Esame di Algoritmi e programmazione
Libreria personale studente.h

```
link NEWnode(Item val, link next);
void LISTinshead(LIST l, Item val);
link LISTinstail(link head, Item val);
Item LISTsearch(link head, Key k);
link LISTdelhead(link h);
link LISTdelkey(link h, Key k);
link LISTdelkeyR(link x, Key k);
Item LISTextrheadP(link *hp);
Item LISTextrkeyP(link *x, Key k);
link LISTsortins(link h, ITEM item);
Item LISTsortsearch(link h, Key k);
link LISTsortdel(link h, Key k);
void LISTshow(link h);
void LISTfree(link h);

BST BSTinit();
void BSTfree (BST bst);
int BSTcount (BST bst);
int BSTempty (BST bst);
Item BSTsearch(BST bst, Key k);
Item BSTmin(BST bst);
Item BSTmax(BST bst);
void BSTinsert_leafR(BST bst, Item x);
void BSTinsert_leafI(BST bst, Item x);
void BSTinsert_root(BST bst, Item x);
void BSTvisit (BST bst, int strategy);
link rotR(link h);
link rotL(link h);
link partR (link h, int r);
void BSTdelete (BST bst, Key k);
Item BSTselect (BST bst, int r);
Item BSTsucc (BST bst, Key k);
Item BSTpred (BST bst, Key k);

void IBSTinit(IBST ibst);
void IBSTfree (IBST ibst);
void BSTinsert (IBST ibst, Item x);
void IBSTdelete (IBST ibst, Item x);
Item IBSTsearch (IBST ibst, Item x);
Int IBSTcount (IBST ibst);
int IBSTempty (IBST ibst);
void IBSTvisit (IBST ibst, int strategy);

PQ PQinit(int maxN);
void PQfree (PQ pq);
int PQempty(PQ pq);
void PQinsert(PQ pq, Item val);
Item PQextractMax(PQ pq);
Item PQshowMax(PQ pq);
void PQdisplay(PQ pq);
int PQsize(PQ pq);
void PQchange(PQ pq, Item val);
void PQchange(PQ pq, int pos, Item val);

Heap HEAPinit(int maxN);
void HEAPfill(Heap h, Item val);
void HEAPsort(Heap h);
void HEAPdisplay(Heap h);
void HEAPfree(Heap h);
void HEAPify(Heap h, int i);
void HEAPbuild(Heap h);
int PARENT(int i);
int RIGHT(int i);
int LEFT(int i);

ST STinit(int maxN);
void STdisplay(ST st);
int STsize(int N);
int STinsert(ST st, Item val);
int STcount(ST st);
int STempty(ST st);
int STselect(ST st, int r);
int STcount (ST st);
void STinsert (ST st, Item val);
int STgetindex(ST tabella, ITEM item);
Item STsearch(ST st, Key k);
Key STsearchByIndex (ST st, int id);
void STdelete(ST st, Key k);
void STfree(ST st);
void STdisplay (ST st);
int hashU(char *v, int M);
int hash (Key k, int M);
int full(ST st, int i);
void STchangePrio (ST st, Item val, int i);

QUEUE QUEUEinit(int maxN);
int QUEUEempty(QUEUE q);
void QUEUEput(QUEUE queue, Item val);
Item QUEUEget(QUEUE q);

void UFinit (int N);
int UFfind (int p, int q);
void UFunion (int p, int q);

Graph GRAPHinit(int V);
void GRAPHfree(Graph G);
Graph GRAPHload(FILE *fin);
void GRAPHstore(Graph G, FILE *fout);
void GRAPHgetIndex(Graph G, char*label);
Edge EDGEcreate(int v, int w, int wt);
void GRAPHinsertE(Graph G, int id1, int id2, int wt);
void GRAPHremoveE(Graph G, int id1, int id2);
void GRAPHshow(Graph G);
void GRAPHedges(Graph G, Edge *a);
void insertE(Graph G, Edge e);
void removeE(Graph G, Edge e);
int randV(Graph G);
Graph GRAPHrand1(Graph G, int V, int E);
Graph GRAPHrand2(Graph G, int V, int E);
```

```
int GRAPHpath(Graph G, int id1, int id2);
void GRAPHpathH (Graph G, int id1, int id2);
void GRAPHbfs(Graph G, int id);
void bfs(Graph G, Edge e, int *time, int *pre, int *st);
void GRAPHdfs(Graph G, int id);
void dfsR(Graph G, Edge e, int *time, int *pre, int *post,
int *st);
int GRAPHscc(Graph G);
void SCCdfsR(Graph G, int w, int *scc, int *time0, int
time1, int *post);
int GRAPHcc(Graph G);
void dfsRcc(Graph G, int V, int id, int *cc);
Graph reverse(Graph G);
void GRAPHmstK(Graph G);
int mstE(Graph G, Edge *mst, Edge *a);
void GRAPHmstP(Graph G);
void mstV(Graph G, int *st, int *wt);
void GRAPHspD(Graph G, int id);
void GRAPHspBF(Graph G, int id);
void DAGrts(Graph G);
void TSdfsRnor(Graph G, int v, int *ts, int *pre, int *time);
void TSdfsRrev(Graph G, int v, int *ts, int *pre, int *time);

Graph GRAPHloadNotoVconelenco(FILE *fin);
Graph GRAPHloadNotoVsenzaelenco(FILE *fin);
Graph GRAPHloadSenzaV(FILE *fin);
int GRAPHlist2mat(Graph G);
int GRAPHmat2list(Graph G);

void BubbleSort(Item A[], int N);
void OptBubbleSort(Item A[], int N);
void SelectionSort(Item A[], int N);
void InsertionSort(Item A[], int N);
void ShellSort(Item A[], int N);
void CountingSort(Item A[], Item B[], int C[], int N, int k);

void QuickSort (Item *A, int N);
void QuickSortR(Item *A, int l, int r);
int partition(Item *A, int l, int r);
void MergeSort (Item *A, int N);
void MergeSortR(Item *A, Item * B, int l, int r);
void Merge(Item *A, Item *B, int l, int q, int r);
void BottomUpMergeSort (Item *A, int N);
```