§3 Финальный этап: индивидуальный тур

На выполнение всех задач отводится 2 часа. Задачи имеют разную сложность и, в целом, их число избыточно для такого времени, однако достаточно подробные критерии оценки дают возможность дифференцировано оценить работы участников.

3.1 Задачи по информатике

Задача 3.1.1 (7 баллов)

Условие:

Ограничение по времени: 1 секунда

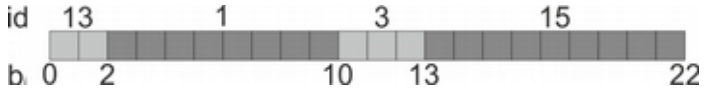
Ограничение по памяти: 32 мегабайта

При управлении сложными системами центральному серверу постоянно приходится отслеживать состояние множества компонент – будь то датчики, исполнительные устройства или другое оборудование. Рассмотрим систему, состоящую из N различных устройств. Каждое из этих устройств имеет свой уникальный целочисленный идентификатор id. Для хранения информации об -ом устройстве в памяти центрального сервера выделяется w_i байт. Память под все эти устройства выделяется одним целым блоком, размера

$$W=\sum_{i}^{\square}\square w_i.$$

В любой момент времени серверу известен адрес в памяти на начало этого блока. Отсупом -го устройства будем называть количество байт b_i, которое нужно прибавить к адресу начала блока, чтобы получить адрес i -го устройства. Отступ первого устройства b_1 всегда равен 0. Отступ каждого последующего устройства рассчитывается по формуле: $b_i=b_{i-1}+w_{i-1}$.

Каждый раз, когда серверу необходимо обратиться к устройству с определенным id, ему необходимо определить место в памяти, где хранится информация об этом устройстве. Ваша задача написать программу, которая будет рассчитывать отступ для устройств по запрашиваемым id.



Формат входных данных

В первой строке даны два целых числа: N – количество различных устройств, и K – количество запросов ($2\le N\le 10^3, 1\le K\le 10^3$).

В следующих N строках заданы по два целых числа: id_i и w_i – идентификатор и объём выделяемой памяти для -го устройства соответственно ($1\le id_i\le 10^5, 1\le w_i\le 10^5$). Устройства даны в том порядке, в котором они хранятся в выделенном блоке памяти.

Следующие K строк содержат по единственному целому числу id_k – идентификатор устройства, для которого необходимо посчитать отступ b_k.

Формат выходных данных

Для каждого из K запросов выведите число b_k в отдельной строке.

Пример

Входные данные	Выходные данные
4 6 13 2 1 8 3 3 15 9 1 13 3 13 1 15	2 0 10 0 2 13

Решение:

Для заданных ограничений задачу можно решить, выделив массив A размером 10^5+1 элементов, где индексом будет являться номер id_i устройства, а значением – накопленная сумма $W_i=\sum_{j=1}^{i-1} w_j$. Тогда для каждого id_k ответом будет являться значение $A[id_k]$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

typedef struct {
    int id;
    long long w_cum;
} Device;


bool cmp1(Device &a, Device &b)
{
    return a.id > b.id;
}
```

```
int main()
{
    int n, k;
    scanf("%d%d\n", &n, &k);
    vector <Device> devs(n);

    int w_cum = 0;
    int max_id = -1;
    for (int i = 0; i < n; i++)
    {
        devs[i].w_cum = w_cum;
        if (i)
        {
            devs[i].w_cum += devs[i - 1].w_cum;
        }
        scanf("%d%d\n", &devs[i].id, &w_cum);
        max_id = max(devs[i].id, max_id);
    }

    vector <int> id_map(max_id + 1);

    for (int i = 0; i < n; i++)
    {
        id_map[devs[i].id] = i;
    }

    for (int i = 0; i < k; i++)
    {
        int tmp;
        scanf("%d\n", &tmp);
        printf("%d\n", devs[id_map[tmp]].w_cum);
    }

    return 0;
}
```

Задача 3.1.2 (13 баллов)

Условие:

Ограничение по времени: 1 секунда

Ограничение по памяти: 32 мегабайта

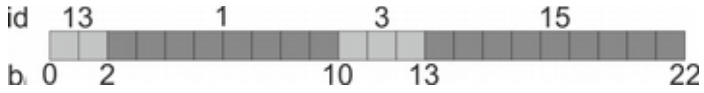
При управлении сложными системами центральному серверу постоянно приходится отслеживать состояние множества компонент – будь то датчики, исполнительные устройства или другое оборудование. Рассмотрим систему, состоящую из N различных устройств. Каждое из этих устройств имеет свой уникальный целочисленный идентификатор id. Для хранения информации об -ом устройстве в памяти центрального сервера выделяется w_i байт. Память под все эти устройства выделяется одним целым блоком, размера

$$W=\sum_{i}^{\square} \square w_i.$$

В любой момент времени серверу известен адрес в памяти на начало этого блока. Отсупом -го устройства будем называть количество байт b_i , которое нужно прибавить к адресу начала блока, чтобы получить адрес i -го устройства. Отступ первого устройства b_1 всегда равен 0. Отступ каждого последующего устройства рассчитывается по формуле:

$b_i = b_{i-1} + w_{i-1}$.

Каждый раз, когда серверу необходимо обратиться к устройству с определенным id , ему необходимо определить место в памяти, где хранится информация об этом устройстве. Ваша задача написать программу, которая будет рассчитывать отступ для устройств по запрашиваемым id .



Формат входных данных

В первой строке даны два целых числа: N – количество различных устройств, и K – количество запросов ($2 \le N \le 10^5, 1 \le K \le 3 \cdot 10^5$).

В следующих N строках заданы по два целых числа: id_i и w_i – идентификатор и объём выделяемой памяти для -го устройства соответственно ($1 \le id_i \le 10^9, 1 \le w_i \le 10^5$). Устройства даны в том порядке, в котором они хранятся в выделенном блоке памяти.

Следующие K строк содержат по единственному целому числу id_k – идентификатор устройства, для которого необходимо посчитать отступ b_k .

Формат выходных данных

Для каждого из K запросов выведите число b_k в отдельной строке.

Пример

Входные данные	**Выходные данные**
4 6 13 2 1 8 3 3 15 9 1 13 3 13 1 15	2 0 10 0 2 13

Решение:

Для заданных ограничений задачу можно решить, используя любой алгоритм поиска со сложностью $O(logN)$, где N – количество различных устройств. Например, метод дихотомии. Данный метод находит элементы в отсортированном массиве, поэтому необходимо предварительно отсортировать элементы по возрастанию величины id любым алгоритмом сортировки со сложностью $O(N \cdot logN)$. Таким образом, общая сложность

решения $O((K+N)\cdot logN)$, что укладывается в лимит по времени для заданных ограничений.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

typedef struct {
    int id;
    long long w_cum;
} Device;

int cmp1(Device &a, Device &b)
{
    return a.id < b.id;
}

int bin_search(vector <Device> &ar, int x)
{
    int l = 0;
    int r = ar.size() - 1;
    while (l <= r)
    {
        int m = (l + r) >> 1;
        if (ar[m].id == x)
        {
            return m;
        }
        else if (ar[m].id < x)
        {
            l = m + 1;
        }
        else // if(ar[m].id > x)
        {
            r = m - 1;
        }
    }
```

```
    return -1;
}

int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, k;
    fscanf(f_in, "%d%d\n", &n, &k);

    vector <Device> devs(n);
    int w_cum = 0;
    for (int i = 0; i < n; i++)
    {
        devs[i].w_cum = w_cum;
        if (i)
        {
            devs[i].w_cum += devs[i - 1].w_cum;
        }

        fscanf(f_in, "%d%d\n", &devs[i].id, &w_cum);
    }

    sort(devs.begin(), devs.end(), cmp1);
    for (int i = 0; i < k; i++)
    {
        int id;
        fscanf(f_in, "%d\n", &id);
        int idx = bin_search(devs, id);
        fprintf(f_out, "%lld\n", devs[idx].w_cum);
    }

    return 0;
}
```

