「あの鹿に触らないで !!」

图论

Graph Theory

御薬袋托托（@bufhdy）

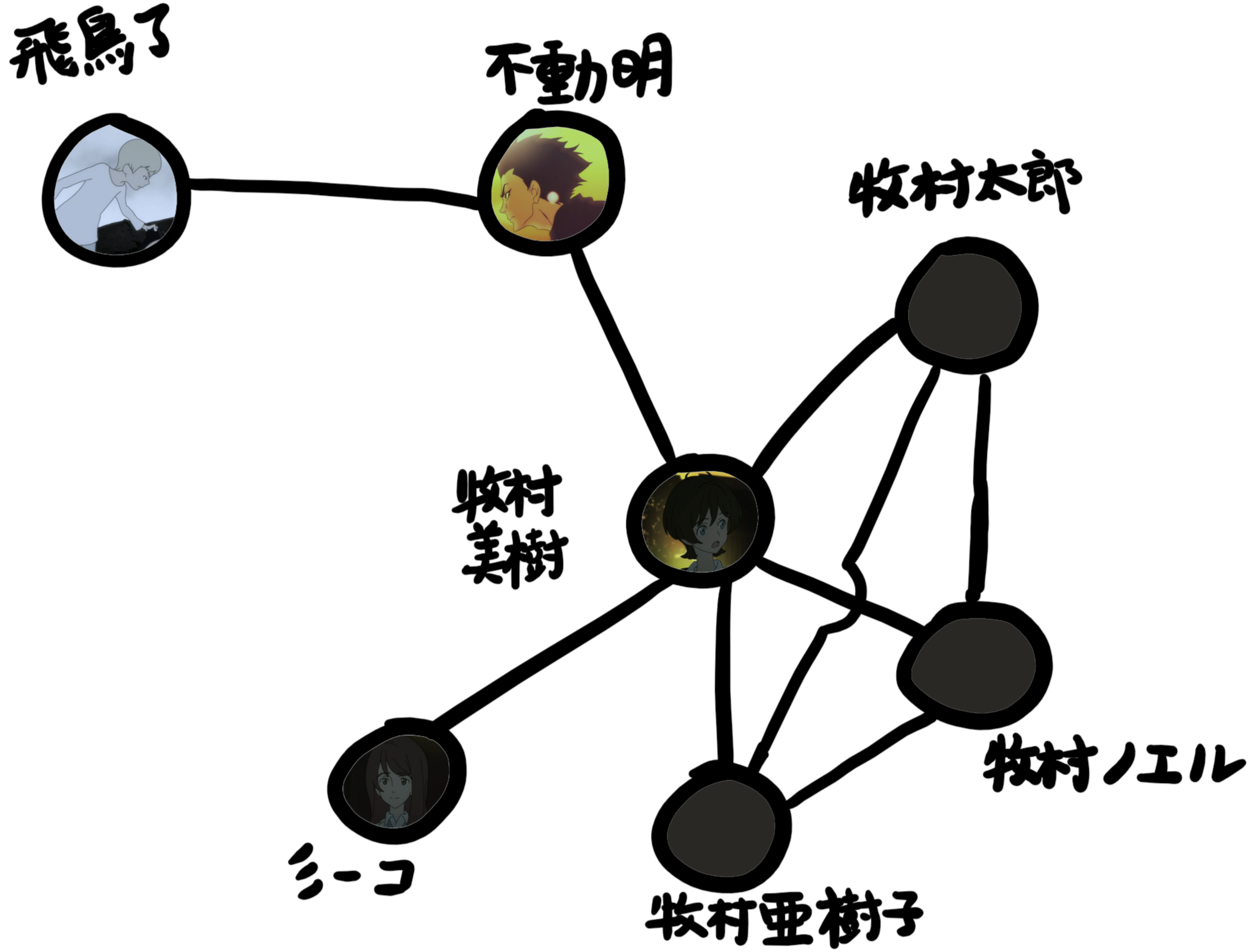
1. 定义 Definitions
2. 图的表示 Storing a Graph
3. 欧拉回路 Eulerian Path
4. 拓扑排序 Topological Sorting
5. 最短路径问题：贝尔曼 – 福特算法
   Shortest Path Problem: Bellman–Ford Algorithm
6. 最小生成树：Kruskal 算法
   Minimum Spanning Tree: Kruskal's Algorithm
7. 最近公共祖先 Lowest Common Ancestor

