Ninivaggi Luca 5312342

MILLER-RABIN PRIMALITY TEST

Utilizziamo un codice in c++ per implementare l' algoritmo Miller-Rabin, integrandolo in modo che il campionamento casuale sia invece un calcolo completo su tutti i valori compresi tra 2 e n-2.

```cpp
struct QUAD {
    int a;
    vector<long> elenco;
    bool testimone;
};

struct NUMERO {
    int n;
    int q;
    int s;
    int n_testimoni;
    int n_bugiardi;
    vector<int> Z_star;
    vector<QUAD> quadratura;
};
```

Dal punto di vista pratico, utilizziamo una struct NUMERO che contiene tutte le informazioni associate a un numero (dispari e maggiore o uguale a di 3):

- la sua scomposizione con q ed s calcolati secondo

$$n = q \cdot 2^s + 1$$

- le quadrature di tutti i suoi valori
- il numero di testimoni e il numero di bugiardi
- il "gruppo moltiplicativo modulo n".

Ogni quadratura è invece relativa a un valore, è descritta da un elenco di interi e può rendere "a" un testimone o un bugiardo.

Utilizziamo inoltre alcune funzioni ausiliarie:

- MCD(a,b) ci permette di poter calcolare il gruppo moltiplicativo di n.
- potenza_mod(a,q,n), utilizzata per poter calcolare l' espressione

$$x \equiv a^q \pmod{n}$$

è stata necessaria poiché q può essere molto grande, e quindi i calcoli rischiano di causare un overflow e rendere inutilizzabile il codice.

```cpp
int MCD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

bool coprimi(int a, int b) {
    return MCD(a,b) == 1;
}

int potenza_mod(int a, int q, int n) {
    int ris = 1;
    while (q > 0) {
        if (q % 2 == 1) {
            ris *= a;
            ris %= n;
        }
        a *= a;
        a %= n;
        q /= 2;
    }
    return ris;
}
```

Di seguito è riportato il contenuto della funzione chiamata sui numeri in input.

```
void primalityTest(int n) {
    NUMERO NUM;
    NUM.n = n;
    NUM.s = 0;
    NUM.q = NUM.n-1;
    while (NUM.q % 2 == 0) {
        NUM.q/=2;
        NUM.s++;
    }
    NUM.n_testimoni = 0;
    NUM.n_bugiardi = 0;

    for (int i = 0; i < n; i++)
        if (coprimi(i,n)) NUM.Z_star.push_back(i);

    for (int a = 2; a < n-1; a++) {
        QUAD quad;
        quad.a = a;
        quad.testimone = true;

        int x = potenza_mod(quad.a,NUM.q,NUM.n);
        quad.elenco.push_back(x == NUM.n -1 ? -1 : x);
        if (x == 1 || x == -1 % NUM.n)
            quad.testimone = false;

        for (int i = 1; i <= NUM.s; i++) {
            if (x == -1 || x == NUM.n-1)
                quad.testimone = false;
            x = int(pow(x,2)) % NUM.n;
            quad.elenco.push_back(x == NUM.n -1 ? -1 : x);
        }
        NUM.quadratura.push_back(quad);
        if (quad.testimone) NUM.n_testimoni++;
        else NUM.n_bugiardi++;
    }
    stampa(NUM,true);
}
```

La funzione calcola inizialmente i valori q,s per poter riscrivere n.

Calcola poi anche l' insieme Z^*_n associato

E infine va a produrre la sequenza delle quadrature

Si noti come per completezza, venga calcolata tutta la sequenza anche per le quadrature "bugiarde", mentre di fatto non sarebbe necessario, basterebbe fermarsi appena si incontra un -1.

RISULTATI OTTENUTI

Osservando i risultati ottenuti, dovremmo poter confermare che, con n composto, abbiamo un abbondanza di testimoni.

La teoria ci dice che se un numero n è primo, tutti i numeri inferiori ad n, hanno una sequenza

$$\{a^q, a^{q2}, \ldots, a^{q2^{s-1}}, a^{q2^s}\}$$

Del tipo

$$\overbrace{(1,\ldots,1)}^{\text{tutti 1}} \text{ o } (\ldots,-1,\overbrace{1,\ldots,1}^{\text{tutti 1}})$$

Perciò se troviamo almeno un valore a per cui la sequenza osservata non è simile a queste, allora n è composto.