Задача 3.1.3 (7 баллов)

Условие:

Ограничение по времени: 1 секунда

Ограничение по памяти: 32 мегабайта

В системе управления иногда возникают простои, когда никаких важных задач выполнять не надо. Это свободное время используется для выполнения некоторых второстепенных задач. Время, которое система гарантировано может потратить на второстепенные задачи, всегда известно и равно T. Всего возможных второстепенных задач N. Для каждой из них известно время выполнения t и некоторая целочисленная величина b, характеризующая пользу, которую несет с собой выполнение этой задачи.

Вам необходимо по известному списку задач определить, какую максимальную суммарную пользу может принести система, оптимальным образом выбрав задачи, которые можно выполнить за суммарное время, не превосходящее T.

Польза от выполнения задачи засчитывается только для полностью завершенных задач. Задачи выполняются последовательно, после завершения одной – сразу начинается выполнение второй. Каждая задача может быть выполнена только один раз.

Формат входных данных

В первой строке даны два целых числа: N и T – количество доступных задач и время на выполнение второстепенных задач ($1 \le N \le 20, 1 \le T \le 10^3$).

В следующих N строках заданы по два целых числа: t_i и b_i – время выполнения и характеристика пользы -ой задачи ($1 \le t_i \le 10^3, 1 \le b_i \le 10^5$).

Формат выходных данных

В ответ выведите единственное число B – максимальную возможную суммарную пользу, которую можно получить, выполнив оптимальный набор задач из списка за суммарное время не превосходящее T .

Пример

Входные данные	**Выходные данные**
5 7 2 8 3 10 3 8 5 5 3 7	18

Решение:

Учитывая, что количество доступных задач N не превосходит 20, различных комбинаций задач, которые можно выполнить без учёта ограничения времени T , равно $2^{20} = 1048576\,10^6$. Такое количество комбинаций можно полностью перебрать за отведенное время (1 секунду). В процессе перебора среди всех наборов задач, суммарное время на выполнение которых не превышает T , необходимо выбрать тот, который даёт наибольшую среди всех величину суммарной пользы B . Это и будет ответом.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;
```

```
int main()
{
    int n, t;
    scanf("%d%d\n", &n, &t);
    vector<pair<int, int> > ar(n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d\n", &ar[i].second, &ar[i].first);
    }

    int total_b = 0;

    for (int i = 1; i < (1 << n); i++)
    {
        int w = 0, b = 0;
        for (int j = 0; (1 << j) < i; j++)
        {
            if (i & (1 << j))
            {
                w += ar[j].second;
                b += ar[j].first;
            }
        }
        if (w <= t)
            total_b = max(total_b, b);
    }

    printf("%d\n", total_b);

    return 0;
}
```

Задача 3.1.4 (13 баллов)

Условие:

Ограничение по времени: 1 секунда

Ограничение по памяти: 32 мегабайта

В системе управления иногда возникают простои, когда никаких важных задач выполнять не надо. Это свободное время используется для выполнения некоторых второстепенных задач. Время, которое система гарантировано может потратить на второстепенные задачи, всегда известно и равно T. Всего возможных второстепенных задач N. Для каждой из них известно время выполнения t и некоторая целочисленная величина b, характеризующая пользу, которую несет с собой выполнение этой задачи.
Вам необходимо по известному списку задач определить, какую максимальную суммарную пользу может принести система, оптимальным образом выбрав задачи, которые можно выполнить за суммарное время, не превосходящее T.

Польза от выполнения задачи засчитывается только для полностью завершенных задач. Задачи выполняются последовательно, после завершения одной – сразу начинается выполнение второй. Каждая задача может быть выполнена только один раз.

Формат входных данных

В первой строке даны два целых числа: N и T – количество доступных задач и время на выполнение второстепенных задач ($1 \le N \le 10^3, 1 \le T \le 10^3$).

В следующих N строках заданы по два целых числа: t_i и b_i – время выполнения и характеристика пользы -ой задачи ($1 \le t_i \le 10^3, 1 \le b_i \le 10^5$).

Формат выходных данных

В ответ выведите единственное число B – максимальную возможную суммарную пользу, которую можно получить, выполнив оптимальный набор задач из списка за суммарное время не превосходящее T .

Пример

Входные данные	**Выходные данные**
5 7 2 8 3 10 3 8 5 5 3 7	18

Решение:

Данная задача является классической задачей динамического программирования – задача о рюкзаке. Пусть $B[i][t]$ – максимальная польза, которую можно получить, выполнив какое-либо подмножество из -первых доступных задач за время, не превышающее t . Запишем рекуррентные соотношения, справедливые для $B[i][t]$:

- $B[0][t] = 0$
- $B[i][t] = B[i-1][t]$, если $t_i > t$
- $B[i][t] = max\left(B[i-1][t], B[i-1][t-t_i] + b_i\right)$, если $t_i < t$.

Ответом на задачу, таким образом, будет являться значение $B[N][T]$. Сложность расчёта этой величины есть $O(N \cdot T)$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;
```

```
int main()
{
    int T, n;
    scanf("%d%d", &n, &T);
    vector <int> t(n);
    vector <int> b(n);
    vector <vector <int> > dp(n + 1, vector<int>(T + 1, 0));

    for (int i = 0; i < n; i++)
    {
        scanf("%d%d", &t[i], &b[i]);
    }

    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= T; j++)
        {
            if (t[i - 1] > j)
            {
                dp[i][j] = dp[i - 1][j];
            }
            else
            {
                 dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - t[i
- 1]] + b[i - 1]);
            }
        }
    }

    printf("%d\n", dp[n][T]);

    return 0;
}
```

