# 参考模版

**Mitake Moca**



# 顺序表

## 约瑟夫问题

```
/*
    n 人从 1 到 n 编号围成一圈，从第 j 个人开始，依次报数，每次报到 k 的人出圈，
问：出圈顺序 / 最后一个剩下的人
    输入：n j k
    输出：n 个人的出圈顺序

    示例输入：20 1 2
    示例输出：2 4 6 8 10 12 14 16 18 20 3 7 11 15 19 5 13 1 17 9
*/
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

typedef struct node  node;
typedef struct node* nptr;

struct node {
    int id;
    nptr prev;
    nptr next;
};

nptr head, tail;
int size;

nptr newnode() {return (nptr)malloc(sizeof(node));}
nptr getnode(int id) {
    nptr p = newnode();
    p -> id = id;
    p -> next = p -> prev = NULL;
    return p;
}

int main() {
    int n = read(), j = read(), k = read();
    // 创建链表
    for(int i = 1;i <= n;i++) {
        nptr p = getnode(i);

        if(head == NULL)
            head = tail = p;
        else
            tail -> next = p, p -> prev = tail, tail = p;
    }

    // 闭环
    tail -> next = head;
    head -> prev = tail;
    size = n;

    // 找到第一个开始报数的人
    for(int i = 1; i < j ;i++) {
        head = head -> next;
    }
```

```
    // 模拟出链的操作
    nptr now = head;
    while(size) {
        // 报数
        for(int i = 1; i < k ;i++)
            now = now -> next;

        // now 应该出圈的人，让 now 出圈
        nptr p = now;
        p -> prev -> next = p -> next;
        p -> next -> prev = p -> prev;
        size--;
        printf("%d ", p -> id);

        // 让 now 重新变成下一次报数开始报 1 的人
        now = now -> next;
    }

    return 0;
}
```

# 栈队

## 计算栈的最小容量

```
/*
    设栈 S 和队列 Q 的初始状态为空，n 个元素依次通过栈 S，一个元素出栈后即进队列 Q
    若 6 个元素出队的序列给定，则栈 S 的容量至少应该是
    输入：第一行一个整数 n 表示元素个数，第二行元素入栈顺序，第三行元素出队顺序
    输出：栈的最小容量

    示例输入：
        6
        a b c d e f
        b d c f e a

    示例输出：
        3

    拓展：
        这道题同样也是以下题目的模板：
            设有一顺序栈 S ，n 个元素依次进栈，如果元素出栈的顺序给定，则栈的容量至少应该是
            即没有队也是这个板子
*/

#include <stdio.h>
```

```c
#include <string.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

char element[105];
char stack[105], queue[105];
int top = -1, front, rear;  // 分别代表栈顶，队头和队尾
int ans = 0;

// 入栈
void push(char ele) {
    stack[++top] = ele;
    if(top >= ans)
        ans = top + 1;
}

// 元素出栈
int pop() {
    return stack[top--];
}

int main() {
    int n = read();
    for(int i = 0; i < n ;i++)
        scanf(" %c", &element[i]);
    for(int i = 0; i < n ;i++)
        scanf(" %c", &queue[i]); // 表示入队顺序
    front = 0, rear = n - 1;

    for(int i = 0; i < n ;i++) {
        push(element[i]);
        // 栈不为空 且 栈顶 = 队头
        while(top > -1 && stack[top] == queue[front])
            pop(), front++;
    }

    printf("%d\n", ans);

    return 0;
}
```

## 判断是否是栈的合法输出序列

```c
/*
    给定一个栈的输入序列，判断某个输出序列是不是栈的合法输出序列
    输入：第一行一个整数 n 表示元素个数，第二行元素入栈顺序，第三行元素出栈顺序
    输出：0 或者 1，1 代表是合法输出序列，0 代表不是合法输出序列
```

```
    示例输入 1：
        5
        a b c d e
        c b a e d
    示例输出 1：
        1

    示例输入 2：
        5
        a b c d e
        d c a b e
    示例输出 2：
        0
*/

#include <stdio.h>
#include <string.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

char element[105];
char stack[105], queue[105];
int top = -1, front, rear;  // 分别代表栈顶，队头和队尾

// 入栈
void push(char ele) {
    stack[++top] = ele;
}

// 元素出栈
int pop() {
    return stack[top--];
}

int main() {
    int n = read();
    for(int i = 0; i < n ;i++)
        scanf(" %c", &element[i]);
    for(int i = 0; i < n ;i++)
        scanf(" %c", &queue[i]); // 表示入队顺序
    front = 0, rear = n - 1;

    for(int i = 0; i < n ;i++) {
        push(element[i]);
        // 栈不为空 且 栈顶 = 队头
        while(top > -1 && stack[top] == queue[front])
            pop(), front++;
```

```
    }

    // 队和栈都为空
    printf("%d\n", top == -1 && rear < front);

    return 0;
}
```

