“Algoritmi efficienti per la generazione e l’individuazione dei numeri primi gemelli”

Autori:

Aleksandre Chikviladze

Roberto Cito

Daniele Carpentieri

Simone Abbatiello

Arjel Buzi

Introduzione

La generazione di numeri primi e la ricerca di uno schema deterministico per generarli sono sempre stati rilevanti in campi come la matematica e l'informatica. Proprio questo studio si basa sulla creazione e realizzazione di algoritmi basati sulle regole di selezione individuate dall'articolo scientifico:

"***The set of prime numbers: Multifractals and multiscale analysis***".

L'algoritmo creato si presta a trovare un numero "n" di coppie di numeri primi gemelli, in un dato tempo t, individuato dal programma stesso. Tali valori di tempo e numero di coppie verranno inserite all'interno di un grafico che mostrerà l'evoluzione della curva che mette in relazione velocità/tempo di esecuzione e il numero di coppie trovate al variare del tempo t.

Per il progetto in questioni sono stati utilizzati diversi sistemi operativi e diverse macchine con prestazioni differenti. I sistemi utilizzati sono i seguenti:

Sistema 1

Processore: i9-11980HK;

RAM: 32GB 3200Mhz;

S.O: Windows 11

Sistema 2

Processore: i9-11980HK;

RAM: 32GB 3200Mhz;

S.O: Linux Ubuntu 24.04LTS

Sistema 3

Processore: i5-7200U;

RAM: 8GB 2133Mhz;

S.O: MacOS Sonoma 14.3.1

Metodo

All'interno dell'articolo scientifico sono state individuate delle formule per generare in modo efficace i numeri da studiare. Le formule considerate rappresentano le due classi dell'insieme dei candidati a numeri primi. Tali classi sono nella forma:

• $C1 = 6k - 1 \; con \; k \in R$

• $C2 = 6k + 1 \;\; con \; k \in R$

Tramite la scomposizione di queste formule è possibile estrapolare delle formule equivalenti che permettono la generazione e l'esclusione dei candidati numeri primi:

• $S1 = 36xy + 6x - 6y - 1 \; derivante \; dalla \; formula \; C1$

Eguagliandola alla formula iniziale è possibile ottenere:

$$36xy \; + \; 6x - 6y - 1 = 6k - 1 \; da \; cui \; ricaviamo \; K1 \; = \; 6xy + x + y$$

La formula K1 permette di generare le posizioni dei numeri non primi all'interno dalla sequenza che è generata dalla formula C1 così da poterli escludere.

$S2 = 36xy - 6x - 6y + 1 \; e \; S3 = 36xy + 6x + 6y + 1 \; derivanti \; dalla \; formula \; C2.$

Eguagliandoli a quella formula iniziale è possibile ottenere:

$$36xy - 6x - 6y - 1 = 6k + 1 \; da\; cui\; ricaviamo\; K2 = 6xy - x - y$$

$$36xy + 6x + 6y + 1 = 6k + 1 \; da\; cui\; ricaviamo\; K3 = 6xy + x + y$$

Come in precedenza, anche qui è possibile estrapolare due formule di esclusione per i numeri non primi, le formule K2 e K3 utilizzate entrambi per la formula C2.

Infine, grazie alla definizione, individuare numeri primi gemelli è molto semplice: dati due numeri primi p e q essi sono primi gemelli se e solo se:

$p < q \; e \; q = p + 2$.

Proprio grazie alle formule di generazione precedentemente individuate è stato possibile procedere con la stesura del codice.

Innanzitutto, vengono allocati dinamicamente due array che conterranno tutti i numeri primi individuati dalle formule generatrici. Altri tre array dinamici sono invece utilizzati per salvare tutte le posizioni individuate dalle formule di esclusione. Queste posizioni saranno sfruttate da una funzione che permette di sostituire tutti i numeri non primi con zero. Una volta effettuata questa operazione avviene il mergesort dei due array di numeri che successivamente conterranno sicuramente soltanto numeri primi. Come ultimo passo, tramite una semplice funzione che controlla se i numeri individuati sono effettivamente primi, vengono riconosciute le coppie di numeri primi gemelli sfruttando la definizione spiegata prima.

```c
#include <stdio.h>
#include <stdlib.h>
int count=0;

int myAtoi(char *s) { //Ricreazione dell'atoi()
    int n=0;
    while(*s) {
        n=(n*10)+(*s-'0');
        s++;
    }
    return n;
}

/*
Questa funzione si occupa di rimuovere tutti i numeri non primi da c attraverso i k calcolati.
Per farlo, semplicemente imposta tutti i numeri rilevati come non primi a 0.
*/
void removeNotPrimes(long long int k[], long long int c[], long long int n, long long int n3) {
    long long int n2;
    for(long long int i=0;i<n;i++) { //Scorro per tutto k
        n2=k[i]-1; //Mi salvo il k dentro l'array e diminuisco di 1 (questo perché gli array
partono da 0)
        if (n2<n3) { //Questo controllo copre il caso in cui l'indirizzo dato da k sia
maggiore di quello disponibile in c, nel caso lo sia non va a modificare (evito il segmentation
fault)
            c[n2]=0; //In caso contrario, imposta la posizione k in c a 0 (così da skipparla
nella stampa)
        }
    }
}

/*
Questa funzione si occupa di fare il merge tra i due array di calcolo, così che possa
restituire un array fusione (tra i 2 array passati) ordinato.
Essa si basa sul principio del Merge Sort.
*/
long long int *merge(long long int a1[], long long int a2[],long long int n1, long long int n2)
{
 long long int i,j,k; //Dichiara 3 interi che verrano usati come puntatori
    long long int *copia=calloc(n1+n2,sizeof(long long int)); //Crea l'array copia di
dimensione uguale alla somma delle dimensioni dei 2 array passati
 for(i=0,j=0,k=0;i<n1 && j<n2;k++) { //Inizia il ciclo (ad ogni iterazione vado avanti
sull'array copia) e si ferma quando uno dei due array è terminato
  if (a1[i]<=a2[j]) { //Se il numero del 1°array è minore o uguale a quello del 2°
   copia[k]=a1[i]; //Copio dal 1°
   i++; //Vado avanti sul 1°
  }
  else { //In caso contario
   copia[k]=a2[j]; //Copia il numero del 2°array nell'array copia
   j++; //Vado avanti sull'array 2
  }
 }
 for(;i<n1;i++,k++) copia[k]=a1[i]; //Copia i restanti elementi dall'array ancora non finito
 for(;j<n2;j++,k++) copia[k]=a2[j];
 return copia; //Restituisco l'array copia
}

//Questa funzione si occupa di trovare e stampare tutti i numeri primi gemelli trovati
void findGemelli(long long int c[], long long int n) {
    long long int i2, i3=1; //Questi due puntatori, uno corrisponderà al numero successivo per
trovare le coppie (i2), l'altro invece si occupa di tenere traccia di quante coppie sono state
trovate
    for(long long int i=0;i3<=n;i=i2) { //i scorre l'array ed ad ogni iterazione viene
reimpostato ad i2
        while(c[i]==0) i++; //Scorre finché ci sono gli 0 (visto che sono da saltare)
        i2=i+1; //i2 va nella posizione avanti a quella di i
        while(c[i2]==0) i2++; //Scorre finché ci sono gli 0 (visto che sono da saltare)
```

```
        if ((c[i]+2)==c[i2]) { //Se il numero dato dalla posizione i, sommato di 2, è uguale a
quello presente nella posizione i2, allora è stata trovata una coppia di numeri primi gemelli
            printf("%lld Coppie di gemelli: %lld %lld\n",i3,c[i],c[i2]); //Viene stampata
            i3++; //E viene incrementato il contatore di coppie di numeri primi
        }
    }
}

//Questa funzione si occupa di trovare e stampare, su file, tutti i numeri primi gemelli
trovati
void findAndSaveGemelli(long long int c[], long long int n) {
    long long int i2, i3=1; //Questi due puntatori, uno corrisponderà al numero successivo per
trovare le coppie (i2), l'altro invece si occupa di tenere traccia di quante coppie sono state
trovate
    FILE *fp;
    fp=fopen("output.txt","w"); //Apro il file
    for(long long int i=0;i3<=n;i=i2) { //i scorre l'array ed ad ogni iterazione viene
reimpostato ad i2
        while(c[i]==0) i++; //Scorre finché ci sono gli 0 (visto che sono da saltare)
        i2=i+1;
        while(c[i2]==0) i2++; //Scorre finché ci sono gli 0 (visto che sono da saltare)
        if ((c[i]+2)==c[i2]) { //Se il numero dato dalla posizione i, sommato di 2, è uguale a
quello presente nella posizione i2, allora è stata trovata una coppia di numeri primi gemelli
            fprintf(fp,"%lld Coppie di gemelli: %lld %lld\n",i3,c[i],c[i2]); //Viene stampata
            i3++; //E viene incrementato il contatore di coppie di numeri primi
        }
    }
    fclose(fp); //Chiudo il file
}

void createCandidate(int N) {
    long long int n2=N*50; //Moltiplico per 50 il numero di input, così da essere sicuro da
avere abbastanza spazio per tutti i valori di c
    //assgno spazio definito in modo arbitrario ai k, così da poter coprire tutti i casi di
input
    long long int *k1=calloc(100000000,sizeof(long long int));
    long long int *k2=calloc(100000000,sizeof(long long int));
    long long int *k3=calloc(100000000,sizeof(long long int));
    long long int *c1=calloc(n2,sizeof(long long int));
    long long int *c2=calloc(n2,sizeof(long long int));

    //generano tutti i k con le 3 regole di selezione
    for (long long int x=1, i=0;x<=10000;x++) {
        for (long long int y=1;y<=10000;y++,i++) {
            k1[i]=((6*x*y)+x-y);
            k2[i]=((6*x*y)-x-y);
            k3[i]=((6*x*y)+x+y);
        }
    }
    //genero i candidati primi
    for(long long int ki=1, i=0;ki<=n2; ki++, i++) {
        c1[i]=(6*ki)-1;
        c2[i]=(6*ki)+1;
    }
    //Richiamo alla funzione di rimozione dei numeri non primi per ogni regola di selezione
    removeNotPrimes(k1,c1,100000000,n2); //Rimuove i non primi dall'array formato dai
candidati della regola 6k-1 con la 1°regola
    removeNotPrimes(k2,c2,100000000,n2); //Rimuove i non primi dall'array formato dai
candidati della regola 6k+1 con la 2°regola
    removeNotPrimes(k3,c2,100000000,n2); //Rimuove i non primi dall'array formato dai
candidati della regola 6k+1 con la 3°regola
    long long int *a=merge(c1,c2,n2,n2); //Infine faccio un merge degli array di, con numeri
non primi rimossi, così da avere un unico array ordinato contenente tutti i numeri primi
    findGemelli(a,N); //Stampo il risultato su terminale
    //findAndSaveGemelli(a,N); //Se si vuole salvare su file si può usare questa funzione
}
```

```c
void createCandidateForK(int ki1, int kf) {
    long long int n2=50000;
    //assgno spazio definito in modo arbitrario ai k e a tutti i valori c, così da poter
coprire tutti i casi di input
    long long int *k1=calloc(100000000,sizeof(long long int));
    long long int *k2=calloc(100000000,sizeof(long long int));
    long long int *k3=calloc(100000000,sizeof(long long int));
    long long int *c1=calloc(n2,sizeof(long long int));
    long long int *c2=calloc(n2,sizeof(long long int));

    //generano tutti i k con le 3 regole di selezione
    for (long long int x=1, i=0;x<=10000;x++) {
        for (long long int y=1;y<=10000;y++,i++) {
            k1[i]=((6*x*y)+x-y);
            k2[i]=((6*x*y)-x-y);
            k3[i]=((6*x*y)+x+y);
        }
    }
    //genero i candidati primi
    for(long long int ki=1, i=0;ki<=n2; ki++, i++) {
        c1[i]=(6*ki)-1;
        c2[i]=(6*ki)+1;
    }
    removeNotPrimes(k1,c1,100000000,n2);
    removeNotPrimes(k2,c2,100000000,n2);
    removeNotPrimes(k3,c2,100000000,n2);
    long long int *a=merge(c1,c2,n2,n2);
    for(;((ki1)%6)!=0;ki1++);
    for(;((kf)%6)!=0;kf--);
    for(long long int i=0,i2=i+1,i3=1;a[i2]<kf;i=i2) {
        while(a[i]==0) i++;
        i2=i+1;
        while(a[i2]==0) i2++;
        if ((a[i]+2)==a[i2] && (a[i]+2)>ki1) {
            printf("%lld Coppie di gemelli: %lld %lld\n",i3,a[i],a[i2]);
            i3++;
        }
    }
}


/*
Questo programma ha 2 funzioni:
1°Calcola un numero n di coppie di numeri primi gemelli
2°Calcola tutte le coppie comprese tra x e y
*/
int main(int argc, char *argv[]) {
    if (argc<3 || argc>4) {
        fprintf(stderr,"Uso corretto:\n./main.exe 1 numero-coppie\n./main.exe 2 ki kf\n");
//Controlla il numero di argomenti, nel caso siano errati, stampa errore
        exit(EXIT_FAILURE); //E termina il programma
    }
    if (argv[1][0]=='1') createCandidate(atoi(argv[2])); //Richiamo alla 1°funzione
    else if(argv[1][0]=='2') createCandidateForK(atoi(argv[2]),atoi(argv[3])); //Richiamo alla
2°funzione
    else {
        fprintf(stderr,"Uso corretto:\n./main.exe 1 numero-coppie\n./main.exe 2 ki kf\n");
//Nel caso in cui gli argomenti siano errati, allora stampa errore
    }
}
```