Задача 3.1.5 (7 баллов)

Условие:

Ограничение по времени: 1 секунда

Ограничение по памяти: 32 мегабайта

В данной задаче рассматривается история выполнения различных операций некой системы управления за некоторый промежуток времени. Известно, что за рассматриваемое время система успела N раз выполнить определенные операции. Большинство из них были выполнены четное число раз, но K операций были выполнены нечетное количество раз.

Каждая операция имеет свой целочисленный идентификационный номер id .

Ваша задача по истории выполненных операций определить K идентификационных номеров id_k тех операций, которые выполнились нечетное количество раз.

Формат входных данных

В первой строке даны два целых числа N и K – общее количество выполненных за день операций и количество операций, выполненных нечетное число раз соответственно ($2 \le N \le 10^7, K = 1$).
В следующих N строках содержится по одному целому числу id_i – идентификатор выполненной операции ($1 \le id_i \le 10^9$).

Формат выходных данных
В единственной строке через пробел выведите K чисел id_k в порядке возрастания – идентификаторы операций, выполненных нечетное число раз.

Пример

Входные данные	**Выходные данные**
9 1 7 3 5 5 3 7 2 2 7	7

Решение:
Обратим внимание на ограничение по памяти в 32 МБ при количестве входных данных порядка 10^7 . Учитывая, что идентификаторы id могут быть в лучшем случае представлены в виде 32-битного целого числа (4 байта), для хранения всех идентификаторов потребуется порядка 40 МБ памяти.
Для решения данной задачи воспользуемся следующими свойствами операции суммирования по модулю 2 ($\oplus$):

- $a \oplus a = 0$
- $0 \oplus a = a$
- $a \oplus b = b \oplus a$
- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

Эти же свойства справедливы для эквивалентной битовой операции «исключающего ИЛИ». Из этих свойств, в частности, следует, что побитовая сумма по модулю 2 для всех входных идентификаторов id_i будет равно сумме по модулю два только тех идентификаторов, которые встретились нечётное число раз. Учитывая, что в условиях данной задачи такой идентификатор единственен, эта сумма по модулю 2 всех идентификаторов и будет являться ответом на задачу. При этом для её вычисления нет необходимости хранить все входящие числа; размер необходимой памяти есть $O(1)$, сложность решения есть $O(N)$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;
```

```
int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, k;
    fscanf(f_in, "%d%d", &n, &k);

    int x = 0;
    for (int i = 0; i < n; i++)
    {
        int tmp;
        fscanf(f_in, "%d", &tmp);
        x ^= tmp;
    }

    fprintf(f_out, "%d\n", x);

    return 0;
}
```

Задача 3.1.6 (20 баллов)
Условие:
Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

В данной задаче рассматривается история выполнения различных операций некой системы управления за некоторый промежуток времени. Известно, что за рассматриваемое время система успела N раз выполнить определенные операции. Большинство из них были выполнены четное число раз, но K операций были выполнены нечетное количество раз.

Каждая операция имеет свой целочисленный идентификационный номер id .
Ваша задача по истории выполненных операций определить K идентификационных номеров id_k тех операций, которые выполнились нечетное количество раз.

Формат входных данных
В первой строке даны два целых числа N и K – общее количество выполненных за день операций и количество операций, выполненных нечетное число раз соответственно ($2 \le N \le 10^7, K = 2$).
В следующих N строках содержится по одному целому числу id_i – идентификатор выполненной операции ($1 \le id_i \le 10^9$).

Формат выходных данных
В единственной строке через пробел выведите K чисел id_k в порядке возрастания – идентификаторы операций, выполненных нечетное число раз.

Пример

Входные данные	**Выходные данные**

8 2 3 2 8 2 2 8 8 3	2 8

Решение:

Обратим внимание на ограничение по памяти в 32 МБ при количестве входных данных порядка 10^7 . Учитывая, что идентификаторы id могут быть в лучшем случае представлены в виде 32-битного целого числа (4 байта), для хранения всех идентификаторов потребуется порядка 40 МБ памяти.

Для решения данной задачи воспользуемся следующими свойствами операции суммирования по модулю 2 ($\oplus$):

- $a \oplus a = 0$
- $0 \oplus a = a$
- $a \oplus b = b \oplus a$
- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

Эти же свойства справедливы для эквивалентной битовой операции «исключающего ИЛИ». Из этих свойств, в частности, следует, что побитовая сумма по модулю 2 для всех входных идентификаторов id_i будет равно сумме по модулю два только тех идентификаторов, которые встретились нечётное число раз. В условиях данной задачи таких идентификаторов ровно два.

Обратим внимание, что если два числа не равны друг другу, то в битовом представлении они так же будут отличаться как минимум в одном бите. Кроме общей суммы по модулю 2 будем так же собирать 30 (количество бит, необходимых для представления числа 10^9) дополнительных сумм в массив S , где каждый элемент $S[j]$ будет хранить сумму по модулю 2 только тех элементов, которые имеют ненулевой -ый бит в двоичном представлении. Так как два искомых идентификатора будут отличаться хотя бы в одном из разрядов битового представления, хотя бы один элемент массива S после выполнения всех суммирований будет отличаться от общей суммы по модулю 2 всех идентификаторов. Этот элемент и будет являться одним из искомых чисел. Второй число получается как результат суммирования по модулю два числа, найденного в массиве S и общей суммы по модулю 2 всех идентификаторов.

Для решения задачи нет необходимости хранить все входящие числа; размер необходимой памяти есть $O(\lceil \log_2 max(id) \rceil)$, сложность решения есть $O(N \cdot \lceil \log_2 max(id) \rceil)$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

int main()
```

```
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, k;
    fscanf(f_in, "%d%d", &n, &k);

    int ar[31] = {};
    int x = 0;
    for (int i = 0; i < n; i++)
    {
        int tmp;
        fscanf(f_in, "%d", &tmp);
        x ^= tmp;
        for (int shift = 0; shift < 31; shift++)
        {
            if (tmp & (1 << shift))
            {
                ar[shift] ^= tmp;
            }
        }
    }

    int a = -1, b = -1;
    for (int i = 0; i < 31; i++)
    {
        if (ar[i])
        {
            if (a == -1)
            {
                a = ar[i];
            }
            else if (a != ar[i])
            {
                b = ar[i];
                break;
            }
        }
    }

    if (a == x)
    {
        a ^= b;
    }
    else if (b == x)
    {
        b ^= a;
    }
    if (b < a)
    {
```

```
            swap(a, b);
        }

        fprintf(f_out, "%d %d\n", a, b);

        return 0;
    }
```

