utilizzare un puntatore NULL è possibile utilizzare un solo particolare Nodo definito sentinella per risparmiare spazio in memoria,e far puntare ad esso la radice e tutti i nodi foglia.Ciò è quanto realizzato nella risoluzione della traccia.L'utilizzo di questi particolari alberi è dovuto al fatto che,gestiti opportunamente e conservando le

proprietà di cui prima,mantengono un tempo costante per le operazioni di modifica,quali inserimento e cancellazione.

3.2.2 Operazione di Inserimento

L'operazione di inserimento di un nodo *z* rientra nelle operazioni di modifica nell'albero che mantengono un costo pari all'altezza dell'albero,O log (n) ,mantenedo costantemente la struttura bilanciata.Questo è possibile grazie all' autobilanciamento che avviene quando si effettua tale operazione,permettendo che le prestazioni non degradino.Tale funzione è eseguita come una normale operazione di inserimento in un albero binario colorando inizialmente il nodo di rosso,per poi invocare una *insert-fixup()* per ripristinare le proprietà eventualmente violate,in particolare all'inserimento di un nodo *z* accade*:*

- Si ricerca nell'albero un elemento con la stessa chiave k del nodo che si creca di inserire e se,un nodo simile non è presente si collega il nuovo nodo all'ultimo nodo dall'algoritmo di riceca,confrontandone la chiave con la chiave del padre per stabilire se esso sarà figlio destro o sinistro dello stesso.
- Si verificano eventuali proprietà violate dall'inserimento in particolar modo la radice potrebbe non essere più nera o,i figli di un nodo rosso essere a loro volta rossi.Si definisce un nodo di questo tipo un nodo problematico e rispetto a quest'ultimo si va incontro a sei simmetriche casistiche per ripristinare le condizioni violate
- Si verifica se siamo nel caso 1 ovvero lo zio *y* di *z* è rosso,in questo caso si colora di nero il padre di *z*,di nero y,e di rosso il nonno il nonno di *z*.Viene spostato infine *z* verso l'alto facendo del nonno di z il nuovo z.
- Si verifica se siamo nel caso 2 ovvero lo zio *y* di *z* è nero e lo stesso *z* è un figlio destro,in questo caso si assume come soluzione quella del caso successivo attraverso una rotazione di *z* a sinistra.
- Si verifica se siamo nel caso 3 ovvero lo zio *y di* z *è* nero e lo stesso z è un figlio sinistro,in questa casistica si colora di nero il padre di z,di rosso il nonno di z,e si ruota a destra il nonno di z.

Una operazione di inserimento eseguita con il supporto della insert-fixup e delle operazioni di rotazione,che rappresentano un semplice aggiornamento dei puntatori dei nodi,conserva il costo O(log (n)) legato a questa tipologia di modifica dell'albero.

3.2.4 Operazione di Cancellazione

L'operazione di cancellazione in un Albero binario Red-Black è una naturale estensione dlla medesima operazione compiuta con i normali alberi di ricerca,con la caratteristica di autobilanciare l'albero al seguito della stessa conservandone le proprietà.Anche in questo caso si fa ricorso ad una funzione di supporto,la *delete-fixup()* fine a ripristinare le condizioni eventualmente violate.In particolar modo all'atto di cancellazione di un nodo *z* da un *Albero Binario Red-Black* accade:

- Si effettua una normale cancellazione da un albero tenendo conto che se il nodo *z* non ha figli allora basta semplicemente cancellarlo;se esso ha un solo figlio si elemina *z* e si collega suo padre a suo figlio;se *z* ha due figli se ne ricerca il successore,si scambia la chiave di *z* con la chiave del successore cancellano poi quest'ultimo.
- Si verifica eventuali proprietà violate dalla cancellazione,in particolar modo ciò si verificherebbe se il nodo cancellate è un nodo nero,rispetto cui ci troveremmo di fronte a un nodo problematico con casistiche e soluzioni da adottare per il ripristino delle condizioni alterate.In questo caso infatti bisogna controllare il colore del fratello *y* di *z* e del colore dei figli dello stesso *y*,per stabilire le opportune strategie di ripristino.In particolare:
- Si verifica se ci troviamo nel caso 1,in cui il fratello *y di* z è rosso.In questo caso si colora di nero *y*,si colora di rosso il padre di *z*,si effettua una rotazione a sinistra e si aggiorna *y*.
- Si verifica se ci troviamo nel caso 2,ovvero il fratello *y* di *z* è nero ed entrambi i figli di *y* sono neri.Si cambia in questo caso il colore di *y* e si aggiorna *z*.
- Si verifica se ci troviamo nel caso 3,rispetto cui il fratello *y* di *z* è nero il figlio sinistro di *y* è rosso ed il destro è nero.In questo caso si colora di nero il figlio sinistro di *y* si colora di rosso lo stesso *y*,si effettua una rotazione a destra e si aggiorna *y*