Per ottimizzare il programma è stato necessario utilizzare tre matrici sparse, cioè matrici “fittizie” che non hanno dimensioni ben definite. All’interno delle matrici vengono generati, in base alla necessità di verificare se un numero è primo o no, le posizioni dei numeri che non sono primi basandosi sulle formule di esclusione illustrate in precedenza. Le matrici sono allocate dinamicamente senza inizializzazione per migliorare l'efficienza del codice, infatti, quando le matrici non vengono inizializzate non occupano spazio in memoria RAM. Le posizioni utili al controllo del numero vengono di volta in volta generate e salvate, riempendo porzioni di matrici. I numeri generati vengono salvati così da poter essere riutilizzati per i successivi controlli. Per ogni formula di esclusione viene creata una matrice sparsa: nella prima matrice sono sottoposti al controllo i numeri generati dalla prima formula C1, mentre alla seconda e terza matrice vengono sottoposti i numeri generati dalla seconda formula C2.

Il controllo avviene in questo modo:

Ipotizzare un candidato numero primo “n”. I confronti partono dal principio della matrice, dalla coordinata $x = 0$ e $y = 0$ che rappresentano rispettivamente le coordinate delle righe e delle colonne della matrice. Se il numero “n” in questione è maggiore del numero generato nella matrice in posizione iniziale, si passa al confronto del numero successivo sulla stessa riga (viene incrementato x di 1). Questo avviene fino a quando il numero “n” è maggiore del numero sulla riga, in quest’ultimo caso si effettua di nuovo il controllo dal primo numero della seconda riga della matrice (è incrementata la y di 1 e la x viene ripristinata a 0).

Questo algoritmo può concludersi soltanto in due modi:

1. il numero “n” è uguale ad un numero nella matrice, in questo caso sappiamo per certo che il numero non è primo.

2. Il numero “n” è minore di un numero all’interno della prima colonna della matrice (cioè in posizione x=j e y=0), il numero è quindi primo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 500000

//Questa funzione ha lo scopo di creare gli array (a 100 a 100) che corrispondono alle righe
void riallocazione(long long int** mat, int *n) {
    for (int i=0;i<100;i++) {
        mat[i]=malloc(N*sizeof(long int)); //Utilizzo la malloc, così da non sprecare memoria
(spiegazione alla fine del codice)*
    }
    (*n)+=100; //Incremento il numero di righe di 100, così da avere traccia di quante righe
vengono calcolate
}

/*
Questa funzione si occupa di calcolare il valore da inserire nella matrice a seconda di quale
delle 3 regole di selezione stiamo usando,
infatti lo swtich, a seconda se sia regola 1,2,3 cambia il calcolo.
Ed infine restituisce il valore calcolato
*/
long int calcolo(int regola, int x, int y) {
    int n=6*x*y;
    switch(regola) {
        case 1:
            return (n+x-y);
        case 2:
            return (n-x-y);
        case 3:
            return (n+x+y);
        default:
            return 0;
    }
}

int regola(long long int n, long long int *mat, int *righe, int regola) { //Questa è la
funzione di controllo per ogni valore del k, è uguale per ogni regola, cambiano solo i
parametri passati
    long long int x=1, y=1; //Crea le variabili di coordinate
    long long int k; //Crea il k che andrò di volta in volta a calcolare
    long long int *m=(long long int *)mat[0]; //E inizializza il puntatore alla 1°riga della
matrice
    while (1) {
        if ((long int *)m[y-1]==NULL) m[y-1]=calcolo(regola,x,y); //Se il valore della matrice
non esiste, allora viene calcolato e salvato nella matrice (se esiste già, semplicemente non lo
calcolo)
        k=m[y-1]; //E salva direttamente in k
        if (n==k) return 0;//Se trova il  k all'interno della matrice, allora è sicuramente
primo e restituisce 0
        else if (n<k && y==1) return 1;  //Altrimenti se è minore del valore di controllo e ci
troviamo sulla 1°colonna, allora restituisce 1, poiché è sicuramente primo
        else if (n<k) { //Altrimenti se è minore, ma non ci troviamo sulla 1°colonna
            y=1; //Si riposiziona sulla prima colonna
            x++; //Cambio riga
            if (x>(*righe)) { //Nel caso si superino il numero di righe già create
                riallocazione((long long int **)(mat+x-1),righe); //Ne vengono allocate altre
100
            }
            m=(long long int *)mat[x-1]; //Cambio di riga sul puntatore della matrice
        }
        else if (n>k) y++;  //Invece, se il valore dato è maggiore di k, allora scorre alla
posizione successiva sulla riga
    }
}

int isPrime(long long int n, int controllo, long long int *mat1, long long int *mat2, long long
int *mat3, int *righe1, int *righe2, int *righe3) { //Questa funzione va a richiamare la
```

```
funzione di controllo con i parametri corretti a seconda della regola che si vuole utilizzare
    if (controllo==1) { //Se è un valore k dei 6k-1, allora fa il controllo solo su regola 1
        return regola(n,mat1,righe1,1);
    }
    else if (controllo==2) { //Se è un valore k dei 6k+1, allora fa il controllo su regola 2 e
regola 3
        return (regola(n,mat2,righe2,2) && regola(n,mat3,righe3,3)); //Se tutte e due
restituiscono che è un numero primo, allora restituisce che è primo, altrimenti se una delle
due restituisce che il numero non è primo, allora restituisce che non è primo
    }
    return 1;
}

//Questa è la funzione che stampa della gestione del calcolo e della stampa dei numeri primi.
void createCandidate(long long int c1[], long long int n) {
    clock_t inizio, fine; //Crea della variabili per il tempo
    inizio = clock(); //Salva il tempo di inizio di esecuzione del programma
    FILE *fp;
    fp=fopen("output.txt","w"); //Creazione un file di output (è possibile anche cambiare e
stampare direttamente a terminale)
    long long int *mat1=malloc(N*sizeof(long long int *)); //Crea le 3 matrici partendo da un
array di puntatori (che corrispondono alle righe)
    long long int *mat2=malloc(N*sizeof(long long int *)); //Poi a cascata vengano
incrementati, quando necessari, di volta in volta di 100
    long long int *mat3=malloc(N*sizeof(long long int *)); //Sempre con la malloc, così da non
occupare memoria inutilmente (spiegazione sotto)
    int righe1=0;
    int righe2=0; //Utilizza delle variabili intere che tengono traccia di quante righe sono
state effettivamente create (così da evitare di andare fuori memoria ed avere un segmentation
fault)
    int righe3=0;
    riallocazione((long long int **)mat1,&righe1);
    riallocazione((long long int **)mat2,&righe2); //Vengono create le prime 100 righe di ogni
matrice
    riallocazione((long long int **)mat3,&righe3);
    for(long long int ki=1, flag, i1=0, i2=1;i2<=n; ki++) { //Ora inizia il calcolo effettivo
dei numeri primi, mi fermo quando il numero di coppie trovate arriva a quello richiesto
dall'utente
        flag=0; //Questa variabile controlla se è stato trovato almeno un numero primo dai 6k-
1 e dai 6k+1
        if(isPrime(ki,1,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Controlla che il k dato
corrisponde ad un numero primo per la regola dei 6k-1
            flag=1; //Se si, imposta flag a 1
            c1[i1]=(6*ki)-1; //Calcola l'effettivo valore
            i1++; //Aumenta il numero di numeri primi trovati
        }
        if(isPrime(ki,2,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Stesso discorso per
quanto riguarda i 6k+1
            flag=1;
            c1[i1]=(6*ki)+1;
            i1++;
        }
        if (flag && (c1[i1-2]+2)==c1[i1-1]) { //Ora, se flag è stato attivato e se nei numeri
primi c'è una coppia di numeri primi gemelli
            fine = clock(); //Fermo il tempo al calcolo
            fprintf(fp,"%lld COPPIA: %lld %lld  T: %f\n",i2,c1[i1-2],c1[i1-1],(((double) (fine
- inizio)) / CLOCKS_PER_SEC)); //E stampa tutto (o su file o a terminale)
            //printf("%lld COPPIA: %lld %lld  T: %f\n",i2,c1[i1-2],c1[i1-1],(((double) (fine -
inizio)) / CLOCKS_PER_SEC));
            i2++; //Incrementa il numero di coppie trovate
        }
    }
}

void createCandidateForK(long long int c1[], int ki, int kf) {
    clock_t inizio, fine; //Crea della variabili di tempo
    inizio = clock();  //Salva lora di inizio di esecuzione del programma
```

