Template For ICPC

Author: ogmc

Email: haoran.mc@outlook.com

目录

1 01-基本算法

1.1 01-lower-upper-bound

```cpp
#include <cstdio>
#define bug printf("<--->\n");
#define NEXTLINE puts("");
int n = 9;
int arr[] = {1, 2, 2, 3, 3, 3, 3, 4, 5};

// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
void lowerBound(int l, int r, int val) {
    while (l < r) {
        int mid = (l + r) >> 1;
        if (arr[mid] >= val) {
            r = mid;
        }
        else
            l = mid + 1;
    }
    printf("%d\n", l);
}

// 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用:
void upperBound(int l, int r, int val) {
    while (l < r) {
        //这里要加一
        int mid = (l + r + 1) >> 1;
        if (arr[mid] <= val)
            l = mid;
        else
            r = mid - 1;
    }
    printf("%d\n", l);
}

int main() {
    int val = 3;
    lowerBound(0, n - 1, val);
    upperBound(0, n - 1, val);
    return 0;
}
```

1.2 01-三次方根

```cpp
#include <cstdio>
const double eps = 1e-8;

bool judge(double mid, double n) {
    if (mid * mid * mid < n)
        return true;
```

```
    else
        return false;
}

// 0.001
double bsearch_3(double n) {
    double l = -10000, r = 10000;
    while (r - l > eps) {
        double mid = (l + r) / 2;
        if (judge(mid, n))
            l = mid;
        else
            r = mid;
    }
    return r;
}

int main() {
    double n;
    scanf("%lf", &n);
    double res = bsearch_3(n);
    printf("%.6f\n", res);
    return 0;
}
```

1.3 01-根据厚度将书分组

```
/*--------------------------------------------------------------------
 *
 * 文件名称: 01-根据厚度将书分组.cpp
 * 创建日期: 2021年05月28日 星期五 22时43分04秒
 * 题 目: <++>
 * 算 法: 二分
 * 描 述: N本书排成一行, 第i本厚度是book[i], 把它们分成连续的M组
 * 使T(厚度最大的一组的厚度)最小, 最小值是多少
 *
 --------------------------------------------------------------------*/

#include <cstdio>
int n = 9, m = 3;
int book[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

//将n本书分成若干组, 每组最大厚度不超过size, 分成的组数是否小于m
bool valid(int size) {
    int thick = 0;
    int group = 1; //初始值要为1
    for (int i = 0; i < n; ++i) {
        if (size - thick >= book[i])
            thick += book[i];
        else {
            group++;
```

```
            thick = book[i];
        }
    }
    return group <= m;
}

int main() {
    int sum_thick = 0;
    for (int i = 0; i < n; ++i)
        sum_thick += book[i];

    int l = 0, r = sum_thick;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (valid(mid))
            r = mid;
        else
            l = mid + 1;
    }
    printf("%d\n", l);
    return 0;
}
```

1.4 01-离散化-区间和

```
/*------------------------------------------------------------------------
 *
 * 文件名称: 02.cpp
 * 创建日期: 2021年06月01日 星期二 13时25分21秒
 * 题 目: AcWing 0802 区间和
 * 算 法: 离散化
 * 描 述: 这个代码质量更高一点
 *
 ------------------------------------------------------------------------*/

#include <cstdio>
#include <vector>
#include <algorithm>
#include <utility>
using namespace std;
// typedef pair<int, int> PII;
using PII = pair<int, int>;
#define pb push_back
#define fi first
#define se second
#define bug printf("<--->\n");
#define NEXTLINE puts("");
const int maxn = 3e5 + 5;
int n, m;
vector<PII> op, query;
vector<int> alls; //存储所有待离散化的值
```

```cpp
int I[maxn], preS[maxn];

int find(int x) { //二分求出x对应的离散化的值
    int l = 0, r = alls.size() - 1;
    while (l < r) {
        int mid = (l + r) >> 1;
        if (alls[mid] >= x)
            r = mid;
        else
            l = mid + 1;
    }
    return r + 1; //映射到1, 2, 3, ..., n
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++i) {
        int x, c;
        scanf("%d %d", &x, &c);
        op.pb({x, c});
        alls.pb(x);
    }
    for (int i = 0; i < m; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        query.pb({l, r});
        alls.pb(l);
        alls.pb(r);
    }

    sort(alls.begin(), alls.end());
    alls.erase(unique(alls.begin(), alls.end()), alls.end());

    for (auto item : op) { //离散化成功
        int idx = find(item.fi);
        I[idx] += item.se;
    }

    for (int i = 1; i <= alls.size(); ++i) //前缀和
        preS[i] = preS[i-1] + I[i];

    /*
     * for (int i = 0; i <= alls.size(); ++i)
     * printf("%d ", preS[i]);
     * NEXTLINE;
     */

    for (auto item : query) {
        int l = find(item.fi),
            r = find(item.se);
        int res = preS[r] - preS[l-1];
        printf("%d\n", res);
    }
```

```
    return 0;
}
```

1.5 01-简单递归-for 循环

```
#include <cstdio>
const int maxn = 30;
#define NEXTLINE puts("");
int n, m;
int path[maxn];

//u表示层数，start表示从第几个数开始搜
void dfs(int u, int start) {
    if (u > m) {
        for (int i = 1; i <= m; ++i)
            printf("%d ", path[i]);
        NEXTLINE;
    }
    else {
        for (int i = start; i <= n; ++i) {
            path[u] = i;
            dfs(u + 1, i + 1);
            // path[u] = 0;
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    dfs(1, 1);
    return 0;
}
```

1.6 01-糖果传递

```
/*----------------------------------------------------------------------
 *
 * 文件名称: 02.cpp
 * 创建日期: 2021年06月02日 星期三 17时14分12秒
 * 题 目: AcWing 0122 糖果传递
 * 算 法: 推公式
 * 描 述: 自己推了一遍，这个题目很好的，建议搞懂
 * a1 --x1--> a2 --x2--> a3 --x3--> a4 ... an
 * ^ |
 * |--------------------xn---------------------
 *
 * 公式: x1 = x1
 * x2 = abs(x1 - (avg - a2))
 * x3 = abs(x1 - (2* avg - (a2 + a3))
 * x4 = abs(x1 - (3* avg - (a2 + a3 + a4))
```

```
 * x5 = abs(x1 - (4* avg - (a2 + a3 + a4 + a5))
 * ......
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e6 + 5;
typedef long long ll;
int n;
ll I[maxn], x[maxn];

int main() {
    scanf("%d", &n);
    ll sum = 0;
    for (int i = 1; i <= n; ++i) {
        scanf("%lld", &I[i]);
        sum += I[i];
    }
    int avg = sum / n;
    x[1] = 0;
    for (int i = 2; i <= n; ++i)
        x[i] = x[i-1] + avg - I[i];

    sort(x + 1, x + n + 1);
    ll x1 = x[n / 2];
    ll res = 0;
    for (int i = 1; i <= n; ++i)
        res += abs(x[i] - x1);
    printf("%lld\n", res);
    return 0;
}
```

1.7 01-递归 +vector

```
/*----------------------------------------------------------------------
 *
 * 文件名称: 01.cpp
 * 创建日期: 2021年05月11日 ---- 15时10分
 * 题 目: AcWing 92 递归实现指数型枚举
 * 算 法: 递归
 * 描 述: 从1~n这n(n < 20)个整数中随机选取任意多个, 输出所有可能的选择方案
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
#include <vector>
using namespace std;
vector<int> chosen;
int n;
```

```

//对于第x个数做出选择
void calc(int x) {
    if (x == n + 1) {
        for (int i = 0; i < chosen.size(); ++i)
            printf("%d ", chosen[i]);
        puts("");
        return ;
    }
    calc(x + 1); //没有选择x的分支
    chosen.push_back(x); //选择x
    calc(x + 1); //选择x的分支
    chosen.pop_back(); //回溯
}

int main() {
    scanf("%d", &n);
    calc(1);
    return 0;
}
```

1.8 02-lhr

```
#include <cstdio>
#include <vector>
#include <utility>
#include <algorithm>
using namespace std;
typedef pair<int, int> PII;
#define bug printf("<--->\n");
const int INF = 0x3f3f3f3f;
vector<PII> segs; // pair在C++中优先以左端点排序
// segment, section, sequence
// const int maxn = 1e5 + 5;

// 将含有交集的区间合并[1, 5] [4, 8] --> [1, 8]
void merge(vector<PII> &segs) {
    // for循环无法将最后一个区间添加到结果容器中
    // 加上这一句，就可以使最后一个区间添加到结果容器中
    segs.push_back({INF, INF});
    vector<PII> res;
    sort(segs.begin(), segs.end());
    int st = segs[0].first,
        ed = segs[0].second;
    for (auto seg : segs) {
        // printf("%d %d\n", seg.first, seg.second);
        if (ed >= seg.first)
            ed = max(ed, seg.second);
        else {
            res.push_back({st, ed});
            st = seg.first,
```

```
            ed = seg.second;
        }
    }
    segs = res;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("in.txt", "r", stdin);
#endif
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        segs.push_back({l, r});
    }
    merge(segs);
    printf("%d\n", (int)segs.size());
    /*
     * for (auto seg : segs)
     * printf("%d %d\n", seg.first, seg.second);
     */
    return 0;
}
```

1.9 02-swap

```
#include <cstdio>
#include <algorithm>
using namespace std;
int cnt;
void Perm(int* arr, int size, int n) {
    if (n == size) {
        for(int i = 0; i < size; ++i)
            printf("%d", arr[i]);
        printf("\n");
        ++cnt;
    }
    else {
        for(int i = n; i < size; ++i) {
            swap(arr[i], arr[n]);
            Perm(arr, size, n+1);
            swap(arr[i], arr[n]);
        }
    }
}

int main() {
    int arr[5] = {1,2,3,4,5};
    Perm(arr, 5, 0);
```

```
    printf("cnt = %d\n", cnt);
    return 0;
}
```

1.10 02-七夕祭

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
typedef long long ll;
int x[maxn], y[maxn];

//下标从1开始
ll solve(int I[], int n) {
    ll avg = 0;
    for (int i = 1; i <= n; ++i)
        avg += I[i];
    avg /= n;

    int x[maxn];
    x[1] = 0;
    for (int i = 2; i <= n; ++i)
        x[i] = x[i-1] + avg - I[i];

    sort(x + 1, x + n + 1);
    ll x1 = x[n / 2]; //中位数
    ll res = 0;
    for (int i = 1; i <= n; ++i)
        res += abs(x[i] - x1);
    return res;
}

int main() {
    int n, m, t;
    scanf("%d %d %d", &n, &m, &t);
    ll sumx = 0, sumy = 0;
    while (t--) {
        int ix, iy;
        scanf("%d %d", &ix, &iy);
        x[ix]++, y[iy]++;
        sumx++, sumy++;
    }
    if (sumx % n && !(sumy % m))
        printf("column %lld\n", solve(y, m));
    else if (!(sumx % n) && sumy % m)
        printf("row %lld\n", solve(x, n));
    else if (!(sumx % n) && !(sumy % m))
        printf("both %lld\n", solve(x, n) + solve(y, m));
    else
        printf("impossible\n");
```

```
    return 0;
}
```

1.11 02-递归 + 哈希

```
/*-----------------------------------------------------------------------
 *
 * 文件名称: 01.cpp
 * 创建日期: 2021年05月11日 ---- 15时10分
 * 题 目: AcWing 92 递归实现指数型枚举
 * 算 法: 递归
 * 描 述: 从1~n这n(n < 20)个整数中随机选取任意多个, 输出所有可能的选择方案
 *
 -----------------------------------------------------------------------*/

#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 20;
int n;
bool st[maxn];

void dfs(int u) {
    if (u > n) {
        for (int i = 1; i <= n; ++i)
            if (st[i])
                printf("%d ", i);
        NEXTLINE;
        return ;
    }
    st[u] = true;
    dfs(u + 1);

    st[u] = false;
    dfs(u + 1);
}

int main() {
    scanf("%d", &n);
    dfs(1);
    return 0;
}
```

1.12 02-递归 + 状压

```
/*-----------------------------------------------------------------------
 *
 * 文件名称: 01.cpp
 * 创建日期: 2021年05月11日 星期二 21时00分48秒
 * 题 目: AcWing 93 递归组合型枚举
 * 算 法: 递归, 二进制状态压缩
```

```
 * 描 述: 使用递归
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
using namespace std;
#define bug printf("<++>\n");
#define NEXTLINE puts("");
int n, m;

void dfs(int u, int cnt, int state) {
    // if (cnt + (n - u) < m || cnt > m) //之所以不需要cnt > m这句剪枝，是因为当cnt ==
        m时会return，所以不可能到达cnt == 4
    if (cnt + n - u < m)
        return ;
    if (cnt == m) {
        for (int i = 0; i < n; ++i)
            if (state >> i & 1)
                printf("%d ", i + 1);
        NEXTLINE;
        return ;
    }

    dfs(u + 1, cnt + 1, state | 1 << u);
    dfs(u + 1, cnt, state);
}

int main() {
    scanf("%d %d", &n, &m);
    dfs(0, 0, 0);
    return 0;
}
```

1.13 03-递归 + 状压

```
/*----------------------------------------------------------------------
 *
 * 文件名称: 01.cpp
 * 创建日期: 2021年05月11日 ---- 15时10分
 * 题 目: AcWing 92 递归实现指数型枚举
 * 算 法: 递归
 * 描 述: 从1~n这n(n < 20)个整数中随机选取任意多个，输出所有可能的选择方案
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
#define NEXTLINE puts("");
int n;

void dfs(int u, int state) {
    if (u == n) {
```

```
        for (int i = 0; i < n; ++i)
            if (state >> i & 1)
                printf("%d ", i + 1);
        NEXTLINE;
        return ;
    }
    dfs(u + 1, state);
    dfs(u + 1, state | 1 << u);
}

int main() {
    scanf("%d", &n);
    dfs(0, 0);
    return 0;
}
```

1.14 03-递归全排列 + 状压

```
#include <cstdio>
#include <vector>
using namespace std;
#define NEXTLINE puts("");
vector<int> path;
int n;

void DFS(int u, int state) {
    if (u == n) {
        for (auto _ : path)
            printf("%d ", _);
        NEXTLINE;
        return ;
    }
    for (int i = 0; i < n; ++i)
        if (!(state >> i & 1)) {
            path.push_back(i + 1);
            DFS(u + 1, state | 1 << i);
            path.pop_back();
        }
}

int main() {
    scanf("%d", &n);
    DFS(0, 0);
    return 0;
}
```

1.15 04-quickSort

```
/*----------------------------------------------------------------
 *
```

```
 * 文件名称: 09—快速排序.cpp
 * 创建日期: 2021年05月30日 星期日 00时53分58秒
 * 题 目: AcWing 0785 快速排序
 * 算 法: 快速排序
 * 描 述: 去看笔记—首元素做基准
 *
 ------------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
#define NEXTLINE puts("");
int n = 10;
int num[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};

void quickSort(int l, int r) {
    if (l >= r) return;

    int i = l - 1, j = r + 1, x = num[l + r >> 1];
    while (i < j) {
        do i ++ ; while (num[i] < x);
        do j -- ; while (num[j] > x);
        if (i < j) swap(num[i], num[j]);
    }
    quickSort(l, j);
    quickSort(j + 1, r);
}

int main() {
    quickSort(0, 9);
    for (int i = 0; i < n; ++i)
        printf("%d ", num[i]);
    NEXTLINE;
    return 0;
}
```

1.16 05-第 k 个数

```
#include <iostream>
using namespace std;
const int maxn = 1000010;
int num[maxn];

int quickSort(int l, int r, int k) {
    if (l >= r) return num[l];

    int i = l - 1, j = r + 1, x = num[l + r >> 1];
    while (i < j) {
        do i ++ ; while (num[i] < x);
        do j -- ; while (num[j] > x);
        if (i < j) swap(num[i], num[j]);
```

```
    }

    if (j - l + 1 >= k)
        return quickSort(l, j, k);
    else
        return quickSort(j + 1, r, k - (j - l + 1));
}

int main() {
    int n, k;
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i ++ )
        scanf("%d", &num[i]);
    printf("%d\n", quickSort(0, n-1, k));
    return 0;
}
```

1.17 06-mergeSort

```
#include <cstdio>
#define NEXTLINE puts("");
int n = 10;
int num[] = {6, 3, 8, 9, 4, 1, 5, 0, 2, 7};
int tmp[10];

void mergeSort(int l, int r) {
    if (l >= r) return;

    int mid = l + r >> 1;
    mergeSort(l, mid);
    mergeSort(mid + 1, r);

    // v v
    // --------------------
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (num[i] <= num[j])
            tmp[k++] = num[i++];
        else
            tmp[k++] = num[j++];

    while (i <= mid)
        tmp[k++] = num[i++];
    while (j <= r)
        tmp[k++] = num[j++];

    for (i = l, j = 0; i <= r; i++, j++)
        num[i] = tmp[j];
}

int main() {
```

```
    mergeSort(0, n-1);
    for (int i = 0; i < n; ++i)
        printf("%d ", num[i]);
    NEXTLINE;
    return 0;
}
```

1.18 07-逆序数

```
#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 1e5 + 5;
int n;
int num[maxn];
int tmp[maxn];
long long inverse;

/*
 * 左半边内部的逆序对数量: merge_sort(L, mid)
 * 左半边内部的逆序对数量: merge_sort(mid + 1, R)
 */
void mergeSort(int l, int r) {
    if (l >= r) return;

    int mid = (l + r) >> 1;
    mergeSort(l, mid);
    mergeSort(mid + 1, r);

    // v v
    // --------------------
    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (num[i] <= num[j])
            tmp[k++] = num[i++];
        else {
            tmp[k++] = num[j++];
            /*
             * ---------i----------------------
             * ------------------------j---------
             * num[i] > num[j]
             * 必有num[i+1], num[i+2], ... num[mid] > num[j]
             * 这些对都是逆序对
             */
            inverse += (long long)(mid - i + 1); //只是在归并排序上添加这句
        }

    while (i <= mid)
        tmp[k++] = num[i++];
    while (j <= r)
        tmp[k++] = num[j++];

```