## 中缀表达式转后缀表达式

```
/*
    中缀表达式转后缀表达式
    输入：一行字符串，表示中缀表达式，每个操作数都是字母变量，中间可以有多余的空格
    输出：一行字符串，表示中缀表达式对应的后缀表达式

    示例输入：A-(B+C/D)*E
    示例输出：ABCD/+E*-
*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

char stack[105];
int top = -1;

void push(char value) {
    stack[++top] = value;
}

char pop() {
    return stack[top--];
}

int isOp(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '(' || ch == ')';
}

// 运算符优先级
int pre(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '(':
```

```
            return 0;
        default:
            return -1;
    }
}

char a[105], b[105];

int main() {
    gets(a);
    puts(a);
    int k = 0;
    int len = strlen(a);

    for (int i = 0; i < len; i++) {
        char ch = a[i];

        if (ch == ' ')
            continue;

        if (isalpha(ch)) {
            b[k++] = ch;
        } else if (ch == '(') {
            push(ch);
        } else if (ch == ')') {
            while (top != -1 && stack[top] != '(') {
                b[k++] = pop();
            }
            pop();
        } else if (isOp(ch)) {  // 处理操作符
            while (top != -1 && pre(stack[top]) >= pre(ch))
                b[k++] = pop();

            push(ch);
        }
    }

    // 弹出剩余的操作符
    while (top != -1)
        b[k++] = pop();

    puts(b);
    return 0;
}
```

## 给定入栈出栈顺序求操作串

```
/*
    用 S 表示入栈操作，X 表示出栈操作，给定元素的入栈顺序和出栈顺序，求相应的 S 和 X 的操作串
    输入：第一行一个整数 n 表示元素个数，第二行元素入栈顺序，第三行元素出栈顺序
```

```
    输出：一行只含有 S 和 X 的字符串，表示操作序列

    示例输入：
        4
        ABCD
        ACDB

    示例输出：
        SXSSXSXX
*/

#include <stdio.h>
#include <string.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

char input[105], output[105];
char stack[105];
int top = -1;

void push(char c) {
    stack[++top] = c;
}

char pop() {
    return stack[top--];
}

int main() {
    int n = read();
    getchar();
    gets(input);
    gets(output);

    int j = 0;   // output 下标
    for(int i = 0; i < n ;i++) {
        push(input[i]);
        putchar('S');
        while(top != -1 && stack[top] == output[j])
            putchar('X'), pop(), j++;
    }
    return 0;
}
```

# 后缀表达式转中缀表达式

```c
/*
    后缀表达式转中缀表达式
    输入：一行字符串，表示后缀表达式，每个操作数都是字母变量，中间可以有多余的空格
    输出：一行字符串，表示后缀表达式对应的中缀表达式

    示例输入：ABCD/+E*-
    示例输出：(A-((B+(C/D))*E))
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char stack[105][105];
int top = -1;

void push(char* str) {
    strcpy(stack[++top], str);
}

char* pop() {
    return stack[top--];
}

int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

int main() {
    char postfix[105];
    char infix[105 * 2]; // 预留足够的空间存储中缀表达式
    gets(postfix);

    int len = strlen(postfix);

    for (int i = 0; i < len; i++) {
        char ch = postfix[i];

        if (ch == ' ')
            continue;

        if (isalpha(ch)) {
            char operand[2] = {ch, '\0'};
            push(operand);
        } else if (isOperator(ch)) {
            char operand2[105], operand1[105];
            strcpy(operand2, pop());
```

```
            strcpy(operand1, pop());

            snprintf(infix, sizeof(infix), "(%s%c%s)", operand1, ch,
operand2);
            push(infix);
        }
    }

    printf("%s\n", stack[top]);
    return 0;
}
```