```
    FILE *fp;
    fp=fopen("output.txt","w"); //Creazione del file di output (è possibile anche cambiare e
stampare direttamente a terminale)
    long long int *mat1=malloc(N*sizeof(long long int *)); //Crea le 3 matrici, esse sono
create partendo da un array di puntatori (che corrispondono alle righe)
    long long int *mat2=malloc(N*sizeof(long long int *)); //Poi a cascata vengano
incrementati, quando necessari, di volta in volta di 100
    long long int *mat3=malloc(N*sizeof(long long int *)); //Sempre con la malloc, così da non
occupare memoria inutilmente (spiegazione sotto)
    int righe1=0;
    int righe2=0; //Utilizza delle variabili intere che mi tengono traccia di quante righe sono
state effettivamente create (così da evitare di andare fuori memoria ed avere un segmentation
fault)
    int righe3=0;
    riallocazione((long long int **)mat1,&righe1);
    riallocazione((long long int **)mat2,&righe2); //Vengono create le prime 100 righe di ogni
matrice
    riallocazione((long long int **)mat3,&righe3);
    for(;((ki)%6)!=0;ki++); //Calcola i k corrispondenti ai numeri dati facendo la formula
inversa
    for(;((kf)%6)!=0;kf--);
    ki/=6;
    kf/=6;
    for(long long int i1=0, i2=0, flag;ki<=kf; ki++) { //Inizio il calcolo effettivo
        flag=0; //Questa variabile controlla se è stato trovato almeno un numero primo dai 6k-
1 e dai 6k+1
        if(isPrime(ki,1,mat1,mat2,mat3,&righe1,&righe2,&righe3)) {  //Controllo che il k dato
corrisponde ad un numero primo per la regola dei 6k-1
            flag=1; //Se si, imposto flag a 1
            c1[i1]=(6*ki)-1; //Mi calcolo l'effettivo valore, nel caso in cui sia un numero
primo
            i1++; //Aumento il numero di numeri primi trovati
        }
        if(isPrime(ki,2,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Stesso discorso per i
6k+1
            flag=1;
            c1[i1]=(6*ki)+1;
            i1++;
        }
        if (flag && (c1[i1-2]+2)==c1[i1-1]) { //Ora, se flag è stato attivato e se nei numeri
primi c'è una coppia di numeri primi gemelli
            fine = clock(); //Ferma il calcolo del tempo
            //fprintf(fp,"%lld Coppie di gemelli: %lld %lld\n",i2+1,c1[i1-2],c1[i1-1]); //E
stampo tutto (o su file o a terminale)
            printf("%lld Coppie di gemelli: %lld %lld\n",i2+1,c1[i1-2],c1[i1-1]);
            i2++; //Incrementa il numero di coppie trovate
        }
    }
}

/*
Il programma offre 2 opzioni:
1° Calcola un numero n di coppie di numeri primi gemelli
2° Calcola le coppie di numeri primi gemelli con valore compreso tra ki e kf
*/
int main(int argc, char *argv[]) {
    if (argc<3 || argc>4) {
        fprintf(stderr,"Uso corretto:\n./main.exe 1 numero-coppie\n./main.exe 2 ki kf\n");
//Controllo sugli argomenti e stampa dell'errore nel caso siano errati
        exit(EXIT_FAILURE);
    }
    long long int *c1=calloc(1000000,sizeof(long long int)); //Allocazione dell'array dove
verranno messi i risultati
    if (argv[1][0]=='1') createCandidate(c1,atoi(argv[2])); //1°funzione
    else if(argv[1][0]=='2') createCandidateForK(c1,atoi(argv[2]),atoi(argv[3])); //2°Funzione
    else {
        fprintf(stderr,"Uso corretto:\n./main.exe 1 numero-coppie\n./main.exe 2 ki kf\n");
```

