МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» им.В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по лабораторной работе № 2
по дисциплине «Алгоритмы и структуры данных, часть 1»
Тема: «Множество как объект»

Студенты гр. 9306 ЕвдокимовО.В.Кныш С.А. Павельев М.С.

Преподаватель Манерагена Валенс

Санкт-Петербург
2020

Содержание

Цель

Сравнение процедурного и объектно-ориентированного подходов на примере задачи обработки множеств.

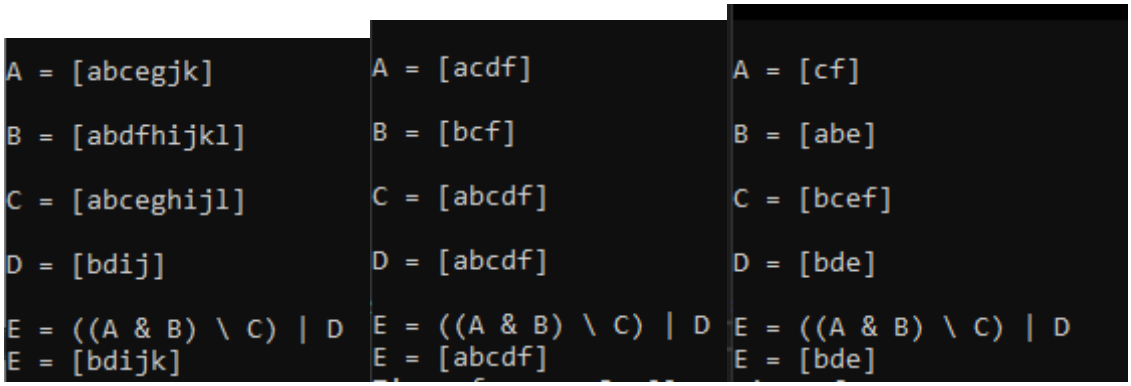
Задание

Универсум: Строчные латинские буквы

Что надо вычислить: Множество, содержащее все буквы, общие для множеств *A* и *B*, за исключением букв, содержащихся в *C*, а также все буквы из *D*

Формализация задания, формула для вычисления пятого множества по четырём заданным — A ∩ B \ C ∪ D

Рисунки с тестами



Результаты измерения времени при использовании процедурного подхода

Способ представления / максимальный размер множества	26	13	6
Список	2.767e-06	1.854e-06	1.08e-06
Машинное слово	2.44e-07	2.17e-07	1.86e-07
Массив битов	7.83e-07	6.67e-07	5.99e-07
Массив	8.93e-07	4.95e-07	2.12e-07

Результаты измерения времени при использовании объектного подхода

Способ представления / максимальный размер множества	26	13	6

Список	3.41e-06	1.769e-06	8.5e-07
Машинное слово	1.19e-07	1.1e-07	9.8e-08
Массив битов	3.07e-07	2.41e-07	1.9e-07
Массив	1.011e-06	8.81e-07	8.23e-07

Результаты эксперимента с отслеживанием вызовов функций-членов

Список

```
creating A from char
A = [abcejkoqrtuvz]
creating B from char
B = [bhjmprvwxz]
creating C from char
C = [abcefgilnstxy]
creating D from char
D = [abdefhnqtuvz]
creating E with default constructor
operation & A with B
creating copy F of A
operation &= F with B
creating copy G of F
deleting F
operation - G with C
creating copy H of G
operation -= H with C
creating copy I of H
deleting H
operation | I with D
creating copy J of I
operation |= J with D
creating copy K of J
deleting J
operation = E with K
deleting K
deleting I
deleting G
E = ((A & B) \ C) | D
E = [abdefhjnqrtuvz]
Time of one = 0

deleting E
deleting D
deleting C
deleting B
deleting A
```

Машинное слово

```
creating A from char
A = [acijmnqrw]

creating B from char
B = [abfghjlmqrswxz]

creating C from char
C = [abcdefhjlmnoprtuwy]

creating D from char
D = [acfghikmnoqrsvxy]

creating E from default constructor
operation & A with B
creating copy F of A
operation &= F with B
creating copy G of F
deleting F
operation - G with C
creating copy H of G
operation -= H with C
creating copy I of H
deleting H
operation | I with D
creating copy J of I
operation |= J with D
creating copy K of J
deleting J
operation = E with K
deleting K
deleting I
deleting G
E = ((A & B) \ C) | D
E = [acfghikmnoqrsvxy]
Time of one = 0.001

deleting E
deleting D
deleting C
deleting B
deleting A
```

