Февраль 19, 2022

10029. Корона2020

Зия подозревает, что заразился коронавирусом. В связи с этим он ведёт исследование на своём ДНК, и, в результате вычислений, выясняет, что три различных числа a, b и c связаны с его ДНК. Зия верит, что, если подставляя в выражение $a <> b <> c$ вместо знаков (<>) + или -, можно получить число 2020, то он не заразился коронавирусом, иначе, если это сделать невозможно, то он заразился. Помогите Зие выяснить, заразился ли он коронавирусом.

Вход. В одной строке заданы числа a, b и c ($1 \le a, b, c \le 10^8$).

Выход. Если Зия не заразится, выведите выражение $a <> b <> c$, которое даёт в результате 2020, иначе выведите слово CORONA. При выводе выражения, между числами и операторами не должно быть пробелов.

Пример входа 1	**Пример выхода 1**
2019 2020 2021	2019-2020+2021
Пример входа 2	**Пример выхода 2**
2019 2020 2022	CORONA

Переберем все возможные знаки между числами a, b, c. Если значение полученного выражения равно 2020, то выводим выражение. Иначе выводим слово CORONA.

Реализация алгоритма
Читаем входные данные.

```
scanf("%d %d %d", &a, &b, &c);
```

Перебираем все возможные знаки между числами. В зависимости от результата выводим ответ.

```
if (a + b + c == 2020) printf("%d+%d+%d", a, b, c); else
if (a + b - c == 2020) printf("%d+%d-%d", a, b, c); else
if (a - b + c == 2020) printf("%d-%d+%d", a, b, c); else
if (a - b - c == 2020) printf("%d-%d-%d", a, b, c); else
                       printf("CORONA\n");
```

9634. Пройти в финал

Добро пожаловать в полуфинальный тур олимпиады! Для того чтобы удачно пройти этот тур олимпиады, организаторы олимпиады считают, что средняя оценка ученика должна быть не меньше 3.5, а также оценка по информатике должна быть A или B. Организаторы также считают, что не все ученики смогут пройти в финал, но они могут и ошибаться, так как эта задача все еще не решена. Решите эту задачу написав программу, которая сможет определить учеников прошедших и не прошедших в финальный тур.

Вход. В первой строке дано количество учеников n ($1 \le n \le 1000$). В последующих n строках записаны: число x_i ($0.0 < x_i \le 5.0$) – средняя оценка i-го ученика и один символ y_i ($y_i \in$ {A, B, C, D, E, F }) – оценка по информатике этого ученика. Ученики пронумерованы от 1 до n.

Выход. Для каждого ученика в отдельной строке выведите 1, если ученик проходит в финал, или 0 в противном случае.

Пример входа	**Пример выхода**
2 3.7 C 4.0 B	0 1

Для каждого ученика следует проверить условие его выхода в финал.

Реализация алгоритма
Читаем входные данные. Обрабатываем n строк.

```
scanf("%d", &n);
while (n--)
{
  scanf("%lf %c", &grade, &ch);
```

Проверяем условие выхода ученика в финал. Выводим ответ.

```
  if (grade >= 3.5 && (ch == 'A' || ch == 'B'))
     printf("1\n");
  else
     printf("0\n");
}
```

940. Мажорирующий элемент

Задан массив длины n, найдите его мажорирующий элемент. Элемент называется мажорирующим, если он встречается в массиве более $\lfloor n/2 \rfloor$ раз.

Вход. Первая строка содержит число n ($1 \le n \le 100$). Вторая строка содержит n натуральных чисел.

Выход. Если массив содержит мажорирующий элемент, то вывести его. Иначе вывести -1.

Пример входа 1
```
7
3 3 5 4 2 3 3
```

Пример выхода 1
```
3
```

Пример входа 2
```
4
2 3 2 3
```

Пример выхода 2
```
-1
```

Пусть x – мажорирующий элемент. Начнем обрабатывать входные числа. Каждое число, равное x, будем класть в стек. При поступлении числа, не равного x, будем извлекать одно число из стека. Тогда в конце обработки данных вершина стека будет содержать мажорирующий элемент.

Изначально очистим стек. При обработке очередного элемента a:

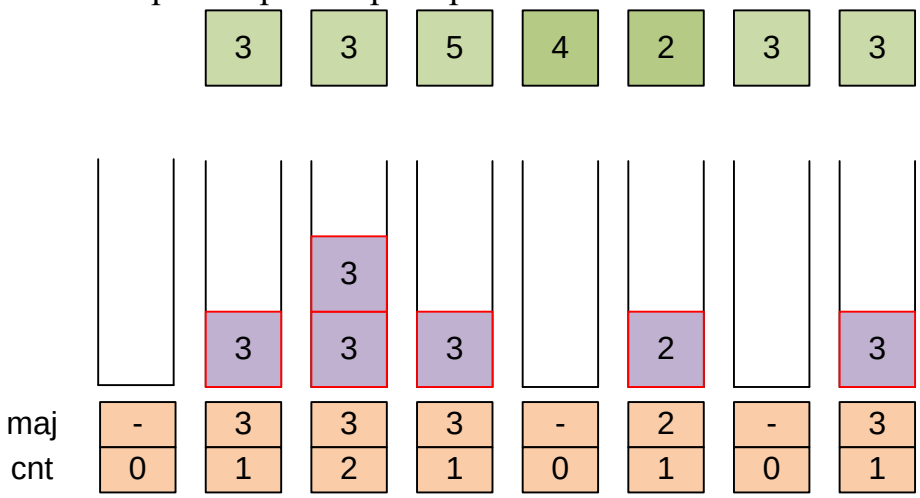
- если стек пустой, то **push**(a);
- если на вершине стека находится число a, то **push**(a);
- если на вершине стека находится число, не равное a, то **pop**();

