Практический тур №3

Задача 1 «Радиолюбитель»

(уровень сложности: Муниципальный, 9-11 кл, средняя)

Условие задачи:

Джон решил заняться радиолюбительством, прочитал в сети Интернет о технологии ЛУТ (лазерно-утюжная технология) и решил попробовать. Суть технологии упрощённо состоит в следующем: сначала на лазерном принтере печатают маску (схему проводников), которую накладывают на заготовку платы, покрытую медью, и травят кислотным раствором. В результате медь растворяется там, где нет маски (т.е. чернил).
Однако, принтер у Джона очень старый, в результате чего некоторые дорожки перетравились и оказались разорванными. Он решил их дорисовать дорогим контактным клеем "Контактол". Естественно, он хочет потратить как можно меньше этого клея.

Джон раньше занимался математикой, поэтому быстро формализовал и упростил задачу. Во-первых, каждый раз достаточно рассматривать только два целых участка повреждённого проводника. Во-вторых, если для соединения каждой пары таких участков потратить минимум клея, то минимум клея уйдёт и на весь проводник.

Осталось дело за малым - научиться оптимально соединять два участка проводника. Участок платы представлен массивом символов N*M, например, так:

		X	X	X	X					X	X	X			
			X	X	X	X					X	X			
	X	X	X	X							X	X	X		
								X	X	X	X	X			
									X	X	X				

Здесь каждый символ 'X' обозначает сохранившийся участок проводника, на котором медь осталась. Два символа 'X' принадлежат одному и тому же участку, если они вертикально или горизонтально соседние (диагонально соседние таковыми не считаются). Гарантируется, что в выбранном участке имеется только два участка проводника.

Джон хочет использовать как можно меньше клея, чтобы объединить два участка проводника в один. В примере выше, он может сделать это, закрасив только три дополнительных клетки (они помечены символами ‘*’ на рисунке ниже).

		X	X	X	X					X	X	X			
			X	X	X	X	*				X	X			
	X	X	X	X			*	*			X	X	X		
								X	X	X	X	X			
									X	X	X				

Помогите Джону определить минимальное количество клеток, которые нужно закрасить, чтобы объединить два участка в один.

Анализ решения

Чтобы решить эту задачу, мы сначала помечаем каждый из двух участков с помощью рекурсивной функции "заливки" **label()**, которая распространяется от одного элемента к другому и устанавливает для каждого символа площадки значение 1 (для первой площадки) или 2 (для второй площадки).

Эта рекурсивная функция сначала вызывается, когда мы находим первый символ "X", после чего она помечает место, содержащее этот "X", единицами; затем продолжаем сканирование, пока не найдем другой "X", после чего эта же функция вызывается, чтобы пометить место, содержащее этот "X", двойками.

Каждый раз, когда вызывается функция label(), она помечает один символ, а затем рекурсивно пытается посетить соседей этого символа, останавливаясь каждый раз, когда мы попадаем на символ, который не является "X".

Одна из проблем, связанных с этим подходом, иногда заключается в том, что если входная сетка достаточно велика, то у нас может не хватить места в стеке, если функция label() повторяется слишком много раз.

К счастью, сетка здесь достаточно мала, так что это не вызывает беспокойства (если вы хотите быть особенно внимательными к этой проблеме, вы можете явно управлять размером стека для рекурсивных вызовов, хотя это немного больше кода).

Как только все наши точки помечены, мы хотим найти символ "1" и символ "2", которые находятся ближе всего друг к другу (т.е. два символа, которые нам нужно соединить с помощью пути для объединения двух площадок). Расстояние здесь измеряется путем взятия суммы абсолютной разницы в координатах - это обычно называют расстоянием "Манхэттен" или "L1".

Поскольку сетка достаточно мала, мы можем просто перебрать все возможные пары символов "1" и "2" и проверить расстояние между ними. Если бы сетка была намного больше, нам пришлось бы использовать несколько более сложные методы, такие как, например, поиск в ширину, чтобы быстро вычислить кратчайшее расстояние от каждого символа в сетке до площадки.