Задача 3.1.7 (33 балла)
Условие:
Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

Умный беспилотный робот пылесос, убирающий грязь с пола автономного космического транспорта, всегда передвигается по заданной последовательности из N векторов. Сначала пылесос находиться в точке $(0;0)$, соответствующей положению его док-станции. Первым шагом он перемещается по вектору v_1. Его координаты становятся равны $(v_{1,x}, v_{1,y})$. Отсюда он следует по вектору v_2 в точку $(v_{1,x}+v_{2,x}, v_{1,y}+v_{2,y})$. Далее по вектору v_3 и т.д. до вектора v_N. После этого робот возвращается на док-станцию. «Почему же этот пылесос называется умным, если ездит он исключительно по заданной последовательности векторов?» – удивитесь вы. Оказывается, когда робот собирается начать движение по очередному вектору, он может выбрать двигаться ему по вектору v_i или по вектору $-v_i$ (в противоположном направлении). Известно, что длина всех векторов v_i не превосходит D.

Переместившись по всем доступным векторам умный пылесос должен иметь связь (конечно же, беспроводную) со своей док-станцией, чтобы по окончании обхода знать, куда вернуться на зарядку. Для обеспечения связи между обоими устройствами, пылесос должен находится на расстоянии не большем чем $\sqrt{2}\cdot D$ от док-станции.

Ваша задача, написать программу, которая должна определить направление движения пылесоса вдоль каждого вектора таким образом, чтобы в конце пути он не отдалился от док-станции с координатами $(0;0)$, на расстояние большее чем $\sqrt{2}\cdot D$. Обратите внимание, что это условие должно выполниться только после прохода по всем N векторам. Промежуточные координаты пылесоса могут превосходить эту величину.

Формат входных данных
В первой строке даны два целых числа: количество векторов N, и расстояние D ($2\le N\le 10^4, 1\le D\le 100$).
В следующих N строках содержится по два целых числа x и y – координаты очередного вектора ($0\le\sqrt{x^2+y^2}\le D$).

Формат выходных данных
Если выбрать правильную последовательность направлений векторов таким образом, чтобы пылесос всегда был на связи с док-станцией невозможно, выведите единственное слово “Impossible” (без кавычек).
Если найти правильную последовательность направлений возможно, в первой строке выведите слово “Possible” (без кавычек), а во второй строке последовательность символов “F” и “B” (без кавычек) по следующему правилу. Если для очередного вектора выбрано направление совпадающее с направлением вектора, выводится символ “F” (от англ. –

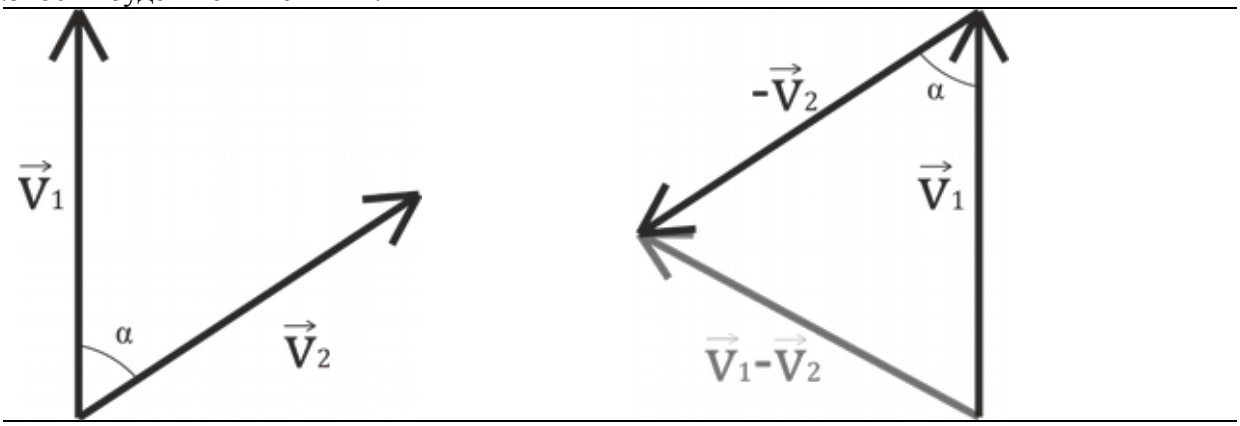
Forward). Если же выбрано направление противоположное направлению вектора, выводится символ “B” (от англ. – Backward).

Пример

Входные данные	**Выходные данные**
5 7 0 7 -7 0 1 -1 -1 1 -6 3	Possible FFFFB

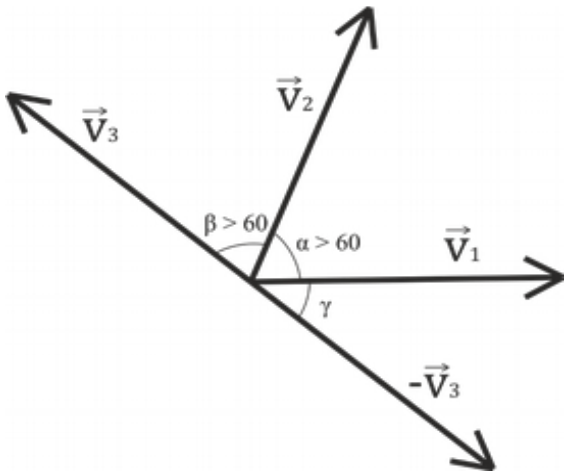
Решение:

Рассмотрим пары векторов, угол между которыми не превосходит $90°$. Если угол между векторами больше $90°$, приведём их к необходимому нам условию, взяв один из векторов со знаком минус. Пусть длина всех рассматриваемых векторов равна D. Если угол между парой векторов длины D равен $60°$, то длина вектора, являющегося разностью этих векторов, будет равен D. Если угол меньше $60°$, то длина вектора их разности будет меньше D.