1. 定义

Difinitions

一个图 G 是一个三元组，即 $G = (V, E, \Phi)$。其中，V 是点（vertex，简作 Vtx）集，E 是边集，Φ 系 E 到 V 的映射／关联函数。

每一条边（edge，或 arc）是一个点对：(v, w)，其中 $v, w \in V$，此时两点邻接（adjacent）。特别地，(v,v) 为一个环（loop）。当边拥有权值（weight，简作 Wgt，或称作 cost），它就是赋权的（weighted）。图的若点对是有序的，就称图是有向图（digraph）。



＊　《Devilman Crybaby》人物关系图　＊

1. 定义

Difinitions

图中一条路径（path）是一个点序 $w_1, w_2, \ldots, w_n$，其中 $(w_i, w_{i+1}) \in E, 1 \le i < N$。路径的长（length，简为 Lgt）是点的总数。长度大于 1 且首尾相连（$w_l = w_n$）的路径叫做圈（cycle）。一个有向无圈图经常来进行研究，它的英文是 DAG。

无向图中某点与其他点连边的个数称为它的度（degree），度数为奇数的称为奇顶点，反之为偶顶点。在有向图中，流出某点的度称作出度（out degree），指向某点的度称作入度（in degree）。

2. 图的表示

Storing a Graph

关于图的存储一共有两种做法，一是邻接矩阵（adjacency matrix），一是邻接表（adjacency list）。

对于邻接矩阵，我们以 `Graph[MAXX][MAXY]` 二维数组形式储存。有两类情况：一是图是否有向，对于有向图，`Graph[u][v]` 表示从 *u* 到 *v* 的一条有向路径；对于无向图，`Graph[u][v]` 或者是 `Graph[v][u]` 都表示 *u* 和 *v* 之间有一条路径。第一类情况说完了。