- Si verifica se ci troviamo nel caso 4,in cui il fratello *y* di *z è* nero il figlio destro di *y* è rosso.In questo caso si imposta il colore di *y* come quello del padre,si colora di nero il padre di *z*,di nero anche il figlio destro di *y* e si effettua una rotazione a sinistra.

Effettuando in questo modo la cancellazione saremo sicuri che la struttura sarà sempre bilanciata conservando dunque il costo di complessità di questo tipo di operazione pari a O(log (n)).

3. 4 Coda di Priorità

La coda di priorità implementata si basa opportunanemente sulle operazioni possibili mediante Alberi Red-black al fine di effettuare le operazioni ad essa richieste con costo minimo.In particolar modo utilizzerà tale struttura per memorizzare i suoi elementi,che verrano poi recuperati attraverso le operazioni di interrogazione o di modifica sull'albero che,essendo costantemente bilanciato richiederanno sempre un costo di O (log(n)) .Trattandosi poi di una coda con priorità massima le operazioni saranno:

- *Insert :* Operazione di inserimento di unn nodo all'interno della coda
- *Maximum:* Restituzione dell'elemento con chiave massima presente nella coda.
- *Extract-max:* Estrazione dell'elemento con chiave massima e cancellazione dello stesso dall'insieme degli elementi appartenenti alla coda.
- Increase-key: Incrementi del valore della chiave *k* di un nodo presente nella coda.

In particolar modo una operazione di inserimento sfrutterà l'inserimento in un albero Red-Black,Maximum una di restituzione del massimo da parte dell'albero ed infine exctract-max ed increase-key una cancellazione di un nodo dall'albero.Tale interfaccia poi sarà restituita all'utente finale al

termine dell'implementazione,per utilizzare opportunamente le funzionalità espresse nei requisiti.

3.5 Implementazione in C++

La stesura del codice è una diretta conseguenza di quanto precedentemente analizzato,ed essa avviene sfruttando opportunamente l'implementazione delle classi in C++.In questa particolare casistica si è dovuto ricorrere ad elementi appartenenti alla classe *string* propria del linguaggio utilizzato,per mantere l'informazione legata al colore dei singoli nodi.In relazione a questi ultimi è stato inoltre necessario l'utilizzo dei puntatori ad oggetti per garantire la correttezza delle operazioni sviluppate.Si è dotato infatti ogni nodo di un puntatore al figlio sinistro al figlio destro ed al padre,oltre ad implementare un particolare nodo sentinella,quale il nodo NIL facendo puntare ad esso le la radice e le foglie dell'albero,risparmiando spazio in memoria.Sono state sfruttate,in corrispondenza dei puntatori la funzione *new* per l'allocazione dinamica della memoria e la *delete* per una esplicita cancellazione della stessa.Per ogni classe è stato definito un costruttore per l'inizializzazione ed un costruttore per la sua cancellazione,oltre a suddividere il codice in file per la definizione delle classi separati dalla loro reale implementazione.

