Отчет

по лабораторной работе 1806 “Мобильные телеграфы”

по дисциплине «Алгоритмы и структуры данных»

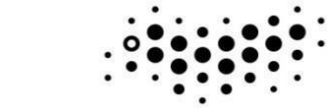
Авторы:

Полит Алексей Денисович

Факультет: СУиР

Группа: R3235

Преподаватель: Тропченко Андрей Александрович



Санкт-Петербург, 2021г

1. Задача

Каждому бойцу 25-й стрелковой дивизии выдали новейшее средство связи — мобильный телеграф. С его помощью можно отправлять телеграммы командованию и боевым товарищам прямо на поле битвы. К сожалению, конструкция телеграфов ещё далека от совершенства — передавать сообщения можно только между некоторыми парами телеграфов.

Каждому устройству присвоен уникальный номер — строка из десяти десятичных цифр. С телеграфа a можно отправить сообщение на телеграф b только в том случае, если из номера a можно получить номер b, изменив в нём ровно одну цифру либо поменяв в нём две цифры местами. Время передачи сообщения с телеграфа a на телеграф b зависит от длины наибольшего общего префикса их номеров — чем больше его длина, тем быстрее передаётся сообщение.

Во время очередного сражения Анка из своей хорошо замаскированной позиции увидела небольшую группу белых, пытающуюся обойти обороняющихся красноармейцев с тыла. Какое минимальное время понадобится на доставку этой информации от Анки до Чапаева по телеграфу, возможно, с помощью других красноармейцев?

2. Исходные данные

В первой строке записано целое число n ($2 \le n \le 50000$) — количество бойцов в дивизии. Во второй строке через пробел в порядке невозрастания записаны десять целых чисел в пределах от 1 до 10000 — время передачи сообщения с одного телеграфа на другой при длине общего префикса их номеров, равной нулю, единице, двум, ..., девяти. Далее идут n строк, содержащие номера телеграфов, выданных бойцам дивизии. Номер телеграфа Анки указан первым, а номер телеграфа Чапаева — последним. Все номера телеграфов попарно различны.

3. Код программы

```
#include <iostream>

#include <bits/stdc++.h>

const int N = 50005;
```

```
const int M = 20000005;
const int max_ = 0x3f3f3f3f;
int n;
int val[15];
std::string ch[N];
std::map<long long, int> mp;
struct edge {
    int v, w, next;
} edge[M];
int head[N], ec;
int dis[N], par[N];
bool vis[N];
std::vector<int> vec;
struct pp{
    int d, u;
    bool operator < (const pp &cmp) const {
        return d > cmp.d;
    }
};
void add_edge(int u, int v, int w) {
    edge[ec] = {v, w, head[u]};
    head[u] = ec++;
};
std::vector<long long> py;
void deal(int id) {
    long long tmp = 0;
    for (int i = 0; i < 10; ++i) {
        tmp = tmp * 10 + ch[id][i] - '0';
```

```
}
const long long tt = tmp;
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 10; ++j) {
        tmp = tmp - (ch[id][i] - '0' - j) * py[i];
        auto it = mp.find(tmp);
        if (it != mp.end()) {
            int len = 0;
            int idx = it->second;
            while (len < 9 && ch[id][len] == ch[idx][len]) {
                len++;
            }
            add_edge(id, idx, val[len]);
            add_edge(idx, id, val[len]);
        }
        tmp = tt;
    }
}
for (int i = 0; i < 10; ++i) {
    for (int j = i + 1; j < 10; ++j) {
        int t1 = ch[id][i] - '0';
        int t2 = ch[id][j] - '0';
        tmp = tmp - (t1 - t2) * py[i] - (t2 - t1) * py[j];
        auto it = mp.find(tmp);
        if (it != mp.end()){
            int len = 0;
            int idx = it->second;
            while (len < 9 && ch[id][len] == ch[idx][len]){
```

```
                len++;
            }
            add_edge(id, idx, val[len]);
            add_edge(idx, id, val[len]);
        }
        tmp = tt;
    }
  }
  mp.insert({tt, id});
}
void dcstr(int x){
  std::priority_queue<pp> pq;
  dis[x] = 0;
  pq.push({dis[x], x});
  while (!pq.empty()){
    int u = pq.top().u;
    pq.pop();
    if (vis[u]) {
      continue;
    }
    vis[u] = true;
    for (int i = head[u]; i != -1 ; i = edge[i].next) {
      int v = edge[i].v;
      if (dis[v] > dis[u] + edge[i].w){
        par[v] = u;
        dis[v] = dis[u] + edge[i].w;
        pq.push({dis[v], v});
      }
```

```
        }
    }
}
int main() {
    long long tmp = 1;
    for (int i = 0; i < 10; ++i) {
        py.push_back(tmp);
        tmp *= 10;
    }
    std::reverse(py.begin(), py.end());
    std::cin >> n;
    memset(head, -1, sizeof(head));
    ec = 0;
    for (int i = 0; i < 10; ++i) {
        std::cin >> val[i];
    }
    for (int i = 1; i <= n; ++i) {
        std::cin >> ch[i];
        deal(i);
    }
    memset(dis, max_, sizeof(dis));
    par[1] = -1;
    dcstr(1);
    if (dis[n] == max_){
        std::cout << "-1";
        return 0;
    }
    std::cout << dis[n] << "\n";
```

```
    int x = n;
    while(x != -1){
        vec.push_back(x);
        x = par[x];
    }
    reverse(vec.begin(), vec.end());
    std::cout << vec.size() << "\n";
    for (int i = 0; i < vec.size(); ++i){
        std::cout << vec[i] << (i == vec.size() - 1? "\n" : " ");
    }
    return 0;
}
```

9368791	15:30:37 17 май 2021	Aleksey	1806. Мобильные телеграфы	G++ 9.2 x64	Accepted		0.843	34 104 КБ

4. Описание алгоритма

   Решение можно разделить на 2 логические части - построение графа и поиск кратчайшего пути. Для построения графа при добавлении нового телеграфа проверяем все возможные комбинации для определения всех связей(меняем каждую цифру на другую и пробуем переставить каждую пару чисел в номере телеграфа). Если получился номер уже существующего телеграфа, то добавляем связь между вершинами (с весом -- длинной общего префикса). После построения графа воспользуемся алгоритмом Дейкстры для поиска кратчайшего пути.