2. 图的表示

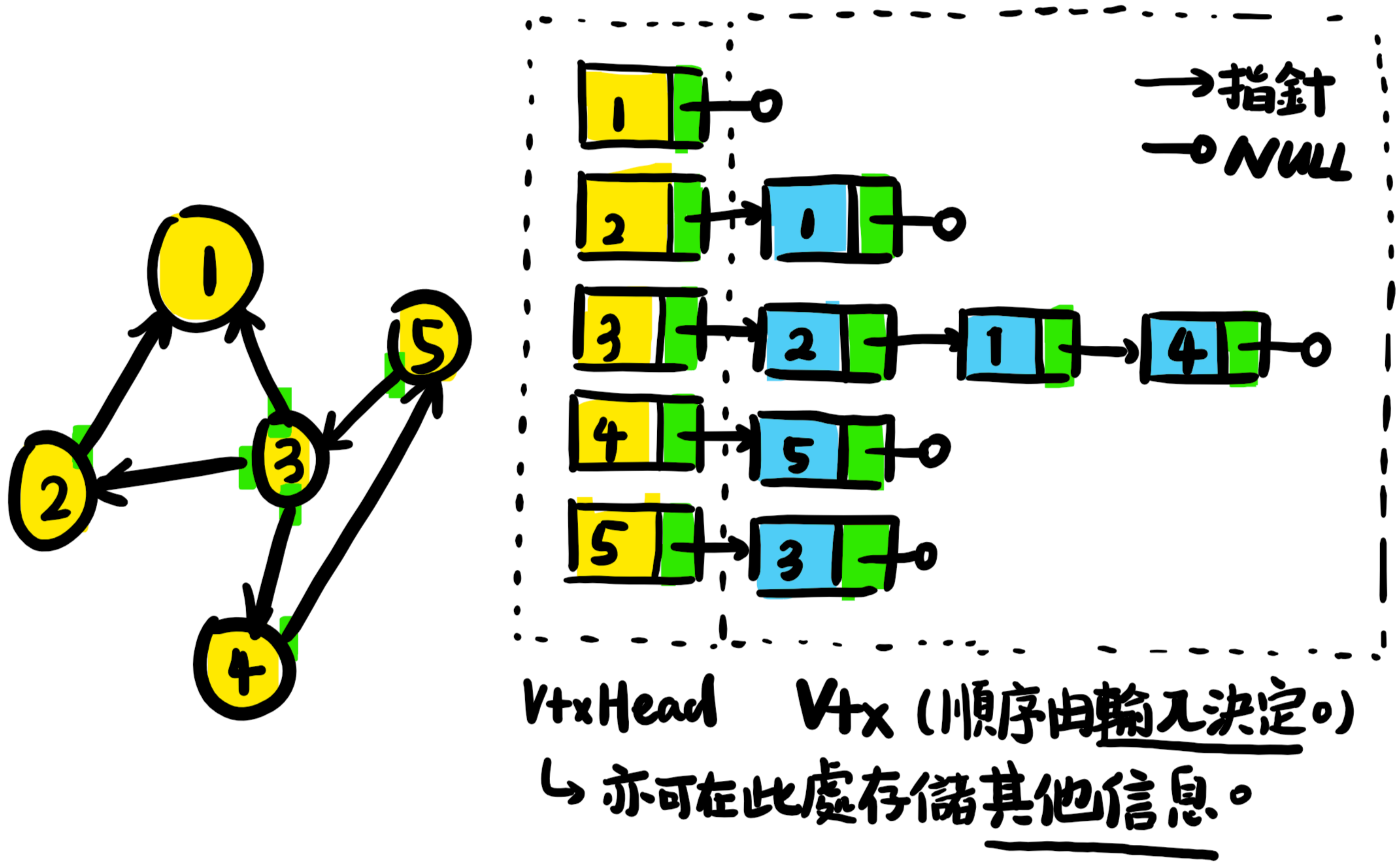
Storing a Graph

第二类是图是否赋权，对于赋权图，我们用 `int` 或 `long long` 数组储存，将未连通的路赋值为 `INT_MAX` ／ `LLONG_MAX` ／ `0x7f7f7f7f` ／ `-1` 等; 对于无权图，使用 `bool` 即可。由于存储空间较大（$\Theta(|V|^2)$），遍历（traversal）较困难，因此如果不是稠密（dense）图，就不常用邻接矩阵。

2. 图的表示

Storing a Graph

邻接表是常用的图的存储方式，广泛适用于稀疏（sparse）图。在此介绍个人研究的方法。我们在 `VtxHead` 结构体中存储每一个顶点的起始信息／其他信息，然后向下生长。`Vtx` 结构体成为了一个链表，这链表一保存 `VtxHead` 所有邻接的点，生长的方式（`Grow( )`）与链表相同。



✳ 邻接表存储图图解 ✳

邻接表存储图：托托的实现

Adjacency List for Storing a Graph: tot's Instance

```
struct Vertex {
    int To;
    Vertex *Next;

    Vertex(void) : To(NotAVertex),
Next(NULL) {}
};

struct VtxHead : Vertex {
    ... // variables that every vertex
should store
    Vertex *Head;

    void operator += (int To)
    {
        if (Head == NULL) {
            Next = new Vertex();
            Next->To = To;
            Head = Next;
```

```
        } else {
            Next->Next = new Vertex();
            Next = Next->Next;
            Next->To = To;
        }
    }

    VtxHead(void) :
        Head(NULL),
        ... ( ... ) {}
} Graph[MAXN];
```

3. 欧拉回路

Eulerian Path

若图 G 中存在一条路径，使得它恰好通过 G 中每条边一次，则称其为<u>欧拉路径</u>（Eulerian path）。特别地，若该路径是一个环路，则称其为<u>欧拉回路</u>。

