МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА**

**«Битовые поля»**

**Выполнил:**студент группы 381706-2

Гущин Александр Владимирович

\_\_\_\_\_\_\_\_\_\_\_  Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_  Подпись

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc532750994)

[Постановка задачи 4](#_Toc532750995)

[Руководство пользователя 5](#_Toc532750996)

[Руководство программиста 6](#_Toc532750997)

[Описание структуры программы 6](#_Toc532750998)

[Описание алгоритмов 6](#_Toc532750999)

[Эксперименты 9](#_Toc532751000)

[Заключение 10](#_Toc532751001)

[Литература 11](#_Toc532751002)

[Приложения 12](#_Toc532751003)

[TBitField.ccp 12](#_Toc532751004)

[TSet.cpp 19](#_Toc532751005)

Введение

Битовые поля широко используется для хранения информации. Отличительной особенностью битовых полей является эффективное использование памяти. Благодаря этому этот тип хранения данных находит разнообразное применение в с сфере математических задач.

Постановка задачи

Необходимо написать реализацию битового поля и множества со всеми сопутствующими конструкторами, методами и перегрузками. Произвести замеры времени.

Руководство пользователя

При запуске программы пользователю предлагается ввести верхнюю границу.

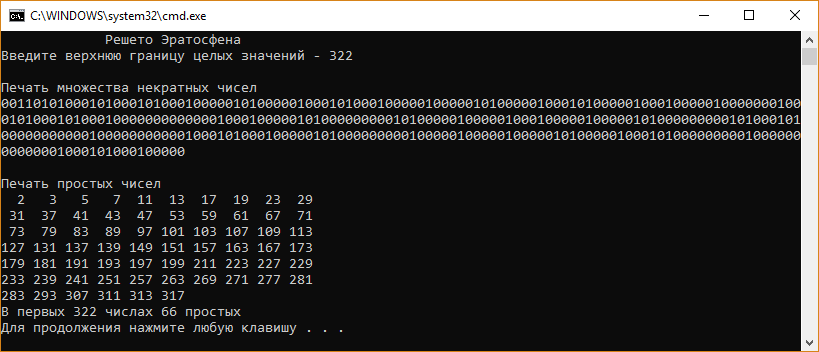


Рис. 1

Руководство программиста

Описание структуры программы

Программа состоит из таких модулей как

* tbitfield.cpp/ tbitfield.h – реализация битового поля
* tset.cpp/tset.h – реализация множества
* sample\_prime\_numbers.cpp – реализация решета Эратосфена