```
//Stampa dell'errore in caso di argomenti errati
    }
}

/*
*Utilizzo la malloc con la matrice, sia per le righe che per le colonne, perché la malloc non
occupa memoria
all'interno della RAM finché non viene inizializzato il valore, quindi sfruttando quest'opzione
vado ad occupare soltanto la memoria necessaria ai calcoli e non occupo la memoria non
necesssaria ai conti con degli 0.
Per controllare se un valore è inizializzato o meno, controllo se esso equivale a NULL, se lo è
vuol dire che non è stato inizializzato
e che occorre calcolarlo, altrimenti viene ricavato il valore di quella locazione.
*/
```

L'ultima versione implementata utilizza un algoritmo diverso ma si basa sulle matrici sparse già implementate precedentemente. I confronti in questo caso verranno effettuati sulla diagonale della matrice.

Ipotizzare ancora una volta un numero "n". Anche in questo caso i confronti partiranno dal principio della matrice $x = 0$, $y = 0$. Se il numero è maggiore del numero in quella posizione saranno incrementate le coordinate in modo da raggiungere il secondo numero della diagonale (x e y sono incrementate entrambe di 1).

Questo avviene fino a quando il numero sulla diagonale non è maggiore del numero “n” inizialmente ipotizzato. Questo metodo divide la matrice in due triangoli, triangolo superiore e triangolo inferiore, rendendo più semplice ed efficace la ricerca del numero. Una volta raggiunto un numero maggiore di “n” sulla diagonale avverranno in parallelo due operazioni: si confronterà “n” con il numero nella riga superiore della stessa colonna (coordinata x-1, finendo nel triangolo superiore della matrice) e quello precedente sulla stessa riga (coordinata y-1, finendo nel triangolo inferiore della matrice). La variazione delle coordinate anche in quest’ultimo caso dipende dal numero in questione generato dalla formula di esclusione e dalla porzione di matrice in cui si stanno effettuando le operazioni. Nel triangolo superiore se il numero “n” è maggiore del numero nella matrice si scorre al numero successivo sulla stessa riga ($x + 1$), mentre, se è minore si passa alla riga superiore ($y - 1$).

Nel triangolo inferiore le operazioni avvengono in modo specchiato. Da questo momento le coordinate varieranno fino a quando:

1. In una riga o colonna confrontata c’è il numero n, anche in questo caso il numero non è primo.

2. Viene raggiunto il limite della matrice e occorre decrementare / incrementare per raggiungere una posizione inesistente della matrice, il numero è quindi primo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define N 500000
int k; //Crea un'unica variabile k globale, così non doverla inizializzare e allocare ad ogni funzione, ma semplicemente poter cambiare il valore

//Questa funzione ha lo scopo di creare gli array incrementandoli di 100 in base alla necessità(corrispondono alle righe della matrice)
void riallocazione(long long int* mat, int *n) {
    for (int i=0;i<100;i++) {
        mat[i]=(long long int)malloc(N*sizeof(long long int)); //Utilizzo la malloc, così da non sprecare memoria (spiegazione sotto)
    }
    (*n)+=100;//Incremento il numero di righe di 100, così da avere traccia di quante righe vengono calcolato
}

/*
Questa funzione si occupa di calcolare il valore da inserire nella matrice a seconda di quale delle 3 regole di selezione stiamo usando,
infatti lo swtich, a seconda se sia regola 1,2,3 cambia il calcolo.
Ed infine restituisce il valore calcolato
*/
long int calcolo(int regola, int x, int y) {
    int n=6*x*y;
    switch(regola) {
        case 1:
            return (n+x-y);
        case 2:
            return (n-x-y);
        case 3:
            return (n+x+y);
        default:
            return 0;
    }
}

int controlloLeft(long int n, long long int *mat, int *righe, int regola, int x, int y) {
//Questa funzione si occupa di fare il controllo (se il numero k dato è un numero primo o meno) sul lato sinistro della matrice
    long long int *m=(long long int *)mat[(x-1)]; //Puntatore che punta alla riga data dalla funzione regola, così da posizionarsi sulla riga da dove occorre iniziare il controllo, in questo caso non avviene nessun controllo sulle righe perchè gia avvenuto nella prima regola
    while(1) {
        if ((long long int *)m[y-1]==NULL) m[y-1]=calcolo(regola,x,y); //Se il valore della matrice non esiste, allora viene calcolato e inserito nella matrice (se esiste già, non viene calcolato)
        k=m[y-1]; //Salva direttamente in k
        if (n==k) return 0; //Se viene trovato il k all'interno della matrice, allora il numero non è primo
        else if(n<k) { //Se n è minore del k calcolato
            y--; //Decrementa la colonna
            if(y<=0) return 1; //Nel caso in cui y tenta di accedere ad una posizione dopo il limite dellaa matrice, allora il numero è sicuramente primo e restituisco 1
        }
        else if (n>k) { //Al contrario, se il numero che controllo è maggiore del k, allora:
            x++; //Increnenta la riga
            if (x>(*righe)) { //Nel caso si superino il numero di righe già create
                riallocazione(mat+x-1,righe); //Ne vengono allocate altre 100
                }
            }
        m=(long long int *)mat[x-1]; //Cambio di riga sul puntatore della matrice
    }
}
```

```c
int controlloRight(long int n, long long int *mat, int *righe, int regola, int x, int y) {
//Questa funzione si occupa di fare il controllo(se il numero k dato è un numero primo o meno) sul lato destro della matrice
    long long int *m=(long long int *)mat[(x-1)]; //Puntatore che punta alla riga data dalla funzione regola, così da posizionarsi sulla riga da dove occorre iniziare il controllo, in questo caso non avviene nessun controllo sulle righe perchè gia avvenuto nella prima regola
    while(1) {
        if ((long long int *)m[y-1]==NULL) m[y-1]=calcolo(regola,x,y); //Se il valore della matrice non esiste viene calcolato e salvato nella matrice (se esiste già, semplicemente non viene calcolato)
        k=m[y-1]; //Salva direttamente in k
        if(n<k && x==1) return 1; //Nel caso in cui ci si trova sulla 1°riga e il numero che viene controllato è minore del k, allora vuol dire che il nostro numero è sicuramente primo
        else if (n>k) y++; //Nel caso in cui il numero sia maggiore di k, allora incrementa di colonna per scorrere in avanti
        else if(n<k && x!=1) { //Invece, nel caso in cui il numero è minore di k e non ci troviamo sulla 1°riga
            x--; //Spostamento di una riga più in alto
            m=(long long int *)mat[x-1]; //Viene aggiornato il puntatore della matrice
        }
        else if (n==k) return 0; //Se viene trovato il k all'interno della matrice, allora il numero non è primo
    }
}

int regola(long int n, long long int *mat, int *righe, int regola) { //Questa è la funzione che va a controllare in quale punto della diagonale si deve andare a cercare se è il numero è primo o meno
    int x=1, y=1; //Inizializza le variabili di coordinate
    long long int *m=(long long int *)mat[0]; //E viene inizializzato il puntatore alla 1°riga della matrice
    do {
        if ((long int *)m[y-1]==NULL) m[y-1]=calcolo(regola,x,y); //Se il valore della matrice non esiste, allora viene calcolato e salvato nella matrice (se esiste già, semplicemente non lo calcolo)
        k=m[y-1]; //Salva direttamente in k
        if (n==k) return 0; //Se viene trovato il  k all'interno della matrice, allora è sicuramente non?????? è primo e restituisco 0???????????
        x++; //Nel caso non si trova, cambio di riga
        if (x>(*righe)) { //Nel caso in cui non ci siano più righe allocate, ne vengono allocate altre
            riallocazione(mat+x-1,righe);
        }
        m=(long long int *)mat[x-1]; //Punto alla prossima riga
        y++; //Incremento anche la colonna(visto che dobbiamo scorrere sulla diagonale, vengono incrementati x e y contemporaneamente di 1 a 1)
    } while(n>k); //Il ciclo si ripete finché il k calcolato non è minore rispetto alla n che stiamo cercando
    return (controlloLeft(n,mat,righe,regola,x-1,y-1) && controlloRight(n,mat,righe,regola,x-1,y-1)); //Infine si effettua il controllo a sinistra e a destra della matrice, infatti se uno dei due restituisce che non è primo, allora non sarà primo, se invece tutte e due restituiscono che è primo, allora è sicuramente primo e restituisco che è primo
}

int isPrime(long int n, int controllo, long long int *mat1, long long int *mat2, long long int *mat3, int *righe1, int *righe2, int *righe3) { //Questa funzione va a richiamare la funzione di controllo con i parametri corretti a seconda della regola che si vuole utilizzare
    if (controllo==1) return regola(n,mat1,righe1,1); //Se è un valore k dei 6k-1, allora fa il controllo solo su regola 1
    else return (regola(n,mat2,righe2,2) && regola(n,mat3,righe3,3)); //Se è un valore k dei 6k+1, allora fa il controllo su regola 2 e regola 3
    //Se tutte e due restituiscono che è un numero primo, allora restituisce che è primo, altrimenti se una delle due restituisce che il numero non è primo, allora restituisce che non è primo
}
```

```
void createCandidate(long int n) {
    clock_t inizio, fine; //Crea della variabili di tempo
    inizio = clock(); //Salva il tempo di inizio di esecuzione del programma
    FILE *fp;
    fp=fopen("output2.txt","w"); //Creazione del file di output (è possibile anche cambiare e
stampare direttamente a terminale)
    long long int *mat1=malloc(N*sizeof(long long int *)); //Crea le 3 matrici, esse sono
create partendo da un array di puntatori (che corrispondono alle righe)
    long long int *mat2=malloc(N*sizeof(long long int *)); ////Vengono create le prime 100
righe di ogni matrice
    long long int *mat3=malloc(N*sizeof(long long int *)); //Sempre con la malloc, così da non
occupare memoria inutilmente (spiegazione sotto)
    int righe1=0;
    int righe2=0; //Inizializza delle variabili intere che tengono traccia di quante righe sono
state effettivamente create (così da evitare di andare fuori memoria ed avere un segmentation
fault)
    int righe3=0;
    riallocazione(mat1,&righe1);
    riallocazione(mat2,&righe2); //Vengono create le prime 100 righe di ogni matrice
    riallocazione(mat3,&righe3);
    long int c1[2]={0}; //Questo è un array con solo 2 posizioni, dove verrano salvate di
volta in volta le coppie
    for(int ki=1, i1=0, i2=1;i2<=n; ki++,i1=0) { //Inizio il calcolo effettivo (in questo caso
calcola finché il numero di coppie trovate non corrisponde a quello richiesto dall'utente)
        if(isPrime(ki,1,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Se il k, preso in esame
(nei 6k-1), corrisponde ad un numero primo
            c1[i1]=(6*ki)-1; //Allora ci calcoliamo il valore corrispondente
            i1++; //E incrementiamo il numero di numeri primi trovati
        }
        if(isPrime(ki,2,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Stesso discorso per i 6k-
1
            c1[i1]=(6*ki)+1;
            i1++;
        }
        if (i1!=0 && (c1[0]+2)==c1[1]) { //Nel caso in cui è stato trovato almeno un numero
primo e se i 2 numeri sono una coppia, allora:
            fine = clock(); //Ferma il conteggio del tempo
            fprintf(fp,"%d COPPIA: %ld %ld  T: %f\n",i2,c1[0],c1[1],(((double) (fine -
inizio)) / CLOCKS_PER_SEC)); //E stampo (su file o su terminale)
            //printf("%d COPPIA: %ld %ld  T: %f\n",i2,c1[0],c1[1],(((double) (fine - inizio))
/ CLOCKS_PER_SEC));
            i2++; //Incrementa il numero di coppie trovate
        }
    }
}

void createCandidateForK(int ki, int kf) {
    clock_t inizio, fine; //Crea della variabili di tempo
    inizio = clock(); //Salva l'orario di inizio esecuzione
    //FILE *fp;
    //fp=fopen("output2.txt","w");
    long long int *mat1=malloc(N*sizeof(long long int *)); //Crea le 3 matrici, esse sono
create partendo da un array di puntatori (che corrispondono alle righe)
    long long int *mat2=malloc(N*sizeof(long long int *)); ////Vengono create le prime 100
righe di ogni matrice
    long long int *mat3=malloc(N*sizeof(long long int *)); //Sempre con la malloc, così da non
occupare memoria inutilmente (spiegazione sotto)
    int righe1=0;
    int righe2=0; //Crea delle variabili intere che mi tengono traccia di quante righe sono
state effettivamente create (così da evitare di andare fuori memoria ed avere un segmentation
fault)
    int righe3=0;
    riallocazione(mat1,&righe1);
    riallocazione(mat2,&righe2); //Vengono create le prime 100 righe di ogni matrice
    riallocazione(mat3,&righe3);
    long int c1[2]={0}; //Questo è un array con solo 2 posizioni, dove verrano salvate passo
passo le coppie
```

```
    for(;((ki)%6)!=0;ki++); //Calcola i k corrispondenti ai numeri dati facendo la formula
inversa
    for(;((kf)%6)!=0;kf--);
    ki/=6;
    kf/=6;
    for(int i1=0, i2=1;ki<=kf; ki++,i1=0) { //Inizia il calcolo effettivo (in questo caso però
k partirà da ki e il ciclo finirà quando ki supererà kf)
        if(isPrime(ki,1,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Controlla  che il k dato
corrisponde ad un numero primo per la regola dei 6k-1
            c1[i1]=(6*ki)-1; //Calcola l'effettivo valore, nel caso in cui sia un numero primo
            i1++; //Incrementa il numero dei numeri primi trovati
        }
        if(isPrime(ki,2,mat1,mat2,mat3,&righe1,&righe2,&righe3)) { //Stesso discorso per i
6k+1
            c1[i1]=(6*ki)+1;
            i1++;
        }
        if (i1!=0 && (c1[0]+2)==c1[1]) { //Nel caso in cui è stato trovato almeno un numero
primo e se i 2 numeri sono una coppia, allora:
            fine = clock(); //Viene salvato l'orario di fine esecuzione
            //fprintf(fp,"%d COPPIA: %ld %ld  T: %f\n",i2,c1[0],c1[1],(((double) (fine -
inizio)) / CLOCKS_PER_SEC));
            printf("%d COPPIA: %ld %ld  T: %f\n",i2,c1[0],c1[1],(((double) (fine - inizio)) /
CLOCKS_PER_SEC)); //E stampo (o su file o su terminale)
            i2++; //Incrementa il numero di coppie trovate
        }
    }
}

/*
Il programma offre 2 opzioni:
1° Calcola un numero n di coppie di numeri primi gemelli
2° Calcola le coppie di numeri primi gemelli con valore compreso tra ki e kf
*/
int main(int argc, char *argv[]) {
    if (argc<3 || argc>4) {
        fprintf(stderr,"Uso corretto:\n./main.exe 1 numero-coppie\n./main.exe 2 ki kf\n");
//Controllo sugli argomenti e stampa dell'errore nel caso siano errati
        exit(EXIT_FAILURE);
    }
    if (argv[1][0]=='1') createCandidate(atoi(argv[2])); //1°Funzione
    else if(argv[1][0]=='2') createCandidateForK(atoi(argv[2]),atoi(argv[3])); //2°Funzione
    else {
        fprintf(stderr,"Uso corretto:\n./main.exe 1 numero-coppie\n./main.exe 2 ki kf\n");
//Stampa dell'errore in caso di argomenti errati
    }
}

/*
Utilizzo la malloc con la matrice, sia per le righe che per le colonne, perché la malloc non
occupa memoria
all'interno della RAM finché non viene inizializzato il valore, quindi sfruttando quest'opzione
vado ad occupare soltanto la memoria necessaria ai calcoli e non occupo la memoria non
necesssaria ai conti con degli 0.
Per controllare se un valore è inizializzato o meno, controllo se esso equivale a NULL, se lo è
vuol dire che non è stato inizializzato
e che devo calcolarlo, altrimenti viene ricavato il valore da quella locazione.
*/
```

L'ultimo algoritmo è diverso dai precedenti, infatti, si è abbandonato l'uso delle regole della selezione e delle matrici, adottando un approccio basato sulle famiglie. Questo nuovo approccio si articola in due liste distinte: una per i numeri primi l'altro per i non primi.

Numeri primi

La lista per i numeri primi inizia con un singolo elemento, il numero 2, che costituisce il punto di partenza per tutti i calcoli successivi.

Denotando il numero primo successivo come "m", applichiamo due formule di selezione:

1: $mk + 1$

2: $mk - 1$

Queste due formule non solo ci permettono di determinare i numeri primi successivi al 2, ma ci permettono anche di stabilire il limite di calcolo per la famiglia successiva. Ad esempio, per la seconda famiglia, "m" sarà uguale a 6 (dato dalla moltiplicazione dei primi 2 numeri primi, 2 e 3). Il limite della prima famiglia è quindi 6. Proseguendo in questo modo, il limite della terza famiglia sarà 30 (dato dalla moltiplicazione dei primi 3 numeri primi 2, 3 e 5), e così via. Inoltre, quando l'algoritmo determina un nuovo numero, entra in scena una funzione scritta appositamente per controllare se il numero è primo o meno.

Una volta effettuato questo controllo, si avviano ulteriori calcoli in cui si somma il prodotto di "m" e "k" ai numeri primi precedenti a "m".

Numeri non primi

Per quanto riguarda i numeri non primi, il processo è analogo a quello dei numeri primi. L'unica differenza è che in questo caso, al prodotto di “m” e “k2 si sommano i numeri non primi.

