Васильев Алексей Игоревич

Балл: 108

Задача №1

Условие

На складе хранится K болтов M12x60 и L шпилек M12x40. Для изготовления одного изделия требуется P крепёжных элементов, из них не менее трёх болтов.

Определите, можно ли использовать весь имеющийся крепёж при изготовлении изделий данного вида.
Входные данные: 3 целых положительных числа - K, L, P, каждое меньше 1000, записанные через пробел.

Выходные данные: возможное количество изготовленных изделий либо 0, если использовать весь крепёж не получится.

Исходные данные	Результат
10 110 40	3
15 150 20	0

Исходный код

```
#include <iostream>

using namespace std;

int main()
{
    int k,l,p,ans = 0;
    cin >> k >> l >> p;
    bool ok = true;
    while(k > 0 && ok) {
        if(k < 3) {
           ok = false;
           break;
        }
        int x = 3 ,y = p-3;
        if(k < 6) {
            x = k;
            y = p-x;
        }
        if(l < y) {
            x = p-l;
            y = l;
        }
        //cout << x << ' ' << y << '\n';
        if(x > k || x < 3) {
            ok = false;
        }
        k -= x;
        l -= y;
        ++ans;
        if(x+y != p) {
            ok = false;
            break;
        }
    }
    if(!ok || k != 0 || l != 0) {
        ans = 0;
    }
    cout << ans;
    return 0;
}
```

Задача №2

Условие

Один из производителей электроники, фирма "Хардстронг", разработала прототип нового вычислительного устройства. Оно позволяет обрабатывать 16-разрядные целые положительные числа и способно выполнять над ними 3 операции: побитовый сдвиг влево на k разрядов (k <= 15), вычитание 1024 и вычитание 1.

В случае побитового сдвига старшие разряды числа, выходящие за пределы разрядной сетки, теряются. Младшие разряды заполняются нулями. Например, сдвиг числа 3 на 2 разряда даст новое значение 12, а сдвиг числа 8193 на 7 даст 128.

В случае получения в результате какой-либо операции отрицательного числа либо нуля выполнение программы завершится исключением (ошибкой).

Выполнение каждой из трёх операций занимает определённое время, не зависящее от значений операндов: сдвиг - 4 такта, вычитание 1024 - 2 такта, вычитание 1 - один такт.

Для заданных целых положительных чисел A и B, каждое меньше 65536, требуется определить минимальное количество тактов для получения числа B из числа A. В случае, если с помощью описанного вычислителя число B из числа A получить невозможно, вывести -1.

Исходные данные	Результат
4 8	4
2049 1024	3

Исходный код

```
#include <bits/stdc++.h>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;
#define pb push_back
#define mp make_pair
#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
#define ll long long
int mod = (1 << 16);
ll f(ll a, ll b) {
 a %= mod;
 if (a < b) {
  return -1;
 }
 ll res = ((a - b) / 1024);
 a -= res * 1024;
 res *= 2;
 if (a < b) {
  return -1;
 }
 res += a - b;
 return res;
}
int main()
{
 ll a, b;
 cin >> a >> b;
 int ans = -1;
 for (ll x = a; x > 0; --x) {
  for (int k = 0; k <= 15; ++k) {
   ll y = x;
   y = (y << k);
   ll cnt = f(y, b) + k * 4 + (a - x);
   if (f(y, b) < 0) {
    continue;
   }
   if (cnt >= 0) {
    if (ans == -1 || cnt < ans) {
     ans = cnt;
    }
   }
  }
 }
 cout << ans;
 return 0;
}
```

Задача №3

Условие

Дано множество отрезков на плоскости. Требуется определить, сколько параллелограммов можно построить из этих отрезков. При этом отрезки можно перемещать, но нельзя менять их размер и угол наклона к осям. Один отрезок можно включать только в один параллелограмм.

Входные данные: в первой строке натуральное N < 100 - количество отрезков. Далее в каждой строке по 4 целых числа, записанных через пробел: x1, y1, x2, y2 - координаты начала и конца отрезка.

Выходные данные: натуральное число - количество параллелограммов.