具有欧拉回路的图为<u>欧拉图</u>，只有欧拉路径的图为<u>半欧拉图</u>。

3. 欧拉回路

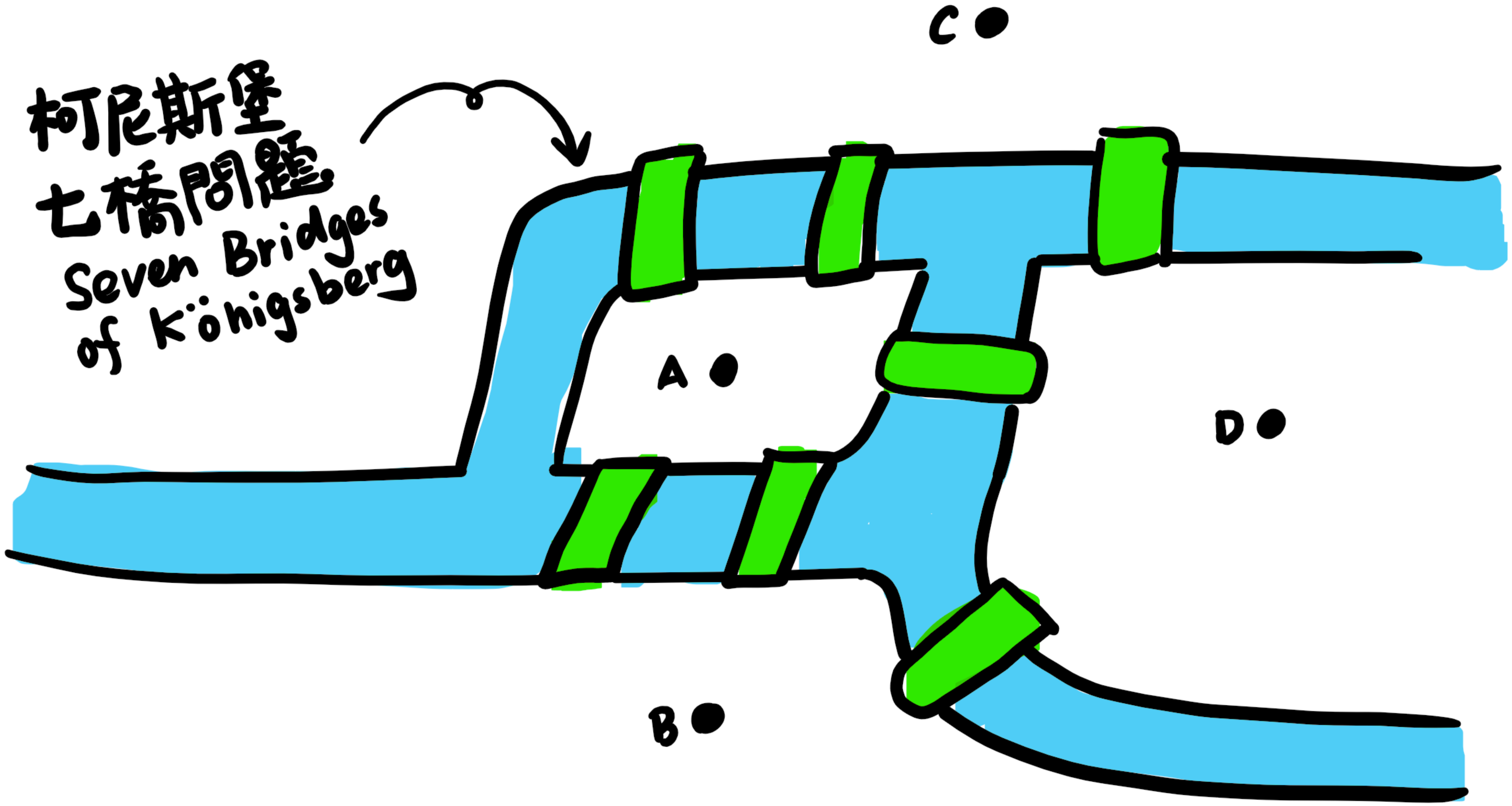
Eulerian Path

对于无向连通图 G，有如下性质：

1. 存在欧拉路径 $\Leftrightarrow$ 图 G 中有两个奇顶点／无奇顶点；

2. 存在欧拉路径且有两个奇顶点，斯两点为图始终；

3. 存在欧拉路径且无奇顶点，则为欧拉回路。



✳ 莱昂哈德·欧拉研究的七桥问题就是一个实例 ✳

3. 欧拉回路

Eulerian Path

对于有向连通图 G，存在欧拉路逕必满足如下任一条件：

1. 其余顶点出入度相等，但有一个顶点出入度之差为 1，有另一顶点出入度之差为 -1，斯两点为图始终；

2. 所有顶点出入度相等，此时路径系欧拉回路。

习题 1：判断欧拉回路

Problem 1: Judge An Eulerian Path

请入：http://www.dsalgo.openjudge.cn/graph/7/

欧拉回路是指不令笔离开始纸面，可画过图中每条边仅一次，且可以回到起点的一条回路。

给定一个无向图，请判断该图是否存在欧拉回路。

习题 1：判断欧拉回路（程序填空）

Problem 1: Judge An Eulerian Path (Filling in the Blank)

```cpp
bool IsVist[MAXN];
int Cnt;
void Search(int Start)
{

    queue<int> Travel;
    Travel.push(__________);
    IsVist[Start] = true;

    while (!Travel.empty()) {
        int From = Travel.front();
        Travel.pop();

        for (Vtx *i = Graph[From].Head;
            i; __________) {
            if (!__________) {
                IsVist[i->To] = true;
                ++Cnt;
                Travel.push(__________);
            }
        }
}
```

