BUAA Algorithm 2023

北京航空航天大学软件学院 2022届 **算法分析与设计** 课程常用算法的模板

- 仓库中的 `.c/.cpp` 文件为平时积累的板子。本 `README.md` 为笔者为应付算法期末考试准备的板子，仅收录了笔者认为有价值、考试易出现的算法，语言为 C/C++，板子中省略了`#include <bits/stdc++.h>` 等语句，样例输入输出均以注释的形式体现。

> 实际上，考试中能用到的板子也就一两个，甚至根本用不到，重要的是随机应变能力。

一、卡特兰数

```c
//catalan-good.c
long long catalan[100] = {1, 1, 2};
int main()
{
    int n, i, j;
    scanf("%d", &n);
    for (i = 3; i <= n; i++)
        for (j = 0; j < i; j++)
            catalan[i] += catalan[j] * catalan[i - j - 1];
    printf("n  : P(n)\n");
    for (i = 0; i <= n; i++)
        printf("%-3d: %lld\n", i, catalan[i]);
    return 0;
}
```

```
样例输入:
10
样例输出:
n  : P(n)
0  : 1
1  : 1
2  : 2
3  : 5
4  : 14
5  : 42
6  : 132
7  : 429
8  : 1430
9  : 4862
10 : 16796
```

二、优先队列自定义函数

Alt text

三、计算方阵的n次幂

```cpp
// 方针的n次方.cpp
#define maxn 130
typedef long long ll;
const int mod = 1e9 + 7;
int n;
ll k, a[maxn][maxn], b[maxn][maxn], tmp[maxn][maxn];
// 思路: A^(k0*2^0 +...+ kn*2^n) = A^(k0*2^0) *...* A^(kn*2^n), k0,...,kn =
0 or 1
// 设置单位阵
void setI(ll a[130][130]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (i == j)
                a[i][j] = 1;
            else
                a[i][j] = 0;
}
// 将矩阵b复制到矩阵a中
void move(ll a[130][130], ll b[130][130]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            a[i][j] = b[i][j];
}
// 计算矩阵乘法a*b，结果存储在tmp矩阵中
void multiply(ll a[130][130], ll b[130][130]) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            tmp[i][j] = 0;
            for (int k = 0; k < n; k++) {
                tmp[i][j] += a[i][k] * b[k][j];
                tmp[i][j] %= mod;
            }
        }
}
void bpow(ll p) {
    // 先将b置为单位阵
    setI(b);
    while (p) {
        if (p & 1) // p为奇数
        {          // 等价于b = b * a
            multiply(b, a);
            move(b, tmp);
        }
        // 等价于a = a * a
        multiply(a, a);
        move(a, tmp);
        p >>= 1; // p /= 2
    }
}
int main() {
```

```
    cin >> n;
    k = n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> a[i][j];
    bpow(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << b[i][j] << ' ';
        cout << '\n';
    }
    return 0;
}
```

样例输入：

```
1
3
1 2 3
1 0 1
2 1 2
```

样例输出：

```
36 29 54
16 11 22
35 25 49
```

四、大数乘法

```
char s1[2005], s2[2005];
int a[2005], b[2005], c[4005]; // 分别以数组形式存储a，b和结果c，|c|<=|a|+|b|
int main() {
    int t;
    scanf("%d", &t);
    while (t--)
    {
        int i, j, k;
        scanf("%s%s", s1, s2);
        for (i = 0; i < 4003; i++)
            c[i] = 0;
        int n = strlen(s1), m = strlen(s2);
        k = n + m; // 相乘后的位数不会大于k
        // 把字符串s1和s2逆序用数字排列
        for (i = 0; i < n; i++)
            a[i] = s1[n - i - 1] - '0';
        for (i = 0; i < m; i++)
            b[i] = s2[m - 1 - i] - '0';
        // 乘运算
        for (i = 0; i < n; i++)
            for (j = 0; j < m; j++)
                c[i + j] += a[i] * b[j];
        for (i = 0; i <= k; i++)
            if (c[i] >= 10) {
```

```
                c[i + 1] += c[i] / 10;
                c[i] %= 10;
            }
        /*去除前导0*/
        i = k;
        while (c[i] == 0)
            i--;
        /*判断两个非负数之积是否为0，以及逆序打印c[]*/
        if (i < 0)
            printf("0");
        else
            for (; i >= 0; i--)
                printf("%d", c[i]);
        printf("\n");
    }
    return 0;
}
样例输入：
2
0
0
123456789
987654321
样例输出：
0
121932631112635269
```

五、快速幂

```
// 快速幂.c
long long fpm(long long a, long long b, long long m) {
    long long ans = 1;
    for (; b; b >>= 1, a = a * a % m)
        if (b & 1) // 选取二进制为1对应的幂来相乘
            ans = ans * a % m;
    return ans;
} // 求解a^b mod m, m为素数
int main() {
    long long a = 3, b = 998244351, m = 998244353;
    printf("%lld", (5 * fpm(a, b, m)) % m);
    return 0;
}
样例输出：665496237
```

六、斐波那契数

```
// #define Max 998244353
struct Matrix {
    unsigned long long M[2][2]; // 数据类型要一致!
```

```
};                                  // 结构体 矩阵。
struct Matrix mul(struct Matrix a, struct Matrix b) // 矩阵相乘，返回值为矩阵
{
    struct Matrix c; // 创建新的矩阵来放矩阵相乘的结果
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++) {
            c.M[i][j] = 0; // 初始化新矩阵
            for (int k = 0; k < 2; k++)
            {
                c.M[i][j] = c.M[i][j] + a.M[i][k] * b.M[k][j];
                c.M[i][j] = c.M[i][j]; //% Max; // 取模别忘了
            }
        }
    return c;
}
// 快速幂
struct Matrix qul(unsigned long long t, struct Matrix a) {
    struct Matrix res;
    // 单位矩阵
    res.M[0][0] = 1, res.M[0][1] = 0;
    res.M[1][0] = 0, res.M[1][1] = 1;
    while (t > 0) {
        if (t % 2)
            res = mul(res, a);
        a = mul(a, a);
        t = t / 2;
    }
    return res;
}
// 最大可计算到1e19，若不取模约为93
int main() {
    unsigned long long n, sum = 0;
    scanf("%llu", &n);
    struct Matrix a,d;
    a.M[0][0] = 1, a.M[0][1] = 1;
    a.M[1][0] = 1, a.M[1][1] = 0;
    if (n == 1 || n == 2)
        printf("1");
    else
    {
        d = qul(n - 2, a);
        sum = (d.M[0][0] + d.M[0][1]); // % Max;
        cout << sum;
    }
    return 0;
}//样例：90 2880067194370816120
```

