Федеральное государственное автономное образовательное учреждение высшего образования
«Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики»
Не кафедра “Вычислительной техники”

Алгоритмы и структуры данных

Задание №4

Выполнила: Калугина Марина

Группа: P3202

г. Санкт-Петербург

2019 г.

1080. Раскраска карты

Обойдем граф в ширину. Если можно покрасить следующий город в синий или красный, не нарушив условие, то красим вершину и переходим к следующей. Если условие нарушается (встречаются 2 города одинакового цвета) - это значит, что покрасить карту в 2 цвета нельзя.

Если после покраски графа остались непосещенное вершины, то это значит, что граф несвязный и обойдем эти вершины в ширину отдельно.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1080.cpp

```
int colors[100];
std::vector<int> edge[100];
int n;

void btf(int st) {
    std::queue<int> q;
    q.push(st);
    colors[st] = 0;

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int i = 0; i < edge[v].size(); ++i) {
            int to = edge[v][i];
            if (colors[v] == colors[to]) {
                std::cout << "-1";
                exit(0);
            }
            if (colors[to] == -1) {
                colors[to] = colors[v] == 0 ? 1 : 0;
                q.push(to);
            }
        }
    }
}

int main(){
    std::cin >> n;

    std::fill_n(colors, 100, -1); // -1 -- не посещен, 0 -- красный, 1 -- синий

    for (int i = 0; i < n; ++i) {
        int e = -1;
        while (e != 0) {
            std::cin >> e;
            if (e != 0) {
                edge[i].push_back(e - 1);
                edge[e - 1].push_back(i);
            }
        }
```

```
    }

    btf(0);

    for(int i = 0; i < n; ++i){
        if(colors[i] == -1) {
            btf(i);
        }
        std::cout << colors[i];
    }

    return 0;
}
```

1450. Российские газопроводы

Воспользуемся алгоритмом Беллмана-Форда.
Создадим матрицу res, res[i] -- максимальная газопроводность от s до i на данную итерацию
На каждой итерации рассматриваем все пути возможные из каждой посещенной вершины.
При нахождении нового максимального значения газопроводности в вершину i - обновляем значение res[i]

В итоге получаем вектор res, в котором находятся максимальные значения от s.
Если значение res[f] == -1, то эта вершина оказалась не посещенной => такого пути не существует.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1450.cpp

```
struct e {
    int a, b, w;
};

std::vector<e> v;

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<int> res(510, -1);
    for (int i = 0; i < m; ++i) {
        int a, b, w;
        std::cin >> a >> b >> w;
        v.push_back({a - 1, b - 1, w});
    }
    int s, f;
    std::cin >> s >> f;
    s--;
    f--;
    res[s] = 0;

    for (int i = 0; i < n - 1; ++i) {
```

```
        for (int j = 0; j < m; ++j) {
            if (res[v[j].a] != -1 && res[v[j].b] < res[v[j].a] + v[j].w) {
                res[v[j].b] = res[v[j].a] + v[j].w;
            }
        }
    }
    if (res[f] != -1) {
        std::cout << res[f];
    } else {
        std::cout << "No solution";
    }

    return 0;
}
```

1160. Network

Воспользуемся алгоритмом Краскала
Отсортируем ребра по возрастанию. Добавляем в граф ребра, начиная с минимального.
Если при добавлении следующего ребра получается цикл, то это ребро не нужно))
В итоге получаем остовное дерево, связанное проводами с минимальными длинами.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1160.cpp

```
struct e {
    int a, b, l;
};

int r[1010], t[1010];
int max = 0;

bool cmp(e a, e b) {
    return a.l < b.l;
}

int find(int x) {
    if (x != t[x]) t[x] = find(t[x]);
    return t[x];
}

int main() {
    int n, m;

    std::cin >> n >> m;
    std::vector<e> v;

    for (int i = 0; i < m; ++i) {
        int a, b, l;
        std::cin >> a >> b >> l;
```

```
        v.push_back({a - 1, b - 1, l});
    }

    std::sort(v.begin(), v.end(), cmp);

    for (int i = 1; i <= n; i++) {
        r[i] = 0;
        t[i] = i;
    }

    for (int i = 0; i < m; ++i) {
        int n1 = v[i].a;
        int n2 = v[i].b;

        if (find(n1) != find(n2)) {
            if (v[i].l > max) {
                max = v[i].l;
            }
            v[i].l *= -1;

            int x = find(n1);
            int y = find(n2);
            if (r[x] > r[y]) t[y] = x;
            else {
                t[x] = y;
                if (r[x] == r[y]) r[y]++;
            }
        }
    }

    std::cout << max << "\n" << n - 1 << "\n";

    for (int j = 0; j < m; ++j){
        if (v[j].l < 0){
            std::cout << v[j].a + 1 << " " << v[j].b + 1 << "\n";
        }
    }
    return 0;
}
```

1160. Currency Exchange

Стратегия задачи похожа на задачу 1450, за исключением того, что теперь максимум ищется по-другому, с учетом того, что за переводы из одной валюты в другую взимается комиссия

Т.е. воспользуемся алгоритмом Беллмана-Форда.
На каждой итерации рассматриваем все пути возможные из каждой посещенной вершины.
При нахождении нового максимального значения - обновляем значение nd
В итоге получаем вектор, в котором находятся максимальные значения от начальной

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1162.cpp

```
struct e {
    int a, b;
    double r, c;
};

std::vector<e> ve;
double nd[110];

int main() {
    int n, m, s;
    double v;
    std::cin >> n >> m >> s >> v;
    nd[s] = v;
    for (int i = 0; i < m; ++i) {
        int a, b;
        double rab, cab, rba, cba;
        std::cin >> a >> b >> rab >> cab >> rba >> cba;
        ve.push_back({a, b, rab, cab});
        ve.push_back({b, a, rba, cba});
    }

    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < ve.size(); ++j) {
            if (nd[ve[j].b] - (nd[ve[j].a] - ve[j].c) * ve[j].r < 0.000001) {
                nd[ve[j].b] = (nd[ve[j].a] - ve[j].c) * ve[j].r;
            }
        }
    }

    for (int i = 0; i < ve.size(); ++i) {
        if ((nd[ve[i].a] - ve[i].c) * ve[i].r - nd[ve[i].b]  > 0.000001) {
            std::cout << "YES";
            return 0;
        }
    };
    std::cout << "NO";
    return 0;
}
```

1806. Мобильные телеграфы

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.

https://github.com/KaluginaMarina/algorithms_and_data_structures/blob/master/1806.cpp

```
const int N = 50005;
const int M = 20000005;
const int max_ = 0x3f3f3f3f;

int n;
int val[15]; // значение для перехода
std::string ch[N]; // номера телеграфов
std::map<long long, int> mp; // мапа всех вершин
struct edge { // все связи в графе
    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;


//для определения всех связей между телеграфами
void deal(int id) {
    long long tmp = 0;
    // восстановление номера телеграфа
    for (int i = 0; i < 10; ++i) {
        tmp = tmp * 10 + ch[id][i] - '0';
    }
    const long long tt = tmp;
    //поиск связи при замене одной цифры
    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;
        }
    }
    //поиск связи при смене 2-х цифр местами
    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});
}

// Алгоритм Дейкстры для поиска кратчайшего маршрута
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;                         // счетчик узлов

    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;
}
```