```
    for (i = l, j = 0; i <= r; i++, j++)
        num[i] = tmp[j];
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &num[i]);
    mergeSort(0, n-1);
    /*
     * for (int i = 0; i < n; ++i)
     * printf("%d ", num[i]);
     * NEXTLINE;
     */
    printf("%lld\n", inverse);
    return 0;
}
```

1.19 一维前缀和

```
/*--------------------------------------------------------------------------
 *
 * 文件名称: 一维前缀和.cpp
 * 创建日期: 2021年05月31日 星期一 01时14分51秒
 * 题 目: AcWing 0795 前缀和
 * 算 法: 前缀和
 * 描 述: <++>
 *
 --------------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int sequ[maxn];
int preS[maxn];

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    // 前缀和的题目从下标1开始读
    for (int i = 1; i <= n; ++i)
        scanf("%d", &sequ[i]);
    for (int i = 1; i <= n; ++i)
        preS[i] = preS[i-1] + sequ[i];
    while (m--) {
        int l, r;
        scanf("%d %d", &l, &r);
        printf("%d\n", preS[r] - preS[l - 1]);
    }
    return 0;
}
```

1.20 一维差分

```cpp
#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 1e5 + 5;
int sequ[maxn];
int diff[maxn];

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &sequ[i]);
        diff[i] = sequ[i] - sequ[i-1];
    }
    while (m--) {
        int l, r, c;
        scanf("%d %d %d", &l, &r, &c);
        diff[l] += c;
        diff[r+1] -= c;
    }
    for (int i = 1; i <= n; ++i)
        diff[i] += diff[i-1];
    for (int i = 1; i <= n; ++i)
        printf("%d ", diff[i]);
    NEXTLINE;
    return 0;
}
```

1.21 二维前缀和

```cpp
/*--------------------------------------------------------------------
 *
 * 文件名称: 二维前缀和.cpp
 * 创建日期: 2021年05月31日 星期一 11时08分42秒
 * 题 目: AcWing 0796 子矩阵的和
 * 算 法: <++>
 * 描 述: <++>
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1005;
typedef long long ll;
int n, m, q;
int g[maxn][maxn];
ll preS[maxn][maxn];

int main() {
```

```
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            scanf("%d", &g[i][j]);
            preS[i][j] = g[i][j] + preS[i-1][j] + preS[i][j-1] - preS[i-1][j-1];
        }
    while (q--) {
        int x1, y1, x2, y2;
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
        int minx = min(x1, x2);
        int miny = min(y1, y2);
        int maxx = max(x1, x2);
        int maxy = max(y1, y2);
        ll res = preS[maxx][maxy] - preS[maxx][miny-1] - preS[minx-1][maxy] +
            preS[minx-1][miny-1];
        printf("%lld\n", res);
    }
    return 0;
}


/*
 * 1 7 2 4
 * 1 4 2 7
 * 2 7 1 4
 * 2 4 1 7
 * tmpx = min(x1, x2);
 * tmpy = min(y1, y2);
 *
 * ------o--------------------
 * ---o----------------------
 * ---------------------------
 * ---------------------------
 * ---------------------------
 * ---------------------------
 */
```

1.22 二维差分

```
#include <cstdio>
#define NEXTLINE puts("");
const int maxn = 1e3 + 5;
int g[maxn][maxn];
int diff[maxn][maxn];
int n, m, q;

void insert(int x1, int y1, int x2, int y2, int c) {
      diff[x1][y1] += c;
      diff[x2+1][y1] -= c;
      diff[x1][y2+1] -= c;
      diff[x2+1][y2+1] += c;
```

```
}

int main() {
    scanf("%d %d %d", &n, &m, &q);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            scanf("%d", &g[i][j]);
            insert(i, j, i, j, g[i][j]);
        }
    while (q--) {
        int x1, y1, x2, y2, c;
        scanf("%d %d %d %d %d", &x1, &y1, &x2, &y2, &c);
        insert(x1, y1, x2, y2, c);
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            diff[i][j] = diff[i][j] + diff[i-1][j] + diff[i][j-1] - diff[i-1][j-1];
            printf("%d ", diff[i][j]);
        }
        NEXTLINE;
    }
    return 0;
}
```

1.23 均分纸牌

```
/*--------------------------------------------------------------------------
 *
 * 文件名称: 01.cpp
 * 创建日期: 2021年06月02日 星期三 10时16分59秒
 * 题 目: AcWing 1536 均分纸牌
 * 算 法: <++>
 * 描 述: 人都给我看傻了, 太项了
 *
 --------------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 100 + 5;
int card[maxn];
int n, sum, res;

int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &card[i]);
        sum += card[i];
    }

    int avg = sum / n;

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

```
        if (card[i] != avg)
            card[i + 1] += card[i] - avg,
                res++;

    printf("%d\n", res);
    return 0;
}
```

2 02-搜索

2.1 01-树的深度优先搜索

```
/*-------------------------------------------------------------------
 *
 * 文件名称: 01-树的深度优先搜索.cpp
 * 创建日期: 2021年04月14日 ---- 19时53分
 * 题 目: 算法竞赛
 * 算 法: 树的深度优先搜索, 链式前向星
 * 描 述: 树的深度优先搜索, 树的dfs序, 树的深度, 树的重心
 *
 -------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
const int inf = 0x3f3f3f3f;
int n, m;
int vert[maxn], edge[maxn], nxet[maxn], head[maxn], tot;
int cnt, dfs[maxn]; //记录DFS序
bool used[maxn];
int d[maxn]; //记录结点深度
int size[maxn];
int res = inf, pos = -1; //res记录重心对应的max_part值, pos记录了重心

//加入有向边(x, y), 权值为z
void add(int x, int y, int z) {
    vert[++tot] = y;
    edge[tot] = z;
    nxet[tot] = head[x];
    head[x] = tot;
}

/*
 * 树的深度优先搜索
 *
 * 求树的dfs序
 * 但由于邻接表的倒序搜索
 * 所以dfs序也是倒序的
 *
```

```
 * 求树的深度
 */
void DFS(int x) {
    dfs[cnt++] = x;
    used[x] = true;
    for (int i = head[x]; i; i = nxet[i]) {
        int y = vert[i]; //因为链式前向星是倒序的，所以此处y = 是x的父结点
        if (used[y]) continue;
        d[y] = d[x] + 1;
        DFS(y);
    }
    dfs[cnt++] = x;
}

/*
 * 0
 * /|\
 * / | \
 * 1 6 [3]
 * /| / \
 * / | / \
 * 7 4 2 5
 * |
 * |
 * 8
 */


//树的重心 ——— 树的重心与边的权值无关
void gravity(int x) {
    used[x] = true;
    size[x] = 1;
    int max_part = 0;
    for (int i = head[x]; i; i = nxet[i]) { //遍历x的子结点
        int y = vert[i];
        if (used[y]) continue;
        gravity(y); //这句很巧妙，没有返回值，而是递归结束条件是子结点的size为1
        size[x] += size[y];
        max_part = max(max_part, size[y]); //比较所有子树的size
    }
    max_part = max(max_part, n-size[x]);
        //删去结点x后父结点处还有一棵树，size为n-size[x]
    if (max_part < res) {
        res = max_part;
        pos = x;
    }
}

int main() {
    freopen("in/in-01.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
```

```
92          scanf("%d %d %d", &x, &y, &z);
93          add(x, y, z);
94      }
95      DFS(0);
96      //DFS深度优先搜索
97      for (int x = 0; x < n; ++x)
98          for (int i = head[x]; i; i = nxet[i]) {
99              int y = vert[i];
100             // int z = edge[i];
101             printf("%d %d\n", x, y);
102         }
103
104     //dfs序
105     printf("dfs序: ");
106     for (int i = 0; i < 2*n; ++i)
107         printf("%d ", dfs[i]);
108     printf("\n");
109
110     //结点深度
111     for (int i = 0; i < n; ++i)
112         printf("depth[%d] = %d\n", i, d[i]);
113
114     memset(used, 0, sizeof(used));
115     gravity(0);
116     printf("重心以及重心对应的max_part值: %d %d\n", pos, res);
117     return 0;
118 }
```

2.2 02-图的深度优先搜索

```
1  /*-----------------------------------------------------------------------
2   *
3   * 文件名称: 02—图的深度优先搜索.cpp
4   * 创建日期: 2021年04月14日 ———— 19时30分
5   * 题 目: 算法竞赛
6   * 算 法: 图论, 链式前向星
7   * 描 述: 图的连通块划分
8   *
9   -----------------------------------------------------------------------*/
10
11 #include <cstdio>
12 const int maxn = 1e5 + 5;
13 int n, m;
14 int used[maxn];
15 int head[maxn], edge[maxn], nxet[maxn], vert[maxn];
16 int cnt, tot;
17
18 //加入有向边(x, y), 权值为z
19 void add(int x, int y, int z) {
20     vert[++tot] = y;
21     edge[tot] = z;
```

```
    nxet[tot] = head[x];
    head[x] = tot;
}

void DFS(int x) {
    used[x] = true;
    for (int i = head[x]; i; i = nxet[i]) {
        int y = vert[i];
        if (used[y]) continue;
        // printf("%d\n", y);
        DFS(y);
    }
}

int main() {
    freopen("in/in-02.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    for (int i = 0; i < n; ++i)
        if (!used[i]) {
            ++cnt;
            DFS(i);
        }
    printf("%d\n", cnt);
    return 0;
}
```

2.3 02-木棍 dfs 剪枝

```
// POJ1011Sticks
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 105;
int sticks[maxn];
int vis[maxn]; /*记录每根木棍的访问*/
int n; /*多组输入*/
int max_len; /*假设最长木棍长度*/
int max_cnt; /*最大木棍数量*/

/*
 * 正在拼接第stick根原始木棍(已经拼好了stick-1根)
 * 第stick根木棍的当前长度为now_len
 * 拼接到第stick根木棍中的上一根小木棍为last
 */
```

```cpp
bool DFS(int stick, int now_len, int last) {
    /*所有原始木棍都已拼好，搜索成功*/
    if (stick > max_cnt)
        return true;
    /*第stick根木棍已经拼好，去拼下一根*/
    if (now_len == max_len)
        return DFS(stick+1, 0, 1);

    /*记录尝试向当前原始木棍拼接的最近的失败的木棍长度*/
    int record = 0;
    for (int i = last; i <= n; ++i)
        if (!vis[i] && now_len + sticks[i] <= max_len && record != sticks[i]) {
            vis[i] = 1;
            if (DFS(stick, now_len + sticks[i], i + 1))
                return true;
            record = sticks[i];
            vis[i] = 0; /*还原现场*/
            /*贪心，再用1根木棍恰好拼完当前原始木棍必然比再用若干根木棍拼完更好*/
            if (now_len == 0 || now_len + sticks[i] == max_len)
                return false;
        }
    /*所有分支均尝试过，搜索失败*/
    return false;
}

int main() {
    while (scanf("%d", &n) && n) {
        int sum = 0;
        int val = 0;
        for (int i = 1; i <= n; ++i) {
            scanf("%d", &sticks[i]);
            sum += sticks[i];
            val = max(sticks[i], val);
        }
        sort(sticks+1, sticks+1+n);
        reverse(sticks+1, sticks+1+n);
        for (int max_len = val; max_len <= sum; ++max_len) {
            if (sum % max_len)
                continue;
            max_cnt = sum / max_len;
            memset(vis, 0, sizeof(vis));
            if (DFS(1, 0, 1))
                break;
        }
        printf("%d\n", max_len);
    }
    return 0;
}
```

2.4 03-树的广度优先搜索

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>
using namespace std;
const int maxn = 1e4; //顶点比1e3少的话，就可以使用邻接矩阵
int n, m; //顶点的个数
int vert[maxn], edge[maxn], nxet[maxn], head[maxn], tot;
int d[maxn]; //对于一张图来讲，d[x]被称为点x的层次(从起点 1 走到点x需要经过的最少点数)

//加入有向边(x, y)，权值为z
void add(int x, int y, int z) {
    vert[++tot] = y;
    edge[tot] = z;
    nxet[tot] = head[x];
    head[x] = tot;
}

void BFS() {
    memset(d, -1, sizeof(d));
    queue<int> quu;
    quu.push(0);
    d[0] = 0;
    while (!quu.empty()) {
        int x = quu.front();
        printf("%d\n", x);
        quu.pop();
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            if (d[y] != -1) continue;
            d[y] = d[x] + 1;
            quu.push(y);
        }
    }
}

int main() {
    freopen("in-03.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    BFS();
    return 0;
}
```

2.5 04-图的广度优先搜索

2.6 BFS+Cantor

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>
using namespace std;
const int maxn = 362880; //状态共9! = 362880种
struct node {
    int state[9];
    int dis;
};
int dir[4][2] = {{-1, 0}, {0, -1}, {1, 0}, {0, 1}};
int used[maxn];
int start[9];
int goal[9];
int fact[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};

bool cantor(int str[], int n) {
    int res = 0;
    for (int i = 0; i < n; ++i) {
        int counted = 0;
        for (int j = i+1; j < n; ++j)
            if (str[i] > str[j])
                ++counted;
        res += counted * fact[n-1-1];
    }

    if (!used[res]) {
        used[res] = 1;
        return 1;
    }
    return 0;
}

int BFS() {
    node head;
    memcpy(head.state, start, sizeof(head.state));
    head.dis = 0;
    queue<node> quu;
    cantor(head.state, 9);
    quu.push(head);
    while (!quu.empty()) {
        head = quu.front();
        if (memcmp(head.state, goal, sizeof(goal)) == 0)
            return head.dis;
        quu.pop();
        int coord;
        for (coord = 0; coord < 9; ++coord)
            if (head.state[coord] == 0)
                break;
        int x = coord % 3;
        int y = coord / 3;
```

```
52          for (int i = 0; i < 4; ++i) {
53              int newx = x + dir[i][0];
54              int newy = y + dir[i][1];
55              int ncoord = newx + 3*newy;
56              if (newx >= 0 && newx < 3 && newy >= 0 && newy < 3) {
57                  node newnode;
58                  memcpy(&newnode, & head, sizeof(node));
59                  swap(newnode.state[coord], newnode.state[ncoord]);
60                  ++newnode.dis;
61                  if (cantor(newnode.state, 9))
62                      quu.push(newnode);
63              }
64          }
65      }
66      return -1;
67  }
68
69  int main() {
70      for (int i = 0; i < 9; ++i)
71          scanf("%d", &start[i]);
72      for (int i = 0; i < 9; ++i)
73          scanf("%d", &goal[i]);
74      int num = BFS();
75      num != -1 ? printf("%d\n", num) : printf("Impossible");
76      return 0;
77  }
```

2.7 子集与二进制关系

```
1   #include <cstdio>
2   void print_subset(int n) {
3       // 0~2^n, 每个i的二进制数代表一个子集, 比如3的二进制为101, 子集是{0, 2}
4       for (int i = 0; i < (1 << n); ++i) {
5           for (int j = 0; j < n; ++j)
6               if (i & (1 << j))
7                   printf("%d ", j);
8           printf("\n");
9       }
10  }
11
12  int main() {
13      int n = 3;
14      // scanf("%d", &n);
15      print_subset(n);
16      return 0;
17  }
```

2.8 组合与二进制关系

```
1   #include <cstdio>
```

```
void print_set(int n, int m) {
    for (int i = 0; i < (1 << n); ++i) {
        int num = 0;
        int kk = i;
        while (kk) {
            kk = kk & (kk-1);
            ++num;
        }
        if (num == m) {
            for (int j = 0; j < n; ++j)
                if (i & (1 << j))
                    printf("%d ", j);
            printf("\n");
        }
    }
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    print_set(n, m);
    return 0;
}
```

3 03-动态规划

3.1 01-Bone-Collector

```
/*------------------------------------------------------------------------
 *
 * 文件名称: 01-Bone-Collector.cpp
 * 创建日期: 2021年03月09日 ---- 15时34分
 * 题 目: hdu2602 Bone Collector
 * 算 法: 01背包
 * 描 述: 骨头收集者带着体积为V的背包去捡骨头, 已知每个骨头的
 * 体积和价值, 求能装进背包的最大价值
 * 第一行是测试数量, 第二行是骨头数量和背包体积,
 * 第三行是每个骨头的价值, 第四行是每个骨头的体积
 *
------------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1005;
struct Bone {
    int val;
    int vol;
} bone[maxn];
int N; //骨头数量
```

```cpp
int V; //背包体积
// dp[i][j]: 前i件物品放在体积为j的背包中最大价值
int dp[maxn][maxn];

int solve() {
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= N; ++i)
        for (int j = 0; j <= V; ++j) {
            if (bone[i].vol > j) //第i个物品太大，装不下
                dp[i][j] = dp[i-1][j];
            else
                dp[i][j] = max(dp[i-1][j], dp[i-1][j-bone[i].vol] + bone[i].val);
        }
    return dp[N][V];
}

int main() {
    int t;
    scanf("%d", &t);
    while (t--) {
        scanf("%d %d", &N, &V);
        //dp题还是从1开始吧，因为需要用到dp[i-1][j]
        for (int i = 1; i <= N; ++i)
            scanf("%d", &bone[i].val);
        for (int i = 1; i <= N; ++i)
            scanf("%d", &bone[i].vol);
        printf("%d\n", solve());
    }
    return 0;
}
```

