ACM ICPC 2010–2011 NEERC Moscow Subregional Contest Moscow, October 23, 2011

A. Age of Retirement

Прежде всего в данной задаче надо было понять формальное условие выхода на пенсию. Это самая ранняя дата, на которую возраст человека в месяцах не меньше, чем возраст выхода на пенсию в государстве. Далее задачу можно решать разными способами, например, подсчетом этих двух величин для каждого из дней интересующего нас временного диапазона: от 01.01.1900 до не более чем 2222 года (согласно ограничениям в задаче, человек, родившийся в 2100 году может выйти на пенсию в возрасте около 120 лет).

Однако, легко заметить, что человек может выйти на пенсию либо в день, когда он родился (но не обязательно в этом же месяце), либо первого числа одного из месяцев. Поэтому можно рассмотреть только эти даты. Но еще проще решать задачу «в месяцах». Переведем возраст человека в месяцы, аналогично поступим с пенсионным возрастом и будем из сравнивать. Если условие выхода на пенсию выполнено, то следует проверить, а есть ли дата рождения человека в найденном месяце. Если нет, то возможно придется выполнить повышение пенсионного возраста еще раз.

Приведем текст программы полностью.

```
#include<stdio.h>

int nday(int rtm) {
    static int D[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int m = rtm % 12;
    if (m != 1) return D[m];
    rtm /= 12;
    if (rtm % 4 != 0 || (rtm % 100 == 0 && rtm % 400 != 0)) return 28;
    return 29;
}

int main() {
  int M, N, K;
  int YY, MM, DD;
  scanf("%d%d%d", &M, &N, &K);
  scanf("%d.%d.%d", &DD, &MM, &YY);
  int rtm = (YY + 60) * 12 + MM - 1; //выход на пенсию
                                     //ожидается в этом месяце
```

```
  int mn1 = 2012 * 12; //месяц до которого действует текущий
                       //возраст выхода на пенсию
  for(;;) {
      if (K == 0 || rtm < mn1 - 1)
           break; // больше повышений не будет или возраст достигнут
      if (rtm == mn1 - 1 && DD <= nday(rtm))
          break; //нужный возраст в месяцах, проверяем дату
      K--;
      mn1+=N;
      rtm+=M; //придется повысить возраст выхода на пенсию еще раз
  }
  if(DD > nday(rtm)) {
      rtm++;
      DD = 1;
  }
  printf("%02d.%02d.%4d\n", DD, rtm % 12 + 1, rtm / 12);
}
```

B. Bridges

Задача решается методом динамического программирования. Будем считать величину $D_{i,j}$, означающую математическое ожидание пройденного расстояния, если мы делаем следующее: в точке $i+1$ стоит телепортер, который отправляет нас в точку 1, как только мы дошли до $i+1$, и мы хотим дойти до туда j раз. Тогда ответом будет $D_{n-1,1}$. Начальные значения (при $i < 0$) очевидно равны 0. Осталось научиться пересчитывать D_i, зная D_{i-1}. Посчитаем $D_{i,j}$. Обозначим p_k вероятность того, что в процессе нашего путешествия (для достижения точки $i+1$ j раз) среди b_i плохих мостов из i в $i+1$ сломалось ровно k. Тогда $D_{i,j} = \sum_k p_k(j+k+D_{i-1,j+k})$ (т.к. до точки i нам придётся дойти $j+k$ раз). Заметим, что при вычислении таким алгоритмом $D_{n,1}$ во всех позициях $j \le S = \sum b_i$. Значит, (если бы мы откуда-то знали p_k), наш алгоритм работает за $O(\sum_i S \cdot b_i) = O(S^2)$ (при фиксированном j мы для каждого i делаем b_i операций).