С увеличением угла между парой векторов (вплоть до $90°$), разность этих векторов так же будет увеличиваться. Когда угол между векторами равен $90°$, с какими бы знаками не были взяты векторы, длина вектора их суммы или разности будет равна $\sqrt{2}\cdot D$ (как гипотенуза равнобедренного прямоугольного треугольника). Таким образом, из двух векторов длины D гарантировано можно получить суммарный вектор длины не больше $\sqrt{2}\cdot D$, правильно расставив знаки плюс или минус (**утверждение 1**).

Пусть угол между парой векторов длины D больше $60°$, но не превосходит $90°$. Невозможно найти третий вектор длины D такой, что наименьший угол между ним и каждым из двух ранее рассмотренных векторов был больше $60°$ (при условии, что третий вектор может быть взять как со знаком +, так и со знаком -).



На рисунке проиллюстрирован возможный выбор третьего вектора v_3 таким образом, чтобы углы между каждой парой векторов были больше $60°$. Учитывая, что $\alpha > 60°$ и $\beta > 60°$, взяв вектор $-v_3$ (со знаком минус), можно получить угол γ между векторами v_1 и $-v_3$ равный $\gamma = 180° - (\alpha + \beta) < 60°$.

Таким образом, среди трёх векторов длины D всегда можно выбрать 2 вектора таким образом, чтобы получить суммарный вектор (с правильно расставленными знаками плюс и минус), длина которого не превысит D (**утверждение 2**).

Рассмотренный случай, где все вектора имеют длину D, является граничным случаем для заданных условий. В случае, когда длина векторов произвольная, но не превосходит D, утверждения 1 и 2 остаются справедливыми.

Воспользовавшись утверждением 2, переберём тройки векторов из входных данных. В каждой тройке векторов найдём два вектора, сумма (с правильно расставленными знаками) которых даст вектор длины не больше D. Объединим эти два вектора в один. Будем проделывать это до тех пор, пока не останется только два вектора. Согласно утверждению 1, любые два вектора длины не больше D можно сложить таким образом, чтобы длина полученного вектора не превосходила $\sqrt{2} \cdot D$, что и требуется по условию задачи.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

typedef struct vec2d {
    int x;
    int y;
    int length()
    {
```

```
        return x * x + y * y;
    }
} vec2d;

typedef struct node {
    bool r_sign;
    node *left;
    node *right;
    vec2d vec;
    int idx;

    node()
    {
        r_sign = true;
        left = 0;
        right = 0;
        vec.x = 0;
        vec.y = 0;
    }
} node;

vec2d sum_vec(const vec2d &a, const vec2d &b)
{
    vec2d vec;
    vec.x = a.x + b.x;
    vec.y = a.y + b.y;

    return vec;
}

vec2d dif_vec(const vec2d &a, const vec2d &b)
{
    vec2d vec;
    vec.x = a.x - b.x;
    vec.y = a.y - b.y;

    return vec;
}

bool sign;
bool signs[1000000];

void dfs(const node *nd)
{
    if (nd->left == 0 && nd->right == 0)
    {
        signs[nd->idx] = sign;
        return;
    }
    if (nd->left != 0)
```

```
    {
        dfs(nd->left);
    }
    if (nd->right != 0)
    {
        bool old_sign = sign;
        if (!sign)
            sign = !nd->r_sign;
        else
            sign = nd->r_sign;
        dfs(nd->right);
        sign = old_sign;
    }
}

int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, l;
    fscanf(f_in, "%d\n%d", &n, &l);

    int maxl = l * l * 2;
    l *= l;

    vector <node *> tree;
    vector <node *> tree_org;
    tree.reserve(n);

    for (int i = 0; i < n; ++i)
    {
        vec2d v = {};
        fscanf(f_in, "%d%d\n", &v.x, &v.y);
        node *nd = new node;
        nd->vec = v;
        nd->idx = i;
        tree.push_back(nd);
        tree_org.push_back(nd);
    }

    while (tree.size() > 2)
    {
        vector <node *> tree_tmp;
        tree_tmp.reserve(tree.size());
        for (int i = 0; i < tree.size() - 1; ++i)
        {
            node *nd_cur = tree[i];
            for (int j = i + 1; j < tree.size(); j++)
            {
```

```
                node *nd_next = tree[j];

                                if (sum_vec(nd_cur->vec, nd_next-
>vec).length() <= l)
                {
                    node *nd = new node;
                       nd->vec = sum_vec(nd_cur->vec, nd_next-
>vec);
                    nd->left = nd_cur;
                    nd->right = nd_next;
                    nd->r_sign = true;
                    nd_cur = nd;
                }
                       else if (dif_vec(nd_cur->vec, nd_next-
>vec).length() <= l)
                {
                    node *nd = new node;
                       nd->vec = dif_vec(nd_cur->vec, nd_next-
>vec);
                    nd->left = nd_cur;
                    nd->right = nd_next;
                    nd->r_sign = false;
                    nd_cur = nd;
                }
                else
                {
                    tree_tmp.push_back(nd_next);
                }
            }
            tree_tmp.push_back(nd_cur);
            if (tree_tmp.size() != tree.size())
            {
                break;
            }
        }
        tree.swap(tree_tmp);
    }

    node root;
    if (tree.size() > 1)
    {
        node *nd_cur = tree[0];
        node *nd_next = tree[1];