Questo algoritmo rappresenta un passo avanti significativo nella computazione dei numeri primi e non primi, offrendo un nuovo approccio basato sulle famiglie di numeri. Con il suo aiuto, possiamo determinare con precisione i numeri primi e non primi, stabilendo al contempo i limiti di calcolo per le famiglie successive. Questo metodo offre un equilibrio ottimale tra efficienza computazionale e precisione dei risultati.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 5000000
int i1=1, i2=1;



//Funzione che replica la funzione atoi, creata così da evitare il linkaggio con la string.h e
migliorare l'efficienza
int myAtoi(char *s) {
    int n=0;
    while(*s) {
        if(*s<'0' || *s>'9') {
            fprintf(stderr,"Argomenti non corretti\n");
            exit(EXIT_FAILURE);
        }
        n=(n*10)+(*s-'0');
        s++;
    }
    return n;
}

//Funzione che stampa l'intero array su terminale
void printArray(int *a, int n) {
    for(int i=0;i<n;i++) printf("%d ",a[i]);
    printf("\n");
}

void printFileArray(int *a, int n) {
    for(int i=0;i<n;i++) printf("%d ",a[i]);
    printf("\n");
}

/*

Abbiamo scelto di utilizzare la funzione di ricerca binaria su un array perché,
per array di grandi dimensioni come i nostri, la ricerca lineare risultava estremamente
inefficiente,
rallentando significativamente il programma.
*/

int binary_search(int *arr, int low, int high, int value) {
    if(value > arr[high - 1]) return high; // Se il valore cercato è maggiore dell'ultimo
elemento dell'array, ritorna high
    while (low < high) { // Ciclo while per continuare la ricerca finché 'low' è minore di
'high'
        // Calcola il punto medio dell'intervallo corrente
        int mid = low + (high - low) / 2;
        if (arr[mid] == value) {
            return mid;         // Se l'elemento a 'mid' è min ore del valore cercato,
restringi la ricerca alla metà superiore
        } else if (arr[mid] < value) {
            low = mid + 1;

        // Se l'elemento a mid è maggiore del valore cercato, restringi la ricerca alla metà
inferiore
        } else {
            high = mid;
        }
    }
    // Se il ciclo termina senza trovare il valore, ritorna la posizione 'low'
    return low;
}


/*
```

```c
/*
inserisci_ordinato aggiunge gli elementi secondo una relazione d'ordine.
Volevamo che gli elementi fossero inseriti secondo una precisa relazione d'ordine,
in modo da poterli analizzare in maniera più efficiente.
*/

void inserisci_ordinato(int *arr, int *size, int value) {
    // Trova la posizione corretta per 'value' utilizzando la ricerca binaria
    int i = binary_search(arr, 0, *size, value);
    for (int j = *size; j > i; j--) {// Sposta tutti gli elementi a destra di 'i' di una
posizione per fare spazio al nuovo valore
        arr[j] = arr[j - 1];
    }
    arr[i] = value; // Inserisce 'value' nella posizione corretta trovata
    (*size)++;// Incrementa la dimensione dell'array
}


/*
La funzione `controlloPrime` verifica se un numero n è primo utilizzando un array preesistente
di numeri primi.
Prima controlla se n è 1 o pari (escluso 2), restituendo 0 in tal caso poiché tali numeri non
sono primi.
Successivamente, itera sull'array primes confrontando gli elementi dall'inizio e dalla fine
verso il centro.
Se n è divisibile per uno degli elementi dell'array, restituisce 0, indicando che n non è
primo.
Se nessuna divisione ha resto zero, la funzione restituisce 1, confermando che n è primo.
*/

int controlloPrime(int *primes, int n) {
    // Controlla se n è uguale a 1 o se è pari, in tal caso non è primo
    if (n == 1 || (n % 2) == 0) return 0;
    // Itera attraverso gli elementi dell'array primes
    for (int i = 0, iFinal = i1 - 1; i < iFinal; i++, iFinal--) {
        // Controlla se il numero è divisibile per uno degli elementi dell'array
        if ((n % primes[i]) == 0 || (n % primes[iFinal]) == 0) return 0;
    }
    // Se il numero supera i controlli precedenti, è considerato primo
    return 1;
}


/*
La funzione ifNotExist controlla atraverso una ricerca binaria se un certo target all'interno
dell'array
esiste oppure no.
*/

int ifNotExist(int *arr, int n, int target) {
    //guarda la ricerca binaria
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return 0;
        } else if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return 1;
}

/*
La funzione calcolaFamiglie genera numeri primi e non primi a partire da un valore di base m,
espandendo progressivamente utilizzando il metodo del Multiscala: abbiamo due array, uno per i
```

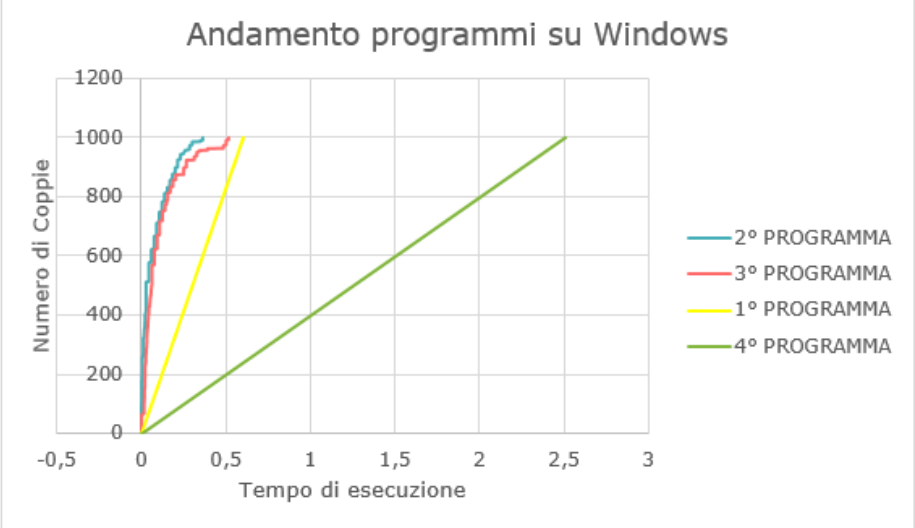
```c
numeri primi
primes e uno per i non primi notprimes. La funzione utilizza diverse variabili per controllare
i limiti di iterazione e per determinare se un numero è primo o meno, aggiungendo i numeri
trovati
agli array appropriati.
*/

int calcolaFamiglie(int *primes, int *notprimes, int m, int *foundPrimes, int n, int i, FILE
*fp, clock_t inizio) {
    int limit=m, first=1, a, b, k; //Imposta limite ad m e first viene iniziallizato ad 1
    clock_t fine;
    for(a=0,b=0,k=1,n=m*k;(a<=limit && b<=limit);k++,n=m*k) { // il ciclo termina quando o a
o b o entrambi superano il limite
        b=n-1; //In n si trova m*k e con esso calcola sia a che b
        if(controlloPrime(primes,b)) { //Controlla che b sia un numero primo
            inserisci_ordinato(primes,&i1,b); //Nel caso lo sia, viene inserito in primes in
modo ordinato
            if(ifNotExist(notprimes,i2,b)) inserisci_ordinato(notprimes,&i2,b); //E nel caso
non esista in notprimes, allora viene aggiunto anche lì (visto che i controlli li facciamo in
modo univoco con un unico array)
            (*foundPrimes)++; //Il numero di numeri primi trovati viene incrementato
            fine=clock();
            fprintf(fp,"Numero primo: %d T: %f\n",*foundPrimes,(((double) (fine - inizio)) /
CLOCKS_PER_SEC)); //Stampa su file il tempo
        }
        a=n+1;
        if(controlloPrime(primes,a)) { //Controlla che a sia un numero primo
            inserisci_ordinato(primes,&i1,a); //Nel caso lo sia, viene inserito in primes in
modo ordinato
            (*foundPrimes)++; //Il numero di numeri primi trovati viene incrementato
            fine=clock();
            fprintf(fp,"Numero primo: %d T: %f\n",*foundPrimes,(((double) (fine - inizio)) /
CLOCKS_PER_SEC)); //Stampa su file il tempo
        }
        if(ifNotExist(notprimes,i2,a)) inserisci_ordinato(notprimes,&i2,a);
        if(first) { //Il limite viene moltplicato col numero primo successivo, così che
troviamo l'effettivo limite
            limit*=primes[i];
            first=0; //first viene riportato a 0, visto che oramai il limite è stato calcolato
        }
     }
    for (; notprimes[i] < m && i < i2; i++) {
        k = 1;
        do {
            a = (m * k) + notprimes[i]; //Ora viene effettuato il calcolo per ogni numero
trovato in notprimes, così da trovare altri possibili numeri primi seguendo la formula m*k+n
            if (controlloPrime(primes, a)) { //Controlla che il numero trovato sia prima
                if (ifNotExist(primes, i1, a)) { //Se si e non si trova in primes
                    inserisci_ordinato(primes, &i1, a); //Allora viene aggiunto
                    (*foundPrimes)++; //E il numero di numeri primi trovati ivene incrementato
                    fine=clock();
                    fprintf(fp,"Numero primo: %d T: %f\n",*foundPrimes,(((double) (fine -
inizio)) / CLOCKS_PER_SEC)); //Stampa su file il tempo
                }
            }
            if (ifNotExist(notprimes, i2, a)) { //Se non si trova in notprimes
                inserisci_ordinato(notprimes, &i2, a); //Allora viene aggiunto anche li
            }
            k++; //Incremento k
        }while (a < limit); //Fa questo calcolo per ogni numero di notprimes e finché quel
numero non supera il limite
    }
    return limit; //Ritorno il limite, così che esso venga salvato in m e venga utilizzato come
prossimo punto di calcolo
}

/*
```

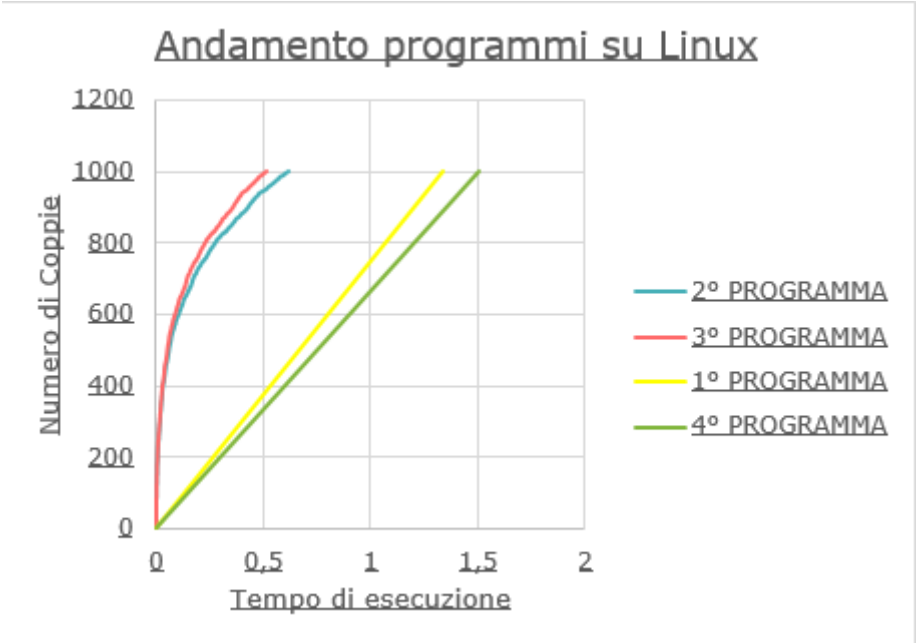
```c
La funzione findPrimes, per prima cosa alloca lo spazio di memoria per i due array,
e passa i valori ad calcolaFamiglie.
*/

void findPrimes(int n) {
    FILE *fp;
    fp=fopen("output.txt","w");
    clock_t inizio, fine;
    size_t size=N*sizeof(int);
    int *primes=malloc(size); //Array dove verrano salvati i numeri primi
    int *notprimes=malloc(size); //Array dove verrano salvati i numeri NON primi
    notprimes[0]=4;
    primes[0]=2; //Inizializza solo un valore
    int foundPrimes=1, m=2, i=1;
    while(foundPrimes<n) { //Cicla finché il numero di numeri primi trovati non supera quello
richiesto
        m=calcolaFamiglie(primes,notprimes,m,&foundPrimes,n,i,fp,inizio);
        i++;
    }
    //printArray(primes,n); //Stampo tutto
}

/*
La funzione findCoppie trova e stampa n coppie di numeri primi gemelli.
Alloca memoria per due array, primes e notprimes, inizializzandoli rispettivamente con 2 e 4.
Utilizza un ciclo while per riempire questi array chiamando la funzione calcolaFamiglie,
finché non vengono trovati almeno 5n numeri primi. Successivamente, itera attraverso l'array
primes
per identificare coppie di numeri primi gemelli, ovvero coppie di primi che differiscono di 2,
stampandole.
Infine, libera la memoria allocata per gli array.