Массив битов

```
creating A from char
A = [ceijklpqstuwz]

creating B from char
B = [aceghlouvwyz]

creating C from char
C = [dehjkruvy]

creating D from char
D = [abcdfgjmsvyz]

creating E from default constructor
creating copy F of A
operation & A with B
operation &= F with B
creating copy G of F
deleting F
creating copy H of G
operation - G with C
operation -= H with C
creating copy I of H
deleting H
creating copy J of I
operation | I with D
operation |= J with D
creating copy K of J
deleting J
operation = E with K
deleting K
deleting I
deleting G
E = ((A & B) \ C) | D
E = [abcdefghijklmnopqrstuvwxyz]
Time of one = 0.001

deleting E
deleting D
deleting C
deleting B
deleting A
```

Массив

```
creating A from char
A = [bfhjmnoprstvwxy]
creating B from char
B = [bcdfgikmoprsvwx]
creating C from char
C = [bdgjlmpqrsvyz]
creating D from char
D = [abchklmnoqsuv]
creating E with default constructor
operation & A with B
creating copy F of A
operation &= F with B
creating copy G of F
deleting G
creating copy H of F
deleting F
operation - H with C
creating copy I of H
operation -= I with C
creating copy J of I
deleting J
creating copy K of I
deleting I
creating copy L of K
operation | K with D
creating copy M of L
operation |= L with D
deleting M
creating copy N of L
deleting L
operation = E with N
deleting N
deleting K
deleting H
E = ((A & B) \ C) | D
E = [fowxabchklmnqsuv]
Time of one = 0.002

deleting E
deleting D
deleting C
deleting B
deleting A
```

Выводы

Машинное слово

Самый быстрый как по реализации, так и по времени выполнения способ представления. Также обладает стабильным временем выполнения, т. е. не завит от размера множеств. Требует знания размера универсума, также использование данного способа при размерах универсума больших 64 могут возникнуть проблемы, т.к не всеми компиляторами определены типы большие 64 бит.

Массив битов

Фактически является ухудшенной версией машинного слова, т. к. во-первых все операции (&(«побитовое и»),|(«побитовое или»)), нужно реализовывать собственноручно, во-вторых каждый объект занимает больше места в памяти. В отличии от машинного слова не имеет ограничения по размеру универсума.

Массив

В десять раз медленнее предыдущий способов представления, не так просто реализуем, использует на одну функцию копирования при каждой операции(вычитание, пересечение, объеденение) больше чем все остальные способы представления.

Список

Наиболее трудно реализуемый тип представления, также не отличающийся особой производительностью(скоростью выполнения). Все достоинства списка, а это быстрая вставка элемента в середину списка и быстрое удаление элемента из середины, практически не используются.

Целесообразность использования классов

Использование классов не целесообразно при использовании списков и массивов, т. к. они начинают работать медленнее, однако целесообразно при использовании машинного слова и массива битов, т. к. ,видимо, из-за оптимизаций компилятора они начинают работать быстрее, иного объяснения этому нет, т. к. алгоритмы остались неизменными.

Код программы

Main.cpp

```
//#include "bool_array.h"
#include "Array.h"
//#include "List.h"
//#include "word.h"
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include <time.h>
using namespace std;

int Set::N = 26, Set::cnt = 0, Set::debug = 1;
const long q0 = 1;

char* generate_union(int k)
{
    int m = rand() % k;
    int n = 0;
    char* set = new char[26];
    for (int i = 0; i < m; i++)
    {
        if (rand() % 2) set[n++] = i + 'a';
    }
```

```
    set[n] = 0;
    return set;
}

int main()
{
    srand(time(nullptr));
    char*** in = new char** [q0];
    {
        Set A('A'), B('B'), C('C'), D('D'), E;


        clock_t begin = clock();
        for (long q = 0; q < q0; q++) E = ((A & B) - C) | D;
        clock_t end = clock();

        cout << endl << "E = ((A & B) \\ C) | D";
        E.Show();
        cout << "Time of one = " << (float)(end - begin) / CLK_TCK /
q0 << endl;
    }
    printf("\
n////////////////////////////////////////////////////////////////////
////////////////////////////\n");
    {
        Set::cnt = 0;
        for (long j = 0; j < q0; j++)
        {
            in[j] = new char* [4];
            for (int i = 0; i < 4; i++)
            {
                in[j][i] = generate_union(Set::N);
            }
        }
        clock_t begin = clock();
        for (long i = 0; i < q0; i++)
        {
            Set A(in[i][0]);
            Set B(in[i][1]);
            Set C(in[i][2]);
            Set D(in[i][3]);
            Set E;
            E = ((A & B) - C) | D;
        }
        clock_t end = clock();
        cout << "\nTime of random = " << (float)(end - begin) /
CLK_TCK / q0 << endl;
    }
    system("pause");
    return 0;
}
```