3.5.1 Diagramma delle classi

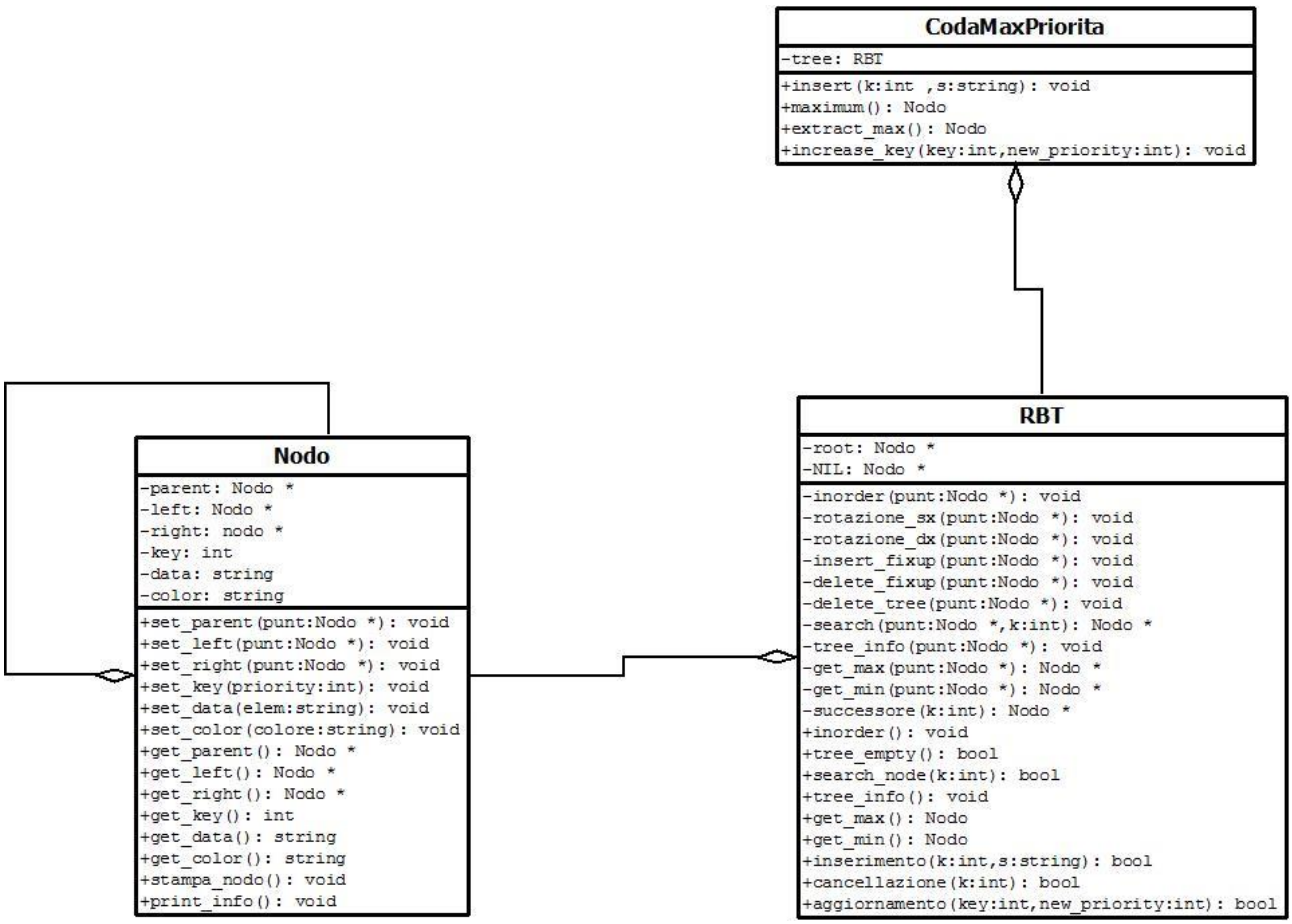

Figura 6:Diagramma UML Coda Max Priorità

LEGENDA

- \+ : Membro Public
- \- : Membro Private
- -<> : Operatore di composizione,è parte di

3.6 Definizione delle classi

- ✓ ***NODO:*** La classe nodo risulta il singolo elemento appartenente al succesivo albero.Esso deve necessariamente mantenere puntatori al padre,figlio destro e figlio sinistro olre alle informazione sul colore e sui dati satellite.
- ✓ ***RBT:*** La classe rispetto cui sarà implementata la struttura dati quale l'*Albero binario Red_Black.* E' caratterizzata da un nodo *root* che identifica la radice dell'albero ed un nodo *NIL* la cui utilità è stata precedentemente introdotta.
- ✓ ***CODAMAXPRIORITA:*** La classe finale che strutta l'implementazione di RBT.

3.7 Definizione dei metodi

3.7.1 Nodo

*+ : set_parent(Nodo *punt)* : imposta correttamente il padre del nodo corrente a punt.