Исходные данные	Результат
5 0 0 5 0 0 1 5 1 0 0 1 1 1 1 2 2 1 2 5 2	1
4 0 0 0 2 0 0 1 2 0 0 2 2 0 0 2 0	0

Исходный код

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <map>
using namespace std;
#define pb push_back
#define mp make_pair
#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
#define ll long long

ll get_len(vector<ll> &kord) {
 ll x1 = kord[0], y1 = kord[1],  x2 = kord[2], y2 = kord[3];
 ll x = x2 - x1;
 ll y = y2 - y1;
 return x * x + y * y;
}
int main()
{
 int n;
 cin >> n;
 vector<vector<ll>> k(n+1,vector<ll>(5));
 for (int i = 1; i <= n; ++i) {
  for (int j = 1; j <= 4; ++j) {
   cin >> k[i][j];
  }
 }
 vector<bool> used(n + 1);
 int ans = 0;
 for (int i = 1; i <= n; ++i) {
  for (int j = i + 1; j <= n; ++j) {
   for (int p = j + 1; p <= n; ++p) {
    for (int t = p + 1; t <= n; ++t) {
     if (used[i] || used[j] || used[p] || used[t]) {
      continue;
     }
     vector<int> idx;
     idx.pb(i);
     idx.pb(j);
     idx.pb(p);
     idx.pb(t);
     for (int a = 0; a < 4; ++a) {
      if (used[i] || used[j] || used[p] || used[t]) {
       continue;
      }
      for (int b = a + 1; b < 4; ++b) {
       ll len1 = get_len(k[idx[a]]), len2 = get_len(k[idx[b]]);
       if (len1 != len2) {
        continue;
       }
       int c = -1, d =-1;
```

```
        for (int pp = 0; pp < 4; ++pp) {
         if (pp != a && pp != b) {
          if (c == -1) {
           c = pp;
          }
          else {
           d = pp;
          }
         }
        }
        ll len3 = get_len(k[idx[c]]), len4 = get_len(k[idx[d]]);
        if (len3 != len4) {
         continue;
        }
        vector<ll> kord;
        kord.pb(k[idx[a]][0]);
        kord.pb(k[idx[a]][1]);
        kord.pb(k[idx[b]][0]);
        kord.pb(k[idx[b]][1]);
        ll cur_len = get_len(kord);
        if (cur_len != len3) {
         continue;
        }
        kord.clear();
        kord.pb(k[idx[a]][2]);
        kord.pb(k[idx[a]][3]);
        kord.pb(k[idx[b]][2]);
        kord.pb(k[idx[b]][3]);
        if (cur_len != len3) {
         continue;
        }
        ++ans;
        used[i] = used[j] = used[p] = used[t] = true;
        break;
       }
      }
     }
    }
   }
  }
 cout << ans;
 return 0;
}
```

Задача №4

Условие

На вход программе подаётся строка из строчных английских букв длиной < 1000 символов. Требуется выполнить "сжатие" этой строки по следующему принципу: подряд идущие повторяющиеся последовательности символов заменить на цепочки из трёх элементов: число - количество повторов, один экземпляр последовательности и точка.

В результате должна получиться строка наименьшей длины.

Входные данные: строка строчных английских букв.

Выходные данные: "сжатая" строка.

Исходные данные	Результат
aaaabbbbbc	4a.5b.c
abcdbcdbcdef	a3bcd.ef

Исходный код

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
#define pb push_back
#define mp make_pair
#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
int f(int x) {
 int res = 0;
 while (x > 0) {
  x /= 10;
  ++res;
 }
 return res;
```

```
}
int main()
{
 string s;
 cin >> s;
 int n = s.size();
 vector <int> p(n);
 int INF = 1e9;
 vector <int> cnt(n, INF);
 vector<pair<int, string> > a(n, pair<int, string>());
 p[0] = -1;
 for (int i = 0; i < n; ++i) {
  if (!i) {
   cnt[i] = 1;
   string t;
   t.pb(s[0]);
   a[i] = mp(1, t);
   continue;
  }
  for (int len = 1; len <= i + 1; ++len) {
   int cur = 1;
   string t;
   for (int j = i; j > i - len; --j) {
    t.pb(s[j]);
   }
   if (len == i + 1) {
    reverse(all(t));
    if (len + f(len) < cnt[i]) {
     p[i] = -1;
     a[i] = mp(1, t);
    }
    break;
   }
   bool ok = true;
   while (ok) {
    ok = false;
    string k;
    for (int j = i - cur * len; j > max(-1, i - len * (cur + 1)); --j) {
     k.pb(s[j]);
    }
    if (k == t) {
     ++cur;
     ok = true;
    }
   }
   reverse(all(t));
   if (i + 1 == len * cur) {
    if (len + f(cur) < cnt[i]) {
     cnt[i] = len + f(cur);
     p[i] = -1;
     a[i] = mp(cur, t);
    }
   }
   else {
    if (len + f(cur) + cnt[i - len * cur] < cnt[i]) {
     cnt[i] = len + f(cur) + cnt[i - len * cur];
     p[i] = i - len * cur;
     a[i] = mp(cur, t);
    }
   }
  }
 }
 vector<int> ans;
 int pos = n - 1;
 while (pos != -1) {
  ans.pb(pos);
  pos = p[pos];
 }
 reverse(all(ans));
 for (int i = 0; i < ans.size(); ++i) {
  pos = ans[i];
  if (a[pos].first > 1) {
   cout << a[pos].first;
  }
  cout << a[pos].second;
  if (i != ans.size() - 1) {
   cout << '.';
  }
 }
 return 0;
}
```

Задача №5

Условие

Учитель информатики Игорь Петрович включил в программу подготовки 11-го класса изучение языка ассемблера.

