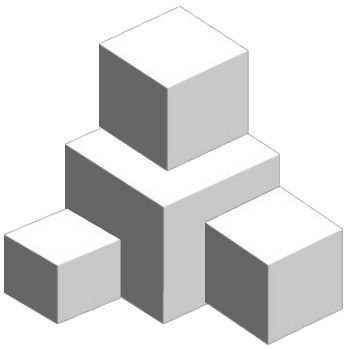
Разбор задач муниципального этапа олимпиады по информатике

1. Упаковка (7-11 класс)

Тема: разбор случаев
Сложность: простая



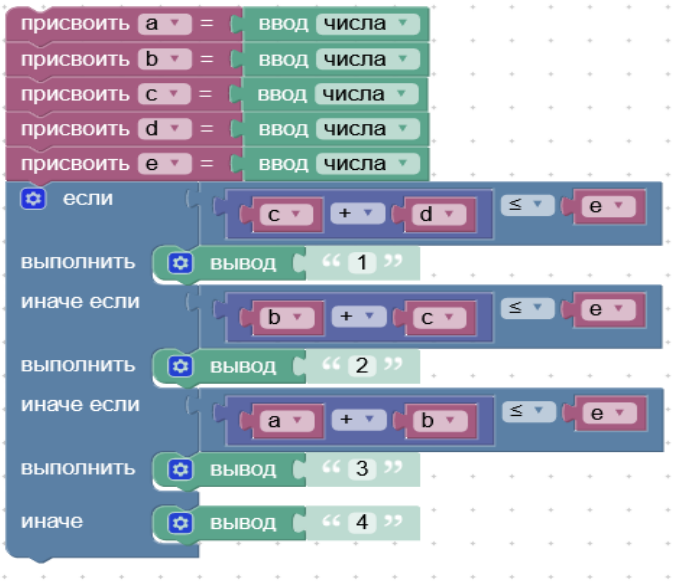
Если поставить самый большой прибор (с размером стороны D) в угол коробки, то к каждой из трех граней можно поставить 3 других прибора, если их размеры не превышают E-D. Таким образом в одну коробку можно положить все приборы, если C+D≤E. Две коробки потребуются, если C+D>E и B+C≤E. Достаточно трех коробок, если два самых маленьких прибора поместятся в одну коробку A+B≤E. Иначе каждый прибор нужно положить в отдельную коробку.

Пример реализации:

```
var a,b,c,d,e,r:integer;
begin
  read(a,b,c,d,e);
  if c+d<=e then
    r:=1
  else if b+c<=e then
    r:=2
  else if a+b<=e then
    r:=3
  else
    r:=4;
  writeln(r);
end.
```

Так как во втором случае после составления команд остаются не более двух человек, можно вывести целая_часть((N+M)/3).

Пример реализации на Blockly:



2. Алгоритм (7-9 класс)

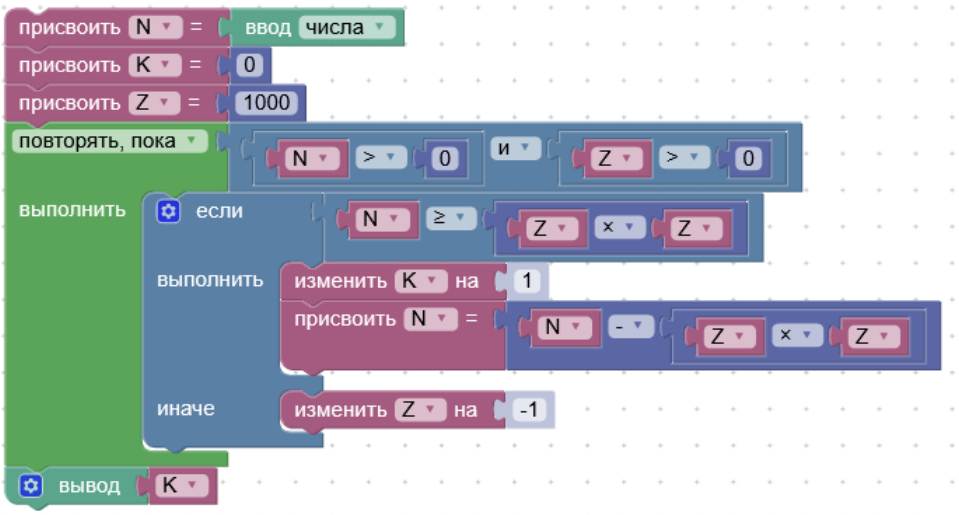
Тема: реализация программы по схеме алгоритма
Сложность: простая

