一、简答

1.

邻接矩阵
优点： 速度快，可直接获得两点之间的关系
缺点： 空间复杂度为 $O(n^2)$，不适合点多边少的稀疏图
邻接表
优点： 只记录了邻接点的信息，空间复杂度为 $O(m)$， m 为边的数目
缺点： 使用 STL 容器，极端情况可能会较慢
链式前向星
优点： 空间度复杂度和邻接表相当，使用多个数组模拟链表，性能较好

2. （1）将所有边按照权值由小到大排序；
   （2）按顺序考虑每条边，只要当前边和已选择的边不够成环，则选择它；
   （3）选择 $(n-1)$ 条边之后形成最小生成树，如果无法选出$(n-1)$ 条边则原图不连通。

二、作业H5

C.公路修建

题目大意

有 n 个节点，要连 m 条边，每连一条边，输出当前的连通块数 - 1.

解法

并查集维护连通块，判断每次连接是否连接两个不同的连通块。

时间复杂度

$O(n+m)$

代码

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

int fa[100010];

int Find(int x) {
    if (x == fa[x]) return x;
    return fa[x] = Find(fa[x]);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    for (int i = 1; i <= n; i++) fa[i] = i;

    int a, b, Cnt = 1;
    for (int i = 1; i <= m; i++) {
        cin >> a >> b;
        int f1 = Find(a), f2 = Find(b);
        if (f1 != f2) {
            Cnt++;
            fa[f1] = f2;
        }
        cout << n - Cnt << '\n';
    }
    return 0;
}
```

E.水渠设计

题目大意

现在有 n 个田地需要灌溉。
可以选择修建 m 个引水渠，第 i 条从第 a 个田地到第 b 个田地，花费 c 元。
现在可以买任意多个抽水机，买一个抽水机需要花费 p 元。如果在一个田地旁边安置一个抽水机，则该田地会被灌溉。
水可以顺着水渠流动。
求让每一块田地都能被灌溉的最小花费。

解法

图重构，加一个超级源点，并向 n 个点连权为 p 的边，然后对这 $n+1$ 个点跑最小生成树即可

时间复杂度

$O((n+m)*log(m))$

代码

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 1e5 + 10;

struct edge
{
    int l, x, y;
    edge(int l, int x, int y):l(l), x(x), y(y) {}
    bool operator < (const edge &a) const {
        return l > a.l;
    }
};

int fa[N];
priority_queue<edge> Q;

int Find(int x) {
    if (x == fa[x]) return x;
    return fa[x] = Find(fa[x]);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(0);

    int n, m, p;
    cin >> n >> m >> p;

    for (int i = 1; i <= n + 1; i++) fa[i] = i;

    int a, b, c;
    for (int i = 1; i <= m; i++) {
        cin >> a >> b >> c;
        Q.push({c, a, b});
    }

    // 源点
    for (int i = 1; i <= n; i++) {
        Q.push({p, i, n + 1});
    }

    ll sum = 0;
    int Cnt = 1;
    while (!Q.empty()) {
        edge top = Q.top();
        Q.pop();
        int fx = Find(top.x), fy = Find(top.y);
        if (fx != fy) {
```

```
            fa[fx] = fy;
            sum += top.l;
            Cnt++;
            if (Cnt == n + 1) break;
        }
    }

    cout << sum << '\n';
    return 0;
}
```