

# BELLMAN FORD

```
#include <cstdio>
#include <climits>
#include <vector>
using namespace std;

struct node {
    long long int dist;
    int pred;
    bool flag;
    int id;
};

struct edge {
    int x;
    int y;
    int w;
};

#define MAX_M 9000000

vector<node> nodes;
edge edges[MAX_M];
int e, v;

bool BellmanFord(int start) {
    for (int i = 0; i < v; i++) {
        if (i == start)
            nodes[i].dist = 0;
        else
            nodes[i].dist = INT_MAX;
        nodes[i].pred = -1;
        nodes[i].id = i;
    }

    for (int i = 0; i < v-1; i++)
    {
        for (int j = 0; j < e; j++) {
            if (nodes[edges[j].y].dist > nodes[edges[j].x].dist + edges[j].w) {
                nodes[edges[j].y].dist = nodes[edges[j].x].dist + edges[j].w;
                nodes[edges[j].y].pred = edges[j].x;
            }
        }
    }

    for (int j = 0; j < e; j++)
        if (nodes[edges[j].y].dist > nodes[edges[j].x].dist + edges[j].w)
            return false;

    return true;
}

int main() {
    int x, y, w;
    scanf("%d %d", &v, &e);
    nodes.resize(v);
    for (int i = 0; i < e; i++) {
        scanf("%d %d %d", &x, &y, &w);
```

```

    x--;
    y--;
    edges[i].x = x;
    edges[i].y = y;
    edges[i].w = w;
}

if (BellmanFord(0)) {
    printf("%lld\n", nodes[v-1].dist);
}
else
    printf("ERRORE! C'e` un ciclo negativo\n");
node* s = &nodes[v-1];
while (s->pred != -1)
{
    printf("%d ", s->id);
    s = &nodes[s->pred];
}
printf("\n");
return 0;
}

```

## DIJKSTRA

```

#include <cstdio>
#include <vector>
#include <queue>

#define DEBUG
#ifndef DEBUG
#define print(...) fprintf(stderr, __VA_ARGS__)
#else
#define print(...)
#endif

#define uint unsigned int
using namespace std;

struct node {
    int id;
    int dist;
    int previous;
    bool closed;

    vector <int> adjs;
    vector <int> weights;

    void add_adj(int id, int weight) {
        this->adjs.push_back(id);
        this->weights.push_back(weight);
    }

    int operator() (const node* a, const node* b) {
        return a->dist > b->dist;
    }
};

int n;
priority_queue<node*, vector<node*>, node> q;
vector<node> graph;

void dijkstra(int source)

```

```

{
    int alt;
    for (int i=0; i<n; i++)
    {
        graph[i].id = i;
        graph[i].dist = -1;
        graph[i].previous = -1;
        graph[i].closed = false;
    }
    graph[source].dist = 0;
    q.push(&graph[source]);
    while (!q.empty())
    {
        node* u = q.top();
        q.pop();
        if (!u->closed)
        {
            u->closed = true;
            for (uint i=0; i<u->adjs.size(); i++)
            {
                int adj = u->adjs[i];
                alt = u->dist + u->weights[i];
                if (alt < graph[adj].dist || graph[adj].dist == -1)
                {
                    graph[adj].dist = alt;
                    graph[adj].previous = u->id;
                    q.push(&graph[adj]);
                }
            }
        }
    }
}

int main()
{
    int m;
    int a,b,w;

    scanf("%d";
    graph.erase(graph.begin(), graph.end());
    graph.resize(n);
    scanf("%d", &m);
    for (int i=0; i<m; i++)
    {
        scanf("%d %d %d", &a, &b, &w);
        graph[a].add_adj(b, w);
        graph[b].add_adj(a, w);
    }
    print("GRAPH:\n");
    for (int i=0; i<n; i++)
    {
        print("Neighbour node %d\n", i);
        for (uint j=0; j< graph[i].adjs.size(); j++)
            print("(%d, %d) ", graph[i].adjs[j], graph[i].weights[j]);
        print("\n");
    }
    dijkstra(0);
    for (int i=0; i<n; i++)
        printf("nodo %d distanza %d\n", i, graph[i].dist);
    return 0;
}

```

# GRAFI

```
#include <cstdio>
#include <vector>
#include <utility>
#include <queue>
#include <vector>
#define DEBUG
#ifndef DEBUG
#define print(...) fprintf(stderr, __VA_ARGS__)
#else
#define print(...)
#endif

#define uint unsigned int
#define MAX_N 10000

using namespace std;

struct node {
    vector<int> adjs;
    vector<int> weights;
    bool open;

    void add_adj(int id, int weight) {
        this->adjs.push_back(id);
        this->weights.push_back(weight);
    }
};

vector<node> graph;
long long int mgraph[MAX_N][MAX_N];

void dfs(int node)
// Only visits graph from here. Will not see unreachable nodes!
{
    int adj;
    if (graph[node].open)
    {
        graph[node].open = false;
        // visiting code
        print("Visiting node %d\n", node);
        for (uint i=0; i<graph[node].adjs.size(); i++)
        {
            adj = graph[node].adjs[i];
            dfs(adj);
        }
    }
}

void bfs(int start_node)
// Only visits graph from here. Will not see unreachable nodes!
{
    int node;
    queue<int> q;
    if (graph[start_node].open)
    {
        graph[start_node].open = false;
        q.push(start_node);
    }
    while (!q.empty())
    {
```

```

node = q.front();
q.pop();
// visiting code
print("Visiting node %d\n", node);
for (uint i=0; i<graph[node].adj.size(); i++)
{
    int adj = graph[node].adj[i];
    if (graph[adj].open)
    {
        graph[adj].open = false;
        q.push(adj);
    }
}
}

void full_dfs(){
//dfs call
for (uint i=0; i<graph.size(); i++)
    graph[i].open = true;
print("DFS\n");
for (uint i=0; i<graph.size(); i++)
    dfs(i);
}

void full_bfs(){
// bfs call
for (uint i=0; i<graph.size(); i++)
    graph[i].open = true;
print("BFS\n");
for (uint i=0; i<graph.size(); i++)
    bfs(i);
}

int tree_test()
{
    bool archi=true; // da non copiare
    int n, m;
    int a,b,w;

    scanf("%d";
    graph.erase(graph.begin(), graph.end());
    graph.resize(n);
    if (archi) //parsa input con archi
    {
        scanf("%d";
        for (int i=0; i<m; i++)
        {
            scanf("%d %d %d", &a, &b, &w);
            graph[a].add_adj(b, w);
        }
    }
    else //parsa input con matrice di pesi positivi
    {
        for (int i=0; i<n; i++)
        {
            for (int j=0; j<n; j++)
            {
                scanf("%d", &w);
                if (w!=-1) // -1 null value
                    graph[i].add_adj(j, w);
            }
        }
    }
}

