# 问题介绍

动态连通性，可以抽象为对一幅图连线。 Union-Find算法主要实现了以下的API:

```
class UF{
  public:
    // p和q相连
    void union(int p, int q);
    // p和q的联通
    bool isConnect(int p, int  q);
    // 连通分量的个数
    int count();

  private:
    int count; // 连通分量
    std::vector<int> parent; // 节点x的父节点为parent[x]
};
```

连通的性质:

1. 自反性： 节点p与其自身连通；
2. 对称性： 节点p与q连通，那么节点q和p连通；
3. 传递性： 节点p与q连通，q与r连通，那么节点p与r连通。

简单union find算法:

```
#include <vector>

class UF {
public:
  UF(int n) {
    count  = n;  // 初始n个节点互不连通
    parent = std::vector<int>(n, 0);

    for (int i = 0; i < n; ++i) {
      parent[i] = i;
    }
  }

  // 连通两个节点
  void Union(int p, int q) {
    int rootP = Find(p);
    int rootQ = Find(q);

    if (rootQ == rootP) {
      return;
    }
```

```
    parent[rootP] = rootQ;

    count--;  // 连通分量个数减一
  }

  // 查找当前节点的parent
  int Find(int x) {
    while (x != parent[x]) {
      x = parent[x];
    }

    return x;
  }

  int Count() const {
    // 返回连通分量个数
    return count;
  }

  bool isConnect(int p, int q) {
    int rootP = Find(p);
    int rootQ = Find(q);

    return rootP == rootQ;
  }

private:
  int              count;   // 连通分量个数
  std::vector<int> parent;  // 节点x的父节点为parent[x]
};
```

## 优化树的平衡性

主要是在连接时将小树接到大树上，避免不均衡。

```
#include <vector>

class UF {
public:
  UF(int n) {
    count  = n;  // 初始n个节点互不连通
    parent = std::vector<int>(n, 0);

    for (int i = 0; i < n; ++i) {
      parent[i] = i;
      weight[i] = 1;
    }
  }

  // 连通两个节点
```

```
  void Union(int p, int q) {
    int rootP = Find(p);
    int rootQ = Find(q);

    if (rootQ == rootP) {
      return;
    }

    if (weight[rootP] >= weight[rootQ]) {
      parent[rootP] = rootQ;
      weight[rootP] += weight[rootQ];
    } else {
      parent[rootQ] = rootP;
      weight[rootQ] += weight[rootP;]
    }

    count--;  // 连通分量个数减一
  }

  // 查找当前节点的parent
  int Find(int x) {
    while (x != parent[x]) {
      x = parent[x];
    }

    return x;
  }

  int Count() const {
    // 返回连通分量个数
    return count;
  }

  bool isConnect(int p, int q) {
    int rootP = Find(p);
    int rootQ = Find(q);

    return rootP == rootQ;
  }

private:
  int              count;   // 连通分量个数
  std::vector<int> parent;  // 节点x的父节点为parent[x]
  std::vector<int> weight;  // 树的重量
};
```

## 路径压缩

```
#include <vector>
```

```
class UF {
public:
  UF(int n) {
    count  = n;  // 初始n个节点互不连通
    parent = std::vector<int>(n, 0);

    for (int i = 0; i < n; ++i) {
      parent[i] = i;
      weight[i] = 1;
    }
  }

  // 连通两个节点
  void Union(int p, int q) {
    int rootP = Find(p);
    int rootQ = Find(q);

    if (rootQ == rootP) {
      return;
    }

    if (weight[rootP] >= weight[rootQ]) {
      parent[rootP] = rootQ;
      weight[rootP] += weight[rootQ];
    } else {
      parent[rootQ] = rootP;
      weight[rootQ] += weight[rootP];
    }

    count--;  // 连通分量个数减一
  }

  // 查找当前节点的parent
  int Find(int x) {
    while (x != parent[x]) {
      parent[x] = parent[parent[x]]; // 压缩路径
      x         = parent[x];
    }

    return x;
  }

  int Count() const {
    // 返回连通分量个数
    return count;
  }

  bool isConnect(int p, int q) {
    int rootP = Find(p);
    int rootQ = Find(q);

    return rootP == rootQ;
  }
```

```
private:
  int               count;   // 连通分量个数
  std::vector<int> parent;  // 节点x的父节点为parent[x]
  std::vector<int> weight;  // 树的重量
};
```