Array.h

```
#pragma once
#include <iostream>
#include <string>
#include <string.h>
using namespace std;

class Set
{
private:
     int n = 0;
     char S, * A;
public:
```

```
      static int N, cnt, debug;

      Set& operator |= (const Set&);
      Set operator | (const Set&) const;
      Set& operator &= (const Set&);
      Set operator & (const Set&) const;
      Set& operator -= (const Set&);
      Set operator - (const Set&) const;
      int power() { return n; }
      void Show();
      Set(char);
      Set();
      Set(string);
      Set(const Set&);
      Set& operator = (const Set&);
      ~Set() {
            if (debug)
                  printf("\ndeleting %c", S);
            delete[] A;
      }
};

Set& Set::operator &= (const Set& B)
{
      if (debug)
            printf("\noperation &= %c with %c", S, B.S);

      Set C(*this);
      n = 0;
      for (int i = 0; i < C.n; ++i)
      {
            for (int j = 0; j < B.n; ++j)
            {
                  if (C.A[i] == B.A[j]) A[n++] = C.A[i];
            }
      }
      A[n] = 0;
      return *this;
}

Set Set::operator & (const Set& B) const
{
      if (debug)
            printf("\noperation & %c with %c", S, B.S);

      Set C(*this);
      return (C &= B);
}

Set& Set::operator |= (const Set& B)
{
      Set C(*this);
      if (debug)
            printf("\noperation |= %c with %c", S, B.S);
      int f;
      for (int i = 0; i < B.n; ++i)
      {
            f = 0;
            for (int j = 0; j < C.n; ++j)
            {
                  if (B.A[i] == C.A[j])  f = 1;
            }
            if (f == 0) A[n++] = B.A[i];
      }
      A[n] = '\0';
      return *this;
```

```
}

Set Set::operator | (const Set& B) const
{
      Set C(*this);
      if (debug)
            printf("\noperation | %c with %c", S, B.S);
      return (C |= B);
}

Set& Set::operator -= (const Set& B)
{
      if (debug)
            printf("\noperation -= %c with %c", S, B.S);
      Set C(*this);
      int f;
      n = 0;
      for (int i = 0; i < C.n; ++i)
      {
            f = 0;
            for (int j = 0; j < B.n; ++j)
            {
                  if (C.A[i] == B.A[j]) f = 1;
            }
            if (f == 0) A[n++] = C.A[i];
      }
      A[n] = 0;
      return *this;
}

Set Set::operator - (const Set& B) const
{
      if (debug)
            printf("\noperation - %c with %c", S, B.S);
      Set C(*this);
      return (C -= B);
}

void Set::Show()
{
      cout << endl << S << " = [" << A << "]" << endl;
}

Set::Set() : n(0), S('A' + cnt++), A(new char[N + 1])
{
      if (debug)
            printf("\ncreating %c with default constructor", S);
      A[0] = 0;
}

Set::Set(string set) : S('A' + cnt++), n(0), A(new char[N + 1])
{
      if (debug)
            printf("\ncreating %c from string", S);
      for (int i = 0; i < set.length(); ++i) A[n++] = set[i];
      A[n] = 0;
}

Set::Set(char) : S('A' + cnt++), n(0), A(new char[N + 1])
{
      if (debug)
            printf("\ncreating %c from char", S);
      for (int i = 0; i < N; i++)
      {
            if (rand() % 2) A[n++] = i + 'a';
      }
```

```
	A[n] = 0;
	cout << '\n' << S << " = [" << A << "]";
}

Set::Set(const Set& B) : S('A' + cnt++), n(B.n), A(new char[N + 1])
{
	if (debug)
		printf("\ncreating copy %c of %c", S, B.S);
	char* dst(A), * src(B.A);
	while (*dst++ = *src++);
}

Set& Set::operator = (const Set& B)
{
	if (debug)
		printf("\noperation = %c with %c", S, B.S);
	if (this != &B)
	{
		char* dst(A), * src(B.A);
		n = B.n;
		while (*dst++ = *src++);
	}
	return *this;
}
```