```

```

        }
        //scanf("\n"); // serve se scansiono con %c
    }
}
print("GRAPH:\n");
for (int i=0; i<n; i++)
{
    print("Neighbour node %d\n",i);
    for (uint j=0; j< graph[i].adjs.size(); j++)
        print("(%d,%d) ",graph[i].adjs[j],graph[i].weights[j]);
    print("\n");
}

full_dfs();
full_bfs();

return 0;
}

```

## FLUSSO

```

#include <cassert>
#include <stack>
#include <queue>
#include <vector>
#include <stdio.h>

using namespace std;

#define DEBUG
#ifdef DEBUG
#define print(...) fprintf(stderr, __VA_ARGS__)
#else
#define print(...)
#endif

#define uint unsigned int
#define MAX_N 1000

struct node {
    vector <int> adjs;
    vector <int> weights;
    bool open;

    void add_adj(int id, int weight) {
        this->adjs.push_back(id);
        this->weights.push_back(weight);
    }
};

int n;
vector<node> graph;
int weights[MAX_N][MAX_N];

int bfs_walk(int source, int dest, stack<int>* path)
// true if dest is reachable
// if true, path is erased and properly filled
{
    bool finished = false;
    int node, min;
    int previous[MAX_N];

```

```

queue<int> q;
for (int i=0; i<n; i++)
    graph[i].open = true;
graph[source].open = false;
previous[source] = -1;
q.push(source);
while(!q.empty() && !finished)
{
    node = q.front();
    q.pop();
    print("Visiting node %d\n", node);
    for (uint i=0; i<graph[node].adjs.size() && !finished; i++)
    {
        int adj = graph[node].adjs[i];
        assert(weights[node][adj] >= 0);
        if (weights[node][adj] != 0) { // NB even if adj is here, this does not
                                       // mean the edge actually exists
            if (graph[adj].open)
            {
                graph[adj].open = false;
                previous[adj] = node;
                q.push(adj);
                if (adj==dest)
                    finished = true;
            }
        }
    }
}
if (finished) {
    min = weights[previous[dest]][dest];
    node = dest;
    while (node != source) {
        path->push(node);
        if (weights[previous[node]][node] < min)
            min = weights[previous[node]][node];
        node = previous[node];
    }
    path->push(source);
    return min;
}
return 0;
}

int max_flow(int source, int sink)
{
    stack<int> s;
    int prev, next, val, flow;
    while((val=bfs_walk(source,sink,&s)) > 0)
    {
        prev = s.top();
        s.pop();
        while(!s.empty())
        {
            next = s.top();
            s.pop();
            weights[prev][next] -= val;
            //se l'arco all'indietro non esiste ancora lo aggiungo
            if (weights[next][prev] == 0)
                graph[next].add_adj(prev,0);
            weights[next][prev] += val;
            prev = next;
        }
    }
}

```

```

    }
}

// compute max flow, basing on weight of newly-created edges
// towards the source in the residual capacity graph
flow = 0;
for (uint i=0; i<graph[source].adjs.size(); i++)
{
    int adj = graph[source].adjs[i];
    //versione senza archi entranti nella sorgente
    //flow += weights[adj][source];
    //versione con archi entranti nella sorgente
    flow += graph[source].weights[i] - weights[source][adj];
}
return flow;
}

int main()
{
    int w;
    int flusso;

    scanf("%d", &n);
    graph.erase(graph.begin(), graph.end());
    graph.resize(n);
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            scanf("%d", &w);
            weights[i][j] = w;
            if (w != 0)
                graph[i].add_adj(j, w);
        }
    }
    print("GRAPH:\n");
    for (int i=0; i<n; i++)
    {
        print("Neighbour node %d\n", i);
        for (int j=0; j< n; j++)
            print("%d ", weights[i][j]);
        print("\n");
    }

    flusso = max_flow(0, n-1);
    printf("Flusso massimo %d\n", flusso);

    return 0;
}

```

## MFSET

```

#include <cstdio>
#include <vector>

using namespace std;

struct node {
    int parent;
    int rank;
};


```

```

#define uint unsigned int
struct mfset {
    vector<node> *nodes;

    mfset(vector<node> *nodes) {
        this->nodes = nodes;
        for (uint i=0; i<nodes->size(); i++)
        {
            (*this->nodes)[i].parent = i;
            (*this->nodes)[i].rank = 0;
        }
    }

    int find(int x) {
        if ((*nodes)[x].parent != x)
            (*nodes)[x].parent = this->find((*nodes)[x].parent);
        return (*nodes)[x].parent;
    }

    void merge(int x, int y) {
        int xRoot = this->find(x);
        int yRoot = this->find(y);
        if (xRoot == yRoot)
            return;

        if ((*nodes)[xRoot].rank < (*nodes)[yRoot].rank)
            (*nodes)[xRoot].parent = yRoot;
        else if ((*nodes)[xRoot].rank > (*nodes)[yRoot].rank)
            (*nodes)[yRoot].parent = xRoot;
        else
        {
            (*nodes)[yRoot].parent = xRoot;
            (*nodes)[xRoot].rank = (*nodes)[xRoot].rank + 1;
        }
    }
};

using namespace std;
vector<node> graph;

int main()
{
    int a,b,n,choice;
    mfset *m;
    printf("Number of nodes?\n");
    scanf("%d",&n);
    graph.resize(n);
    m = new mfset(&graph);
    while (true)
    {
        printf("1 -> merge\n2 -> find\n");
        scanf("%d",&choice);
        if (choice==1)
        {
            printf("elementi da mergiare?\n");
            scanf("%d %d",&a,&b);
            m->merge(a,b);
        }
        else
        {

```

```

        printf("elemento di cui cercare rappresentante?\n");
        scanf("%d", &a);
        b = m->find(a);
        printf("è %d\n", b);
    }
    return 0;
}

```

## OVERLOADING

```

struct sell_order {

    int price;
    int *num;
};

bool operator==(const sell_order& so1, const sell_order& so2) {
    return so1.price==so2.price;
}
bool operator<(const sell_order& so1, const sell_order& so2) {
    return so1.price>so2.price;
}

```

## RANGE TREE

```

#include <cstdio>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <time.h>

#define left(n) ((n+1)*2-1)
#define right(n) ((n+1)*2)

using namespace std;

struct point {
    int x, y;

    point(int x, int y) {
        this->x = x;
        this->y = y;
    }

    point() {}
};

bool by_x(const point& a, const point& b)
{
    return a.x < b.x;
}

bool by_y(const point& a, const point& b)
{
    return a.y < b.y;
}

struct node {