Если по окончанию обработки всех чисел на вершине стека имеется число x (если стек пустой, то мажоранты нет), то следует проверить, является ли оно мажорирующим элементом. Для этого следует посчитать, сколько раз число x встречается в исходном наборе чисел. Если x встречается более $\lfloor n/2 \rfloor$ раз, то ответ утвердительный.

Поскольку в стеке в любой момент времени находится одно число (возможно несколько раз), то промоделируем стек двумя переменными:

- *maj* – число в стеке;
- *cnt* – количество раз, которое число *maj* находится в стеке;

Пример. Рассмотрим первый пример.



По окончанию алгоритм стеке находится 3. Проверим, является ли она мажорирующим элементом. Для этого следует посчитать, сколько раз число 3 встречается в исходном массиве. Число 3 в массиве длины $n = 7$ встречается 4 раза. Поскольку $4 > \lfloor 7/2 \rfloor$, что число 3 является мажорирующим элементом.

Реализация алгоритма

Входную последовательность чисел сохраним в массиве m.

```
int m[110];
```

Объявим переменные *maj* и *cnt* для моделирования стека:

- *maj* – число в стеке;
- *cnt* – количество раз, которое число *maj* находится в стеке;

```
int maj, cnt;
```

Читаем входные числа в массив m.

```
scanf("%d",&n);
for(i = 0; i < n; i++)
  scanf("%d",&m[i]);
```

Изначально устанавливаем стек пустым.

```
maj = 0; cnt = 0;
```

Обрабатываем входные числа, моделируем работу стека.

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

Если стек пустой (*cnt* = 0), то кладем в него элемент m[*i*].

```
  if (cnt == 0) {maj = m[i]; cnt++;}
```

Если текущий элемент m[*i*] совпадает с вершиной стека *maj*, то кладем m[*i*] в стек.

```
  else if (m[i] == maj) cnt++;
```

Иначе извлекаем элемент из стека.

```
  else cnt--;
}
```

В переменной *cnt* подсчитаем, сколько раз число *maj* встречается в массиве m.

```
cnt = 0;
for(int i = 0; i < n; i++)
  if (m[i] == maj) cnt++;
```

Если $cnt > \lfloor n/2 \rfloor$, то мажоранта существует. Иначе нет (*res* присвоим -1).

```
if(2 * cnt > n) res = maj; else res = -1;
```

Выводим ответ.

```
printf("%d\n",res);
```

10762. Ряд солдат

Имеется ряд *n* солдат, пронумерованных 0 до *n* – 1. Все они выстроены таким образом, что солдат *i* может видеть только солдат с индексами от 0 до *i* – 1. Будем говорить, что солдат имеет четкую видимость, если он не меньше роста, чем все те, кто стоит перед ним. Если он не имеет четкой видимости, то это значит, что по крайней мере один из других стоящих перед ним солдат, выше него.

Для каждого солдата определите, имеет ли он четкую видимость. И если нет, то определите номер ближайшего предыдущего солдата, который выше его.

Вход. Первая строка содержит количество солдат *n* ($1 \le n \le 10^5$). Вторая строка содержит рост *n* солдат.

Выход. Выведите *n* чисел. *i* - ое число должно содержать номер ближайшего предыдущего солдата, который выше *i* - го солдата ростом. Если *i* - ый солдат имеет четкую видимость, то выведите -1.

