
CAPITOLO 1. Introduzione

```
#include <stdio.h>
#define N 10000
main()
  { int i, p, q, t, id[N];
    for (i = 0; i < N; i++) id[i] = i;
    while (scanf("%d %d\n", &p, &q) == 2)
      {
        if (id[p] == id[q]) continue;
        for (t = id[p], i = 0; i < N; i++)
          if (id[i] == t) id[i] = id[q];
        printf(" %d %d\n", p, q);
      }
 }
```

```
    for (i = p; i != id[i]; i = id[i]) ;
    for (j = q; j != id[j]; j = id[j]) ;
    if (i == j) continue;
    id[i] = j;
    printf(" %d %d\n", p, q);
```

```
#include <stdio.h>
#define N 10000
main()
  { int i, j, p, q, id[N], sz[N];
    for (i = 0; i < N; i++)
```

```
      { id[i] = i; sz[i] = 1; }
    while (scanf("%d %d\n", &p, &q) == 2)
      {
        for (i = p; i != id[i]; i = id[i]) ;
        for (j = q; j != id[j]; j = id[j]) ;
        if (i == j) continue;
        if (sz[i] < sz[j])
             { id[i] = j; sz[j] += sz[i]; }
        else { id[j] = i; sz[i] += sz[j]; }
        printf(" %d %d\n", p, q);
      }
 }
-----
    for (i = p; i != id[i]; i = id[i])
      { int t = i; i = id[id[t]]; id[t] = i; }
    for (j = q; j != id[j]; j = id[j]) ;
      { int t = j; j = id[id[t]]; id[t] = j; }
```

CAPITOLO 2. Principi di progettazione degli algoritmi

```
int search(int a[], int v, int l, int r)
  { int i;
    for (i = l; i <= r; i++)
      if (v == a[i]) return i;
    return -1;
  }
-----
int search(int a[], int v, int l, int r)
  {
    while (r >= l)
      { int m = (l+r)/2;
        if (v == a[m]) return m;
        if (v < a[m]) r = m-1; else l = m+1;
      }
    return -1;
  }
```

CAPITOLO 3. Strutture dati elementari

```
#include <stdio.h>
int lg(int);
main()
  { int i, N;
    for (i = 1, N = 10; i <= 6; i++, N *= 10)
      printf("%7d %2d %9d\n", N, lg(N), N*lg(N));
  }
int lg(int N)
```

```
  {  int i;
     for (i = 0; N > 0; i++, N /= 2) ;
     return i;
  }
-----
#include <stdlib.h>
typedef int numType;
numType randNum()
  { return rand(); }
main(int argc, char *argv[])
  { int i, N = atoi(argv[1]);
    float m1 = 0.0, m2 = 0.0;
    numType x;
    for (i = 0; i < N; i++)
      {
        x = randNum();
        m1 += ((float) x)/N;
        m2 += ((float) x*x)/N;
      }
    printf("       Average: %f\n", m1);
    printf("Std. deviation: %f\n", sqrt(m2-m1*m1));
 }
-----
typedef struct { float x; float y; } point;
float distance(point a, point b);
-----
#include <math.h>
#include "Point.h"
float distance(point a, point b)
  { float dx = a.x - b.x, dy = a.y - b.y;
    return sqrt(dx*dx + dy*dy);
  }
-----
#define N 10000
main()
  { int i, j, a[N];
    for (i = 2; i < N; i++) a[i] = 1;
    for (i = 2; i < N; i++)
      if (a[i])
        for (j = i; j < N/i; j++) a[i*j] = 0;
    for (i = 2; i < N; i++)
      if (a[i]) printf("%4d ", i);
    printf("\n");
  }
-----
#include <stdlib.h>
main(int argc, char *argv[])
  { long int i, j, N = atoi(argv[1]);
    int *a = malloc(N*sizeof(int));
```

```
    if (a == NULL)
      { printf("Insufficient memory.\n"); return; }
    ...
-----
#include <stdlib.h>
int heads()
  { return rand() < RAND_MAX/2; }
main(int argc, char *argv[])
  { int i, j, cnt;
    int N = atoi(argv[1]), M = atoi(argv[2]);
    int *f = malloc((N+1)*sizeof(int));
    for (j = 0; j <= N; j++) f[j] = 0;
    for (i = 0; i < M; i++, f[cnt]++)
      for (cnt = 0, j = 0; j <= N; j++)
        if (heads()) cnt++;
    for (j = 0; j <= N; j++)
      {
        printf("%2d ", j);
        for (i = 0; i < f[j]; i+=10) printf("*");
        printf("\n");
      }
 }
-----
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "Point.h"
float randFloat()
  { return 1.0*rand()/RAND_MAX; }
main(int argc, char *argv[])
 { float d = atof(argv[2]);
   int i, j, cnt = 0, N = atoi(argv[1]);
   point *a = malloc(N*(sizeof(*a)));
   for (i = 0; i < N; i++)
     { a[i].x = randFloat(); a[i].y = randFloat(); }
   for (i = 0; i < N; i++)
     for (j = i+1; j < N; j++)
       if (distance(a[i], a[j]) < d) cnt++;
   printf("%d edges shorter than %f\n", cnt, d);
 }
-----
#include <stdlib.h>
typedef struct node* link;
struct node { int item; link next; };
main(int argc, char *argv[])
  { int i, N = atoi(argv[1]), M = atoi(argv[2]);
    link t = malloc(sizeof *t), x = t;
    t->item = 1; t->next = t;
    for (i = 2; i <= N; i++)
```

```
      {
        x = (x->next = malloc(sizeof *x));
        x->item = i; x->next = t;
      }
    while (x != x->next)
      {
        for (i = 1; i < M; i++) x = x->next;
        x->next = x->next->next; N--;
      }
    printf("%d\n", x->item);
  }
-----
link reverse(link x)
  { link t, y = x, r = NULL;
    while (y != NULL)
      { t = y->next; y->next = r; r = y; y = t; }
    return r;
  }
-----
    struct node heada, headb;
    link t, u, x, a = &heada, b;
    for (i = 0, t = a; i < N; i++)
      {
        t->next = malloc(sizeof *t);
        t = t->next; t->next = NULL;
        t->item = rand() % 1000;
      }
    b = &headb; b->next = NULL;
    for (t = a->next; t != NULL; t = u)
      {
        u = t->next;
        for (x = b; x->next != NULL; x = x->next)
          if (x->next->item > t->item) break;
        t->next = x->next; x->next = t;
      }
-----
typedef struct node* link;
struct node { itemType item; link next; };
typedef link Node;
void initNodes(int);
link newNode(int);
void freeNode(link);
void insertNext(link, link);
link deleteNext(link);
link Next(link);
 int Item(link);
-----
#include "list.h"
main(int argc, char *argv[])
```

```
  { int i, N = atoi(argv[1]), M = atoi(argv[2]);
    Node t, x;
    initNodes(N);
    for (i = 2, x = newNode(1); i <= N; i++)
      { t = newNode(i); insertNext(x, t); x = t; }
    while (x != Next(x))
      {
        for (i = 1; i < M; i++) x = Next(x);
        freeNode(deleteNext(x));
      }
    printf("%d\n", Item(x));
  }
-----
#include <stdlib.h>
#include "list.h"
link freelist;
void initNodes(int N)
  { int i;
    freelist = malloc((N+1)*(sizeof *freelist));
    for (i = 0; i < N+1; i++)
      freelist[i].next = &freelist[i+1];
    freelist[N].next = NULL;
  }
link newNode(int i)
  { link x = deleteNext(freelist);
    x->item = i; x->next = x;
    return x;
  }
void freeNode(link x)
  { insertNext(freelist, x); }
void insertNext(link x, link t)
  { t->next = x->next; x->next = t; }
link deleteNext(link x)
  { link t = x->next; x->next = t->next; return t; }
link Next(link x)
  { return x->next; }
int Item(link x)
  { return x->item; }
-----
#include <stdio.h>
#define N 10000
main(int argc, char *argv[])
  { int i, j, t;
    char a[N], *p = argv[1];
    for (i = 0; i < N-1; a[i] = t, i++)
      if ((t = getchar()) == EOF) break;
    a[i] = 0;
    for (i = 0; a[i] != 0; i++)
      {
```

```
        for (j = 0; p[j] != 0; j++)
          if (a[i+j] != p[j]) break;
        if (p[j] == 0) printf("%d ", i);
      }
    printf("\n");
  }
-----
int **malloc2d(int r, int c)
  { int i;
    int **t = malloc(r * sizeof(int *));
    for (i = 0; i < r; i++)
      t[i] = malloc(c * sizeof(int));
    return t;
  }
-----
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define Nmax 1000
#define Mmax 10000
char buf[Mmax]; int M = 0;
int compare(void *i, void *j)
  { return strcmp(*(char **)i, *(char **)j); }
main()
  { int i, N;
    char* a[Nmax];
    for (N = 0; N < Nmax; N++)
      {
        a[N] = &buf[M];
        if (scanf("%s", a[N]) == EOF) break;
        M += strlen(a[N])+1;
      }
    qsort(a, N, sizeof(char*), compare);
    for (i = 0; i < N; i++) printf("%s\n", a[i]);
  }
-----
#include <stdio.h>
#include <stdlib.h>
main()
  { int i, j, adj[V][V];
    for (i = 0; i < V; i++)
      for (j = 0; j < V; j++)
        adj[i][j] = 0;
    for (i = 0; i < V; i++) adj[i][i] = 1;
    while (scanf("%d %d\n", &i, &j) == 2)
      { adj[i][j] = 1; adj[j][i] = 1; }
  }
-----
#include <stdio.h>
```

```
#include <stdlib.h>
typedef struct node *link;
struct node
  { int v; link next; };
link NEW(int v, link next)
  { link x = malloc(sizeof *x);
    x->v = v; x->next = next;
    return x;
  }
main()
  { int i, j; link adj[V];
    for (i = 0; i < V; i++) adj[i] = NULL;
    while (scanf("%d %d\n", &i, &j) == 2)
      {
        adj[j] = NEW(i, adj[j]);
        adj[i] = NEW(j, adj[i]);
      }
  }
-----
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "Point.h"
typedef struct node* link;
struct node { point p;  link next; };
link **grid; int G; float d; int cnt = 0;