```

```

//intervallo (l,r) ricoperto sull'asse x
int l, r;
//punti ordinati secondo asse y
vector<point> points;
bool leave;
};

class range_tree {
vector<node*> tree;

void build_node(int l, int r, point* points, int pos) {
    node* nodo = new node();

    nodo->l = points[l].x;
    nodo->r = points[r].x;
    if (nodo->l == nodo->r)
    {
        nodo->leave = true;
        for (int i=l; i<=r; i++)
            nodo->points.push_back(points[i]);
        sort(nodo->points.begin(), nodo->points.end(), by_y);
    }
    else
    {
        nodo->leave = false;
        int m = (l+r)/2;
        build_node(l,m,points, left(pos));
        build_node(m+1,r,points, right(pos));
        nodo->points.resize(tree[left(pos)]->points.size()+
                             tree[right(pos)]->points.size());
        merge(tree[left(pos)]->points.begin(),
              tree[left(pos)]->points.end(),
              tree[right(pos)]->points.begin(),
              tree[right(pos)]->points.end(),
              nodo->points.begin(),
              by_y);
    }
    if ((int) tree.size() < pos+1)
        tree.resize(pos+2);
    tree[pos] = nodo;
}

void find_recursive(int pos, int x1, int x2, int y1, int y2, vector<point*>* output){
    node* nodo = tree[pos];
    //se l'intervallo del nodo è contenuto nell'intervallo cercato
    //predi tutti
    if (nodo->l >= x1 && nodo->r <= x2) {
        vector<point>::iterator a = lower_bound(nodo->points.begin(),
                                                nodo->points.end(),
                                                point(0,y1),
                                                by_y);
        vector<point>::iterator b = upper_bound(nodo->points.begin(),
                                                nodo->points.end(),
                                                point(0,y2),
                                                by_y)--;
        while (a!=b) {
            output->push_back(*a);
            a++;
        }
    }
}

```

```

//se l'intervallo del nodo interseca l'intervallo cercato
//cerca nel figlio destro e nel sinistro
else if (nodo->l <= x2 && x1 <= nodo->r) {
    find_recursive(left(pos),x1,x2,y1,y2,output);
    find_recursive(right(pos),x1,x2,y1,y2,output);
}
//se sono disgiunti non fai niente
}

public:
//points array di punti, n numero di punti
range_tree(point* points, int n) {
    sort(points,points+n,by_x);
    tree.resize(n);
    build_node(0,n-1,points,0);
}

~range_tree() {
    for (unsigned int i=0; i<tree.size(); i++)
        delete tree[i];
    tree.erase(tree.begin(),tree.end());
}

void find(int x1, int x2, int y1, int y2, vector<point>* output) {
    find_recursive(0,x1,x2,y1,y2,output);
}

void print(int pos, int lev) {
    if (tree[pos]->leave) {
        for (int i=0; i<lev; i++) printf("    ");
        printf("Intervallo %d %d -> ",tree[pos]->l,tree[pos]->r);
        for (int i=0; i<(int)tree[pos]->points.size(); i++)
            printf(" (%d,%d)",tree[pos]->points[i].x,tree[pos]->points[i].y);
        printf("\n");
    }
    else {
        print(left(pos),lev+1);
        for (int i=0; i<lev; i++) printf("    ");
        printf("Intervallo %d %d -> ",tree[pos]->l,tree[pos]->r);
        for (int i=0; i<(int)tree[pos]->points.size(); i++)
            printf(" (%d,%d)",tree[pos]->points[i].x,tree[pos]->points[i].y);
        printf("\n");
        print(right(pos),lev+1);
    }
}
};

range_tree* rt;

#define N 30

int main()
{
    int a, b, c, d, t;
    point points[N];
    vector<point> output;
    srand ( time(NULL) );
    while (true)
    {
        printf("point\n");
        for (int i=0; i<N; i++)
        {

```

```

    points[i].x = rand() % 20 -10;
    points[i].y = rand() % 20 -10;
    printf ("%d,%d) ",points[i].x,points[i].y);
}
printf ("\n");
rt = new range_tree(points,N);
//rt->print(0,0);
a = rand()%20-10; b = rand()%20-10;
if (a>b) {t=a; a=b; b=t;}
c = rand()%20-10; d = rand()%20-10;
if (c>d) {t=c; c=d; d=t;}
printf("search [%d,%d]x[%d,%d]\n",a,b,c,d);
rt->find(a,b,c,d,&output);
printf("result:\n");
for (int i=0; i<(int)output.size(); i++)
    printf("%d,%d) ",output[i].x,output[i].y);
printf ("\n");
delete rt;
output.erase(output.begin(),output.end());
getchar();
}
return 0;
}

```

## SISTEMA LINEARE

```

//numero equazioni
#define MAX_N 1600

int n;
double matrix[MAX_N][MAX_N+1];

void annulla(int riga, int part)
{
    double rapp;
    //printf("azzero riga %d con %d\n",riga,part);
    rapp = matrix[riga][part] / matrix[part][part] * -1;
    for (int j=part; j<=n; j++)
        matrix[riga][j] += matrix[part][j] * rapp;
}

void triangola()
{
    for (int i=0; i<n-1; i++)
    {
        for (int j=i+1; j<n; j++)
            if (matrix[j][i]<-0.00000000000001 || matrix[j][i]>0.00000000000001)
                annulla(j,i);
    }
}

void risolvi()
{
    double val;
    for (int i=n-1; i>=0; i--)
    {
        //printf("risolvo %d\n",i);
        val = matrix[i][n] / matrix[i][i];
        matrix[i][n] /= matrix[i][i];
        matrix[i][i] = 1;
    }
}

```

```

    for (int j=0; j<i; j++)
    {
        matrix[j][n] -= matrix[j][i] * val;
        matrix[j][i] = 0;
    }
    //print_matrix();
}
}

```

## EUCLIDE - GCD

```

#define lint long long int

lint gcd(lint a, lint r) {
    lint new_r = a % r;
    if (new_r == 0)
        return r;
    else
        return gcd(r, new_r);
}

```

## SOMMATORIE

$$\sum_{k=0}^n k^2 = \frac{n(n+1)(2n+1)}{6} \quad \sum_{k=0}^n k^3 = \frac{n^2(n+1)^2}{4} \quad \ln(\prod_{k=1}^n a_k) = \sum_{k=1}^n \ln(a_k)$$

se  $a_0 \dots a_n$  è una successione qualunque, allora:

$$\sum_{k=1}^{n-1} (a_k - a_{k-1}) = a_n - a_0 \quad \sum_{k=0}^{n-1} (a_k - a_{k+1}) = a_0 - a_n$$

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = 1 - \frac{1}{n} \quad (x+y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k} \quad \sum_{k=0}^{n-1} \binom{k}{s} = \binom{n}{s+1} \quad \sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$$