## 前缀表达式转中缀表达式

```
/*
    前缀表达式转中缀表达式
    输入：一行字符串，表示前缀表达式，每个操作数都是字母变量，中间可以有多余的空
格
    输出：一行字符串，表示前缀表达式对应的中缀表达式

    示例输入：-A*+B/CDE
    示例输出：(A-((B+(C/D))*E))
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char stack[105][105];
int top = -1;

void push(char* str) {
    strcpy(stack[++top], str);
}

char* pop() {
    return stack[top--];
}

int isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

int main() {
    char prefix[105];
    char infix[105 * 2]; // 预留足够的空间存储中缀表达式
    gets(prefix);

    int len = strlen(prefix);
```

```
    for (int i = len - 1; i >= 0; i--) {
        char ch = prefix[i];

        if (ch == ' ')
            continue;

        if (isalpha(ch)) {
            char operand[2] = {ch, '\0'};
            push(operand);
        } else if (isOperator(ch)) {
            char operand1[105], operand2[105];
            strcpy(operand1, pop());
            strcpy(operand2, pop());

            snprintf(infix, sizeof(infix), "(%s%c%s)", operand1, ch,
operand2);
            push(infix);
        }
    }

    printf("%s\n", stack[top]);
    return 0;
}
```

# 二叉树

## 二叉查找树查找某元素的比较次数

```
/*
    采用逐点插入法建立某序列的二叉查找树后，查找数据元素 x 共进行多少次元素间的比
较
    输入：第一行一个整数 n，表示二叉查找树的元素个数；第二行 n 个整数，表示用来建
立二叉查找树的元素；
        第三行一个整数 x，表示要被查找的元素
    输出：一个整数，表示查找 x 进行的比较次数

    示例输入：
        10
        54 28 16 34 73 62 95 60 26 43
        62

    示例输出：
        3
*/

#include <stdio.h>
#include <string.h>

int read() {
```

```
    int tem;
    scanf("%d",&tem);
    return tem;
}

int a[10005];   // 表示树
int vis[10005]; // 标记树上有没有结点

void insert(int i) {
    int tem = 0;
    while(vis[tem]) {
        if(i < a[tem])
            tem = tem * 2 + 1;
        else
            tem = tem * 2 + 2;
    }
    a[tem] = i;
    vis[tem] = 1;
}

int main() {
    int n = read();
    for(int i = 0; i < n ;i++)
        insert(read());

    int x = read();
    int tem = 0, cnt = 0;
    while(vis[tem]) {
        cnt++;
        if(x < a[tem])
            tem = tem * 2 + 1;
        else if(x == a[tem])
            break;
        else
            tem = tem * 2 + 2;
    }

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

## 中序后序求先序（中缀后缀求前缀）

**原理：** https://www.bilibili.com/video/BV1Ki4y1x7Lm

```
/*
    给定算数表达式的中缀形式和后缀形式求前缀形式
    输入：两行，第一行是表达式的中缀形式，第二行是表达式的后缀形式
    输出：一行，表达式的前缀形式

    示例输入 1：
```

```
        A+B*C-D/E
        ABC*+DE/-
    示例输出 1:
        -+A*BC/DE

    示例输入 2:
        DBCAFEG
        DCBFGEA
    示例输出 2:
        ABDCEFG
*/

#include <stdio.h>
#include <string.h>
char a[10005], b[10005];

// 查找某个字符在字符串中的位置,返回下标
int search(char x[], char y){
    int i = 1;
    while(x[i] != y) i++;
    return i;
}

//中序数组从 l 到 r, 后序数组从 l2 到 r2
void dfs(int l, int r, int l2, int r2)
{
    if((l > r) || (l2 > r2)) return ;
    printf("%c", b[r2]);//后序数组的最后一位,即为当前的根
    int t = search(a, b[r2]);
    dfs(l, t - 1, l2, l2 + t - 1 - l);
    dfs(t + 1, r, r2 - r + t, r2 - 1);
}

// 输入 中序和后序 -> 前序
// 注意输入的顺序
int main() {
    scanf("%s", a + 1);//读入中序排列
    int n = strlen(a + 1);
    scanf("%s", b + 1);//读入后序排列
    dfs(1, n, 1, n);
    return 0;
}
```

## 中序前序求后序（中缀前缀求后缀）

**原理：** https://www.bilibili.com/video/BV1Ki4y1x7Lm

```
/*
    给定算数表达式的中缀形式和前缀形式求后缀形式
    输入：两行，第一行是表达式的中缀形式，第二行是表达式的前缀形式
    输出：一行，表达式的后缀形式
```

```
    示例输入 1:
        A+B*C-D/E
        -+A*BC/DE
    示例输出 1:
        ABC*+DE/-

    示例输入 2:
        DBCAFEG
        ABDCEFG
    示例输出 2:
        DCBFGEA

*/

#include <stdio.h>
#include <string.h>

//求后序遍历
char a[10005], b[10005], c[10005];
int num;

