Notes

周赛 6 T1

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

const int N = 1003;

bool g[N][N];
bool vis[N];

void dfs(int x, int n) {
    vis[x] = true;
    cout << x << ' ';
    for (int i = 1 ; i <= n ; ++i)
        if (g[x][i] && !vis[i])
            dfs(i, n);
}

int main() {
    int n, x;
    cin >> n >> x;
    for (int i = 1 ; i <= n ; ++i)
        for (int j = 1 ; j <= n ; ++j)
            cin >> g[i][j];
    dfs(x, n);
    for (int i = 1 ; i <= n ; ++i)
        if (!vis[i]) dfs(i, n);
    return 0;
}
```

邻接矩阵: 一个 $n \times n$ 的矩阵，$g[i][j]$ 表示 i 和 j 之间是否有边。

邻接表: 每个点，维护它到哪些边有点 (应用场景: 图点数多，但边没有全满)

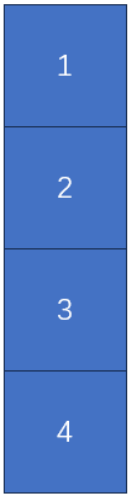
- vector 存储，最简单
- 链表(链式前向星)：每个点维护一个链表一样的结构，减少分配内存的开销。

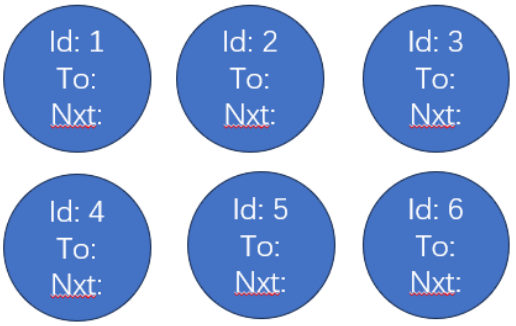
```cpp
struct node {
      int to;  // 指向的点
    int nxt; // 下一个链表节点的编号
};
const int N = 1e5 + 3; // 点数
const int M = 5e5 + 3; // 边数

int head[N]; // 每个点的第一个链表节点编号
node e[M];     // 每个链表节点表示一条有向边
int tot;     // 链表分配节点的总数

void adde(int x, int y) {
    e[++tot] = node {x, head[x] };
    head[x] = tot;
```

```
}
```

	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	1
4	1	1	1	0

1	2	4	
2	1	3	4
3	2	4	
4	1	2	3





dfs 要点:

- 递归实现，所以要注意递归边界 —— 需要设置 vis 标记
- 题目要求是最小字典序，所以先遍历编号最小的
- 特别地，有些节点可能由 x 无法到达，所以必须要额外遍历所有节点，以所有节点为 root 开始遍历。

（当然，bfs 也不是不行）

周赛 6 T2

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```cpp
using namespace std;

const int N = 1e6 + 3;

vector <int> a[N];
bool vis[N];

void dfs(int x) {
    vis[x] = true;
    for (int v : a[x])
        if (!vis[v]) dfs(v);
}

int main() {
    int n, m;
    cin >> n >> m;
    while (m--) {
        int x, y;
        cin >> x >> y;
        a[x].push_back(y);
        a[y].push_back(x);
    }
    dfs(1);
    for (int i = 2 ; i <= n ; ++i)
        if (vis[i]) cout << i << ' ';
    return 0;
}
```