gridinsert(float x, float y)
  { int i, j; link s;
    int X = x*G +1; int Y = y*G+1;
    link t = malloc(sizeof *t);
    t->p.x = x; t->p.y = y;
    for (i = X-1; i <= X+1; i++)
      for (j = Y-1; j <= Y+1; j++)
        for (s = grid[i][j]; s != NULL; s = s->next)
          if (distance(s->p, t->p) < d) cnt++;
    t->next = grid[X][Y]; grid[X][Y] = t;
  }
main(int argc, char *argv[])
 { int i, j, N = atoi(argv[1]);
   d = atof(argv[2]); G = 1/d;
   grid = malloc2d(G+2, G+2);
   for (i = 0; i < G+2; i++)
     for (j = 0; j < G+2; j++)
       grid[i][j] = NULL;
   for (i = 0; i < N; i++)
     gridinsert(randFloat(), randFloat());
   printf("%d edges shorter than %f\n", cnt, d);
 }
-----
```

```
#include <math.h>
#include <stdlib.h>
typedef int numType;
#define R 1000
numType randNum()
  { return rand() % R; }
main(int argc, char *argv[])
  { int i, N = atoi(argv[1]);
    int *f = malloc(R*sizeof(int));
    float m1 = 0.0, m2 = 0.0, t = 0.0;
    numType x;
    for (i = 0; i < R; i++) f[i] = 0;
    for (i = 0; i < N; i++)
      {
        f[x = randNum()]++;
        m1 += (float) x/N;
        m2 += (float) x*x/N;
      }
    for (i = 0; i < R; i++) t += f[i]*f[i];
    printf("       Average: %f\n", m1);
    printf("Std. deviation: %f\n", sqrt(m2-m1*m1));
    printf("    Chi-square: %f\n", (R*t/N)-N);
 }
```

CAPITOLO 4. Tipi di dati astratti

```
void STACKinit(int);
 int STACKempty();
void STACKpush(Item);
Item STACKpop();
```

```
#include <stdio.h>
#include <string.h>
#include "Item.h"
#include "STACK.h"
main(int argc, char *argv[])
  { char *a = argv[1]; int i, N = strlen(a);
    STACKinit(N);
    for (i = 0; i < N; i++)
      {
        if (a[i] == '+')
          STACKpush(STACKpop()+STACKpop());
        if (a[i] == '*')
          STACKpush(STACKpop()*STACKpop());
        if ((a[i] >= '0') && (a[i] <= '9'))
          STACKpush(0);
        while ((a[i] >= '0') && (a[i] <= '9'))
```

```
          STACKpush(10*STACKpop() + (a[i++]-'0'));
      }
    printf("%d \n", STACKpop());
  }
-----
#include <stdio.h>
#include <string.h>
#include "Item.h"
#include "STACK.h"
main(int argc, char *argv[])
  { char *a = argv[1]; int i, N = strlen(a);
    STACKinit(N);
    for (i = 0; i < N; i++)
      {
        if (a[i] == ')')
          printf("%c ", STACKpop());
        if ((a[i] == '+') || (a[i] == '*'))
          STACKpush(a[i]);
        if ((a[i] >= '0') && (a[i] <= '9'))
          printf("%c ", a[i]);
      }
    printf("\n");
  }
-----
#include <stdlib.h>
#include "Item.h"
#include "STACK.h"
static Item *s;
static int N;
void STACKinit(int maxN)
  { s = malloc(maxN*sizeof(Item)); N = 0; }
int STACKempty()
  { return N == 0; }
void STACKpush(Item item)
  { s[N++] = item; }
Item STACKpop()
  { return s[--N]; }

-----
#include <stdlib.h>
#include "Item.h"
typedef struct STACKnode* link;
struct STACKnode { Item item; link next; };
static link head;
link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
```

```
void STACKinit(int maxN)
  { head = NULL; }
int STACKempty()
  { return head == NULL; }
STACKpush(Item item)
  { head = NEW(item, head); }
Item STACKpop()
  { Item item = head->item;
    link t = head->next;
    free(head); head = t;
    return item;
  }
-----
void UFinit(int);
 int UFfind(int, int);
 int UFunion(int, int);
-----
#include <stdio.h>
#include "UF.h"
main(int argc, char *argv[])
  { int p, q, N = atoi(argv[1]);
    UFinit(N);
    while (scanf("%d %d", &p, &q) == 2)
      if (!UFfind(p, q))
        { UFunion(p, q); printf(" %d %d\n", p, q); }
 }
-----
#include <stdlib.h>
#include "UF.h"
static int *id, *sz;
void UFinit(int N)
  { int i;
    id = malloc(N*sizeof(int));
    sz = malloc(N*sizeof(int));
    for (i = 0; i < N; i++)
      { id[i] = i; sz[i] = 1; }
  }
int find(int x)
  { int i = x;
    while (i != id[i]) i = id[i]; return i; }
int UFfind(int p, int q)
  { return (find(p) == find(q)); }
int UFunion(int p, int q)
  { int i = find(p), j = find(q);
    if (i == j) return;
    if (sz[i] < sz[j])
         { id[i] = j; sz[j] += sz[i]; }
    else { id[j] = i; sz[i] += sz[j]; }
  }
```

```
-----
void QUEUEinit(int);
 int QUEUEempty();
void QUEUEput(Item);
Item QUEUEget();
-----
#include <stdlib.h>
#include "Item.h"
#include "QUEUE.h"
typedef struct QUEUEnode* link;
struct QUEUEnode { Item item; link next; };
static link head, tail;
link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
void QUEUEinit(int maxN)
  { head = NULL; }
int QUEUEempty()
  { return head == NULL; }
QUEUEput(Item item)
  {
    if (head == NULL)
      { head = (tail = NEW(item, head)); return; }
    tail->next = NEW(item, tail->next);
    tail = tail->next;
  }
Item QUEUEget()
  { Item item = head->item;
    link t = head->next;
    free(head); head = t;
    return item;
  }
-----
#include <stdlib.h>
#include "Item.h"
static Item *q;
static int N, head, tail;
void QUEUEinit(int maxN)
  { q = malloc((maxN+1)*sizeof(Item));
    N = maxN+1; head = N; tail = 0; }
int QUEUEempty()
  { return head % N == tail; }
void QUEUEput(Item item)
  { q[tail++] = item; tail = tail % N; }
Item QUEUEget()
  { head = head % N; return q[head++]; }
```

```
-----
#include <stdlib.h>
static int *s, *t;
static int N;
void STACKinit(int maxN)
  { int i;
    s = malloc(maxN*sizeof(int));
    t = malloc(maxN*sizeof(int));
    for (i = 0; i < maxN; i++) t[i] = 0;
    N = 0;
  }
int STACKempty()
  { return !N; }
void STACKpush(int item)
  {
    if (t[item] == 1) return;
    s[N++] = item; t[item] = 1;
  }
int STACKpop()
  { N--; t[s[N]] = 0; return s[N]; }
-----
#include <stdio.h>
#include <math.h>
#include "COMPLEX.h"
#define PI 3.141592625
main(int argc, char *argv[])
  { int i, j, N = atoi(argv[1]);
    Complex t, x;
    printf("%dth complex roots of unity\n", N);
    for (i = 0; i < N; i++)
      { float r = 2.0*PI*i/N;
        t = COMPLEXinit(cos(r), sin(r));
        printf("%2d %6.3f %6.3f ", i, Re(t), Im(t));
        for (x = t, j = 0; j < N-1; j++)
          x = COMPLEXmult(t, x);
        printf("%6.3f %6.3f\n", Re(x), Im(x));
      }
 }
-----
typedef struct { float Re; float Im; } Complex;
Complex COMPLEXinit(float, float);
  float Re(Complex);
  float Im(Complex);
Complex COMPLEXmult(Complex, Complex);
-----
#include "COMPLEX.h"
Complex COMPLEXinit(float Re, float Im)
  { Complex t; t.Re = Re; t.Im = Im; return t; }
float Re(Complex z)
```

```
  { return z.Re; }
float Im(Complex z)
  { return z.Im; }
Complex COMPLEXmult(Complex a, Complex b)
  { Complex t;
    t.Re = a.Re*b.Re - a.Im*b.Im;
    t.Im = a.Re*b.Im + a.Im*b.Re;
    return t;
  }
-----
typedef struct complex *Complex;
Complex COMPLEXinit(float, float);
  float Re(Complex);
  float Im(Complex);
Complex COMPLEXmult(Complex, Complex);
-----
#include <stdlib.h>
#include "COMPLEX.h"
struct complex { float Re; float Im; };
Complex COMPLEXinit(float Re, float Im)
  { Complex t = malloc(sizeof *t);
    t->Re = Re; t->Im = Im;
    return t;
  }
float Re(Complex z)
  { return z->Re; }
float Im(Complex z)
  { return z->Im; }
Complex COMPLEXmult(Complex a, Complex b)
  {
    return COMPLEXinit(Re(a)*Re(b) - Im(a)*Im(b),
                       Re(a)*Im(b) + Im(a)*Re(b));
  }

-----
typedef struct queue *Q;
void QUEUEdump(Q);
   Q QUEUEinit(int maxN);
 int QUEUEempty(Q);
void QUEUEput(Q, Item);
Item QUEUEget(Q);
-----
#include <stdio.h>
#include <stdlib.h>
#include "Item.h"
#include "QUEUE.h"
#define M 10
main(int argc, char *argv[])
  { int i, j, N = atoi(argv[1]);
```

```
    Q queues[M];
    for (i = 0; i < M; i++)
      queues[i] = QUEUEinit(N);
    for (i = 0; i < N; i++)
      QUEUEput(queues[rand() % M], j);
    for (i = 0; i < M; i++, printf("\n"))
      for (j = 0; !QUEUEempty(queues[i]); j++)
        printf("%3d ", QUEUEget(queues[i]));
  }
-----
#include <stdlib.h>
#include "Item.h"
#include "QUEUE.h"
typedef struct QUEUEnode* link;
struct QUEUEnode { Item item; link next; };
struct queue { link head; link tail; };
link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
Q QUEUEinit(int maxN)
  { Q q = malloc(sizeof *q);
    q->head = NULL; q->tail = NULL;
    return q;
  }
int QUEUEempty(Q q)
  { return q->head == NULL; }
void QUEUEput(Q q, Item item)
  {
    if (q->head == NULL)
      { q->tail = NEW(item, q->head)
        q->head = q->tail; return; }
    q->tail->next = NEW(item, q->tail->next);
    q->tail = q->tail->next;