*/

void findCoppie(int n) {
    FILE *fp;
    fp=fopen("output.txt","w");
    clock_t inizio, fine;
    size_t size=N*sizeof(int);
    int *primes=malloc(size); //Array dove verrano salvati i numeri primi
    int *notprimes=malloc(size); //Array dove verrano salvati i numeri NON primi
    notprimes[0]=4;
    primes[0]=2; //Inizializza solo un valore
    int foundPrimes=1, m=2, i=1;
    while(foundPrimes<(n*5)) {
        m=calcolaFamiglie(primes,notprimes,m,&foundPrimes,n,i,fp,inizio); //Avvia il calcolo,
calcolando il tanti numeri primi quanto il quintuplo del numero di coppie richieste
        i++;
    }
    for(int i1=0, i2, i3=1;i3<=n;i1=i2) { //i1 rappresenta il 1°numero, i2 il 2°numero e i3 il
conto delle coppie trovate
        i2=i1+1; //i2 viene spostato al successivo di i1
        if((primes[i1]+2)==primes[i2]) { //Se l'elemento in i1, sommato di 2, è uguale
all'elemento in i2, allora ha trovato una coppia di numeri primi gemelli
            printf("%d COPPIA: %d %d\n",i3,primes[i1],primes[i2]); //Stampa
            i3++; //Incrementa il numero di coppie trovate
        } //i1 viene spostato ad i2
    }
}