// 查找 y 在 x 里的位置
int search(char x[], char y) {
    int i = 0;
    while(x[++i] != y);
    return i;
}
void dfs(int l, int r, int l2, int r2) {
    if((l > r) || (l2 > r2)) return ;
    c[num] = b[l2];
    num--;
    int t = search(a, b[l2]);
    dfs(t + 1, r, r2 + t + 1 - r, r2); // 先搜右子树
    dfs(l, t - 1, l2 + 1, l2 + t - l); // 再搜左子树
}

// 输入 中序和前序 -> 后序
// 注意输入的顺序
int main() {
    scanf("%s", a + 1);
    scanf("%s", b + 1);
    num = strlen(a + 1);
    int n = num;
    dfs(1, n, 1, n);
    printf("%s",c + 1);
    return 0;
}
```

# 前序中序后序遍历完全二叉树的结果

**原理：** https://www.bilibili.com/video/BV1RZ4y1n7V2、https://www.bilibili.com/video/BV1gp4y1D76v、https://www.bilibili.com/video/BV1Yg4y1i7CY

```
/*
    已知某完全二叉树采用顺序存储结构，给定结点的存放次序
    求该二叉树的前序序列、中序序列和后序序列
    输入：一个字符串表示二叉树对应的各个节点
    输出：三行，分别代表前序序列、中序序列和后序序列

    示例输入：
        ABCDEFGHIJ

    示例输出：
        ABDHIEJCFG
        HDIBJEAFCG
        HIDJEBFGCA
*/

#include <stdio.h>
#include <string.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

// 数组代替二叉树
char str[10005];
int n;

void preorder(int i) {
    if(i >= n)
        return;
    printf("%c", str[i]);
    preorder(2 * i + 1);
    preorder(2 * i + 2);
}

void midorder(int i) {
    if(i >= n)
        return;
    midorder(2 * i + 1);
    printf("%c", str[i]);
    midorder(2 * i + 2);
}

void postorder(int i) {
    if(i >= n)
```

```
        return;
    postorder(2 * i + 1);
    postorder(2 * i + 2);
    printf("%c", str[i]);
}

int main() {
    gets(str);
    n = strlen(str);

    preorder(0);
    puts("");

    midorder(0);
    puts("");

    postorder(0);
    puts("");

    return 0;
}
```

## 哈夫曼树的带权路径长度

```
/*
    给定叶子结点的权，求哈夫曼树的带权路径长度
    输入：第一行一个整数 n，表示叶子结点的个数；第二行 n 个整数表示各个节点的权
    输出：一个整数表示带权路径长度

    示例输入：
        5
        3 9 6 2 5

    示例输出：
        55
*/

#include <stdio.h>

int read() {
    int tem;
    scanf("%d", &tem);
    return tem;
}

int heap[10005];
int n;

void swap(int *a, int *b) {
    int c = *a;
    *a = *b;
```

```
    *b = c;
}

// 维护一个小顶堆
void adjust(int i, int n) {
    int j = 2 * i + 1, temp = heap[i];

    while (j < n) {
        if (j < n - 1 && heap[j] > heap[j + 1])
            j++;
        if (temp <= heap[j])
            break;
        heap[(j - 1) / 2] = heap[j];
        j = 2 * j + 1;
    }
    heap[(j - 1) / 2] = temp;
}

int pop() {
    int min = heap[0];
    heap[0] = heap[--n];
    adjust(0, n);
    return min;
}

void push(int x, int n) {
    int i = n - 1;
    while (i > 0 && heap[(i - 1) / 2] > x) {
        heap[i] = heap[(i - 1) / 2];
        i = (i - 1) / 2;
    }
    heap[i] = x;
}

int main() {
    n = read();

    for (int i = 0; i < n; i++)
        heap[i] = read();

    // 数组 -> 堆 的初始化操作
    for (int i = (n - 1 - 1) / 2; i >= 0; i--)
        adjust(i, n);

    int ans = 0;

    while (n > 1) {
        int min1 = pop();
        int min2 = pop();
        int sum = min1 + min2;
        ans += sum;
        push(sum, ++n);
```

```
    }

    printf("%d\n", ans);

    return 0;
}
```

# 排序与查找

按照 PPT 上实现的排序和查找等可以去第七次作业的第二题、第四题里找代码。

## 字符串排序-二维数组实现