Осталось посчитать p_k. Эти величины означают вероятность того, что для случайного процесса хождения между точками i и $i+1$ мы достигнем j успешных проходов за $j+k$ попыток. Можно попытаться выписать формулу явно, а можно и посчитать это отдельной несложной динамикой. А именно, пусть $d^{a,b}_{j,s}$ будет вероятность того, что при путешествии между двумя точками, между которыми первоначально было a хороших и b плохих мостов, мы сделаем s проходов, в результате чего j плохих мостов разрушатся. Тогда $p_k = d^{a_i,b_i}_{k,j-1} \cdot \frac{a}{a+b-i}$ (т.к. последний проход обязан быть успешным). Как же посчитать $d^{a,b}_{j,s}$? Ответ — очевидной динамикой $d^{a,b}_{j,s} = d^{a,b}_{j-1,s} \cdot \frac{b-j+1}{a+b-j+1} + d^{a,b}_{j,s-1} \cdot \frac{a}{a+b-j}$ (начальное значение $d^{a,b}_{0,0} = 1$). Пар a, b всего $O(n)$, состояний динамики для фиксированных a, b всего лишь $O(b_i \cdot S)$, пересчи-

тываются они за $O(1)$, значит, итоговое время работы $O(\sum b_i \cdot S) = O(S^2)$. Значит, итоговое время работы обеих динамик $O(S^2)$.

C. Crisis

Если $P_1 = P_2 = 0$, то все P_i тождественно равны нулю. Пусть хотя бы одно из чисел P_1 или P_2 не равно нулю.

Если для какого-то i $P_i < P_{i+1}$, то уже следующий элемент будет отрицательным. В частности, в случае $P_1 = 0,\ P_2 \neq 0$ отрицательным будет уже P_3, и остаётся рассмотреть только случаи, когда первые $k \geq 1$ чисел последовательности не равны нулю.

Рассмотрим три подряд идущих числа $P_i,\ P_{i+1},\ P_{i+2}$. Если $P_{i+2} > P_i/2$, то (так как $P_{i+2} = P_i - P_{i+1}$) $P_{i+1} < P_i/2 < P_{i+2}$, и P_{i+3} уже отрицательно. Таким образом, чтобы P_{i+3} было неотрицательно, P_{i+2} не должно превосходить $P_i/2$. Соответственно, получаем, что P_{2i+2} не превосходит $P_2/2^i$, то есть не позднее $2 \cdot log_2(min(P_1, P_2) + 1$ шагов в последовательности или появится отрицательное число, или появится нуль. В последнем случае рассмотрим первый нуль в последовательности. Так как по построению у нас перед ним идёт положительное число x, то за нулём следует снова x, а далее $-x$. То есть через два шага получается отрицательное число. Тем самым оценка количества шагов сверху — $2 \cdot log_2(min(P_1, P_2)) + 3$, с учётом ограничений это не превосходит $2 \cdot (32 + 3) = 70$.

С учётом этого, решение представляет собой моделирование процесса в цикле до тех пор, пока соответствующее число не станет отрицательно. Случай (0,0) может быть рассмотрен отдельно, как вариант можно завершить цикл после 70 итераций и вывести (0,0).

D. Dining Room

Есть несколько разных решений этой задачи.

Решение первое, наиболее простое. Будем решать задачу методом "квадратичного спуска". Пусть, например, к нам пришёл интроверт. Запустим двоичный поиск по искомому максимальному расстоянию. Внутри двоичного поиска для текущего значения d надо уметь проверять следующее утверждение: "правда ли, что есть хотя бы один стол, находящийся вне всех кругов с центрами в занятых столах и радиусами d". Будем делать это методом квадратичного спуска. У нас будет рекурсивная функция, принимающая на вход прямоугольник и решающая задачу для него. Если прямоугольник вырожденный — решение тривиально. Если прямоугольник целиком лежит в каком-то круге, то искомой точки там быть не может. Иначе разбиваем его на 4 примерно равных прямоугольника и запускаемся от них рекурсивно. Для экстраверта можно делать то же самое, только проверять, не лежит ли текущий

прямоугольник полностью вне какого-либо круга. Почему это будет работать быстро? Заметим, что в этом решении прямоугольники будут измельчаться только вдоль интересующей нас области пересечения, поэтому суммарное число рекурсивных вызовов будет $O(n+m)$ (в среднем; из-за того, что мы делаем двоичный поиск, будет получаться даже меньше). Объяснение не совсем формальное, но, т.к. расположение занятых столов не имеет какой-то особой структуры, специально против этого алгоритма, то эта оценка в данной задаче верна. На каждый вызов тратится $O(q)$ операций на просмотр всех кругов, поэтому итоговая сложность $O(q^2 nm \log(n+m))$.