七、多数问题-分治

```
// 找到数列中出现次数最多的元素（众数）
int a[100];
```

```
int cnt(int a[], int pivot, int left, int right) {
    int num = 0;
    for (int i = left; i <= right; i++)
        if (a[i] == pivot)
            num++;
    return num;
}
int majority(int a[], int left, int right) {
    int lpivot, rpivot, mid = (left + right) / 2;
    if (left == right)
        return a[left] < a[right] ? a[left] : a[right];
    else {
        lpivot = majority(a, left, mid);
        rpivot = majority(a, mid + 1, right);
    }
    if (lpivot == rpivot)
        return lpivot; // m1,m2为两分支出现次数较多的解，接下来求解哪个更多
    else {
        int ltimes = cnt(a, lpivot, left, right);
        int rtimes = cnt(a, rpivot, left, right);
        return ltimes > rtimes ? lpivot : rpivot;
    }
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    printf("%d", majority(a, 0, n - 1));
    return 0;
}
样例输入:
5
样例输出:
1 2 2 3 2 4
2
```

八、动态规划

1、0-1背包

```
long long v[10005], w[10005], f[10005], t, m;
long long dp[1005][1005];
int main() {
    cin >> m >> t; // m商品数目 t背包容量
    for (int i = 1; i <= m; i++)
        cin >> v[i] >> w[i]; // w是重量,V是价值
    /*// 朴素：二维dp
    for (int i = 1; i <= m; i++) // 对第i个商品
    {
        for (int j = t; j >= 0; j--) // 目前背包容量为t
```

```
        {
            if (j >= w[i]) // 容量足够放下该商品，看放入或不放入哪个价值大
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);
            else // 放不下，只好不放入
                dp[i][j] = dp[i - 1][j];
        }
    }
    cout << dp[m][t] << '\n';
    */
    // 优化后：
    for (int i = 1; i <= m; i++)
        for (int k = t; k >= w[i]; k--)
            // 更新前，f[k]中存i-1的价值；更新后存i的价值，后续覆盖，再用不到i-1的值
            // 容量只需遍历到i的重量，再往下就放不下了，语句就是f[j]=f[j]
            f[k] = max(f[k], f[k - w[i]] + v[i]);
    cout << f[t] << '\n';
    return 0;
}
输入样例
1
3 5
3 1
5 3
4 2
输出样例
9
```

2、最长公共子序列、子串

```
char s1[2005], s2[2005];
char subsequence[2005], substring[2005];
int f[2005][2005]; // 最长公共子串LCsubstring，删去一段前缀和一段后缀得到
// f[i][j]表示子串s1[0,i-1](第i个字符结尾)和子串s2[0,j-1]的最长公共子串
int g[2005][2005]; // 最长公共子序列LCSequence，删去字符得到，更宽泛
// g[i][j]同上，只不过是最长公共子序列
int flag1, flag2[2005][2005];
// flag1存储最后一位两字符串相等的位置
// flag2存储
int LCSubstring() {
    int res = 0, i, j;
    int m = strlen(s1), n = strlen(s2);
    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            if (s1[i - 1] == s2[j - 1])
            { // s1[i-1]==s2[j-1]，最长子串就是前面子串加上该字符
                f[i][j] = f[i - 1][j - 1] + 1;
                if (res <= f[i][j])
                {
                    res = f[i][j];
                    flag1 = i; // 只要res增加就更新flag1位置
                }
```

```
                res = max(f[i][j], res);
            } // res时刻存储最长的子串
            else
                f[i][j] = 0;
        }
    }
    return res;
}
int LCSubsequence() // 最长公共子序列，算导上的
{
    int i, j, m = strlen(s1), n = strlen(s2);
    for (i = 1; i <= m; i++)

        for (j = 1; j <= n; j++) {
            if (s1[i - 1] == s2[j - 1]) { // 只有这一步是真正增加了子序列长度
                g[i][j] = g[i - 1][j - 1] + 1;
                flag2[i][j] = 0;
            }
            else if (g[i - 1][j] > g[i][j - 1]) { // 这步和下一步只是更新g[i]
[j]
                g[i][j] = g[i - 1][j];
                flag2[i][j] = 1;
            }
            else {
                g[i][j] = g[i][j - 1];
                flag2[i][j] = -1;
            }
        }
    return g[m][n];
}
int cnt2; // cnt2就是当前子序列的位置
void PrintLCSequence(int i, int j) {
    if (i == 0 || j == 0)
        return;
    if (flag2[i][j] == 0) {
        PrintLCSequence(i - 1, j - 1);
        subsequence[cnt2] = s1[i - 1];
        cnt2++;
        // cout << s1[i - 1];//去掉注释就是直接输出
    }
    else if (flag2[i][j] == 1)
        PrintLCSequence(i - 1, j);
    else // 值为-1
        PrintLCSequence(i, j - 1);
}
int main() {
    memset(g, 0, sizeof(g));
    memset(f, 0, sizeof(f));
    memset(subsequence, 0, sizeof(subsequence));
    flag1 = 0;
    cnt2 = 0;
    memset(flag2, 0, sizeof(f));
    cin >> s1 >> s2;
    int ans = LCSubstring();
```

```
    cout << ans << "\n";
    for (int i = flag1 - ans; i < flag1; i++)
        cout << s1[i];
    cout << '\n'
         << LCSubsequence() << "\n";
    PrintLCSequence(strlen(s1), strlen(s2));
    cout << subsequence << '\n';
    return 0;
}
/*样例输入:
2
iwannafuckyouoff
youaresonofabitch
lapispureruby
jadestarsepia
样例输出:
3
you
5
youof
2
pi
4
asre*/
```

