Решение. Построим граф. Пусть вершины графа обозначают перекрёстки в городе, а ребра графа – дороги. Каждая вершина представляет собой запись с полями:

- Количество выходящих и входящих в вершину ребер.
- Путь: массив, хранящий указатели на вершины, в которые можно попасть из текущей вершины.
- Длина пути: массив, хранящий длины всех рёбер текущей вершины.
- Законность пути: массив, хранящий информацию для каждого ребра о том, нарушим ли мы правила, если перейдём по этому ребру.
- Переменная, обозначающая, помечена вершина или нет.
- Номер вершины.
- Следующая вершина (все вершины линейно упорядочены по номерам, и эта переменная указывает на вершину с номером, большим на единицу).

Для того чтобы запомнить наилучший путь, использован связный список, каждый элемент которого представляет собой запись с переменными:

- Номер вершины графа (путь запоминается номерами вершин)
- Предыдущий элемент списка

Изначально мы находимся в первой вершине. Нужно попасть в последнюю. Номер последней вершины – количество вершин в графе

Ввод данных

Из первой строки входного файла считываются два числа: количество вершин и рёбер. Из второй строки входного файла считывается количество топлива, которое измеряется в тех же единицах, что и длина рёбер. Т.е. N единиц топлива хватит на N единиц пути.

Создаётся исходная структура графа, где все вершины связаны линейно в виде списка. Каждая вершина получает свой номер и ссылку на следующую вершину.

Из входного файла считывается построчно информация о каждом ребре. Каждое ребро описывается так: начальная вершина ребра, конечная вершина ребра, длина ребра. На основе этих данных заполняется граф.

Поиск пути. Для поиска пути использована рекурсивная функция:

Поиск пути (параметры: текущая вершина, количество использованного топлива, количество нарушенных правил) возвращает переменную логического типа (показывает, что нужно записывать наилучший путь)

Введём переменную A логического типа и B целого типа

1. Изначально Поиск пути становится False
2. Помечаем текущую вершину
3. Если текущая вершина – последняя, то
   - 3.1. Если количество нарушений меньше минимального количества нарушений, то
     - 3.1.1. Минимальное количество нарушений принимает значение текущего количества нарушений
     - 3.1.2. Поиск пути принимает значение True
     - 3.1.3. Создаём новый текущий элемент списка пути (это глобальная переменная)
     - 3.1.4. Записываем в этот элемент номер текущей вершины
     - 3.1.5. Указатель на предыдущий элемент = nil
4. Иначе:
   - 4.1. A принимает значение False;

4.2. Для i от 1 до количества рёбер текущей вершины
4.2.1. Если вершина, в которую приводит i-ый путь, не помечена и использованное топливо + длина i-того пути меньше исходного количество топлива, то
4.2.1.1.Если i-ый путь законный, то B=0, иначе B=1
4.2.1.2.Если поиск пути (вершина, в которую приводит i-ый путь; использованное топливо + длина i-го пути, количество нарушений правил + B) = True, тогда
4.2.1.2.1. A = True
4.3. Если A = True, тогда
4.3.1. Создаём новый элемент списка пути
4.3.2. Записываем в этот элемент номер текущей вершины
4.3.3. Указатель на предыдущий элемент у этого элемента указывает на текущий элемент списка пути
4.3.4. Новый элемент становится текущим
4.3.5. Поиск пути принимает значение True
5. Делаем текущую вершину не помеченной

Основная программа.

1. Ввод данных
1. Минимальное количество нарушений = количество дорог + 1
2. Если Поиск пути (первая вершина, 0, 0) = True
2.1. Пока текущий элемент списка пути не первый элемент списка, то
2.1.1. Выводим номер вершины графа, хранящийся в текущем элементе списка пути
2.1.2. Текущим становится предыдущий для текущего элемент
2.2. Выводим номер последней вершины графа
3. Иначе выводим, что решения нет