Ассемблер - низкоуровневый язык программирования, каждая команда которого прямо соответствует машинной команде (инструкции процессора). Многие команды работают с регистрами - ячейками памяти, расположенными внутри процессора.

На текущий момент ученики Игоря Петровича прошли 3 команды: MOV, ADD, SUB.

Синтаксис этих команд таков:

- MOV <приёмник>, <источник> - копирует содержимое источника в приёмник.
- ADD <приёмник>, <источник> - прибавляет содержимое источника к приёмниу.
- SUB <приёмник>, <источник> - вычитает содержимое источника из приёмника.

Источником для каждой команды может быть числовая константа или название регистра, приёмником - только название регистра.

Доступные регистры - AX, BX, CX, DX. Ёмкость каждого регистра - 16 бит.

Значение регистров на момент начала выполнения программы не определено (то есть заранее неизвестно, может быть любым).

Для контроля успеваемости учеников Игорю Петровичу нужно оценивать понимание учениками "сложных" моментов в работе процессора. Требуется помочь ему составить программу, которая будет определять возникновение следующих ситуаций в коде на ассемблере: переполнение, переход через 0, использование неопределённых значений.

Переполнением будем называть ситуацию, когда в результате арифметической операции результат превысил 65535 или оказался меньше -32768. При этом старший разряд результата отбрасывается, а младшие остаются в регистре. Например, при прибавлении 65535 + 2 в регистре окажется 1, а при вычитании из -32768 1 в регистре окажется 32767. Отрицательные числа хранятся в виде дополнительного кода.

Переходом через 0 будем считать ситуацию, возникающую при изменении знака приёмника в результате сложения или вычитания.

Вход: фрагмент программы на ассемблере. Первая строка - целое положительное число, означающее количество команд и не превышающее 100. Каждая команда записана отдельной строкой, аргументы разделяются пробелами. Числовые константы лежат в диапазоне от -32768 до 65535.

Выход: 3 числа через пробел - количество случившихся переполнений, переходов через 0, использований неопределённых значений.

Исходные данные	Результат

3
MOV AX, 0
MOV BX, 5
SUB AX, BX
	0 1 0

4
MOV AX, 1000
MOV BX, 65000
ADD AX, BX
SUB CX, 128
	1 0 1

Исходный код

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <map>
using namespace std;
#define pb push_back
#define mp make_pair
#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
#define ll long long
int trs(string s) {
```

```
 int res = 0;
 bool m = false;
 for (int i = 0; i < s.size(); ++i) {
  if (!i && s[i] == '-') {
   m = true;
   continue;
  }
  res = res * 10 + (s[i] - '0');
 }
 if (m) {
  res *= -1;
 }
 return res;
}

int main()
{
 int L = -32768, R = 65535, mod = (1 << 16), mod2 = 32768;
 //cout << abs((L - 100) % 32768);
 map<string, ll> cnt;
 map<string, bool> was;
 vector<int> ans(3);
 string s, t, x;
 int T;
 cin >> T;
 while (T--) {
  cin >> s >> t;
  t.pop_back();
  cin >> x;
  bool num = true;
  if (x.size() == 2 && x[1] == 'X') {
   num = false;
  }
  if (t != "AX" && t != "BX" && t != "CX" && t != "DX") {
   ans[2]++;
   continue;
  }
  if (s == "MOV") {
   if (num) {
    was[t] = true;
    cnt[t] = trs(x);
   }
   else {
    if (!was[x] || !was[t]) {
     ans[2]++;
    }
    else {
     cnt[t] = cnt[x];
    }
   }
   continue;
  }
  if (!was[t] || (!num && !was[x])) {
   ans[2]++;
   continue;
  }
  if (s == "ADD") {
   int y;
   if (num) {
    y = trs(x);
   }
   else {
    y = cnt[x];
   }
   int sum = cnt[t] + y, c = cnt[t];
   if (sum > R || sum < L) {
    ans[0]++;
   }
   if ((sum >= 0 && c < 0) || (sum < 0 && c >= 0)) {
    ans[1]++;
   }
   if (sum >= 0) {
    cnt[t] = sum % mod;
   }
   else {
    cnt[t] = sum % mod2;
   }
  }
  else {
   int y;
   if (num) {
    y = trs(x);
   }
   else {
    y = cnt[x];
   }
   int sum = cnt[t] - y, c = cnt[t];
```