## GEOMETRIA ANALITICA

### Circonferenza

Formula:  $x^2 + y^2 - 2ax - 2by + c = 0$ ,  $c = a^2 + b^2 - r^2$

coefficiente angolare della retta tangente in un suo punto di ascissa  $x_0$ :  $m = \pm \frac{x_0 - a}{\sqrt{r^2 - (x_0 - a)^2}}$

### Ellisse

Formula:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

## CALCOLO COMBINATORIO

-Permutazioni semplici di  $n$  oggetti distinti  $P_n = n!$

-Disposizioni semplici di  $n$  oggetti distinti presi a  $k$  a  $k$  (o di classe  $k$ ),  $k \leq n$   $D_{n,k} = \frac{n!}{(n-k)!}$

-Combinazioni semplici di  $n$  oggetti distinti presi a  $k$  a  $k$  (o di classe  $k$ ), con  $k \leq n$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

-Permutazioni con ripetizioni di  $n$  oggetti di cui  $k$  uguali fra loro ( $k \leq n$ ) e  $(n-k)$  distinti

$$P^{(n)}_n = \frac{n!}{k!}$$

-Disposizioni con ripetizione di  $n$  oggetti presi a  $k$  a  $k$  (o di classe  $k$ )  $D'_{n,k} = n^k$

-Combinazioni con ripetizione di  $n$  oggetti distinti presi a  $k$  a  $k$  (o di classe  $k$ )

$$C'_{n,k} = \binom{n+k-1}{k}$$

## ALGEBRA

$$(a+b+c)^2 = a^2 + b^2 + c^2 + 2ab + 2ac + 2bc$$

$$a^3 + b^3 = (a+b)(a^2 - ab + b^2)$$

$$a^3 - b^3 = (a-b)(a^2 + ab + b^2)$$

## IOMANIP

```
#include <iomanip>
```

**setiosflags(ios\_base::fmtflags mask)** Setta il formato specificato da mask che resta valido fino ad una modifica successiva, come ios::setf( ).

**resetiosflags (ios\_base::fmtflags mask)** Rimuove dal formato corrente lo stile mask.

**setbase (int base)** Setta la base numerica in cui stampare i numeri. Basi valide sono 2, 8, 10, e 16.

**setw (int n)** Numero di caratteri da allocare per il prossimo output: es:

```
cout << setw(10);
cout << "abc";
= "      abc"
```

**setfill (char c)** usato assieme a setw() specifica il carattere da usare per il riempimento:

```
cout << setfill ('x') << setw (10);
cout << 77 << endl;
```

```
= "xxxxxxxx77"
```

**setprecision (int n)** Numero di cifre decimali da stampare se si è in modalità fixed oppure scientific, numero di cifre totali da stampare altrimenti, NOTA: se il numero non necessita di tutta la precisione vengono stampati degli 0.

ios::base flags da usare come mask in formato ios::constante

field	member constant	effect when set
independent flags	boolalpha	read/write bool elements as alphabetic strings (true and false).
	showbase	write integral values preceded by their corresponding numeric base prefix.

	showpoint	write floating-point values including always the decimal point.
	showpos	write non-negative numerical values preceded by a plus sign (+).
	skipws	skip leading whitespaces on certain input operations.
	unitbuf	flush output after each inserting operation.
	uppercase	write uppercase letters replacing lowercase letters in certain insertion operations.
	dec	read/write integral values using decimal base format.
<i>numerical base (basefield)</i>	hex	read/write integral values using hexadecimal base format.
	oct	read/write integral values using octal base format.
<i>float format (floatfield)</i>	fixed	write floating point values in fixed-point notation.
	scientific	write floating-point values in scientific notation.
	internal	the output is padded to the <i>field width</i> by inserting <i>fill characters</i> at a specified internal point.
<i>adjustment (adjustfield)</i>	left	the output is padded to the <i>field width</i> appending <i>fill characters</i> at the end.
	right	the output is padded to the <i>field width</i> by inserting <i>fill characters</i> at the beginning.

## STL

### -Map

```

include <map>
#include <iostream>
using namespace std;
struct comp
{
    bool operator()(const char* s1, const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};

int main()
{
    map<const char*, int, comp> months; // mappa stringhe in interi

    months["january"] = 31;
    ...

    cout << "june -> " << months["june"] << endl;
    map<const char*, int, comp>::iterator cur = months.find("june");
    map<const char*, int, comp>::iterator prev = cur;
    map<const char*, int, comp>::iterator next = cur;
    ++next;
    --prev;
    cout << "Previous (in alphabetical order) is " << (*prev).first << endl;
    cout << "Next (in alphabetical order) is " << (*next).first << endl;
}

```

## -Heap

```
#include <iostream>
#include <algorithm>
#include <iterator>

using namespace std;

//funzione di comparazione x costruire un minheap invece del canonico maxheap
struct cmp_minheap
{
    bool operator()(int a , int b)
    {
        return b<=a;
    }
};

int main()
{
    int arr[]={1,2,3,4,5,6,7,8,9,10};

    int dim=sizeof(arr)/sizeof(int);
    make_heap(arr,arr+dim-1); //costruisci un heap su arr-1

    //stampa a video tutto il contenuto di arr separato da spazi
    copy(arr, arr+dim-1, ostream_iterator<int>(cout, " ")); //9 8 7 4 5 6 3 2 1
    cout << endl;

    push_heap(arr, arr+dim); //inserisco nello heap l' ultimo elemento
    copy(arr, arr+dim, ostream_iterator<int>(cout, " ")); //10 9 7 4 8 6 3 2 1 5
    cout << endl;

    sort_heap(arr,arr+dim);
    copy(arr, arr+dim, ostream_iterator<int>(cout, " ")); //1 2 3 4 5 6 7 8 9 10
    cout << endl;

    int arr2[]={1,2,3,4,5,6,7,8,9,10};
    int dim2=sizeof(arr2)/sizeof(int);
    make_heap(arr2,arr2+dim2);

    pop_heap(arr2, arr2+dim2); //levo l'elemento massimo e lo butto in fondo
    copy(arr2, arr2+dim2-1, ostream_iterator<int>(cout, " ")); //9 8 7 4 5 6 3 1
2
    cout << endl << "arr[N-1] = " << arr2[dim2-1] << endl; // 10

    return 0;
}

//GEOMETRIA COMPUTAZIONALE

#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
```

```

struct point{
    int x;
    int y;
    point() {}
    point(int tx, int ty) {
        x=tx; y=ty;
    }
};

int prodvett(point a, point b, point c){
    return a.x*b.y - a.y*b.x + a.y*c.x - a.x*c.y + b.x*c.y - c.x*b.y;
}

point base;

bool operator < (const point& a, const point &b){
    return (prodvett(base,a,b)<0);
}

vector<point> convexhull(vector<point> points) {
    int id=0;
    for(uint i=1;i<points.size();i++)
        if(points[i].y<points[id].y || (points[i].y==points[id].y &&
points[i].x<points[id].x))
            id=i;
    point start=points[id];
    points[id]=points[points.size()-1];
    points.pop_back();
    vector<point> result;
    result.push_back(start);
    base=start;
    sort(points.begin(),points.end());
    uint i=1;
    result.push_back(points[0]);
    while(i<points.size()){
        if(prodvett(result[result.size()-2],result[result.size()-1],points[i])>=0){
            result.pop_back();
        }else{
            result.push_back(points[i]);
            i++;
        }
    }
    return result;
}

int main()
{
    vector<point> v;
    v.push_back(point(1,1));
    v.push_back(point(3,3));
    v.push_back(point(3,1));
}