Второе решение, более интуитивно понятное, но не такое быстрое. Будем делать то же самое, что и в прошлом решении (двоичный поиск по ответу), но проверку будем делать не квадратичным поиском, а более естественным алгоритмом. Переберём координату x, после этого надо проверить, есть ли среди точек с такой фиксированной координатой искомая. Переберём все круги и найдём отрезки, которые они высекают на прямой с данной координатой x. Границ отрезков $O(q)$, проверить, есть ли непокрытая точка, можно за $O(q \log q)$. Итоговое (уже строго доказанное) время работы получается $O(nq^2 \log(n+m) \log q)$.

E. Exhibition Hall

Это типичная задача на полный перебор вариантов. На каждом шагу выбирается точка P — самая левая из самых нижних вершин незанятой области, определяются размеры наибольшего прямоугольника, который можно разместить вершиной в точке P. Рассматриваются все незанятые прямоугольники в двух ориентациях, и те из них, которые подходят по размеру, по очереди помещаются в эту точку, и перебор вызывается рекурсивно.

Для определения точки P можно воспользоваться тем фактом, что она является либо нижней левой вершиной стены (в самом начале перебора), либо верхней левой или нижней правой вершиной одного из прямоугольников, уже помещенных на стену.

F. File Sharing

Искомое время в задаче — это время, когда самые последние активизировавшиеся клиенты получат весь файл целиком. Для решения задачи посчитаем количество A самых последних одновременно активизирующихся клиентов в момент времени T их активации.

В этот момент времени каждый из новых клиентов может как максимум получить от каждого из $(N-A)$ старых клиентов по одной независимой части файла, плюс 1 независимую часть от сервера — большей скорости они достичь не смогут из-за отсутствия дополнительных источников и ограничения каналов по скорости. При этом они действительно смогут это сделать, т.к. на одном произвольном предыдущем

моменте времени каждый из $(N - A)$ клиентов может получить одну независимую часть файла от сервера. Таким образом, в момент времени T каждый из последних клиентов получит по $(N - A + 1)$ частей файла.

Далее заметим, что на каждом шаге, начиная с момента времени T мы можем выбрать рассылку от сервера N новых частей файла, которых ранее не было в раздаче: по одной новой части каждому клиенту. Поэтому на каждом шаге, начиная с момента времени $T + 1$ все клиенты могут переслать всем остальным клиентам свою новую часть, полученную в момент времени T, а также получить одну новую часть от сервера. Таким образом, в момент времени $T+1$ и далее каждый клиент получает N новых частей от других клиентов и сервера за 1 шаг.

Итого ответ будет равен:
$T + 1$, если $(N - A + 1) \geq K$
$T + 1 + \lceil (K - (N - A + 1))/N \rceil$, если $(N - A + 1) < K$

G. tourist

В задаче требуется для заданного ориентированного графа найти путь, содержащий заданное множество вершин.

Для решения задачи рассмотрим конденсацию данного графа — граф, получающийся из исходного объединением сильно связных компонент в вершины (подробнее см. в книге Кормена раздел "Сильно связные компоненты"). Если какая-то вершина исходного графа входила в множество обязательных для посещения вершин, то вершина, соответствующая содержащей её компоненте сильной связности, должна будет лежать на искомом пути.

Теперь решим исходную задачу для графа, не содержащего циклов.

Рассмотрим топологическую сортировку данного графа (подробнее см. в книге Кормена раздел "Топологическая сортировка"). Возьмём отмеченные вершины в том порядке, в котором они входят в эту топологическую сортировку. Если искомый путь существует, то самой первой вершиной в этом списке будет вершина старта, самой последней — вершина финиша, а для каждой пары соседних вершин этого списка будет существовать некоторый путь из более ранней вершины в следующую. Если это не выполняется, значит, что либо из стартовой вершины нельзя добраться до всех остальных, либо не из всех отмеченных вершин можно добраться до финиша, либо для каких-то двух отмеченных вершин u и v не существует пути как из u в v, так и из v в u (если бы были оба таких пути, вершины u и v объединились бы в одну компоненту сильной связности, а если нет ни одного пути, то очевидно, что эти две вершины одному пути принадлежать не могут).