           if (sum_vec(nd_cur->vec, nd_next->vec).length() <=
dif_vec(nd_cur->vec, nd_next->vec).length())
        {
            root.vec = sum_vec(nd_cur->vec, nd_next->vec);
            root.left = nd_cur;
            root.right = nd_next;
```

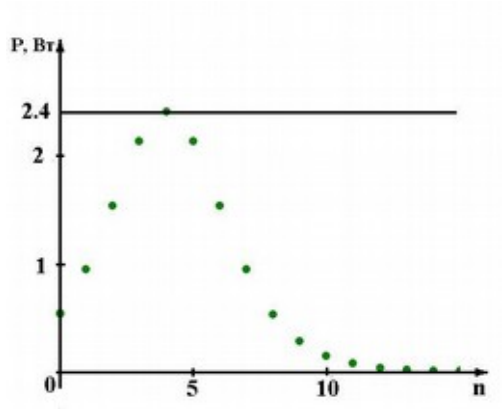
```
                root.r_sign = true;
            }
            else
            {
                root.vec = dif_vec(nd_cur->vec, nd_next->vec);
                root.left = nd_cur;
                root.right = nd_next;
                root.r_sign = false;
            }
        }
        else
        {
            root = *tree[0];
        }

        fprintf(f_out, "Possible\n");

        sign = true;
        dfs(&root);
        for (int i = 0; i < n; i++)
        {
            fprintf(f_out, signs[i] ? "F" : "B");
        }