  }
Item QUEUEget(Q q)
  { Item item = q->head->item;
    link t = q->head->next;
    free(q->head); q->head = t;
    return item;
  }
-----
#include <stdio.h>
#include <stdlib.h>
#include "POLY.h"
main(int argc, char *argv[])
  { int N = atoi(argv[1]); float p = atof(argv[2]);
    Poly t, x; int i, j;
```

```
    printf("Binomial coefficients\n");
    t = POLYadd(POLYterm(1, 1), POLYterm(1, 0));
    for (i = 0, x = t; i < N; i++)
      { x = POLYmult(t, x); showPOLY(x); }
    printf("%f\n", POLYeval(x, p));
 }
-----
typedef struct poly *Poly;
 void showPOLY(Poly);
 Poly POLYterm(int, int);
 Poly POLYadd(Poly, Poly);
 Poly POLYmult(Poly, Poly);
float POLYeval(Poly, float);
-----
#include <stdlib.h>
#include "POLY.h"
struct poly { int N; int *a; };
Poly POLYterm(int coeff, int exp)
  { int i; Poly t = malloc(sizeof *t);
    t->a = malloc((exp+1)*sizeof(int));
    t->N = exp+1; t->a[exp] = coeff;
    for (i = 0; i < exp; i++) t->a[i] = 0;
    return t;
  }
Poly POLYadd(Poly p, Poly q)
  { int i; Poly t;
    if (p->N < q->N) { t = p; p = q; q = t; }
    for (i = 0; i < q->N; i++) p->a[i] += q->a[i];
    return p;
  }
Poly POLYmult(Poly p, Poly q)
  { int i, j;
    Poly t = POLYterm(0, (p->N-1)+(q->N-1));
    for (i = 0; i < p->N; i++)
      for (j = 0; j < q->N; j++)
        t->a[i+j] += p->a[i]*q->a[j];
    return t;
  }
float POLYeval(Poly p, float x)
  { int i; double t = 0.0;
    for (i = p->N-1; i >= 0; i--)
      t = t*x + p->a[i];
    return t;
  }
```

CAPITOLO 5. Ricorsione e alberi

```
int factorial(int N)
```

```
  {
    if (N == 0) return 1;
    return N*factorial(N-1);
  }
-----
int puzzle(int N)
  {
    if (N == 1) return 1;
    if (N % 2 == 0)
         return puzzle(N/2);
    else return puzzle(3*N+1);
  }
-----
int gcd(int m, int n)
  {
    if (n == 0) return m;
    return gcd(n, m % n);
  }
-----
char *a; int i;
int eval()
  { int x = 0;
    while (a[i] == ' ') i++;
    if (a[i] == '+')
      { i++; return eval() + eval(); }
    if (a[i] == '*')
      { i++; return eval() * eval(); }
    while ((a[i] >= '0') && (a[i] <= '9'))
      x = 10*x + (a[i++]-'0');
    return x;
  }
-----
int count(link x)
  {
    if (x == NULL) return 0;
    return 1 + count(x->next);
  }
void traverse(link h, void (*visit)(link))
  {
    if (h == NULL) return;
    (*visit)(h);
    traverse(h->next, visit);
  }
void traverseR(link h, void (*visit)(link))
  {
    if (h == NULL) return;
    traverseR(h->next, visit);
    (*visit)(h);
  }
```

```
link delete(link x, Item v)
  {
    if (x == NULL) return NULL;
    if (eq(x->item, v))
      { link t = x->next; free(x); return t; }
    x->next = delete(x->next, v);
    return x;
  }
-----
Item max(Item a[], int l, int r)
  { Item u, v; int m = (l+r)/2;
    if (l == r) return a[l];
    u = max(a, l, m);
    v = max(a, m+1, r);
    if (u > v) return u; else return v;
  }
-----
void hanoi(int N, int d)
  {
    if (N == 0) return;
    hanoi(N-1, -d);
    shift(N, d);
    hanoi(N-1, -d);
  }
-----
rule(int l, int r, int h)
  { int m = (l+r)/2;
    if (h > 0)
      {
        rule(l, m, h-1);
        mark(m, h);
        rule(m, r, h-1);
      }
  }
-----
rule(int l, int r, int h)
  {
    int i, j, t;
    for (t = 1, j = 1; t <= h; j += j, t++)
      for (i = 0; l+j+i <= r; i += j+j)
        mark(l+j+i, t);
  }
-----
int F(int i)
  {
    if (i < 1) return 0;
    if (i == 1) return 1;
    return F(i-1) + F(i-2);
  }
```

```
-----
int F(int i)
  { int t;
    if (knownF[i] != unknown) return knownF[i];
    if (i == 0) t = 0;
    if (i == 1) t = 1;
    if (i > 1) t = F(i-1) + F(i-2);
    return knownF[i] = t;
  }
-----
int knap(int cap)
  { int i, space, max, t;
    for (i = 0, max = 0; i < N; i++)
      if ((space = cap-items[i].size) >= 0)
        if ((t = knap(space) + items[i].val) > max)
          max = t;
    return max;
  }
-----
int knap(int M)
  { int i, space, max, maxi, t;
    if (maxKnown[M] != unknown) return maxKnown[M];
    for (i = 0, max = 0; i < N; i++)
      if ((space = M-items[i].size) >= 0)
        if ((t = knap(space) + items[i].val) > max)
          { max = t; maxi = i; }
    maxKnown[M] = max; itemKnown[M] = items[maxi];
    return max;
  }
-----
void traverse(link h, void (*visit)(link))
  {
    if (h == NULL) return;
    (*visit)(h);
    traverse(h->l, visit);
    traverse(h->r, visit);
  }
-----
void traverse(link h, void (*visit)(link))
  {
    STACKinit(max); STACKpush(h);
    while (!STACKempty())
      {
        (*visit)(h = STACKpop());
        if (h->r != NULL) STACKpush(h->r);
        if (h->l != NULL) STACKpush(h->l);
      }
  }
-----
```

```
void traverse(link h, void (*visit)(link))
  {
    QUEUEinit(max); QUEUEput(h);
    while (!QUEUEempty())
      {
        (*visit)(h = QUEUEget());
        if (h->l != NULL) QUEUEput(h->l);
        if (h->r != NULL) QUEUEput(h->r);
      }
  }
-----
int count(link h)
  {
    if (h == NULL) return 0;
    return count(h->l) + count(h->r) + 1;
  }
int height(link h)
  { int u, v;
    if (h == NULL) return -1;
    u = height(h->l); v = height(h->r);
    if (u > v) return u+1; else return v+1;
  }
-----
void printnode(char c, int h)
  { int i;
    for (i = 0; i < h; i++) printf("  ");
    printf("%c\n", c);
  }
void show(link x, int h)
  {
    if (x == NULL) { printnode("*", h); return; }
    show(x->r, h+1);
    printnode(x->item, h);
    show(x->l, h+1);
  }
-----
typedef struct node *link;
struct node { Item item; link l, r };
link NEW(Item item, link l, link r)
  { link x = malloc(sizeof *x);
    x->item = item; x->l = l; x->r = r;
    return x;
  }
link max(Item a[], int l, int r)
  { int m = (l+r)/2; Item u, v;
    link x = NEW(a[m], NULL, NULL);
    if (l == r) return x;
    x->l = max(a, l, m);
    x->r = max(a, m+1, r);
```

```
    u = x->l->item; v = x->r->item;
    if (u > v)
      x->item = u; else x->item = v;
    return x;
  }
-----
char *a; int i;
typedef struct Tnode* link;
struct Tnode { char token; link l, r; };
link NEW(char token, link l, link r)
  { link x = malloc(sizeof *x);
    x->token = token; x->l = l; x->r = r;
    return x;
  }
link parse()
  { char t = a[i++];
    link x = NEW(t, NULL, NULL);
    if ((t == '+') || (t == '*'))
      { x->l = parse(); x->r = parse(); }
    return x;
  }
-----
void traverse(int k, void (*visit)(int))
  { link t;
    (*visit)(k); visited[k] = 1;
    for (t = adj[k]; t != NULL; t = t->next)
      if (!visited[t->v]) traverse(t->v, visit);
  }
-----
void traverse(int k, void (*visit)(int))
  { link t;
    QUEUEinit(V); QUEUEput(k);
    while (!QUEUEempty())
      if (visited[k = QUEUEget()] == 0)
        {
          (*visit)(k); visited[k] = 1;
          for (t = adj[k]; t != NULL; t = t->next)
            if (visited[t->v] == 0) QUEUEput(t->v);
        }
  }
-----
```

CAPITOLO 6. Metodi di ordinamento elementare

```
-----
#include <stdio.h>
#include <stdlib.h>
typedef int Item;
#define key(A) (A)
```

```
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }
#define compexch(A, B) if (less(B, A)) exch(A, B)
void sort(Item a[], int l, int r)
  { int i, j;
    for (i = l+1; i <= r; i++)
      for (j = i; j > l; j--)
        compexch(a[j-1], a[j]);
  }
main(int argc, char *argv[])
  { int i, N = atoi(argv[1]), sw = atoi(argv[2]);
    int *a = malloc(N*sizeof(int));
    if (sw)
      for (i = 0; i < N; i++)
        a[i] = 1000*(1.0*rand()/RAND_MAX);
    else
      while (scanf("%d", &a[N]) == 1) N++;
    sort(a, 0, N-1);
    for (i = 0; i < N; i++) printf("%3d ", a[i]);
    printf("\n");
  }
-----
void selection(Item a[], int l, int r)
  { int i, j;
    for (i = l; i < r; i++)
      { int min = i;
        for (j = i+1; j <= r; j++)
            if (less(a[j], a[min])) min = j;
        exch(a[i], a[min]);
      }
  }
-----
void insertion(Item a[], int l, int r)
  { int i;
    for (i = l+1; i <= r; i++) compexch(a[l], a[i]);
    for (i = l+2; i <= r; i++)
      { int j = i; Item v = a[i];
        while (less(v, a[j-1]))
          { a[j] = a[j-1]; j--; }
        a[j] = v;
      }
  }
-----
void bubble(Item a[], int l, int r)
  { int i, j;
    for (i = l; i < r; i++)
      for (j = r; j > i; j--)
        compexch(a[j-1], a[j]);
  }
```

```
-----
void shellsort(Item a[], int l, int r)
  { int i, j, h;
    for (h = 1; h <= (r-l)/9; h = 3*h+1) ;
    for ( ; h > 0; h /= 3)
      for (i = l+h; i <= r; i++)
