ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники

Кафедра «ВТ»

ЛАБОРАТОРНАЯ РАБОТА № _4_

ПО ДИСЦИПЛИНЕ «*Алгоритмы и структуры данных*»

Выполнил(а): Чан Чунг Дык

Группа: Р3202

Санкт-Петербург

2019

1080. Map coloring

Algorithm :

+ Build a tree. Using DFS to color all the node of tree. If node u – color red, so all the node v, which connect to u will be colored blue. We can't color the tree if two node connect each others has the same color.

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 110;
int dd[maxn][maxn];
int colour[maxn];
int n;
bool ok;
void DFS(int x) {
  for (int i = 1; i <= n; i++)
    if (dd[i][x] == 1 || dd[x][i] == 1) {
      if (colour[i] == -1) {
        colour[i] = 1 - colour[x];
        DFS(i);
      } else if (colour[x] == colour[i])
        ok = false;
    }
}
int main() {
  cin >> n;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      dd[i][j] = 0;
  for (int i = 1; i <= n; i++) {
    int x;
    cin >> x;
    while (x != 0) {
      dd[i][x] = 1;
      dd[x][i] = 1;
      cin >> x;
    }
  }
  ok = true;
  for (int i = 1; i <= n; i++)
    colour[i] = -1;
  for (int i = 1; i <= n; i++) {
    if (colour[i] == -1) {
      colour[i] = 0;
      DFS(i);
    }
  }
  if (ok) {
    for (int i = 1; i <= n; i++)
      cout << colour[i];
  } else
    cout << -1;
}
```

1160. Network

Algorithm : Using Kruskal to find mimimum tree in graph.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
const int maxn = 1000 + 10;
const int maxm = 15000 + 10;
int n, m;
int par[maxn];
bool dd[maxm];
struct edge {
  int u, v, ts;
};
vector<edge> cable;

int anc(int p) {
  if (par[p] == p)
    return p;
  else
    return par[p] = anc(par[p]);
}

void join(int u, int v) { par[anc(u)] = anc(v); }

int myComp(edge tmp, edge _tmp) { return tmp.ts < _tmp.ts; }

void kruskal() {
  int res = 0;
  int count = 0;
  int maxTS = 0;
  sort(cable.begin(), cable.end(), myComp);
  for (int i = 0; i <= m; i++)
    dd[i] = false;
  for (int i = 1; i <= n; i++)
    par[i] = i;
  int h = 0;
  for (auto cab : cable) {
    if (anc(cab.u) != anc(cab.v)) {
      join(cab.u, cab.v);
      dd[h] = true;
      res += cab.ts;
      if (cab.ts > maxTS)
```

```
        maxTS = cab.ts;
      count++;
    }
    h++;
  }
  cout << maxTS << endl;
  cout << count << endl;
  for (int i = 0; i < m; i++)
    if (dd[i])
      cout << cable[i].u << " " << cable[i].v << endl;
}

int main() {
  cin >> n >> m;

  for (int i = 0; i < m; i++) {
    int x, y, z;
    cin >> x >> y >> z;
    edge tmp;
    tmp.u = x;
    tmp.v = y;
    tmp.ts = z;
    cable.push_back(tmp);
  }
  kruskal();
}
```

1450. Russian Pipelines

The main idea is find the maximum path from S -> F. So we will find the minimum path with negative gas transfer.
Using fordbellmanqueue.

```
#include <iostream>
#include <queue>
using namespace std;

#define X first
#define Y second
const int maxn = 500 + 10;
typedef pair<int, int> ii;
vector<ii> a[maxn];
int n, m;

int d[maxn];
```

```
bool inqueue[500 * 500];

void bellman(int u) {
  int i, v, uv;
  queue<int> qu;
  for (i = 1; i <= n; i++)
    d[i] = 1000111000;
  d[u] = 0;
  qu.push(u);
  inqueue[u] = true;
  while (qu.size()) {
    u = qu.front();
    inqueue[u] = false;
    qu.pop();

    for (i = 0; v = a[u][i].Y; i++) {
      uv = a[u][i].X;
      if (d[v] > d[u] + uv) {
        d[v] = d[u] + uv;
        if (!inqueue[v]) {
          qu.push(v);
          inqueue[v] = true;
        }
      }
    }
  }
}

main() {
  int p, q, w, i, u, v;
  cin >> n >> m;
  while (m--) {
    cin >> p >> q >> w;
    w = 0 - w;
    a[p].push_back(ii(w, q));
  }
  cin >> u >> v;
  for (i = 1; i <= n; i++)
    a[i].push_back(ii(0, 0));
  bellman(u);

  if (d[v] != 1000111000)
    cout << 0 - d[v];
  else
    cout << "No solution";
}
```

1806. Mobile Telegraphs

The main idea is find minimum cost path from the first fighter to the last fighter. We will create a graph, which show link each others. We have edge, each of them will have cost to tranfer information from one to other.