Пример реализации:

```
var n,z,k:integer;
begin
  read(n);
  k:=0;
  z:=1000;
  while (n>0) and (z>0) do
```

```
  begin
    if n>=z*z then
    begin
      k:=k+1;
      n:=n-z*z;
    end
    else
      z:=z-1;
  end;
  writeln(k);
end.
```

Пример реализации на Blockly:



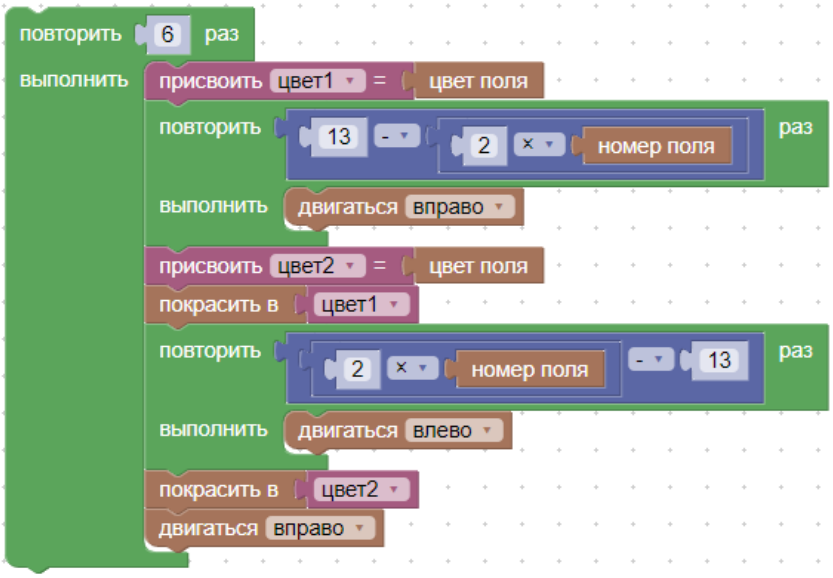
3. Обратный порядок (7-8 классы)

Тема: исполнители
Сложность: простая

Необходимо поменять цвет шести пар клеток: 1 и 12, 2 и 11, …, 5 и 6. Для этого запоминаем цвет первой клетки из пары в переменной цвет1, перемещаемся к второй клетке пары, запоминаем её цвет в переменной цвет2, красим её в цвет1, перемещаемся обратно к первой клетке пары и красим её в цвет2.

Расстояние для перемещения между клетками пары можно либо вычислить по номеру клетки (пример 1), либо делать перемещения, постоянно уменьшая расстояние на 2 (пример 2). Также вместо цикла можно повторить блок операторов для каждой пары клеток, изменяя длину перемещения.

