Задача 1. «Соревнование картингистов»

Данная задача является наиболее простой в комплекте задач для первого тура, и ее решение основано на вычислении для каждого участника гонки суммарного времени прохождения им всех кругов. Далее остается только определить минимальное значение среди вычисленных чисел, чтобы получить требуемый ответ.

Приведем фрагмент программы, реализующий описанный подход.

```
readln(n, m);
ans := 0;
ansname := '';
for i := 1 to n do begin
  readln(name);
  sum := 0;
  for j := 1 to m do begin
    read(t);
    sum := sum + t;
  end;
  if (ans = 0) or (sum <= ans) then begin
    ans := sum;
    ansname := name;
  end;
  readln;
end;
writeln(ansname);
```

Задача 2. «Дипломы»

Анализ условия данной задачи позволяет сделать вывод, что число дипломов, которые будет размещены вдоль одной из сторон квадратной доски, не превосходит $\sqrt{n}$. Докажем это методом «от противного».

Пусть вдоль вертикальной стороны квадратной доски размещено a дипломов, а вдоль горизонтальной – b дипломов. Тогда общее число дипломов составляет $a \cdot b$. Если оба числа a и b превосходят $\sqrt{n}$, то общее число дипломов будет больше, чем n, что невозможно (их число равно n).

С учетом сказанного, введем в рассмотрение величину x – число дипломов, размещенных вдоль какой-либо стороны квадратной доски. Теперь, для решения исходной задачи необходимо перебрать все значения x от 1 до $\sqrt{n}$ (округленного вверх), и для каждого их этих значений рассмотреть два случая:

когда x дипломов размещено вдоль вертикальной стороны квадратной доски или когда x дипломов размещено вдоль горизонтальной стороны. Для каждого из этих случаев по соответствующей формуле необходимо найти число дипломов, которые должны быть размещены вдоль другой стороны квадратной доски, после чего необходимо вычислить получающийся размер квадратной доски.

Отметим, что для полного решения этой задачи необходимо использовать 64-битный целочисленный тип данных, так как ответ может быть не представим в рамках 32-битного типа данных.

Существует и альтернативный подход к решению данной задачи, основанный на использовании двоичного поиска по стороне искомого квадрата. Суть его заключается в том, что для каждого значения x можно определить, поместятся ли n дипломов размером w на h в квадрат со стороной, равной x. Такое решение позволит получить ответ за время $O(1)$.

Задача 3. «Булева функция»

Данную задачу можно решать различными способами. Рассмотрим два из них. Первый способ основан на использовании идей динамического программирования, а второй – на анализе возможных вариантов вычисления функций f.

При использовании идей динамического программирования введем в рассмотрение величину $a_{i,j}$ – максимальное число единиц, которое можно получить, используя i булевых значений, при этом, результат цепного вычисления функции f от них равен j. Не сложно заметить, что в этом случае для получения ответа на исходную задачу необходимо использовать значение $a_{n,1}$.

Для вычисления значений $a_{i,j}$ можно использовать следующую формулу:

$$a_{i,j} = \sum_{u,v:\, f(u,v)=j} (a_{i-1,u} + v).$$

В этой формуле u соответствует цепному значению функции f для первых $(i-1)$ булевых значений, а v – i-му булевому значению. Поскольку значение

функции для i булевых значений должно быть равно j, то далее остается только рассмотреть случаи, когда f$(u,v)=j$.

Второй способ решения исходной задачи основан на том факте, что существует всего 16 различных вариантов вычисления функций f. Найдем для каждого варианта искомый набор булевых значений a_i.

Если f(1,1)=1, то ответом будет последовательность из одних единиц. Пусть теперь f(1,1)=0. Тогда при f(0,0)=1 искомым ответом будет последовательность из единиц с одним нулем на конце.

Пусть теперь f(0, 0) = f(1, 1) = 0. Если f(u,v) всегда равно нулю, то ответом будет сообщение “No solution”. Если теперь f(0, 1) = 1, то ответом будет последовательность из единиц и в конце этой последовательности будет 01. Наконец, если f(1, 0) = 1, то ответом будет последовательность, начинающаяся с единицы, после которой идет последовательность из нулей.

Не сложно показать, что время работы обоих решений составляет $O(n)$.

Задача 4. «Кольцевая автодорога»

При решении данной задачи в первую очередь требуется найти число окружностей, равноудаленных от четырех данных точек. Чтобы определить это число, рассмотрим все возможные варианты расположения точек внутри или снаружи окружности.

Если все четыре точки расположены внутри или снаружи окружности, то не сложно определить, что, существует окружность, на которой они могут быть размещены. В этом случае ответом будет являться слово «Infinity».

Если три точки лежат внутри или снаружи, то центр окружности будет совпадать с центром описанной окружности этих трех точек. Перебрав все возможные варианты выбора трех точек из четырех (существует четыре варианта) и для каждого варианта построив описанную окружность и ее центр, можно добавить построенные центры в список-ответ.