Программа на Паскале

```
program path_search;
type cross=^T_Cross;
    T_cross = record
      way_num: byte;
      way: array[1..4] of cross;
      len: array[1..4] of integer;
      law: array[1..4] of boolean;
      checked: boolean;

      num: integer;
      next: cross;
    end;
    way_save=^T_Way;
    T_Way = record
      num: integer;
      last: way_save;
    end;

var crosses, roads, fuel_start: integer;
      first,current: cross;
```

```
       law_breaks_min: integer;
      first_2,way2: way_save;
            way3: way_save;


procedure Create_Crosses(col: integer);{--------------------------------}
var i,j: integer;
begin
  new(first);
  current:=first;
  for i:=1 to col do begin
    for j:=1 to 4 do
      current^.way[j]:=nil;
    current^.way_num:=0;

    current^.num:=i;
    new(current^.next);
    current:=current^.next;
    current^.checked:=false;
  end;

end;

procedure SetRoad(number,Road_finish,Road_lenght: integer; road_right: boolean);
var cur,cur1: cross;
    i: integer;
begin
  cur:=first;
  for i:=1 to number-1 do
     cur:=cur^.next;


  cur1:=first;
  for i:=1 to road_finish-1 do
     cur1:=cur1^.next;

  Inc(cur^.way_num);
  i:=cur^.way_num;
  with cur^ do begin
     way[i]:=cur1;
     len[i]:=road_lenght;
     law[i]:=road_right;
  end;
end;

procedure Input; {-----------------------------------------------------}
var f: text;
    i: integer;
    a,b,l: integer;
begin
   writeln;writeln;writeln;
   assign(f,'C:\TEMP\INPUT.TXT');
```

```
  reset(f);
  Readln(f,crosses,roads);
  Readln(f,Fuel_start);

  Create_Crosses(crosses);

  for i:=1 to roads do begin
    Readln(f,a,b,l);
    SetRoad(a,b,l,True);
    SetRoad(b,a,l,False);
{     writeln(a,' ',b,' ',l);}
  end;

  close(f);

end;

Procedure Output;{-----------------------------------------------------}
begin
  writeln;
  writeln(crosses,' ',roads);
  writeln(Fuel_start);
end;

function Pass(Cur: cross; fuel,law_breaks: integer): boolean;
var i: integer;
    l_b: byte;
    b: boolean;
begin
  Pass:=false;
  Cur^.Checked:=True;
  If Cur^.num=crosses then begin
{     Writeln('New Way found:');                              }
    If law_breaks<law_breaks_min then begin
       law_breaks_min:=law_breaks;
{       writeln('- BEST in law');                        }
       Pass:=True;

       new(way2);
{       way2:=first_2;}
       way2^.num:=Cur^.num;
       way2^.last:=nil;
    end;
  end else begin
  with Cur^ do begin
    b:=false;
    for i:=1 to way_num do begin
      If (not (way[i]^.Checked)) and (fuel+len[i]<=fuel_start) then begin
        If law[i] then l_b:=0 else l_b:=1;
         If Pass(way[i],fuel+len[i],law_breaks+l_b) then b:=true; end;
    end;
    If b then begin way3:=way2; new(way2); way2^.num:=Cur^.num;
```

```
          way2^.last:=way3;
          pass:=true;
     end;
   end;
  end;
  Cur^.Checked:=False;
end;

begin {------------------------------------------------------------}
  Input;

  Law_breaks_Min:=roads+1;
  If Pass(first,0,0) then begin
    writeln('Answer: ');
    while way2<>nil do begin
      write(' ',way2^.num);
      way2:=way2^.last;
     end;
  end else writeln('no answer');
  ReadLn;
```

Программа на Си

```
//     way to benzokolonka EARCH
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
struct cross {
    int way_num;
    int way[3];
    int len[3];
    int law[4];
    int checked;
    int num;
    } cross_arr[100];
int way2[100],way3[100];
int current_way_pos=0;
int crosses,roads,fuel_start;
int current,max;
int law_breaks_min;
void create_crosses(int col) {
max = col;
max++;
for (int i=0; i<=max; i++) {
    for (int j=0; j<=3; j++) {
          cross_arr[i].way[j]=-1; }
    cross_arr[i].way_num=0;
    cross_arr[i].num=i;
    cross_arr[i].checked = 0;
    }
}
```

```
void setroad(int number, int road_finish,
             int road_length, int road_right) {
int  i;
i=cross_arr[number].way_num;
cross_arr[number].way_num=cross_arr[number].way_num+1;
i++;
cross_arr[number].way[i]=road_finish;
cross_arr[number].len[i]=road_length;
cross_arr[number].law[i]=road_right;
}

void Input() {
FILE *f;
int a,b,l;
   cout <<"\n" <<"\n" <<"\n";
  f = fopen("C:/TEMP/INPUT.TXT","r");
  fscanf(f,"%d\n",&crosses);
  fscanf(f,"%d\n",&roads  );
  cout <<crosses <<" " <<roads;
  fscanf(f,"%d\n",&fuel_start);
  cout <<"\n";
  cout <<fuel_start;
  cout <<"\n";
  create_crosses(crosses);
  for (int i=1; i<=roads; i++) {
    fscanf(f,"%d\n",&a);
    fscanf(f,"%d\n",&b);
    fscanf(f,"%d\n",&l);
    cout <<a <<" " <<b<<" " <<l <<" " <<"\n";
    setroad(a-1,b-1,l,1);
    setroad(b-1,a-1,l,0);
  }
   fclose(f);
}

int pass(int cur, int fuel,int law_breaks) {
  int l_b;
  int b;
  int pass_bool;
  pass_bool=0;
  cross_arr[cur].checked=1;
  if (cross_arr[cur].num==crosses-1) {
     if (law_breaks<law_breaks_min) {
     law_breaks_min = law_breaks;
     pass_bool = 1;
     current_way_pos=0;
     way2[0]=cross_arr[cur].num;
     }
  } else {
     b=0;
     for (int i=1; i<=cross_arr[cur].way_num; i++) {
       if ((cross_arr[cross_arr[cur].way[i]].checked == 0) &&
```

```
            (fuel+cross_arr[cur].len[i]<=fuel_start)) {
              if (cross_arr[cur].law[i]==1) {l_b=0;}
              else {l_b=1;}
               if (pass(
                                cross_arr[cur].way[i],
                        fuel+cross_arr[cur].len[i],
                        law_breaks+l_b)==1) {b=1;}
              }
              }
              if (b==1) {
                    current_way_pos++;
                    way2[current_way_pos]=cross_arr[cur].num;
                    pass_bool=1;
              }
            }
  cross_arr[cur].checked=0;
  return pass_bool;
}

void main() {
  Input();
  law_breaks_min=roads+1;
  if (pass(0,0,0)) {
     cout <<"\n";
     cout <<"Answer";
           cout <<" ";
     for (int i=current_way_pos; i>=0; i--){
           cout <<way2[i]+1;
           cout <<" ";
           }
     } else {
          cout <<"\n";
          cout <<"No answer    "  ;   }
  getch();
}
```