Пример реализации 1:

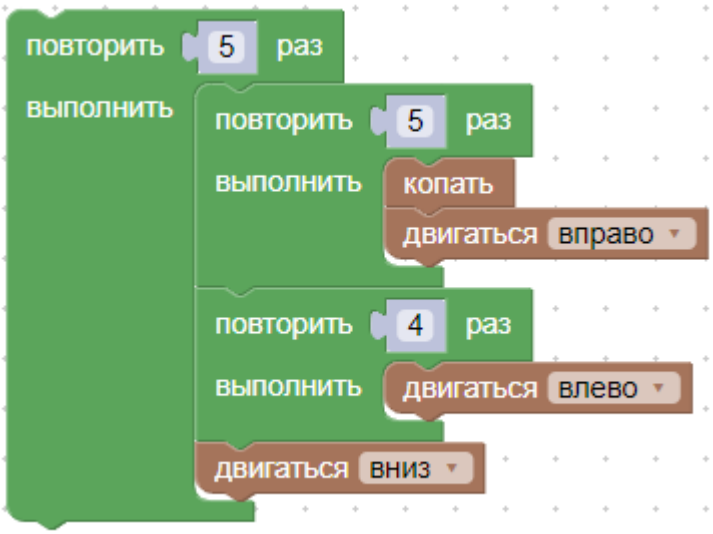


Пример реализации 2:

4. Клад (7-8 класс)

Тема: исполнители, разбиение на подзадачи
Сложность: ниже среднего

Для решения подзадачи 1 можно раскопать все 25 клеток поля:



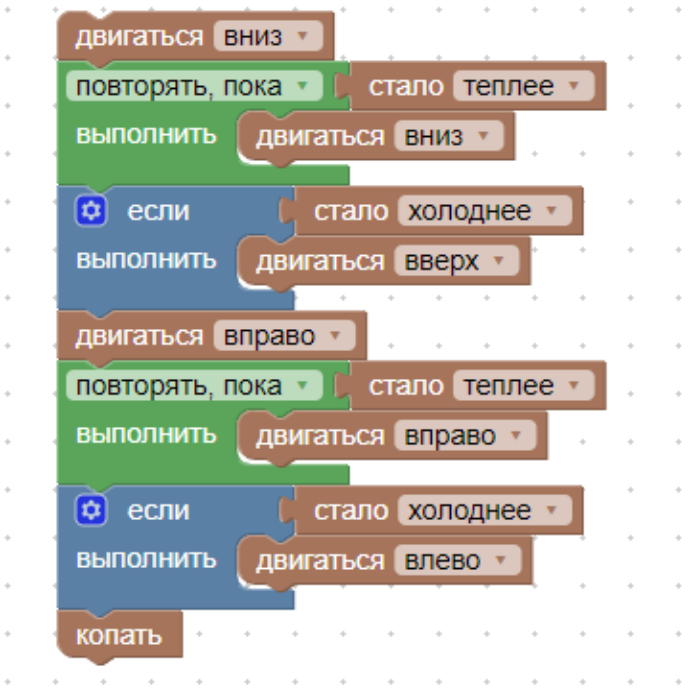
Для решение подзадачи 3 (и всех подзадач) нужно встать на клетку с кладом и раскопать её.

Сначала найдем строку, в которой находится класс. Будем двигаться вниз, пока не станет холоднее или мы дойдем до последней строки. Если стало холоднее, то нужно вернуться на строку выше. Наибольшее количество перемещений потребуется, если клад будет в предпоследней строке. Тогда нам потребуется 4 перемещения вниз и 1 одно вверх (итого 5 перемещений). Только дойдя до последней строки, можно различить случаи, когда клад находится в последней строке (стало теплее) от случая предпоследней строки (стало холоднее).

Аналогичным образом находим столбец, в котором находится клад, за 5 или менее перемещений.

Копаем в клетке, в которой робот оказался после перемещений.