```
/*
    对给定的字符串进行字典序排序
    输入：第一行一个整数 n，表示字符串的个数；
        后面 n 行，每行一个字符串，表示待排序的字符串
    输出：n 行，表示排序后的字符串

    示例输入：
        5
        orange
        banana
        undefined
        apple
        otter

    示例输出：
        apple
        banana
        orange
        otter
        undefined

*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

char str[1005][1005];
int cmp(const void *a, const void *b) {
    char *x = (char *)a;
```

```
    char *y = (char *)b;
    return strcmp(x, y);
}

int main() {
    int n = read();
    for(int i = 0; i < n ;i++)
        scanf(" %s", str[i]);

    qsort(str, n, sizeof(str[0]), cmp);
    for(int i = 0; i < n ;i++)
        puts(str[i]);
    return 0;
}
```

## 字符串排序-指针实现

```
/*
    对给定的字符串进行字典序排序
    输入：第一行一个整数 n，表示字符串的个数；
        后面 n 行，每行一个字符串，表示待排序的字符串
    输出：n 行，表示排序后的字符串

    示例输入：
        5
        orange
        banana
        undefined
        apple
        otter

    示例输出：
        apple
        banana
        orange
        otter
        undefined

*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

char* str[1005];
```

```
char tem[1005];
int cmp(const void *a, const void *b) {
    char *x = *(char **)a;
    char *y = *(char **)b;
    return strcmp(x, y);
}

int main() {
    int n = read();
    for(int i = 0; i < n ;i++) {
        scanf(" %s", tem);
        str[i] = (char *)malloc(strlen(tem) + 1);
        strcpy(str[i], tem);
    }

    qsort(str, n, sizeof(str[0]), cmp);
    for(int i = 0; i < n ;i++)
        puts(str[i]);
    return 0;
}
```

## 在大顶堆尾部插入元素调整过程的比较次数

**原理：** https://www.bilibili.com/video/BV1Eh4y1j7Y1

```
/*
    已知某序列是大顶堆，在序列尾部插入新元素 x ，将其再调整为大顶堆。输出调整过程中元素之间进行的比较次数
    输入：第一行一个整数 n，表示大顶堆中元素的个数；第二行 n 个整数，表示大顶堆中的元素；
        第三行一个整数 x，表示新插入的元素
    输出：一个整数，表示调整过程中元素之间进行的比较次数

    示例输入：
        5
        25 13 10 12 9
        18

    示例输出：
        2
*/

#include <stdio.h>

int read() {
    int tem;
    scanf("%d", &tem);
    return tem;
}

int heap[10005];
```

```c
void swap(int *a, int *b) {
    int c = *a;
    *a = *b;
    *b = c;
}

int main() {
    int n = read();
    for (int i = 0; i < n; i++)
        heap[i] = read();

    int x = read();
    heap[n] = x;
    n++;

    int ans = 0;
    int i = n - 1;
    // 向上调整大顶堆
    while (i > 0) {
        int parent = (i - 1) / 2;
        ans++;
        if (heap[i] > heap[parent]) {
            swap(&heap[i], &heap[parent]);
            i = parent;
        } else {
            break;
        }
    }

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

## 折半查找某元素的比较下标

```
/*
    将若干数据元素依次存放于一个一维数组中，然后采用折半查找方法查找元素12，输出被比较过的数组元素的下标
    输入：第一行一个整数 n，表示数组中元素的个数；第二行 n 个整数，表示数组中的元素；
        第三行一个整数 x，表示要被查找的元素
    输出：被比较过的数组元素的下标

    示例输入：
        10
        2 4 6 8 10 12 14 16 18 20
        12

    示例输出：
        4 7 5
```

```
*/

#include <stdio.h>
#include <string.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

int a[10005];

int main() {
    int n = read();
    for(int i = 0; i < n ;i++)
        a[i] = read();

    int x = read();
    int l = 0, r = n - 1;
    while(l <= r) {
        int mid = (l + r) >> 1;
        printf("%d ", mid);
        if(a[mid] < x)
            l = mid + 1;
        else if(a[mid] == x)
            break;
        else
            r = mid - 1;
    }
    return 0;
}
```