Теперь осталось рассмотреть варианты, когда две точки лежат внутри окружности, а две – снаружи. Рассмотрим каждый из этих вариантов, учитывая, что их три.

Если окружность равноудалена от двух точек, то и ее центр тоже будет равноудален от них. Геометрическое место таких точек – серединный перпендикуляр к отрезку, соединяющему две данные точки. Теперь осталось построить два серединных перпендикуляра к двум отрезкам – отрезку, соединяющему внутренние точки, и отрезку, соединяющему внешние точки. Если эти серединные перпендикуляры совпадают, то ответом будет являться слово «Infinity». Если они параллельны, то соответствующий вариант расположения точек внутри и снаружи окружности не может реализоваться. Если они не параллельны, то точка их пересечения является центром равноудаленной окружности. Эту точку необходимо добавить в список-ответ.

Таким образом, можно получить список, состоящий из центров окружностей. Удалив из него совпадающие центры окружностей, получим ответ, равный числу центров окружностей в получившемся списке.

Задача 5. «Миша и негатив»

Начнем решение данной задачи с определения структуры данных для хранения исходного изображения и построенного Мишиной программой негатива. Простейший анализ показывает, что для этой цели целесообразно использовать двумерные массивы символьного типа, которые обозначим `a[1..h, 1..w]` и `b[1..h, 1..w]` соответственно. Заметим также, что пиксель с координатами `(i, j)` неправильно сформирован Мишиной программой, если выполняется условие: $a[i, j] \neq \text{not}\, b[i, j]$, где `not` – операция инвертирования цвета (она сопоставляет белому цвету черный и наоборот).

Учитывая вышесказанное, а также тот факт, что по условию задачи в изображениях встречаются только пиксели двух цветов, можно сформулировать условие, определяющее ошибочные пиксели в негативе. Это условие можно записать так: $a[i, j] = b[i, j]$. Теперь, чтобы решить поставленную задачу, необходимо определить количество одинаковых символов в двух заданных описаниях изображений.

Задача 6. «Треугольник Максима»

Решение данной задачи основано на последовательном анализе каждой пары последовательных частот и выводе, что искомый ответ лежит на некотором луче.

Пусть предыдущая рассмотренная частота равна P, а текущая T, тогда следует рассмотреть четыре возможных случая:

1) $P < T$ и результат сравнения `closer` – тогда искомая частота не меньше, чем $(P+T)/2$;
2) $P < T$ и результат сравнения `further` – тогда искомая частота не больше, чем $(P+T)/2$;
3) $P > T$ и результат сравнения `closer` – тогда искомая частота не больше, чем $(P+T)/2$;
4) $P > T$ и результат сравнения `further` – тогда искомая частота не меньше, чем $(P+T)/2$.

Отдельно необходимо также рассмотреть случай $P=T$, однако в этом случае никакой результат не несет дополнительной информации.

Выполнив вышеописанный анализ, можно получить ответ на исходную задачу путем определения точки пересечения всех лучей и исходного отрезка.

Задача 7. «Производство деталей»

При решении данной задачи сначала определим формальную схему, которую будем использовать в дальнейшем. Тот факт, что между производством различных деталей нет циклических зависимостей, позволяет представить заданные характеристики производства деталей в виде ациклического ориентированного графа, в котором вершины соответствуют деталям, а дуги – зависимостям между их производством (дуга идет из вершины i в вершину j, если для производства детали с номером i необходима деталь с номером j).

Используя описанный граф, теперь можно найти какой-нибудь порядок, в котором будет осуществлено производство всех деталей. Для этого выполним топологическую сортировку этого графа, то есть, расположим вершины графа на прямой таким образом, чтобы все дуги были направлены слева направо.

Далее перейдем к нахождению порядка, при котором деталь с номером 1 будет произведена как можно быстрее.

С этой целью найдем в полученном графе все вершины, из которых достижима вершина с номером 1. Эти вершины соответствуют тем деталям, которые необходимо произвести для производства детали номер 1. Для поиска указанного набора деталей необходимо применить алгоритм поиска в глубину или в ширину для транспонированного графа, то есть, графа, в котором направление каждой дуги изменено на противоположное.

Минимальное время, необходимое для производства детали с номером 1, равно сумме времен производства ее и соответствующих деталей. Требуемый порядок производства деталей можно определить следующим образом: сначала изготовить все детали, необходимые для производства детали с номером 1 (в порядке топологической сортировки), затем – саму деталь с номером 1.

Задача 8. «Новое слово в рекламе»

Начнем решение данной задачи с некоторых обозначений и определений.

1.Пусть S — строка длиной L. Обозначим i-й символ этой строки как $S[i]$ $(1 \le i \le L)$.

2.Пусть даны k слов B_1, ..., B_k, каждое из которых имеет длину, равную L. Назовем *транспонированной конкатенацией* слов B_1, ..., B_k строку $B_1[1]...B_k[1]B_1[2]...B_k[2]...B_1[L]...B_k[L]$. Именно эта строка получается, если записать слова B_1, ..., B_k одно над другим и затем выписывать символы по столбцам слева направо, а в каждом столбце — сверху вниз.