Пример входа

```
10
5 3 3 4 9 2 7 5 2 4
```

Пример выхода

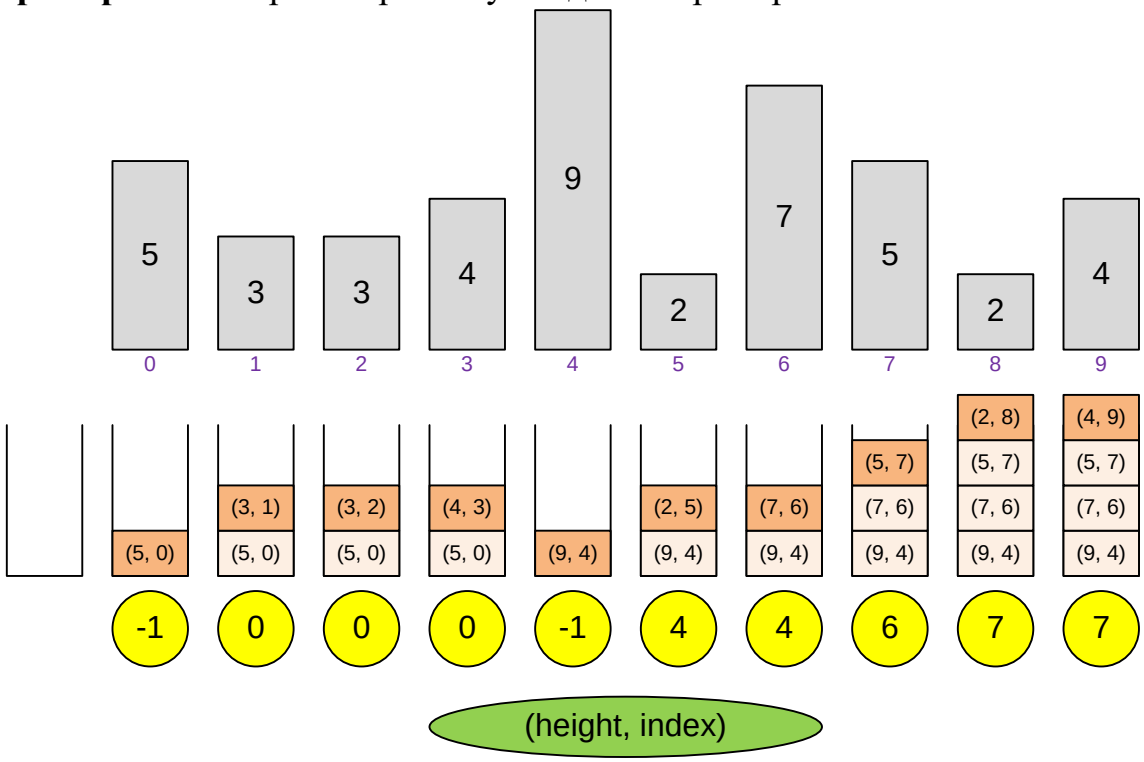
```
-1 0 0 0 -1 4 4 6 7 7
```

Объявим стек пар, который будет хранить информацию про солдат: высоту и индекс солдата. Будем обрабатывать солдат последовательно слева направо. При обработке *i*-го солдата:

- удалим из стека солдат с высотами, не большими высоты *i*-го солдата;
- солдат на вершине стека будет ближайшим предыдущим солдатом, который выше *i* – го;
- занесем в стек информацию о текущем солдате;

В любой момент времени стек хранит солдат по убыванию их высот. Когда приходит следующий солдат, из стека удаляются все солдаты, не выше него. После чего новый солдат занимает место на вершине стека.

Пример. Рассмотрим обработку солдат из примера.



Реализация алгоритма

Объявим стек для хранения высот и индексов солдат.

```
stack<pair<int, int> > soldiersStack; // (height, index)
```

Читаем входные данные.

```
scanf("%d", &n);
h.resize(n);
```

```
for (i = 0; i < n; i++)
  scanf("%d", &h[i]);
```

Последовательно обрабатываем *n* солдат.

```
for (i = 0; i < h.size(); i++)
{
```

Удаляем из стека солдат, высоты которых не больше высоты *i*-го солдата.

```
  while (!soldiersStack.empty() && soldiersStack.top().first <= h[i])
    soldiersStack.pop();
```

Если стек пустой, то *i*-ый солдат имеет четкую видимость. То есть он видит всех солдат перед ним, и никто из них не блокирует ему вид.

```
  if (soldiersStack.empty())
    printf("-1 ");
  else
```

Текущий солдат в стеке – ближайший предыдущий солдат, который выше *i* -го солдата ростом.

```
    printf("%d ", soldiersStack.top().second);
```

Заносим текущего солдата в стек.

```
  soldiersStack.push(make_pair(h[i], i));
}
```

8355. Книжная полка

Мамед раскладывает свои книги на полку. Если на полке нет ни одной книги, то он просто ставит её, если есть, то ставит либо справа, либо слева от уже расставленных книг. Забирает книги он так же, то есть снимает только с правого или левого края. По задаваемой информации требуется смоделировать действия Мамеда и вывести номера книг, которые он будет снимать.

Вход. В первой строке содержится число n ($1 \le n \le 10000$) – количество операций, которые выполнил Мамед. Далее в n строках находится информация об операциях. Каждая операция постановки книги на полку описывается парой чисел.

Первое из них (1 или 2) показывает, книга ставится с левого края или с правого соответственно, второе целое число (от 0 до 10000) обозначает номер книги. Операции снятия книги с полки описывается одним числом – 3 или 4, с левого и правого края соответственно.

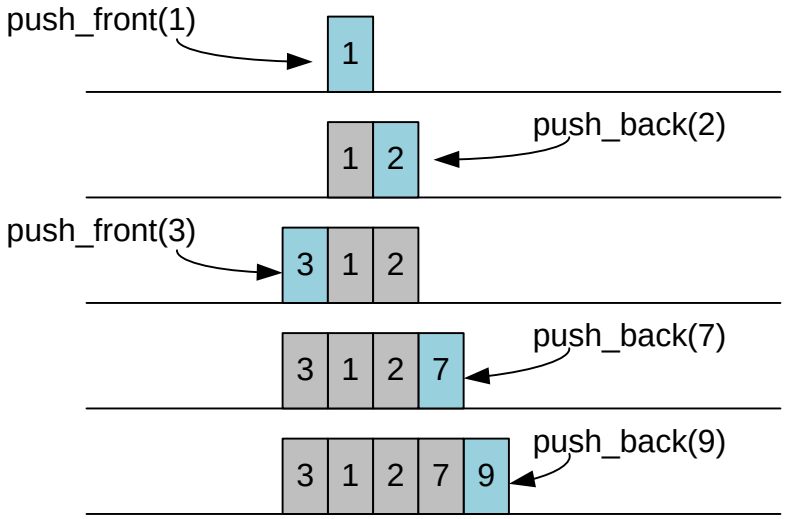
Выход. Для каждой операции снятия книги с полки вывести номер снимаемой книги.