3.2 01-Corn-Fields

```cpp
/*------------------------------------------------------------------------
 *
 * 文件名称: 01-Corn-Fields.cpp
 * 创建日期: 2021年03月10日 ---- 20时08分
 * 题 目: poj3254 Corn Fields
 * 算 法: 状态压缩DP
 * 描 述: 输入一个矩阵，选择不相邻的方格的方案数
 * | 1 | 2 | 3 |
 * | | 4 | |
 * 可以的方案数有{}、{1}、{2}、{3}、{4}、{1, 3}、{1, 4}、{3, 4}、{1, 3, 4}
 *
 * 单看第一行，使用二进制描述方格，1表示种玉米，0表示不种玉米
 * | 编号 | 1 | 2 | 3 | 4 | 5 |
 * | 方案 | 000 | 001 | 010 | 100 | 101 |
 *
 * 单看第二行，使用二进制描述方格，1表示种玉米，0表示不种玉米
 * | 编号 | 1 | 2 |
 * | 方案 | 000 | 010 |
```

```
 *
 * 如果第二行选编号1，第一行可以选五种不冲突方案
 * 如果第二行选编号2，会与第一行010冲突，其他四种没问题
 *
 * dp[i][j]表示第i行采用第j种编号的方案时前i行可以得到
 * 的可行方案总数
 * 例如dp[2][2] = 4表示第二行使用第二种方案时的方案总数是4
 *
 * 状态转移方程dp[i][k] = dp[i-1][j] (j From 1 To n)
 * 最后一行的dp[m][k]相加就得到了答案
 *
 ----------------------------------------------------------------------*/

//<++>
```

3.3 01-hdu1257

```
/*----------------------------------------------------------------------
 *
 * 文件名称：01-hdu1257.cpp
 * 创建日期：2021年03月09日 ---- 15时56分
 * 题 目：最少拦截系统
 * 算 法：最长递增子序列
 * 描 述：输入导弹总个数以及导弹依次飞来的高度
 *
----------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 10005;
int n;
int high[maxn];
int LIS() {
    int res = 1;
    int dp[maxn];
    dp[1] = 1;
    for (int i = 2; i <= n; ++i) {
        int maxHigh = 0;
        for (int j = 1; j < i; ++j) {
            if (dp[j] > maxHigh && high[j] < high[i])
                maxHigh = dp[j];
            dp[i] = maxHigh + 1;
            if (dp[i] > res)
                res = dp[i];
        }
    }
    return res;
}

int main() {
    while (scanf("%d", &n)) {
        for (int i = 1; i <= n; ++i)
```

```
35          scanf("%d", &high[i]);
36      printf("%d\n", LIS());
37   }
38   return 0;
39 }
```

3.4 01-二维费用的背包问题

```
1  /*--------------------------------------------------------------------------
2   *
3   * 文件名称: 01—二维费用的背包问题.cpp
4   * 创建日期: 2021年04月28日 ———— 10时37分
5   * 题 目: AcWing 0008 二维费用问题
6   * 算 法: 二维费用背包
7   * 描 述: <++>
8   *
9   --------------------------------------------------------------------------*/
10
11 #include <cstdio>
12 #define _max(a, b) (a > b ? a : b)
13 #define _min(a, b) (a < b ? a : b)
14 const int maxn = 1e3 + 5;
15 int N, V, M;//物品数量，背包体积，背包最大承重
16 int vo[maxn], we[maxn], va[maxn]; //第i件物品体积、重量和价值
17 int dp[maxn][maxn];
18
19 int main() {
20 #ifndef ONLINE_JUDGE
21    freopen("in.txt", "r", stdin);
22    // freopen("out.txt", "w", stdout);
23 #endif
24    scanf("%d %d %d", &N, &V, &M);
25    for (int i = 1; i <= N; ++i)
26       scanf("%d %d %d", &vo[i], &we[i], &va[i]);
27    for (int i = 1; i <= N; ++i)
28       for (int j = V; j >= vo[i]; --j)
29          for (int k = M; k >= we[i]; --k)
30             dp[j][k] = _max(dp[j][k], dp[j-vo[i]][k-we[i]] +
                   va[i]);//动态转移方程，01 背包的思路
31    printf("%d\n", dp[V][M]);
32    return 0;
33 }
```

3.5 01-多重背包

```
1  /*--------------------------------------------------------------------------
2   *
3   * 文件名称: 01—多重背包.cpp
4   * 创建日期: 2021年04月10日 ———— 10时39分
5   * 题 目: AcWing 0004 多重背包
```

```
 * 算 法: 多重背包
 * 描 述: 每种物品选ai次转化为有ai个物品选1次, 即01背包
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 105;
int N, V; //骨头数量, 背包体积
int dp[maxn];

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 0; i < N; ++i) {
        int vo, va, a;
        scanf("%d %d %d", &vo, &va, &a);
        for (int j = V; j > 0; --j)
            for (int k = 1; k <= a && k*vo <= j; ++k)
                dp[j] = max(dp[j], dp[j-k*vo] + k*va);
    }
    printf("%d\n", dp[V]);
    return 0;
}
```

3.6 01-完全背包问题

```
/*----------------------------------------------------------------------
 *
 * 文件名称: 01.cpp
 * 创建日期: 2021年03月30日 ---- 15时30分
 * 题 目: AcWing 0003 完全背包问题
 * 算 法: 动态规划
 * 描 述: 与01背包问题代码相差无几
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 1005;
struct Bone {
    int val;
    int vol;
} bone[maxn];
int N, V;
int dp[maxn][maxn];

int solve() {
    //这里必须是从1开始, 因为从0开始会dp[i-1][j]出界
```

```
    for (int i = 1; i <= N; ++i)
        for (int j = 0; j <= V; ++j) {
            if (bone[i].vol > j)
                dp[i][j] = dp[i-1][j];
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-bone[i].vol] + bone[i].val);
        }
    return dp[N][V];
}

int main() {
    scanf("%d %d", &N, &V);
    for (int i = 1; i <= N; ++i)
        scanf("%d %d", &bone[i].vol, &bone[i].val);
    printf("%d\n", solve());
    return 0;
}
```

3.7 1-拦截子弹 ++

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 25;
int bomb[maxn];
int dp[maxn];
int link[maxn];
int main() {
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d", &bomb[i]);
    ++n;
    bomb[n-1] = -1;
    memset(link, -1, sizeof(link));
    dp[n-1] = 1;
    int maxi;
    int idx;
    int start = - 1;
    int res = 1;
    for (int i = n-2; i >= 0; --i) {
        maxi = dp[i];
        idx = i;
        for (int j = i+1; j < n; ++j)
            if (bomb[j] <= bomb[i] && dp[j] > maxi) {
                maxi = dp[j];
                idx = j;
            }
```

```
32          dp[i] = ++maxi;
33          link[i] = idx;
34          if (dp[i] > res) {
35              res = dp[i];
36              start = i;
37          }
38      }
39      printf("%d\n", res);
40      for (int i = start; i != -1; i = link[i])
41          printf("%d ", bomb[i]);
42      return 0;
43  }
```

3.8 01-最少硬币问题

```
1   /*------------------------------------------------------------------------
2    *
3    * 文件名称: 最少硬币问题.cpp
4    * 创建日期: 2021年03月08日 ---- 14时16分
5    * 题 目: 硬币问题
6    * 算 法: 动态规划
7    * 描 述: 有5种硬币, 面值分别为1, 5, 10, 25, 50, 数量无限, 输入非负整数
8    * 表示需要付的钱数, 要求付钱时选择最少的硬币数
9    *
10  ------------------------------------------------------------------------*/
11
12  #include <cstdio>
13  #include <algorithm>
14  using namespace std;
15  const int maxn = 251; //购物需要的钱数不超过250
16  const int inf = 0x3f3f3f3f;
17  int type[5] = {1, 5, 10, 25, 50}; //5种硬币面值
18  int minCoins[maxn]; //minCoins[i]代表: 付i块钱最少需要多少硬币
19  #define bug printf("<---->\n");
20
21  void solve() {
22      for (int i = 0; i < maxn; ++i) //动态转移方程需要此初始化
23          minCoins[i] = inf;
24      minCoins[0] = 0; //初始条件
25      for (int i = 0; i < 5; ++i) {
26          for (int j = type[i]; j < maxn; ++j)
27              minCoins[j] = min(minCoins[j], minCoins[j-type[i]]+1);
28      }
29  }
30
31  int main() {
32      solve();
33      for (int i = 0; i < maxn; ++i)
34          printf("minCoins[%d] = %d\n", i, minCoins[i]);
35      return 0;
36  }
```

3.9 01-最长公共子序列

```cpp
/*------------------------------------------------------------------------
 *
 * 文件名称: 01—最长公共子序列.cpp
 * 创建日期: 2021年03月09日 ———— 15时48分
 * 题 目: hdu1159 Common Subsequence
 * 算 法: 动态规划
 * 描 述: 求两个序列的最长公共子序列, 注意子序列不是子串
 *
 ------------------------------------------------------------------------*/

#include <iostream>
#include <string>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1005;
int dp[maxn][maxn];
string str1, str2;
int LCS() {
   memset(dp, 0, sizeof(dp));
   for (int i = 1; i <= str1.length(); ++i)
      for (int j = 1; j <= str2.length(); ++j) {
         if (str1[i-1] == str2[j-1])
            str1[i-1] = dp[i-1][j-1] + 1;
         else
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
      }
   return dp[str1.length()][str2.length()];
}

int main() {
   while (cin >> str1 >> str2)
      cout << LCS() << endl;
   return 0;
}
```

3.10 1-木棍加工 ++

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 5e4+5;
struct node {
   int x;
   int y;
} stick[maxn];
int dp[maxn];

bool cmp(node a,node b) {
```

```
    return (a.x > b.x || (a.x == b.x && a.y > b.y)) ? true : false;
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d %d", &stick[i].x, &stick[i].y);
    sort(stick, stick+n, cmp);
    fill(dp, dp+n, 1);
    int res = 1;
    for (int i = 0; i < n; ++i)
      for (int j = i-1; j >= 0; --j)
        if (stick[i].y > stick[j].y)
            dp[i] = max(dp[i], dp[j]+1),
            res = max(res, dp[i]);
    printf("%d\n", res);
    return 0;
}
```

3.11 01-混合背包

```
/*------------------------------------------------------------------------
 *
 * 文件名称: 01-混合背包.cpp
 * 创建日期: 2021年04月07日 ---- 17时56分
 * 题 目: AcWing 0007 混合背包问题
 * 算 法: 混合背包
 * 描 述: 转化为多重背包二进制优化
 *
 ------------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int N, V;//物品数量, 背包体积
int v[maxn];//体积
int w[maxn];//价值
int dp[maxn];

int main() {
    scanf("%d %d", &N, &V);
    int idx = 1; //拆分后物品总数
    for (int i = 1; i <= N; ++i) {
        int vo, va, a;
        scanf("%d %d %d", &vo, &va, &a);
        int b = 1; //二进制分组优化, 1, 2, 4, 8, 16, ...
        //将混合背包转化为多重背包
        if (a < 0)
```

```
30          a = 1;
31      else if (a == 0) //如果是完全背包，则在最优情况下，只能取总体积/该物品体积向下取整
32          a = V / vo;
33      //如果是a - b > 0，那么下面就不需要if语句判断，直接执行if语句里面的语句
34      while (a - b >= 0) {
35          a -= b;
36          v[idx] = vo * b;
37          w[idx] = va * b;
38          ++idx;
39          b *= 2;
40      }
41      if (a > 0) {
42          v[idx] = a * vo;
43          w[idx] = a * va;
44          ++idx;
45      }
46   }
47   for (int i = 1; i <= idx; ++i)
48      for (int j = V; j >= v[i]; --j)
49          dp[j] = max(dp[j], dp[j-v[i]] + w[i]);
50   printf("%d\n", dp[V]);
51   return 0;
52 }
```

3.12 01-石子合并

```
1  /*--------------------------------------------------------------------
2   *
3   * 文件名称：01-石子合并.cpp
4   * 创建日期：2021年03月09日 ---- 17时08分
5   * 题 目：石子合并
6   * 算 法：区间DP
7   * 描 述：每次只能合并相邻的两堆石子，合并的花费为这两堆石子的总数
8   * 每组数据第一行是n个整数，接下来有n堆石子
9   * 状态转移方程：dp[i][j] = min(dp[i][k] + dp[k+1][j]) + sum[i][j-i+1];
10  *
11  --------------------------------------------------------------------*/
12
13 #include <cstdio>
14 #include <algorithm>
15 using namespace std;
16 const int maxn = 255;
17 const int inf = 0x3f3f3f3f;
18 int n;
19 int sum[maxn]; //前i堆的和
20
21 int minval() {
22    int dp[maxn][maxn]; //从第i堆石子到第j堆石子的最小花费
23    for (int i = 1; i <= n; ++i)
24       dp[i][i] = 0;
25    for (int len = 1; len <= n; ++len)
```

```
        //len是i与j之间的距离，也就是说区间DP是从小区间到大区间DP
26      for (int i = 1; i <= n-len; ++i) { //从第i堆开始
27          int j = i + len; //从第j堆结束
28          dp[i][j] = inf;
29          for (int k = i; k < j; ++k) //i与j之间用k进行分割
30              dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + (sum[j]-sum[i-1]));
31      }
32  return dp[1][n];
33  }
34
35  int main() {
36      while (scanf("%d", &n)) {
37          sum[0] = 0;
38          for (int i = 1; i <= n; ++i) {
39              int x;
40              scanf("%d", &x);
41              sum[i] = sum[i-1] + x;
42          }
43          printf("%d\n", minval());
44      }
45      return 0;
46  }
```

3.13 02-一维

```
1   /*-------------------------------------------------------------------
2    *
3    * 文件名称: 02-一维.cpp
4    * 创建日期: 2021年03月30日 ---- 16时14分
5    * 题 目: AcWing 0003 01背包
6    * 算 法: 动态规划
7    * 描 述: 使用一维数组
8    *
9    -------------------------------------------------------------------*/
10
11  #include <cstdio>
12  #include <cstring>
13  #include <algorithm>
14  using namespace std;
15  const int maxn = 1005;
16  struct Bone {
17      int val;
18      int vol;
19  } bone[maxn];
20  int N; //骨头数量
21  int V; //背包体积
22  int dp[maxn];
23
24  int solve() {
25      memset(dp, 0, sizeof(dp));
26      for (int i = 1; i <= N; ++i)
```

```
27        for (int j = V; j >= bone[i].vol; --j)
28            /*
29             * dp[j-bone[i].vol]此处体积一定是上个物品的最优解
30             * 因为还没有dp[j-bone[i].vol] = max(dp[j-bone[i].vol],
                   dp[j-2*bone[i].vol]+2*bone[i].val)
31             * 如果j从小到大循环的话，dp[j-bone[i].vol]中是从dp[j-2*bone[i].vol]转移来的
32             *
                   表示在背包体积为j时，可以放入的当前物品数量不定，可以无穷件，只要不超过背包容积
33             */
34            dp[j] = max(dp[j], dp[j-bone[i].vol] + bone[i].val);
35    return dp[V];
36 }
37
38 int main() {
39    int t;
40    scanf("%d", &t);
41    while (t--) {
42        scanf("%d %d", &N, &V);
43        //dp题还是从1开始吧，因为需要用到dp[i-1][j]
44        for (int i = 1; i <= N; ++i)
45            scanf("%d", &bone[i].val);
46        for (int i = 1; i <= N; ++i)
47            scanf("%d", &bone[i].vol);
48        printf("%d\n", solve());
49    }
50    return 0;
51 }
```

3.14 02-一维 1

```
1  /*-----------------------------------------------------------------------
2   *
3   * 文件名称: 02-一维.cpp
4   * 创建日期: 2021年03月30日 ---- 16时20分
5   * 题 目: AcWing 0003 完全背包
6   * 算 法: 动态规划
7   * 描 述: 与01背包问题代码相差无几
8   *
9   -----------------------------------------------------------------------*/
10
11 #include <cstdio>
12 #include <algorithm>
13 using namespace std;
14 const int maxn = 1005;
15 struct Bone {
16    int val;
17    int vol;
18 } bone[maxn];
19 int N, V;
20 int dp[maxn];
21
```

```
int solve() {
   for (int i = 1; i <= N; ++i)
      for (int j = 0; j <= V-bone[i].vol; ++j)
         /*
          * dp[j] = max(dp[j], dp[j-bone[i].vol] + bone[i].val);
          * 为什么不使用上面一句呢?
          * 因为不是从0开始的, dp[j-bone[i].vol]的开始位置不能确定
          */
         dp[j+bone[i].vol] = max(dp[j+bone[i].vol], dp[j]+bone[i].val);
   return dp[V];
}

int main() {
   scanf("%d %d", &N, &V);
   for (int i = 1; i <= N; ++i)
      scanf("%d %d", &bone[i].vol, &bone[i].val);
   printf("%d\n", solve());
   return 0;
}
```

3.15 02-不要 4-记忆化

```
/*------------------------------------------------------------------
 *
 * 文件名称: 02-不要4-记忆化.cpp
 * 创建日期: 2021年03月10日 ---- 08时12分
 * 题 目: hdu2089 不要62
 * 算 法: 数位DP
 * 描 述: 一个数字如果包含‘4’或者'62', 它是不吉利的, 给定m和n
 * 0 < m < n < 1e6, 统计[m, n]范围内的吉利数
 * 只是为了理解数位DP, 所以简化题目要求, 只排除了'4'
 * 记忆化搜索的思路就是在递归dfs()中搜索所有可能的情况, 遇到已
 * 经算过的记录在dp[]中的结果就直接使用, 不再重复计算
 * dp[i]表示i位数中符合条件的数字个数
 *
 ------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 12;
int dp[maxn]; //dp[i]表示i位数符合要求的个数, dp[2]表示00 ~ 99内符合要求的个数
int digit[maxn];
int dfs(int len, int ismax) { //如果ismax == 1,
   int res = 0;
   int maxx;
   if (!len) //已经递归到0位数, 返回
      return 1;
   if (!ismax && dp[len] != -1) //记忆化搜索, 如果已经计算过, 就直接使用
      return dp[len];
   maxx = (ismax ? digit[len] : 9); //用来判断搜索到什么位置, 比如324在十位只用搜索到2
   for (int i = 0; i <= maxx; ++i) {
```

```
        if (i == 4) //排除4
            continue;
        res += dfs(len-1, ismax && i == maxx);
    }
    if (!ismax)
        dp[len] = res;
    return res;
}

int main() {
    int n;
    int len = 0;
    memset(dp, -1, sizeof(dp));
    scanf("%d", &n);
    while(n) {
        digit[++len] = n % 10;
        n /= 10;
    }
    printf("%d\n", dfs(len, 1));
    return 0;
}
```