Теперь найдём путь, в ориентированном графе без циклов, содержащий заданные вершины в определённом порядке. Это можно сделать, например, так. Пусть v_1, v_2, ..., v_k — список обязательных вершин в порядке обхода. Найдём путь из v_2 в v_1 по обратным рёбрам, затем путь из v_3 в v_2 по обратным рёбрам, и т.д. до пути из

v_k в v_{k-1}. Будем находить путь, например, поиском в ширину (можно и поиском в глубину), при этом пометки о том, что некоторые вершины уже были посещены, между итерациями снимать не будем. Это гарантирует нам время работы этой части алгоритма $O(N^2)$, где N — количество вершин в графе (каждое ребро обойдём не более одного раза, а рёбер $O(N^2)$).

Каждая вершина найденного пути представляет собой компоненту сильной связности исходного графа. Теперь научимся обходить все вершины одной компоненты.

Допустим, в компоненте выделена некоторая вершина s_i. Запустим, например, поиск в ширину внутри этой компоненты из вершины s_i по прямым рёбрам, затем ещё один — по обратным. Таким образом, мы теперь умеем строить путь из вершины s_i в любую вершину этой компоненты и обратно. Так можно обойти все вершины данной компоненты. Это в сумме по всем компонентам не более $O(N^2)$ операций.

Теперь найдём, какие рёбра исходного графа соединяют соседние компоненты на найденном пути — нам нужно по одному ребру для каждой пары соседних компонент. Таким образом, для каждой компоненты мы определим её стартовую вершину s_i, а также вершину f_i, в которой нужно закончить обход этой компоненты.

Теперь можно восстановить весь путь: порядок обхода компонент сильной связности известен, каждую компоненту обходить умеем, ребро, соединяющее две последовательные компоненты знаем.

Теперь убедимся, что найденный путь всегда содержит не более N^2 вершин. Худшим случаем будет, если у нас всего одна компонента сильной связности. Тогда при обходе всех её вершин можно получить суммарный путь длиной $(1+2+\ldots+(N-1))\cdot 2 = N(N-1)$. Теперь, если прибавить к этому максимальную длину пути из стартовой вершины в конечную, получим $N(N-1)+(N-1) = N^2-1$. А так как рёбер на пути на одно меньше, чем вершин, то получаем ровно N^2, что не превосходит ограничения из условия. Если компонент сильной связности больше одной, то несложно доказать, что максимальное количество вершин на пути, построенном таким алгоритмом, будет меньше, чем N^2.

Таким образом, получили решение за $O(N^2)$ по времени и по памяти.

H. Heroes of Money and Magic

Заметим, что в каждый момент времени может быть 2 граничных состояния инвестора — "без акций" и "куплено максимальное количество акций". При этом оптимальное поведение может достигаться только на этих граничных условиях, т.к. если в какой-то день выгодно осуществить продажу акций, то оптимально будет их продать все и при этом в начале дня иметь максимально возможное на данный момент количество. Не следует забывать, что иногда не выгодно ни продавать, ни покупать акции.

Для решения достаточно пройти весь массив цен один раз и аккуратно вычислять лучшее возможное состояние "без акций" и "куплено максимальное количество ак-

ций" в каждый день. При этом начальное состояние для каждого дня равно состоянию предыдущего дня, чтобы финансовый результат не ухудшился в результате операций. В конце получим два состояния, из которых надо выбрать лучшее (продав акции для оценки второго состояния). Записывая переходы между состояниями на каждом шаге нетрудно восстановить и вывести все действия инвестора для каждого дня.

I. I, V, X, L, C, D, M Problem

Так как натуральных чисел, представимых по указанным правилам римскими цифрами всего 3999, то задачу проще всего решать полным перебором всех вариантов. А именно, переберем все числа от 3999 до 1. Переведем каждое из них римскую систему счисления. Проверим, что в строке было достаточно римских цифр для записи этого числа. Из таких вариантов выберем максимальный по длине, а при равной длине - по значению.

Для реализации этого алгоритма при считывании данных подсчитаем в массиве `cnt`, проиндексированном символами с кодами до 127, сколько соответствующих символов находится в строке, при этом строчные латинские буквы сразу переводятся в прописные путем вычитания из их кода 32. Затем нас будут интересовать только элементы этого массива `cnt['I'], cnt['V'], cnt['X'], cnt['L'], cnt['C'], cnt['D'], cnt['M']`.