```
    if (sum > R || sum < L) {
     ans[0]++;
    }
    if ((sum >= 0 && c < 0) || (sum < 0 && c >= 0)) {
     ans[1]++;
    }
    if (sum >= 0) {
     cnt[t] = sum % mod;
    }
    else {
     cnt[t] = sum % mod2;
    }
   }
  }
  for (int i = 0; i < 3; ++i) {
   cout << ans[i] << ' ';
  }
  return 0;
}
```

Задача №6

Условие

Из журнала отправки электронных писем по протоколу SMTP требуется определить адреса, отправка на которые не удалась.

Пример журнала:

mail1@gmail.com
250 bmstu.ru is pleased to meet you
250 cdp@bmstu.ru sender accepted
250 mail1@gmail.com accepting mail from a client address
354 Enter mail, end with "." on a line by itself
!!mail1@gmail.com 250
250 SMTP state reset

----
razin-ii 99@example.com
250 bmstu.ru is pleased to meet you
250 cdp@bmstu.ru sender accepted
555 syntax error in RCPT TO parameters
250 SMTP state reset

----
mail2@yandex.ru
250 bmstu.ru is pleased to meet you
250 cdp@bmstu.ru sender accepted
250 mail2@yandex.ru accepting mail from a client address
354 Enter mail, end with "." on a line by itself
!!mail2@yandex.ru 250
250 SMTP state reset

Каждая запись журнала состоит из нескольких строк. Записи разделяются последовательностью "----------" (10 дефисов). В первой строке записи указан почтовый адрес, на который направляется письмо. Далее идут строки, начинающиеся либо с трёхзначного кода SMTP, либо с группы из 8 пробелов, либо пустые строки. Если один из кодов, связанных с письмом, лежит в диапазоне от 500 до 599, значит, при отправке имела место ошибка. Так, второй адрес в примере содержит пробел, что привело к ошибке 555.

Входные данные: журнал отправки электронных писем. Журнал завершается строкой, состоящей из одного символа - точки.

Выходные данные: список адресов из журнала по порядку, попытка отправки писем на которые завершилась неудачно. Вывести по одному адресу в строке.

Исходный код

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <map>
```

```
using namespace std;
#define pb push_back
#define mp make_pair
#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
#define ll long long

bool check(string &s) {
 if (s.size() < 3) {
  return false;
 }
 string t;
 t.pb(s[0]);
 t.pb(s[1]);
 t.pb(s[2]);
 int x = 0;
 for (int i = 0; i < 3; ++i) {
  x = x * 10 + (t[i] - '0');
 }
 return x >= 500 && x <= 599;
}
int main()
{
 string p = "----------", s, cur;
 vector<string> ans;
 getline(cin, cur);
 bool ok = true;
 while (ok) {
  ok = false;
  bool was = false;
  getline(cin, s);
  while (s != p && s != ".") {
   if (check(s)) {
    was = true;
   }
   getline(cin,s);
  }
  if (was) {
   ans.pb(cur);
  }
  if (s == ".") {
   break;
  }
  ok = true;
  getline(cin, cur);
 }
 for (int i = 0; i < ans.size(); ++i) {
  cout << ans[i] << '\n';
 }
 return 0;
}
```

Задача №7

Условие

Разреженной называется такая матрица, в которой количество ненулевых элементов существенно меньше количества нулевых. Разреженные матрицы можно хранить в форматах специального вида, например, только индексы ненулевых элементов и их значения.

Требуется написать программу, которая для разреженной матрицы большого размера определит суммы элементов в каждом столбце.

Входные данные: в первой строке три натуральных числа N, M и K, записанные через пробел. N - количество строк в разреженной целочисленной матрице, M - количество столбцов, K - количество ненулевых элементов. N и M не превышают 10^6, K меньше N*M и не превышает 10^4. Далее по одному в строке указаны ненулевые элементы этой матрицы. Для представления каждого элемента через пробел записаны индексы его строки и столбца, а также значение. Индексы считаются с единицы, значения элементов по модулю не превышают 1000.

Выходные данные: M чисел по одному в строке - суммы элементов в соответствующих столбцах матрицы.

Исходные данные	Результат
4 3 1 1 1 10	10 0 0

Исходный код

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <map>
using namespace std;
#define pb push_back
#define mp make_pair
#define all(a) a.begin(),a.end()
#define rall(a) a.rbegin(),a.rend()
#define ll long long

int main()
{
 int n, m, k;
 cin >> n >> m >> k;
 map<pair<int, int>, bool> was;
 vector<int> ans(m + 1);
 for (int i = 0; i < k; ++i) {
  int x, y, c;
  cin >> y >> x >> c;
  if (was[mp(y,x)]) {
   continue;
  }
  was[mp(y, x)] = true;
  ans[x] += c;
 }
 for (int i = 1; i <= m; ++i) {
  cout << ans[i] << '\n';
 }
 return 0;
}
```