3、最长回文子串

```
char s[2005];
int dp[2005][2005]; // 起始为i，终末为j的字符串是否为回文串
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int ans = 0;
    memset(dp, 0, sizeof(dp));
    cin >> s;
    int n = strlen(s), len, i, j;
    for (len = 1; len <= n; len++)
        for (i = 0; i <= n - len; i++) {
            j = i + len - 1;
            if (i == j)  dp[i][j] = 1;
            else if (s[i] == s[j] && dp[i + 1][j - 1] == len - 2)
                dp[i][j] = len;
            ans = max(ans, dp[i][j]);
        }
    cout << ans << '\n';
    return 0;
}
/*
输入样例
3
asdfdsaasdfdsa
```

```
aba
aabb
输出样例
14
3
2
*/
```

4、钢管切割-cutRod

```
#define INF 1000000000003
long long p[10006];
long long r[10006]; // 最优切割方案
int s[10006];       // 切割方案
int cnt;            // 把钢管切成cnt段
long long rod(long long p[], int n) {
    for (int i = 0; i < 10005; i++)
        s[i] = 0;
    r[0] = 0;
    for (int j = 1; j <= n; j++) {
        long long q = -INF;
        for (int i = 1; i <= j; i++)
            if (q < p[i] + r[j - i]) {
                q = p[i] + r[j - i];
                s[j] = i;
            }
        r[j] = q;
    }
    return r[n];
}
int main() {
    int n, i;
    scanf("%d", &n);
    // 第二行 n 个正整数 p1,p2,…,pn（1≤pi≤1e7），表示钢管的价格。
    for (i = 1; i <= n; i++)
        scanf("%lld", &p[i]);
    // 第一行一个正整数，表示最大总销售价格。
    printf("%lld\n", rod(p, n));
    i = n;
    while (i > 0) {
        cnt++;
        i = i - s[i];
    }
    i = n;
    // 第二行一个正整数 k，表示钢管被分割成 k 段。
    printf("%d\n", cnt);
    // 第三行 k 个正整数 a1,…,ak，表示钢管的分割方式，需保证 ∑ai=n
    while (i > 0) {
        printf("%d ", s[i]);
        i = i - s[i];
    }
```

```
    return 0;
}
```

5、n条流水线m个装配站输出字典序最小的方案

```
const long long inf = 1e18;
int p[12][22];       // 站点用时
int t[12][12];       // 站点间用时
long long f[12][22]; // 存储最优时间消耗
int l[12][22];       // 存储站点信息
int path[22];        // 存储每个站点在哪条流水线上
int main() {
    /*本题思想：既然要输出字典序最小的，不妨从后往前（从右往左）动态规划
    这样到了最左端再从左往右遍历，决策权从左向右，符合字典序最小要求（左边尽可能小）
    */
    int T;
    cin >> T;
    while (T--)
    {
        int m, n, i, j, k;
        cin >> m >> n; // n 条流水线有 m 个装配站
        for (i = 1; i <= n; i++)
            for (j = 1; j <= m; j++)
                cin >> p[i][j];
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                cin >> t[i][j];
        for (i = 1; i <= n; i++) {
            f[i][m] = p[i][m];
            l[i][m] = i;
        }
        // f[1][m] = p[1][m], l[1][m] = 1;
        // f[2][m] = p[2][m], l[2][m] = 2;
        // f[3][m] = p[3][m], l[3][m] = 3;
        for (j = m - 1; j >= 1; j--) { // 从右往左编号为j的装配站
            for (i = 1; i <= n; i++) { // 从1到n编号为i的流水线
                f[i][j] = inf;
                for (k = 1; k <= n; k++) { // 对流水线i，下一站（往左）若去往流
水线k
                    long long a = f[k][j + 1] + t[i][k] + p[i][j];
                    if (a < f[i][j]) {
                        f[i][j] = a;
                        l[i][j] = k;
                    }
                }
            }
        }
        long long ans = inf;
        int cur = 0;
        for (i = 1; i <= n; i++)
            if (f[i][1] < ans) {
```

```
                ans = f[i][1];
                cur = i;
            }
        // 此时ans为最优时间，cur是从右往左最后一个装配站的流水线序号
        for (i = 1; i <= m; i++) {
// 对装配站从左往右遍历，由于此前dp的时候挑选了当时情况下的最小流水线，
// 故path依次存的就是字典序最小路径
            path[i] = cur;
            cur = l[cur][i];
        }
        cout << ans << '\n'; // 制造一个成品的最少时间
        for (i = 1; i <= m; i++)
            cout << "Station" << i << ": Line" << path[i] << '\n';
    }
    return 0;
}
```