Перевод числа в римскую систему счисления осуществляется так. Сначала ставим столько букв M, сколько возможно, вычитая при этом из значения числа 1000. Затем ставим комбинацию CM, если оставшееся число не меньше 900. Потом аналогично последовательно поступаем с 500, 400, 100, 90, 50, 40, 10, 5, 4 и 1. Заметим, что при таком порядке сравнения и записи соответствующих цифр, больше трех одинаковых цифр подряд получится не может (для чисел, не превосходящих 3999).

J. Jams

Отсортируем все мнения пользователей по возрастанию скоростей, также добавим скорость = 0, если таковой не было в данных. Получим точки на отрезке $[0, 1000000]$, в каждой из них посчитаем суммарное количество голосов за каждый цвет: $Vote[V_i][Color]$. Далее за линейное время посчитаем суммарное количество голосов за каждый цвет до $i-$ скорости включительно: $SumVoteBefore[V_i][Color]$. Из этого будет также известно общее количество голосов за каждый цвет $SumVote[Color]$, а также легко вычисляется количество голосов за каждый цвет строго после $i-$ скорости: $SumVoteAfter[V_i][Color] = SumVote[Color] - SumVoteBefore[V_i][Color]$.

Заметим, что если выбраны некие границы скоростей A и B для раскраски пробок, то количество несовпадений выбранной раскраски с мнениями пользователей в этом случае равно: $Penalty(A, B) = (SumVoteAfter[A][0] + SumVoteBefore[A][1]) + (SumVoteBefore[B][1] + SumVoteAfter[B][2]) = Penalty1(A) + Penalty2(B)$. T.e.

функция $Penalty(A, B)$ на самом деле распадается на 2 части: зависимую только от A и зависимую только от B, с дополнительным ограничением из задачи $A \le B$. Также отметим, что если A или B не совпадают с каким-либо из заданных значений скоростей, то, уменьшив их до ближайшей заданной скорости, ответ становится более оптимальным без изменения значения функции $Penalty$, поэтому A и B должны быть равны одной из скоростей из входных данных (или 0).

Теперь за линейное время пройдем массив данных от начала до конца и динамическим программированием посчитаем для каждой скорости V_i минимально возможное значение $Penalty1(A)$ при $A \le V_i$, а потом за линейное время пройдем массив данных от конца в начало и динамическим программированием посчитаем для каждой скорости V_i минимально возможное значение $Penalty2(B)$ при $V_i \le B$.

Когда части функции $Penalty(A, B)$ посчитаны для каждой возможной скорости, остается пройти массив скоростей еще раз и найти минимальное значение суммы минимально возможных $Penalty1(A)$ и $Penalty2(B)$ для каждой скорости. Найдя глобальный минимум, необходимо вывести соответствующие значения A и B, на которых достигаются минимумы $Penalty1(A)$ и $Penalty2(B)$.

K. King's Palace Garden

Очевидно, что область сада будет состоять из треугольников A_iOA_j, где O — положение дворца, а A_i и A_j — положения последовательных вершин сада. При этом, во-первых, отрезок A_iA_j виден из точки O, как идущий справа налево (т.е. переход от направления OA_i к OA_j идет против часовой стрелки), а во-вторых, внутри треугольника нет ни одного баобаба. Построим ориентированный граф G, вершинами которого будут допустимые треугольники AOB, а ребра будут идти из вершины AOB в COD тогда и только тогда, когда B и C совпадают, а переход от направления к AB к направлению CD идёт против часовой стрелки. Задача сводится к тому, чтобы найти в графе цикл, который обходит точку O ровно один раз, а сумма площадей его вершин максимальна.

Отсортируем точки A_i по направлениям OA_i. Порядок будет циклическим, т.е. во всех случаях мы считаем, что после точки A_k идут $A_{k+1}, A_{k+2}, \ldots, A_n, A_1, \ldots, A_{k-1}$.

Чтобы для точки A_k определить допустимые треугольники A_kOA_l, надо выбрать точки A_l, для которых направление A_kA_l находится против часовой стрелки от направлений $A_kA_{k+1}, ..., A_kA_{l-1}$, но по часовой стрелке от A_kO. Назовём множество вершин A_kOA_l для конкретного значения k блоком вершин G_k.