```

```

v.push_back(point(1,3));
v.push_back(point(3,2));
v.push_back(point(2,1));
v=convexhull(v);
for(uint i=0;i<v.size();i++)
    cout<<v[i].x<<" "<<v[i].y<<endl;
return 0;
}

```

## CMP

```

int cmp(const void *a, const void *b) {
    const lint *ia = (const lint *)a;
    const lint *ib = (const lint *)b;
    return *ia - *ib;
}
qsort(arr, n, sizeof(lint), cmp);

```

## OPZIONI g++

g++ -Wall -Werror -O2 -g -static

## 2. LE FORMULE DI ADDIZIONE E SOTTRAZIONE

Le formule di addizione

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

$$\tan(\alpha + \beta) = \frac{\tan \alpha + \tan \beta}{1 - \tan \alpha \cdot \tan \beta}$$

La formula della tangente è valida solo se:

$$\alpha + \beta \neq \frac{\pi}{2} + k\pi, \alpha \neq \frac{\pi}{2} + k_1\pi, \beta \neq \frac{\pi}{2} + k_2\pi.$$

Le formule di sottrazione

$$\sin(\alpha - \beta) = \sin \alpha \cos \beta - \cos \alpha \sin \beta$$

$$\cos(\alpha - \beta) = \cos \alpha \cos \beta + \sin \alpha \sin \beta$$

$$\tan(\alpha - \beta) = \frac{\tan \alpha - \tan \beta}{1 + \tan \alpha \cdot \tan \beta}$$

La formula della tangente è valida solo se:

$$\alpha - \beta \neq \frac{\pi}{2} + k\pi, \alpha \neq \frac{\pi}{2} + k_1\pi, \beta \neq \frac{\pi}{2} + k_2\pi.$$

Una funzione del tipo  $y = a \sin x + b \cos x$  può essere ricondotta alla forma sinusoidale

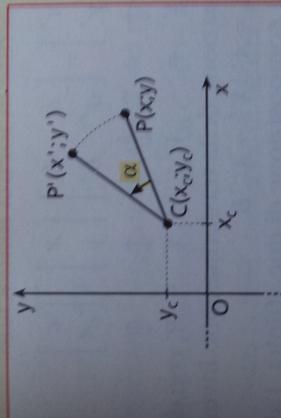
$y = r \sin(x + \alpha)$ , con:

$$r = \sqrt{a^2 + b^2} \quad \text{e} \quad \tan \alpha = \frac{b}{a}.$$

$\alpha$  è detto **angolo aggiunto**.

$$\sin^2 x + \cos^2 x = 1$$

## 5. LA ROTAZIONE



La rotazione di centro  $C(x_c, y_c)$  e angolo  $\alpha$  ha equazioni:

$$\begin{cases} x'' = (x - x_c) \cos \alpha - (y - y_c) \sin \alpha + x_c \\ y'' = (x - x_c) \sin \alpha + (y - y_c) \cos \alpha + y_c \end{cases}$$

che si possono anche scrivere nella forma:

$$\begin{cases} x'' = x \cos \alpha - y \sin \alpha + p \\ y'' = x \sin \alpha + y \cos \alpha + q \end{cases}$$

$$\begin{aligned} \text{con } p &= x_c - x_c \cos \alpha + y_c \sin \alpha \\ q &= y_c - x_c \sin \alpha - y_c \cos \alpha. \end{aligned}$$

## 3. LE FORMULE DI DUPLICAZIONE

$$\sin 2\alpha = 2 \sin \alpha \cos \alpha$$

$$\cos 2\alpha = \cos^2 \alpha - \sin^2 \alpha = \begin{cases} 1 - 2 \sin^2 \alpha \\ 2 \cos^2 \alpha - 1 \end{cases}$$

$$\tan 2\alpha = \frac{2 \tan \alpha}{1 - \tan^2 \alpha}$$

La formula della tangente vale solo se:

$$\alpha \neq \frac{\pi}{4} + k \frac{\pi}{2} \wedge \alpha \neq \frac{\pi}{2} + k\pi.$$

Sono utili anche le seguenti formule:

$$\sin^2 \alpha = \frac{1 - \cos 2\alpha}{2}, \quad \cos^2 \alpha = \frac{1 + \cos 2\alpha}{2}$$

## 4. LE FORMULE DI BISEZIONE

$$\sin \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{2}}$$

$$\cos \frac{\alpha}{2} = \pm \sqrt{\frac{1 + \cos \alpha}{2}}$$

$$\tan \frac{\alpha}{2} = \pm \sqrt{\frac{1 - \cos \alpha}{1 + \cos \alpha}} =$$

$$= \frac{\sin \alpha}{1 + \cos \alpha} = \frac{1 - \cos \alpha}{\sin \alpha}$$

Il segno dipende dal quadrante in cui si trova il lato termine di  $\frac{\alpha}{2}$ .

La formula della tangente vale solo se:

$$\alpha \neq \pi + 2k\pi,$$

per le prime due forme, mentre per la terza solo se:

$$\alpha \neq \pi + k\pi.$$

**Il teorema dei seni.** In un triangolo le misure dei lati sono proporzionali ai seni degli angoli opposti.

$$\frac{a}{\sin \alpha} = \frac{b}{\sin \beta} = \frac{c}{\sin \gamma}$$

$$a^2 = b^2 + c^2 - 2bc \cos \alpha$$

Area triangolo =  
 $\frac{1}{2} * \text{lato1} * \text{lato2}$   
 $* \text{angolo compreso}$

## 5. LE FORMULE PARAMETRICHE

$$\operatorname{sen} \alpha = \frac{2 \operatorname{tg} \frac{\alpha}{2}}{1 + \operatorname{tg}^2 \frac{\alpha}{2}}, \text{ con } \alpha \neq \pi + 2k\pi$$

$$\cos \alpha = \frac{1 - \operatorname{tg}^2 \frac{\alpha}{2}}{1 + \operatorname{tg}^2 \frac{\alpha}{2}}, \text{ con } \alpha \neq \pi + 2k\pi$$