## 结构体 qsort

```
/*
    结构体排序，先按学生名字字典序排序；当学生名字相同时，按照 id 从小到大排序

    输入：
        第一行一个整数 n，表示学生数目
        接下来 n 行，每行一个整数和一个字符串（不带空格），表示学生 id 和姓名
    输出：
        n 行，每行输出学生 id 和 name

    示例输入：
        5
        1 moca
        2 ran
        3 tsugu
        4 moca
        5 lisi
```

```
    示例输出：
        5 lisi
        1 moca
        4 moca
        2 ran
        3 tsugu
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int readInt() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

typedef struct student student;
struct student {
    char name[15];
    int id;
};

student a[20];

int cmp(const void *a, const void *b) {
    student x = *(student *)a;
    student y = *(student *)b;
    if(strcmp(x.name, y.name))
        return strcmp(x.name, y.name);
    return x.id - y.id;
}

int main() {
    int n = readInt();
    for(int i = 1; i <= n; i++)
        a[i].id = readInt(), scanf("%s", a[i].name);

    qsort(a + 1, n, sizeof(a[0]), cmp);
    for(int i = 1; i <= n; i++)
        printf("%d %s\n", a[i].id, a[i].name);
    return 0;
}
```

# 堆排序中获得初始大顶堆

```
/*
    堆排序中，用关键字序列创建初始大顶堆，得到的初始大顶堆是什么

    输入：
        第一行一个整数 n，表示数字个数
        接下来一行输入 n 个整数，用一个空格分隔，表示排序的 n 个数
    输出：
        一行 n 个整数，用一个空格分隔，表示初始大顶堆

    示例输入：
        7
        6 9 1 5 8 4 7

    示例输出：
        9 8 7 5 6 4 1
*/

#include <stdio.h>
#include <string.h>

int readInt() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

int a[100005];

void swap(int *a, int *b) {
    int c = *a;
    *a = *b;
    *b = c;
}

void adjust(int k[], int i, int n) {
    int j, temp;
    temp = k[i];
    j = 2 * i + 1;
    while(j < n) {
        if(j < n - 1 && k[j] < k[j + 1])
            j++;
        if(temp >= k[j])
            break;
        k[(j - 1) / 2] = k[j];
        j = 2 * j + 1;
    }
    k[(j - 1) / 2] = temp;
}
```

```
int main() {
    int n = readInt();

    for(int i = 0; i < n ;i++)
        a[i] = readInt();

    // 维护堆
    for(int i = n / 2 - 1; i >= 0; i--)
        adjust(a, i, n);

    // 输出初始大顶堆
    for(int i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}
```

# 图

## Prim求最小生成树

**原理：** https://www.bilibili.com/video/BV1CS4y1e7qZ

```
/*
    给出一个无向、无自环、无重边的无向连通图，用 Prim 算法求它的最小生成树
    输入：第一行输入两个数，第一个表示图上点的个数 n （点的编号从 0 到 n - 1），第二个表示图上边的个数 m ；
        第二行一个整数，表示第一个加入最小生成树的点；
        后面 m 行，每行代表一条边，四个数字，分别表示边的编号，两个端点编号，边的权重
    输出：第一行一个权重，表示最小生成树的权重；第二行从 n - 1 个整数，表示选取边的编号，按加入最小生成树的顺序给出

    示例输入：
        6 10
        0
        1 0 1 600
        2 0 2 100
        3 0 3 500
        4 1 2 500
        5 2 3 500
        6 1 4 300
        7 2 4 600
        8 2 5 400
        9 3 5 200
        10 4 5 600

    示例输出：
        1500
```

```
        2 8 9 4 6

*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

const int INF = 0x3f3f3f3f;

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

typedef struct node  node;
typedef struct node* nptr;

struct node {
    int id;
    int v;
    int weight;
    nptr next;
};

int vis[10005];
int dis[10005];
int from[10005];    // 记录每个点分别是由哪条边加入的最小生成树
int ans[10005]; // 记录用来构建最小生成树的边集
nptr head[10005];

nptr newnode() { return (nptr)malloc(sizeof(node)); }
nptr getnode(int id, int v, int weight) {
    nptr p = newnode();
    p -> id = id;
    p -> v = v;
    p -> weight = weight;
    p -> next = NULL;
    return p;
}

nptr addedge(int id, int u, int v, int weight) {
    nptr e1 = getnode(id, v, weight);
    e1 -> next = head[u];
    head[u] = e1;

    nptr e2 = getnode(id, u, weight);
    e2 -> next = head[v];
    head[v] = e2;
}
```

```
int main() {
    int n = read(), m = read(), cnt = 0;    // cnt 表示加入最小生成树边的个数
    int totalWeight = 0;
    int sta = read();

    for(int i = 0; i < n ;i++)
        if(i != sta)
            dis[i] = INF;

    for(int i = 0; i < m ;i++) {
        int id = read(), u = read(), v = read(), weight = read();
        addedge(id, u, v, weight);
        addedge(id, v, u, weight);
    }

    // 每次选择一个距离最小的点加入最小生成树的点集
    for(int i = 0; i < n ;i++) {
        int tem = -1, minDis = INF;    // 表示当前选择的距离最短的点和最短距离
        for(int j = 0; j < n ;j++)
            if(dis[j] < minDis && !vis[j])
                minDis = dis[j], tem = j;
        ans[cnt++] = from[tem];
        vis[tem] = 1;
        totalWeight += minDis;

        for(nptr p = head[tem]; p ; p = p -> next)
            if(!vis[p -> v] && dis[p -> v] > p -> weight)
                dis[p -> v] = p -> weight, from[p -> v] = p -> id;
    }

    printf("%d\n", totalWeight);
    for(int i = 1; i < n ;i++)
        printf("%d ", ans[i]);

    return 0;
}
```