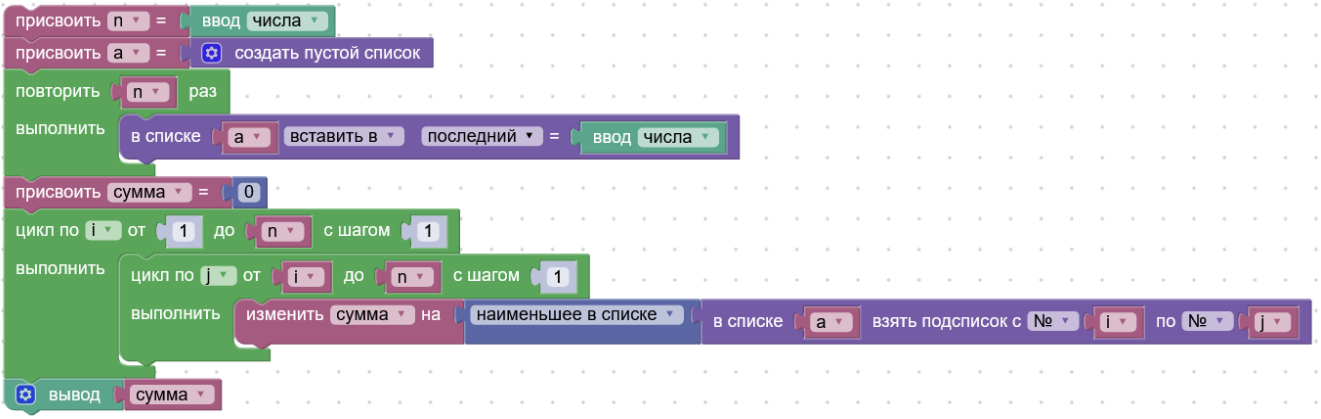
Пример реализации



5. Сумма минимумов (7-8 класс)

Тема: поиск минимума, перебор, оптимизация перебора
Сложность: выше среднего

Для решения подзадачи 1 просто считаем сумму по всем подспискам:



Для решения подзадачи 2 оптимизируем поиск минимума. Так как в процессе поиска минимума мы находим минимум просмотренной части списка, можно искать минимумы для всех подсписков с i-го до n-го элемента и добавлять текущее значение минимума к сумме.

```
var a: array[1..35000] of integer;
  n,i,j,m:integer;
  sum:int64;
begin
  read(n);
  sum:=0;
  for i:=1 to n do
    read(a[i]);
  for i:=1 to n do
  begin
    m:=a[i];
    sum:=sum+m;
    for j:=i+1 to n do
    begin
      if m>a[j] then m:=a[j];
      sum:=sum+m;
```