答案有错！！

用邻接表，链式前向星方法演示一遍，应该会错一个点。

正好可以证明为什么这里不能用邻接矩阵存储。

- 无向图，要双向加边。
- 有向图，单向即可。

周赛 6 T3

```cpp
#include <iostream>
#include <queue>
using namespace std;

struct point { int x, y; };

const int N = 4007;
const int dir[2][4] = {
    { 0, 1 , 0 , -1 },
    { 1, 0 , -1, 0  }
};

char g[N][N];
bool sink[N][N];
bool vis[N][N];
```

```
bool bfs(point input, int n) {
    queue <point> q;
    q.push(input);
    bool ground = false;
    vis[input.x][input.y] = true;
    while (!q.empty()) {
        point p = q.front(); q.pop();
        int x = p.x, y = p.y;
        if (!sink[x][y]) ground = true;
        for (int i = 0 ; i < 4 ; ++i) {
            int tx = x + dir[0][i];
            int ty = y + dir[1][i];
            if (tx < 0 || tx >= n || ty < 0 || ty >= n)
                continue;
            if (g[tx][ty] == '#') {
                if (!vis[tx][ty]) {
                    vis[tx][ty] = true;
                    q.push({tx, ty});
                }
            }
        }
    }
    // cout << '\n';
    return ground;
}


signed main() {
    int n;
    cin >> n;

    for (int i = 0 ; i < n ; ++i)
        for (int j = 0 ; j < n ; ++j)
            cin >> g[i][j];

    for (int i = 0 ; i < n ; ++i)
        for (int j = 0 ; j < n ; ++j)
            if (g[i][j] == '#') {
                for (int t = 0 ; t < 4 ; ++t) {
                    int tx = i + dir[0][t];
                    int ty = j + dir[1][t];

                    if (g[tx][ty] != '#')
                        sink[i][j] = true;
                }
            }

    int ans = 0;
    for (int i = 0 ; i < n ; ++i)
        for (int j = 0 ; j < n ; ++j)
            if (!vis[i][j] && g[i][j] == '#')
                if (!bfs(point {i,j}, n))
                    ++ans;

    cout << ans;
    return 0;
```

```
}
```

代码量不小，带着写。

注意，题目输入有不满足范围的。 数据有错，答案也是错的。即使按照错误的思路来做。

分为两步（当然可以合并在一起写）

- 判断是否会沉。
- bfs 找岛屿，然后判断是否有不会沉的点。

介绍 bfs 框架: 队列实现，同样需要标记 visit 防止无限循环。

dir 数组: 快速遍历四个方向。常用技巧。

月赛 2 T1

```
#include <iostream>
using namespace std;

const int N = 10003;
int a[N];

int main(){
    int n;
    cin >> n;
    for (int i = 1 ; i <= n ; ++i)
        cin >> a[i];
    int x;
    cin >> x;
    for (int i = 1 ; i <= n ; ++i) {
        if (x == a[i]) {
            cout << i;
            return 0;
        }
    }
    cout << "not find";
    return 0;
}
```

简单的循环查找。没有难度。

如果无聊的话，自然可以用二分查找。

标准库也行，需要 `#include <algorithm>` ，用 `std::lower_bound()` 函数即可。可以演示。（返回的是指针）

总之很简单的一道题。

月赛 2 T2

```
#include <iostream>
#include <algorithm>
using namespace std;
```

```
const int N = 1e6 + 3;
int a[N];

signed main() {
	int n,m ;
	cin >> n >> m;
	for (int i = 1 ; i <= n ; ++i)
		cin >> a[i];
	while (m--) {
		int x;
		cin >> x;
		int m = lower_bound(a + 1, a + n + 1, x) - a;
		if (a[m] != x) {
			cout << -1 << ' ';
		} else {
			cout << m << ' ';
		}
	}
	return 0;
}
```

真正的二分查找，使用了 lower_bound 函数。这就是一个基础版本，没啥好看的。

或许可以手写一下二分。

月赛 2 T3

二分答案板子题。

```
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 1e5 + 3;
int x[N];

bool check(int d, int n, int c) {
    // cout << d << ' ' << --c << '\n';
    --c;
    int last = x[1];
    for (int i = 2 ; i <= n ; ++i) {
        if (x[i] - last >= d) {
            last = x[i];
            if (--c == 0) {
                // cout << d << '\n';
                return true;
            }
        }
    }
    // cout << '\n';
    return false;
}

int main(){
    int n, c;
```

```
    cin >> n >> c;
    for (int i = 1 ; i <= n ; ++i)
        cin >> x[i];
    sort(x + 1, x + 1 + n);
    int l = 1, r = x[n] - x[1];
    while (l != r) {
        int mid = (l + r) >> 1;
        if (check(mid,n,c)) l = mid + 1;
        else                r = mid;
    }
    cout << l - 1;
    return 0;
}
```

注意题目没说数据保证排好序了。要提前排序。

然后二分答案流程，重点是 check 函数。

二分答案要点:

- 在答案相关的领域上 二分
- check 函数，验证一个答案是否合法，需比较高效(贪心/半暴力)