Рассмотрим k слов B_1, ..., B_k и строку s, являющуюся подстрокой строки t — транспонированной конкатенацией этих слов. Пусть вхождение строки s в строку t начинается с символа, соответствующего j-му символу слова B_i. Тогда можно показать, что верны следующие утверждения:

1) Строка $W_1 = s[1]s[k+1]s[2k+1]...$ является подстрокой B_i, начинающейся с j-го символа B_i.

2) Строка $W_2 = s[2]s[k+2]...$ является подстрокой B_{i+1}, начинающейся с j-го символа B_{i+1}.

3) Строка $W_{k-i+1} = s[k-i+1]s[2k-i+1]...$ является подстрокой B_k, начинающейся с j-того символа B_k.

4) Строка $W_{k-i+2} = s[k-i+2]s[2k-i+2]...$ является подстрокой B_1, начинающейся с $(j+1)$-го символа B_1.

5) Строка $W_k = s[k]s[2k]...$ является подстрокой B_{i-1}, начинающейся с $(j+1)$-го символа B_{i-1}.

Заметим, что обратное также верно: если выполняются все утверждения о подпоследовательностях W_m, то s является подстрокой t.

Проиллюстрируем сказанное на примере, когда B_1 = «LGRA», B_2 = «OWDR», B_3 = «NOHD». Слово «WORDHARD» является подстрокой транспонированной конкатенации этих слов, как показано на рис. ниже).

L	G	R	A
O	W	D	R
N	O	H	D

Поскольку здесь $i = 2$, $j = 2$, то тогда W_1 = «WDR» будет являться подстрокой B_2, W_2 = «OHD» – подстрокой B_3 и W_3 = «RA» – подстрокой B_1. Существуют и вырожденные случаи, например, если $i = 1$, тогда все W_m будут являться подстроками B_m, начинающимися с позиции j (см. рис. ниже).

L	G	R	A
O	W	D	R
N	O	H	D

Теперь, пусть даны n слов B_1, ..., B_n, а также слово s, и необходимо узнать, можно ли выбрать из них k слов (одно слово можно выбрать несколько раз), чтобы s являлось подстрокой их транспонированной конкатенации. Для этого выполним следующие действия:

- Заведем двумерный массив целых величин $Z[1..L][1..k]$. Изначально этот

массив заполним нулями.

- Выпишем для слова s подпоследовательности W_1, ..., W_k. Для каждой такой подпоследовательности W_i и для каждого слова B_j найдем все индексы t, начиная с которых W_i будет входить в B_j как подстрока. Для каждого такого индекса установим $Z[t][i] = j$.
- Переберем все возможные положения начала вхождения s в транспонированную конкатенацию, то есть, все индексы (i, j), означающие, что $s[1]$ попадает в $B_i[j]$. Для каждого такого положения можно проверить, существует ли такой набор слов, что s входит в их транспонированную конкатенацию, начиная с этого положения.

Справедливость последнего утверждения можно проверить следующим образом. Действительно, если все значения $Z[j][i]$, $Z[j][i+1]$, ..., $Z[j][k]$, $Z[j+1][1]$, ..., $Z[j+1][i-1]$ не равны нулю, то слова с индексами, записанными в этих ячейках массива, образуют требуемый набор слов. Эта проверка выполняется за $O(k)$ действий для каждой пары $(i; j)$. Однако все множество проверок может быть выполнено быстрее — достаточно поддерживать число ненулевых элементов для j-го и $(j+1)$-го индексов среди требуемого подмножества ячеек массива. Если перебирать положения начала вхождения в порядке возрастания j, а среди равных — в порядке возрастания i, то требуемые значения можно обновлять за $O(1)$.

Теперь для решения задачи достаточно перебрать k от 1 до $|s|$ и проверить, существует ли для данного k ответ, используя изложенный выше алгоритм.

Проанализируем асимптотическую сложность данного решения. Для каждого k сложность этапов составляет:

- для инициализации массива — $O(L \cdot k)$;
- для подсчета вхождений W_i в B_j :

  а) при использовании простого алгоритма поиска подстроки в строке:

  $O(k \cdot (n \cdot L \cdot |s| / k)) = O(|s| \cdot n \cdot L)$;

  б) при использовании алгоритма Кнута-Морриса-Пратта:

$O(k \cdot (n \cdot L + |s| / k)) = O(k \cdot n \cdot L + |s|)$;

- для поиска ответа — $O(n \cdot L + k)$.

Сложность алгоритма для фиксированного k при использовании простого алгоритма поиска подстроки составляет $O(|s| \cdot n \cdot L)$ и при использовании алгоритма Кнута-Морриса-Пратта – $O(k \cdot n \cdot L)$. С учетом этого, сложность всего решения будет составлять $O(|s|^2 \cdot n \cdot L)$ для обоих алгоритмов.

Следует отметить, что использование более простого алгоритма поиска подстроки на практике приводит к увеличению производительности, так как константный множитель при его асимптотической оценке меньше, чем у алгоритма Кнута-Морриса-Пратта.