Текст программы

```
#include <stdio.h>
#define MAX_N 50
#define MAX_M 50

char G[MAX_N][MAX_M + 1];
int N, M;

inline int abs(int x)
{
    return x < 0 ? -x : x;
}

void label(int r, int c, char ch)
{
    if (G[r][c] != 'X') return;
    G[r][c] = ch;
    if (r > 0) label(r - 1, c, ch);
    if (c > 0) label(r, c - 1, ch);
    if (r < N - 1) label(r + 1, c, ch);
    if (c < M - 1) label(r, c + 1, ch);
}

 int mindist(void)
 {
     int r1, r2, c1, c2, min = MAX_N + MAX_M;

     for (r1 = 0; r1 < N; r1++)
         for (c1 = 0; c1 < M; c1++)
         {
             if (G[r1][c1] == '1')
                 for (r2 = 0; r2 < N; r2++)
                     for (c2 = 0; c2 < M; c2++)
                     {
                         if (G[r2][c2] == '2')
                             if (abs(r1 - r2) + abs(c1 - c2) < min)
                                 min = abs(r1 - r2) + abs(c1 - c2);
                     }
         }
     return min - 1;
 }
```

```
int main(void)
{
    FILE* stream;
    int r, c;
    char ch = '0';

    freopen_s(&stream, "grass.in", "r", stdin);

    scanf_s("%d %d", &N, &M);
    for (r = 0; r < N; r++)
        fgets(G[r], MAX_M, stream);
    fclose(stream);

    for (r = 0; r < N; r++)
        for (c = 0; c < M; c++)
            if (G[r][c] == 'X')
                label(r, c, ++ch);

    freopen_s(&stream, "grass.out", "w", stdout);
    printf("%d\n", mindist());
    fclose(stream);

    return 0;
}
```

Задача 2 «Странное сложение»

(уровень сложности: Муниципальный, 9-11 кл, средняя)

Условие задачи:

Маленький мальчик нашёл учебник по арифметике и прочитал главу про сложение многоразрядных чисел. К сожалению, он не смог толком понять, что такое перенос и постоянно забывал его делать. Он решил изучить, а сколько максимально чисел можно сложить правильно, не сделав ни одного переноса. В качестве данных он взял числа $w_1 \ldots w_N$ из какой-то таблицы в том же учебнике

Анализ решения

Задачу можно решить методом "грубой силой", путем перечисления всех возможных подмножеств чисел и отслеживания самого большого из них, которое не требует переноса.
Один из способов сделать это перечисление рекурсивно, как показано в приведенном ниже коде, где мы рассматриваем число № 1, а затем рекурсивно перечисляем все решения, которые содержат число № 1, а затем все решения, которые не содержат число № 1.

Попутно мы сокращаем поиск и возвращаемся назад, если когда-либо заметим, что наше текущее решение неосуществимо (т.е. если оно генерирует перенос), или если количество оставшихся чисел плюс количество чисел, которые мы уже сложили, не могут быть лучше, чем лучшее решение, которое мы создали до сих пор.

Еще один возможный "трюк" для перечисления всех 2^20 подмножеств чисел для этой задачи - просто посчитать от 0 до 2^20-1. Нули и единицы в двоичном представлении нашего счетчика целых чисел задают определенное подмножество, и мы сможем генерировать каждое возможное подмножество по мере подсчета.

Например, если счетчик равен (в двоичном виде) 10011000000000000000, то это означает, что мы рассматриваем подмножество, включающее числа с номерами 1, 4 и 5 (соответствуют позициям единичного бита).

Текст программы

```
#include <fstream>

using namespace std;
int n, w[20], best = 0;

/* Можно ли сложить x и y без переноса? */
int check(int x, int y)
{
    for (; x > 0 && y > 0; x /= 10, y /= 10)
        if (x % 10 + y % 10 >= 10) return 0;
    return 1;
}
```

```
/*
   Рекурсивная функция генерации подмножеств
   sum = накапливает сумму всех уже выбранных в подмножество чисел.
   count = количество уже просуммированных чисел.
*/
void rec(int x, int sum, int count)
{
    if (count > best) best = count;
    if (x >= n || count + n - x <= best) return;
    if (check(sum, w[x]))
        rec(x + 1, sum + w[x], count + 1);
    rec(x + 1, sum, count);
}

int main()
{
    ifstream fin("addition.in");
    ofstream fout("addition.out");

    fin >> n;
    for (int i = 0; i < n; i++)
        fin >> w[i];
    fin.close();

    rec(0, 0, 0);

    fout << best << endl;
    fout.close();
    return 0;
}
```

Задача 3 «Постройка дорог»

(уровень сложности: Муниципальный, 9-11 кл, очень сложная)

Условие задачи:

Область города Бездорожье славится тем, что в ней нет нормальных дорог. Она состоит из N населённых пунктов, которые соединены сетью из N-1 грунтовых дорог. Между каждой парой пунктов существует только один путь.
Жителям надоело каждые осень и весну "месить грязь" и они решили построить асфальтовые дороги вместо старых грунтовых. Строительство поручили строительной фирме. При этом жильцы не очень ей доверяли и периодически требовали отчёта о проделанной работе.