Пример входа	**Пример выхода**
10 1 1 2 2 1 3 2 7 2 9 3 4 3 3 4	3 9 1 2 7

Полка, на которую ставятся книги, представляет собой двустороннюю очередь. В задаче следует промоделировать следующие операции:

push_front(*x*) – книгу номер *x* ставим слева;
push_back(*x*) – книгу номер *x* ставим справа;
pop_front() – снимаем книгу с левого края;
pop_back() – снимаем книгу с правого края;

Пример. Рассмотрим порядок расстановки книг на полке.



Книги с полки извлекаются в следующем порядке:

pop_front() – книга номер 3;
pop_back() – книга номер 9;
pop_front() – книга номер 1;
pop_front() – книга номер 2;
pop_back() – книга номер 7;

Реализация алгоритма
Объявим рабочую очередь.

```
deque<int> s;
```

Читаем количество операций *n*.

```
scanf("%d",&n);

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

Читаем код операции *cmd*.

```
  scanf("%d",&cmd);
  if (cmd == 1)
  {
```

Ставим книгу номер *val* на полку с левого края.

```
    scanf("%d",&val);
    s.push_front(val);
  } else
  if (cmd == 2)
  {
```

Ставим книгу номер *val* на полку с правого края.

```
    scanf("%d",&val);
    s.push_back(val);
  } else
  if (cmd == 3)
  {
```

Выводим номер книги с левого края и снимаем ее.

```
    printf("%d\n",s.front());
    s.pop_front();
  } else
  {
```

Выводим номер книги с правого края и снимаем ее.

```
    printf("%d\n",s.back());
    s.pop_back();
  }
}
```

548. Планирование полета

Авиакомпания "NCPC Авиалинии" выполняет полеты между *n* городами, пронумерованными от 1 до *n*, по всему миру. Однако у нее имеется только *n* – 1

различных рейсов (в обоих направлениях), поэтому для путешествия между любыми двумя городами приходится пользоваться несколькими рейсами. Поскольку менеджмент убедился, что существует возможность путешествовать между любыми двумя городами, то существует в точности одно множество рейсов, которыми может воспользоваться пассажир для перемещения между двумя городами (считая что Вы воспользуетесь только одной авиалинией).

В последнее время в NCPC Авиалиниях многие часто летающие пассажиры жаловались, что им приходится слишком часто менять рейсы, чтобы добраться до конечного пункта назначения. Поскольку NCPC Авиалинии не хотят потерять своих клиентов, и при этом сохранить удобство своих рейсов, они решили отменить один из своих рейсов и заменить его другим. Помогите авиакомпании написать программу, которая поможет найти рейс, который следует отменить, а также новый рейс, который следует добавить, чтобы максимальное количество смены рейсов пассажиром при путешествии между парами городов, которые обслуживают NCPC Авиалинии, было минимальным.

Входные данные будут таковыми, что всегда можно улучшить максимальное число смены рейсов.

Вход. Первая строка содержит количество городов n ($4 \le n \le 2500$) в авиалиниях NCPC. Следующие $n - 1$ строка описывает рейсы. Каждый рейс задается парой городов a и b ($1 \le a, b \le n$).

Выход. Вывести следует три строки. Первая строка содержит наименьшее количество рейсов, которое следует сменить при путешествии между парами городов после изменения одного из рейсов. Вторая строка содержит номера городов отменяемого рейса. Третья строка содержит номера городов, между которыми будет добавлен новый рейс.

Если существует несколько решений, вывести любое.

Пример входа	**Пример выхода**
4 1 2 2 3 3 4	2 3 4 2 4

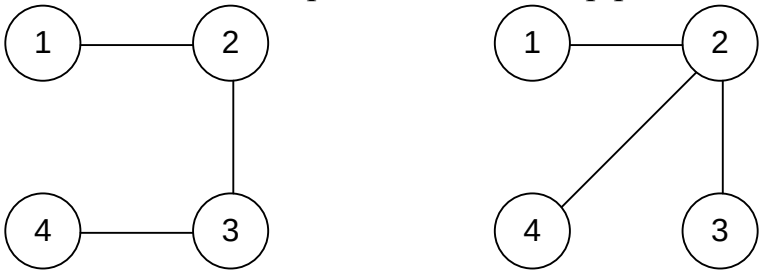
Поскольку входной граф представляет собой дерево и $n \le 2500$, то количество ребер не больше 2500. Переберем все ребра, попробовав удалить соответствующий рейс. После удаления ребра получим два дерева t_1 и t_2. Пусть их диаметры соответственно равны $diam_1$ и $diam_2$. Новый маршрут должен соединять центры этих деревьев. После его добавления получим дерево с диаметром

$$\mathbf{max}(diam_1, diam_2, \lceil diam_1 / 2 \rceil + \lceil diam_2 / 2 \rceil + 1)$$

Теорема. Если диаметр дерева равен d, то максимальное расстояние от его центра до вершин равно $\lceil d/2 \rceil$.

Осталось найти такое ребро, после удаления которого в результате соединения центров полученных деревьев образуется дерево минимального диаметра.

Пример. Слева представлен входной граф, его диаметр равен 3.
Справа – после изменения одного рейса, его диаметр равен 2.



Реализация алгоритма

Объявим глобальные константы и массивы.