输入样例

```
1
3 3
10 1 10
8 5 10
7 10 8
0 2 2
1 1 2
1 2 3
```

输出样例

```
19
Station1: Line3
Station2: Line1
Station3: Line1
```

6、MCM-矩阵相乘最优次数问题

```
#define INF 9223372036854775800
long long p[1006], Min[1006][1006], Max[1006][1006], s[1006][1006];
// Min[1][n]为最优解，Max[1][n]为最坏解，s存储最优解分割点
// p为n个矩阵的规模，Ai=pi-1*pi
// matrix-chain multiplication
void printAns(int i, int j)
{
    if (i == j) cout << "A" << i;
    else {
        cout << "(";
        printAns(i, s[i][j]);
        printAns(s[i][j] + 1, j);
        cout << ")";
    }
}
int main()
{
    int n, i, j, l, k;
```

```
    cin >> n; // 矩阵的个数
    for (i = 0; i <= n; i++) cin >> p[i];
        // 表示矩阵 Ai 的行数和列数为 ai 和 ai+1（前矩阵列数=后矩阵行数）
    for (i = 1; i <= n; i++) Min[i][i] = 0;
    for (l = 2; l <= n; l++) {
        for (i = 1; i <= n - l + 1; i++) {
            j = i + l - 1;
            Min[i][j] = INF;
            for (k = i; k <= j - 1; k++) {
                long long q = Min[i][k] + Min[k + 1][j] + p[i - 1] * p[k]
* p[j];
                if (q < Min[i][j]) {
                    Min[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    } // 计算最优
    for (l = 2; l <= n; l++) {
        for (i = 1; i <= n - l + 1; i++) {
            j = i + l - 1;
            Max[i][j] = -1;
            for (k = i; k <= j - 1; k++) {
                long long q = Max[i][k] + Max[k + 1][j] + p[i - 1] * p[k]
* p[j];
                if (q > Max[i][j]) {
                    Max[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    } // 计算最坏
    cout << Min[1][n] << '\n';
    // cout << fixed << setprecision(4) << (double)Max[1][n] / Min[1][n]
<< endl;
    // C3-E 只要求输出上一行最多是最少的多少倍。下面输出最优解：
    printAns(1, n);
    return 0;
}
```

八、排序

1、快速排序

```
ver1: pivot在第一位
void QuickSort(int A[], int n) {
    QSort(A, 0, n - 1);//首位、末位索引
}
void QSort(int A[], int low, int high) {
    int pivot;
    if (low < high) {
```

```
        pivot = Partition(A, low, high);
        QSort(A, low, pivot - 1);
        QSort(A, pivot + 1, high);
    }
}
int Partition(int A[], int low, int high) {
    int pivot;
    pivot = A[low];
    // 从线性表的两端交替地向中间扫描
    while (low < high) {
        while (low < high && A[high] >= pivot)
            high--;
        A[low] = A[high];
        while (low < high && A[low] <= pivot)
            low++;
        A[high] = A[low];
    }
    A[low] = pivot;
    return low;
}
// 调用的时候，直接：QuickSort(a, n)，a为数组，n为元素个数
```

```
ver2: pivot在最后一位
void swap(int *a, int *b) {
    int tmp = *b;
    *b = *a;
    *a = tmp;
}
int Partition(int a[], int p, int r) {
    int pivot = a[r]; // 将pivot设置为最后一个元素
    int i = p - 1;
    for (int j = p; j < r; j++) {
        if (a[j] < pivot) {
            i++;
            swap(&a[j], &a[i]);
        }
    }
    swap(&a[++i], &a[r]);
    return i;
}
void QuickSort(int a[], int p, int r) {
    if (p < r) {
        int q = Partition(a, p, r);
        QuickSort(a, p, q - 1);
        QuickSort(a, q + 1, r);
    }
}
// 调用的时候，直接 QuickSort(a, 0, n - 1)，分别为首尾下标
```

2、归并排序

```
void merge(int a[], int p, int q, int r) {
    int n1 = q - p + 1; // 左数组的元素个数
    int n2 = r - q;
    int i, j, k;
    int left[100000] = {0}, right[100000] = {0};
    for (i = 0; i < n1; i++) left[i] = a[p + i];
    for (j = 0; j < n2; j++) right[j] = a[q + j + 1];
    left[n1] = right[n2] = INF; // INF需要定义2147483647
    i = j = 0;
    for (k = p; k <= r; k++) {
        if (left[i] <= right[j]) {
            a[k] = left[i];
            i++;
        }
        else {
            a[k] = right[j];
            j++;
        }
    }
}
void MergeSort(int a[], int p, int r) {
    if (p < r) {
        int q = (p + r) / 2;
        MergeSort(a, p, q);
        MergeSort(a, q + 1, r);
        merge(a, p, q, r);
    }
} // 调用MergeSort(a, 0, n - 1);即可，分别为首尾下标
```

九、图算法

1、BFS,DFS

```
#define MAXN 10006
const int INF = 2147483646;
vector<int> graph[MAXN]; // vector邻接表存矩阵点
int disBFS[MAXN];
int disDFS[MAXN][2], timee;   // 0-d-发现时间，1-f-完成时间，
int piBFS[MAXN], piDFS[MAXN]; // 最短路径的前驱结点
vector<int> topo;             // 存储拓扑排序后的结果
queue<int> q;
int verNum; // 顶点个数
void insertEdge(int head, int v) { // 加边操作
    graph[head].push_back(v);
}
void BFS(int s) // 适用于无权重图求最短路径
{
    int i, j;
    for (i = 1; i <= verNum; i++) { // 此处应为图编号最大数，假设从1开始
        disBFS[i] = INF;
```

```
        piBFS[i] = -1;
    }
    disBFS[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (j = 0; j < graph[u].size(); j++)
            if (disBFS[graph[u][j]] == INF) {
                disBFS[graph[u][j]] = disBFS[u] + 1;
                piBFS[graph[u][j]] = u;
                q.push(graph[u][j]);
            }
    }
}
void DFSVISIT(int u) {
    timee++;
    disDFS[u][0] = timee; // u.d=time
    for (int i = 0; i < graph[u].size(); i++)
        if (disDFS[graph[u][i]][0] == INF) {
            piDFS[graph[u][i]] = u;
            DFSVISIT(graph[u][i]);
        }
    timee++;
    disDFS[u][1] = timee;
    topo.insert(topo.begin(), u); // 为了拓扑排序
}
void DFS() {
    int i;
    for (i = 1; i <= verNum; i++) {
        disDFS[i][0] = disDFS[i][1] = INF;
        piDFS[i] = -1;
    }
    timee = 0;
    for (i = 1; i <= verNum; i++)
        if (disDFS[i][0] == INF) DFSVISIT(i);
}
int main() {
    /*insertEdge并统计verNum*/
    /*BFS: 调用BFS(1)，disBFS[u]即源节点1到u的距离，piBFS[u]即u的前驱结点*/
    /*DFS: 调用DFS()，disDFS[i][0]和disDFS[i][1]分别为发现时间和完成时间，
    piDFS[i]为i的前驱结点，topo存储的是其中一种topo排序*/
}
```