Если в графе G есть ребро из A_pOA_q в A_qOA_r, то в нем есть и все ребра из A_pOA_q в A_qOA_s для всех A_s, идущих после A_r (считая от A_q). Это нам пригодится при поиске цикла с наибольшей площадью. Второй факт, который нам понадобится — что в любом цикле найдется точка с минимальной координатой y. Поэтому нам достаточно перебрать все точки A_k, y-координата которых меньше, чем y-координата точки O, и рассмотреть циклы, которые начинаются с вершин A_kOA_l, для которых

направление A_kA_l идет в направлении $(1,0)$ или против часовой стрелки от него, и заканчиваются в одной из вершин A_mOA_k, для которых направление A_mA_k идет по часовой стрелке от $(1,0)$. Кроме того, в цикле не должно быть вершин A_pOA_q, для которых точка A_k лежит между A_p и A_q (в порядке их нумерации).

Задача решается методом динамического программирования. Пусть A_k — предполагаемая точка цикла с минимальным y. Для каждой вершины A_pOA_q будем искать максимальную площадь вершин на путях от блока G_k, которые можно продолжить этой вершиной A_pOA_q. В начале выберем первую вершину A_kOA_l, для которой направление A_kA_l идёт против часовой стрелки от $(1,0)$, и укажем, что для неё площадь равна $S_{kl} = 0$. Для всех остальных вершин графа установим площадь -1. Теперь для каждого блока от G_k до G_{k-1} выполним следующую процедуру. Пусть это блок G_p. Будем перебирать вершины A_pOA_q пока не найдём вершину, для которой $S_{pq} \le 0$. Теперь для всех вершин A_pOA_r, начиная с A_pOA_q, берём максимум S площади S_{pr_1} для r_1 от p до r. В блоке G_r находим первую вершину A_rOA_s, в которую ведёт ребро из A_pOA_r. Находим площадь $S_1 = S + 2 \cdot S(A_pOA_r)$. Поскольку координаты всех вершин целые, эта площадь тоже будет целым 64-битным числом. Дальше есть три варианта. Если $s = k$ и направление A_rA_k идет по часовой стрелке от $(1,0)$, то цикл замкнулся, мы проверяем, является ли площадь максимальной из уже найденных, и если да — запоминаем эту площадь и этот цикл. Если s идёт раньше, чем k (считая от p), присваиваем вершине A_rOA_s площадь $S_{rs} := max(S_{rs}, S_1)$. Если же s идет позже, чем k, не делаем ничего. После того, как все блоки перебраны, мы имеем максимальную площадь контура и список его вершин. Осталось их аккуратно распечатать (не забываем, что полученная нами площадь вдвое больше ответа).

Время работы алгоритма $O(N^3)$. В качестве нижней точки перебираем $O(N)$ вариантов, для каждого варианта перебираем $O(N)$ блоков, для каждого блока перебираем $O(N)$ точек (при выборе максимума из S_{pr_1} подсчитываем частичные максимумы — их пересчёт для каждой точки делаем за $O(1)$).

L. Lamps of the Mind

Вначале заметим, что каждый треугольник однозначно задается парой своих наименьших сторон (под длинами сторон будем понимать не их геометрические длины, а расстояние по дуге окружности без третей точки). В качестве первой части решения будем заполнять квадратный булев массив $a[i][j]$ означающий, что существует треугольник подходящий под ограничения у которого расстояние от первой точки до второй по часовой стрелке равно i, а до третей — j $(i < j)$. Затем, пробежавшись по этому массиву, можно найти пару минимальных сторон из i, $j - i$ и $n - j$, и посчитать количество различных таких пар (например, вновь заполняя квадратный массив).

Осталось понять как быстро реализовать первую часть. Пусть первая точка k. Рассмотрим вторую точку на расстоянии i. Посмотрим, какие точки мы можем взять

в качестве третей, это все точки с положительными ограничениями кроме, может быть, совпадающих по цвету с k-ой и $k + i$-ой точками. Стало быть зная битовый массив хороших точек и точек каждого цвета, можно битовыми операциями обновить массив a[i] для k-ой точки. Заметим также, что можно брать i в качестве минимальной стороны. Получим (n/3)*n*n/32 операций.