*+ : set_left(Nodo *punt)* : imposta correttamente il figlio sinistro a punt.

*+ : set_right(Nodo *punt)* : imposta correttamente il figlio destro a punt.

+ : *set_key(int priority)* : setta il valore della chiave per il nodo punt.

+ : *set_data(string elem)* : imposta il valore delle informazioni contenute nel nodo corrente.

+ : *set_color(string colore)* : definisce il colore del nodo corrente.

+ : *get_parent()* : restituisce un puntatore al padre del nodo.

+ : *get_left()* : restutuisce un puntatore al figlio sinistro del nodo.

+ : *get_right()*: restituisce un puntatore al figlio sinistro del nodo.

+ : *get_key()* : restituisce il valore int della chiave.

+ : *get_data()* :restituisce i dati satellite contenuti nel nodo corrente.

+ : *get_color()*:restituisce sottoforma di stringa il colore del nodo corrente.

+ : *stampa_nodo()*:stampa le informazioni necessarie del nodo quali dati colore e priorità

+ : *print_info()*:stampa le info riguardanti il padre,il figlio destro e il figlio sinistro del nodo.

3.7.2 RBT

- : *inorder(Nodo *punt)*;visita dell'albero inorder,ovvero prima il sottoalbero sinitro,poi il sottoalbero destro ed infine la radice.

- : *rotazione_sx(Nodo *punt):* Operazione di supporto all'inserimento e alla cancellazione in un albero red-black.Si tratta di semplici aggiornamenti dei puntatori ai nodi dell'albero.

- : *rotazione_dx(Nodo *punt)*:Simmetrica alla operazione precedente

- : *insert_fixup(Nodo *punt)*:Operazione che segue quella di inserimento del nodo preliminare.Di tipo private di permette di ripristinare le proprietà dell'albero al seguito della sua modifica

considerando tutti i casi possibili e le rispettive soluzioni definite nella sezione 'Alberi Binari Red-Black 3.2.1'

- *: delete_fixup(Nodo *punt)*:Metodo private che sussegue la cancellazione,ripristinado eventuali condizioni violate secondo le condizioni e le rispettive soluzioni espresse in 'Alberi Binari Red-Black 3.2.1'.

- *: delete_tree(Nodo *punt):*Cancellazione esplicita dei nodi dell'albero.Appartiene ai metodi privati della classe.

- *: search(Nodo *punt,int k)*:Algoritmo di ricerca di un elemento nell'albero.

- *: tree_info(Nodo *punt)*:Stampa le informazioni complete,padre figlio destro e figlio sinistro,di ogni nodo.

- *: get_max(Nodo *punt):*Restituisce il massimo elemento nell'albero.

- *: get_min(Nodo *punt)*:Restituisce il minimo elemento nell'albero.

- *: successore(int k)*:Individuazione del successore del nodo con chiave k.

+*:inorder():* Overloading che richiama opportunamente l'omonimo metodo dell'interfaccia proteta.

+ *: tree_empty()*:funzione predicativa,ci indica se l'albero è vuoto.

+*:search_node(int k):* richiama il metodo *search* dell'interrfaccia privata al fine di verificare o meno un nodo con chiave k presente nell'albero.

+*tree_info():* Overloading del metodo dell'interfaccia privata

+*:get_max():* richiama l'omonimo metodo dell'interfaccia privata

+*:get_min():* richiama l'omonimo metodo dell'interfaccia privata

+ *: inserimento (int k,string s)*:Operazione preliminare di inserimento del nodo così come avviene in un semplice albero di ricerca.

+ *: cancellazione(int k)*:Operazione preliminare di cancellazione,tenendo conto delle casistiche precedentemente esposte.

+*:aggiornamento(int key,int new_priority):* utilizzata dalla coda per effettuare correttamente l'operazione di incremento priorità del nodo con chiave *key*

3.7.3 CodaMaxPriorita

+ *: insert(int k,string s)*:Operazione di inserimento nella coda di un nuovo nodo con priorità *k* e con dati satelite *s*.

+ *: maximum():*Restituisce la chiave massima tra i nodi presenti nella coda.

+ *: exctract_max()*: Restituisce e cancella dalla coda il nodo con chiave massima.