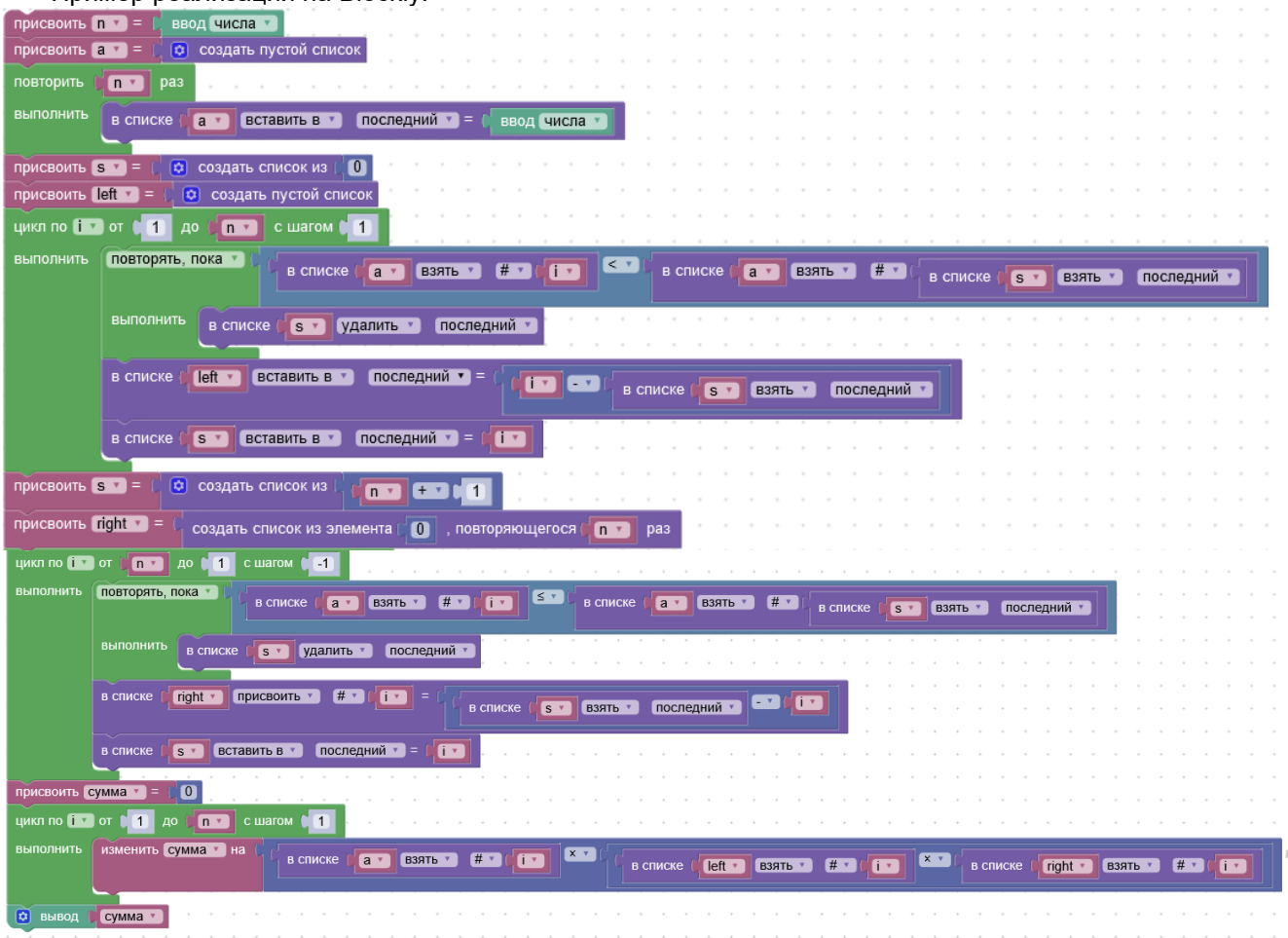
```
    end;
  end;
  writeln(sum);
end.
```

Для полного решения задачи заметим, что минимумы в сумме повторяется несколько раз. Количество повторений зависит от количества подсписков, в которых данный элемент является минимумом. Будем учитывать только подсписки, в которых данный элемент является *первым минимумом* (в подсписке может быть несколько одинаковых минимальных значений, поэтому нужно сделать уточнение, для какого из минимальных элементов этот подсписок будет учитываться). Нужно подсчитать для каждого элемента, сколько значений слева строго больше его (сохраним этот результат+1 в left[i]) и сколько значений справа больше или равны ему (сохраним этот результат+1 в right[i]). Например, для 5-го элемента (равного 2) в списке a=[5,2,4,3,2,4,2,2,1,7] два элемента [4,3] до предыдущей 2 больше a[5], и три элемента [4,2,2], больше или равны a[5]. Можно выбрать 3 варианта для начальной части подсписка, включающего a[5] - ничего не дописывать в начало или дописать [3] или дописать [4,3]. Аналогично есть 4 варианта для конечной части подсписка. Итого можно составить 3*4=12 подсписков. Таким образом элемент a[i] в сумму входит left[i]*right[i] раз. Для получения ответа нужно просуммировать a[i]*left[i]*right[i].

Расчет left можно организовать, просматривая список слева направо и запоминая во вспомогательном списке s номера предыдущих минимумов. Первоначально в этот список помещаем 0. Если очередной элемент a[i] будет меньше предыдущего минимума, то нужно убрать все номера предыдущих минимумов s[j], которые оказались больше a[i]. Когда в s не останется таких номеров, в left записываем разницу между i и номером последнего минимума из s и добавляем i в s.

Расчет right можно выполнить, просматривая список справа налево, аналогичным способом.