## Kruskal求最小生成树

**原理：** https://www.bilibili.com/video/BV1yt4y1H7F9

```
/*
    给出一个无向、无自环、无重边的无向连通图，用 Kruskal 算法求它的最小生成树
    输入：第一行输入两个数，第一个表示图上点的个数 n （点的编号从 0 到 n - 1），第二个表示图上边的个数 m ；
        后面 m 行，每行代表一条边，四个数字，分别表示边的编号，两个端点编号，边的权重
    输出：第一行一个权重，表示最小生成树的权重；第二行从 n - 1 个整数，表示选取边的编号，按加入最小生成树的顺序给出

    示例输入：
```

```
        6 10
        1 0 1 600
        2 0 2 100
        3 0 3 500
        4 1 2 500
        5 2 3 500
        6 1 4 300
        7 2 4 600
        8 2 5 400
        9 3 5 200
        10 4 5 600

    示例输出：
        1500
        2 9 6 8 4

*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

typedef struct edge  edge;

struct edge {
    int id;
    int u;
    int v;
    int weight;
};

int f[10005];   // 用来维护并查集
edge e[10005];  // 用来维护所有边
int ans[10005]; // 记录 MST 选择的边

// 查
int find(int i) {
    return f[i] = f[i] == i ? i : find(f[i]);
}

// 并
void link(int u, int v) {
    f[find(u)] = find(v);
}

int cmp(const void *a, const void *b) {
```

```
    edge x = *(edge *)a;
    edge y = *(edge *)b;
    return x.weight - y.weight ? x.weight - y.weight : x.id - y.id;
}

int main() {
    int n = read(), m = read();
    for(int i = 0; i < m ;i++)
        e[i].id = read(), e[i].u = read(), e[i].v = read(), e[i].weight =
read();
    qsort(e, m, sizeof(e[0]), cmp);
    int totalWeight = 0, cnt = 0;

    // 并查集初始化
    for(int i = 0; i < n;i++)
        f[i] = i;

    for(int i = 0; i < m ;i++) {
        if(find(e[i].u) != find(e[i].v))
            ans[++cnt] = e[i].id, totalWeight += e[i].weight, link(e[i].u,
e[i].v);
    }

    printf("%d\n", totalWeight);
    for(int i = 1; i < n ;i++)
        printf("%d ", ans[i]);
    return 0;
}
```

## 拓扑排序

**原理：** https://www.bilibili.com/video/BV17g41197sa

```
/*
    给出一个有向无环图，输出它的拓扑序
    输入：第一行输入两个数，第一个表示图上点的个数 n （点的编号从 1 到 n），第二
个表示图上边的个数 m ；
        后面 m 行，每行代表一条边，两个数字，第一个数字代表边的起点，第二个数字代
表边的终点
    输出：一行，表示边的拓扑序

    示例输入：
        6 8
        1 2
        1 4
        2 6
        3 1
        3 4
        4 5
        5 2
        5 6
```

```
    示例输出：
        3 1 4 5 2 6

*/
#include <stdio.h>
#include <string.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

// 但是我们期末图不考编程，只考选填，就那么几个点，我就拿临接矩阵建图了
int e[1005][1005];
int vis[1005];
int ans[1005];  // 点的拓扑序
int de[1005];   // 点的入度
int cnt, n, m;

void topoLogic(int i) {
    if(vis[i])
        return;
    vis[i] = 1;
    for(int j = 1; j <= n ;j++)
        if(e[i][j])
            topoLogic(j);
    ans[++cnt] = i;
}

int main() {
    n = read(), m = read();
    for(int i = 0; i < m ;i++) {
        int u = read(), v = read();
        e[u][v] = 1, de[v]++;
    }

    for(int i = 1; i <= n ;i++)
        if(de[i] == 0)
            topoLogic(i);

    for(int i = n; i >= 1 ;i--)
        printf("%d ", ans[i]);
    return 0;
}
```

