Headers.h

• Ordinamenti

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

void MergeSort(Item *A, int N);
void MergeSortR(int *val, int l, int r);
void Merge(int *val,int l,int r);
void QuickSort(Item *A, int N);
void quicksortR(Item *A, int l, int r);
int partition (Item *A, int l, int r );
```

• Item

```
int KEYcompare(Key k1, Key k2);
Key KEYscan();
Item ITEMscan();
void ITEMshow(Item val);
int ITEMless(Item val1, Item val2);
int ITEMgreater(Item val1, Item val2);
int ITEMcheckvoid(Item val);
Item ITEMsetvoid();
```

• SET

```
typedef struct set *SET;
SET SETinit(int maxN);
void SETfill(SET s, Item val);
int SETsearch(SET s, Key k);
SET SETunion(SET s1, SET s2);
SET SETintersection(SET s1, SET s2);
int SETsize(SET s);
int SETempty(SET s);
void SETdisplay(SET s);
```

• Liste

```
link newNode(Item val, link next);
link listInsHead (link h, Item val);
link listInsTail(link h, Item val);
Item listSearch(link h, Key k);
link listDelHead(link h);
Item listExtrHeadP(link *hp);
link listDelKey(link h, Key k);
Item listExtrKeyP(link *hp, Key k);
link SortListIns(link h, Item val);
```

```
Item SortListSearch(link h, Key k);
link SortListDel(link h, Key k);
link listReverse(link x);

//STACK
typedef struct stack *STACK;

STACK STACKinit(int maxN);
int STACKempty(STACK s);
void STACKpush(STACK s, Item val);
Item STACKpop (STACK s);
```

- **PQ**

```
typedef struct pqueue *PQ;

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, int pos, Item val);
```

- **Heap**

```
typedef struct heap *Heap;

Heap HEAPinit(int maxN);
void HEAPfree(Heap h);
void HEAPfill(Heap h, Item val);
void HEAPsort(Heap h);
void HEAPify(Heap h,int i);
void HEAPdisplay(Heap h);
```

- **ST**

```
typedef struct symboltable *ST;

ST STinit(int maxN);
void STinsert(ST st, Item val); //int i
Item STsearch(ST st, Key k);
void STdelete(ST st, Key k);
void STdisplay(ST st);
void STfree(ST st);
int STcount(ST st);
int STempty(ST st);
char *STsearchByIndex(ST st,int i);
```

- ## Calcolo combinatorio

```
Int princ_molt(int pos,Livello *val,int *sol,int n,int count);
int disp_sempl(int pos,int *val,int *sol,int *mark,int n,int k,int count);
int disp_ripet(int pos,int *val,int *sol,int n,int k,int count);
int perm_sempl(int pos,int *val,int *sol,int *mark,int n,int count);
int perm_ripet (int pos,int *dist_val,int *sol,int *mark,int n,int n_dist,int count);
int comb_sempl(int pos,int *val,int *sol,int n,int k,int start,int count);
int comb_ripet(int pos,int *val,int *sol,int n,int k,int start,int count);

int powerset_disp_rip(int pos,int *val,int *sol,int k,int count);
int powerset_div_conq(int pos,int *val,int *sol,int k,int start,int count);
int powerset_comb_sempl(int *val,int k,int *sol);

void SP_rec(int n,int m,int p,int *sol,int *val);
void SetPartitions(int n,int *sol,int *val);
```

- ## BST

```
typedef struct binarysearchtree *BST;
BST BSTinit() ;
void BSTfree(BST bst);
int BSTcount(BST bst);
int BSTempty(BST bst);
Item BSTsearch(BST bst, Key k);
void BSTinsert_leafI(BST bst, Item x);
void BSTinsert_leafR(BST bst, Item x);
void BSTinsert_root(BST bst, Item x);
Item BSTmin(BST bst);
Item BSTmax(BST bst);
void BSTvisit(BST bst, int strategy);
void BSTdelete(BST bst, Key k);
Item BSTselect(BST bst, int r);
Item BSTsucc(BST bst, Key k);
Item BSTpred(BST bst, Key k);
//Interval BST
typedef struct item { int low; int high; } Item;
Item ITEMscan();
Item ITEMsetNull();
ITEMcheckNull(Item val);
ITEMstore(Item val);
ITEMhigh();
ITEMlow();
```

```
ITEMoverlap(Item val1, Item val2);
ITEMeq(Item val1, Item val2);
ITEMlt(Item val1, Item val2);
ITEMlt_int(Item val1, int val2);
```

- **Graph**

```
typedef struct edge { int v; int w; int wt; } Edge;
typedef struct graph *Graph;

Graph GRAPHinit(int V);
int **MATRIXint(int r,int c,int val);
link NEW(int v,int wt,link next);
Void GRAPHfree(Graph G);
void GRAPHload(FILE *fin);
void GRAPHstore(Graph G, FILE *fout);
int GRAPHgetIndex(Graph G, char *label);
void GRAPHinsertE(Graph G, int id1, int id2);
void insertE(Graph G,Edge e);
void GRAPHremoveE(Graph G, int id1, int id2);
void GRAPHedges(Graph G, Edge *a);

int GRAPHpath(Graph G, int id1, int id2);
int pathR(Graph G,int v,int w,int *visited);
void GRAPHpathH(Graph G, int id1, int id2); //grafi non orientati
int pathRH(Graph G,int v,int w,int d,int *visited);

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 GRAPHcc(Graph G);
int dfsRcc(Graph G,int v,int id,int *cc);
int GRAPHscc(Graph G);
void SCCdfsR(Graph G,int w,int *scc,int *time0,int time1,int *post);
```

- **DAG: Ordinamento topologico**

```
void DAGrts(Dag D);
void TSdfsR(Dag D, int v,int *ts,int *pre,int *time);
```

- **UF**

```
static int *id,*sz;
void UFinit(int N);
int UFfind(int p,int q);
void UFunion(int p,int q);
```

- **Alberi_Ricoprenti_Minimi**

```
void GRAPHmstK(Graph G);
int mstE(Graph G, Edge *mst, Edge *a);

void GRAPHmstP(Graph G);
void mstV(Graph G, int *st, int *wt);
```

- **Cammini_Minimi (*st,*mindist)**

```
void GRAPHspD(Graph G, int id);
void GRAPHspBF(Graph G, int id);
```