Описание алгоритмов

1. int GetMemIndex(const int n) const - метод класса. Возвращает нужный индекс в массиве из int.
2. TELEM GetMemMask (const int n) const - метод класса. Возвращает маску для бита n.
3. TBitField(int len) – конструктор с параметром. Принимает длину битового поля. Содержит проверку на отрицательность длины. Сначала выделяется память на нужное количество int. После чего производится обнуление.
4. TBitField(const TBitField &bf) – конструктор копирования. Сначала выделяется память на нужное количество int. После чего производится копирование. Сложность O(n).
5. ~TBitField() – деструктор. Удаляет массив из int.
6. int GetLength(void) const – метод класса. Возвращает длину битового поля.
7. void SetBit(const int n) - метод класса. Ставит 1 в битовое поле на позицию n. Содержит проверку на выход за пределы поля. Установка производится логическим | между нужным int и маской от n.
8. void ClrBit(const int n) - метод класса. Ставит 0 в битовое поле на позицию n. Содержит проверку на выход за пределы поля. Установка производится & между нужным int и отрицанием маски от n.
9. int GetBit(const int n) const - метод класса. Возвращает 0 или 1, смотря что стоит в битовом поле на позиции n. Содержит проверку на выход за пределы поля.
10. int operator==(const TBitField &bf) const – перегрузка операции == . Возвращает 0 если массивы различаются или 1 если они одинаковы. Сложность O(n).
11. int operator!=(const TBitField &bf) const - перегрузка операции != . Возвращает 1 если массивы различаются или 0 если они одинаковы. Сложность O(n).
12. TBitField& operator=(const TBitField &bf) - перегрузка операции =. Сначала перераспределяет память под новый размер. Затем приравнивает массивы. Сложность O(n).
13. TBitField operator|(const TBitField &bf) – перегрузка операции |. Производит операцию | между параллельными элементами из 2 массивов. После чего возвращает получившееся битовое поле. Содержит проверку длинны битовых полей. Сложность O(n).
14. TBitField operator&(const TBitField &bf) – перегрузка операции &. Производит операцию & между параллельными элементами из 2 массивов. После чего возвращает получившееся битовое поле. Содержит проверку длинны битовых полей. Сложность O(n).
15. TBitField operator~(void) перегрузка операции ~. Производит операцию ~ между элементами всеми элементами массива. После чего возвращает получившееся битовое поле. Сложность O(n).
16. friend istream &operator>>(istream &istr, TBitField &bf) – перегрузка ввода.
17. friend ostream &operator<<(ostream &ostr, const TBitField &bf) – перегрузка вывода.
18. TSet(int mp) - конструктор с параметром. Принимает длину битового поля. Содержит проверку на отрицательность длины. Выделяет память на нужное количество int.
19. TSet(const TSet &s) - конструктор копирования. Сначала выделяется память на нужное количество int. Затем производит копирование. Сложность O(n).
20. TSet(const TBitField &bf) - конструктор преобразования типа. Переводит TBitField в TSet. Сложность O(n).
21. operator TBitField() – преобразование типа. Переводит TBitField в TSet. Сложность O(n).
22. int GetMaxPower(void) const – возвращает мощность множества.
23. void InsElem(const int Elem) – метод. Добавляет элемент в множество.
24. void DelElem(const int Elem) – метод. Удаляет элемент из множества.
25. int IsMember(const int Elem) const – метод. Проверяет на наличие элемента в множестве.
26. int operator== (const TSet &s) const - перегрузка операции == . Возвращает 0 если множества различаются или 1 если они одинаковы. Сложность O(n).
27. int operator!= (const TSet &s) const - перегрузка операции != . Возвращает 1 если множества различаются или 0 если они одинаковы. Сложность O(n).
28. TSet& operator=(const TSet &s) - перегрузка операции =. Сначала перераспределяет память под новый размер. Затем приравнивает множества. Сложность O(n).
29. TSet operator+ (const int Elem) - объединение с элементом.
30. TSet operator- (const int Elem) - разность с элементом.
31. TSet operator+ (const TSet &s) – объединение множеств. Сложность O(n).
32. TSet operator\* (const TSet &s) – пересечение множеств.
33. TSet operator~ (void) – обратное множество. Сложность O(n).
34. friend istream &operator>>(istream &istr, TSet &bf) - перегрузка ввода.
35. friend ostream &operator<<(ostream &ostr, const TSet &bf) - перегрузка вывода.

Эксперименты

Для эксперимента был взята логическая операция ИЛИ из TBitField.



рис. 2

Заключение

Благодаря этой работе у меня появился опыт взаимодействия с битовыми полями. Битовые поля значительно уменьшают затраты на хранение информации и помогают решать многие практические задания.

Литература

1. <https://ru.wikipedia.org/wiki/Битовое_поле> (Дата обращения 15.12.2018)
2. <http://www.c-cpp.ru/books/bitovye-polya> (Дата обращения 15.12.2018)

Приложения

TBitField.ccp

#include "tbitfield.h"

TBitField::TBitField(int len)

{

if (len >= 0)

{

BitLen = len;

MemLen = len / (8 \* sizeof(TELEM)) + 1;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

pMem[i] = 0;

}

else

throw(\_\_NEG\_SIZE);

}

TBitField::TBitField(const TBitField &bf) // {

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

pMem[i] = bf.pMem[i];

}

TBitField::~TBitField()

{

delete[] pMem;

}

int TBitField::GetMemIndex(const int n) const {

if (n > BitLen || n < 0)

throw(\_\_IND\_IS\_OUT\_OF\_RANGE);

return n / (8 \* sizeof(TELEM));

}

TELEM TBitField::GetMemMask(const int n) const {

if (n > BitLen || n < 0)

throw(\_\_IND\_IS\_OUT\_OF\_RANGE);

int shift = n % (8 \* sizeof(TELEM));

return 1 << shift;

}

int TBitField::GetLength(void) const // получить длину (к-во битов)

{

return BitLen;

}

void TBitField::SetBit(const int n)

{

if (n > BitLen || n < 0)

throw(\_\_IND\_IS\_OUT\_OF\_RANGE);

pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] | GetMemMask(n);

}

void TBitField::ClrBit(const int n)

{

if (n > BitLen || n < 0)

throw(\_\_IND\_IS\_OUT\_OF\_RANGE);

pMem[GetMemIndex(n)] = pMem[GetMemIndex(n)] & ~GetMemMask(n);

}

int TBitField::GetBit(const int n) const {

if (n > BitLen || n < 0)

throw(\_\_IND\_IS\_OUT\_OF\_RANGE);

return ((GetMemMask(n) &

pMem[GetMemIndex(n)]) == GetMemMask(n) ? 1 : 0);