# 迪杰斯特拉求单源最短路

**原理：** https://www.bilibili.com/video/BV1Ya411L7gb

```
/*
    给出一个有向无环图，求指定原点到指定终点的最短路
    输入：第一行输入两个数，第一个表示图上点的个数 n （点的编号从 1 到 n），第二个表示图上边的个数 m ；
        第二行输入两个数字，表示起点和终点编号
        后面 m 行，每行代表一条边，三个数字，第一个数字代表边的起点，第二个数字代表边的终点，第三个代表边的权重
    输出：第一行一个整数，代表最短路长度；
        第二行输出从起点到终点经过的所有顶点的编号，若有多种最优方案输出数字字典序最小的一个

    示例输入：
        7 10
        1 7
        1 2 3
        1 3 2
        1 4 3
        2 3 3
        2 5 1
        3 5 3
        6 3 1
        4 6 3
        5 7 1
        6 7 5

    示例输出：
        5
        1 2 5 7

*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int read() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

const int INF = 0x3f3f3f3f;

typedef struct node  node;
typedef struct node* nptr;

struct node {
```

```
    int v;
    int weight;
    nptr next;
};
nptr head[10005];       // 邻接表数组
int from[10005];   // 每个点被谁更新
int dis[10005];     // 每个点的最短距离
int vis[10005];     // 每个点是否被访问
int road[10005];    // 记录从起点到终点的路径

nptr newnode() { return (nptr)malloc(sizeof(node)); }
nptr getnode(int v, int weight) {
    nptr p = newnode();
    p -> v = v;
    p -> weight = weight;
    p -> next = NULL;
    return p;
}

void addedge(int u, int v, int weight) {
    nptr p = getnode(v, weight);
    p -> next = head[u];
    head[u] = p;
}

int main() {
    int n = read(), m = read();
    int sta = read(), end = read();

    for(int i = 0; i < m ;i++) {
        int u = read(), v = read(), weight = read();
        addedge(u, v, weight);  // 单向边
    }

    for(int i = 1; i <= n ;i++)
        if(i != sta)
            dis[i] = INF;

    for(int i = 1; i <= n ;i++) {
        int tem = -1, minDis = INF;
        for(int j = 1; j <= n ;j++)
            if(!vis[j] && dis[j] < minDis)
                minDis = dis[j], tem = j;

        vis[tem] = 1;
        for(nptr p = head[tem]; p ; p = p -> next)
            if(!vis[p -> v] && p -> weight + dis[tem] < dis[p -> v])
                dis[p -> v] = p -> weight + dis[tem], from[p -> v] = tem;
    }

    printf("%d\n", dis[end]);
```

```
    int tem = end, cnt = 0;
    while(tem) {
        road[++cnt] = tem;
        tem = from[tem];
    }

    for(int j = cnt; j > 0 ;j--)
        printf("%d ", road[j]);

    return 0;
}
```

# 文件

## 文件读写 freopen

```
#include <stdio.h>
#include <string.h>

// 文件读写的前提是输入文件存在

int readInt() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

int a[105];

int main() {
    freopen("in.txt", "rb", stdin);
    int n;

    // 假设先从文件中读入 n，再读入 n 个整数
    n = readInt();
    for(int i = 1; i <= n ;i++)
        scanf("%d", &a[i]);

    // 再将 n 个整数输出到文件中
    freopen("out.txt", "wb", stdout);
    for(int i = 1; i <= n ;i++)
        printf("%d\n", a[i]);

    return 0;
}
```

## 文件读写 fscanf

```c
#include <stdio.h>
#include <string.h>

// 文件读写的前提是输入文件存在

int readInt() {
    int tem;
    scanf("%d",&tem);
    return tem;
}

int a[105];

int main() {
    FILE *in = fopen("in.txt", "rb");
    int n;

    // 假设先从文件中读入 n，再读入 n 个整数
    fscanf(in, "%d", &n);
    for(int i = 1; i <= n ;i++)
        fscanf(in, "%d", &a[i]);

    // 再将 n 个整数输出到文件中
    FILE *out = fopen("out.txt", "wb");
    for(int i = 1; i <= n ;i++)
        fprintf(out, "%d\n", a[i]);

    return 0;
}
```