bool_array.h

```
#pragma once
#include <iostream>
#include <string>
#include <string.h>
using namespace std;

class Set
{
private:
	bool S[26];
	char A;
public:
	static int N, cnt, debug;

	Set& operator = (const Set& B);
	Set& operator |= (const Set&);
	Set operator | (const Set&);
	Set& operator &= (const Set&);
	Set operator & (const Set&);
	Set& operator -= (const Set&);
	Set operator - (const Set&);
	Set(string in);
	Set(char S);
	void Show();
	Set(const Set& B);
	Set();
	~Set()
	{
		if (debug)
			printf("\ndeleting %c", A);
	}
};

Set& Set::operator = (const Set& B)
{
	if (debug)
		printf("\noperation = %c with %c", A, B.A);
	if (this != &B)
	{
```

```
            for (int i = 0; i < 26; i++)
                  if (B.S[i])
                        S[i] = 1;
      }
      return *this;
}

Set& Set::operator -= (const Set& B)
{
      if (debug)
            printf("\noperation -= %c with %c", A, B.A);
      for (int i = 0; i < 26; i++)
            S[i] = (S[i] && !B.S[i]);
      return *this;
}

Set Set::operator - (const Set& B)
{
      Set C(*this);
      if (debug)
            printf("\noperation - %c with %c", A, B.A);
      return (C -= B);
}

Set& Set::operator |= (const Set& B)
{
      if (debug)
            printf("\noperation |= %c with %c", A, B.A);
      for (int i = 0; i < 26; i++)
            S[i] = S[i] || B.S[i];
      return *this;
}

Set Set::operator | (const Set& B)
{
      Set C(*this);
      if (debug)
            printf("\noperation | %c with %c", A, B.A);
      return (C |= B);
}

Set& Set::operator &= (const Set& B)
{
      if (debug)
            printf("\noperation &= %c with %c", A, B.A);
      for (int i = 0; i < 26; i++)
            S[i] = bool(S[i] && B.S[i]);
      return *this;
}

Set Set::operator & (const Set& B)
{
      Set C(*this);
      if (debug)
            printf("\noperation & %c with %c", A, B.A);
      return (C &= B);
}

Set::Set() :A('A' + cnt++) {
      if (debug)
            printf("\ncreating %c from default constructor", A);
      for (int i = 0; i < 26; i++) {
            S[i] = false;
      }
}
```

```
Set::Set(char A) :A('A' + cnt++)
{
	if (debug)
		printf("\ncreating %c from char", A);
	for (int i = 0; i < 26; i++) {
		S[i] = false;
	}

	for (int i = 0; i < N; i++)
	{
		if (rand() % 2) S[i] = true;
	}

	cout << endl<< A << " = [";
	for (int i = 0; i < 26; i++)
		if (S[i])
			cout << char(i + 'a');
	cout << "]" << endl;
}

Set::Set(const Set& B) : A('A'+cnt++)
{
	if (debug)
		printf("\ncreating copy %c of %c",A , B.A);
	for (int i = 0; i < 26; i++) {
		S[i] = B.S;
	}
}

Set::Set(string in) : A('A' + cnt++) {
	if (debug)
		printf("\ncreating %c from string", A);
	for (int i = 0; i < 26; i++) {
		S[i] = false;
	}
	for (char c : in)
		if (c != 0)
			S[int(c) - 'a'] = true;
}

void Set::Show()
{
	cout << endl << A << " = [";
	for (int i = 0; i < 26; i++)
		if (S[i])
			cout << char(i + 'a');
	cout << "]" << endl;
}
```