Пример реализации на Blockly:



2. Парный танец (10-11 класс)

Тема: поиск минимума или сортировка, жадный алгоритм.
Сложность: ниже среднего

Пусть рост мальчиков и девочек дан в неубывающем порядке. Тогда в пару для мальчика с самым маленьким ростом A_1 нужно поставить девочку с самым маленьким ростом B_1. Если выбрать девочку с другим ростом, например, таким что $|B_i-A_1|$ является минимальным, то девочку с ростом B_1 нужно будет поставить в пару к мальчику более высокого роста $A_j \geq A_1$. Тогда суммарная разница в росте возрастет на

$|A_1-B_i|$, если $B_i>A_1$, так как B_1 находится дальше от A_j, чем от A_1. Поэтому можно либо отсортировать мальчиков и девочек по росту и расставить их пары в порядке возрастания роста, либо находить мальчика и девочку с минимальным ростом из детей без пары и ставить их в пару, пока все дети не получат пару.

Пример реализации:

```
var n,i,j,mj,t,sum:integer;
    a,b:array[1..100] of integer;
begin
  read(n);
  for i:=1 to n do
    read(a[i]);
  for i:=1 to n do
    read(b[i]);
  sum:=0;
  for i:=1 to n do
  begin
    mj:=i;
    for j:=i+1 to n do
      if a[mj]>a[j] then
        mj:=j;
    t:=a[mj]; { перестановка минимального и i-го }
    a[mj]:=a[i];
    a[i]:=t;
    mj:=i;
    for j:=i+1 to n do
      if b[mj]>b[j] then
        mj:=j;
    t:=b[mj]; { перестановка минимального и i-го }
    b[mj]:=b[i];
    b[i]:=t;
    sum:=sum+abs(a[i]-b[i]);
  end;
  writeln(sum);
end.
```

3. Фуршет (9-11 класс)

Тема: жадный алгоритм, структуры данных или сортировка
Сложность: средняя

Для решения подзадачи 1 можно проверить для каждого от 1 до 100, сколько блюд может съесть Владимир. Блюдо можно съесть, если до последнего съеденного блюда проверяемого типа было не менее чем K блюд.

В подзадаче 2 можно хранить номер последнего появления блюда t и количество потенциально съеденных Владимиром блюд в специальном массиве. После просмотра блюд нужно найти в этом массиве блюдо с максимальным количеством.

Пример реализации:

```
type dish=record x,k:integer; end;
var t:array [1..100000] of dish;
  n,k,i,ti,mt,mk:integer;
begin
  read(n,k);
  for i:=1 to 100000 do
    t[i].x:=-k;
  for i:=1 to n do
  begin
    read(ti);
    if t[ti].x+k<i then
    begin
      t[ti].x:=i;
      inc(t[ti].k);
    end;
  end;
  mk:=0;
  for i:=1 to 100000 do
  begin
```

```
    if t[i].k>mk then
    begin
      mt:=i;
      mk:=t[i].k;
    end;
  end;
  writeln(mt,' ',mk);
end.
```

В подзадаче 3 вместо массива можно использовать ассоциативный массив или сортировку (в этом случае блюда одного типа будут рядом и можно проверить разницу между индексами).

Пример реализации для ассоциативного массива (словаря):

```
type dish=record x,k:integer; end;
var t:=new Dictionary<integer, dish>;
    val:dish;
    n,k,ti,mt,mk:integer;
begin
  read(n,k);
  for var i:=1 to n do
  begin
    read(ti);
    if t.ContainsKey(ti) then
    begin
      val:=t[ti];
      if val.x+k<i then
      begin
        val.x:=i;
        inc(val.k);
        t[ti]:=val;
      end;
    end
    else
    begin
      val.x:=i;
      val.k:=1;
      t.Add(ti,val);
    end;
  end;
  mk:=0;
  mt:=0;
  writeln(t.count);
  foreach var i in t.Keys do
  begin
    k:=t[i].k;
    if (k>mk) or (k=mk) and (i<mt) then
    begin
      mt:=i;
      mk:=k;
    end;
  end;
  writeln(mt,' ',mk);
end.
```