How we can create graph? Normally, we use loop to find 2 telegraph cost how many to transfer. But it will execute around 50000*50000*10 operations. It ‘s so big. So now, with each telegraphs, we will make all of numbers, which can be created from the original. With each created number, we will find out them in list of numbers. So it maybe execute 50000*9*10*log2(50000) . It ‘s smaller .

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <queue>
#include <map>
#include <iterator>
using namespace std;

const int maxn = 50000 + 10;

typedef pair<int, int> ii;
int n;
int dd[10];
int d[maxn];
bool inqueue[maxn];
int trace[maxn];
string arr[maxn];
vector<ii> ts[maxn];
map<long long, int> tele;

int find_tele(long long x) {
  if (tele.find(x) != tele.end())
    return tele.find(x)->second;
  return -1;
}
long long pow(int x, int y) {
  long long res = 1;
  for (int i = 0; i < y; i++)
    res *= (long long)x;
  return res;
}

void change_one_number(long long x, int position) {
  // cout << x << " " << position << endl;
  int pos[10];
```

```
  long long _x = x;
  for (int i = 9; i >= 0; i--) {
    int c = _x % 10;
    _x /= 10;
    pos[i] = c;
  }
  for (int i = 0; i < 10; i++) {
    // cout << pos[i]<< endl;
    for (int j = 1; j < pos[i]; j++) {
      long long tmp = x - pow(10, 9 - i) * (long long)(pos[i] - j);

      int x_change = find_tele(tmp);
      if (x_change == -1)
        continue;
      //  cout << position << " " << x_change << endl;
      ts[position].push_back(ii(dd[i], x_change));
      ts[x_change].push_back(ii(dd[i], position));
    }
    // cout << "bigger" << endl;
    for (int j = pos[i] + 1; j < 10; j++) {
      long long tmp = x + pow(10, 9 - i) * (long long)(j - pos[i]);

      int x_change = find_tele(tmp);
      if (x_change == -1)
        continue;
      //  cout << position << " " << x_change << endl;
      ts[position].push_back(ii(dd[i], x_change));
      ts[x_change].push_back(ii(dd[i], position));
    }
    // cout << "smaller" << endl;
  }
}

void change_two_number(long long x, int position) {
  // cout << x << " " << position << endl;
  int pos[10];
  long long _x = x;
  for (int i = 9; i >= 0; i--) {
    int c = _x % 10;
    _x /= 10;
    pos[i] = c;
  }
  for (int i = 0; i < 10; i++) {
    for (int j = i + 1; j < 10; j++) {
      if (pos[i] == pos[j])
        continue;
```

```
      long long tmp = x;
      if (pos[i] < pos[j]) {
        tmp += pow(10, 9 - i) * (pos[j] - pos[i]);
        tmp -= pow(10, 9 - j) * (pos[j] - pos[i]);
      } else {
        tmp -= pow(10, 9 - i) * (pos[i] - pos[j]);
        tmp += pow(10, 9 - j) * (pos[i] - pos[j]);
      }
      int x_change = find_tele(tmp);
      if (x_change == -1)
        continue;
      // cout << position << " " << x_change << endl;
      ts[position].push_back(ii(dd[i], x_change));
      ts[x_change].push_back(ii(dd[i], position));
    }
  }
}

void init() {
  for (auto i : tele) {
    change_one_number(i.first, i.second);
    change_two_number(i.first, i.second);
  }
}

void bellman(int u) {
  int i, v, uv;
  queue<int> qu;
  for (int i = 0; i <= n; i++)
    trace[i] = i;
  for (i = 0; i <= n; i++)
    d[i] = 1000111000;
  d[u] = 0;
  qu.push(u);
  inqueue[u] = true;

  while (qu.size()) {
    u = qu.front();
    inqueue[u] = false;
    qu.pop();

    for (i = 0; v = ts[u][i].second; i++) {
      uv = ts[u][i].first;
      if (d[v] > d[u] + uv) {
        d[v] = d[u] + uv;
        trace[v] = u;
```

```
        if (!inqueue[v]) {
          qu.push(v);
          inqueue[v] = true;
        }
      }
    }
  }
}
void try_trace() {
  int tr[maxn];
  int sl = 0;
  int x = n;
  while (x != 1) {
    tr[sl] = x;
    sl++;
    x = trace[x];
  }
  tr[sl] = 1;
  cout << ++sl << endl;
  for (int i = sl - 1; i >= 0; i--) {
    cout << tr[i] << " ";
  }
}
int main() {
  cin >> n;
  for (int i = 0; i < 10; i++)
    cin >> dd[i];
  long long tmp;
  for (int i = 1; i <= n; i++) {
    cin >> tmp;
    tele.insert(pair<long long, int>(tmp, i));
  }
  init();

  for (int i = 1; i <= n; i++)
    ts[i].push_back(ii(0, 0));
  bellman(1);

  if (d[n] != 1000111000) {
    cout << d[n] << endl;
    try_trace();
  } else
    cout << -1;
}
```