```
#define MAX 2510
#define INF 2000000000
vector<vector<int> > g;
int next[MAX];
```

Запускаем поиск в глубину из вершины v. Вершина *prev* хранит предка v. Движение по ребру (v_1, v_2) запрещено в обоих направлениях (оно считается удаленным). В процессе поиска строим массив next: если из вершины v переходим к *to*, то положим next[v] = *to*.

Функция depth возвращает расстояние от v до самой дальней вершины дерева.

```
int depth(int v, int prev = -1)
{
  int height = 0;
  next[v] = v;
  for(int i = 0; i < g[v].size(); i++)
  {
    int to = g[v][i];
    if ((v == v1) && (to == v2)) continue;
    if ((to == v1) && (v == v2)) continue;

    if (to != prev)
    {
      int h = depth(to,v) + 1;
      if (h > height)
      {
        height = h;
        next[v] = to;
      }
    }
  }
```

```
  return height;
}
```

Стартуем из вершины *v* и двигаемся по ребрам дерева поиска в глубину *steps* шагов.

```
int RunTree(int v, int steps)
{
  for (int i = 0; i < steps; i++)
    v = next[v];
  return v;
}
```

Основная часть программы. Читаем входные данные, строим граф.

```
scanf("%d", &n);
g.resize(n+1);
for(i = 0; i < n - 1; i++)
{
  scanf("%d %d", &u,&v);
  g[u].push_back(v);
  g[v].push_back(u);
}

memset(next,-1,sizeof(next));
res = INF;
```

Перебираем ребра графа.

```
for(i = 1; i <= n; i++)
  for(j = 0; j < g[i].size();j++)
    {
      int to = g[i][j];
      v1 = i; v2 = to;
```

Пробуем выбросить ребро $(i, to) = (v_1, v_2)$, получив два отдельных дерева.

```
      int d1 = depth(v1);
      int d2 = depth(v2);
      int u1 = RunTree(v1,d1);
      int u2 = RunTree(v2,d2);
```

Вычисляем диаметры деревьев.

```
      int diam1 = depth(u1);
      int diam2 = depth(u2);

      int CurDiam = max(max(diam1,diam2),(diam1+1)/2 + (diam2+1)/2 +
1);
      if (CurDiam < res)
      {
```

Удаляем ребро (del_1, del_2) и добавляем (add_1, add_2).

```
        res = CurDiam;
        del1 = v1; del2 = v2;
```

```
        add1 = RunTree(u1, diam1/2);
        add2 = RunTree(u2, diam2/2);
      }
    }
```

Выводим ответ.

```
printf("%d\n%d %d\n%d %d\n",res,del1,del2,add1,add2);
```

1427. Калькулятор

У студента Васи есть младший брат Петя, который пошёл в первый класс и начал изучать арифметику. На дом в первом классе задали решить много примеров на сложение и вычитание. Петя попросил Васю проверить домашнее задание. Увидев две страницы написанных каракулями примеров, Вася пришёл в ужас от объёма работы и решил научить Петю использовать для самопроверки компьютер. Для этого Васе нужно написать программу, которая вычисляла бы решение требуемых арифметических примеров.

Вход. Одна строка, в которой могут встречаться цифры и символы ‘+’ и ‘-’. Длина строки не превышает 10000 символов, значение всех чисел в ней не превышают 10000.

Выход. Вывести одно целое число – результат вычислений.

Пример входа	**Пример выхода**
1+22-3+4-5+123	142

Отметим, что первое слагаемое может оказаться отрицательным числом. Последовательно чистаем числа и знаки операции, выполняя соответствующие действия.

Реализация алгоритма

Первое слагаемое читаем в переменную *res*.

```
scanf("%d",&res);
```

Если следующий за очередным числом символ *ch* является знаком операции (сложения или вычитания), то читаем следующее число *a* и прибавляем его к или вычитаем его из общего результата.

Второй вариант чтения данных до конца строки: читать символ *ch* после числа и пока он не равен ‘\n’ продолжать обрабатывать данные.

```
while(scanf("%c",&ch), (ch == '+') || (ch == '-'))
//while(scanf("%c",&ch), ch != '\n')
{
  scanf("%d",&a);
  if (ch == '+') res += a; else res -= a;
}
```

Выводим результат вычислений.