2、Dijkstra

```
typedef long long ll;
typedef pair<int, ll> PIL;
typedef pair<ll, int> PLI;
const ll inf = 2e18; // 2*10^18
class Dijstra {
```

```
public:
    int n;
    vector<vector<PIL>> g; // 图的邻接表
    vector<long long> dis; // 各点到源点的距离
    Dijstra(int n)         // 迪杰斯特拉类的构造函数，即初始化
    {
        this->n = n;
        g.resize(n);
        dis.resize(n, inf);
    }
    void add(int u, int v, long long w) { g[u].emplace_back(v, w); }
    void solve(int s) {
        vector<bool> vis(n);                    // 是否已找到过，找到过则为true
        priority_queue<PLI, vector<PLI>, greater<PLI>> q;
        // 小顶堆优先队列，分别为与源点距离、点序号
        dis[s] = 0;                             // 源点距离置为0
        q.push({0, s});                         // 将源点距离、序号存入优先队列
        while (!q.empty()) {
            int u = q.top().second;
            q.pop();
            if (vis[u]) continue;
            vis[u] = true; // 相当于S=Su{u}
            for (auto e : g[u]) {
                int v = e.first;
                ll w = e.second;
                if (dis[v] > dis[u] + w) {
                    dis[v] = dis[u] + w;
                    q.push({dis[v], v});
                }
            }
        }
    }
};
int main()
{
    int n, m, s; // n=结点数，m=边数，s=源点编号
    cin >> n >> m >> s;
    // --s;
    Dijstra dj(n + 1);
    // 构建Dijkstra类实例，注意！！！此处是n+1，涉及到q、dis、vis的初始化
    for (int i = 0; i < m; i++) {
        int x, y, w; // x,y为边的两端，w为weight
        cin >> x >> y >> w;
        // --x; //如构造实例不传入n+1，则s x y均需减1，且后面输出是i+1
        // --y;
        dj.add(x, y, w); // 无向图，插入两遍
        dj.add(y, x, w);
    }
    dj.solve(s); // 调用solve方法，s为源点
    vector<int> ans;
    for (int i = 1; i <= n; i++) cout << dj.dis[i] << " ";
    return 0;
}
/*样例：
```

```
5 5 1
1 2 5
3 5 8
3 2 1
1 3 2
2 5 6
输出: 0 3 2 2000000000000000000 9; 4与源点无通路
*/
```

3、最小生成树-Kruskal

```
typedef long long ll;
int father[100005]; // 并查集
typedef struct d {
    int u, v;
    ll w;
} D;
D edge[500005];
bool cmp(D a, D b) { return a.w < b.w; }
int Find(int i) // 找最先祖先，需要在这里更新！！
{
    if (father[i] == i) return father[i];
    return father[i] = Find(father[i]);
} // 递归寻找一个节点的祖先，若祖先就是自己则返回
int main() {
    int n, m, i;
    ll ans = 0;
    cin >> n >> m;
    for (i = 1; i <= n; i++)
        father[i] = i; // 初始时，每个点都是单独的连通分量
    for (i = 0; i < m; i++)
        cin >> edge[i].u >> edge[i].v >> edge[i].w;
    sort(edge, edge + m, cmp);
    for (i = 0; i < m; i++) {
        int u = edge[i].u, v = edge[i].v;
        int visu = Find(u), visv = Find(v); // 分别为u和v的祖先
        if (visu != visv) { // 若两个点有共同祖先，则不能将边加进去
            father[visu] = Find(visv); // u和v的祖先均改为v的祖先，合并操作，可写
成merge函数
            ans += edge[i].w;
        }
        // for (int j = 1; j <= n; j++)
        //     cout << father[j];
        // cout << '\n';
        }
    cout << ans << '\n'; // 输出最短路径长度
}
```

4、最大流-Dinic

```
const int MAXV = 110;
const int MAXE = 10080;
const long long INF = 1e15;
struct Edge {
    int from;
    int to;
    int flow;
    Edge *rev;
    Edge *next; // 该边下一个边
};
Edge E[MAXE];
Edge *head[MAXV];
Edge *cur[MAXV];
Edge *top = E; // 实际上是E数组的下标
int v, e, s, t, depth[MAXV];
bool BFS(int, int);
void Insert(int, int, int);
long long Dinic(int, int);
long long DFS(int, long long, int);
int main()
{
    int tt;
    cin >> tt;
    while (tt--) // problem: head没清空
    {
        int i;
        top = E;
        memset(head, 0, sizeof(head));//这句话必须有！！！
        cin >> v >> e >> s >> t;
        // 点数v，边数e，源点s，汇点t
        for (i = 0; i < e; i++) {
            int a, b, c;
            cin >> a >> b >> c;
            // 有向边a到b的最大容量为c
            Insert(a, b, c);
        }
        cout << Dinic(s, t) << '\n';
    }//输出的整数为s到t的最大流
    return 0;
}
long long Dinic(int s, int t) {
    long long ans = 0;
    while (BFS(s, t))
        ans += DFS(s, INF, t);
    return ans;
}
bool BFS(int s, int t) {
    memset(depth, 0, sizeof(depth));
    queue<int> q;
    q.push(s);
    depth[s] = 1;
    cur[s] = head[s];
    while (!q.empty()) {
```

```
            s = q.front();
            q.pop();
            for (Edge *i = head[s]; i != NULL; i = i->next) {
                if (i->flow > 0 && depth[i->to] == 0) {
                    depth[i->to] = depth[s] + 1;
                    cur[i->to] = head[i->to];
                    if (i->to == t)
                        return true;
                    q.push(i->to);
                }
            }
        }
        return false;
    }
    long long DFS(int s, long long flow, int t) {
        // flow参数代表上游的边对增广路上的流量的限制
        if (s == t || flow <= 0)
            return flow;
        long long rest = flow; // 实际上保存的就是最大可能的流入流量
        for (Edge *i = cur[s]; i != NULL; i = i->next) {
            if (i->flow > 0 && depth[i->to] == depth[s] + 1) {
                long long tmp = DFS(i->to, min(rest, (long long)i->flow), t);
                if (tmp <= 0)
                    depth[i->to] = 0;
                rest -= tmp;
                i->flow -= tmp;
                i->rev->flow += tmp;
                if (rest <= 0)
                    break;
            }
        } // 把所有后继结点都访问过之后的 rest 值即为无法排出的流入量的值
        return flow - rest;
    }
    void Insert(int from, int to, int flow) {
        top->from = from;
        top->to = to;
        top->flow = flow;
        top->rev = top + 1; // 后面会定义，是反向边
        top->next = head[from];
        head[from] = top++;
        // 构造反向边
        top->from = to;
        top->to = from;
        top->flow = 0;          // 反向边
        top->rev = top - 1;     // 这与前面构成一组互相反向的边
        top->next = head[to]; // E[top].pushback(head[to])
        head[to] = top++;
    }
```