## 6. LE FORMULE DI PROSTAFERESI E DI WERNER

### Le formule di prostaferesi

$$\operatorname{sen} p + \operatorname{sen} q = 2 \operatorname{sen} \frac{p+q}{2} \cdot \cos \frac{p-q}{2}$$

$$\operatorname{sen} p - \operatorname{sen} q = 2 \cos \frac{p+q}{2} \cdot \operatorname{sen} \frac{p-q}{2}$$

$$\cos p + \cos q = 2 \cos \frac{p+q}{2} \cdot \cos \frac{p-q}{2}$$

$$\cos p - \cos q = -2 \operatorname{sen} \frac{p+q}{2} \cdot \operatorname{sen} \frac{p-q}{2}$$

### Le formule di Werner

$$\operatorname{sen} \alpha \operatorname{sen} \beta = \frac{1}{2} [\cos(\alpha - \beta) - \cos(\alpha + \beta)]$$

$$\cos \alpha \cos \beta = \frac{1}{2} [\cos(\alpha + \beta) + \cos(\alpha - \beta)]$$

$$\operatorname{sen} \alpha \cos \beta = \frac{1}{2} [\operatorname{sen}(\alpha + \beta) + \operatorname{sen}(\alpha - \beta)]$$

## 7. IL PERIODO DELLE FUNZIONI GONIOMETRICHE

Se una funzione è la somma di due funzioni goniometriche, il suo periodo si determina come nell'esempio seguente.

► Calcoliamo il periodo di  $y = \operatorname{sen} 4x - \operatorname{sen} 7x$ . Il periodo di  $\operatorname{sen} 4x$  è  $\frac{2\pi}{4} = \frac{\pi}{2}$ , quello di  $\operatorname{sen} 7x$  è  $\frac{2\pi}{7}$ .

Scriviamo i periodi con lo stesso denominatore:

$$\frac{\pi}{2} = \frac{7}{14}\pi; \quad \frac{2}{7}\pi = \frac{4}{14}\pi,$$

$$\text{m.c.m.}(7, 4) = 28.$$

$$\text{Il periodo cercato è } \frac{28}{14}\pi = 2\pi.$$

Per particolari funzioni possiamo utilizzare le formule studiate nell'unità.

► Calcoliamo il periodo di  $y = \frac{\operatorname{sen} x}{2 \cos \frac{x}{2}}$ :

$$\frac{\operatorname{sen} x}{2 \cos \frac{x}{2}} = \frac{\operatorname{sen} 2 \cdot \frac{x}{2}}{2 \cos \frac{x}{2}} =$$

$$= \frac{2 \operatorname{sen} \frac{x}{2} \cos \frac{x}{2}}{2 \cos^2 \frac{x}{2}} = \operatorname{sen} \frac{x}{2}.$$

$$\text{Il periodo della funzione è } \frac{2\pi}{\frac{1}{2}} = 4\pi.$$

## 4. L'INTEGRAZIONE PER PARTI

La formula di **integrazione per parti** è la seguente:  $\int f(x)g'(x)dx = f(x) \cdot g(x) - \int f'(x)g(x)dx$ .

► Calcoliamo  $\int x \cdot \cos x dx$ .

Scegliamo  $g'(x) = \cos x$  e  $f(x) = x$ .

•  $\int \cos x dx = \operatorname{sen} x + c$ , quindi  $g(x) = \operatorname{sen} x$ ;

• calcoliamo  $f'(x) = 1$ ;

• applichiamo la formula:

$$\int x \cdot \cos x dx = x \cdot \operatorname{sen} x - \int 1 \cdot (\operatorname{sen} x) dx = \\ = x \cdot \operatorname{sen} x + \cos x + c.$$

$$\blacktriangleright \int \ln x dx = x(\ln x - 1) + c.$$

$$\blacktriangleright \int \log_2 x dx = \int \frac{\ln x}{\ln 2} dx = \frac{x(\ln x - 1)}{\ln 2} + c.$$

$$\int x^\alpha dx = \frac{x^{\alpha+1}}{\alpha+1} + c \quad (\alpha \in \mathbb{R} - \{-1\})$$

$$\int \frac{1}{x} dx = \ln|x| + c$$

$$\int e^x dx = e^x + c$$

$$\int a^x dx = \frac{a^x}{\ln a} + c$$

$$\int \sin x dx = -\cos x + c$$

$$\int \cos x dx = \sin x + c$$

$$\int \frac{1}{\cos^2 x} dx = \tan x + c$$

$$\int \frac{1}{\sin^2 x} dx = -\cot x + c$$

$$\int \frac{1}{\sqrt{1-x^2}} dx = \begin{cases} \arcsin x + c \\ -\arccos x + c \end{cases}$$

$$\int \frac{1}{1+x^2} dx = \begin{cases} \arctan x + c \\ -\text{arccot } x + c \end{cases}$$

## 4. LE DISTRIBUZIONI DI PROBABILITÀ DI USO FREQUENTE

### La distribuzione uniforme discreta

La variabile casuale discreta  $X$  assume i valori  $x_1, x_2, \dots, x_n$  e tutti i valori *banno la stessa probabilità*  $P(X=x) = \frac{1}{n}$ , dove  $n$  è il numero dei valori che può assumere  $X$ .

Se i valori di  $X$  sono  $1, 2, 3, \dots, n$ , il valore medio e la varianza valgono rispettivamente:

$$M(X) = \frac{n+1}{2};$$

$$var(X) = \frac{n^2-1}{12}.$$

### La distribuzione binomiale o di Bernoulli

La variabile casuale discreta  $X$  assume i valori  $0, 1, 2, \dots, n$  relativi al numero delle volte in cui si è verificato un evento sottoposto a  $n$  prove, tutte nelle *stesse condizioni*; la probabilità  $p$  del verificarsi dell'evento è ogni volta la stessa.

Essendo  $q = 1-p$  la probabilità contraria, e  $x$  il numero delle volte in cui si è verificato l'evento, dallo schema delle prove ripetute otteniamo la distribuzione di probabilità:

$$P(X=x) = \binom{n}{x} p^x q^{n-x}$$

In questo caso, abbiamo:

$$M(X) = np \quad \text{e} \quad var(X) = npq.$$

### La distribuzione ipergeometrica

La variabile casuale discreta  $X$  assume i valori  $0, 1, 2, \dots, n$  relativi al numero delle volte in cui si è verificato un evento sottoposto a  $n$  prove avvenute contemporaneamente (o successivamente in modo dipendente).