        { int j = i; Item v = a[i];
          while (j >= l+h && less(v, a[j-h]))
            { a[j] = a[j-h]; j -= h; }
          a[j] = v;
        }
  }
-----
#include <stdlib.h>
#include "Item.h"
#include "Array.h"
main(int argc, char *argv[])
  { int i, N = atoi(argv[1]), sw = atoi(argv[2]);
    Item *a = malloc(N*sizeof(Item));
    if (sw) randinit(a, N); else scaninit(a, &N);
    sort(a, 0, N-1);
    show(a, 0, N-1);
  }
-----
void randinit(Item [], int);
void scaninit(Item [], int *);
void show(Item [], int, int);
void sort(Item [], int, int);
-----
#include <stdio.h>
#include <stdlib.h>
#include "Item.h"
#include "Array.h"
void randinit(Item a[], int N)
  { int i;
    for (i = 0; i < N; i++) a[i] = ITEMrand();
  }
void scaninit(Item a[], int *N)
  { int i = 0;
    for (i = 0; i < *N; i++)
      if (ITEMscan(&a[i]) == EOF) break;
    *N = i;
  }
void show(itemType a[], int l, int r)
  { int i;
    for (i = l; i <= r; i++) ITEMshow(a[i]);
    printf("\n");
  }
-----
```

```
typedef double Item;
#define key(A) (A)
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { Item t = A; A = B; B = t; }
#define compexch(A, B) if (less(B, A)) exch(A, B)
Item ITEMrand(void);
 int ITEMscan(Item *);
void ITEMshow(Item);
-----
#include <stdio.h>
#include <stdlib.h>
#include "Item.h"
double ITEMrand(void)
         { return 1.0*rand()/RAND_MAX; }
   int ITEMscan(double *x)
         { return scanf("%f", x); }
  void ITEMshow(double x)
         { printf("%7.5f ", x); }
-----
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Item.h"
static char buf[100000];
static int cnt = 0;
int ITEMscan(char **x)
  { int t;
    *x = &buf[cnt];
    t = scanf("%s", *x); cnt += strlen(*x)+1;
    return t;
  }
void ITEMshow(char *x)
  { printf("%s ", x); }
-----
struct record { char name[30]; int num; };
typedef struct record* Item;
#define exch(A, B) { Item t = A; A = B; B = t; }
#define compexch(A, B) if (less(B, A)) exch(A, B);
 int less(Item, Item);
Item ITEMrand();
 int ITEMscan(Item *);
void ITEMshow(Item);

-----
struct record data[maxN];
int Nrecs = 0;
int ITEMscan(struct record **x)
  {
    *x = &data[Nrecs];
```

```
    return scanf("%30s %d\n",
             data[Nrecs].name, &data[Nrecs++].num);
  }
void ITEMshow(struct record *x)
  { printf("%3d %-30s\n", x->num, x->name); }
-----
insitu(dataType data[], int a[], int N)
  { int i, j, k;
    for (i = 0; i < N; i++)
      { dataType v = data[i];
        for (k = i; a[k] != i; k = a[j], a[j] = j)
          { j = k; data[k] = data[a[k]]; }
        data[k] = v; a[k] = k;
      }
  }
-----
typedef struct node *link;
struct node { Item item;  link next; };
link NEW(Item, link);
link init(int);
void show(link);
link sort(link);
-----
link listselection(link h)
  { link max, t, out = NULL;
    while (h->next != NULL)
      {
        max = findmax(h);
        t = max->next; max->next = t->next;
        t->next = out; out = t;
      }
    h->next = out;
    return(h);
  }
-----
void distcount(int a[], int l, int r)
  { int i, j, cnt[M];
    int b[maxN];
    for (j = 0; j <  M; j++) cnt[j] = 0;
    for (i = l; i <= r; i++) cnt[a[i]+1]++;
    for (j = 1; j <  M; j++) cnt[j] += cnt[j-1];
    for (i = l; i <= r; i++) b[cnt[a[i]]++] = a[i];
    for (i = l; i <= r; i++) a[i] = b[i];
  }
-----
void insertion(itemType a[], int l, int r)
  { int i, j;
    for (i = l+1; i <= r; i++)
      for (j = i; j > l; j--)
```

```
        if (less(a[j-1], a[j])) break;
        else exch(a[j-1], a[j]);
  }
```

CAPITOLO 7. Quicksort

```
 int partition(Item a[], int l, int r);
void quicksort(Item a[], int l, int r)
  { int i;
    if (r <= l) return;
    i = partition(a, l, r);
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);
  }
```

```
int partition(Item a[], int l, int r)
  { int i = l-1, j = r; Item v = a[r];
    for (;;)
      {
        while (less(a[++i], v)) ;
        while (less(v, a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a[i], a[j]);
      }
    exch(a[i], a[r]);
    return i;
  }
```

```
#define push2(A, B)  push(B); push(A);
void quicksort(Item a[], int l, int r)
  { int i;
    stackinit(); push2(l, r);
    while (!stackempty())
      {
        l = pop(); r = pop();
        if (r <= l) continue;
        i = partition(a, l, r);
        if (i-l > r-i)
          { push2(l, i-1); push2(i+1, r); }
        else
          { push2(i+1, r); push2(l, i-1); }
      }
  }
```

```
#define M 10
void quicksort(Item a[], int l, int r)
  { int i;
```

```
    if (r-l <= M) return;
    exch(a[(l+r)/2], a[r-1]);
    compexch(a[l], a[r-1]);
      compexch(a[l], a[r]);
        compexch(a[r-1], a[r]);
    i = partition(a, l+1, r-1);
    quicksort(a, l, i-1);
    quicksort(a, i+1, r);
  }
void sort(Item a[], int l, int r)
  {
    quicksort(a, l, r);
    insertion(a, l, r);
  }
-----
#define eq(A, B) (!less(A, B) && !less(B, A))
void quicksort(Item a[], int l, int r)
  { int i, j, k, p, q; Item v;
    if (r <= l) return;
    v = a[r]; i = l-1; j = r; p = l-1; q = r;
    for (;;)
      {
        while (less(a[++i], v)) ;
        while (less(v, a[--j])) if (j == l) break;
        if (i >= j) break;
        exch(a[i], a[j]);
        if (eq(a[i], v)) { p++; exch(a[p], a[i]); }
        if (eq(v, a[j])) { q--; exch(a[q], a[j]); }
      }
    exch(a[i], a[r]); j = i-1; i = i+1;
    for (k = l  ; k < p; k++, j--) exch(a[k], a[j]);
    for (k = r-1; k > q; k--, i++) exch(a[k], a[i]);
    quicksort(a, l, j);
    quicksort(a, i, r);
  }
-----
select(Item a[], int l, int r, int k)
  { int i;
    if (r <= l) return;
    i = partition(a, l, r);
    if (i > k) select(a, l, i-1, k);
    if (i < k) select(a, i+1, r, k);
  }
-----
select(Item a[], int l, int r, int k)
  {
    while (r > l)
      { int i = partition(a, l, r);
        if (i >= k) r = i-1;
```

```
        if (i <= k) l = i+1;
      }
  }
```

CAPITOLO 8. Merging e Mergesort

```
mergeAB(Item c[], Item a[], int N, Item b[], int M )
  { int i, j, k;
    for (i = 0, j = 0, k = 0; k < N+M; k++)
      {
        if (i == N) { c[k] = b[j++]; continue; }
        if (j == M) { c[k] = a[i++]; continue; }
        c[k] = (less(a[i], b[j])) ? a[i++] : b[j++];
      }
  }
```

```
Item aux[maxN];
merge(Item a[], int l, int m, int r)
  { int i, j, k;
    for (i = m+1; i > l; i--) aux[i-1] = a[i-1];
    for (j = m; j < r; j++) aux[r+m-j] = a[j+1];
    for (k = l; k <= r; k++)
       if (less(aux[i], aux[j]))
          a[k] = aux[i++]; else a[k] = aux[j--];
  }
```

```
void mergesort(Item a[], int l, int r)
  { int m = (r+l)/2;
    if (r <= l) return;
    mergesort(a, l, m);
    mergesort(a, m+1, r);
    merge(a, l, m, r);
  }
```

```
#define maxN 10000
Item aux[maxN];
void mergesortABr(Item a[], Item b[], int l, int r)
  { int m = (l+r)/2;
    if (r-l <= 10) { insertion(a, l, r); return; }
    mergesortABr(b, a, l, m);
    mergesortABr(b, a, m+1, r);
    mergeAB(a+l, b+l, m-l+1, b+m+1, r-m);
  }
void mergesortAB(Item a[], int l, int r)
  { int i;
    for (i = l; i <= r; i++) aux[i] = a[i];
    mergesortABr(a, aux, l, r);
  }
```

```
-----
#define min(A, B) (A < B) ? A : B
void mergesortBU(Item a[], int l, int r)
  { int i, m;
    for (m = 1; m < r-l; m = m+m)
      for (i = l; i <= r-m; i += m+m)
        merge(a, i, i+m-1, min(i+m+m-1, r));
  }
-----
link merge(link a, link b)
  { struct node head; link c = &head;
    while ((a != NULL) && (b != NULL))
      if (less(a->item, b->item))
        { c->next = a; c = a; a = a->next; }
      else
        { c->next = b; c = b; b = b->next; }
    c->next = (a == NULL) ? b : a;
    return head.next;
  }
-----
link merge(link a, link b);
link mergesort(link c)
  { link a, b;
    if (c->next == NULL) return c;
    a = c; b = c->next;
    while ((b != NULL) && (b->next != NULL))
      { c = c->next; b = b->next->next; }
    b = c->next; c->next = NULL;
    return merge(mergesort(a), mergesort(b));
  }
-----
link mergesort(link t)
  { link u;
    for (Qinit(); t != NULL; t = u)
      { u = t->next; t->next = NULL; Qput(t); }
    t = Qget();
    while (!Qempty())
      { Qput(t); t = merge(Qget(), Qget()); }
    return t;
  }
```

CAPITOLO 9. Code con priorità e Heapsort

```
-----
void PQinit(int);
 int PQempty();
void PQinsert(Item);
Item PQdelmax();
-----
```

```
#include <stdlib.h>
#include "Item.h"
static Item *pq;
static int N;
void PQinit(int maxN)
  { pq = malloc(maxN*sizeof(Item)); N = 0; }