```
printf("%d\n",res);
```

4077. Зарплата в корпорации

Вы работаете менеджером в большой корпорации. Каждый работник может иметь несколько прямых менеджеров и несколько непосредственных подчиненных. Его подчиненные, в свою очередь, также могут иметь своих подчиненных. А его прямые менеджеры могут иметь своих менеджеров. Будем говорить, что x является боссом y, если существует такая последовательность работников a, b, …, d, что x является менеджером a, a является менеджером b и так далее, а d является менеджером y (если x является прямым менеджером y, то x является боссом y). Если a является боссом b, то b не может быть боссом a. Согласно новой политике корпорации зарплата работника, не имеющего подчиненных, равна 1. Иначе зарплата работника равна сумме зарплат всех его подчиненных.

Вам заданы отношения между работниками. Необходимо найти зарплату всех работников.

Вход. Содержит несколько тестов. Первая строка каждого теста содержит количество работников n ($n \le 50$). В следующих n строках заданы отношения между работниками: j-ый символ i-ой строки равен 'Y', если работник i является прямым менеджером работника j, и 'N' иначе.

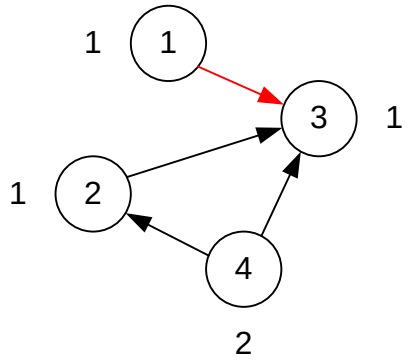
Выход. Для каждого теста вывести в отдельной строке суммарную зарплату всех работников.

Пример входа	**Пример выхода**
1 N 4 NNYN NNYN NNNN NYYN 6 NNNNNN YNYNNY	1 5 17

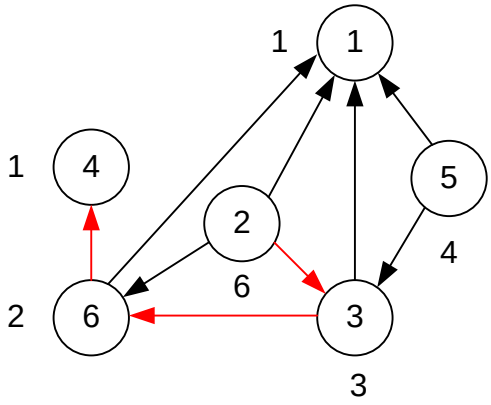
```
YNNNNY
NNNNNN
YNYNNN
YNNYNN
```

Совершим поиск в глубину на ориентированном графе, заданного матрицей смежности. Функция ***dfs***(*i*) вычисляет зарплату *i*-го работника, и сохраняет ее в ячейке salary[*i*]. Сумма зарплат всех работников равна сумме элементов массива *salary*.

Пример. Графы, заданные во втором и третьем тестах, изображены ниже. Возле каждой вершины записана зарплата работника, ей соответствующая. Дуги дерева поиска в глубину изображены красным цветом.



сумма зарплат = 5



сумма зарплат = 17

Реализация алгоритма

Матрицу смежности графа храним в массиве *rel*. Зарплату *i*-го работника подсчитываем в ячейке salary[*i*].

```
#define MAX 51
long long salary[50];
char rel[MAX][MAX];
```

Функция ***dfs*** совершает поиск в глубину из вершины *i* – вычисление зарплаты *i*-го работника.

```
long long dfs(int i)
{
  int j;
  long long &res = salary[i];
```

Если зарплата *i*-го работника уже подсчитана (salary[*i*] ≠ 0), то завершаем поиск.

```
  if (salary[i] > 0) return salary[i];
```

Иначе запускаем поиск в глубину со всех сыновей вершины *i*, вычисляем зарплаты всех подчиненных *i*-го работника. Зарплата *i*-го работника равна сумме зарплат всех его подчиненных.

```
  for(res = j = 0; j < n; j++)
    if (rel[i][j] == 'Y') res += dfs(j);
```

Если *res* = 0, то у *i*-го работника нет подчиненных. Устанавливаем его зарплату равной 1.

```
  if (res == 0) res = 1;
  return res;
}
```

Основная часть программы. Читаем в массив rel матрицу смежности графа.

```
while(scanf("%d\n",&n) == 1)
{
  for(i = 0; i < n; i++) gets(rel[i]);
  memset(salary,0,sizeof(salary));
```

Запускаем поиск в глубину на ориентированном графе. Находим зарплату каждого работника.

```
  for(i = 0; i < n; i++)
    if (!salary[i]) dfs(i);
```

Вычисляем и выводим суммарную зарплату всех работников *res*.