5、字典序最大topo排序（kahn算法）

```
vector<int> graph[100005]; // 图的邻接表
int indegree[100005];      // 记录各点的入度
priority_queue<int> q;     // 维护入度为0的大顶堆
int main()
{
    int n, i, m;
    cin >> n >> m;//n个点，m个边
    for (i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;//表示u、v间存在有向边
        graph[u].push_back(v);
        ++indegree[v];
    }
    for (i = 1; i <= n; i++)
        if (indegree[i] == 0) q.push(i);
    while (!q.empty()) {
        int v = q.top();
        q.pop();
        cout << v << " ";
        for (i = 0; i < graph[v].size(); i++) {
            --indegree[graph[v][i]];
            if (indegree[graph[v][i]] == 0)
                q.push(graph[v][i]);
        }
    }
    return 0;
}//输出一行，为n个点的topo排序
```

十、计算几何

1、线段相交，平行

```
typedef pair<int, int> pii;
vector<pii> point;
// 判断线段p1p2和p3p4是否相交
int direction(pii pi, pii pj, pii pk) { //叉积
    pii newpoint; //(pk-pi)x(pj-pi)为负，pk在pj的逆时针方向上
    int x1 = (pk.first - pi.first);
    int y1 = (pk.second - pi.second);
    int x2 = (pj.first - pi.first);
    int y2 = (pj.second - pi.second);
    return x1 * y2 - x2 * y1;
}
bool parrel(pii p1, pii p2, pii p3, pii p4)
{ // 线段p1p2和p3p4是否平行
    ll x1 = (p2.first - p1.first);
    ll y1 = (p2.second - p1.second);
    ll x2 = (p4.first - p3.first);
    ll y2 = (p4.second - p3.second);
    if (x1 * y2 - x2 * y1 == 0)
```

```
        return true;
    return false;
}
bool on_segment(pii pi, pii pj, pii pk) {
    int xi = pi.first, yi = pi.second;
    int xj = pj.first, yj = pj.second;
    int xk = pk.first, yk = pk.second;
    if ((min(xi, xj) <= xk && xk <= max(xi, xj))
    && (min(yi, yj) <= yk && yk <= max(yi, yj)))
        return true;
    else
        return false;
}
bool segments_intersect(pii p1, pii p2, pii p3, pii p4)
{ // 线段p1p2和p3p4是否相交
    int d1 = direction(p3, p4, p1);
    int d2 = direction(p3, p4, p2);
    int d3 = direction(p1, p2, p3);
    int d4 = direction(p1, p2, p4);
    if (((d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0))
    && ((d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0)))
        return true;
    else if (d1 == 0 && on_segment(p3, p4, p1))
        return true;
    if (d2 == 0 && on_segment(p3, p4, p2))
        return true;
    else if (d3 == 0 && on_segment(p1, p2, p3))
        return true;
    else if (d4 == 0 && on_segment(p1, p2, p4))
        return true;
    else
        return false;
} // 调用该函数！！相交则返回true
```

2、凸包-Andrew

```
#define MAXN 100006
typedef long long ll;
struct point {
    ll x, y;
} p[MAXN], s[MAXN], p0; // s是栈，实际上是数组
ll direction(point pi, point pj, point pk) {
    //(pk-pi)x(pj-pi)为正，pk在pj的逆时针方向上
    ll x1 = (pk.x - pi.x);
    ll y1 = (pk.y - pi.y);
    ll x2 = (pj.x - pi.x);
    ll y2 = (pj.y - pi.y);
    return x1 * y2 - x2 * y1;
}
bool cmp(point &a, point &b) {
    return a.x != b.x ? a.x < b.x : a.y < b.y;
```

```
}
// 横坐标第一关键字，纵坐标第二关键字，则最小和最大元素一定在凸包上
// 升序枚举求下凸壳，降序求上凸壳
void Andrew_scan(int n) {
    sort(p + 1, p + 1 + n, cmp);
    int t = 0, top = 0, i;
    for (i = 1; i <= n; i++) {
        while (top > 1 && direction(s[top - 1], s[top], p[i]) >= 0) // 叉积
>0，右转
            top--;
        s[++top] = p[i];
    }
    t = top;
    for (i = n - 1; i >= 1; i--) {
        while (top > t && direction(s[top - 1], s[top], p[i]) >= 0)
            top--;
        s[++top] = p[i];
    }
    // 此时，栈s中存储的是凸包上所有的点！！
    // 下面，求凸包围成的面积
    ll res = 0;
    for (i = 2; i < top - 1; i++)
        res += direction(s[1], s[i], s[i + 1]);
    if (res % 2 == 0)
        cout << abs(res) / 2 << ".0\n"; // 这个巧妙
    else
        cout << abs(res) / 2 << ".5\n";
}
int main() {
    int t; cin >> t;
    while (t--) {
        memset(p, 0, sizeof(p));
        memset(s, 0, sizeof(s));
        int n, i; cin >> n;
        for (i = 1; i <= n; i++)
            cin >> p[i].x >> p[i].y;
        Andrew_scan(n);
    }
    return 0;
}
/*
样例输入：
2
4
-1000000000 -1000000000
-1000000000 1000000000
1000000000 1000000000
1000000000 -1000000000
5
1 0
0 1
1 1
2 1
1 2
```

```
样例输出:
4000000000000000000.0
2.0
*/
```