 int PQempty()
  { return N == 0; }
void PQinsert(Item v)
  { pq[N++] = v; }
Item PQdelmax()
  { int j, max = 0;
    for (j = 1; j < N; j++)
      if (less(pq[max], pq[j])) max = j;
    exch(pq[max], pq[N-1]);
    return pq[--N];
  }
-----
fixUp(Item a[], int k)
  {
    while (k > 1 && less(a[k/2], a[k]))
      { exch(a[k], a[k/2]); k = k/2; }
  }
-----
fixDown(Item a[], int k, int N)
  { int j;
    while (2*k <= N)
      { j = 2*k;
        if (j < N && less(a[j], a[j+1])) j++;
        if (!less(a[k], a[j])) break;
        exch(a[k], a[j]); k = j;
      }
  }
-----
#include <stdlib.h>
#include "Item.h"
static Item *pq;
static int N;
void PQinit(int maxN)
  { pq = malloc((maxN+1)*sizeof(Item)); N = 0; }
 int PQempty()
  { return N == 0; }
void PQinsert(Item v)
  { pq[++N] = v; fixUp(pq, N); }
Item PQdelmax()
  {
    exch(pq[1], pq[N]);
    fixDown(pq, 1, N-1);
    return pq[N--];
```

```
  }
-----
void PQsort(Item a[], int l, int r)
  { int k;
    PQinit();
    for (k = l; k <= r; k++) PQinsert(a[k]);
    for (k = r; k >= l; k--) a[k] = PQdelmax();
  }
-----
#define pq(A) a[l-1+A]
void heapsort(Item a[], int l, int r)
  { int k, N = r-l+1;
    for (k = N/2; k >= 1; k--)
      fixDown(&pq(0), k, N);
    while (N > 1)
      { exch(pq(1), pq(N));
        fixDown(&pq(0), 1, --N); }
  }
-----
typedef struct pq* PQ;
typedef struct PQnode* PQlink;
    PQ PQinit();
   int PQempty(PQ);
PQlink PQinsert(PQ, Item);
  Item PQdelmax(PQ);
  void PQchange(PQ, PQlink, Item);
  void PQdelete(PQ, PQlink);
  void PQjoin(PQ, PQ);
-----
#include <stdlib.h>
#include "Item.h"
#include "PQfull.h"
struct PQnode { Item key; PQlink prev, next; };
struct pq { PQlink head, tail; };
PQ PQinit()
  { PQ pq = malloc(sizeof *pq);
    PQlink h = malloc(sizeof *h),
           t = malloc(sizeof *t);
    h->prev = t; h->next = t;
    t->prev = h; t->next = h;
    pq->head = h; pq->tail = t;
    return pq;
  }
int PQempty(PQ pq)
  { return pq->head->next->next == pq->head; }
PQlink PQinsert(PQ pq, Item v)
  { PQlink t = malloc(sizeof *t);
    t->key = v;
    t->next = pq->head->next; t->next->prev = t;
```

```
    t->prev = pq->head; pq->head->next = t;
  }
Item PQdelmax(PQ pq)
  { Item max; struct PQnode *t, *x = pq->head->next;
    for (t = x; t->next != pq->head; t = t->next)
      if (t->key > x->key) x = t;
    max = x->key;
    x->next->prev = x->prev;
    x->prev->next = x->next;
    free(x); return max;
  }
-----
void PQchange(PQ pq, PQlink x, Item v)
  { x->key = v; }
void PQdelete(PQ pq, PQlink x)
  { PQlink t;
    t->next->prev = t->prev;
    t->prev->next = t->next;
    free(t);
  }
void PQjoin(PQ a, PQ b)
  { PQlink atail, bhead;
    a->tail->prev->next = b->head->next;
    b->head->next->prev = a->tail->prev;
    a->head->prev = b->tail;
    b->tail->next = a->head;
    free(a->tail); free(b->head);
  }
-----
 int less(int, int);
void PQinit();
 int PQempty();
void PQinsert(int);
 int PQdelmax();
void PQchange(int);
void PQdelete(int);
-----
#include "PQindex.h"
typedef int Item;
static int N, pq[maxPQ+1], qp[maxPQ+1];
void exch(int i, int j)
  { int t;
    t = i; i = j; j = t;
    t = qp[i]; qp[i] = qp[j]; qp[j] = t;
  }
void PQinit() { N = 0; }
 int PQempty() { return !N; }
void PQinsert(int k)
  { qp[k] = ++N; pq[N] = k; fixUp(pq, N); }
```

```
 int PQdelmax()
  {
    exch(pq[1], pq[N]);
    fixDown(pq, 1, --N);
    return pq[N+1];
  }
void PQchange(int k)
  { fixUp(pq, qp[k]); fixDown(pq, qp[k], N); }
-----
PQlink pair(PQlink p, PQlink q)
  { PQlink t;
    if (less(p->key, q->key))
         { p->r = q->l; q->l = p; return q; }
    else { q->r = p->l; p->l = q; return p; }
  }
-----
PQlink PQinsert(PQ pq, Item v)
  { int i;
    PQlink c, t = malloc(sizeof *t);
    c = t; c->l = z; c->r = z; c->key = v;
    for (i = 0; i < maxBQsize; i++)
      {
        if (c == z) break;
        if (pq->bq[i] == z) { pq->bq[i] = c; break; }
        c = pair(c, pq->bq[i]); pq->bq[i] = z;
      }
    return t;
  }
-----
Item PQdelmax(PQ pq)
  { int i, j, max; PQlink x; Item v;
    PQlink temp[maxBQsize];
    for (i = 0, max = -1; i < maxBQsize; i++)
      if (pq->bq[i] != z)
        if ((max == -1) || (pq->bq[i]->key > v))
          { max = i; v = pq->bq[max]->key; }
    x = pq->bq[max]->l;
    for (i = max; i < maxBQsize; i++) temp[i] = z;
    for (i = max ; i > 0; i--)
      { temp[i-1] = x; x = x->r; temp[i-1]->r = z; }
    free(pq->bq[max]); pq->bq[max] = z;
    BQjoin(pq->bq, temp);
    return v;
  }
-----
#define test(C, B, A) 4*(C) + 2*(B) + 1*(A)
void BQjoin(PQlink *a, PQlink *b)
  { int i; PQlink c = z;
    for (i = 0; i < maxBQsize; i++)
```

```
      switch(test(c != z, b[i] != z, a[i] != z))
        {
          case 2: a[i] = b[i]; break;
          case 3: c = pair(a[i], b[i]);
                  a[i] = z; break;
          case 4: a[i] = c; c = z; break;
          case 5: c = pair(c, a[i]);
                  a[i] = z; break;
          case 6:
          case 7: c = pair(c, b[i]); break;
        }
  }
void PQjoin(PQ a, PQ b)
  { BQjoin(a->bq, b->bq); }
```

CAPITOLO e10. Ordinamento digitale (on line)

```
quicksortB(int a[], int l, int r, int w)
  { int i = l, j = r;
    if (r <= l || w > bitsword) return;
    while (j != i)
      {
        while (digit(a[i], w) == 0 && (i < j)) i++;
        while (digit(a[j], w) == 1 && (j > i)) j--;
        exch(a[i], a[j]);
      }
    if (digit(a[r], w) == 0) j++;
    quicksortB(a, l, j-1, w+1);
    quicksortB(a, j, r, w+1);
  }
void sort(Item a[], int l, int r)
  {
    quicksortB(a, l, r, 0);
  }
```

```
#define bin(A) l+count[A]
void radixMSD(Item a[], int l, int r, int w)
  { int i, j, count[R+1];
    if (w > bytesword) return;
    if (r-l <= M) { insertion(a, l, r); return; }
    for (j = 0; j < R; j++) count[j] = 0;
    for (i = l; i <= r; i++)
      count[digit(a[i], w) + 1]++;
    for (j = 1; j < R; j++)
      count[j] += count[j-1];
    for (i = l; i <= r; i++)
      aux[l+count[digit(a[i], w)]++] = a[i];
    for (i = l; i <= r; i++) a[i] = aux[i];
```

```
    radixMSD(a, l, bin(0)-1, w+1);
    for (j = 0; j < R-1; j++)
      radixMSD(a, bin(j), bin(j+1)-1, w+1);
  }
-----
#define ch(A) digit(A, D)
void quicksortX(Item a[], int l, int r, int D)
  {
    int i, j, k, p, q; int v;
    if (r-l <= M) { insertion(a, l, r); return; }
    v = ch(a[r]); i = l-1; j = r; p = l-1; q = r;
    while (i < j)
      {
        while (ch(a[++i]) < v) ;
        while (v < ch(a[--j])) if (j == l) break;
        if (i > j) break;
        exch(a[i], a[j]);
        if (ch(a[i])==v) { p++; exch(a[p], a[i]); }
        if (v==ch(a[j])) { q--; exch(a[j], a[q]); }
      }
    if (p == q)
      { if (v != '\0') quicksortX(a, l, r, D+1);
        return; }
    if (ch(a[i]) < v) i++;
    for (k = l; k <= p; k++, j--) exch(a[k], a[j]);
    for (k = r; k >= q; k--, i++) exch(a[k], a[i]);
    quicksortX(a, l, j, D);
    if ((i == r) && (ch(a[i]) == v)) i++;
    if (v != '\0') quicksortX(a, j+1, i-1, D+1);
    quicksortX(a, i, r, D);
  }
-----
void radixLSD(Item a[], int l, int r)
  {
    int i, j, w, count[R+1];
    for (w = bytesword-1; w >= 0; w--)
      {
        for (j = 0; j < R; j++) count[j] = 0;
        for (i = l; i <= r; i++)
          count[digit(a[i], w) + 1]++;
        for (j = 1; j < R; j++)
          count[j] += count[j-1];
        for (i = l; i <= r; i++)
          aux[count[digit(a[i], w)]++] = a[i];
        for (i = l; i <= r; i++) a[i] = aux[i];
      }
  }

----------
```

CAPITOLO e11. Metodi speciali di ordinamento (on line)

```
shuffle(itemType a[], int l, int r)
  { int i, j, m = (l+r)/2;
    for (i = l, j = 0; i <= r; i+=2, j++)
      { aux[i] = a[l+j]; aux[i+1] = a[m+1+j]; }
    for (i = l; i <= r; i++) a[i] = aux[i];
  }
unshuffle(itemType a[], int l, int r)
  { int i, j, m = (l+r)/2;
    for (i = l, j = 0; i <= r; i+=2, j++)
      { aux[l+j] = a[i]; aux[m+1+j] = a[i+1]; }
    for (i = l; i <= r; i++) a[i] = aux[i];
  }
```

```
mergeTD(itemType a[], int l, int r)
  { int i, m = (l+r)/2;
    if (r == l+1) compexch(a[l], a[r]);
    if (r < l+2) return;
    unshuffle(a, l, r);
    mergeTD(a, l, m);
    mergeTD(a, m+1, r);
    shuffle(a, l, r);
    for (i = l+1; i < r; i+=2)
      compexch(a[i], a[i+1]);
  }
```

```
mergeBU(itemType a[], int l, int r)
  { int i, j, k, N = r-l+1;
    for (k = N/2; k > 0; k /= 2)
      for (j = k % (N/2); j+k < N; j += (k+k))