Se effettuiamo le prove su  $N$  elementi di cui  $K$  verificano l'evento e  $N-K$  non lo verificano, la distribuzione di probabilità della variabile  $X$  è:

$$P(X=x) = \frac{\binom{K}{x} \binom{N-K}{n-x}}{\binom{N}{n}}.$$

Ponendo  $p = \frac{K}{N}$  e  $q = \frac{N-K}{N}$ , si ottiene:

$$M(X) = np \quad \text{e} \quad var(X) = npq \frac{N-n}{N-1}.$$

Al crescere di  $N$  la distribuzione ipergeometrica tende alla distribuzione binomiale.

### La distribuzione di Poisson

La variabile casuale discreta  $X$  assume un numero illimitato di valori  $0, 1, 2, 3, \dots$  relativi al numero delle volte in cui può verificarsi

un evento in un intervallo di tempo o in un contesto fissato. La sua distribuzione di probabilità è:

$$P(X = x) = \frac{\lambda^x}{x!} e^{-\lambda}$$

dove  $\lambda$  è un parametro che coincide con il valore medio e con la varianza di  $X$ :

$$M(X) = \lambda \quad \text{e} \quad \text{var}(X) = \lambda.$$

Questa distribuzione approssima quella binomiale quando il numero delle prove  $n$  è elevato e il valore della probabilità  $p$  è piccolo essendo  $\lambda = np$ .

## 5. IL TEOREMA DI BIENAYMÉ-CEBICEV E IL TEOREMA DI BERNOULLI

Il teorema di Bienaym -Cebicev permette di calcolare la probabilit  che i valori di una variabile casuale  $X$  differiscano in valore assoluto dal suo valore medio  $\mu$  di almeno un valore  $\varepsilon > 0$ , essendo noto il valore dello scarto quadratico medio  $\sigma$ :

$$P(|X - \mu| \geq \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2} \quad \text{con } \varepsilon > \sigma.$$

### Teorema di Bernoulli

Se un evento ha probabilit  costante  $p$  di verificarsi a ogni prova, la probabilit  che la frequenza relativa differisca in valore assoluto da  $p$  per meno di  $\varepsilon > 0$ , piccolo a piacere, tende a 1 quando il numero delle prove tende all'infinito.

## 6. LE VARIABILI CASUALI STANDARDIZZATE

Una variabile casuale  $X$ , avente valore medio  $\mu$  e scarto quadratico medio  $\sigma$ , pu  essere trasformata in un'altra variabile casuale  $Z$ , detta la **variabile standardizzata di  $X$** , mediante la seguente relazione:

$$Z = \frac{X - \mu}{\sigma}$$

I valori di questa nuova variabile casuale, detti **punti  $z$** , mantengono le stesse probabilit  dei valori di  $X$ . Per  $Z$  si ha:  $\mu = 0$  e  $\sigma = 1$ .

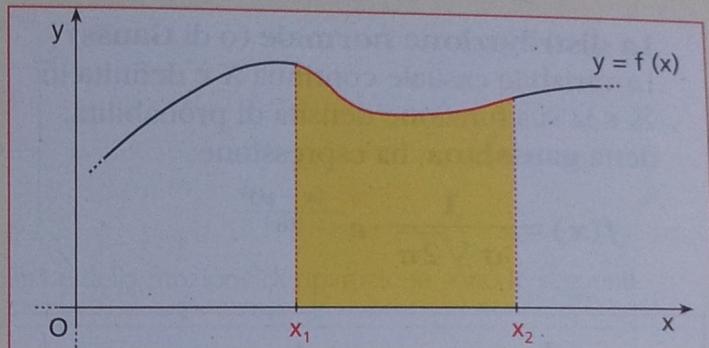
## 7. LE VARIABILI CASUALI CONTINUE

Una **variabile casuale  $X$**  si dice **continua** se assume tutti i valori reali compresi in un intervallo  $I$  che pu  essere limitato o illimitato. La probabilit  che una variabile casuale conti-

nua assuma valori contenuti in certo intervallo  $[x_1; x_2]$ , ossia  $P(x_1 \leq X \leq x_2)$ ,   determinata dall'area compresa tra il grafico di una particolare funzione  $f(x)$ , detta **funzione densit  di probabilit **, e l'asse delle ascisse nell'intervallo  $[x_1; x_2]$ .

Per avere una distribuzione di probabilit  occorre che  $f(x)$  soddisfi alle seguenti condizioni:

1.  $f(x) \geq 0$  in  $\mathbb{R}$ ;
2. l'area sottesa dal grafico di  $f(x)$  in  $\mathbb{R}$    uguale a 1.



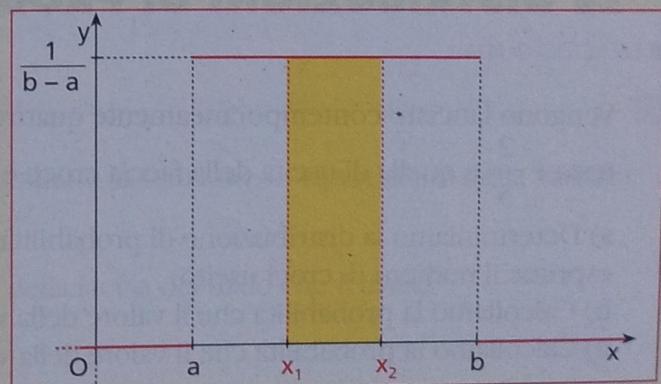
Analogamente al caso discreto possiamo parlare di **funzione di ripartizione**

$F(x) = P(X \leq x)$  e di valore medio, varianza e scarto quadratico medio.

### La distribuzione uniforme continua

La variabile casuale continua  $X$    definita in un intervallo limitato  $[a; b]$  e la sua funzione densit  di probabilit   :

$$f(x) = \begin{cases} 0 & \text{se } x < a \\ \frac{1}{b-a} & \text{se } a \leq x \leq b \\ 0 & \text{se } x > b \end{cases}$$



$$P(x_1 \leq X \leq x_2) = \int_{x_1}^{x_2} \frac{1}{b-a} dx = \frac{x_2 - x_1}{b-a}.$$

Funzione di ripartizione:

$$F(x) = \begin{cases} 0 & \text{se } x < a \\ \frac{x-a}{b-a} & \text{se } a \leq x \leq b \\ 1 & \text{se } x > b \end{cases}$$

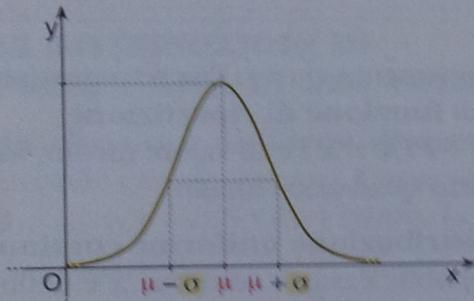
Valore medio:  $M(X) = \frac{a+b}{2};$

Varianza:  $\text{var}(X) = \frac{(b-a)^2}{12}.$

### La distribuzione normale (o di Gauss)

La variabile casuale continua  $X$  è definita in  $\mathbb{R}$  e la sua funzione densità di probabilità, detta gaussiana, ha espressione:

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$



a. Grafico della gaussiana di parametri  $\mu$  e  $\sigma$ .