/*
Questo programma ha 2 funzioni:
1°Calcolo dei primi n numeri primi, n dato dall'utente da temrinale
2°Calcolo delle prime n coppie di numeri primi gemelli, n sempre dato dall'utente da terminale
*/
int main(int argc, char *argv[]) {
```

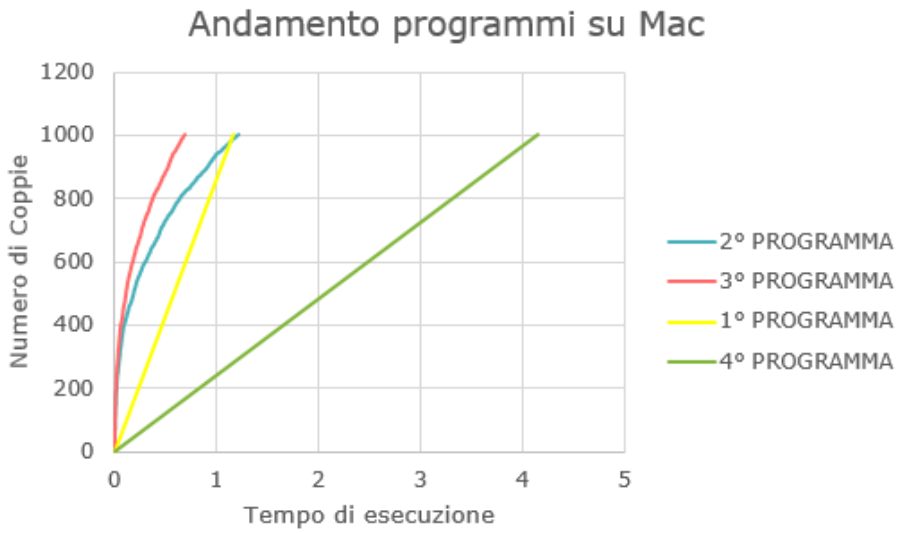
```
    if(argc!=3) { //Controlla che il numero degli argomenti sia corretto
        fprintf(stderr,"Uso corretto:\n./a.exe 1 (n numeri primi)\n./a.exe 2 (n coppie
gemelle)\n"); //In caso di numero errato, stampa l'errore e termina
        exit(EXIT_FAILURE);
    }
    else {
        switch(argv[1][0]) { //Controlla quale funzione devo eseguire
            case '1':
                findPrimes(myAtoi(argv[2])); //Calcolo dei primi n numeri primi
                break;
            case '2':
                findCoppie(myAtoi(argv[2])); //Calcolo delle prime n coppie di numeri primi
gemelli
                break;
            default:
                fprintf(stderr,"Uso corretto:\n./a.exe 1 (n numeri primi)\n./a.exe 2 (n coppie
gemelle)\n"); //In caso la funzione non corrispondi, stampa l'errore etermina
                exit(EXIT_FAILURE);
        }
    }
    return 0;
}
```

Risultati

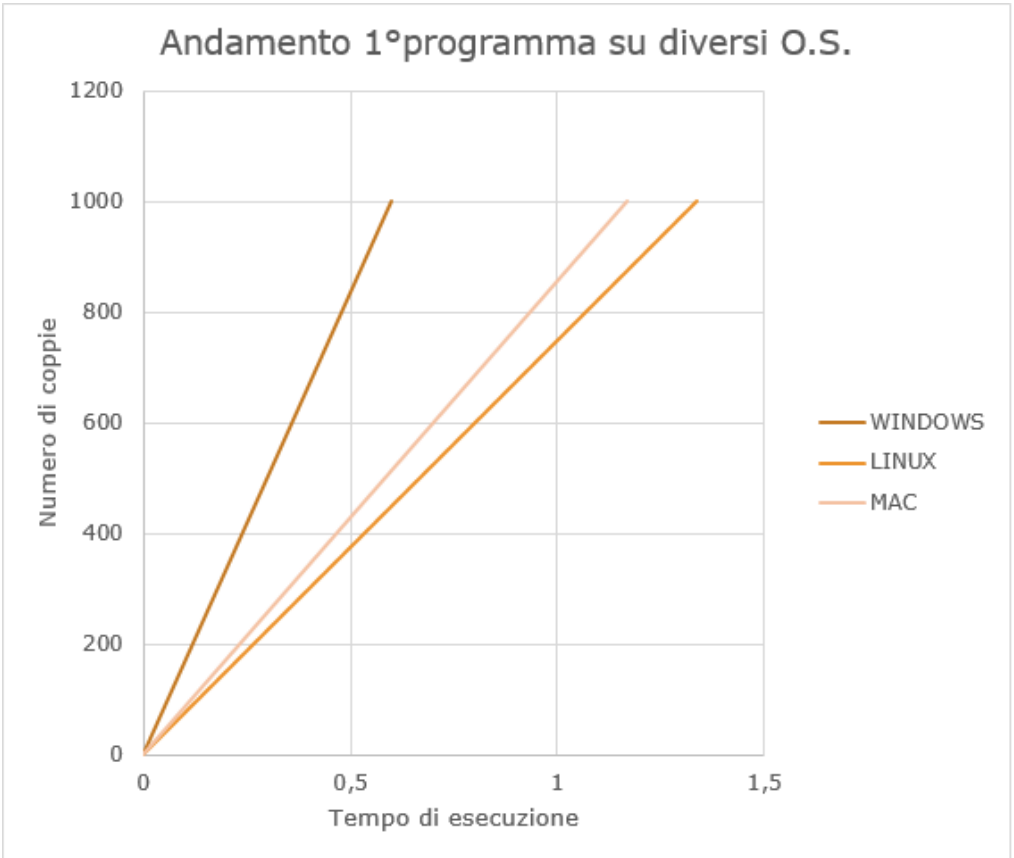
Qui è possibile osservare l'efficienza dei vari codici implementati e come essa varia sui vari dispositivi su cui sono stati testati.

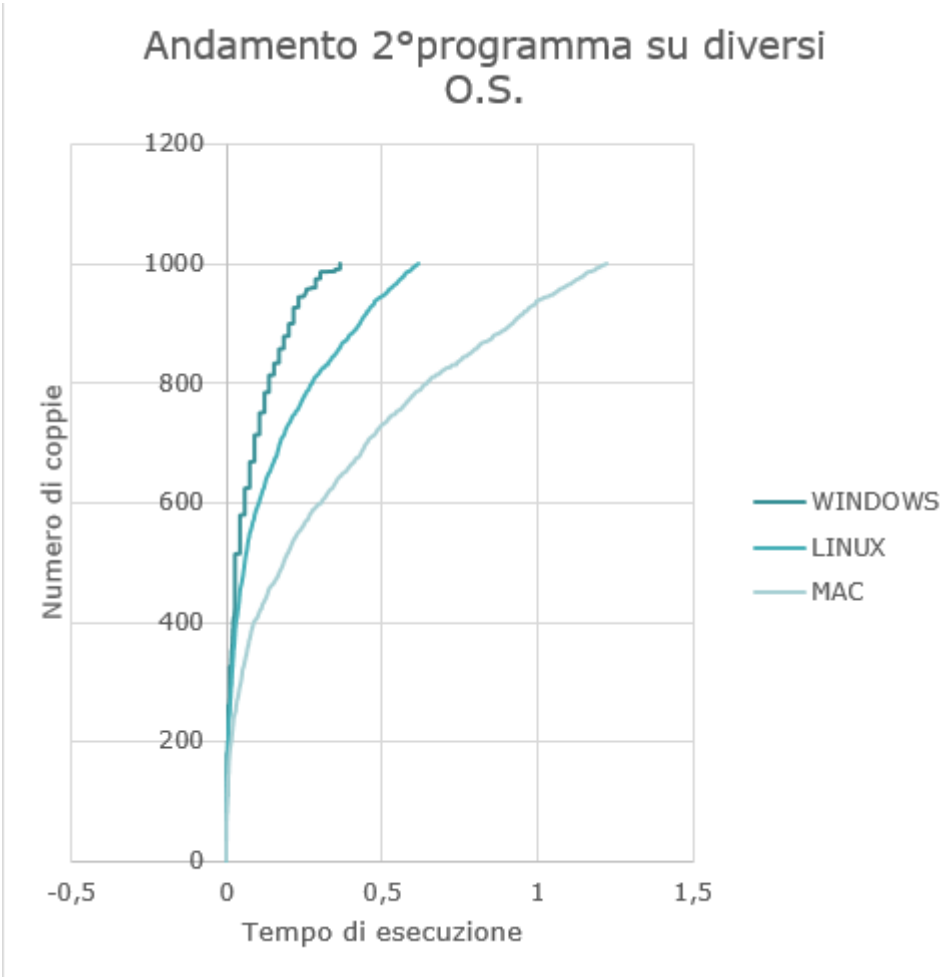


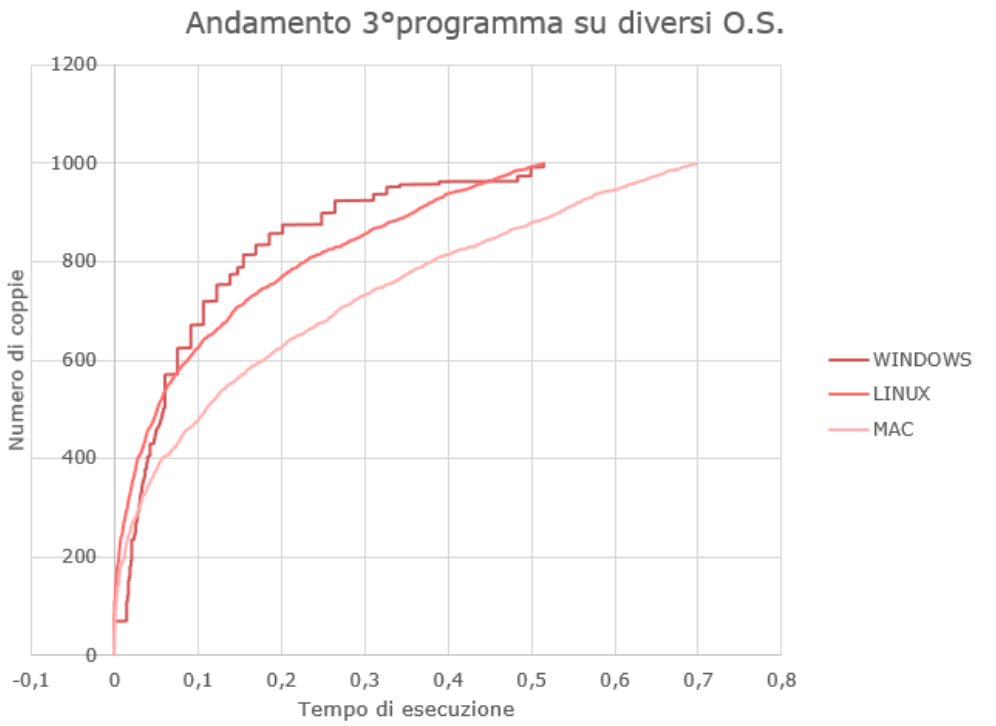


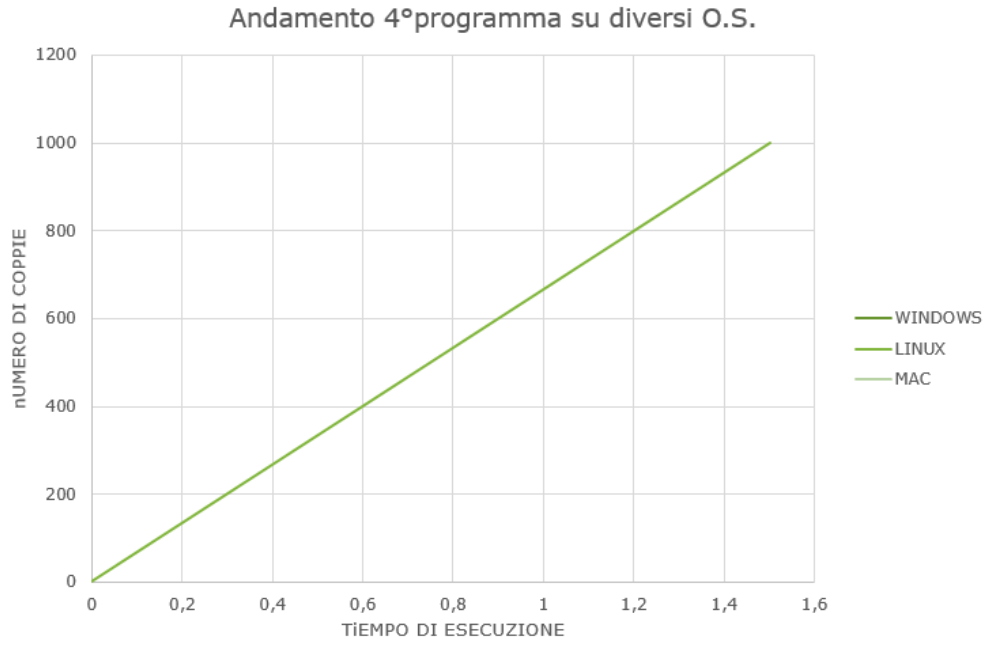


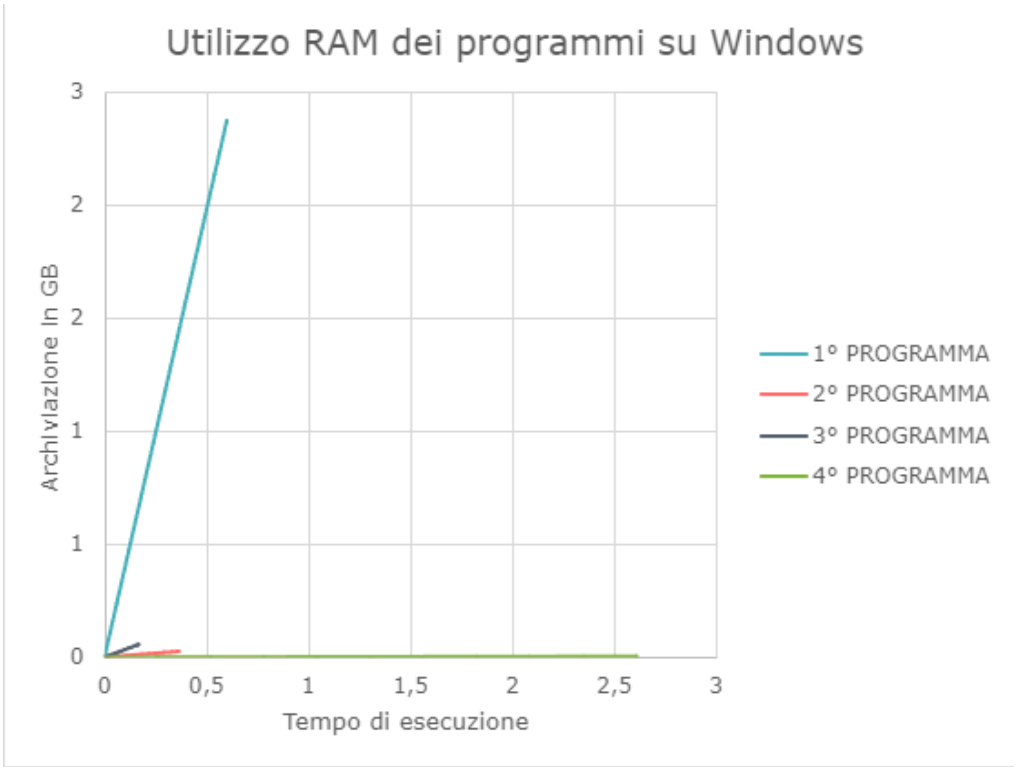
CONFRONTO TRA I PROGRAMMI SU SISTEMI OPERATIVI DIVERSI:

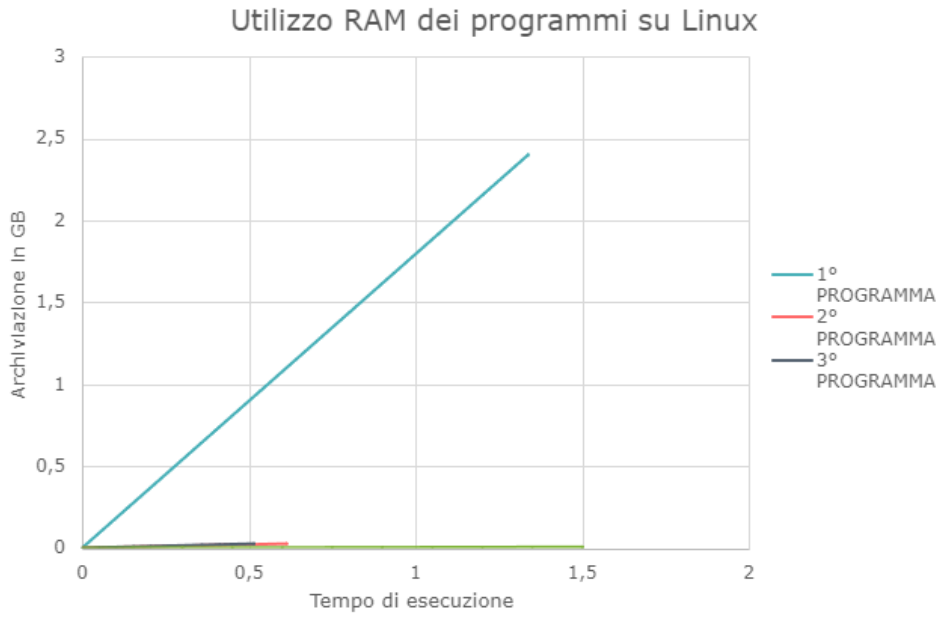


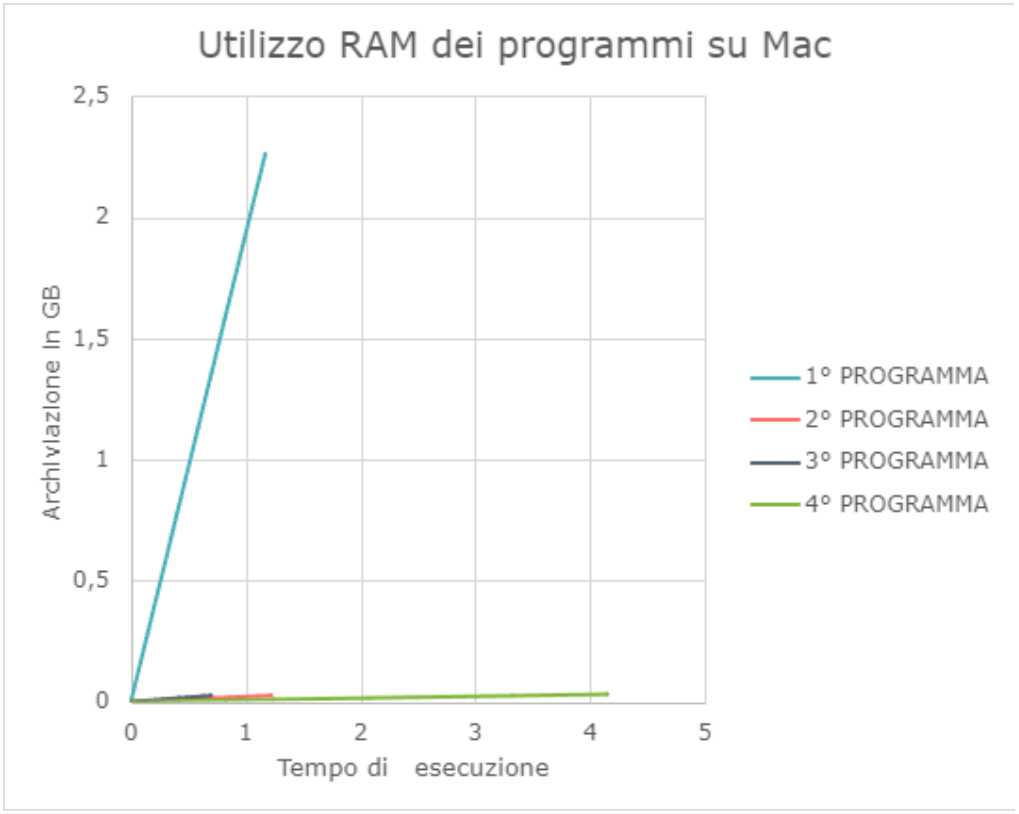

Andamento 2°programma su diversi O.S.
1200
1000
800
600
400
200
0
Numero di coppie
-0,5
0
0,5
1
1,5
Tempo di esecuzione
WINDOWS
LINUX
MAC

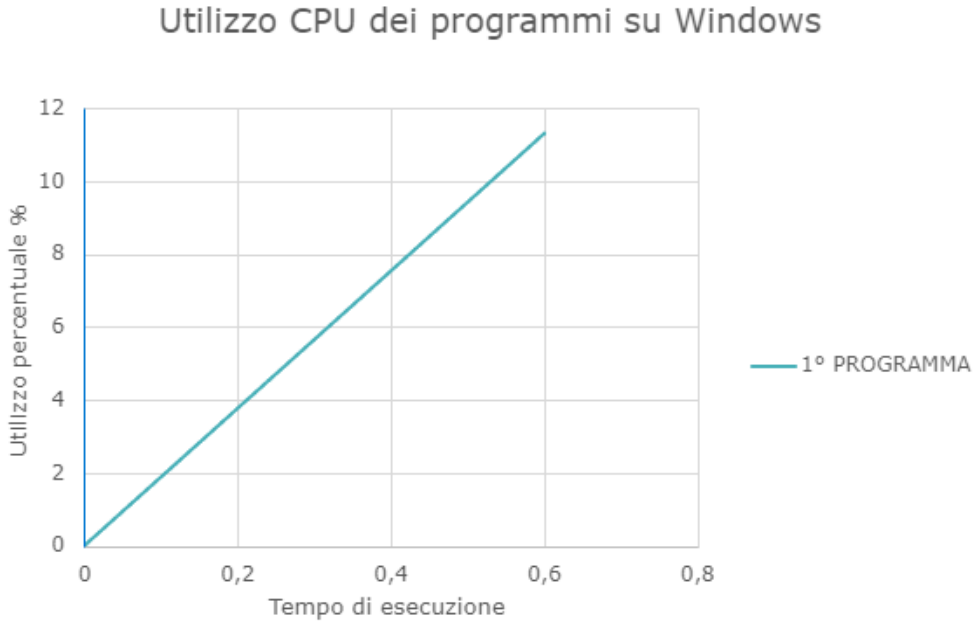