        for (i = 0; i < k; i++)
          compexch(a[l+j+i], a[l+j+i+k]);
  }
```

```
void batchersort(itemType a[], int l, int r)
  { int i, j, k, p, N = r-l+1;
    for (p = 1; p < N; p += p)
      for (k = p; k > 0; k /= 2)
        for (j = k%p; j+k < N; j += (k+k))
          for (i = 0; i < k; i++)
            if (j+i+k < N)
              if ((j+i)/(p+p) == (j+i+k)/(p+p))
                compexch(a[l+j+i], a[l+j+i+k]);
  }
```

```
id = 1;
for (i = 1; i <= S; i++) a[i] = strGet(0);
while (a[1] < z)
```

```
  {
    strPut(id, (c = a[1]));
    if ((a[1] = strGet(0)) < c) mark(a[1]);
    fixDown(1, S);
    if (marked(a[1]))
      {
        for (i = 1; i <= S; i++) unmark(a[i]);
        strPut(id, z);
        id = id % S + 1;
      }
  }
strPut(id, z);
-----
void compexch(int x, int y)
  { int t;
    t = stage[a[x]]; if (t < stage[a[y]]) t = stage[a[y]];
t++;
    stage[a[x]] = t; stage[a[y]] = t;
    printf("%3d %3d %3d\n", t, a[x], a[y]);
  }

----------
CAPITOLO 12. Symbol Tables and BSTs
-----
void STinit(int);
 int STcount();
void STinsert(Item);
Item STsearch(Key);
void STdelete(Item);
Item STselect(int);
void STsort(void (*visit)(Item));
-----
#include <stdio.h>
#include <stdlib.h>
#include "Item.h"
#include "ST.h"
void main(int argc, char *argv[])
 { int N, maxN = atoi(argv[1]), sw = atoi(argv[2]);
    Key v; Item item;
    STinit(maxN);
    for (N = 0; N < maxN; N++)
      {
        if (sw) v = ITEMrand();
          else if (ITEMscan(&v) == EOF) break;
        if (STsearch(v) != NULLitem) continue;
        key(item) = v;
        STinsert(item);
      }
    STsort(ITEMshow); printf("\n");
```

```
    printf("%d keys ", N);
    printf("%d distinct keys\n", STcount());
 }
-----
static Item *st;
static int M = maxKey;
void STinit(int maxN)
  { int i;
    st = malloc((M+1)*sizeof(Item));
    for (i = 0; i <= M; i++) st[i] = NULLitem;
  }
int STcount()
  { int i, N = 0;
    for (i = 0; i < M; i++)
      if (st[i] != NULLitem) N++;
    return N;
  }
void STinsert(Item item)
  { st[key(item)] = item; }
Item STsearch(Key v)
  { return st[v]; }
void STdelete(Item item)
  { st[key(item)] = NULLitem; }
Item STselect(int k)
  { int i;
    for (i = 0; i < M; i++)
      if (st[i] != NULLitem)
        if (k-- == 0) return st[i];
  }
void STsort(void (*visit)(Item))
  { int i;
    for (i = 0; i < M; i++)
      if (st[i] != NULLitem) visit(st[i]);
  }
-----
static Item *st;
static int N;
void STinit(int maxN)
  { st = malloc((maxN)*sizeof(Item)); N = 0; }
int STcount()
  { return N; }
void STinsert(Item item)
  { int j = N++; Key v = key(item);
    while (j>0 && less(v, key(st[j-1])))
      {  st[j] = st[j-1]; j--; }
    st[j] = item;
  }
Item STsearch(Key v)
  { int j;
```

```
    for (j = 0; j < N; j++)
      {
        if (eq(v, key(st[j]))) return st[j];
        if (less(v, key(st[j]))) break;
      }
    return NULLitem;
  }
Item STselect(int k)
  { return st[k]; }
void STsort(void (*visit)(Item))
  { int i;
    for (i = 0; i < N; i++) visit(st[i]);
  }
-----
typedef struct STnode* link;
struct STnode { Item item; link next; };
static link head, z;
static int N;
static link NEW(Item item, link next)
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
void STinit(int max)
  { N = 0; head = (z = NEW(NULLitem, NULL)); }
int STcount() { return N; }
Item searchR(link t, Key v)
  {
    if (t == z) return NULLitem;
    if (eq(key(t->item), v)) return t->item;
    return searchR(t->next, v);
  }
Item STsearch(Key v)
  { return searchR(head, v); }
void STinsert(Item item)
  { head = NEW(item, head); N++; }
-----
Item search(int l, int r, Key v)
  { int m = (l+r)/2;
    if (l > r) return NULLitem;
    if eq(v, key(st[m])) return st[m];
    if (l == r) return NULLitem;
    if less(v, key(st[m]))
         return search(l, m-1, v);
    else return search(m+1, r, v);
  }
Item STsearch(Key v)
  { return search(0, N-1, v); }
-----
```

```
#include <stdlib.h>
#include "Item.h"
typedef struct STnode* link;
struct STnode { Item item; link l, r; int N };
static link head, z;
link NEW(Item item, link l, link r, int N)
  { link x = malloc(sizeof *x);
    x->item = item; x->l = l; x->r = r; x->N = N;
    return x;
  }
void STinit()
  { head = (z = NEW(NULLitem, 0, 0, 0)); }
int STcount() { return head->N; }
Item searchR(link h, Key v)
  { Key t = key(h->item);
    if (h == z) return NULLitem;
    if eq(v, t) return h->item;
    if less(v, t) return searchR(h->l, v);
             else return searchR(h->r, v);
  }
Item STsearch(Key v)
  { return searchR(head, v); }
link insertR(link h, Item item)
  { Key v = key(item), t = key(h->item);
    if (h == z) return NEW(item, z, z, 1);
    if less(v, t)
         h->l = insertR(h->l, item);
    else h->r = insertR(h->r, item);
    (h->N)++; return h;
  }
void STinsert(Item item)
  { head = insertR(head, item); }
-----
void sortR(link h, void (*visit)(Item))
  {
    if (h == z) return;
    sortR(h->l, visit);
    visit(h->item);
    sortR(h->r, visit);
  }
void STsort(void (*visit)(Item))
  { sortR(head, visit); }
-----
void STinsert(Item item)
  { Key v = key(item); link p = head, x = p;
    if (head == NULL)
      { head = NEW(item, NULL, NULL, 1); return; }
    while (x != NULL)
      {
```

```
        p = x; x->N++;
        x = less(v, key(x->item)) ? x->l : x->r;
      }
    x = NEW(item, NULL, NULL, 1);
    if (less(v, key(p->item))) p->l = x;
                          else p->r = x;
  }
-----
#define null(A) (eq(key(A), key(NULLitem)))
static char text[maxN];
main(int argc, char *argv[])
  { int i, t, N = 0; char query[maxQ]; char *v;
    FILE *corpus  = fopen(*++argv, "r");
    while ((t = getc(corpus)) != EOF)
      if (N < maxN) text[N++] = t; else break;
    text[N] = '\0';
    STinit(maxN);
    for (i = 0; i < N; i++) STinsert(&text[i]);
    while (gets(query) != NULL)
      if (!null(v = STsearch(query)))
           printf("%11d %s\n", v-text, query);
      else printf("(not found) %s\n", query);
  }
-----
link rotR(link h)
  { link x = h->l; h->l = x->r; x->r = h;
    return x; }
link rotL(link h)
  { link x = h->r; h->r = x->l; x->l = h;
    return x; }
-----
link insertT(link h, Item item)
  { Key v = key(item);
    if (h == z) return NEW(item, z, z, 1);
    if (less(v, key(h->item)))
      { h->l = insertT(h->l, item); h = rotR(h); }
    else
      { h->r = insertT(h->r, item); h = rotL(h); }
    return h;
  }
void STinsert(Item item)
  { head = insertT(head, item); }
-----
Item selectR(link h, int k)
  { int t = h->l->N;
    if (h == z) return NULLitem;
    if (t > k) return selectR(h->l, k);
    if (t < k) return selectR(h->r, k-t-1);
    return h->item;
```

```
  }
Item STselect(int k)
  { return selectR(head, k); }
-----
link partR(link h, int k)
  { int t = h->l->N;
    if (t > k )
      { h->l = partR(h->l, k); h = rotR(h); }
    if (t < k )
      { h->r = partR(h->r, k-t-1); h = rotL(h); }
    return h;
  }
-----
link joinLR(link a, link b)
  {
    if (b == z) return a;
    b = partR(b, 0); b->l = a;
    return b;
  }
link deleteR(link h, Key v)
  { link x; Key t = key(h->item);
    if (h == z) return z;
    if (less(v, t)) h->l = deleteR(h->l, v);
    if (less(t, v)) h->r = deleteR(h->r, v);
    if (eq(v, t))
      { x = h; h = joinLR(h->l, h->r); free(x); }
    return h;
  }
void STdelete(Key v)
  { head = deleteR(head, v); }
-----
link STjoin(link a, link b)
  {
    if (b == z) return a;
    if (a == z) return b;
    b = STinsert(b, a->item);
    b->l = STjoin(a->l, b->l);
    b->r = STjoin(a->r, b->r);
    free(a);
    return b;
  }
-----
```

CAPITOLO e13. Alberi bilanciati (on line)

```
-----
link balanceR(link h)
  {
    if (h->N < 2) return h;
```

```
    h = partR(h, h->N/2);
    h->l = balanceR(h->l);
    h->r = balanceR(h->r);
    return h;
  }
-----
link insertR(link h, Item item)
  { Key v = key(item), t = key(h->item);
    if (h == z) return NEW(item, z, z, 1);
    if (rand()< RAND_MAX/(h->N+1))
      return insertT(h, item);
    if less(v, t) h->l = insertR(h->l, item);
             else h->r = insertR(h->r, item);
    (h->N)++; return h;
  }
void STinsert(Item item)
  { head = insertR(head, item); }
-----
link STjoinR(link a, link b)
  {
    if (a == z) return b;
    b = STinsert(b, a->rec);
    b->l = STjoin(a->l, b->l);
    b->r = STjoin(a->r, b->r);
    fixN(b); free(a);
    return b;
  }