+ *:increase_key(int priority,int key)*:Incrementa la priorità di un nodo all'interno della coda,se esiste nella coda un nodo di tale priorità.Tale operazione è possibile se e solo se *priority* è maggiore della chiave *key* che si cerca di incrementare e se *key* è effettivamente presente nella coda.

3.8 Test di esecuzione

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/coda$ ./CodaMaxPriorita
**************************************************
*            ABATE ANTONIO 0124/874              *
*              Coda Max Priorità                 *
**************************************************

--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 5
Inserire dati: Progetto
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 11
Inserire dati: Algoritmi
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 2
--------------ELEMENTO MASSIMO------------------
Priorità: 11
Dati: Algoritmi
Colore: ROSSO
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA:
```

Figura 7:Test 1,Coda Max Priorità

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/coda$ ./CodaMaxPriorita
**************************************************
*            ABATE ANTONIO 0124/874              *
*              Coda Max Priorità                 *
**************************************************

--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 4
Inserire dati: Algoritmi
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 3
--------------ELEMENTO MASSIMO ESTRATTO----------
Priorità: 4
Dati: Algoritmi
Colore: NERO
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA:
```

Figura 8:Test 2,Coda Max Priorità

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/coda$ ./CodaMaxPriorita
**************************************************
*            ABATE ANTONIO 0124/874              *
*              Coda Max Priorità                 *
**************************************************

--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 7
Inserire dati: Progetto
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 4
Inserie priorità da modificare: 7
Inserire nuova priorità: 2
Errore nell'aggiornare la priorità
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 4
Inserie priorità da modificare: 7
Inserire nuova priorità: 9
Priorità correttamente aggiornata
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA:
```

Figura 9: Test 3,Coda Max Priorità

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/coda/coda2$ ./CodaMaxPriorita
**************************************************
*            ABATE ANTONIO 0124/874              *
*              Coda Max Priorità                 *
**************************************************

--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 1
Inserire dati: A
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 3
Inserire dati: B
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 6
Inserire dati: C
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 4
Inserie priorità da modificare: 3
Inserire nuova priorità: 6
ERRORE: nodo di priorita 6 gia presente nella Coda
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA:
```

Figura 10: Test 4,Coda Max Priorità

```
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/coda$ ./CodaMaxPriorita
**************************************************
*            ABATE ANTONIO 0124/874              *
*              Coda Max Priorità                 *
**************************************************

--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 1
Inserire priorità: 6
Inserire dati: Progetto
Nodo corretamente inserito
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 3
---------------ELEMENTO MASSIMO ESTRATTO----------
Priorità: 6
Dati: Progetto
Colore: NERO
--------SCEGLIERE L'OPERAZIONE DA COMPIERE--------
1)Inserimento
2)Visualizza Massima Priorità
3)Estrazione dell'elemento di Max Priorità
4)Incrementa Priorità
0)Uscita
SCELTA: 0
antonio@debian:~/Dropbox/Uni/02-IIAnno/ISemestre/ASD/progetto/coda$
```

Figura 11: Test 5,Coda Max Priorità

3.8.1 Commenti

Come si evince di test l'utente può interfacciarsi alla coda mediante un menù a tendina che espone le operazioni possibili sulla stessa. Viene mostrato a video le informazioni del nodo inserito,estratto o selezionato come priorità ,oltre a permettere una operazione di incremento di priorità solo se la nuova chiave è maggiore di quella precedentemente definita.

3.8.2 Elenco dei sorgenti

1. main.cpp
2. Header.h
3. Nodo.h
4. Nodo.cpp
5. RBT.h
6. RBT.cpp
7. CodaMaxPriorita.h
8. CodaMaxPriorita.cpp

3.8.3 Note compilazione

- Compilatore: GCC
- Versione: 4.9.2-10
- Ambiete di sviluppo: Debian Linux 8.2
- Editor: Sublime Text

Per la compilazione in ambiente Linux posizionarsi nella cartella in cui sono presenti i file sorgenti e da terminale digitare:

- g++ -o CodaMaxPriorita main.cpp Nodo.cpp Nodo.h RBT.cpp RBT.h CodaMaxPriorita.cpp CodaMaxPriorita.h Header.h

Per l'esecuzione posizionarsi nella cartella in cui è stata effettuata la compilazionee da terminale digitare:

- ./CodaMaxPriorita