В итоге всё строительство дороги состоит из M шагов. На каждом шаге происходит одна из двух вещей:

- Строительная компания выбирает два населённых пункта и строит между ними асфальтовую дорогу.
- Жители требуют от компании отчёта в виде количества построенных на данный момент дорог между указанными населёнными пунктами.

Помогите строительной компании отвечать на вопросы.

.

Анализ решения

Алгоритм решения данной задачи основывается на представлении множества последовательно строящихся дорог в виде дерева и его анализа при выполнении запросов.

Сначала анализируется дерево исходных дорог. Используется подобие алгоритма Дейкстры: выполняется обход в ширину, при этом вычисляется множество вспомогательных массивов (векторов), обеспечивающих быстрое выполнение необходимых далее действий (массивы времена входа/выхода в вершину **tin/tout**, группировка, начала групп **fir**) - функции **dfs***.

Порядок обхода устанавливается таким образом, чтобы вершины групп были последовательны и не перемешивались с другими группами. Порядок сохранён в массиве **el**. Всё дерево виртуально преобразуется в бинарное (грубо говоря, потомки слева - меньше, справа - больше, но в приведенном решении условие более сложное, типа золотого сечения).

Далее, с использованием вспомогательных массивов выполняется последовательность запросов. За постройку дорог отвечают **upd**, за подсчёт - **Get, Get_cnt**. Функции имеют перегруженные аналоги: те, которые имеют больше 2 аргументов, принимают ещё границы по вершинам, в пределах которых нужно работать.

В целом решение достаточно громоздкое и сложное для понимания, всё организовано на массивах (векторах), которых слишком много. Не самым оптимальным образом реализованы **Upd/get**, делается ставка на инициализированность глобальных переменных (классы, может и инициализируются, но заполнитель может быть и не 0 - это не стандарт; в Visual Studio, к примеру, решение не работает), тем не менее, программа решает задачу на компиляторе Яндекс.Контеста.