link STjoin(link a, link b)
  {
    if (rand()/(RAND_MAX/(a->N+b->N)+1) < a->N)
         STjoinR(a, b);
    else STjoinR(b, a);
  }
-----
link joinLR(link a, link b)
  {
    if (a == z) return b;
    if (b == z) return a;
    if (rand()/(RAND_MAX/(a->N+b->N)+1) < a->N)
         { a->r = joinLR(a->r, b); return a; }
    else { b->l = joinLR(a, b->l); return b; }
  }
-----
link splay(link h, Item item)
  { Key v = key(item);
    if (h == z) return NEW(item, z, z, 1);
    if (less(v, key(h->item)))
      {
        if (hl == z) return NEW(item, z, h, h->N+1);
```

```
        if (less(v, key(hl->item)))
          { hll = splay(hll, item); h = rotR(h); }
        else
          { hlr = splay(hlr, item); hl = rotL(hl); }
        return rotR(h);
      }
    else
      {
        if (hr == z) return NEW(item, h, z, h->N+1);
        if (less(key(hr->item), v))
          { hrr = splay(hrr, item); h = rotL(h); }
        else
          { hrl = splay(hrl, item); hr = rotR(hr); }
        return rotL(h);
      }
  }
void STinsert(Item item)
  { head = splay(head, item); }
-----
link RBinsert(link h, Item item, int sw)
  { Key v = key(item);
    if (h == z) return NEW(item, z, z, 1, 1);
    if ((hl->red) && (hr->red))
      { h->red = 1; hl->red = 0; hr->red = 0; }
    if (less(v, key(h->item)))
      {
        hl = RBinsert(hl, item, 0);
        if (h->red && hl->red && sw) h = rotR(h);
        if (hl->red && hll->red)
          { h = rotR(h); h->red = 0; hr->red = 1; }
      }
    else
      {
        hr = RBinsert(hr, item, 1);
        if (h->red && hr->red && !sw) h = rotL(h);
        if (hr->red && hrr->red)
          { h = rotL(h); h->red = 0; hl->red = 1; }
      }
    fixN(h); return h;
  }
void STinsert(Item item)
  { head = RBinsert(head, item, 0); head->red = 0; }
-----
Item searchR(link t, Key v, int k)
  {
    if (eq(v, key(t->item))) return t->item;
    if (less(v, key(t->next[k]->item)))
      {
        if (k == 0) return NULLitem;
```

```
        return searchR(t, v, k-1);
      }
    return searchR(t->next[k], v, k);
  }
Item STsearch(Key v)
  { return searchR(head, v, lgN); }
-----
typedef struct STnode* link;
struct STnode { Item item; link* next; int sz; };
static link head, z;
static int N, lgN;
link NEW(Item item, int k)
  { int i; link x = malloc(sizeof *x);
    x->next = malloc(k*sizeof(link));
    x->item = item; x->sz = k;
    for (i = 0; i < k; i++) x->next[i] = z;
    return x;
  }
void STinit(int max)
  {
    N = 0; lgN = 0;
    z = NEW(NULLitem, 0);
    head = NEW(NULLitem, lgNmax);
  }
-----
int randX()
  { int i, j, t = rand();
    for (i = 1, j = 2; i < lgNmax; i++, j += j)
      if (t > RAND_MAX/j) break;
    if (i > lgN) lgN = i;
    return i;
  }
void insertR(link t, link x, int k)
  { Key v = key(x->item);
    if (less(v, key(t->next[k]->item)))
      {
        if (k < x->sz)
          { x->next[k] = t->next[k];
            t->next[k] = x; }
        if (k == 0) return;
        insertR(t, x, k-1); return;
      }
    insertR(t->next[k], x, k);
  }
void STinsert(Key v)
  { insertR(head, NEW(v, randX()), lgN); N++; }
-----
void deleteR(link t, Key v, int k)
  { link x = t->next[k];
```

```
    if (!less(key(x->item), v))
      {
        if (eq(v, key(x->item)))
          { t->next[k] = x->next[k]; }
        if (k == 0) { free(x); return; }
        deleteR(t, v, k-1); return;
      }
    deleteR(t->next[k], v, k);
  }
void STdelete(Key v)
  { deleteR(head, v, lgN); N--; }
```

CAPITOLO 14. Hashing

```
int hash(char *v, int M)
  { int h = 0, a = 127;
    for (; *v != '\0'; v++)
      h = (a*h + *v) % M;
    return h;
  }
```

```
int hashU(char *v, int M)
  { int h, a = 31415, b = 27183;
    for (h = 0; *v != '\0'; v++, a = a*b % (M-1))
        h = (a*h + *v) % M;
    return h;
  }
```

```
static link *heads, z;
static int N, M;
void STinit(int max)
  { int i;
    N = 0; M = max/5;
    heads = malloc(M*sizeof(link));
    z = NEW(NULLitem, NULL);
    for (i = 0; i < M; i++) heads[i] = z;
  }
Item STsearch(Key v)
  { return searchR(heads[hash(v, M)], v); }
void STinsert(Item item)
  { int i = hash(key(item), M);
    heads[i] = NEW(item, heads[i]); N++; }
void STdelete(Item item)
  { int i = hash(key(item), M);
    heads[i] = deleteR(heads[i], item); }
```

```
#include <stdlib.h>
#include "Item.h"
```

```
#define null(A) (key(st[A]) == key(NULLitem))
static int N, M;
static Item *st;
void STinit(int max)
  { int i;
    N = 0; M = 2*max;
    st = malloc(M*sizeof(Item));
    for (i = 0; i < M; i++) st[i] = NULLitem;
  }
int STcount() { return N; }
void STinsert(Item item)
  { Key v = key(item);
    int i = hash(v, M);
    while (!null(i)) i = (i+1) % M;
    st[i] = item; N++;
  }
Item STsearch(Key v)
  { int i = hash(v, M);
    while (!null(i))
      if eq(v, key(st[i])) return st[i];
      else i = (i+1) % M;
    return NULLitem;
  }
-----
void STdelete(Item item)
  { int j, i = hash(key(item), M); Item v;
    while (!null(i))
      if eq(key(item), key(st[i])) break;
      else i = (i+1) % M;
    if (null(i)) return;
    st[i] = NULLitem; N--;
    for (j = i+1; !null(j); j = (j+1) % M, N--)
      { v = st[j]; st[j] = NULLitem; STinsert(v); }
  }
-----
void STinsert(Item item)
  { Key v = key(item);
    int i = hash(v, M);
    int k = hashtwo(v, M);
    while (!null(i)) i = (i+k) % M;
    st[i] = item; N++;
  }
Item STsearch(Key v)
  { int i = hash(v, M);
    int k = hashtwo(v, M);
    while (!null(i))
      if eq(v, key(st[i])) return st[i];
      else i = (i+k) % M;
    return NULLitem;
```

```
  }
-----
void expand();
void STinsert(Item item)
  { Key v = key(item);
    int i = hash(v, M);
    while (!null(i)) i = (i+1) % M;
    st[i] = item;
    if (N++ > M/2) expand();
  }
void expand()
  { int i; Item *t = st;
    init(M+M);
    for (i = 0; i < M/2; i++)
      if (key(t[i]) != key(NULLitem))
        STinsert(t[i]);
    free(t);
  }

----------
```
CAPITOLO e15. Ricerca digitale (on line)
```
-----
Item searchR(link h, Key v, int w)
  { Key t = key(h->item);
    if (h == z) return NULLitem;
    if eq(v, t) return h->item;
    if (digit(v, w) == 0)
         return searchR(h->l, v, w+1);
    else return searchR(h->r, v, w+1);
  }
Item STsearch(Key v)
  { return searchR(head, v, 0); }
-----
#define leaf(A) ((h->l == z) && (h->r == z))
Item searchR(link h, Key v, int w)
  { Key t = key(h->item);
    if (h == z) return NULLitem;
    if (leaf(h))
      return eq(v, t) ? h->item : NULLitem;
    if (digit(v, w) == 0)
         return searchR(h->l, v, w+1);
    else return searchR(h->r, v, w+1);
  }
Item STsearch(Key v)
  { return searchR(head, v, 0); }
-----
void STinit()
  { head = (z = NEW(NULLitem, 0, 0, 0)); }
link split(link p, link q, int w)
```

```
  { link t = NEW(NULLitem, z, z, 2);
    switch(digit(p->item, w)*2 + digit(q->item, w))
      {
        case 0: t->l = split(p, q, w+1); break;
        case 1: t->l = p; t->r = q; break;
        case 2: t->r = p; t->l = q; break;
        case 3: t->r = split(p, q, w+1); break;
      }
    return t;
  }
link insertR(link h, Item item, int w)
  { Key v = key(item), t = key(h->item);
    if (h == z) return NEW(item, z, z, 1);
    if (leaf(h))
      { return split(NEW(item, z, z, 1), h, w); }
    if (digit(v, w) == 0)
         h->l = insertR(h->l, item, w+1);
    else h->r = insertR(h->r, item, w+1);
    return h;
  }
void STinsert(Item item)
  { head = insertR(head, item, 0); }
-----
Item searchR(link h, Key v, int w)
  {
    if (h->bit <= w) return h->item;
    if (digit(v, h->bit) == 0)
         return searchR(h->l, v, h->bit);
    else return searchR(h->r, v, h->bit);
  }
Item STsearch(Key v)
  { Item t = searchR(head->l, v, -1);
    return eq(v, key(t)) ? t : NULLitem;
  }
-----
void STinit()
  { head = NEW(NULLitem, 0, 0, -1);
    head->l = head; head->r = head; }
link insertR(link h, Item item, int w, link p)
  { link x; Key v = key(item);
    if ((h->bit >= w) || (h->bit <= p->bit))
      {
        x = NEW(item, 0, 0, w);
        x->l = digit(v, x->bit) ? h : x;
        x->r = digit(v, x->bit) ? x : h;
        return x;
      }
    if (digit(v, h->bit) == 0)
         h->l = insertR(h->l, item, w, h);
```

```
    else h->r = insertR(h->r, item, w, h);
    return h;
  }
void STinsert(Item item)
  { int i;
    Key v = key(item);
    Key t = key(searchR(head->l, v, -1));
    if (v == t) return;