}

TBitField& TBitField::operator=(const TBitField &bf) {

delete[] pMem;

BitLen = bf.BitLen;

MemLen = bf.MemLen;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++)

pMem[i] = bf.pMem[i];

return \*this;

}

int TBitField::operator==(const TBitField &bf) const {

if (BitLen != bf.BitLen)

return 0;

for (int i = 0; i < MemLen; i++)

if (pMem[i] != bf.pMem[i])

return 0;

return 1;

}

int TBitField::operator!=(const TBitField &bf) const {

if (BitLen != bf.BitLen)

return 0;

for (int i = 0; i < MemLen; i++)

if (pMem[i] != bf.pMem[i])

return 1;

return 0;

}

TBitField TBitField::operator|(const TBitField &bf) {

TBitField T(BitLen > bf.BitLen ? \*this : bf);

for (int i = 0; i < MemLen; i++)

T.pMem[i] = pMem[i] | bf.pMem[i];

return T;

}

TBitField TBitField::operator&(const TBitField &bf) {

TBitField T(BitLen > bf.BitLen ? \*this : bf);

for (int i = 0; i < MemLen; i++)

T.pMem[i] = pMem[i] & bf.pMem[i];

return T;

}

TBitField TBitField::operator~(void) {

TBitField T(\*this);

for (int i = 0; i < MemLen; i++)

T.pMem[i] = ~pMem[i];

int shift = 32 - BitLen % (8 \* sizeof(TELEM));

T.pMem[MemLen - 1] =

T.pMem[MemLen - 1] << shift >> shift;

return T;

}

istream &operator>>(istream &istr, TBitField &bf) // ввод

{

int i = 0;

char sym;

do

{

istr >> sym;

} while (sym != ' ');

while (1)

{

if (sym == '0')

bf.ClrBit(i++);

else if (sym == '1')

bf.SetBit(i++);

else

break;

}

return istr;

}

ostream &operator<<(ostream &ostr, const TBitField &bf) {

for (int i = 0; i < bf.BitLen; i++)

{

if (bf.GetBit(i) == 1)

ostr << '1';

else

ostr << '0';

}

return ostr;

}

TSet.cpp

#include "tset.h"

TSet::TSet(int mp) : BitField(mp)

{

if (mp > 0)

MaxPower = mp;

else

throw(\_\_NEG\_SIZE);

}

TSet::TSet(const TSet &s) : BitField(s.MaxPower)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

}

TSet::TSet(const TBitField &bf) : BitField(bf.GetLength())

{

MaxPower = bf.GetLength();

BitField = bf;

}

TSet::operator TBitField()

{

return TBitField(MaxPower);

}

int TSet::GetMaxPower(void) const

return MaxPower;

}

int TSet::IsMember(const int Elem) const {

return BitField.GetBit(Elem);

}

void TSet::InsElem(const int Elem) {

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) {

BitField.ClrBit(Elem);

}

TSet& TSet::operator=(const TSet &s)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

int TSet::operator==(const TSet &s) const {

return (MaxPower == s.MaxPower &&

BitField == s.BitField ? 1 : 0);

}

int TSet::operator!=(const TSet &s) const

{

return (MaxPower != s.MaxPower ||

BitField != s.BitField ? 1 : 0);

}

TSet TSet::operator+(const TSet &s)

{

TSet T(MaxPower > s.MaxPower ? MaxPower : s.MaxPower);

T.BitField = BitField | s.BitField;

return T;

}

TSet TSet::operator+(const int Elem)

{

BitField.SetBit(Elem);

return \*this;

}

TSet TSet::operator-(const int Elem)

{

BitField.ClrBit(Elem);

return \*this;

}

TSet TSet::operator\*(const TSet &s)

{

TSet T(MaxPower > s.MaxPower ?

MaxPower : s.MaxPower);

T.BitField = BitField & s.BitField;

return T;

}

TSet TSet::operator~(void)

{

BitField = ~BitField;

return \*this;

}

istream &operator>>(istream &istr, TSet &s) {

int temp;

char sym;

do{

istr >> sym;

} while (sym != '{');

do {

istr >> temp;

s.InsElem(temp);

do {

istr >> sym;

} while (sym != ',' && sym != '}');

} while (sym != '}');

return istr;

}

ostream& operator<<(ostream &ostr, const TSet &s)

{

int i, n;

char sym = ' ';

ostr << '{';

n = s.GetMaxPower();

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

return ostr;

}