L' efficienza della ricerca di un a "testimone" è garantita dalla formula che ci che i bugiardi sono meno di

$$\mathcal{P}(n) < \frac{|\mathbb{Z}_n^*|/2}{(n-1)} < \frac{(n-1)/2}{(n-1)} = \frac{1}{2}$$

Dove P(n) è la probabilità di campionare un bugiardo, se n composto.

```
N = 11
#Z* = 10
scomposizione: 5 * 2 ^1
#testimoni = 0
#bugiardi = 8
rapporto testimoni/tot: 0/8=0
   a = 2--> [-1, 1]
   a = 3--> [1, 1]
   a = 4--> [1, 1]
   a = 5--> [1, 1]
   a = 6--> [-1, 1]
   a = 7--> [-1, 1]
   a = 8--> [-1, 1]
   a = 9--> [1, 1]
```

Come dimostrazione, scegliendo n = 11, numero primo, osserviamo che non abbiamo testimoni, infatti abbiamo che tutti i valori di a hanno una "quadratura" che soddisfa quanto scritto prima.

Utilizzando un numero primo la formula risulta inoltre al suo caso limite: la probabilità sarà infatti inferiore sicuramente a ½, poiché Z^*_n è l' insieme di tutti i numeri coprimi con n, cioè tutti gli n-1.

Con N = 985 osserviamo che abbiamo 978 testimoni della sua composizione, e 4 valori che sono dei bugiardi, ovvero che se campionati ci farebbero pensare che il numero sia primo.

Coerentemente con quanto descritto prima, la probabilità di campionare un testimone è 0.995

Per completezza, sono indicati i valori bugiardi con la relativa sequenza.

Si conferma che la probabilità di campionare un bugiardo è inferiore a 784/(2*984) = 39%

```
N = 985
#Z* = 784
scomposizione: 123 * 2 ^3
#testimoni = 978
#bugiardi = 4
rapporto testimoni/tot: 978/982=0.995927
   a = 183--> [802, -1, 1, 1]
   a = 408--> [577, -1, 1, 1]
   a = 577--> [408, -1, 1, 1]
   a = 802--> [183, -1, 1, 1]
```

```
N = 1079
#Z* = 984
scomposizione: 539 * 2 ^1
#testimoni = 1076
#bugiardi = 0
rapporto testimoni/tot: 1076/1076=1
```

Con N = 1079, abbiamo invece tutti i valori che sono testimoni: in questo caso la probabilità di concludere erroneamente che n è primo è nulla.

```
N = 1241
#Z* = 1152
scomposizione: 155 * 2 ^3
#testimoni = 1218
#bugiardi = 20
rapporto testimoni/tot: 1218/1238=0.983845
   a = 83--> [927, 557, -1, 1]
   a = 168--> [1012, 319, -1, 1]
   a = 229--> [1073, 922, -1, 1]
   a = 270--> [740, 319, -1, 1]
   a = 314--> [1158, 684, -1, 1]
   a = 319--> [922, -1, 1, 1]
   a = 355--> [825, 557, -1, 1]
   a = 416--> [886, 684, -1, 1]
   a = 501--> [971, 922, -1, 1]
   a = 557--> [684, -1, 1, 1]
   a = 684--> [557, -1, 1, 1]
   a = 740--> [270, 922, -1, 1]
   a = 825--> [355, 684, -1, 1]
   a = 886--> [416, 557, -1, 1]
   a = 922--> [319, -1, 1, 1]
   a = 927--> [83, 684, -1, 1]
   a = 971--> [501, 319, -1, 1]
   a = 1012--> [168, 922, -1, 1]
   a = 1073--> [229, 319, -1, 1]
   a = 1158--> [314, 557, -1, 1]
```

Infine, con n =1241 notiamo come ci siano un gran numero di bugiardi, ma nonostante questo, la probabilità di non sbagliarsi è praticamente del 100%.

Anche in questo caso, la probabilità è inferiore a 46%, perciò la formula è verificata.