    for (i = 0; digit(v, i) == digit(t, i); i++) ;
    head->l = insertR(head->l, item, i, head);
  }
-----
void sortR(link h, void (*visit)(Item), int w)
  {
    if (h->bit <= w) { visit(h->item); return; }
    sortR(h->l, visit, h->bit);
    sortR(h->r, visit, h->bit);
  }
void STsort(void (*visit)(Item))
  { sortR(head->l, visit, -1); }
-----
typedef struct STnode *link;
struct STnode { link next[R]; };
static link head;
void STinit() { head = NULL; }
link NEW()
  { int i;
    link x = malloc(sizeof *x);
    for (i = 0; i < R; i++) x->next[i] = NULL;
    return x;
  }
Item searchR(link h, Key v, int w)
  { int i = digit(v, w);
    if (h == NULL) return NULLitem;
    if (i == NULLdigit) return v;
    return searchR(h->next[i], v, w+1);
  }
Item STsearch(Key v)
  { return searchR(head, v, 0); }
link insertR(link h, Item item, int w)
  { Key v = key(item);
    int i = digit(v, w);
    if (h == NULL) h = NEW();
    if (i == NULLdigit) return h;
    h->next[i] = insertR(h->next[i], v, w+1);
    return h;
  }
void STinsert(Item item)
  { head = insertR(head, item, 0); N++; }
```

```
-----
typedef struct STnode* link;
struct STnode { Item item; int d; link l, m, r; };
static link head;
void STinit() { head = NULL; }
link NEW(int d)
  { link x = malloc(sizeof *x);
    x->d = d; x->l = NULL; x->m = NULL; x->r = NULL;
    return x;
  }
Item searchR(link h, Key v, int w)
  { int i = digit(v, w);
    if (h == NULL) return NULLitem;
    if (i == NULLdigit) return v;
    if (i < h->d) return searchR(h->l, v, w);
    if (i == h->d) return searchR(h->m, v, w+1);
    if (i > h->d) return searchR(h->r, v, w);
  }
Item STsearch( Key v)
  { return searchR(head, v, 0); }
link insertR(link h, Item item, int w)
  { Key v = key(item);
    int i = digit(v, w);
    if (h == NULL) h = NEW(i);
    if (i == NULLdigit) return h;
    if (i < h->d) h->l = insertR(h->l, v, w);
    if (i == h->d) h->m = insertR(h->m, v, w+1);
    if (i > h->d) h->r = insertR(h->r, v, w);
    return h;
  }
void STinsert(Key key)
  { head = insertR(head, key, 0); }
-----
char word[maxW];
void matchR(link h, char *v, int i)
  {
    if (h == z) return;
    if ((*v == '\0') && (h->d == '\0'))
      { word[i] = h->d; printf("%s ", word); }
    if ((*v == '*') || (*v == h->d))
      { word[i] = h->d; matchR(h->m, v+1, i+1); }
    if ((*v == '*') || (*v < h->d))
      matchR(h->l, v, i);
    if ((*v == '*') || (*v > h->d))
      matchR(h->r, v, i);
  }
void STmatch(char *v)
  { matchR(head, v, 0); }
-----
```

```
#define internal(A) ((A->d) != NULLdigit)
link NEWx(link h, int d)
  { link x = malloc(sizeof *x);
    x->item = NULLitem; x->d = d;
    x->l = NULL; x->m = h; x->r = NULL;
    return x;
  }
link split(link p, link q, int w)
  { int pd = digit(p->item, w),
        qd = digit(q->item, w);
    link t = NEW(NULLitem, qd);
    if (pd < qd) { t->m = q; t->l = NEWx(p, pd); }
    if (pd == qd) { t->m = split(p, q, w+1); }
    if (pd > qd) { t->m = q; t->r = NEWx(p, pd); }
    return t;
  }
link insertR(link h, Item item, int w)
  { Key v = key(item);
    int i = digit(v, w);
    if (h == NULL)
      return NEWx(NEW(item, NULLdigit), i);
    if (!internal(h))
      return split(NEW(item, NULLdigit), h, w);
    if (i < h->d) h->l = insertR(h->l, v, w);
    if (i == h->d) h->m = insertR(h->m, v, w+1);
    if (i > h->d) h->r = insertR(h->r, v, w);
    return h;
  }
void STinsert(Key key)
  { int i = digit(key, 0);
    heads[i] = insertR(heads[i], key, 1);
  }
-----
Item searchR(link h, Key v, int w)
  { int i = digit(v, w);
    if (h == NULL) return NULLitem;
    if (internal(h))
      {
        if (i < h->d) return searchR(h->l, v, w);
        if (i == h->d) return searchR(h->m, v, w+1);
        if (i > h->d) return searchR(h->r, v, w);
      }
    if eq(v, key(h->item)) return h->item;
    return NULLitem;
  }
Item STsearch(Key v)
  { return searchR(heads[digit(v, 0)], v, 1); }
-----
typedef struct STnode* link;
```

```
struct STnode { Item item; link l, m, r; };
static link head;
#define z NULL
void STinit() { head = z; }
link NEW(char *v)
  { link x = malloc(sizeof *x);
    x->item = v; x->l = z; x->m = z; x->r = z;
    return x;
  }
Item searchR(link h, char *v)
  { char *t;
    if (h == z) return NULLitem;
    if (*v == '\0') return h->item;
    if (*v < *(h->item)) searchR(h->l, v);
    if (*v > *(h->item)) searchR(h->r, v);
    if (*v == *(h->item)) t = searchR(h->m, v+1);
    return null(t) ? t : v;
  }
Item STsearch(char *v)
  { char *t = searchR(head, v);
    if (eq(v, t)) return t;
    return NULLitem;
  }
link insertR(link h, char *v)
  {
    if (h == z) h = NEW(v);
    if ((*v ==  *(h->item)) && (*v != '\0'))
      h->m = insertR(h->m, v+1);
    if (h == z) h = NEW(v);
    if (*v < *(h->item)) h->l = insertR(h->l, v);
    if (*v > *(h->item)) h->r = insertR(h->r, v);
    return h;
  }
void STinsert(char *v)
  { head = insertR(head, v); }
```

CAPITOLO e 16. Ricerca esterna (on line)

```
typedef struct STnode* link;
typedef struct
  { Key key; union { link next; Item item; } ref; }
entry;
struct STnode { entry b[M]; int m; };
static link head;
static int H, N;
link NEW()
  { link x = malloc(sizeof *x);
    x->m = 0;
```

```
    return x;
  }
void STinit(int maxN)
  { head = NEW(); H = 0; N = 0; }
-----
Item searchR(link h, Key v, int H)
  { int j;
    if (H == 0)
      for (j = 0; j < h->m; j++)
        if (eq(v, h->b[j].key))
          return h->b[j].ref.item;
    if (H != 0)
      for (j = 0; j < h->m; j++)
        if ((j+1 == h->m) || less(v, h->b[j+1].key))
          return searchR(h->b[j].ref.next, v, H-1);
    return NULLitem;
  }
Item STsearch(Key v)
  { return searchR(head, v, H); }
-----
link insertR(link h, Item item, int H)
  { int i, j; Key v = key(item); entry x; link u;
    x.key = v; x.ref.item = item;
    if (H == 0)
      for (j = 0; j < h->m; j++)
        if (less(v, h->b[j].key)) break;
    if (H != 0)
      for (j = 0; j < h->m; j++)
        if ((j+1 == h->m) || less(v, h->b[j+1].key))
          {
            u = insertR(h->b[j++].ref.next, v, H-1);
            if (u == NULL) return NULL;
            x.key = u->b[0].key; x.ref.next = u;
            break;
          }
    for (i = ++(h->m); (i > j) && (i != M); i--)
      h->b[i] = h->b[i-1];
    h->b[j] = x;
    if (h->m < M) return NULL; else return split(h);
  }
void STinsert(Item item)
  { link t, u = insertR(head, item, H);
    if (u == NULL) return;
    t = NEW(); t->m = 2;
    t->b[0].key = head->b[0].key;
    t->b[0].ref.next = head;
    t->b[1].key =  u->b[0].key;
    t->b[1].ref.next = u;
    head = t; H++;
```

```
  }
-----
link split(link h)
  { int j; link t = NEW();
    for (j = 0; j < M/2; j++)
      t->b[j] = h->b[M/2+j];
    h->m = M/2; t->m = M/2;
    return t;
  }
-----
typedef struct STnode* link;
struct STnode { Item b[M]; int m; int k; };
static link *dir;
static int d, D, N;
link NEW()
  { link x = malloc(sizeof *x);
    x->m = 0;  x->k = 0;
    return x;
  }
void STinit(int maxN)
  {
    d = 0; N = 0; D = 1;
    dir = malloc(D*(sizeof *dir));
    dir[0] = NEW();
  }
-----
Item search(link h, Key v)
  { int j;
    for (j = 0; j < h->m; j++)
      if (eq(v, key(h->b[j])))
        return h->b[j];
    return NULLitem;
  }
Item STsearch(Key v)
  { return search(dir[bits(v, 0, d)], v); }
-----
link split(link h)
  { int j; link t = NEW();
    while (h->m == M)
      {
        h->m = 0; t->m = 0;
        for (j = 0; j < M; j++)
          if (bits(h->b[j], h->k, 1) == 0)
               h->b[(h->m)++] = h->b[j];
          else t->b[(t->m)++] = h->b[j];
        t->k = ++(h->k);
        if (t->m == M) h = t;
      }
    insertDIR(t, t->k);
```

```
  }
void insert(link h, Item item)
  { int i, j; Key v = key(item);
    for (j = 0; j < h->m; j++)
      if (less(v, key(h->b[j]))) break;
    for (i = (h->m)++; i > j; i--)
      h->b[i] = h->b[i-1];
    h->b[j] = item;
    if (h->m == M) split(h);
  }
void STinsert(Item item)
  { insert(dir[bits(key(item), 0, d)], item); }
-----
void insertDIR(link t, int k)
  { int i, m, x = bits(t->b[0], 0, k);
    while (d < k)
      { link *old = dir;
        d += 1; D += D;
        dir = malloc(D*(sizeof *dir));
        for (i = 0; i < D; i++) dir[i] = old[i/2];
        if (d < k) dir(bits(x, 0, d) ^ 1) = NEW();
      }
    for (m = 1; k < d; k++) m *= 2;
    for (i = 0; i < m; i++) dir[x*m+i] = t;
  }
```