word.h

```
#pragma once
#include <iostream>
#include <string>
#include <string.h>
using namespace std;

class Set
{
private:
	long int k;
	char S;
public:

	static int N, cnt,debug;
```

```
      Set& operator = (const Set& B);
      Set& operator |= (const Set&);
      Set operator | (const Set&);
      Set& operator &= (const Set&);
      Set operator & (const Set&);
      Set& operator -= (const Set&);
      Set operator - (const Set&);
      Set(char);
      Set(string in);
      Set(const Set& B);
      void Show();
      Set();
      ~Set() {
            if (debug)
                  printf("\ndeleting %c", S);
      }
};

Set& Set::operator = (const Set& B)
{
      if (debug)
            printf("\noperation = %c with %c", S, B.S);
      k = B.k;
      return *this;
}

Set& Set::operator -= (const Set& B)
{
      if (debug)
            printf("\noperation -= %c with %c", S, B.S);
      k = k & (~B.k);
      return *this;
}

Set Set::operator - (const Set& B)
{
      if (debug)
            printf("\noperation - %c with %c", S, B.S);
      Set C(*this);
      return (C -= B);
}

Set& Set::operator |= (const Set& B)
{
      if (debug)
            printf("\noperation |= %c with %c", S, B.S);
      k = k | B.k;
      return *this;
}

Set Set::operator | (const Set& B)
{
      if (debug)
            printf("\noperation | %c with %c", S, B.S);
      Set C(*this);
      return (C |= B);
}

Set& Set::operator &= (const Set& B)
{
      if (debug)
            printf("\noperation &= %c with %c", S, B.S);
      k = k & B.k;
      return *this;
}
```

```
Set Set::operator & (const Set& B)
{
      if (debug)
            printf("\noperation & %c with %c", S, B.S);
      Set C(*this);
      return (C &= B);
}

Set::Set():S('A' + cnt++) {
      if (debug)
            printf("\ncreating %c from default constructor", S);
      k = 0;
}

Set::Set(char S):S('A' + cnt++)
{
      if (debug)
            printf("\ncreating %c from char", S);
      k = 0;
      for (int i = 0; i < N; i++)
      {
            if (rand() % 2) k = k | 1<<i;
      }

      long int q = k;
      cout <<endl<< S << " = [";
      for (int i = 0; i < 26; i++) {
            if (q % 2 != 0)
                  cout << char('a' + i);
            q = q / 2;
      }
      cout << "]" << endl;
}

Set::Set(const Set& B) : S('A' + cnt++)
{
      if (debug)
            printf("\ncreating copy %c of %c", S, B.S);

      k = B.k;
}

Set::Set(string in):S('A' + cnt++) {
      if (debug)
            printf("\ncreating %c from string", S);
      k = 0;
      for (char c : in)
            if (c != 0)
                  k = k | (1 << (int(c) - 'a'));
}



void Set::Show()
{
      long int q = k;
      cout <<endl<< S << " = [";
      for (int i = 0; i < 26; i++) {
            if (q % 2 != 0)
                  cout << char('a' + i);
            q = q / 2;
      }
      cout<<"]"<< endl;
}
```