3.16 02-二进制分组优化

```
/*--------------------------------------------------------------------------
 *
 * 文件名称: 01-二进制分组优化.cpp
 * 创建日期: 2021年03月30日 ---- 20时05分
 * 题 目: hdu2191 悼念512汶川大地震
 * 算 法: 多重背包
 * 描 述: 输入数据首先包含一个正整数t, 表示有t组测试用例
 * 每组测试用例第一行是两个整数V和N(1 <= V <= 100, 1 <= N <= 100)
 * 分别表示背包的容量和大米的种类, 然后是m行数据
 * 每行包含3个数vo, va和a(1 <= vo <= 20, 1 <= va <= 200, 1 <= a <= 20)
 * 分别表示大米的体积、每袋的价值以及对应种类大米的袋数
 * 输出能够购买的大米的最大价值(题意有更改)
 *
 --------------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
//虽然题目的N <= 100, 但二进制分组后会变多, 所以需要更多存储空间
const int maxn = 1005;
struct Bone {
    int vol;
    int val;
} bone[maxn];
int N, V;
int dp[maxn];
```

```
28
29 int main() {
30    int t;
31    scanf("%d", &t);
32    while (t--) {
33       scanf("%d %d", &V, &N);
34       int idx = 1; //拆分后物品总数
35       for (int i = 1; i <= N; ++i) {
36          int vo, va, a; //体积，价值，大米袋数
37          scanf("%d %d %d", &vo, &va, &a);
38          int b = 1; //二进制分组优化，1, 2, 4, 8, 16, ...
39          while (a - b > 0) {
40             a -= b;
41             bone[idx].vol = b * vo;
42             bone[idx].val = b * va;
43             ++idx;
44             b *= 2;
45          }
46          bone[idx].vol = a * vo; //补充不足指数的差值
47          bone[idx].val = a * va;
48          ++idx;
49       }
50       memset(dp, 0, sizeof(dp));
51       for (int i = 1; i < idx; ++i) //对拆分后的物品进行0-1背包
52          for (int j = V; j >= bone[i].vol; --j)
53             dp[j] = max(dp[j], dp[j-bone[i].vol] + bone[i].val);
54       printf("%d\n", dp[V]);
55    }
56    return 0;
57 }
```

3.17 02-回文串

```
1  /*--------------------------------------------------------------------
2   *
3   * 文件名称：02-回文串.cpp
4   * 创建日期：2021年03月09日 ---- 17时37分
5   * 题 目：poj3280 Cheapest Palindrome
6   * 算 法：区间DP
7   * 描 述：给定字符串s，长度为m，由n个小写字母构成，在s的任意位置
8   * 增删字母，把它变为回文串，增删特定字母的花费不同，求最小花费
9   * 3 4
10  * abcb
11  * b 350 700
12  * c 200 800
13  * a 1000 1100
14  *
15  * 900
16  *
17  --------------------------------------------------------------------*/
18
```

```cpp
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
const int maxn = 2005;
int weight[30];
int dp[maxn][maxn];

int main() {
    int n, m;
    while (cin >> n >> m) {
        string str;
        cin >> str;
        for (int i = 0; i < n; ++i) {
            int x, y;
            char ch;
            cin >> ch >> x >> y; //读取每个字符的插入和删除花费
            weight[ch-'a'] = min(x, y); //取其中的最小值
        }
        for (int i = m-1; i >= 0; --i) //i是子区间的起点
            for (int j = i+1; j < m; ++j) { //j是子区间的终点
                if (str[i] == str[j])
                    dp[i][j] = dp[i+1][j-1];
                else
                    dp[i][j] = min(dp[i+1][j] + weight[str[i]-'a'], dp[i][j-1] +
                        weight[str[j]-'a']);
            }
        cout << dp[0][m-1] << endl;
    }
    return 0;
}
```

3.18 2-拦截导弹

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
using namespace std;
const int maxn = 25;
const int inf = 0x3f3f3f3f;
int bomb[maxn];
int dp[maxn]; /*当前导弹作为结尾时能拦截的导弹数*/
int link[maxn];
int main() {
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
    int n;
    scanf("%d", &n);
    ++n;
    bomb[0] = inf;
```

```
        /*必须设置第一个是最大的，保证之后的最大的能被link在这次之后，否则之后的最大的link是它自己，而最后的输出
    for (int i = 1; i <= n; ++i)
        scanf("%d", &bomb[i]);
    memset(link, -1, sizeof(link));
    int maxi;
    int idx;
    int res = 1;
    int start = -1;
    dp[0] = 1; /*第一个导弹一定能被拦截*/
    /*dp[1]不需要预先设定，如果程序中没有找到前面比它高的导弹，会自动设定1*/
    /*而dp[0]通过循环自己设定1，因为会同时改变idx的值*/
    for (int i = 1; i < n-1; ++i) {
        maxi = dp[i]; /*首先认为当前的dp是最大的，如果能发现更大的，那就更改为更大的*/
        idx = i; /*idx寻找前面比当前导弹高的，且dp更大的导弹的下标*/
        /*一定要倒着寻找，找到最近的*/
        for (int j = i-1; j >= 0; --j)
            if (bomb[j] >= bomb[i] && dp[j] > maxi) {
                maxi = dp[j];
                idx = j;
            }
        dp[i] = ++maxi;
        link[i] = idx;
        if (dp[i] > res) { /*最多的能拦截的导弹数*/
            res = dp[i];
            start = i;
        }
    }
    printf("%d\n", res);
    vector<int> vec;
    for (int i = start; i != -1; i = link[i])
        if (i != 0)
            vec.push_back(bomb[i]);
    for (auto it = vec.begin(); it != vec.end(); ++it)
        printf("%d ", *it);
    return 0;
}
```

3.19 02-最少硬币组合

```
/*-----------------------------------------------------------------------
 *
 * 文件名称：最少硬币问题.cpp
 * 创建日期：2021年03月08日 ---- 14时16分
 * 题 目：硬币问题
 * 算 法：动态规划
 * 描 述：有5种硬币，面值分别为1, 5, 10, 25, 50，数量无限，输入非负整数
 * 表示需要付的钱数，要求付钱时选择最少的硬币数
 * 而且打印出硬币组合
 *
 -----------------------------------------------------------------------*/

```

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 251; //购物需要的钱数不超过250
const int inf = 0x3f3f3f3f;
int type[5] = {1, 5, 10, 25, 50}; //5种硬币面值
int minCoins[maxn]; //minCoins[i]代表: 付i块钱最少需要多少硬币
int coinPath[maxn]; //记录最少硬币路径
#define bug printf("<---->\n");

void solve() {
    for (int i = 0; i < maxn; ++i) //动态转移方程需要此初始化
        minCoins[i] = inf;
    minCoins[0] = 0; //初始条件
    for (int i = 0; i < 5; ++i) {
        for (int j = type[i]; j < maxn; ++j)
            if (minCoins[j] > minCoins[j-type[i]]+1) {
                coinPath[j] = type[i]; //j块钱时选择的最后一块硬币是type[i]
                minCoins[j] = minCoins[j-type[i]] + 1;
            }
    }
}

void print(int money) {
    while (money) {
        printf("%d ", coinPath[money]);
        money -= coinPath[money];
    }
    printf("\n");
}

int main() {
    solve();
    for (int i = 0; i < maxn; ++i) {
        printf("minCoins[%d] = %d\n", i, minCoins[i]);
        printf("==> ");
        print(i);
    }
    return 0;
}
```

3.20 03-单调队列优化

```cpp
/*-----------------------------------------------------------------------
 *
 * 文件名称: 03-单调队列优化.cpp
 * 创建日期: 2021年04月07日 ---- 17时52分
 * 题 目: AcWing 0006 多重背包问题
 * 算 法: 多重背包, 单调队列优化
 * 描 述: 执行单调队列的三个惯例操作
 * 1. 检查队头合法性
```

```
 * 2. 取队头为最优策略，更新
 * 3. 把新策略插入队尾，入队前检查队尾单调性，排除无用决策
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
const int maxn = 20005;
int N, V;
int dp[maxn], tmp[maxn];
int quu[maxn]; //队首元素是dp[j], dp[j-v], dp[j-2v], ... dp[j-(k-v)v]里最大的数

int main() {
   freopen("in.txt", "r", stdin);
   freopen("out.txt", "w", stdout);
   scanf("%d %d", &N, &V);
   for (int i = 0; i < N; ++i) {
      int vo, va, a; //体积，价值，当前物品数量
      scanf("%d %d %d", &vo, &va, &a);
      memcpy(tmp, dp, sizeof(dp)); //tmp为dp[i-1], dp为dp[i]
      /*
       * 枚举余数即等价类
       * 背包体积，j < vo是因为vo可以由j = 0 + vo得到
       * 再下面一个for循环就是将每个等价类进行单调队列操作
       * 每个等价类有0, 0+1vo, 0+2vo, 0+3vo ...
       * 1, 1+1vo, 1+2vo, 1+3vo ...
       * 2, 2+1vo, 2+2vo, 2+3vo ...
       * ...
       * 所有等价类中的数就是0 ~ V，也就是背包的第二维
       */
      for (int j = 0; j < vo; ++j) {
         int hh = 0, tt = -1; //hh为队首 tt为队尾
         for (int k = j; k <= V; k += vo) { //枚举同一等价类的背包体积
            dp[k] = tmp[k]; //此句意义不大，确实意义不大
            //如果当前位置表示的背包体积>队首体积+当前物品总体积，很显然队首体积不合法了，出队
            if (hh <= tt && quu[hh] < k-a*vo)
               ++hh;
            /*
             *
                使用tmp，决策上一件物品是队首体积时的最优解+队首体积到当前体积需要的当前物品总价值
             *
             * |-------- quu[hh]指向的位置，tmp[quu[hh]存储这里的最优解
             * v
             *
                ---------------------------------------------------------
             * | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
             *
                ---------------------------------------------------------
             * | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
             *
                ---------------------------------------------------------
```

```
58              * ^ ^
59              * |——— 填充到当前体积需要的物品数(k-quu[hh])/vo*va ———|
60              */
61             if (hh <= tt)
62                dp[k] = max(dp[k], tmp[quu[hh]]+(k-quu[hh])/vo*va);
63             /*
64              * tmp[quu[tt]] 放入上一件物品体积在队尾体积处时的最优解
65              * (quu[tt]-j)/vo*va
                    等价类的第一个体积到队尾体积全部填充当前物品所需要的总价值
66              * tmp[k] 放入上一件物品体积在当前体积处时的最优解
67              * (k-j)/vo*va 等价类的第一个体积到当前体积全部填充当前物品所需要的总价值
68              *
69              * 在动态转移方程里是tmp[quu[hh]] + (k-quu[h])/vo*va
70              * 也就是最近的k个物品的决策里的最大值
71              * 但为什么在这里就是tmp[quu[tt]] - (quu[tt]-j)/vo*va呢
72              *
73              * 前a-1个数在单调队列中的决策
74              * dp[k] = max(dp[k], dp[k-vo]+va, dp[k-2vo]+2va, dp[k-3vo]+3va, ...
75              * 在单调队列中无法实现，转化为
76              * dp[k]-(k-j)/vo*va, dp[k-vo]-(k-vo-j)/vo*va,
                    dp[k-2vo]-(k-2vo-j)/vo*va, ...
77              * 会发现，上下两式各项都相差(k-j)/vo*va，所以可以转化
78              *
79              *
                    队列中肯定要放最大值，这里的最大值还要根据k的位置确定，因为在求dp[k]时，队首元素到
80              *
                    当前位置k所需要的(k-quu[hh])/vo*va是随着k的变化而变化的，所以队列中不能存储确定的值，只能存
81              *
                    对于当前位置，队列中的元素到达当前位置的总价值是无法确定的，但是到达等价类的第一个位置的总价值
82              *
                    确定的，而且等价类的第一个位置到当前位置的总价值也是确定的，所以单调队列中存储的本应是前a个位
83              *
                    达当前位置的总价值由大到小排序(部分不满足由大到小的位置出队)，然而到达当前位置的总价值无法确定
84              * 转化为到达等价类第一个位置的总价值
85              */
86             while (hh <= tt && tmp[quu[tt]] - (quu[tt]-j)/vo*va <= tmp[k] -
                   (k-j)/vo*va)
87                --tt;
88             //无论如何，将当前位置插入队列
89             quu[++tt] = k;
90          }
91       }
92    }
93    printf("%d\n", dp[V]);
94    return 0;
95 }
```

3.21 3-导弹拦截

```
1 #include <cstdio>
2 #include <cstring>
```

```cpp
#include <algorithm>
using namespace std;
const int maxn = 25;
int bomb[maxn];
int dp[maxn];
int main() {
   //freopen("in.txt", "r", stdin);
   //freopen("out.txt", "w", stdout);
   int n;
   scanf("%d", &n);
   for (int i = 0; i < n; ++i)
      scanf("%d", &bomb[i]);
   fill(dp, dp+n, 1);
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < i; ++j)
         if (bomb[i] <= bomb[j])
            dp[i] = max(dp[i], dp[j]+1);
            // 表示当前导弹发射后最大能拦截的导弹数
   int res = 0;
   for (int i = 0; i < n; ++i)
      res = max(res, dp[i]);
   printf("%d\n", res);
   return 0;
}
```

3.22 03-所有硬币组合-1

```cpp
/*----------------------------------------------------------------------
 *
 * 文件名称: 最少硬币问题.cpp
 * 创建日期: 2021年03月08日 ---- 14时16分
 * 题 目: 硬币问题
 * 算 法: 动态规划
 * 描 述: 有5种硬币, 面值分别为1, 5, 10, 25, 50, 数量无限, 输入非负整数
 * 表示需要付的钱数, 要求付钱时选择最少的硬币数
 * 而且打印出硬币组合
 * 没有硬币数量限制
 *
 ----------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 251;
int type[5] = {1, 5, 10, 25, 50};
int dp[maxn];
void solve() {
   dp[0] = 1;
   for (int i = 0; i < 5; ++i)
      for (int j = type[i]; j < maxn; ++j)
         dp[j] = dp[j] + dp[j-type[i]];
}

```

```
int main() {
   solve();
   for (int i = 0; i < maxn; ++i)
      printf("dp[%d] = %d\n", i, dp[i]);
   return 0;
}
```

3.23 04-所有硬币组合-2

```
/*-------------------------------------------------------------------------
 *
 * 文件名称: 最少硬币问题.cpp
 * 创建日期: 2021年03月08日 ---- 14时16分
 * 题 目: 硬币问题
 * 算 法: 动态规划
 * 描 述: 有5种硬币, 面值分别为1, 5, 10, 25, 50, 数量无限, 输入非负整数
 * 表示需要付的钱数, 要求付钱时选择最少的硬币数
 * 而且打印出硬币组合
 * 硬币数量限制为100
 *
 -------------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 251;
const int coin = 101; //限制最多选择100个硬币
int type[5] = {1, 5, 10, 25, 50};
int dp[maxn][coin];
int ans[maxn];
void solve() {
   dp[0][0] = 1;
   for (int i = 0; i < 5; ++i)
      for (int j = 1; j < coin; ++j)
         for (int k = type[i]; k < maxn; ++k)
            dp[k][j] += dp[k-type[i]][j-1];
}

int main() {
   solve();
   for (int i = 0; i < maxn; ++i)
      for (int j = 0; j < coin; ++j)
         ans[i] += dp[i][j];
   for (int i = 0; i < maxn; ++i)
      printf("ans[%d] = %d\n", i, ans[i]);
   return 0;
}
```

4 04-字符串

4.1 01-KMP

```cpp
#include <cstdio>
#include <cstring>
#include <cstdlib>

/*前缀和*/
void prefix_table(char pattern[], int prefix[]) {
   int len = strlen(pattern);
   prefix[0] = 0;
   int ptr = 0;

   int i = 1;
   while (i < len) {
      if (pattern[i] == pattern[ptr]) {
         ++ptr;
         prefix[i] = ptr;
         ++i;
      }
      else {
         if (ptr > 0)
            ptr = prefix[ptr-1];
         else {
            prefix[i] = 0;
            ++i;
         }
      }
   }
}

/*对前缀和操作*/
void move_prefix_table(int prefix[], int n) {
   for (int i = n-1; i > 0; --i)
      prefix[i] = prefix[i-1];
   prefix[0] = -1;
}

void kmp(char pattern[], char text[]) {
   int n = strlen(pattern);
   int m = strlen(text);
   int* prefix = (int *)malloc(sizeof(int) * n);

   prefix_table(pattern, prefix);
   move_prefix_table(prefix, n);

   // pattern[j] , len(pattern) = n
   // text[i] , len(text) = m

   int i = 0;
   int j = 0;
   while (i < m) {
      if (j == n-1 && text[i] == pattern[j]) {
         printf("Found pattern at %d\n", i-j);
         j = prefix[j];
```

```
53          }
54          if (text[i] == pattern[j]) {
55              ++i;
56              ++j;
57          }
58          else {
59              j = prefix[j];
60              if (j == -1) {
61                  ++i;
62                  ++j;
63              }
64          }
65      }
66  }
67
68  int main() {
69      char pattern[] = "ABABCABAA";
70      char text[] = "ABABABCABAABABAABAB";
71      kmp(pattern, text);
72      return 0;
73  }
```