```
  for(res = i = 0; i < n; i++)
    res += salary[i];
  printf("%lld\n",res);
}
```

798. Платформы

В старых играх можно столкнуться с такой ситуацией. Герой прыгает по платформам, висящим в воздухе. Он должен перебраться от одного края экрана до другого. При прыжке с платформы на соседнюю, у героя уходит $|y_2 - y_1|$ энергии, где y_1 и y_2 – высоты, на которых расположены эти платформы. Кроме того, есть суперприём, позволяющий перескочить через платформу, но на это затрачивается $3 \cdot |y_3 - y_1|$ энергии.

Известны высоты платформ в порядке от левого края до правого. Найдите минимальное количество энергии, достаточное, чтобы добраться с 1-й платформы до *n*-й (последней) и список (последовательность) платформ, по которым нужно пройти.

Вход. Первая строка содержит количество платформ n ($2 \le n \le 100000$), вторая n целых чисел, значения которых не превышают по модулю 400 – высоты платформ.

Выход. В первой строке выведите минимальное количество энергии. Во второй – количество платформ, по которым нужно пройти, а в третьей выведите список этих платформ.

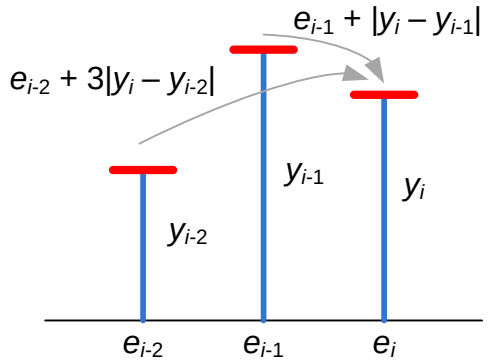
Пример входа	**Пример выхода**
4 1 2 3 30	29 4 1 2 3 4

Пусть e[i] содержит минимальное количество энергии, достаточное для того чтобы добраться с 1-й платформы до i-ой. Очевидно, что e[1] = 0 (добраться с первой платформы до первой стоит ноль энергии), а e[2] = $|y_2 - y_1|$, так как на вторую платформу можно попасть только с первой.

В ячейке p[i] будем хранить номер платформы, с которой прыгнули на i-ую. Изначально положим p[1] = -1 (сначала мы находимся на первой платформе), а также p[2] = 1.

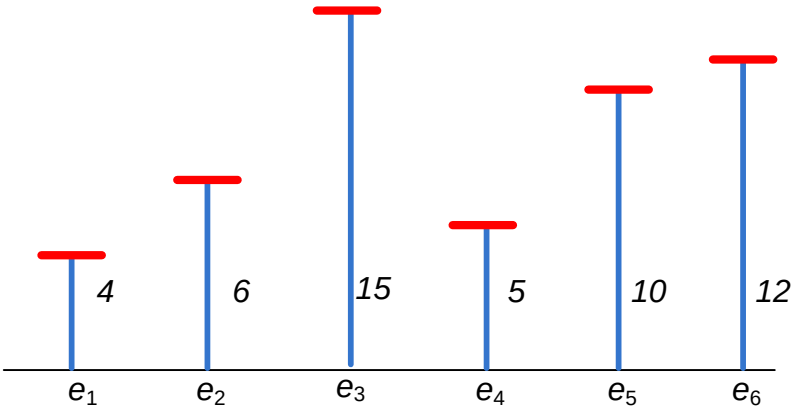
На i-ую платформу ($i \ge 3$) можно прыгнуть либо с (i – 1)-ой, затратив e[i – 1] + $|y_i - y_{i-1}|$ энергии, либо с (i – 2)-ой, совершив суперприем и затратив e[i – 2] + $3 \cdot |y_i - y_{i-2}|$ энергии. Отсюда

$$e[i] = \min(e[i-1] + |y_i - y_{i-1}| , e[i-2] + 3 \cdot |y_i - y_{i-2}|)$$

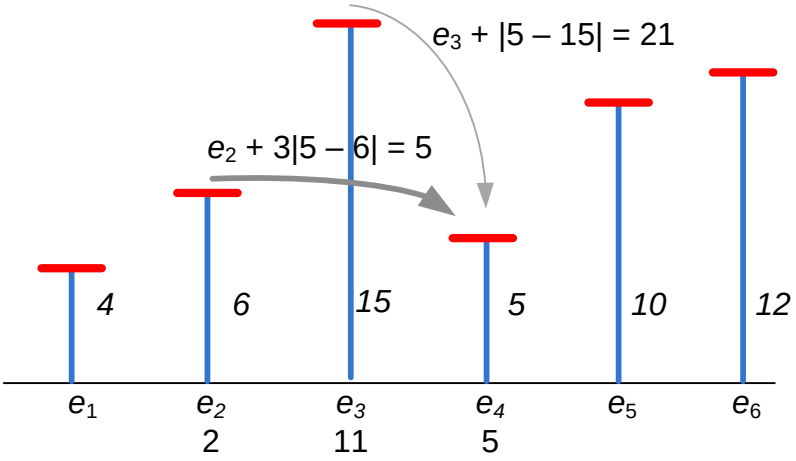


Если на i-ую платформу прыжок совершается с (i – 1)-ой, то устанавливаем p[i] = i – 1. Если с (i – 2)-ой, то положим p[i] = i – 2. Для нахождения количества платформ, по которым совершались прыжки с первой до n-ой, следует пройтись с n-ой платформы до первой двигаясь каждый раз с i-ой платформы на p[i]-ую.

Пример. Пусть имеются 6 платформ с высотами 4, 6, 15, 5, 10, 12.



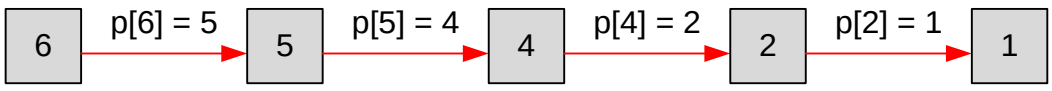
Вычисляем затраченные энергии при прыжках:
e[1] = 0, p[1] = -1;
e[2] = |6 – 4| = 2, p[2] = 1;
e[3] = min(e[2] + |15 – 6|, e[1] + 3 * |15 – 4|) = min(11, 33) = 11, p[3] = 2;



e[4] = min(e[3] + |5 – 15|, e[2] + 3 * |5 – 6|) = min(21, 5) = 5, p[4] = 2;
e[5] = min(e[4] + |10 – 5|, e[3] + 3 * |10 – 15|) = min(10, 26) = 10, p[5] = 4;
e[6] = min(e[5] + |12 – 10|, e[4] + 3 * |12 – 5|) = min(12, 26) = 12, p[6] = 5;

i	1	2	3	4	5	6
y_i	4	6	15	5	10	12
e_i	0	2	11	5	10	12
p_i	-1	1	2	2	4	5

Для восстановления пути следует двигаться с конечной (6-ой) платформы назад по указателям p[i]:



Список платформ, по которым следует пройти, будет следующим:
1, 2, 4, 5, 6

Упражнение. Заполните массивы для следующих входных данных.

i	1	2	3	4	5	6
y_i	8	4	1	5	12	3
e_i						
p_i						

Реализация алгоритма

Объявим необходимые массивы.

