**SPRAWOZDANIE**

|  |  |  |  |
| --- | --- | --- | --- |
| **Przedmiot** | Algorytmy i struktury danych | **Zadanie** | 3\_2 |
| **Autor** | Krzysztof Chrapowicz | **Grupa** | WCY22IY1S1 |
| **Temat** | Sito Atkina-Bernsteina | | |

1. Treść zadania

Zaimplementuj algorytm wyznaczania liczb pierwszych

z zadanego przedziału [2,n]. (Użyj listy jednokierunkowej

zaimplementowanej wskaźnikowo)

* 1. Metoda realizacji

Algorytm opiera się na wykorzystaniu matematycznego podejścia do identyfikacji liczb pierwszych poprzez wyznaczenie wzorów dla potencjalnych kandydatów na liczby pierwsze.

* 1. Założenia / ograniczenia dotyczące danych:
     1. Dane wejściowe *(opis)*

Liczba n oznaczająca górny zakres ciągu liczb pierwszych

* + 1. Dane wyjściowe *(opis)*

Liczby pierwsze od 2 do n.

1. Realizacja
   1. Algorytm *(lista kroków lub schemat blokowy)*

Algorytm Sita Atkina-Bernsteina w języku C++ rozpoczyna się od inicjalizacji struktury danych, reprezentowanej jako lista jednokierunkowa. W tym celu używane są dwie funkcje pomocnicze: append, odpowiedzialna za dodawanie elementów do listy, i freeList, służąca do zwalniania pamięci po zakończeniu działania programu.

Główna funkcja atkinBernstein rozpoczyna od utworzenia dynamicznej tablicy isPrime, inicjalizowanej wartościami logicznymi false, reprezentującej, czy dana liczba jest liczbą pierwszą. Następnie, przy użyciu matematycznego podejścia Atkina-Bernsteina, oznacza potencjalne liczby pierwsze i eliminuje z listy te, które nie spełniają warunków.

Po wykonaniu algorytmu, pozostałe liczby pierwsze są dodawane do listy jednokierunkowej primeList, a następnie wyświetlane na ekranie. Na koniec, pamięć zajmowana przez listę oraz tablicę dynamiczną jest zwalniana.

* 1. Kod źródłowy

#include <iostream>

#include <cmath>

using namespace std;

struct Node {

int value;

Node\* next;

};

Node\* append(Node\* tail, int value) {

Node\* newNode = new Node;

newNode->value = value;

newNode->next = NULL;

if (tail != NULL) {

tail->next = newNode;

}

return newNode;

}

void freeList(Node\* head) {

while (head != NULL) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

void atkinBernstein(int n) {

Node\* primeList = NULL;

Node\* tail = NULL;

bool\* isPrime = new bool[n + 1];

for (int i = 0; i <= n; i++) {

isPrime[i] = false;

}

int sqrtN = static\_cast<int>(sqrt(n));

for (int x = 1; x <= sqrtN; x++) {

for (int y = 1; y <= sqrtN; y++) {

int num = 4 \* x \* x + y \* y;

if (num <= n && (num % 12 == 1 || num % 12 == 5))

isPrime[num] = !isPrime[num];

num = 3 \* x \* x + y \* y;

if (num <= n && num % 12 == 7)

isPrime[num] = !isPrime[num];

num = 3 \* x \* x - y \* y;

if (x > y && num <= n && num % 12 == 11)

isPrime[num] = !isPrime[num];

}

}

for (int x = 5; x <= sqrtN; x++) {

if (isPrime[x]) {

for (int y = x \* x; y <= n; y += x \* x) {

isPrime[y] = false;

}

}

}

for (int i = 2; i <= n; i++) {

if (isPrime[i]) {

if (primeList == NULL) {

primeList = new Node;

primeList->value = i;

primeList->next = NULL;

tail = primeList;

} else {

tail = append(tail, i);

}

}

}

Node\* current = primeList;

while (current != NULL) {

cout << current->value << " ";

current = current->next;

}

freeList(primeList);

delete[] isPrime;

}

int main() {

int n;

cout << "Podaj n: ";

cin >> n;

cout << "Liczby pierwsze: 2 3 ";

atkinBernstein(n);

cout << endl;

return 0;

}

* 1. Dane wejściowe *(ekran uruchomienia programu)*

Obraz zawierający zrzut ekranu, czarne, wyświetlacz, tekst

Opis wygenerowany automatycznie

* 1. Dane wyjściowe *(ekran uruchomienia programu)*

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

1. Wnioski *(złożoność asymptotyczna algorytmu przy użyciu notacji* ***O lub innej i inne wnioski)***

W związku z tym, iż największą złożoność programu wykazują pętle i żadna nie jest zagnieżdżona, to złożoność obliczeniowa algorytmu wynosi O(n).