Andamento 3°programma su diversi O.S.
Numero di coppie
1200
1000
800
600
400
200
0
-0,1
0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
Tempo di esecuzione
WINDOWS
LINUX
MAC

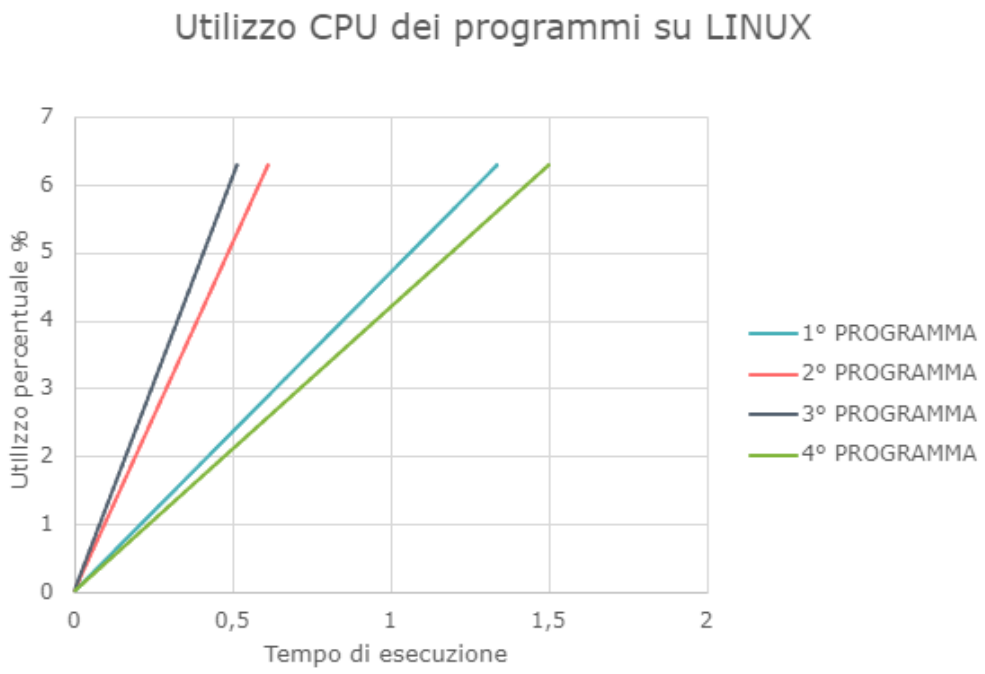

Andamento 4°programma su diversi O.S.
nUMERO DI COPPIE
1200
1000
800
600
400
200
0
0
0,2
0,4
0,6
0,8
1
1,2
1,4
1,6
TiEMPO DI ESECUZIONE
WINDOWS
LINUX
MAC

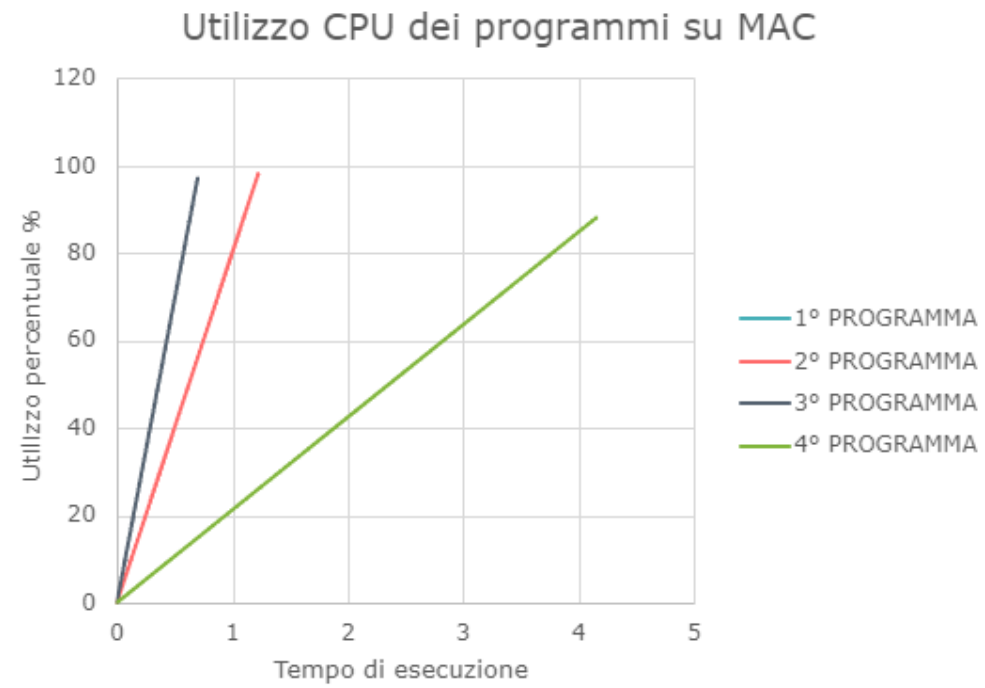

Utilizzo RAM dei programmi su Windows
Archiviazione in GB
Tempo di esecuzione
0
1
1
2
2
3
0
0,5
1
1,5
2
2,5
3
1° PROGRAMMA
2° PROGRAMMA
3° PROGRAMMA
4° PROGRAMMA

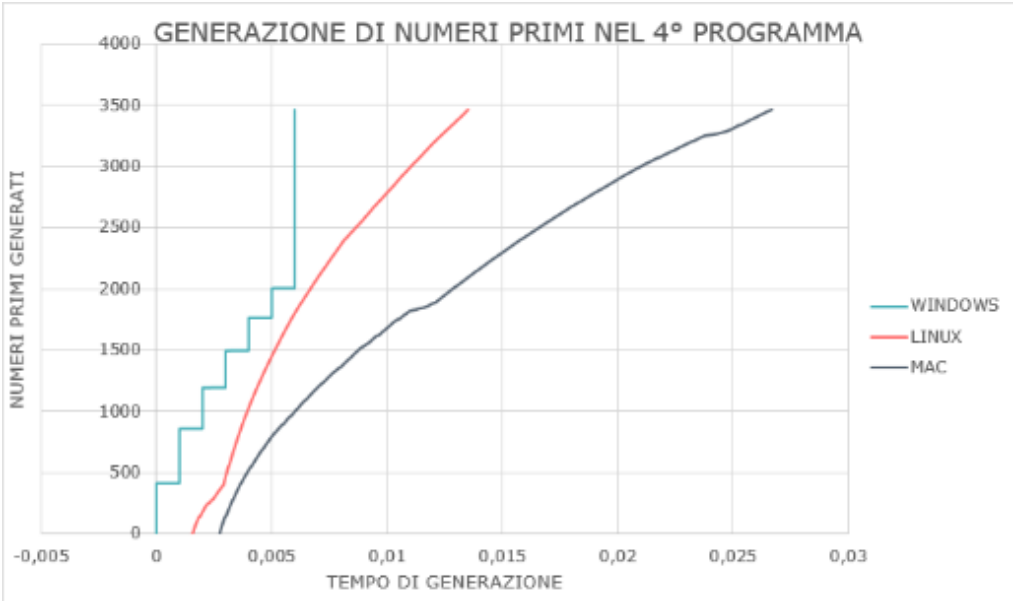

Utilizzo RAM dei programmi su Linux
Archiviazione in GB
Tempo di esecuzione
0
0,5
1
1,5
2
2,5
3
0
0,5
1
1,5
2
1° PROGRAMMA
2° PROGRAMMA
3° PROGRAMMA


Utilizzo RAM dei programmi su Mac
2,5
2
1,5
1
0,5
0
Archiviazione in GB
0
1
2
3
4
5
Tempo di esecuzione
1° PROGRAMMA
2° PROGRAMMA
3° PROGRAMMA
4° PROGRAMMA


Utilizzo CPU dei programmi su Windows
12
10
8
6
4
2
0
Utilizzo percentuale %
0
0,2
0,4
0,6
0,8
Tempo di esecuzione
1° PROGRAMMA


Utilizzo CPU dei programmi su LINUX
Utilizzo percentuale %
0
1
2
3
4
5
6
7
0
0,5
1
1,5
2
Tempo di esecuzione
1° PROGRAMMA
2° PROGRAMMA
3° PROGRAMMA
4° PROGRAMMA


Utilizzo CPU dei programmi su MAC
Utilizzo percentuale %
0
20
40
60
80
100
120
0
1
2
3
4
5
Tempo di esecuzione
1° PROGRAMMA
2° PROGRAMMA
3° PROGRAMMA
4° PROGRAMMA



NOTA: il primo e il quarto programma hanno un grafico lineare perché i numeri sono generati tutti in una volta per poi venir salvati negli appositi array, i quali appena completi verranno stampati totalmente. Di conseguenza i tempi di esecuzione risultano essere quelli finali della stampa e non quelli finali calcolati dalla generazione e dalla successiva stampa di ogni singola coppia.

Riferimenti

-The set of prime numbers “Multifractals and multiscale analysis” Autore: Gerardo Iovane