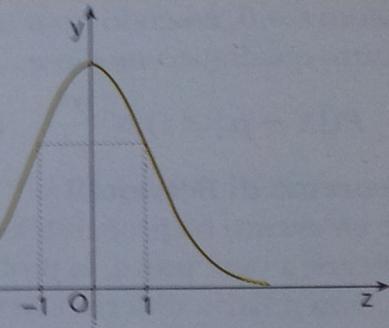
La variabile  $X$ , detta variabile casuale normale e indicata con  $N(\mu; \sigma^2)$ , ha valore medio  $\mu$  e scarto quadratico medio  $\sigma$ .  $P(x_1 \leq X \leq x_2)$  si calcola trasformando  $X$  nella variabile casuale standardizzata  $Z = N(0; 1)$ , che ha funzione gaussiana:

$$f(z) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{z^2}{2}}$$

e utilizzando la tavola di Sheppard che fornisce il valore  $P(z)$  dell'area sottostante la curva  $f(z)$  per valori  $z \geq 0$ . Proprietà che derivano dalla simmetria della curva gaussiana:

$$P(-z < Z < 0) = P(0 < Z < z);$$

$$P(-\infty < Z < -z) = 0,5 - P(0 < Z < z).$$



b. Grafico della gaussiana standardizzata.

$$\int [f(x)]^\alpha f'(x) dx = \frac{|f(x)|^{x+1}}{\alpha+1} + c \quad (\alpha \in \mathbb{R} - \{-1\}) \quad \int \frac{f'(x)}{\cos^2 f(x)} dx = \operatorname{tg} f(x) + c$$

$$\int \frac{f'(x)}{f(x)} dx = \ln |f(x)| + c \quad \int \frac{f'(x)}{\sin^2(f(x))} dx = -\operatorname{cotg} f(x) + c$$

$$\int f'(x) e^{f(x)} dx = e^{f(x)} + c \quad \int \frac{f'(x)}{\sqrt{1 - [f(x)]^2}} dx = \begin{cases} \arcsen f(x) + c \\ -\arccos f(x) + c \end{cases}$$

$$\int f'(x) a^{f(x)} dx = \frac{a^{f(x)}}{\ln a} + c \quad \int \frac{f'(x)}{1 + [f(x)]^2} dx = \begin{cases} \arctg f(x) + c \\ -\operatorname{arccotg} f(x) + c \end{cases}$$

$$\int f'(x) \operatorname{sen} f(x) dx = -\cos f(x) + c \quad \int \frac{f'(x)}{\sqrt{a^2 - [f(x)]^2}} dx = \begin{cases} \arcsen \frac{f(x)}{|a|} + c \\ -\arccos \frac{f(x)}{|a|} + c \end{cases} \quad (a \in \mathbb{R} - \{0\})$$

$$\int f'(x) \cos f(x) dx = \operatorname{sen} f(x) + c \quad \int \frac{f'(x)}{a^2 + [f(x)]^2} dx = \begin{cases} \frac{1}{a} \operatorname{arctg} \frac{f(x)}{a} + c \\ -\frac{1}{a} \operatorname{arccotg} \frac{f(x)}{a} + c \end{cases} \quad (a \in \mathbb{R} - \{0\})$$

## Le derivate

### Potenze di $x$

$$D k = 0$$

$$D x^a = ax^{a-1}, \quad a \in \mathbb{R}$$

$$D x = 1$$

$$D \sqrt{x} = \frac{1}{2\sqrt{x}}, \quad x > 0$$

$$D \sqrt[n]{x} = \frac{1}{n \sqrt[n]{x^{n-1}}}, \quad x > 0, n \in \mathbb{N}$$

$$D \left( \frac{1}{x} \right) = -\frac{1}{x^2}$$

### Funzioni logaritmiche ed esponenziali

$$D a^x = a^x \ln a, \quad a > 0$$

$$D e^x = e^x$$

$$D \log_a x = \frac{1}{x} \log_a e, \quad x > 0$$

$$D \ln x = \frac{1}{x}, \quad x > 0$$

### Funzioni goniometriche

$$D \sin x = \cos x$$

$$D \sin x^\circ = \frac{\pi}{180^\circ} \cos x^\circ$$

$$D \cos x = -\sin x$$

$$D \cos x^\circ = -\frac{\pi}{180^\circ} \sin x^\circ$$

$$D \operatorname{tg} x = \frac{1}{\cos^2 x} = 1 + \operatorname{tg}^2 x$$

$$D \operatorname{cotg} x = -\frac{1}{\sin^2 x} = -(1 + \operatorname{cotg}^2 x)$$

### Inverse delle funzioni goniometriche

$$D \operatorname{arctg} x = \frac{1}{1+x^2}$$

$$D \operatorname{arcotg} x = -\frac{1}{1+x^2}$$

$$D \operatorname{arcsen} x = \frac{1}{\sqrt{1-x^2}}$$

$$D \operatorname{arccos} x = -\frac{1}{\sqrt{1-x^2}}$$

## Le regole di derivazione

$$D [k \cdot f(x)] = k \cdot f'(x)$$

$$D [f(x) + g(x)] = f'(x) + g'(x)$$

$$D [f(x) \cdot g(x)] = f'(x) \cdot g(x) + f(x) \cdot g'(x)$$

$$D [f(x) \cdot g(x) \cdot z(x)] = f'(x) \cdot g(x) \cdot z(x) + f(x) \cdot g'(x) \cdot z(x) + f(x) \cdot g(x) \cdot z'(x)$$

$$D [f(x)]^a = a [f(x)]^{a-1} \cdot f'(x), \quad a \in \mathbb{R}$$

$$D \left[ \frac{1}{f(x)} \right] = -\frac{f'(x)}{f^2(x)}$$

$$D \left[ \frac{f(x)}{g(x)} \right] = \frac{f'(x) \cdot g(x) - f(x) \cdot g'(x)}{g^2(x)}$$

$$D [f(g(x))] = f'(z) \cdot g'(x), \quad \text{con } z = g(x)$$

$$D [f(g(z(x)))] = f'(u) \cdot g'(t) \cdot z'(x), \quad \text{con } t = z(x), u = g(t)$$

$$D [f(x)]^{g(x)} = [f(x)]^{g(x)} \left[ g'(x) \ln f(x) + \frac{g(x) \cdot f'(x)}{f(x)} \right]$$

$$D [f^{-1}(y)] = \frac{1}{f'(x)}, \quad \text{con } x = f^{-1}(y)$$