4.2 02-Manacher

```
1   #include <cstdio>
2   #include <string>
3   #include <iostream>
4   #include <vector>
5   #include <algorithm>
6   using namespace std;
7   #define bug printf("<------->\n");
8
9   vector<int> radius, let; /*回文子串半径*/
10  string expa_str; /*扩展后的串*/
11  /*原串、最长回文子串开始位置、最长回文子串长度*/
12  void Manacher(const string &str, int &pos, int &max_len) {
13      int orig_len = str.length(); /*原串长度*/
14      int expa_len = (orig_len + 1) << 1; /*扩展串长度*/
15      max_len = 0;
16      radius.resize(expa_len + 1);
17      expa_str.resize(expa_len + 1);
18      //@#0#1#2#3#4#5#6#7#8#9#$
19      expa_str[0] = '@';
20      expa_str[1] = '#';
21      /*expa_len是扩展串的长度减一，不过串从零开始*/
22      expa_str[expa_len] = '$';
23      for (int i = 1; i <= orig_len; ++i) {
24          /*偶位置对应原串字符*/
25          expa_str[i << 1] = str[i-1];
26          /*不更改奇位置，只更改偶位置*/
27          expa_str[i << 1 | 1] = '#';
```

```
    }
    radius[1] = 1; /*显然回文子串半径大于等于1*/
    //本应该计算到enpa_len + 1的，但那个位置是'#'，不需要计算了
    for (int max_R = 0, center = 0, i = 2; i < expa_len; ++i) {
        /*根据i探查到的位置是否超过了最右回文子串的右边界，初始化i的回文半径，核心操作*/
        radius[i] = i < max_R ? min(max_R-i, radius[2*center-i]) : 1;
        /*暴力拓展中心在i位置的最长回文子串半径radius[i]*/
        while (expa_str[i-radius[i]] == expa_str[i+radius[i]])
            ++radius[i];
        /*更新最右回文子串的右边界*/
        if (radius[i] + i > max_R) {
            max_R = radius[i] + i;
            center = i;
        }
        /*更新最长回文子串的位置与长度*/
        if (radius[i] - 1 > max_len) {
            /*原串的最长回文子串*/
            max_len = radius[i] - 1;
            /*原串的最长回文子串的起点*/
            pos = (center - radius[i] + 1) >> 1;
        }
    }
}

//odd为false，字符串为奇回文串
//以x为中心的最长回文子串的长度
//如果为偶回文串，则以x，x+1中心为中心的最长回文子串的长度
int start_mid(int x, bool odd) {
    return odd ? radius[(x+1) << 1] - 1 : radius[(x+1) << 1 | 1] - 1;
}

//知道回文左边界，且在Manacher函数运行结束后使用
int start_left(int x, string str) {
    //let[i]表示以x为起点，的最长回文子串的中心位置
    int expand_len = (str.length() + 1) << 1;
    //let数组在扩展之后的字符串上，以位置i为起点的最长回文子串的中心在哪里
    let.resize(expand_len + 1);
    //计算维护以每个位置为起点的最长回文串
    for (int i = 0; i <= expand_len; i++)
        let[i] = 0;
    for (int i = 2; i < expand_len; i++)
        if (let[i - radius[i] + 1] < i + 1)
            let[i - radius[i] + 1] = i + 1;
    for (int i = 1; i <= expand_len; i++)
        if (let[i] < let[i - 1])
            let[i] = let[i - 1];
    //返回以x为起点的最长回文子串的长度
    return let[(x + 1) << 1] - ((x + 1) << 1);
}

int main() {
    //string str = "0123456789";
    string str = "DDDDBBDBA|BCBAAABBAAABCB|DB";
```

```
    string subStr;
    int pos;
    int max_len;
    Manacher(str, pos, max_len);
    printf("pos = %d\n", pos);
    printf("max_len = %d\n", max_len);
    cout << "subStr = " << str.substr(pos, max_len) << endl;
    printf("length = %d\n", start_left(pos, str));
    printf("length = %d\n", start_mid(pos+max_len/2-1, false));
    return 0;
}
```

4.3 04-最长公共子串

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 200005;
char text[maxn];
int sa[maxn];
int rk[maxn];
int cnt[maxn];
int t1[maxn];
int t2[maxn];
int height[maxn];
int n;

void calc_sa() {
    int m = 127;
    int *x = t1;
    int *y = t2;
    for (int i = 0; i < m; ++i) cnt[i] = 0;
    for (int i = 0; i < n; ++i) cnt[x[i] = text[i]]++;
    for (int i = 1; i < m; ++i) cnt[i] += cnt[i-1];
    for (int i = n-1; i >= 0; --i) sa[--cnt[x[i]]] = i;
    for (int k = 1; k <= n; k *= 2) {
        int p = 0;
        for (int i = n-k; i < n; ++i) y[p++] = i;
        for (int i = 0; i < n; ++i)
            if (sa[i] >= k)
                y[p++] = sa[i] - k;
        for (int i = 0; i < m; ++i) cnt[i] = 0;
        for (int i = 0; i < n; ++i) cnt[x[y[i]]]++;
        for (int i = 1; i < m; ++i) cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; --i) sa[--cnt[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (int i = 1; i < n; ++i)
            x[sa[i]] =
```

```
38              y[sa[i-1]] == y[sa[i]] && y[sa[i-1] + k] == y[sa[i] + k] ? p-1 : p++;
39      if (p >= n) break;
40      m = p;
41   }
42}
43
44/*求辅助数组height[]*/
45/*定义height[]为sa[i-1]和sa[i]也就是排名相邻的两个后缀的最长公共前缀长度*/
46void getheight(int n) {
47   int k = 0;
48   for (int i = 0; i < n; ++i)
49      rk[sa[i]] = i;
50   for (int i = 0; i < n; ++i) {
51      if (k)
52         k--;
53      int j = sa[rk[i]-1];
54      while (text[i+k] == text[j+k])
55         k++;
56      height[rk[i]] = k;
57   }
58}
59
60int main() {
61   int len1;
62   int res;
63   while (scanf("%s", text) != EOF) {
64      n = strlen(text);
65      len1 = n;
66      text[n] = '$'; //用$分割两个字符串
67      scanf("%s", text + n + 1); //将第2个字符串和第1个字符串合并
68      n = strlen(text);
69      calc_sa();
70      getheight(n);
71      res = 0;
72      for (int i = 0; i < n; ++i)
73         //找最大的height[i]，并且它对应的sa[i-1]和sa[i]分别属于前后两个字符串
74         if (height[i] > res &&
75               ((sa[i-1] < len1 && sa[i] >= len1) || (sa[i-1] >= len1 && sa[i] <
                  len1)))
76            res = height[i];
77      printf("%d\n", res);
78   }
79}
```

5 05-数学问题

5.1 01-Bash-Game-sg

```
1/*-----------------------------------------------------------------------
2 *
3 * 文件名称: 01-Bash-Game-sg.cpp
```

```
 * 创建日期: 2021年03月19日 ---- 10时17分
 * 题 目: hdu1846
 * 算 法: sg函数
 * 描 述: <++>
 *
 ------------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 1005;
int n; //石子数量
int m; //一次最多可拿多少石子
int sg[maxn];
int st[maxn]; //后继结点

void SG() {
    memset(sg, 0, sizeof(sg));
    for (int i = 1; i <= n; ++i) {
        memset(st, 0, sizeof(st));
        for (int j = 1; j <= m && i-j >= 0; ++j)
            st[sg[i-j]] = 1; //把i的后继结点(i-1, i-2, i-3, ... , i-j)放到集合st中
        for (int j = 0; j <= n; ++j) //计算sg[i]
            if (!st[j]) {
                sg[i] = j;
                break;
            }
    }
}

int main() {
    int c;
    scanf("%d", &c);
    while (c--) {
        scanf("%d %d", &n, &m);
        SG();
        /*sg != 0 先手胜; sg == 0 后手胜*/
        /*if sg != 0 胜*/
        sg[n] ? printf("first\n") : printf("second\n");
    }
    return 0;
}
```

5.2 01-Catalan

```
#include <cstdio>
#include <vector>
using namespace std;
typedef long long LL;
const LL maxn = 100;
const LL MOD = 1e9 + 9;
vector <long long> fact(maxn + 1, 1LL);
```

```cpp
vector <long long> inv(maxn + 1, 1LL);
LL Catalan[maxn];

template<typename T>
/*快速幂*/
T binPow(T a, T n) {
    T res = 1;
    while (n) {
        if (n & 1) res = (1LL * res * a) % MOD;
        a = (1LL * a * a) % MOD;
        n >>= 1;
    }
    return res;
}

/*阶乘*/
void Factorial() {
    for (int i = 1; i <= maxn; ++i) {
        fact[i] = (fact[i - 1] * i) % MOD;
        inv[i] = binPow(fact[i], MOD - 2);
    }
}

/*组合数*/
LL C(int k, int n) {
    if (k > n) return 0;
    int multiply = (1LL * fact[n] * inv[k]) % MOD;
    multiply = (1LL * multiply * inv[n - k]) % MOD;
    return multiply;
}

/*用来求n较小的卡特兰数，相比较不容易溢出*/
void Catalan_Num() {
    Catalan[0] = Catalan[1] = 1;
    for (int i = 2; i <= 35; ++i) {
        for (int j = 0; j < i; ++j)
            Catalan[i] += Catalan[j] * Catalan[i-j-1];
        printf("%d -> %lld\n", i, Catalan[i]);
    }
}

/*用来求n非常大的卡特兰数，需要求模*/
LL Catalan_Num(int n) {
    return C(n, 2*n) / (n+1);
}

int main() {
    Catalan_Num();
    Factorial();
    printf("num = %lld\n", Catalan[10]);
    printf("num = %lld\n", Catalan_Num(10));
    return 0;
}
```

5.3 01-gcd-lcm

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

int Gcd(int num1, int num2) {
    return !num2 ? num1 : Gcd(num2, num1 % num2);
}

int LCM(int num1, int num2) {
    return num1 / Gcd(num1, num2) * num2;
}

int main() {
    printf("%d\n", Gcd(12, 32));
    int gcd = __gcd(12, 32);
    printf("%d\n", gcd);
    return 0;
}
```

5.4 01-快速幂测试

```cpp
/*--------------------------------------------------------------------------
 *
 * 文件名称: 快速幂.cpp
 * 创建日期: 2021年03月10日 ---- 21时43分
 * 算 法: 快速幂
 * 描 述: 使用了模板template
 *
 --------------------------------------------------------------------------*/

#include <cstdio>
const int mod = 99991;

// 计算 a^b % m
// 1.17秒
long long binaryPow(long long a, long long b, long long m) {
    if (b == 0) return 1;

    if (b % 2 == 1)
        return a * binaryPow(a, b - 1, m) % m;
    else {
        long long mul = binaryPow(a, b / 2, m);
        return mul * mul % m;
    }
}

//0.45秒
long long binPow(long long base, long long expo) {
    long long res = 1;
```

```
    while (expo != 0) {
        if (expo & 1)
            res = (1LL * res * base) % mod;
        base = (1LL * base * base) % mod;
        expo >>= 1;
    }
    return res;
}

/*
 *template<typename T>
 *T binPow(T base, T expo) {
 * if (!expo) return 1;
 * if (expo % 2)
 * return base * binPow(base, expo - 1) % mod;
 * else
 * return binPow(base, expo / 2) % mod * binPow(base, expo / 2) % mod;
 *}
 */

template<typename T>
T binPow(T base, T expo) {
    T res = 1;
    while (expo != 0) {
        if (expo & 1)
            res = (1LL * res * base) % mod;
        base = (1LL * base * base) % mod;
        expo >>= 1;
    }
    return res;
}

int main() {
    long long base = 23, expo = 19898283988388888;
    long long res;
    for (int i = 0; i < 1000000; ++i)
        //res = binaryPow(base, expo, mod);
        res = binPow(base, expo);
    printf("%lld\n", res);
    return 0;
}
```

5.5 01-排列组合

```
/*----------------------------------------------------------------------
 *
 * 文件名称: 01-排列组合.cpp
 * 创建日期: 2021年03月12日 ---- 22时16分
 * 题 目: hdu1521 排列组合
 * 算 法: 指数型母函数
 * 描 述: 有n种物品, 并且知道每种物品的数量, 求从中选出m件物品
```

```
 * 的排列数，例如有两种物品A, B，并且数量都是1,从中选两件物品
 * 则排列有"AB"和"BA"两种
 *
 * 输入：每组输入数据有两行，第一行是两个数n和m(1 <= m, n <= 10)
 * 表示物品数；第二行有n个数，分别表示这n件物品的数量
 *
 * 输出：对应每组数据输出排列数(任何运算不会超过2^31的范围)
 *
 ----------------------------------------------------------------------*/
```

5.6 01-整数除以 2

```cpp
#include <cstdio>
int main() {
    int i = -3;
    printf("%d\n", i / 2); //得到-1，说明C++编译器实现为除以2向0取整
    return 0;
}
```

5.7 01-素数

```cpp
#include <cstdio>
#include <cmath>

bool isPrime1(int n) {
    if (n <= 1)
        return false;
    int sqr = (int)sqrt(1.0 * n);

    for (int i = 2; i <= sqr; i++)
        if (n % i == 0)
            return false;

    return true;
}

//如果n没有接近int型变量的上届
bool isPrime2(int n) {
    if (n <= 1)
        return false;

    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
            return false;

    return true;
}

int main() {
    int num;
```

```
    scanf("%d", &num);
    printf("%d\n", isPrime1(num));
    return 0;
}
```

5.8 01-费马小定理

```
/*----------------------------------------------------------------------
 *
 * 文件名称: 费马小定理.cpp
 * 创建日期: 2021年03月11日 ---- 20时07分
 * 算 法: 费马小定理求逆元
 * 描 述: x^(mod-2)就是逆元
 *
 ----------------------------------------------------------------------*/

#include<cstdio>
typedef long long ll;
const int mod = 99991;

/*
 *ll binPow(ll base, ll expo, ll mod) {
 * if (expo == 0) return 1;
 *
 * if (expo % 2 == 1)
 * return base * binPow(base, expo-1, mod) % mod;
 * else {
 * ll mul = binPow(base, expo/2, mod) % mod;
 * return mul % mod * mul % mod;
 * }
 *}
 */

ll binPow(ll base, ll expo, ll mod) {
    ll res = 1;
    while (expo != 0) {
        if (expo & 1)
            res = (1ll * res * base) % mod;

        base = (1ll * base * base) % mod;
        expo >>= 1;
    }
    return res;
}

ll inv(ll x) {
    return binPow(x, mod-2, mod);
}

int main() {
    for (int i = 0; i < 20; ++i)
```

```
45          printf("%lld ", inv((ll)i));
46      return 0;
47  }
```

5.9 01-进制转换

```
1   /*除了十进制的数用一个int型变量直接存储之外，其他进制的数都用vector容器存储*/
2   #include <cstdio>
3   #include <vector>
4   #include <cmath>
5   using namespace std;
6   // B: bit 二进制
7   // T: ternary 三进制
8   // Q: quaternary 四进制
9   // O: octonary 八进制
10  // D: decimal 十进制
11  // H: hexadecimal 十六进制
12
13  // 将一个P进制的数转换为D进制的数
14  int anytoD(vector<int> num_P, int base_P) {
15      int num_D = 0;
16      for (int i = 0; i < (int)num_P.size(); ++i)
17          num_D += (num_P[i] * pow(base_P, i));
18      return num_D;
19  }
20
21  // 将一个D进制的数转换为C进制的数
22  vector<int> numto;
23  void Dtoany(int num_D, int base_C) {
24      do {
25          numto.push_back(num_D % base_C);
26          num_D /= base_C;
27      }while (num_D != 0);
28  }
29
30  int main() {
31      int num = 101110011;
32      int base_P = 2;
33      vector<int> num_P;
34      while (num) {
35          num_P.push_back(num % 10);
36          num /= 10;
37      }
38      int base_C = 8;
39      int num_D = anytoD(num_P, base_P);
40      Dtoany(num_D, base_C);
41      for (auto it = numto.end()-1; it != numto.begin()-1; --it)
42          printf("%d", *it);
43      return 0;
44  }
```

5.10 02-二进制状态压缩

```cpp
/*--------------------------------------------------------------------------
 *
 * 文件名称: 02—二进制状态压缩.cpp
 * 创建日期: 2021年05月08日 ---- 22时44分
 * 题 目: CH 0103
 * 算 法: 旅行商问题, 哈密顿路径
 * 描 述: 0~n—1这n个点, 从点0开始行走到点n—1结束, 每个点都要经过
 * 且每个点到另外一个点有权重, 问最小的花费是多少
 * 发现:
 * 假设已经经过了0, 1, 2, 3这四个点, 由于要从0开始, 那么会有3! = 6种方式
 * 0 —> 1 —> 2 —> 3 18
 * 0 —> 2 —> 1 —> 3 20
 * ...
 * 在上面的两种方案里, 对于后面的点的选择, 一定不会选择第二种路径
 * 因为我们只需要在经过相同的点且最后的位置相同的行走方式中找到
 * 权重和最小的那种方式就行了, 不管在这些点中是如何行走的, 不会影响到
 * 后面点的选择
 * 1. 哪些点被用过
 * 2. 现在停在哪些点上
 *
 * dp[state][j] = dp[state_k][k] + weight[k][j];
 * state_k是state去掉j的集合, state_k要包含点k
 * 状态压缩, 用一个数二进制中哪些是1来表示这个点被经过
 * 0, 1, 4 —> 10011
 *
 ----------------------------------------------------------------------------*/

#include <cstdio>
#include <cstring>
const int maxn = 20;
#define _min(a, b) (a < b ? a : b)
/*
 * dp[i][j]中i的二进制为1的点已经被经过, 当前处于点j
 * 如dp[7][1]中7的二进制为0111, 则有点0、点1、点2都已经被经过, 当前位于点1
 * 状态转移方程: dp[i][j] = min(dp[i][j], dp[i ^ (1<<j)][k] + weight[k][j]);
 * 既然状态为i的点都被经过, 而当前位于点j, 显然上一个状态是dp[i ^ (1<<j)][k]
    (k是状态i中非j的点)
 */
int dp[1 << maxn][maxn];
int weight[maxn][maxn];

int hamilton(int n) {
   memset(dp, 0x3f, sizeof(dp));
   dp[1][0] = 0; //起始点为0, 所以点0到点0的距离是0
   for (int i = 1; i < (1 << n); ++i)
      for (int j = 0; j < n; ++j) //枚举当前所在的点
         if ((i >> j) & 1)
            //判断路径i中是否包括当前点j,如果包括当前点, 则可以进行状态转移
            for (int k = 0; k < n; ++k) //要完成点k到点j的转移, 所以要来枚举k
               if (i — (1<<j) >> k & 1)
                  //只有去除掉点j后路线中仍然包含点k才能说明路线是在点k的基础上向点j转移的
```

```
                    dp[i][j] = _min(dp[i][j], dp[i-(1<<j)][k] + weight[k][j]);
    return dp[(1 << n)-1][n-1];
        //由于题目要求计算从点n到点n-1的路径长度，所以(1<<n)-1的二进制形式为111...111[共有(n-1)个1]
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", &weight[i][j]);
    int res = hamilton(n);
    printf("%d\n", res);
    return 0;
}
```