3、点到线段最短距离

```
struct Point {
    double x, y;
};
const double eps = 1e-6;
double distance(Point A, Point B) {
    return sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
}
double GetNearest(Point A, Point B, Point C) {//A到PQ最短距离
    double a = distance(A, B);
    double b = distance(A, C);
    double c = distance(B, C);
    if (a * a - (b * b + c * c) > eps)
        return b;
    if (b * b - (a * a + c * c) > eps)
        return a;
    double l = (a + b + c) / 2;
    double s = sqrt(l * (l - a) * (l - b) * (l - c));
    return 2 * s / c;
}
```

十一、字符串

1、KMP

```
#define MAXN 100010
char P[MAXN], T[MAXN]; // P为模式串，T为文本串
int m, n, q;           // m为模式串长度，n为文本串长度
int pi[MAXN];          // pi[q]：前缀Pq的前缀Pk是前缀Pq的后缀（最长的）
// Gary：字符串P的长度为q的前缀，它的前缀等于后缀的最小长度
void compute_prefix_function() // 求解p[q]，与T毫无关系
{
    pi[1] = 0;
    int k = 0;
    for (q = 2; q <= m; q++) {
        while (k > 0 && P[k + 1] != P[q])
            k = pi[k]; // 如果失配，那么就不断向回跳，直到可以继续匹配
        if (P[k + 1] == P[q]) k++;
        pi[q] = k;
    }
}
void KMP_matcher() {
    compute_prefix_function();
```

```
    q = 0;
    for (int i = 1; i <= n; i++) { // 匹配文本串
        while (q > 0 && P[q + 1] != T[i])
            q = pi[q];
        if (P[q + 1] == T[i]) q++;
        if (q == m) {
            cout << i - m + 1 << '\n'; // 从小到大输出P在T中出现的位置，下标1开始
            q = pi[q];
        }
    }
}
/** P和T下标分别从0开始的 **/
void compute_prefix_function0() // 求解p[q]，与T毫无关系
{
    pi[0] = 0;
    int k = 0;
    for (q = 1; q < m; q++) {
        while (k > 0 && P[k] != P[q])
            k = pi[k - 1]; // 如果失配，那么就不断向回跳，直到可以继续匹配
        if (P[k] == P[q])
            k++;
        pi[q] = k;
    }
}
void KMP_matcher0() {
    compute_prefix_function0();
    q = 0;
    for (int i = 0; i < n; i++) { // 匹配文本串

        while (q > 0 && P[q] != T[i])
            q = pi[q - 1];
        if (P[q] == T[i])
            q++;
        if (q == m) {
            cout << i - m + 1 << '\n'; // 从小到大输出P在T中出现的位置，下标1开始
            q = pi[q - 1];
        }
    }
}
```

2、求前缀与后缀相同的位置

```
#define MAXN 1000010
int pi[MAXN], out[MAXN];
char P[MAXN];
int main()
{
    int tt; cin >> tt;
    while (tt--)
    {
        memset(pi, 0, sizeof(pi));
```

```
        cin >> (P + 1);
        int m = strlen(P + 1);
        pi[1] = 0;
        int k = 0;
        for (int q = 2; q <= m; q++) {
            while (k > 0 && P[k + 1] != P[q]) k = pi[k];
            if (P[k + 1] == P[q]) k++;
            pi[q] = k;
        }
        /*正常kmp算前缀函数
        由前缀函数pi定义：前缀中最长的使得真前缀与真后缀相等的长度
        因此最大解为m，次大解为pi[m]，
        把次大解作为新的最大解，则它的次大解为pi[pi[m]]，
        以此类推，递推到pi[pi[pi...pi[m]]]==0时结束，
        0的前一项就是最小的解，第一项是最大的解，逆序输出即可*/
        int fuck = m, sum = 0;
        out[sum++] = fuck;
        while (fuck > 0) {
            out[sum++] = pi[fuck];
            fuck = pi[fuck];
        }
        for (int i = sum - 2; i >= 0; i--)
            cout << out[i] << ' ';
        cout << '\n';
    }
    return 0;
}
/*
样例输入：
3
aaaaa
abcdabcd
hsah
样例输出：
1 2 3 4 5
4 8
1 4
*/
```

3、KMP-无重合

```
#define MAXN 100010
char P[MAXN], T[MAXN]; // P为模式串，T为文本串
int m, n, q;           // m为模式串长度，n为文本串长度
int pi[MAXN];          // pi[q]：前缀Pq的前缀Pk是前缀Pq的后缀（最长的）
// Gary：字符串P的长度为q的前缀，它的前缀等于后缀的最小长度
void compute_prefix_function() // 求解p[q]，与T毫无关系
{
    pi[1] = 0;
    int k = 0;
    for (q = 2; q <= m; q++)
```

```
    {
        while (k > 0 && P[k + 1] != P[q])
            k = pi[k]; // 如果失配，那么就不断向回跳，直到可以继续匹配
        if (P[k + 1] == P[q])
            k++;
        pi[q] = k;
    }
}
void KMP_matcher() {
    compute_prefix_function();
    q = 0;
    int cnt = 0, end = 0;
    for (int i = 1; i <= n; i++) { // 匹配文本串
        while (q > 0 && P[q + 1] != T[i]) q = pi[q];
        if (P[q + 1] == T[i]) q++;
        if (q == m) {
            cnt++;
            if (i - m + 1 > end) {
                cout << i - m + 1 << ' '; // 从小到大输出P在T中出现的位置，下标1
开始
                end = i;
            }
            q = pi[q];
        }
    }
    if (cnt == 0) cout << "-1";
    cout << '\n';//输出所有匹配位置的下标，字符串规定从1开始
}
```