Текст программы

```
#define sz(x) ((int)(x).size())

#include<vector>
#include<fstream>
using namespace std;

const int mod = 1e9 + 7;
const int N = 1e5 + 5;

vector<int> v[N], was(N), h(N), tin(N), tout(N), el, fir(N),
            cpt(N), g_le(N, mod), g_ri(N, mod);
int timer = 1, cnt[N], kol[N], cnt_down[N], timer2 = 1, group[N];
void dfs0(int pos)
{
    was[pos] = 1;
    cnt_down[pos] = 1;
    for (int x : v[pos])
    {
        if (!was[x])
        {
            dfs0(x);
            cnt_down[pos] += cnt_down[x];
        }
    }
}
void dfs(int pos)
{
    was[pos] = 1;
    tin[pos] = timer++;
    fir[pos] = sz(el);
    group[pos] = timer2;
    if (g_le[timer2] == mod)
        g_le[timer2] = pos;
    g_ri[timer2] = pos;
    el.emplace_back(pos);
    int i = -1;
```

```
    vector <int>  v1, v2;
    for (int x : v[pos])
    {
        i++;
        if (!was[x])
        {
            cpt[pos]++;
            par[x] = { pos, g[pos][i].second };
            h[x] = h[pos] + 1;
            if (cnt_down[x] >= (cnt_down[pos] + 1) / 2)
                v1.emplace_back(x);
            else v2.emplace_back(x);
        }

    }
    if (!sz(v1))
        timer2++;

    for (int x : v1)
    {
        dfs(x);
        el.emplace_back(pos);
    }

    for (int x : v2)
    {
        dfs(x);
        el.emplace_back(pos);
    }

    tout[pos] = timer++;
}
int lg[N * 3];
pair<int, int> st[N * 3][20];

pair<int, int> Get(int l, int r)
{
    int i = lg[r - l + 1];
    return min(st[l][i], st[r - (1 << i) + 1][i]);
}
```

```cpp
void DoST()
{
    lg[1] = 0;
    int n = sz(el);
    for (int i = 2; i <= n; i++) {
        lg[i] = lg[i / 2] + 1;
    }

    for (int i = 0; i < n; i++)
    {
        st[i][0] = { h[el[i]], el[i] };
    }

    for (int i = 1; i < 20; i++)
    {
        for (int j = 0; j + (1 << i) <= n; j++)
        {
            st[j][i] = min(st[j][i - 1], st[j + (1 << (i - 1))][i - 1]);
        }
    }
}
bool Fa(int a, int b)
{
    return tin[a] <= tin[b] && tout[a] >= tout[b];
}

int GetFa(int a, int b)
{
    int a1 = fir[a], b1 = fir[b];
    if (Fa(a, b)) return a;
    if (Fa(b, a)) return b;
    pair<int, int> o = Get(min(a1, b1), max(a1, b1));
    return o.second;
}

vector<long long> tree, gi;
void Push(int v, int L, int R)
{
    int c = (L + R) / 2;
    tree[v * 2] += (c - L) * gi[v];
    tree[v * 2 + 1] += (R - c) * gi[v];
    gi[v * 2] += gi[v];
    gi[v * 2 + 1] += gi[v];
    gi[v] = 0;
}
```

```
void Upd(int v, int L, int R, int l, int r)
{
    if (L == l && R == r)
    {
        tree[v] += R - L;
        gi[v]++;
        return;
    }
    Push(v, L, R);
    int c = (L + R) / 2;
    if (l < c)
        Upd(v * 2, L, c, l, min(r, c));
    if (c < r)
        Upd(v * 2 + 1, c, R, max(l, c), r);
    tree[v] = tree[v * 2] + tree[v * 2 + 1];
}
long long Get(int v, int L, int R, int l, int r)
{
    if (L == l && R == r)
        return tree[v];
    Push(v, L, R);
    int c = (L + R) / 2;
    long long an = 0;
    if (l < c)
        an += Get(v * 2, L, c, l, min(r, c));
    if (c < r)
        an += Get(v * 2 + 1, c, R, max(l, c), r);
    return an;
}
```

```
void Upd(int a, int b)
{
    if (a == b) return;
    int gr = group[a];
    int L = g_le[gr];
    if (Fa(L, b))
        L = b;
    int qq = fir[L] + 1;
    if (L != b) qq = fir[L];
    Upd(1, 0, sz(el), qq, fir[a] + 1);
    if (L == b) return;
    a = par[L].first;

    while (true)
    {
        int gr = group[a];
        int L = g_le[gr];
        if (Fa(L, b))
            L = b;
        int qq = fir[L] + 1;
        if (L != b) qq = fir[L];
        Upd(1, 0, sz(el), qq, fir[a] + 1);
        if (L == b) return;
        a = par[L].first;
    }
}
```

```
long long Get_cnt(int a, int b)
{
    if (a == b) return 0;
    long long an = 0;
    int gr = group[a];
    int L = g_le[gr];
    if (Fa(L, b))
        L = b;
    int qq = fir[L] + 1;
    if (L != b) qq = fir[L];
    an += Get(1, 0, sz(el), qq, fir[a] + 1);
    if (L == b) re an;
    a = par[L].first;

    while (true)
    {
        int gr = group[a];
        int L = g_le[gr];
        if (Fa(L, b))
            L = b;
        int qq = fir[L] + 1;
        if (L != b) qq = fir[L];
        an += Get(1, 0, sz(el), qq, fir[a] + 1);
        if (L == b) return an;
        a = par[L].first;
    }
}
void solve()
{
    ifstream fin("roadbuild.in");
    ofstream fout("roadbuild.out");

    int n, m;
    fin >> n >> m;

    for (int i = 0; i < (n - 1); i++)
    {
        int l, r;
        fin >> l >> r;
        l--; r--;
        v[l].emplace_back(r);
        v[r].emplace_back(l);
        g[l].emplace_back(r, i);
        g[r].emplace_back(l, i);
    }
```

```
    dfs0(0);
    for (int i = 0; i < (n); i++)
        was[i] = 0;
    dfs(0);
    DoST();
    tree.resize(sz(el) * 4);
    gi.resize(sz(el) * 4);
    vector<pair<int, int>> road;
    for (int i = 0; i < (m); i++)
    {
        char type;
        fin >> type;
        if (type == 'P')
        {
            int a, b;
            fin >> a >> b;
            a--; b--;
            int c = GetFa(a, b);
            if (a != c) Upd(a, c);
            if (b != c) Upd(b, c);
        }
        else
        {
            int a, b;
            fin >> a >> b;
            a--; b--;
            if (h[a] > h[b]) swap(a, b);
            int c = GetFa(a, b);
            long long an = 0;
            if (a != c) an += Get_cnt(a, c);
            if (b != c) an += Get_cnt(b, c);
            fout << an << '\n';
        }
    }
    fin.close();
    fout.close();
}

int main()
{
    solve();
}
```