Пример реализации сортировкой:

```
type dish=record x,k:integer; end;
var t:array of dish;
    m,temp:dish;
  n,k,i,pk,tx,tk,mx,mk:integer;
procedure qsort(l,r:integer);
var i,j:integer;
begin
  if l>=r then  exit;
  m:=t[l+random(r-l+1)];
  i:=l;
  j:=r;
  while i<=j do
  begin
    while (t[i].x<m.x) or (t[i].x=m.x) and (t[i].k<m.k) do  inc (i);
```

```
      while (t[j].x>m.x) or (t[j].x=m.x) and (t[j].k>m.k) do  dec (j);
      if i<=j then
      begin
        temp:=t[i];t[i]:=t[j];t[j]:=temp;
        inc(i); dec(j);
      end;
    end;
    qsort(l,j);
    qsort(i,r);
  end;
  begin
    read(n,k);
    setlength(t,n);
    for i:=1 to n do
    begin
      read(t[i].x);
      t[i].k:=i;
    end;
    qsort(1,n);
    pk:=t[1].k;
    tx:=t[1].x;
    tk:=1;
    mx:=tx;
    mk:=tk;
    for i:=2 to n do
    begin
      if t[i].x<>tx then
      begin
        pk:=t[i].k;
        tx:=t[i].x;
        tk:=1;
      end
      else
      begin
        if t[i].k>pk+k then
        begin
          inc(tk);
          pk:=t[i].k;
          if tk>mk then
          begin
            mx:=tx;
            mk:=tk;
          end;
        end
      end;
    end;
    writeln(mx,' ',mk);
  end.
```

4. Высадка (9-11 класс)

Тема: моделирование, структуры данных
Сложность: средняя

Для подзадачи 1 можно использовать моделирование.

Пример реализации:

```
var n,m,i,j,x,k:integer;
    d:int64;
  p:array[1..100000] of integer;
begin
  read(n);
  for i:=1 to n do
    read(p[i]);
  read(m);
  for i:=1 to m do
  begin
```

```
    read(x,k);
    d:=0;
    j:=x;
    while k>0 do
    begin
      if k>p[j] then
      begin
        k:=k-p[j];
        d:=d+int64(p[j])*(j-x);
        p[j]:=0;
        inc(j);
        if j>n then
        begin
          j:=1;
          x:=x-n;
        end;
      end
      else
      begin
        d:=d+int64(k)*(j-x);
        p[j]:=p[j]-k;
        k:=0;
      end;
    end;
    writeln(d);
  end;
end.
```

Для полного решения необходимо воспользоваться ассоциативным массивом, в котором будут храниться только поселения с ненулевым количеством свободных мест.

Пример реализации:

```
#include <iostream>
#include <set>
using namespace std;
typedef long long ll;
int main()
{
  ios::sync_with_stdio(0); // для ускорения ввода
  cin.tie(0);
  int n,a,q,k,x;
  cin>>n;
  set<pair<int,int>> se;
  for(int i=1; i<=n; ++i)
  { int a;
    cin>>a;
    se.insert({i,a});
  }
  cin>>q;
  for(int i=0;i<q;++i)
  {
    ll d=0;
    cin>>x>>k;
    auto it=se.lower_bound({x,0});
    if(it==se.end()) {
      it=se.begin();
      x-=n;
    }
    while(k>0)
    { int f=it->first;
      int s=it->second;
      auto itr=it;
      it++;
      se.erase(itr);
      if(s>k)
      {
        d+=(ll)k*(f-x);
        se.insert({f, s - k});
```

```
      }
      else
      {
        d+=(ll)s*(f-x);
        if(it==se.end()) {
          it=se.begin();
          x-=n;
        }
      }
      k-=s;
    }
    cout<<d<<"\n";
  }
}
```

5. Экспедиция (9-11 класс)

Тема: поиск минимума, бинарный поиск, структуры данных
Сложность: выше средней

Подзадачи 1 и 2 можно решить полным перебором. Важно только учесть возможность переполнения типа int при вычислениях и использовать 64-битный целый тип.