4、有限自动机，输出转移矩阵

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 1000005;
char s[MAXN];        // 字母表：a-j
int delta[MAXN][10]; // 有限状态转换表
int i, j, q, m, pi[MAXN];
void compute_prefix_function() // 求解p[q]，与T毫无关系
{
    pi[1] = 0;
    int k = 0;
    for (q = 2; q <= m; q++)
    {
        while (k > 0 && s[k + 1] != s[q])
            k = pi[k]; // 如果失配，那么就不断向回跳，直到可以继续匹配
        if (s[k + 1] == s[q])
            k++;
        pi[q] = k;
    }
    // for (i = 1; i <= m; i++)
    //     cout << pi[i] << ' ';
```

```
    // cout << '\n';
}
int main()
{ // 输出自动转移方程
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> (s + 1);
    m = strlen(s + 1);
    compute_prefix_function();
    // COMPUTE_TRANSITION_FUNCTION
    for (j = 0; j < 10; j++)
        delta[0][j] = 0;
    delta[0][s[1] - 'a'] = 1; // 初始化自动机第一位为1
    for (q = 1; q <= m; q++)  // 我们遍历到m，但输出至m-1
    {
        for (j = 0; j < 10; j++) // for each a in alphabet
        {
            if (q < m && s[q + 1] == j + 'a')
                delta[q][j] = q + 1;
            else
                delta[q][j] = delta[pi[q]][j];
        }
    }

    for (i = 0; i < m; i++)
    {
        for (j = 0; j < 10; j++)
            cout << delta[i][j] << ' ';
        cout << '\n';
    }
    return 0;
}
```

十二、FFT

```
const int MAXN = 1e6 + 5;
const double Pi = acos(-1.0);
struct Complex
{
    double x, y;
    Complex(double xx = 0, double yy = 0) { x = xx, y = yy; }
    // double len2() const { return x * x + y * y; }
    // Complex bar() const { return Complex(x, -y); }
} a[MAXN], b[MAXN];
Complex operator+(Complex a, Complex b) { return Complex(a.x + b.x, a.y +
b.y); }
Complex operator-(Complex a, Complex b) { return Complex(a.x - b.x, a.y -
b.y); }
// Complex operator*(double a, Complex b) { return Complex(a * b.x, a *
b.y); }
// Complex operator*(Complex a, double b) { return Complex(a.x * b, a.y *
```

```
b); }
Complex operator*(Complex a, Complex b) { return Complex(a.x * b.x - a.y *
b.y, a.x * b.y + a.y * b.x); }
// Complex operator/(Complex a, double b) { return Complex(a.x / b, a.y /
b); }
// Complex operator/(Complex a, Complex b) { return a * b.bar() /
b.len2(); }
int N, M;
int l, r[MAXN];
int limit = 1;
void FFT(Complex *A, int type)
{
    for (int i = 0; i < limit; i++)
        if (i < r[i])
            swap(A[i], A[r[i]]);              // 求出要迭代的序列
    for (int mid = 1; mid < limit; mid <<= 1) // 待合并区间的中点
    {
        Complex Wn(cos(Pi / mid), type * sin(Pi / mid)); // 单位根
        for (int R = mid << 1, j = 0; j < limit; j += R) // R是区间的右端点,
j表示前已经到哪个位置了
        {
            Complex w(1, 0);                            // 幂
            for (int k = 0; k < mid; k++, w = w * Wn) // 枚举左半部分
            {
                Complex x = A[j + k], y = w * A[j + mid + k]; // 蝴蝶效应
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
        }
    }
}
int main()
{
    cin >> N >> M; // 灵活改变！此处N和M为两多项式最高次数
    for (int i = 0; i <= N; i++)
        cin >> a[i].x;
    for (int i = 0; i <= M; i++)
        cin >> b[i].x;
    while (limit <= N + M)
        limit <<= 1, l++;
    // l=lg(N+M), limit=2^l
    // 这里只是多项式相乘如此实现，如果只是单纯的FFT，输入n，limit就是1<<n
    for (int i = 0; i < limit; i++)
        r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
    // 在原序列中 i 与 i/2 的关系是：i可以看做是i/2的二进制上的每一位左移一位得来
    // 那么在反转后的数组中就需要右移一位，同时特殊处理一下复数
    // 1表示从系数变为点值
    //-1表示从点值变为系数
    FFT(a, 1);
    FFT(b, 1);
    for (int i = 0; i <= limit; i++)
        a[i] = a[i] * b[i];
    FFT(a, -1);
    // 求逆DFT：1. a与y互换（板子里y就是a）2. wn^-1替换wn；3. 将计算结果每个元素除以
```

```
n
    for (int i = 0; i <= N + M; i++)
        printf("%d ", (int)(a[i].x / limit + 0.5));
    return 0;
}
```

十三、补充：Floyd算法，所有结点对的最短路径问题

```
typedef long long ll;
const ll INF = 1e18;
ll D[305][305];
void floyd(int n)
{
    for (int k = 1; k <= n; k++)
        for (int j = 1; j <= n; j++)
            for (int i = 1; i <= n; i++)
                D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
}
int main()
{
    int n, m, i, j, u, v;
    ll w;
    // memset(D, 0, sizeof(D));
    cin >> n >> m;
    for (i = 0; i <= n; i++)
        for (j = 0; j <= n; j++)
            if (i != j)
                D[i][j] = INF;
            else
                D[i][j] = 0;
    for (i = 0; i < m; i++) {
        cin >> u >> v >> w;
        D[u][v] = min(D[u][v], w); // 特别注意输入：因为可能有重边！！！
    }
    floyd(n);
    int q, uu, vv;
    cin >> q;
    // q次访问，下面的uu和vv代表询问从点uu到vv的最小距离
    for (i = 0; i < q; i++) {
        cin >> uu >> vv;
        if (D[uu][vv] == INF)
            cout << "-1" << '\n';
        else
            cout << D[uu][vv] << '\n';
    }
    return 0;
}
```