Задача 4 «Ремонт Дорог»

(уровень сложности: Муниципальный, 9-11 кл, сложная)

Условие задачи:

Каждое утро Вася отправляется из дома на работу. Путь состоит из N остановок. Будем считать дом остановкой № 1, а место работы - № N

.

Все остановки соединены M дорогами, с каждой из которых ассоциировано время проезда. Никакие две остановки не соединены непосредственно более чем одной дорогой, и существует маршрут дорог от любой остановки к любой другой. Когда Вася едет от одной остановки к другой, он всегда выбирает маршрут с минимальным временем движения.

Злобные дорожники решили сделать Васе маленькую неприятность, начав ремонт одной из M дорог, тем самым удваивая время проезда по ней. Они хотят выбрать такую дорогу, чтобы максимально увеличить время движения, которое Вася проедет от дома до места работы. Помогите определить, насколько они способны увеличить время проезда на работу.

Вася активно пользуется системой Яндекс.Пробки и до начала пути знает об изменении времени движения по одной из дорог.

Анализ решения

Обратите внимание, что как только мы выбрали ребро, мы можем легко выбрать кратчайший путь за O (M log N) или O (N^2) время, просто изменив длину ребра и выполнив алгоритм поиска кратчайшего пути Дейкстры. Однако, поскольку существует M ребер, это дает общую сложность O (M^2), и программа получается слишком медленной.

Чтобы уменьшить сложность, мы можем заметить, что если ребро, которое мы решили удвоить, не находится на исходном кратчайшем пути от 1 до N, то конечная длина кратчайшего пути остается неизменной. Это означает, что нам нужно только попробовать удвоить ребра на исходном кратчайшем пути от 1 до N, и их всего O (N). Это дает нам лучшую сложность либо O (NM log N), либо O (N^3), последнее решение реализовано ниже.

Для эффективной обработки больших массивов данных (как обычно в последних задачах) нужно оценить целесообразность использования специальных классов-контейнеров, позволяющих быстро выполнять операции, которые на обычных массивах выполняются долго. В данной программе это проиллюстрировано использованием класса **vector** для обработки одномерных и двумерных массивов.

Текст программы

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;

FILE *in =  fopen ("rfix.in", "r"),
     *out = fopen ("rfix.out", "w");

const int MAXN = 505;

int N, M, edge [MAXN][MAXN], dist [MAXN], prev [MAXN];
bool visited [MAXN];

int best_path (int start, int end)
{
    memset (dist, 63, sizeof (dist));
    memset (visited, false, sizeof (visited));
    memset (prev, -1, sizeof (prev));
    dist [start] = 0;

    while (true)
    {
        int close = -1;

        for (int i = 0; i < N; i++)
            if (!visited [i] && (close == -1 || dist [i] < dist [close]))
                close = i;

        if (close == -1) break;

        visited [close] = true;

        for (int i = 0; i < N; i++)
        {
            int ndist = dist [close] + edge [close][i];

            if (ndist < dist [i])
            {
                dist [i] = ndist;
                prev [i] = close;
            }
        }
    }
    return dist [end];
}
```

```
int main ()
{
    memset (edge, 63, sizeof (edge));
    fscanf (in, "%d %d", &N, &M);
    for (int i = 0; i < M; i++)
    {
        int a, b, len;
        fscanf (in, "%d %d %d", &a, &b, &len);
        a--; b--; edge [a][b] = edge [b][a] = len;
    }

    int original = best_path (0, N - 1);
    vector <int> path;
    for (int i = N - 1; i != -1; i = prev [i])
        path.push_back (i);

    int most_doubled = original;
    for (int i = 0; i + 1 < (int) path.size (); i++)
    {
        int a = path [i], b = path [i + 1];
        edge [a][b] *= 2;
        edge [b][a] *= 2;
        most_doubled = max (most_doubled, best_path (0, N - 1));
        edge [a][b] /= 2;
        edge [b][a] /= 2;
    }
    fprintf (out, "%d\n", most_doubled - original);
    return 0;
}
```