5.11 02-埃氏筛法

```
#include <cstdio>
const int maxn = 1e6;
int pnum = 0;
int prime[maxn]; //存储1 ~ 1e6内的所有素数
bool sifter[maxn] = { 0 }; //对1 ~ 1e6内的数标记，未被筛掉的素数标记为false

void findPrime() {
    for (int i = 2; i <= maxn; i++)
        if (sifter[i] == false) {
            prime[pnum++] = i;
            for (int j = i + i; j < maxn; j += i)
                sifter[j] = true;
        }
}

int main() {
    findPrime();
    //输出所有素数
    for (int i = 0; i < pnum; i++)
        printf("%d ", prime[i]);

    if (sifter[99991] == 0)
        printf("\n\n99991 is a prime number\n");
    else
        printf("\n\n99991 is not a prime number\n");

    printf("\n");
    return 0;
}
```

5.12 02-扩展欧几里得

```cpp
#include <cstdio>
const int mod = 99991;

int exGcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }

    int gcd = exGcd(b, a%b, x, y);
    int temp = x;
    x = y;
    y = temp - a / b * y;
    return gcd;
}

/*返回值是a模m的逆元*/
int inv(int a) {
    int x;
    int y;
    int gcd = exGcd(a, mod, x,
        y);//此时得到的x是方程的一个解，但不一定是方程的最小正整数解，x可能为负
    gcd += 1;
    //exGcd(a, mod, x, y);
    return (x % mod + mod) % mod; //(x % m + m) % m
        是方程最小正整数解，也就是a模m的逆元
}

int main() {
    for (int i = 0; i < 20; ++i)
        printf("%d ", inv(i));
    return 0;
}
```

5.13 02-扩展欧几里得算法

```cpp
//可以使用扩展欧几里得算法对方程 ax + by = c 求解
#include<iostream>
using namespace std;

int exgcd(int a, int b, int &x, int &y /*使用引用*/ ) {
    if(b == 0) {
        x = 1;
        y = 0;
        return a;
    }
//欧几里得算法中直接return，而这里由于需要递归得到 x 与 y ，所以不能直接return
    int gcd = exgcd(b, a % b, x, y);
    int temp = x; //为了得到 y_(n) 需要保存 x_(n+1)
    x = y;
```

```
    y = temp - a / b * y;
    return gcd;
}

int main() {
    int a, b, x, y;
    cin >> a >> b;
    int gcd = exgcd(a, b, x, y);

    cout << "a = " << a << endl << "b = " << b << endl;
    cout << "x = " << x << endl << "y = " << y << endl;
    cout << "gcd = " << gcd << endl;
    cout << "a * x + b * y = " << gcd << endl;
    cout << a << " * " << x << " + " << b << " * " << y << " = " << gcd << endl;

    return 0;
}
```

5.14 03-lowbit

```
#include <cstdio>
#define lowbit(x) ((x) & -(x))

int main() {
    printf("%d\n", lowbit(6));
    return 0;
}
```

5.15 03-wythoff-Game

```
/*--------------------------------------------------------------------
 *
 * 文件名称: 03-wythoff-Game.cpp
 * 创建日期: 2021年03月19日 ---- 14时59分
 * 题 目: hdu1527
 * 算 法: 博弈论
 * 描 述: <++>
 *
 --------------------------------------------------------------------*/

#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;
double gold = (1 + sqrt(5)) / 2; //黄金分割 = 1.61803398...

int main() {
    int n, m;
    while (scanf("%d %d", &n, &m)) {
        int mini = min(n, m);
```

```
        int maxi = max(n, m);
        double k = double(maxi - mini);
        int test = (int)(k * gold); //乘以黄金分割数，然后取整
        /*test == mini 先手败*/
        test == mini ? printf("0\n") : printf("1\n");
    }
    return 0;
}
```

5.16 03-快速幂模板

```
#include <cstdio>
typedef long long ll;
int mod;

/* 看题目会不会超过int，选择合适的参数类型
 * 这里还是尽量不要用T，显得很蠢，自己判断会不会超过int，决定使用ll还是int
 */
template<typename> T
T binPow(T base, T expo) {
    T res = 1;
    while (expo) {
        if (expo & 1)
            res = (1LL * res * base) % mod;
        base = (1LL * base * base) % mod;
        //也就是base分别是2, 4, 8, 16, ...
        expo >>= 1;
    }
    return res % mod;
}

int main() {
    int a, b;
    scanf("%d %d %d", &a, &b, &mod);
    int res = (int)binPow((ll)a, (ll)b);
    printf("%d\n", res % mod);
    return 0;
}
```

5.17 03-欧拉筛法

```
#include <cstdio>

const int maxn = 5e8;
int seek = 0;
int prime[maxn];
//最终为false的数是素数
bool sifter[maxn];
//bool* sifter = (bool *)memset(sifter, 0, sizeof(sifter));

```

```
10 void sievePrime() {
11    //从2开始，判断i是否为素数
12    for (int i = 2; i <= maxn; i++) {
13       if (sifter[i] == false)
14          prime[seek++] = i;
15 
16       for (int j = 0; j < seek; j++) {
17          //这里的循环就是为了标记当前素数i的倍数不是素数
18          //不停止的话，就超界了
19          if (i * prime[j] > maxn)
20             break;
21 
22          //每一个倍数标记为不是素数
23          sifter[i * prime[j]] = true;
24 
25          //当前素数为i的最小素因子，分析下一个i
26          if (i % prime[j] == 0)
27             break;
28       }
29    }
30 }
31 
32 int main() {
33    sievePrime();
34    for (int i = 0; i < seek; i++)
35       printf("%d ", prime[i]);
36 
37    /*
38     *if (sifter[99991] == 0)
39     * printf("\n\n99991 is a prime number\n");
40     *else
41     * printf("\n\n99991 is not a prime number\n");
42     */
43 
44    printf("\n");
45    printf("seek = %d", seek); //78498
46    printf("\n");
47    return 0;
48 }
49 
50 /* *
51  * i = 2, 未筛去, prime[1] = 2, j = 1, i*prime[j] = 2*2 = 4 < maxn, 4被筛去,
      i%prime[j] == 2%2 == 0, break
52  * i = 3, 未筛去, prime[2] = 3, j = 1, i*prime[j] = 3*2 = 6 < maxn, 6被筛去,
      i%prime[j] == 3%2 == 1,
53  * j = 2, i*prime[j] = 3*3 = 9 < maxn, 9被筛去, i%prime[j] == 3%3 == 0, break
54  * i = 4, 已筛去, , j = 1, i*prime[j] = 4*2 = 8 < maxn, 8被筛去, i%prime[j] == 4%2
      == 0, break
55  *
56  * 这里为什么i % prime[j] == 0, 就要break呢?
57  * 如果不break, 4*3 = 12就要被筛去, 实际上12会在i = 6, prime[j] = prime[1] = 2, 6*2 =
      12被筛去
58  * 2是12的最小素因子, 比如100的最小素因子是2, 所以100会在50 * 2时筛选掉, 避免重复筛选
```

```
 * 合数的所有最小因子都是素数，此算法用到这一点
 */
```

5.18 03-递推

```cpp
#include<cstdio>
int inv[100001];
int mod = 99991;

//在(mod 99991)的意义下，2 — 10000分别对应的逆元
void Inv() {
    inv[1] = 1;
    for (int i = 2; i < 10000; ++i) {
        inv[i] = — mod/i * inv[mod%i] % mod;
        inv[i] = (mod + inv[i]) % mod;
        /*inv[i] = (mod — mod/i) * inv[mod%i] % mod;*/ /*会溢出*/
    }
}

int main() {
    Inv();
    for (int i = 0; i < 20; ++i)
        printf("%d ", inv[i]);
    return 0;
}
```

5.19 04-n! 中质因子 p 的个数

```cpp
int factor(int n, int p) {
    if (n < p)
        return 0;

    return n / p + factor(n / p , p);
}
```

5.20 04-终极递推

```cpp
#include <cstdio>
typedef long long ll;
const int maxn = 1e7;
ll inv[maxn];
ll mod = 999983;

//生成一个表
void Inv() {
    inv[1] = 1;
    for (int i = 2; i < maxn; ++i)
        inv[i] = (mod — mod/i) % mod * inv[mod%i] % mod;
}
```

```
13
14 //求base的逆元
15 ll InvKB(ll base) {
16    if (base == 1)
17       return 1;
18    return InvKB(mod%base) * (mod-mod/base) % mod;
19 }
20
21 int main() {
22    Inv();
23    for (int i = 0; i < 20; ++i)
24       printf("%lld ", inv[i]);
25
26    printf("%lld\n", InvKB(9));
27    return 0;
28 }
```

5.21 04-高精度加法

```
1  #include <cstdio>
2  #include <vector>
3  #include <string>
4  #include <iostream>
5  #define NEXTLINE cout << endl;
6  using namespace std;
7
8  // C = A + B, A >= 0, B >= 0
9  vector<int> add(vector<int> &A, vector<int> &B) {
10    if (A.size() < B.size())
11       return add(B, A);
12
13    vector<int> C;
14    int t = 0;
15    for (int i = 0; i < A.size(); ++i) {
16       t += A[i];
17       if (i < B.size())
18          t += B[i];
19       C.push_back(t % 10);
20       t /= 10;
21    }
22
23    if (t)
24       C.push_back(t);
25    return C;
26 }
27
28 int main() {
29    ios::sync_with_stdio(false);
30    cin.tie(NULL), cout.tie(NULL);
31
32    string n1, n2;
```

```
    vector<int> A, B;
    cin >> n1 >> n2; //n1 = 123456

    // A = [6, 5, 4, 3, 2, 1];
    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');
    for (int i = n2.length() - 1; i >= 0; --i)
        B.push_back(n2[i] - '0');

    auto C = add(A, B);
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    NEXTLINE;
    return 0;
}
```

5.22 05-素因子

```
#include <cstdio>
#include <cmath>
#include <vector>
using namespace std;
const int maxn = 1e2;
/*int primeFactor[maxn];*/

vector<int> primeFactor;
template<typename T>
void PrimeFactor(T num) {
    for (T i = 2; pow(i, 2) <= num; ++i)
        if (!(num % i)) {
            primeFactor.push_back(i);
            /*primeFactor[cnt++] = i;*/
            while (!(num % i)) num /= i;
        }
    if (num > 1) primeFactor.push_back(num);
    /*primeFactor.clear();*/
}

int main() {
    int num;
    scanf("%d", &num);
    PrimeFactor(num);
    printf("cnt = %d\n", (int)primeFactor.size());
    for (int i = 0; i < (int)primeFactor.size(); ++i)
        printf("%d ", primeFactor[i]);
    return 0;
}
```

5.23 05-高精度减法

```cpp
#include <cstdio>
#include <vector>
#include <string>
#include <iostream>
#include <algorithm>
using namespace std;
#define NEXTLINE cout << endl;

// C = A - B, 满足A >= B, A >= 0, B >= 0
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); ++i) {
        t = A[i] - t;
        if (i < B.size())
            t -= B[i];
        C.push_back((t + 10) % 10);
        t < 0 ? t = 1 : t = 0;
    }

    while (C.size() > 1 && C.back() == 0)
        C.pop_back();
    return C;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    string n1, n2;
    cin >> n1 >> n2;
    vector<int> A, B;

    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');
    for (int i = n2.length() - 1; i >= 0; --i)
        B.push_back(n2[i] - '0');

    //判断A < B, 则swap
    if (n1.length() < n2.length() || (n1.length() == n2.length() && n1 < n2))
        swap(A, B);

    auto C = sub(A, B);

    if (n1.length() < n2.length() || (n1.length() == n2.length() && n1 < n2))
        cout << "-";
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    NEXTLINE;
    return 0;
}
```

5.24 06-质因子分解

```cpp
#include <cstdio>
#include <cmath>

struct Factor {
    int x;
    int cnt;
} factor[10];

const int maxn = 1e4;
bool sifter[maxn] = {0};
int pnum = 0;
int j = 0;
int prime[maxn];


void findPrime() {
    for (int i = 2; i <= maxn; i++) {
        //发现素数
        if (sifter[i] == false) {
            prime[pnum++] = i;
            for (int j = i + i; j < maxn; j += i)
                sifter[j] = true;
        }
    }
}

void findFactor(int num) {
    int sqrnum = sqrt(1.0 * num);

    for (int i = 0; i < sqrnum; i++)
        if (num % prime[i] == 0) {
            factor[j].x = prime[i];
            factor[j].cnt = 0;
            while (num % prime[i] == 0) {
                factor[j].cnt++;
                num /= prime[i];
            }
            j++;
        }
    if (num != 1) {
        factor[j].x = num;
        factor[j].cnt = 1;
    }
}

int main() {
    int num = 12345;
    findPrime();
    findFactor(num);

    for (int i = 0; i <= j; i++)
```

```
52          printf("%d^%d + ", factor[i].x, factor[i].cnt);
53
54      printf("\b= ");
55      printf("%d\n", num);
56      return 0;
57  }
```

5.25 06-高精度乘法

```
1   #include <cstdio>
2   #include <iostream>
3   #include <string>
4   #include <vector>
5   using namespace std;
6   #define NEXTLINE cout << endl;
7
8   // C = A * b, A >= 0, b >= 0
9   vector<int> mul(vector<int> &A, int b) {
10      vector<int> C;
11      for (int i = 0, t = 0; i < A.size() || t; ++i) {
12          if (i < A.size())
13              t += A[i] * b;
14          C.push_back(t % 10);
15          t /= 10;
16      }
17
18      while (C.size() > 1 && C.back() == 0)
19          C.pop_back();
20      return C;
21  }
22
23  int main() {
24      ios::sync_with_stdio(false);
25      cin.tie(NULL), cout.tie(NULL);
26
27      string n1;
28      int b;
29      cin >> n1 >> b;
30      vector<int> A;
31
32      for (int i = n1.length() - 1; i >= 0; --i)
33          A.push_back(n1[i] - '0');
34
35      auto C = mul(A, b);
36      for (int i = C.size() - 1; i >= 0; --i)
37          cout << C[i];
38      NEXTLINE;
39      return 0;
40  }
```

5.26 07-高精度除法

```cpp
#include <cstdio>
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
#define NEXTLINE cout << endl;

// A / b = C ... r, A >= 0, b > 0
vector<int> div(vector<int> &A, int b, int &r) {
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; --i) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }

    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0)
        C.pop_back();
    return C;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL), cout.tie(NULL);

    string n1;
    int b;
    cin >> n1 >> b;
    vector<int> A;

    for (int i = n1.length() - 1; i >= 0; --i)
        A.push_back(n1[i] - '0');

    int r;
    auto C = div(A, b, r);
    for (int i = C.size() - 1; i >= 0; --i)
        cout << C[i];
    cout << endl << r << endl;
    // NEXTLINE;
    return 0;
}
```

5.27 Fibonacci

```cpp
#include <cstdio>
#include <cmath>
```

```cpp
#include <vector>
using namespace std;

template<typename T>
T Fibo0(T n) {
    if (n <= 1) return 1;
    else return Fibo(n-1) + Fibo(n-2);
}

template<typename T>
T Fibo1(T n) {
    if (n <= 1) return 1;

    std::vector<int> table(n + 1);
    table[0] = table[1] = 1;
    for (int i = 2; i <= n; ++i)
        table[i] = table[i-1] + table[i-2];
    return table.back();
}

template<typename T>
T Fibo2(T n) {
    const double sqrt5 = std::sqrt(5);
    const double phi = (1 + sqrt5) / 2;
    return (T)(std::pow(phi, n+1) / sqrt5 + 0.5);
}

template<typename T>
T Fibo3(T n) {
    static std::vector<T> arr;
    if (n <= 1) return 1;
    else if (n >= (T)arr.size())
        arr.resize(n+1);

    if (!arr[n])
        arr[n] = Fibo3(n-1) + Fibo3(n-2);
    return arr[n];
}

int main() {
    for (int i = 0; i < 10; ++i)
        printf("%d ", Fibo1(i));

    printf("\n");
    for (int i = 0; i < 10; ++i)
        printf("%d ", Fibo2(i));

    printf("\n");
    for (int i = 0; i < 10; ++i)
        printf("%d ", Fibo3(i));

    printf("\n");
    return 0;
```

```
}
```

6 06-数据结构

6.1 01-Segment-Tree

```
/*--------------------------------------------------------------------
 *
 * 文件名称: Segment_Tree.cpp
 * 创建日期: 2021年04月23日 ---- 21时06分
 * 题 目: <++>
 * 算 法: 线段树
 * 描 述: 线段树结点中没有存储区间，但由于build, update, query
 * 操作的参数都包含了start与end，所以相当于存储了区间
 *
 --------------------------------------------------------------------*/

#include <cstdio>
using namespace std;
const int maxn = 1000;
int arr[] = {1, 3, 5, 7, 9, 11};
int tree[maxn];
int n = 6; //数组中元素的个数

// build(0, 0, n-1)，从根节点开始建树，从arr数组的start到end，node是树的结点
void build(int node, int start, int end) {
    if (start == end)
        tree[node] = arr[start];
    else {
        int mid = (start + end) / 2;
        int left_node = 2 * node + 1;
        int right_node = 2 * node + 2;

        build(left_node, start, mid);
        build(right_node, mid+1, end);

        tree[node] = tree[left_node] + tree[right_node];
    }
}

// update(0, 0, n-1, idx, val)，arr[idx] = val
void update(int node, int start, int end, int idx, int val) {
    if (start == end) {
        arr[idx] = val;
        tree[node] = val;
    }
    else {
        int mid = (start + end) / 2;
        int left_node = 2 * node + 1;
        int right_node = 2 * node + 2;