        return 0;
    }
```

3.2 Задачи по физике

Задача 3.2.1 (32 балла)

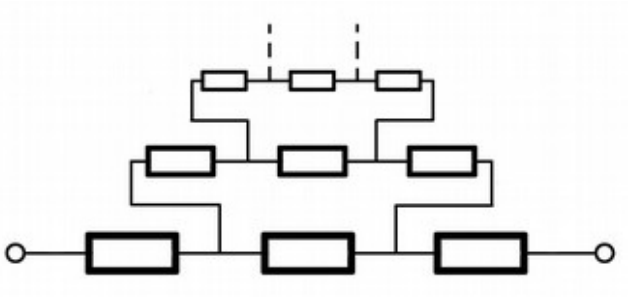
Условие:

Фрактальная цепь устроена так, как изображено на рисунке. Длина резисторов при каждом следующем шаге («строчке» на рисунке) уменьшается вдвое, а сечение остается неизменным. Исследователи присоединили эту цепь к аккумулятору с ЭДС 42В и измерили выделяющуюся на цепи мощность. Затем нарастили цепь на одну «строчку» снизу (в соответствии с правилом изменения геометрических размеров длина элементов строчки в два раза больше предыдущей нижней строчки), подключили к тому же источнику и снова измерили получившуюся мощность. Проделав так несколько раз, исследователи получили график зависимости выделяющейся на цепи мощности от числа подключенных дополнительно «строчек» (точка 0 соответствует исходной цепи). По этим данным найдите сопротивление одного элемента нижней «строчки» в исходной (до начала присоединения) цепи, если все элементы в одной «строке» одинаковые.



Решение:

Вообще говоря, мощность, выделяющуюся на бесконечной цепи вряд ли возможно измерить, однако любая реальная цепь конечна и ограничена своими размерами или минимальным размером элемента. При достаточно большом числе элементов цепи, ее сопротивление можно оценить приближением бесконечной цепи.



Выделяющаяся мощность зависит от сопротивления цепи и сопротивления источника:

$P=\frac{U^2 R_ц}{\left(R_ц+r\right)^2}$, где $R_ц$ - сопротивление цепи, а r – сопротивление источника.

Эти сопротивления можно найти, решив систему уравнений для тех или иных точек, однако можно использовать соотношение для максимума выделяющейся мощности:

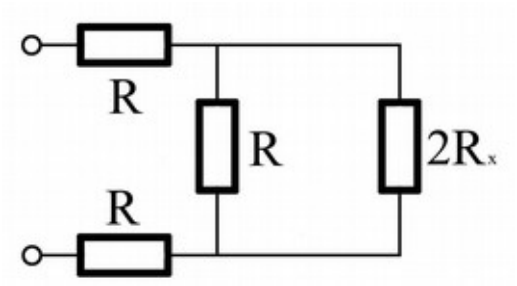
$$P_{max}=\frac{U^2}{4R_ц}$$

Это выражением можно получить, например, продифференцировав эту функцию мощности по R_x , и, найдя нули производной, обнаружить, что максимум достигается при $R_x=r$, соответственно.

Напряжение и максимальная мощность нам известны из условия, соответственно, необходимо вычислить сопротивление цепи.

Заметим, что при добавлении дополнительного элемента цепь изменяется на точно такую же, в которой размеры всех элементов увеличиваются вдвое.
При уменьшении вдвое геометрических размеров в два раза сопротивление резистора увеличивается в два раза, т.к. сопротивление прямо пропорционально длине и обратно пропорционально площади элемента.

При этом, цепь без первой «строчки» во всем подобна исходной цепи, кроме того, что сопротивление каждого резистора в ней в два раза больше, чем сопротивление резисторов в исходной цепи. Тогда полное сопротивление такой цепи будет в два раза больше, чем у исходной. Тогда исходную цепь можно представить так, как представлено на рисунке, где R_x - сопротивление исходной цепи.



Для сопротивления этой цепи получаем уравнение на сопротивление:

$$R_x=2R+\frac{2RR_x}{R+2R_x}$$

Решая которое получаем $R_x=\frac{5+\sqrt{41}}{4}R$.

Теперь заметим, что максимальная мощность соответствует подключению 4 дополнительных строчек, а каждое подключение уменьшает сопротивление в 2 раза.

Тогда $R_ц = \frac{U^2}{4P_{max}} = \frac{R_x}{2^4} = \frac{R_x}{16}$

Откуда

$$R = \frac{4U^2}{P_{max}\left(\frac{5+\sqrt{41}}{4}\right)} \approx 84 \quad \text{Ом}$$

При этом надо иметь в виду, что погрешность графика и погрешность использования приближения с бесконечной цепью вообще говоря дает погрешность ответа в ± 2 Ома.

Ответ:

$$R = \frac{4U^2}{P_{max}\left(\frac{5+\sqrt{41}}{4}\right)} \approx 84 \quad \text{Ом}$$

.

Критерии оценки:

- Записано выражение для полезной мощности – 4 балла
- Если не учтено сопротивление ЭДС – 2 балла за полную мощность в цепи.
- Сделано предположение об изменении сопротивления элемента цепи при изменении размеров – 4 балл.
- Сделано предположении об изменении сопротивления всей цепи при добавлении «строчки» - до 8 баллов
- Сделано предположение о равенстве сопротивления цепи и максимальной мощности или построена система уравнений для R и r – 8 балла.
- Получен правильный ответ в рамках допустимой погрешности – 8 балла
- Или получен правильный ответ в рамках погрешности в 5Ом – 4 балл.
- Решения удовлетворяющего критериям, приведенным выше, нет, но приведены разумные рассуждения направленные на решение задачи - 0.5 балла.

Задача 3.2.2 (5 баллов)

Условие:

При включении к аккумулятору двух одинаковых приборов параллельно, выделяющаяся на них полезная мощность оказалась на 68,1% больше, чем при подключении к этому аккумулятору одного прибора, а при включении последовательно с ними пробного сопротивления R_0 = 100 Ом выделяющаяся на приборах мощность оказалась на 76,4% меньше, чем при подключении одного прибора. Найдите сопротивление приборов.

Решение:

Запишем мощность выделяющуюся на одном прибое R, с учетом внутреннего сопротивления ЭДС r.

$$P_1 = \frac{U^2 R}{(R+r)^2}$$

На двух приборах, подключенных параллельно:

$$P_2 = \frac{U^2 \frac{R}{2}}{\left(\frac{R}{2} + r\right)^2}$$

На двух приборах и пробном сопротивлении:

$$P_3 = \frac{U^2 \frac{R}{2}}{\left(\frac{R}{2} + R_{пр} + r\right)^2}$$

Из условия известно, что:

$$\frac{P_2}{P_1} = (1 + 0{,}681) = 1{,}681 \quad , \qquad \frac{P_3}{P_1} = (1 - 0{,}764) = 0{,}236$$

Используя выражения выше:

$$\begin{cases} \frac{P_2}{P_1} = \frac{(R+r)^2}{2\left(\frac{R}{2} + r\right)^2} = 1{,}681 \\ \frac{P_3}{P_1} = \frac{(R+r)^2}{2\left(\frac{R}{2} + R_{пр} + r\right)^2} = 1{,}236 \end{cases}$$

Решая получившуюся систему относительно R и r получаем:

$$R = \frac{\frac{0{,}486\sqrt{2}}{0{,}486\sqrt{2} - 1}}{\frac{1 - \frac{0{,}486\sqrt{2}}{2}}{0{,}486\sqrt{2} - 1} - \frac{1 - \frac{1.296\sqrt{2}}{2}}{1{,}296\sqrt{2} - 1}} R_{пр} = 100\,Ом$$

Ответ: R = 100 Ом.

Критерии оценки:

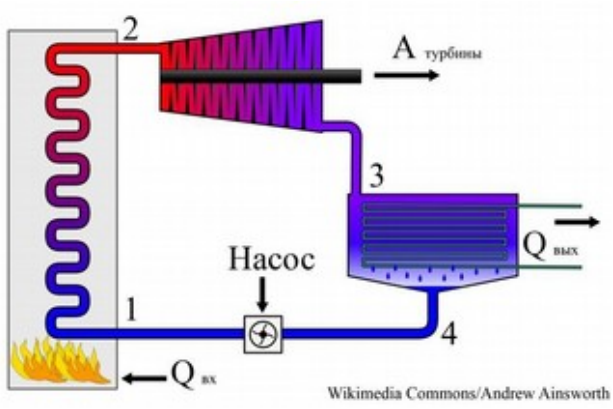
- Построена или описана схема подключений – 2 балла
- Правильно записана полезная мощность, выделяющаяся в каждом случае – 6 баллов (по 2 балла за случай)
- Составлена система уравнений, приводящая к ответу – 6 баллов
- Получено выражение для правильного ответа – 4 балла.
- Получен правильный ответ – 2 балла
- Решения удовлетворяющего критериям, приведенным выше, нет, но приведены разумные рассуждения, направленные на решение задачи - 2 балла.

Задача 3.2.3 (9 баллов)

Условие:

Во многих энергетических установках для получения полезной работы используется паровая турбина, работающая по циклу Ренкина. Схема установки работающей по такому циклу показана на рисунке. На этапе 1-2 вода испаряется и нагревается при постоянном давлении, 2-3 - теплоизолированная турбина, в которой, расширяясь, пар совершает работу. На этапе 3-4 пар конденсируется и в специальном резервуаре остывает, а затем на этапе 4-1