```
#define MAX 100001
int y[MAX], e[MAX], p[MAX], res[MAX];
```

Считываем входные данные.

```
scanf("%d",&n);
for(i = 1; i <= n; i++) scanf("%d",&y[i]);
```

Устанавливаем начальные значения ячеек.

```
e[1] = 0;  e[2] = abs(y[2] - y[1]);
p[1] = -1; p[2] = 1;
```

В цикле вычисляем значения ячеек e[i] и p[i] ($3 \le i \le n$), сравнивая значения e[$i-1$] + $|y_i - y_{i-1}|$ и e[$i-2$] + 3*$|y_i - y_{i-2}|$.

```
for(i = 3; i <= n; i++)
{
  a = e[i-1] + abs(y[i] - y[i-1]);
  b = e[i-2] + 3 * abs(y[i] - y[i-2]);
  if (a < b)
  {
    e[i] = a; p[i] = i - 1;
  } else
  {
    e[i] = b; p[i] = i - 2;
  }
}
```

В переменной *ptr* вычисляем количество платформ, по которым нужно пройти. При этом на платформу i следует прыгать с платформы p[i].

```
ptr = 0;
for(i = n; i > 0; i = p[i])
  res[ptr++] = i;
```

Минимальное количество энергии, которое следует потратить для попадания с первой платформы на n-ую, равно e[n]. Количество платформ, по которым следует пройти, равно *ptr*.

```
printf("%d\n%d\n",e[n],ptr);
```

Выводим список номеров платформ.

```
for(i = ptr - 1; i >= 0; i--)
  printf("%d ",res[i]);
printf("\n");
```

5101. Ходжа Насреддин

Ходжа Насреддин находится в левой верхней клетке таблицы размером $n \times n$, а его осел – в правой нижней. Ходжа ходит только вправо или вниз, осел – только влево или вверх.

Сколькими способами они могут встретиться в одной клетке? (Два способа считаются различными, если в них маршруты Ходжи или осла различны).

Вход. Одно число n ($1 \le n \le 50$).

Выход. Выведите одно число – количество способов, которыми Ходжа и осел встретятся. Так как это число может быть очень большим, выведите его по модулю 9929.

Пример входа	**Пример выхода**
3	30

Пусть a[i][j] содержит количество путей, которыми Ходжа Насреддин может добраться из клетки (1, 1) в клетку (i, j).

Пусть b[i][j] содержит количество путей, которыми осел может добраться из клетки (n, n) в клетку (i, j).

Количество способов, которыми Ходжа и осел встретятся в клетке (i, j), равно a[i][j] * b[i][j]. Остается посчитать искомую сумму произведений по модулю:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} \cdot b_{ij} \bmod 9929$$

Пример. Для n = 3 построим массивы:

1	1	1
1	2	3
1	3	6

a_{ij}

6	3	1
3	2	1
1	1	1

b_{ij}

6	3	1
3	4	3
1	3	6

$a_{ij} \cdot b_{ij}$

$$\sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij} \cdot b_{ij} = (6 + 3 + 1) + (3 + 4 + 3) + (1 + 3 + 6) = 30$$

Реализация алгоритма
Объявим рабочие массивы.

```
#define MAX 55
#define MOD 9929
int a[MAX][MAX], b[MAX][MAX];
```

Читаем входное значение *n*.

```
scanf("%d", &n);
memset(a, 0, sizeof(a));
memset(b, 0, sizeof(b));
```

Вычисляем значения ячеек массивов a и b.

```
a[1][1] = 1;
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
  a[i][j] = (a[i][j] + a[i - 1][j] + a[i][j - 1]) % MOD;

b[n][n] = 1;
for (i = n; i >= 1; i--)
for (j = n; j >= 1; j--)
  b[i][j] = (b[i][j] + b[i + 1][j] + b[i][j + 1]) % MOD;
```

В переменной *res* находим количество способов, которыми Ходжа и осел могут встретиться.

```
res = 0;
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
  res = (res + a[i][j] * b[i][j]) % MOD;
```

Выводим ответ.

```
printf("%d\n", res);
```