```

```
46          if (idx <= mid)
47              update(left_node, start, mid, idx, val);
48          else
49              update(right_node, mid+1, end, idx, val);
50
51          tree[node] = tree[left_node] + tree[right_node];
52      }
53  }
54
55  // query(0, 0, n-1, L, R), 求arr[L], arr[L+1], ... arr[R]之间的和
56  int query(int node, int start, int end, int L, int R) {
57      printf("start = %d\n", start);
58      printf("end = %d\n", end);
59
60      if (R < start || L > end)
61          return 0;
62      else if (L <= start && R >= end)
63          return tree[node];
64      else if (start == end)
65          return tree[node];
66      else {
67          int mid = (start + end) / 2;
68          int left_node = 2 * node + 1;
69          int right_node = 2 * node + 2;
70          int sum_left = query(left_node, start, mid, L, R);
71          int sum_right = query(right_node, mid+1, end, L, R);
72          return sum_left + sum_right;
73      }
74  }
75
76  int main() {
77      build(0, 0, n-1);
78      for (int i = 0; i <= 2*n+2; ++i)
79          printf("tree[%d] = %d\n", i, tree[i]);
80      printf("\n");
81
82      update(0, 0, n-1, 4, 6);
83      for (int i = 0; i <= 2*n+2; ++i)
84          printf("tree[%d] = %d\n", i, tree[i]);
85      printf("\n");
86
87      int s = query(0, 0, n-1, 2, 5);
88      printf("%d\n", s);
89      return 0;
90  }
```

6.2 01-Sliding-Window

```
1  /*-------------------------------------------------------------------
2   *
3   * 文件名称: 01-单调队列.cpp
```

```
 * 创建日期: 2021年04月01日 ---- 19时33分
 * 题 目: poj2823 Sliding Window
 * 算 法: 单调队列
 * 描 述: <++>
 *
 ------------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e6 + 5;
int arr[maxn];
int quu[maxn];
int n, k;

/*
 * 所以队列是递增的, 队头是最小值
 * 队列中存储的是下标
 * 如果下一个元素比队列中的小或者相等, 那么队尾出队, 也就是--tail
 */
void getmin() {
    int head = 0;
    int tail = 0;
    // 先处理前k-1个
    for (int i = 1; i < k; ++i) {
        while (tail >= head && arr[i] <= arr[quu[tail]])
            --tail;
        quu[++tail] = i;
    }
    for (int i = k; i <= n; ++i) {
        while (tail >= head && arr[i] <= arr[quu[tail]])
            --tail;
        quu[++tail] = i;
        while (quu[head] <= i - k)
            ++head;
        printf("%d ", arr[quu[head]]);
    }
    printf("\n");
}

/*与上一个函数一个板子, 只是这个队列是递减的*/
void getmax() {
    int head = 0;
    int tail = 0;
    for (int i = 1; i < k; ++i) {
        while (tail >= head && arr[i] >= arr[quu[tail]])
            --tail;
        quu[++tail] = i;
    }
    for (int i = k; i <= n; ++i) {
        while (tail >= head && arr[i] >= arr[quu[tail]])
            --tail;
        quu[++tail] = i;
        while (quu[head] <= i - k)
            ++head;
```

```
57          printf("%d ", arr[quu[head]]);
58      }
59      printf("\n");
60  }
61
62  int main() {
63      scanf("%d %d", &n, &k);
64      for (int i = 1; i <= n; ++i)
65          scanf("%d", &arr[i]);
66      getmin();
67      getmax();
68      return 0;
69  }
```

6.3 01-单调栈

```
1   /*------------------------------------------------------------------------
2    *
3    * 文件名称: 01-单调栈.cpp
4    * 创建日期: 2021年04月01日 ---- 16时52分
5    * 题 目: poj3250 bad hair day
6    * 算 法: 单调栈
7    * 描 述: 如果来了一个个高的，那就--top直到前面那个是比自己高
8    * 一点的(top代表的是当前的能够被几只牛看得见)，因为夹在他们
9    * 之间的那群牛往后都看不见新加入的了，如果来了一只个矮的，那就
10   * ++top，表示这个新牛能够被++top只牛看得见
11   *
12   ------------------------------------------------------------------------*/
13
14  #include <cstdio>
15  const int maxn = 80005;
16  int stk[maxn];
17  int top;
18
19  int main() {
20      int n;
21      long long res = 0;
22      scanf("%d", &n);
23      for (int i = 0; i < n; ++i) {
24          int high;
25          scanf("%d", &high);
26          while (top > 0 && high >= stk[top-1])
27              --top;
28          res += ++top; //能够被几个人看得见
29          stk[top] = high;
30      }
31      printf("%lld\n", res);
32      return 0;
33  }
```

6.4 02-最大子序列和

```cpp
/*--------------------------------------------------------------------------------
 *
 * 文件名称: 02—最大子序列和.cpp
 * 创建日期: 2021年04月08日 ———— 21时02分
 * 题 目: ch1201
 * 算 法: 单调队列
 * 描 述: 给定一个长度为N的整数序列(可能有负数), 从中找出一段
 * 长度不超过M的连续子序列, 使得所有子序列中所有数的和最大
 *
 *
 --------------------------------------------------------------------------------*/

#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn = 3e5 + 5;
int sum[maxn];
int quu[maxn];
int head = 0;
int tail = 0;
int res;

int main() {
    // freopen("in.txt", "r", stdin);
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0; i < n; ++i) {
        scanf("%d", &sum[i]);
        if (i)
            sum[i] += sum[i-1];
    }
    for (int i = 0; i < n; ++i) {
        while (head <= tail && i-quu[i] > m) //超出M的范围
            ++head;
        res = max(res, sum[i] - sum[quu[head]]);
        //为了保证队列单调, 要使quu[tail]的位置的前缀和小于sum[i]才行
        while (head <= tail && sum[i] <= sum[quu[tail]])
            --tail;
        quu[++tail] = i;
    }
    printf("%d\n", res);
    return 0;
}
```

6.5 02-逆序链表迭代

```cpp
#include <cstdio> //NULL在cstdio头文件中被定义
struct node {
    int data;
```

```
4       struct node* next;
5   };
6   typedef struct node* pNode;
7
8   pNode creatList() {
9       int n;
10      scanf("%d", &n);
11
12      pNode cursor = new node;
13      pNode head = cursor;
14      scanf("%d", &cursor -> data);
15      n--;
16
17      while (n--) {
18          pNode chains = new node;
19          chains -> next = NULL;
20          scanf("%d", &chains -> data);
21          cursor -> next = chains;
22          cursor = chains;
23      }
24      return head;
25  }
26
27  pNode reverse(pNode head) {
28      pNode current = head;
29      pNode next = NULL, result = NULL;
30      while (current != NULL) {
31          next = current -> next;
32          current -> next = result;
33          result = current;
34          current = next;
35      }
36      return result;
37  }
38
39  pNode Reverse(pNode head) {
40      pNode succ = NULL;
41      pNode tail = NULL;
42      while (head != NULL) {
43          succ = head -> next;
44          head -> next = tail; //pNode = pNode
45          tail = head;
46          head = succ;
47      }
48      return tail;
49  }
50
51  void outputList(pNode head) {
52      pNode cursor = head;
53      while (cursor -> next != NULL) {
54          printf("%d ", cursor -> data);
55          cursor = cursor -> next;
56      }
```

```
    printf("%d", cursor -> data);
}

int main() {
    printf("%d", (int)sizeof(pNode));
    printf("%d", (int)sizeof(node));
    pNode head = creatList();
    //pNode result = reverse(head);
    pNode result = Reverse(head);
    outputList(result);
    return 0;
}
```

6.6 Template-并查集

```
//蓝书，本质是路径压缩
#include <cstdio>
const int maxn = 1e6 + 5;
int n, m;
int fa[maxn];
bool isroot[maxn];
int groups = 0;

void init() {
    for (int i = 0; i < n; ++i) {
        fa[i] = -1;
        isroot[i] = false;
    }
}

int root(int x) {
    if (fa[x] == -1)
        return x;
    return fa[x] = root(fa[x]);
}

void union_vert(int x, int y) {
    fa[root(x)] = root(y);
}

int main() {
    scanf("%d %d", &n, &m);
    init();
    for (int i = 0; i < m; ++i) {
        int ver1, ver2;
        scanf("%d %d", &ver1, &ver2);
        union_vert(ver1, ver2);
    }
    for (int i = 0; i < n; ++i)
        isroot[root(i)] = true;
    for (int i = 0; i < n; ++i)
```

```
        groups += isroot[i];
    printf("%d", groups);
    return 0;
}
```

6.7 对顶堆

```
/*------------------------------------------------------------------------
 *
 * 文件名称: 对顶堆.cpp
 * 创建日期: 2021年06月02日 星期三 22时21分50秒
 * 题 目: AcWing 0106 动态中位数
 * 算 法: 对顶堆
 * 描 述: 维护一个动态中位数
 * 一般堆用he(heap)表示, 大根堆用hE表示
 * 哈希用ha(hash)表示
 *
 ------------------------------------------------------------------------*/

#include <cstdio>
#include <queue>
#include <vector>
using namespace std;
#define NEXTLINE puts("");

int main() {
    // t组数, m是各组编号, n是各组数据个数
    int t, m, n;
    scanf("%d", &t);
    while (t--) {
        priority_queue<int> hE; // hE为大根堆
        priority_queue<int, vector<int>, greater<int>> he; // he为小根堆
        scanf("%d %d", &m, &n);
        printf("%d %d\n", m, (n + 1) / 2);
        int cnt = 0;
        for (int i = 1; i <= n; ++i) {
            int _;
            scanf("%d", &_);
            he.push(_);
            if (hE.size() && hE.top() > he.top()) {
                int a = he.top(),
                    b = hE.top();
                he.pop(), hE.pop();
                he.push(b), hE.push(a);
            }
            while (he.size() > hE.size()) {
                hE.push(he.top());
                he.pop();
            }
            if (i & 1)//奇数
                printf("%d%c", hE.top(), ++cnt % 10 == 0 ? '\n' : ' ');//每10个数换一行
```

```
45          }
46          if (cnt % 10)
47              NEXTLINE
48      }
49      return 0;
50  }
```

6.8 并查集合并压缩

```
1   //合并压缩是在'并'的过程压缩路径
2   //额外需要rank[]数组，储存当前树的高度
3   //需要分组数时，同样使用isroot[]数组
4   #include <cstdio>
5   const int maxn = 1e6;
6   int fa[maxn];
7   bool isroot[maxn];
8   int ranks[maxn];
9
10  void init(int vert) {
11      for (int i = 0; i < vert; ++i) {
12          fa[i] = -1;
13          isroot[i] = false;
14          ranks[i] = -1;
15      }
16  }
17
18  int root1(int x) {
19      if (fa[x] == -1)
20          return x;
21      else
22          return root1(fa[x]);
23  }
24
25  int root2(int x) {
26      int rootx = x;
27      while (fa[rootx] != -1)
28          rootx = fa[rootx];
29      return rootx;
30  }
31
32  void union_vert(int x, int y) {
33      int rootx = root1(x);
34      int rooty = root1(y);
35
36      if (rootx != rooty)
37          return ;
38      else {
39          if (ranks[rootx] > ranks[rooty])
40              fa[rooty] = rootx;
41          else if (ranks[rooty] > ranks[rootx])
42              fa[rootx] = rooty;
```

```
        else {
            fa[rootx] = rooty;
            ranks[rooty]++;
        }
    }
}

int main() {
    int vert;
    int groups;
    scanf("%d", &vert);
    init(vert);
    scanf("%d", &groups);
    while (groups--) {
        int ver1, ver2;
        scanf("%d %d", &ver1, &ver2);
        union_vert(ver1, ver2);
    }

    for (int i = 0; i < vert; ++i)
        isroot[root1(i)] = true;

    int ans = 0;
    for (int i = 0; i < vert; ++i)
        ans += isroot[i];
    printf("%d\n", ans);
    return 0;
}
```

6.9 并查集路径压缩

```
//路径压缩是在'查'的过程压缩路径
//没有额外需要
//另外，isroot[]可以很好的与并查集结合，找到分组数
#include <cstdio>
const int maxn = 1e6;
int fa[maxn];
bool isroot[maxn];

void init(int vert) {
    for (int i = 0; i < vert; ++i) {
        fa[i] = -1;
        isroot[i] = false;
    }
}

//查，有返回值，为此结点的根结点
int root1(int x) {
    if (fa[x] == -1)
        return x;
    else {
```

```
21          //这条语句只执行了一遍
22          int rootx = root1(fa[x]);
23          //路径压缩
24          fa[x] = rootx;
25          return rootx;
26      }
27  }
28
29  int root2(int x) {
30      int rootx = x;
31      while (fa[rootx] != -1) {
32          rootx = fa[rootx];
33      }
34  //路径压缩
35      while (fa[x] != -1) {
36          int ano_x = x;
37          fa[x] = rootx;
38          x = fa[ano_x];
39      }
40      return rootx;
41  }
42
43  //根据题目有无返回值，isroot[]数组可找到分组数
44  //如果有返回值，根据返回值判断当前值是否在同一集合中
45  void union_vert(int x, int y) {
46      int rootx = root1(x);
47      int rooty = root1(y);
48
49      if (rootx != rooty)
50          fa[rootx] = rooty;
51  }
52
53  int main() {
54      int n;
55      int groups;
56      scanf("%d", &n);
57      init(n);
58      scanf("%d", &groups);
59
60      for (int i = 0; i < groups; ++i) {
61          int ver1, ver2;
62          scanf("%d %d", &ver1, &ver2);
63          union_vert(ver1, ver2);
64      }
65
66      for (int i = 0; i < n; ++i)
67          //默认编号从0到vert
68          isroot[root1(i)] = true;
69
70      int ans = 0;
71      for (int i = 0; i < n; ++i)
72          ans += isroot[i];
73      printf("%d", ans);
```

```

    return 0;
}
```

6.10 数状数组

```
/*------------------------------------------------------------------------
 *
 * 文件名称: 数状数组.cpp
 * 创建日期: 2021年03月07日 ---- 11时36分
 * 题 目: poj2182
 * 算 法: 数状数组
 * 描 述: <++>
 *
------------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e6;
int tree[maxn];
int n;
#define lowbit(x) ((x) & -(x))

/*tree[x] += d*/
void add(int x, int d) {
    while (x <= n) {
        tree[x] += d;
        x += lowbit(x);
    }
}

/*sum(tree[1], tree[x])*/
int sum(int x) {
    int sum = 0;
    while (x > 0) {
        sum += tree[x];
        x -= lowbit(x);
    }
    return sum;
}

int findpos(int x) {
    int L = 1;
    int R = n;
    while (L < R) {
        int mid = (L + R) >> 1;
        sum(mid) < x ? L = mid+1 : R = mid;
    }
    return L;
}

int main() {
```

```
46      int pre[maxn];
47      int ans[maxn];
48      scanf("%d", &n);
49      pre[1] = 0;
50      for (int i = 2; i <= n; ++i)
51          scanf("%d", &pre[i]);
52      for (int i = 1; i <= n; ++i)
53          tree[i] = lowbit(i);
54      for (int i = n; i > 0; --i) {
55          int x = findpos(pre[i] + 1);
56          add(x, -1);
57          ans[i] = x;
58      }
59      for (int i = 1; i <= n; ++i)
60          printf("%d\n", ans[i]);
61      return 0;
62  }
```

7 07-图论

7.1 01-BFS-邻接矩阵

```
1   #include <cstdio>
2   #include <cstring>
3   #include <queue>
4   using namespace std;
5   const int maxn = 1000; /*如果顶点数大于1000，就不再建议使用邻接矩阵*/
6   int n, m; /*分别是顶点个数，边的个数*/
7   bool g[maxn][maxn];
8   bool inq[maxn]; /*当前在队列中的顶点*/
9
10  void BFS(int vert) {
11      queue<int> quu;
12      quu.push(vert);
13      inq[vert] = true;
14      while (!quu.empty()) {
15          int vert = quu.front();
16          printf("%d ", vert);
17          quu.pop();
18          for (int i = 0; i < n; ++i)
19              if (inq[i] == false && g[vert][i]) {
20                  quu.push(i);
21                  inq[i] = true;
22              }
23      }
24  }
25
26  int main() {
27      freopen("in.txt", "r", stdin);
28      scanf("%d %d", &n, &m);
29      memset(g, 0, sizeof(g));
```

```
    for (int i = 0; i < m; ++i) {
        int ver1;
        int ver2;
        scanf("%d %d", &ver1, &ver2);
        g[ver1][ver2] = 1;
        g[ver2][ver1] = 1;
    }
    /*一般图为连通图，那么只需要一次广搜遍历就行了*/
    /*如果为非连通图，需要检查每一个顶点*/
    for (int i = 0; i < n; ++i)
        if (inq[i] == false)
            BFS(i);
    return 0;
}
```

7.2 01-DFS-邻接矩阵

```
#include <cstdio>
#include <cstring>
const int maxn = 1000; /*顶点数再多就不再建议使用邻接矩阵了*/
int n, m; /*分别是顶点个数，边的个数*/
int g[maxn][maxn];
bool used[maxn]; /*DFS搜索需要标记已访问过的顶点*/

void DFS(int vert, int depth) {
    printf("%d ", vert);
    used[vert] = true; /*将已访问过的顶点标记为已被访问*/
    for (int i = 0; i < n; ++i)
        /*邻接矩阵初始化时就需要初始化为无穷大*/
        /*所以也就是在这里判断是否是未输入的边*/
        if (used[i] == false && g[vert][i])
            DFS(i, depth + 1);
}

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    memset(g, 0, sizeof(g));
    for (int i = 0; i < m; ++i) {
        int ver1;
        int ver2;
        scanf("%d %d", &ver1, &ver2);
        g[ver1][ver2] = 1;
        g[ver2][ver1] = 1;
    }
    /*遍历图需要搜索每个顶点，首先需要判断是否已经访问*/
    for (int i = 0; i < n; ++i)
        if (used[i] == false)
            DFS(i, 1); /*1表示第一层*/
    return 0;
}
```