холодная вода по теплоизолированному трубопроводу с помощью помпы сжимается до начального давления и объема и подается в нагреватель. Считайте, что температуры в цикле меняются от 28˚C до 550˚C, а давления от 4кПа до 16Мпа. Постройте цикл Ренкина на диаграмме (PV) и оцените, во сколько раз полезная работа, полученная с помощью такой турбины, окажется меньше полезной работы, совершенной в соответствующей идеальной машине при одинаковых затратах тепла, если на нагрев пара пошло всего 53% подведенного к системе тепла.



Указание: на адиабате давление и объем связаны уравнением $pV^k = const$, причем для водяного пара показатель адиабаты равен 4/3.

Решение:

Цикл Ренкина представляет собой цикл, состоящий из двух адиабат и двух изобар.

Фактически в задаче спрашивается отношение КПД предложенного цикла Ренкина и КПД цикла Карно, работающего на предложенных температурах.

КПД цикла Карно может быть легко найден по известной формуле:

$\eta_{кр} = 1 - \frac{T_2}{T_1}$, где $T_2 - температура\ холодильника$, $а\ T_1 - нагревателя$.

В нашем случае $\eta_{кр} = 1 - \frac{28+273}{550+273} = 0{,}634$

Теперь найдем КПД цикла Ренкина. Работа в цикле совершается только перегретым паром и только его вклад нужно учитывать при расчете работы в КПД, однако теплота и достаточно значительная подводится для нагревания и испарения воды, после ее конденсации, по этому в подведенной теплоте необходимо учесть соответствующую поправку.
С учетом этого КПД может быть записан как:

$\eta = \frac{A}{\frac{Q_{подв}}{0{,}53}} = \frac{0{,}53\,A}{Q_{подв}}$, где $Q_{подв}$ - теплота, которая подводится к пару, а A – работа пара.

Вычислим этот КПД без учета поправочного коэффициента. Поскольку на адиабатах не подводится тепло, вся работа может быть вычислена через изменение теплоты на изобарах.

$A = Q_{12} - Q_{34}$, а эти теплоты в свою очередь можно определить через молярную теплоемкость при постоянном давлении для 3 атомного газа: $c_p = 4R$:

$Q_{12} = 4Rv\Delta T_{12}$ и $Q_{34} = 4Rv\Delta T_{34}$

Используя закон Менделеева-Клапейрона эти выражения можно переписать как:

$Q_{12} = 4P_2(V_2 - V_1)$ и $Q_{34} = 4P_4(V_3 - V_4)$. Здесь учтено, что $P_2 = P_1$ и $P_4 = P_3$

Тогда КПД можно записать, как:

$$\eta=\frac{0{,}53\,A}{Q_{подв}}=0{,}53\,\frac{Q_{12}-Q_{34}}{Q_{12}}=0{,}53\,\frac{4\,P_2(V_2-V_1)-4\,P_4(V_3-V_4)}{4\,P_2(V_2-V_1)}=0{,}53\left(1-\frac{P_4(V_3-V_4)}{P_2(V_2-V_1)}\right)$$

Поскольку отношение давлений мы знаем, для получения ответа необходимо выразить неизвестное отношение объемов. Воспользуемся для этого уравнением адиабаты.

$$\frac{P_1V_1^{\gamma}}{P_4V_4^{\gamma}}=1 \quad \text{и} \quad \frac{P_2V_2^{\gamma}}{P_3V_3^{\gamma}}=1$$

Откуда с учетом $P_2=P_1$ и $P_4=P_3$:

$$\frac{V_1}{V_4}=\left(\frac{P_4}{P_1}\right)^{\frac{1}{\gamma}} \quad \text{и} \quad \frac{V_2}{V_3}=\left(\frac{P_4}{P_1}\right)^{\frac{1}{\gamma}} \quad \text{или} \quad V_4=aV_1 \quad \text{и} \quad V_3=aV_2 \text{ , где } \quad a=\left(\frac{P_4}{P_1}\right)^{\frac{-1}{\gamma}}=518.7$$

Тогда искомый КПД может быть записан в виде:

$$\eta=0{,}53\left(1-\frac{P_4}{P_2}\frac{a(V_2-V_1)}{(V_2-V_1)}\right)=0.53\left(1-a\frac{P_4}{P_2}\right)=0.463$$

Тогда отношение полезных работ, равное отношению КПД оказывается:

$$\frac{A_{Ренкин}}{A_{Карно}}=\frac{0{,}463}{0{,}634}=0{,}731$$

Ответ: отношение работ равно 0,731

Критерии оценки:

- Построена диаграмма цикла Ренкина – 4 балла (по 1 за каждый правильный участок).
- Сделан вывод о том, что отношение работ равно отношению КПД – 4 балла.
- Вычислен КПД цикла Карно – 4 балла.
- Выражена работа и/или количество теплоты, подведенное к пару в этом процессе. – 4 балла.
- Записано и использовано в вычислениях уравнение адиабаты – 4 балла
- Получено выражение для КПД цикла Ренкина – 8 баллов.
- Учтено, что не вся подводимая энергия идет на нагрев пара, а часть необходима для нагрева и испарения воды – 4 балла.
- Получен правильный ответ – 4 балла.
- Решения удовлетворяющего критериям, приведенным выше, нет, но приведены разумные рассуждения направленные на решение задачи - 0.5 балла.

Задача 3.2.4 (12 баллов)

Условие:

Солнечные электростанции башенного типа представляют собой башню, на вершине которой находится резервуар с теплоносителем, окруженную зеркалами гелиостатами, которые поворачиваются вслед за Солнцем и отражают его свет так, чтобы отражение попадало в резервуар с теплоносителем на вершине башни. Теплоноситель поддерживают при температуре порядка 290˚С перед подачей в резервуар, где он нагревается до 590 ˚С. Считайте, что зеркала отражают 100% света, падающего на них, точно в резервуар, резервуар

поглощает 95% падающего света, мощность солнечного излучения линейно растет до максимума в течение рассвета, а затем вечером падает обратно до нуля в течение заката (по 1 часу), а в остальное время светового дня остается неизменной. (14 часов), теплоноситель всегда остается в жидком состоянии. Оцените массу теплоносителя, который проходит через башню за световой день. Суммарная площадь зеркал 300м^2, удельная теплоемкость теплоносителя с = 4кДж/кг˚С. Поглощением солнечной энергии в атмосфере пренебрегите.

Солнечная постоянная - суммарный поток солнечного излучения, проходящий за единицу времени через единичную площадку, ориентированную перпендикулярно потоку, на расстоянии одной астрономической единицы от Солнца вне земной атмосферы. L = 1367 Вт/м^2,

Решение:

Рассчитаем количество энергии, которое передаст Солнце нагревателю. Пренебрегая потерями в атмосфере (хотя, конечно, они весьма существенны) это значение можно оценить, как:

$E=\alpha LS\left(t_1+2\frac{t_2}{2}\right)$, где α – коэффициент поглощения резервуара, L – солнечная постоянная, S – площадь поверхности зеркал, t_1 - время солнечного дня, t_2 – время заката или рассвета. Деление пополам возникает из-за того, что во время рассвета и заката мощность растет линейно от нуля до максимума, а соответственно среднее значение равно половине максимального.

В свою очередь тепло, которое примет теплоноситель за весь день резервуар можно оценить, считая, что весь теплоноситель нагревается в резервуаре и сразу после нагревания откачивается:

$Q=c_{уд}m\left(T_к-T_н\right)$, тогда искомая масса:

$$m=\frac{\alpha LS\left(t_1+2\frac{t_2}{2}\right)}{c_{уд}\left(T_к-T_н\right)}=\frac{0.95\cdot1367\cdot300\cdot\left(14+2\frac{1}{2}\right)\cdot3600}{4000\cdot\left(590-290\right)}кг=23.7\cdot10^3кг$$

Ответ: $m=17.8\cdot10^3кг$

Критерии оценки:

- Получено выражение для энергии от Солнца – 2 балла.
- Правильно учтено изменение мощности света утром и вечером – 4 балла
- Сделано предположение о нагревании теплоносителя и записано количество теплоты – 2 балла
- Получено выражение для правильного ответа – 3 балла.
- Получен правильный ответ – 1 балл.
- Если получен правильный ответ без учета изменения мощности – по 1 баллу за выражение и число.
- Решения удовлетворяющего критериям, приведенным выше, нет, но приведены разумные рассуждения, направленные на решение задачи - 2 балла.