Пример реализации:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
typedef long long ll;
vector<pair<ll, ll>> a;
ll n, m, c, t, best, ans;
int main()
{
      cin >> n >> m >> c;
      a.resize(n);
      for (int i = 0; i < n; i++)
            cin >> a[i].first >> a[i].second;
      for (int i = 0; i < m; i++)
      {
            best = -1;
            cin >> t;
            for (int j = 0; j < n; j++)
            {
                  ans = llabs(a[j].first - t) * c + a[j].second;
                  if (ans < best || best == -1)
                        best = ans;
            }
            cout << best << "\n";
      }
}
```

Для полного решения предположим сначала, что дорога односторонняя. Например, пусть движение разрешено только слева направо. Тогда для достижения каждой точки дороги мы можем использовать только площадки слева от нее. Попробуем выбрать из них наиболее "выгодную". Заметим, что если одна из площадок "выгоднее" другой в какой-то точке X, то во всех точках Y правее X это свойство будет сохраняться, т.к. полная стоимость использования обеих площадок при достижении любой точки будет расти на одну и ту же величину C*(Y-X). Отсортируем площадки по их расположению слева направо, и для каждой будем отвечать на вопрос: выгоднее нам будет на ней высадиться, или приехать с "самой выгодной" из расположенных левее. Соответственно, площадка либо игнорируется полностью, либо становится новой "самой выгодной" для какого-то отрезка справа от себя (как минимум, до следующей площадки). Запомним только "выгодные" площадки, и затем для каждой экспедиции двоичным поиском найдем ближайшую слева площадку, они и будет оптимальным выбором. И наконец вспомним, что дорога двусторонняя. Просто решим задачу 2 раза независимо для "левосторонней" и "правосторонней" дорог, и выберем минимум из 2 ответов. Сложность: O(NlogN + MlogN).

Пример реализации:

```
#include <iostream>
#include <vector>
```

```
#include <algorithm>
using namespace std;
typedef long long ll;
typedef pair<int,int> pii;
vector<pii> areas, aleft, aright;
ll c;
constexpr ll inf=2e18;
constexpr int inf1=1e9+1;
int main()
{
  ios::sync_with_stdio(0); // для ускорения ввода
  cin.tie(0);
      int n, m;
      cin >> n >> m >> c;
      areas.resize(n);
      for (int i = 0; i < n; i++)
         cin >> areas[i].first >> areas[i].second;
      sort(areas.begin(), areas.end());
      aleft.push_back(active=areas.front());
      for (int i = 1; i < n; i++)
        if((areas[i].first - active.first) * c + active.second > areas[i].second)
          aleft.push_back(active=areas[i]);
      aright.push_back(active=areas.back());
      for (int i = n - 2; i >= 0; i--)
        if((active.first-areas[i].first) * c + active.second > areas[i].second)
          aright.push_back(active=areas[i]);
      reverse(aright.begin(), aright.end());
      for (int i = 0; i < m; i++)
      {
            int t;
            ll best = inf;
            cin >> t;
            //find the nearest efficient area leftwards, if exists
            auto l = upper_bound(aleft.begin(), aleft.end(), make_pair(t, inf1));
            if (l != aleft.begin())
            {
                  l--;
                  ll lans = (t - l->first) * c + l->second;
                  if (lans < best)
                        best = lans;
            }
            auto r = upper_bound(aright.begin(), aright.end(), make_pair(t, 0));
            if (r != aright.end())
            {
                  ll rans = (r->first - t) * c + r->second;
                  if (rans < best)
                        best = rans;
            }
            cout << best << "\n";
      }
}
```