如果还有时间

拓展: 做实验

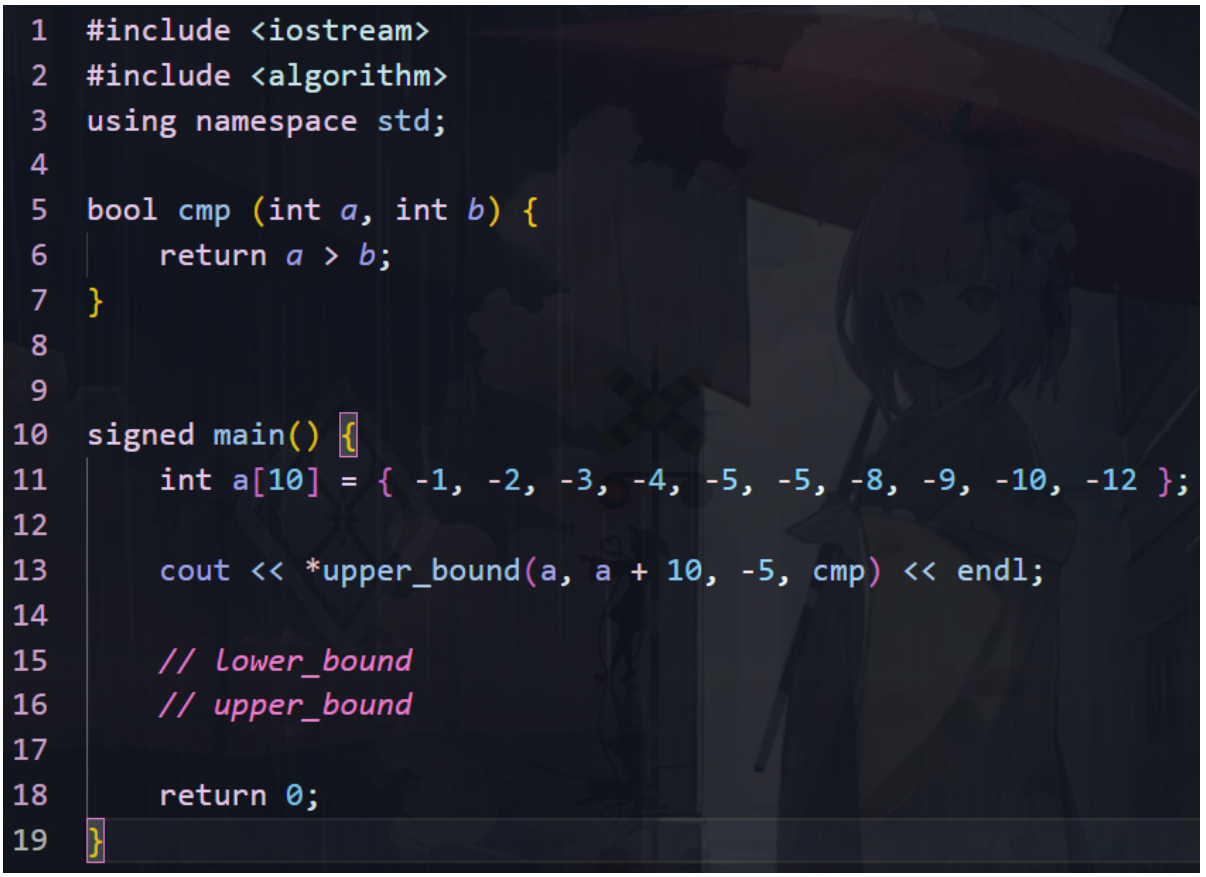
lower_bound 第一个大于等于 x

upper_bound 第一个严格大于 x

binary_search 是否存在

用 STL 重写二分答案。

```
#include <iostream>
#include <algorithm>

bool cmp(int middle, int input) {
    return middle > input;
}

signed main() {
    int a[10] = {-1, -2, -3, -4, -5, -6, -7, -8, -9, -10};
    int *p = std::lower_bound(a, a + 10, 5 /*input*/, cmp);
    if (p == a + 10) {
        std::cout << "Not found" << std::endl;
    } else {
        std::cout << "Found at " << p - a << std::endl;
    }
    return 0;
}
```



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

bool cmp (int a, int b) {
    return a > b;
}


signed main() {
    int a[10] = { -1, -2, -3, -4, -5, -5, -8, -9, -10, -12 };

    cout << *upper_bound(a, a + 10, -5, cmp) << endl;

    // lower_bound
    // upper_bound

    return 0;
}
```

二分通用方法:

```cpp
// 搜索范围为 [l, r)
while (l != r) {
    int mid = l + ((r - l) >> 1);
       if (check(mid)) l = mid + 1;
    else            r = mid;
}
// l == r 是第一个不满足 check 的解
// 最后一个满足 check 的解是 l - 1
```