2023算法期末

1 简答

1. 分析快速排序的最坏复杂度
2. 结合红黑树的定义分析搜索时间是$O(logn)$
3. 有向图的收缩图是否存在环?
4. 分析无向 dfs 是否有 DE，DE 有什么性质
5. 一个 NPC 问题 A 找到了多项式时间内的算法，证明 P=NP

2 寻找任意两个点间最大有效概率

图上每条边有有效概率$0 < p(e) < 1$, 一条路径的有效概率为其上面所有边的有效概率之积，

$O(n^3)$ 求任意点对的路径的最大有效概率，证明复杂度，并归纳证明算法的正确性

3 证明 KNI 问题是 NP 和 NPC

KNI 问题：一组骑士在圆桌上安排座位。把每个人看成图上一个节点，一条边代表两个人之间互相仇恨。仇恨的两个人不能坐在一起。问能不能找到一个合法的座位安排?

UHC 问题：一个无向图是否存在 哈密尔顿回路?

1. 证明 KNI 问题为 NP 的
2. 证明 KNI 问题为 NPC 的

4 网格上的最长对角线

n×n 的正方形网格，黑格子不能选，求白格子连成的最长对角线（左上到右下，左下到右上），需要一个 $O(n^2)$ 的算法。

5 删除最少的边让图不连通

无向带权图 G，G_k 为 G 去掉所有权重大于等于 k 的图

$O(|E|log|E|)$ 找到最大的 k 使得 G_k 不连通

6 逐步给出一个线性时间复杂度内判断 G 是否为半联通图

半联通图：有向图上，任意两个节点 u，v 之间，要么 u 到 v 有边，要么 v 到 u 有边，则称为半

联通的

1. 考虑有向无环图 G，证明 G 是半联通的 iff 将 G 拓扑排序后（所有的边从拓扑序号小的指向大的），G 的拓扑序满足 $i < j$ 的两个节点，一定有一条路径从拓扑序号 i 的节点到拓扑序号为 j 的节点。
2. 考虑任意有向图 G，设计一个时间复杂度线性（$O(|V| + |E|)$）的算法，判断其是否为半联通的

Solution

1

1. 最不均匀的$Partition$，顺序或倒序。
2. 红黑树的黑色节点的子节点只能是红色，红色节点的子节点可以是红色或黑色，这种红黑相间的限制，使得$n \geq 4^h$, 故$h \leq \log(n)$，使得深度限制在$\log(n)$内，而查找最坏查找h次.
3. 不存在，反证法，否则可能进一步收缩。
4. 不存在，假设存在边e为DE，设e的两端点为u, v, 则u遍历到v时，v会先遍历该边，故e为BE，u再遍历时属于二次遍历。
5. 根据定义证明就行，$NP \subseteq P$, 且$P \subseteq NP$, 故$NP = P$。

2

修改$Floyd$里的关键代码$f[i][j]=max(f[i][j],f[i][k]+f[k][j])$

3

第一问根据构造的结果判断相邻两点间是否有边即可，时间复杂度$O(nm)=O(n^3)$, 为多项式时间，故为NP

第二问将哈密尔顿问题的补图作为KNI问题的输入即可，归约是多项式时间的，再分别证明归约的$true$和$false$对于两者都成立，最后结合NP即可。

4

状态设计：$f[i][j][k]$

$k=0$时代表从$(i，j)$开始从左上到右下的最长对角线, $k=1$时表示从(i,j)开始从右上到左下的最长对角线。

状态转移方程：

$$f[i][j][0]=\begin{cases}0 & A[i][j]=0\\ f[i+1][j+1]+1 & A[i][j]==1\end{cases}$$

$$f[i][j][1]=\begin{cases}0 & A[i][j]=0\\ f[i+1][j-1]+1 & A[i][j]==1\end{cases}$$

初始化：$f[i][j][0]=f[i][j][k]=A[i][j]$

最终遍历所有$f[i][j][0],f[i][j][1]$，取最大即可。

5

类似$kruskal$算法，找到MST中最大的边即可。

```
#include <algorithm>
#include <cstring>
#include <iostream>
using namespace std;
const int Size = 1e6 + 10;
int fa[Size], sz[Size], res = -1;
struct
{
    int u, v, w;
} a[Size];
int get(int x)
{
    return fa[x] == x ? fa[x] : fa[x] = get(fa[x]);
}
void merge(int x, int y)
{
    x = get(x), y = get(y);
    if (x != y)
    {
        if (sz[x] < sz[y])
            sz[y] += sz[x], fa[x] = y;
        else
            sz[x] += sz[y], fa[y] = x;
    }
}
int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
        cin >> a[i].u >> a[i].v >> a[i].w;
    sort(a + 1, a + m + 1, [](auto a, auto b){ return a.w < b.w; });
    for (int i = 1; i <= n; i++)
        fa[i] = i, sz[i] = 1;
    for (int i = 1; i <= m; i++)
    {
        auto [u, v, w] = a[i];
        merge(u, v);
```

```
        if (sz[get(u)] == n)
        {
            res = w;
            break;
        }
    }
    cout << res << endl;
}
```

6

第一问

正推：假设存在$u, v, u.topo < v.topo$，且不存在从u到v的道路，由于该图为半联通，所以一定存在v到u的路径，设该路径为$v, w_1, w_2 \ldots, w_k, u$, 由于拓扑排序规则，必有$v.topo < w_1.topo < w_2.topo < \ldots < u.topo$，矛盾。

逆推：由于题设条件，所以对任何一对节点u, v, 不妨另$u.topo < v.topo$, 则有一条从u到v的路径，对于任意的u，v都成立，故为半连通图。

第二问

先求出SCC，获得压缩图G, 若是半联通，则$i->i+1$有路径, 根据拓扑排序规则，拓扑序为i的节点必定有一个子节点的拓扑序是$i+1$, 从1开始拓扑，在拓扑过程中判断是否有某个节点拓扑序为$i+1$。此外若拓扑一次后有白色节点，则一定不是半联通。

```
//代码里省去SCC步骤
#include <iostream>
using namespace std;
const int Size = 1e6 + 10;
int ver[Size], Next[Size], head[Size], st[Size], Rank[Size], tot, num, n, m;
void add(int x, int y)
{
    ver[++tot] = y, Next[tot] = head[x], head[x] = tot;
}
bool topo(int x)
{
    st[x] = 1;
    int MinRank = 1e9;
    for (int i = head[x]; i; i = Next[i])
    {
        int y = ver[i];
        if (st[y])
            MinRank = min(MinRank, Rank[y]);
        else if (!topo(y))
            return false;
        else
            MinRank = min(MinRank, Rank[y]);
    }
    Rank[x] = num--;
    if (Rank[x] != n && Rank[x] + 1 != MinRank)
        return false;
    return true;
}
int main()
{
    cin >> n >> m;
    num = n;
    for (int i = 1; i <= m; i++)
    {
        int x, y;
        cin >> x >> y;
        add(x, y), add(y, x);
    }
    if (!topo(1))
        cout << "False" << endl;
    else
    {
        int tag = 0;
        for (int i = 1; i <= n; i++)
            if (!st[i])
```

```
                tag = 1;
        if (tag)
            cout << "False" << endl;
        else
            cout << "True" << endl;
    }
    return 0;
}
```