7.3 01-Floyd

```cpp
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
int n, m;
int dist[305][305];

void Floyd() {
   /*Floyd求任意两点间最短路径*/
   for (int k = 0; k < n; ++k)
      for (int i = 0; i < n; ++i)
         for (int j = 0; j < n; ++j)
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}

int main() {
   scanf("%d %d", &n, &m);
   memset(dist, 0x3f, sizeof(dist));
   for (int i = 0; i < n; ++i)
      dist[i][i] = 0; /*结点i到结点i的距离为0*/
   for (int i = 0; i < n; ++i) {
      int x, y, z;
      scanf("%d %d %d", &x, &y, &z);
      dist[x][y] = min(dist[x][y], z); /*多重边*/
   }
   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j)
         printf("%d ", dist[i][j]);
      printf("\n");
   }
   return 0;
}
```

7.4 01-kruskal

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;
struct Info {int x, y, z;} edge[500010];
int n, m;
int fa[100010];
int res;
bool operator <(Info a, Info b) {
   return a.z < b.z;
}

int root(int x) {
   if (fa[x] == -1)
      return x;
```

```
    return fa[x] = root(fa[x]);
}

void union_vert() {
    for (int i = 0; i < n; ++i) {
        int x_root = root(edge[i].x);
        int y_root = root(edge[i].y);
        if (x_root == y_root)
            continue;
        fa[x_root] = y_root;
        res += edge[i].z;
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i)
        scanf("%d %d %d", &edge[i].x, &edge[i].y, &edge[i].z);
    sort(edge, edge+m);
    /*init*/
    for (int i = 0; i < n; ++i)
        fa[i] = -1;
    printf("%d\n", res);
    return 0;
}
```

7.5 01-prim

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1000;
int n, m;
int source;
int target;
int g[maxn][maxn];
int dist[maxn];
bool used[maxn];
int res;

void prim() {
    memset(dist, 0x3f, sizeof(dist));
    memset(used, 0, sizeof(used));
    dist[source] = 0;
    for (int i = 0; i < n; ++i) {
        int vert = -1;
        for (int j = 0; j < n; ++j)
            if (used[j] == false && (vert == -1 || dist[j] < dist[vert]))
                vert = j; /*筛选出未在集合中的距离集合最近的顶点，*/
        used[vert] = true; /*将vert加到集合中*/
```

```
        for (int j = 0; j < n; ++j)
            if (used[vert] == false)
                dist[j] = min(dist[j], g[vert][j]); /*更新顶点距离*/
    }
}

int main() {
    scanf("%d %d", &n, &m);
    memset(g, 0x3f, sizeof(g));
    for (int i = 0; i < n; ++i)
        g[i][i] = 0;
    for (int i = 0; i < n; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        g[x][y] = z;
        g[y][x] = z;
    }
    prim();
    for (int i = 0; i < n; ++i)
        res += dist[i]; /*i到源点的距离和*/
    printf("%d \n", res);
    return 0;
}
```

7.6 01-SPFA

```
#include <cstdio>
#include <queue>
#include <cstring>
using namespace std;
const int N = 1e5 + 5;
const int M = 1e5 + 5;
int n, m;
int source;
int tot;
int head[N];
int vert[M];
int edge[M];
int nxet[M];
int dist[N];
bool used[N];
queue<int> quu;

void add(int x, int y, int z) {
    /*输入的第tot条边的入边是y，权重为z*/
    vert[++tot] = y;
    edge[tot] = z;
    /*nxet数组中存放上一次输入x时的tot值*/
    /*第一次输入x时，nxet[tot] == 0，也是下面循环结束的依据*/
    nxet[tot] = head[x];
    head[x] = tot;
```

```
}

void spfa() {
    memset(dist, 0x3f, sizeof(dist));
    memset(used, 0, sizeof(used));
    dist[source] = 0;
    used[source] = true;
    quu.push(source);
    while (quu.size()) {
        /*取出队首元素*/
        int x = quu.front();
        quu.pop();
        used[x] = 0;
        /*扫描所有出边*/
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            int z = edge[i];
            if (dist[y] > dist[x] + z) {
                dist[y] = dist[x] + z;
                if (used[y] == false)
                    quu.push(y),
                    used[y] = true;
            }
        }
    }
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
    }
    spfa();
    for (int i = 0; i < n; ++i)
        printf("%d\n", dist[i]);
    return 0;
}
```

7.7 01-邻接矩阵

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1000;
int n; /*顶点个数*/
int m; /*边的个数*/
int source = 1;
int g[maxn][maxn];
```

```
10 int dist[maxn]; /*d数组，存放源点到各顶点的距离*/
11 bool used[maxn];
12 int pre[maxn]; /*表示从起点到顶点vert的最短路径上vert的前一个顶点*/
13
14 void dijkstra() {
15    memset(dist, 0x3f, sizeof(dist));
16    memset(used, 0, sizeof(used));
17    for (int i = 0; i < n; ++i)
18       pre[i] = i; /*初始时，每个顶点的前驱为自身*/
19    dist[source] = 0; /*源点到源点的距离为0，可更改*/
20    for (int i = 0; i < n; ++i) {
          /*每次仅开放一个顶点，所以需要重复访问n-1次(源点不需重复访问)*/
21       int vert = -1; /*特判源点*/
22       for (int j = 0; j < n; ++j)
             /*找到未访问过的顶点中有路径可以到达的距离源点最短的顶点*/
23          if (used[j] == false && (vert == -1 || dist[j] < dist[vert])) /*特判源点*/
24             vert = j;
25       used[vert] = true; /*标记已被访问*/
26       for (int j = 0; j < n; ++j) /*更新能够被更新的顶点与距源点source的最短距离*/
27          dist[j] = min(dist[j], dist[vert] + g[vert][j]);
28       /*
29        * //如果需要求出最短路径，选择这个for循环，删除上一个for循环
30        *for (int j = 0; j < n; ++j) [>更新能够被更新的顶点与距源点的最短距离<]
31        * if (dist[vert] + g[vert][j] < dist[j]) {
32        * dist[j] = dist[vert] + g[vert][j];
33        * pre[j] = vert;
34        * }
35        */
36    }
37 }
38
39 /*从源点到目标点vert的最短路径*/
40 void route(int source, int vert) {
41    if (vert == source) {
42       printf("%d ", source);
43       return ;
44    }
45    route(source, pre[vert]);
46    printf("%d ", vert);
47 }
48
49 int main() {
50    freopen("in.txt", "r", stdin);
51    freopen("out.txt", "w", stdout);
52    scanf("%d", &n);
53    scanf("%d", &m);
54    source = 1; /*源点为1*/
55    memset(g, 0x3f, sizeof(g));
56    for (int i = 1; i <= n; ++i) /*没有0顶点*/
57       g[i][i] = 0;
58    for (int i = 1; i <= m; ++i) {
59       int vert;
60       int edge;
```

```
61          int weig;
62          scanf("%d", &vert);
63          scanf("%d", &edge);
64          scanf("%d", &weig);
65          g[vert][edge] = min(g[vert][edge], weig);
66      }
67      dijkstra();
68      for (int i = 1; i <= n; ++i)
69          printf("%d\n", dist[i]);
70      /*输出source到6的路径*/
71      route(source, 6);
72      return 0;
73  }
```

7.8 02-BFS-邻接表

```
1   #include <cstdio>
2   #include <queue>
3   #include <vector>
4   using namespace std;
5   const int maxn = 1e4; //顶点比1e3少的话，就可以使用邻接矩阵
6   int n; //顶点的个数
7   //没有边权
8   vector<int> g[maxn];
9   bool inq[maxn]; //当前在队列中的顶点
10
11  void BFS(int vertex) {
12      queue<int> quu;
13      quu.push(vertex);
14      inq[vertex] = true;
15      while (!quu.empty()) {
16          int vert = quu.front();
17          quu.pop();
18          for (int i = 0; i < (int)g[vert].size(); ++i) {
19              int v = g[vert][i];
20              if (inq[v] == false) {
21                  quu.push(v);
22                  inq[v] = true;
23              }
24          }
25      }
26  }
27
28  int main() {
29      //一般图为连通图，那么只需要一次广搜遍历就行了
30      //如果为非连通图，需要检查每一个顶点
31      for (int i = 0; i < n; ++i)
32          if (inq[i] == false)
33              BFS(i);
34      return 0;
35  }
```

7.9 02-DFS-邻接表

```cpp
#include <cstdio>
#include <vector>
using namespace std;
const int maxn = 1e4; //顶点数比1e3少的话，就可以使用邻接矩阵
int n, m; //顶点的个数，边的个数
vector<int> g[maxn]; //邻接矩阵与邻接表都使用一个变量名，好记
bool used[maxn]; //DFS搜索需要标记已访问过的顶点

void DFS(int vertex) {
    used[vertex] = true;
    for (int i = 0; i < (int)g[vertex].size(); ++i) {
        int vert = g[vertex][i];
        if (used[vert] == false)
            DFS(vert);
    }
}

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int ver1, ver2;
        scanf("%d %d", &ver1, &ver2);
        g[ver1].push_back(ver2);
    }
    for (int i = 0; i < n; ++i)
        if (used[i] == false)
            DFS(i);
    return 0;
}
```

7.10 02-二叉堆

```cpp
#include <cstdio>
#include <cstring>
#include <queue>
#include <utility>
using namespace std;
const int N = 2000; //顶点数的最大值
const int M = 2000; //边数的最大值
int n, m;
int source;
int tot;
int head[N], vert[M], edge[M], nxet[M];
int dist[N]; //d数组，存放源点到各顶点的最短距离
bool used[N];
int pre[N]; //表示从起点到顶点vert的最短路径上vert的前一个顶点
// 大根堆(优先队列)，pair的第二维是顶点编号
// pair的第一维是dist的相反数(利用相反数变成小根堆)
```

```cpp
priority_queue<pair<int, int>> pqu;

void add(int x, int y, int z) {
    // 输入的第tot条边的入边是y，权重为z
    vert[++tot] = y;
    edge[tot] = z;
    // nxet数组中存放上一次输入x时的tot值
    // 第一次输入x时，nxet[tot] == 0，也是下面循环结束的依据
    nxet[tot] = head[x];
    head[x] = tot;
}

void dijkstra() {
    memset(dist, 0x3f, sizeof(dist));
    memset(used, 0, sizeof(used));
    for (int i = 0; i < n; ++i)
        pre[i] = i; //初始时，每个顶点的前驱为自身
    dist[source] = 0; //源点到源点的最段距离为0
    pqu.push(make_pair(0, 1)); //将源点到源点的路径放入队列
    while (pqu.size()) {
        // 取出堆顶，x为未访问的顶点中路径最短的顶点
        int x = pqu.top().second;
        pqu.pop();
        // 如果x已经被标记了，cotinue
        if (used[x])
            continue;
        // 标记x
        used[x] = true;
        // 遍历x的所有出边，出边编号：y，边权：z
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            int z = edge[i];
            if (dist[y] > dist[x] + z) {
                dist[y] = dist[x] + z;
                pre[y] = x;
                //堆里面的是更新后的-dist[y]与y
                pqu.push(make_pair(-dist[y], y));
            }
        }
    }
}

void route(int source, int vert) {
    if (vert == source) {
        printf("%d ", source);
        return ;
    }
    route(source, pre[vert]);
    printf("%d ", vert);
}

int main() {
    freopen("in.txt", "r", stdin);
```

```
    freopen("out.txt", "w", stdout);
    scanf("%d", &n);
    scanf("%d", &m);
    source = 1; //源点为1
    for (int i = 1; i <= m; ++i) {
        int x, y, z; //顶点，顶点，边权
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
    }
    dijkstra();
    for (int i = 1; i <= n; ++i)
        printf("%d\n", dist[i]);
    // route(1, 6);
    return 0;
}
```

7.11 02-拓扑排序

```
#include <cstdio>
#include <queue>
using namespace std;
const int maxn = 1e5 + 5;
int n, m;
int head[maxn], vert[maxn], nxet[maxn], edge[maxn], tot;
int deg[maxn];
int sequ[maxn], cnt;

void add(int x, int y) {
    vert[++tot] = y;
    nxet[tot] = head[x];
    head[x] = tot;
    ++deg[y];
}

void topSort() {
    queue<int> quu;
    for (int i = 0; i < n; ++i)
        if (!deg[i])
            quu.push(i);
    while (!quu.empty()) {
        int x = quu.front();
        quu.pop();
        sequ[cnt++] = x;
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            if (!--deg[y])
                quu.push(y);
        }
    }
}

```

```
int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        add(x, y);
    }
    topSort();
    for (int i = 0; i < cnt; ++i)
        printf("%d ", sequ[i]);
    cnt == n ? printf("无环\n") : printf("无环\n");
    return 0;
}
```

7.12 03-BFS-链式前向星

```
#include <cstdio>
#include <cstring>
#include <queue>
#include <vector>
using namespace std;
const int maxn = 1e4; //顶点比1e3少的话，就可以使用邻接矩阵
int n, m; //顶点的个数
int vert[maxn], edge[maxn], nxet[maxn], head[maxn], tot;
bool used[maxn];

//加入有向边(x, y)，权值为z
void add(int x, int y, int z) {
    vert[++tot] = y;
    edge[tot] = z;
    nxet[tot] = head[x];
    head[x] = tot;
}

void BFS() {
    queue<int> quu;
    quu.push(0);
    used[0] = true;
    while (!quu.empty()) {
        int x = quu.front();
        printf("%d\n", x);
        quu.pop();
        for (int i = head[x]; i; i = nxet[i]) {
            int y = vert[i];
            if (used[y]) continue;
            quu.push(y);
            used[y] = true;
        }
    }
}
```

```

int main() {
    freopen("in.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    BFS();
    return 0;
}
```

7.13 03-DFS-链式前向星

```
/*--------------------------------------------------------------------------
 *
 * 文件名称: 02—图的深度优先搜索.cpp
 * 创建日期: 2021年04月14日 ———— 19时30分
 * 题 目: 算法竞赛
 * 算 法: 图论, 链式前向星
 * 描 述: <++>
 *
 --------------------------------------------------------------------------*/

#include <cstdio>
const int maxn = 1e5 + 5;
int n, m;
int used[maxn];
int head[maxn], edge[maxn], nxet[maxn], vert[maxn];
int cnt, tot;

//加入有向边(x, y), 权值为z
void add(int x, int y, int z) {
    vert[++tot] = y;
    edge[tot] = z;
    nxet[tot] = head[x];
    head[x] = tot;
}

void DFS(int x) {
    used[x] = true;
    for (int i = head[x]; i; i = nxet[i]) {
        int y = vert[i];
        if (used[y]) continue;
        printf("%d\n", y);
        DFS(y);
    }
}

```

```
int main() {
    freopen("in/in-02.txt", "r", stdin);
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; ++i) {
        int x, y, z;
        scanf("%d %d %d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    for (int i = 0; i < n; ++i)
        if (!used[i]) {
            ++cnt;
            DFS(i);
        }
    return 0;
}
```

7.14 Bellman-Ford

```
#include <cstdio>
#include <vector>
#include <cstring>
using namespace std;
const int maxn = 1000;
int n, m;
/*目标点、到目标点的距离*/
struct Info {int aim, dis;};
vector<Info> g[maxn];
int dist[maxn];
int source = 0;

bool Bellman() {
    memset(dist, 0x3f, sizeof(dist));
    dist[source] = 0;
    /*Bellman算法的松弛操作不会超过n-1轮*/
    for (int i = 0; i < n-1; ++i)
        /*对每条边进行松弛操作n-1次*/
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < (int)g[j].size(); ++k) {
                int vert = g[j][k].aim;
                int edge = g[j][k].dis;
                if (dist[j] + edge < dist[vert])
                    dist[vert] = dist[j] + edge;
            }
    /*再松弛最后一次，如果还能被松弛，说明有负环*/
    for (int j = 0; j < n; ++j)
        for (int k = 0; k < (int)g[j].size(); ++k) {
            int vert = g[j][k].aim;
            int edge = g[j][k].dis;
            if (dist[j] + edge < dist[vert])
                return false;
```

```
33          }
34      return true;
35  }
36
37  int main() {
38      scanf("%d %d", &n, &m);
39      for (int i = 0; i < m; ++i) {
40          int x, y, z;
41          scanf("%d %d %d", &x, &y, &z);
42          Info info;
43          info.aim = y;
44          info.dis = z;
45          g[x].push_back(info);
46      }
47      //输出0表示有负环，反之没有
48      printf("%d\n", Bellman());
49      return 0;
50  }
```

8 08-杂项

8.1 mt19937

```
1  #include <cstdio>
2  #include <ctime>
3  #include <random>
4  using namespace std;
5  int main() {
6      //std::mt19937 myrand(seed) ，seed可不填，不填seed则会使用默认随机种子
7      mt19937 myrand(time(0));
8      printf("%ld\n", myrand());
9      return 0;
10 }
```

8.2 shuffle

```
1  #include <cstdio>
2  #include <random>
3  #include <algorithm>
4  using namespace std;
5  int arr[100];
6
7  int main() {
8      mt19937 myrand(time(0)); //默认随机种子是rand()，这里使用了time(0)
9      int n = myrand() % 10 + 10; //n是(10—19)之间的随机数
10
11     for (int i = 0; i < n; ++i)
12         arr[i] = i;
13     printf("n = %d\n", n);
```

```
14
15      shuffle(arr+3, arr+9, myrand); //将数组(arr[3] – arr[9])之间的数随机打乱
16      for (int i = 0; i < n; ++i) {
17          if (i == 3 || i == 9)
18              printf("| ");
19          printf("%d ", arr[i]);
20      }
21      printf("\n");
22  }
```

8.3 约瑟夫环

```
1   #include <cstdio>
2
3   int Joseph(int n,int m) {
4       int J = 0;
5       for(int i = 2; i <= n; i++)
6           J = (J + m) % i;
7       /*树组中的下标*/
8       return J;
9   }
10
11  int main() {
12      char car[12] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'};
13      printf("%c\n", car[Joseph(11, 3)]);
14      return 0;
15  }
```