List.h

```
#pragma once
#include<string>
#include<iostream>


class Set {
private:

	struct Node {
		char el;
		Node* next = nullptr;
		Node(char c, Node* n = nullptr) : el(c), next() { }
		~Node() { delete next; }


		/*void* operator new(size_t size) {
			return malloc(size);
		}

		void operator delete(void* p) {
			free(p);
		}*/
	};

	char S;

	Node* first = nullptr;
	Node* last = nullptr;

public:
	static int N, cnt, debug;

	Set& operator |= (const Set&);
	Set operator | (const Set&)const;
	Set& operator &= (const Set& b);
	Set operator & (const Set&)const;
	Set& operator -= (const Set&);
	Set operator - (const Set&)const;
	Set& operator= (const Set&);
	char* to_String();
	Set() :S('A' + cnt++)
	{
		if (debug)
			printf("\ncreating %c with default constructor", S);
	};
	Set(const Set&);
	Set(std::string in);
	~Set() {
		if (debug)
			printf("\ndeleting %c", S);
		delete first;
	}
	Set(char S);
	void add(char c);
	int get_len();
	Node* delete_elem(Node* lst);
	void Show();

};

char* Set::to_String()
{
	Node* cur;
```

```
	char* res = new char[27];
	int l = 0;
	char c;

	for (int i = 0; i < 26; i++)
	{
		cur = this->first;

		c = 'a' + i;
		while (cur != nullptr)
		{
			if (cur->el == c)
				res[l++] = cur->el;

			cur = cur->next;
		}
	}
	res[l] = '\0';
	return res;
}

void Set::Show()
{
	char* res;
	res = to_String();
	std::cout << std::endl << S << " = [" << res << "]" << std::endl;
	delete res;
}

Set& Set::operator &= (const Set& B)
{
	if (debug)
		printf("\noperation &= %c with %c", S, B.S);

	bool flag = false;
	Set::Node* curB;
	Set::Node* curT = this->first;

	while (curT != nullptr)
	{
		curB = B.first;
		flag = false;
		while (curB != nullptr)
		{
			if (curB->el == curT->el)
				flag = true;
			curB = curB->next;
		}
		if (!flag)
			curT = this->delete_elem(curT);
		if (curT != nullptr)
			curT = curT->next;
	}

	return (*this);
}

Set Set::operator & (const Set& B) const
{
	if (debug)
		printf("\noperation & %c with %c", S, B.S);
	Set C(*this);
	return (C &= B);
}

Set& Set::operator-= (const Set& B)
```

```
{
	Node* curT = this->first;
	Node* curB;
	if (debug)
		printf("\noperation -= %c with %c", S, B.S);

	while (curT != nullptr)
	{
		curB = B.first;
		while (curB != nullptr)
		{
			if (curT != nullptr && (curT->el == curB->el))
			{
				curT = this->delete_elem(curT);
			}
			curB = curB->next;
		}
		if (curT != nullptr)
			curT = curT->next;
	}

	return (*this);
}

Set Set::operator - (const Set& B) const
{
	if (debug)
		printf("\noperation - %c with %c", S, B.S);
	Set C(*this);
	return (C -= B);
}

Set& Set::operator |= (const Set& B)
{
	bool flag = false;
	Set::Node* curB = B.first;
	Set::Node* curT;

	if (debug)
		printf("\noperation |= %c with %c", S, B.S);

	while (curB != nullptr)
	{
		curT = this->first;
		flag = false;
		while (curT != nullptr)
		{
			if (curT->el == curB->el)
				flag = true;

			curT = curT->next;
		}
		if (!flag)
			this->add(curB->el);
		curB = curB->next;
	}
	return (*this);
}

Set Set::operator | (const Set& B) const
{
	if (debug)
		printf("\noperation | %c with %c", S, B.S);
	Set C(*this);
	return (C |= B);
}
```

```
Set& Set::operator= (const Set& B)
{
      Node* curB = B.first;
      if (debug)
            printf("\noperation = %c with %c", S, B.S);

      if (this != &B)
      {
            if (first != nullptr)
            {
                  delete (this->first);
                  first = nullptr;
                  last = nullptr;
            }
            while (curB != nullptr)
            {
                  this->add(curB->el);
                  curB = curB->next;
            }
      }

      return *this;
}

Set::Node* Set::delete_elem(Node* lst)
{
      Node* temp = nullptr;
      if (this->get_len() != 0) {
            if (lst == this->first) {
                  this->first = lst->next;
                  temp = this->first;
            }
            else
            {
                  temp = this->first;
                  while (temp->next != lst)
                  {
                        temp = temp->next;
                  }
                  temp->next = lst->next;
            }
            free(lst);
      }
      return(temp);
}

int Set::get_len()
{
      Node* cur;
      int c = 0;
      cur = this->first;
      while (cur->next != nullptr)
      {
            cur = cur->next;
            c++;
      }

      return c;
}

Set::Set(const Set& a): S('A' + cnt++) {

      Node* curA = a.first;
      if (debug)
            printf("\ncreating copy %c of %c", S, a.S);
```

```
	while (curA != nullptr)
	{
		add(curA->el);
		curA = curA->next;
	}
}

Set::Set(char S) : S('A' + cnt++)
{
	if (debug)
		printf("\ncreating %c from char", S);
	for (int i = 0; i < N; i++)
	{
		if (rand() % 2) add('a' + i);
	}

	char* res = to_String();

	std::cout << '\n' << S << " = [" << res << "]";

	delete[] res;
}

Set::Set(std::string in) : S('A'+cnt++) {
	if (debug)
		printf("\ncreating %c from string", S);

	for (int i = 0; i < in.length(); i++)
		add(in[i]);
}

void Set::add(char c) {
	if (this->first != nullptr) {
		Node* _new = new Node(c);
		if (last == nullptr)
			last = first;
		last->next = _new;
		last = _new;
	}
	else {
		Node* _first = new Node(c);
		first = _first;
		first->next = nullptr;
	}
}
```