习题 1：判断欧拉回路（程序填空）

Problem 1: Judge An Eulerian Path (Filling in the Blank)

```
bool IsVist[MAXN];
int Cnt;
void Search(int Start)
{

    queue<int> Travel;
    Travel.push(Start);
    IsVist[Start] = true;

    while (!Travel.empty()) {
        int From = Travel.front();
        Travel.pop();

        for (Vtx *i = Graph[From].Head;
            i; i = i->Next) {
            if (!IsVist[i->To]) {
                IsVist[i->To] = true;
                ++Cnt;
                Travel.push(i->To);
            }
        }
}
```

习题 1：判断欧拉回路（程序填空）

Problem 1: Judge An Eulerian Path (Filling in the Blank)

```
for (int i = 1; i ⩽ m; ++i) {
    int u, v;
    cin >> u >> v;

    Graph[u] __________;
    Graph[v] += u;

    ++Graph[u].Degree;
    __________;
}

Search(1);

if (__________) {
    cout << "0" << endl;
    return;
}

for (int i = 1; i ⩽ n; ++i)
```

```
    if (Graph[i].Degree __________) {
        cout << "0" << endl;
        return;
    }

cout << __________ << endl;
```

习题 1：判断欧拉回路（程序填空）

Problem 1: Judge An Eulerian Path (Filling in the Blank)

```cpp
for (int i = 1; i <= m; ++i) {
    int u, v;
    cin >> u >> v;

    Graph[u] += v;
    Graph[v] += u;

    ++Graph[u].Degree;
    ++Graph[v].Degree;
}

Search(1);

if (Cnt != n) {
    cout << "0" << endl;
    return;
}

for (int i = 1; i <= n; ++i)
    if (Graph[i].Degree % 2 == 1) {
        cout << "0" << endl;
        return;
    }

cout << "1" << endl;
```

习题 2：单词拼接

Problem 2: Link Words

（测试样例在下分文件中，请完成后上交统一测评。）

甲给了乙 n 个单词，如果一个单词的最后一个字母和另一个单词的第一个字母相同，那么两个单词就可以连接在一起组成一个新的单词。现在甲想要乙计算一下，给定的 n 个单词是否可以全部连接在一起。

习题 2：单词拼接

Problem 2: Link Words

输入格式

第一行输入一个整数 n，代表一共有 n 个单词（$1 \le n \le 10,0000$）。

接下来输入 n 行，每行输入一个单词。单词均由小写字母组成，每个单词长度不超过 20。

输出格式

输出一行，如果所有的单词都可以连接在一起并且可以形成一个环，那么输出 `Euler loop`；如果所有单词都可以连接在一起，但是不会形成环，输出 `Euler path`；如果所有单词不能连在一起，那么输出 `impossible`。

习题 2：单词拼接（程序填空）

Problem 2: Link Words (Filling in the Blank)

```
// This is for adjacency matrix…

bool IsVisited[MAXN] = { false };
int Count = 1;
void Search(int Start)
{
    __________ Travel;
    Travel.push(Start);
    IsVisited[Start] = true;

    while (!__________) {
        int From = Travel.__________;
        Travel.pop();

        for (int i = 0; i < 26; ++i) {
            if (IsConnected[From][i] &&
                !IsVisited[i]) {
                IsVisited[i] = __________;
                ++Count;
```

```
                Travel.push(__________);
            }
        }
    }
}
```

习题 2：单词拼接（程序填空）

Problem 2: Link Words (Filling in the Blank)

```
// This is for adjacency matrix…

bool IsVisited[MAXN] = { false };
int Count = 1;
void Search(int Start)
{
    queue<int> Travel;
    Travel.push(Start);
    IsVisited[Start] = true;

    while (!Travel.empty()) {
        int From = Travel.front();
        Travel.pop();

        for (int i = 0; i < 26; ++i) {
            if (IsConnected[From][i] &&
                !IsVisited[i]) {
                IsVisited[i] = true;
                ++Count;
```

```
                Travel.push(i);
            }
        }
    }
}
```

习题 2：单词拼接（程序填空）

Problem 2: Link Words (Filling in the Blank)

```cpp
int a = 0, b = 0, c = 0;
for (int i = 0; i < 26; ++i)
    if (IsAppeared[i])
        if (Indegree[i] - __________ == -1)
            ++a;
        else if (Indegree[i] - Outdegree[i] == __________)
            ++b;
        else if (Indegree[i] == Outdegree[i])
            ++__________;

if (c == n) {
    cout << "Euler loop" << endl;
    return 0;
} else if (a == 1 && b == 1 && c == __________) {
    cout << __________ << endl;
    return 0;
}
```

习题 2：单词拼接（程序填空）

Problem 2: Link Words (Filling in the Blank)

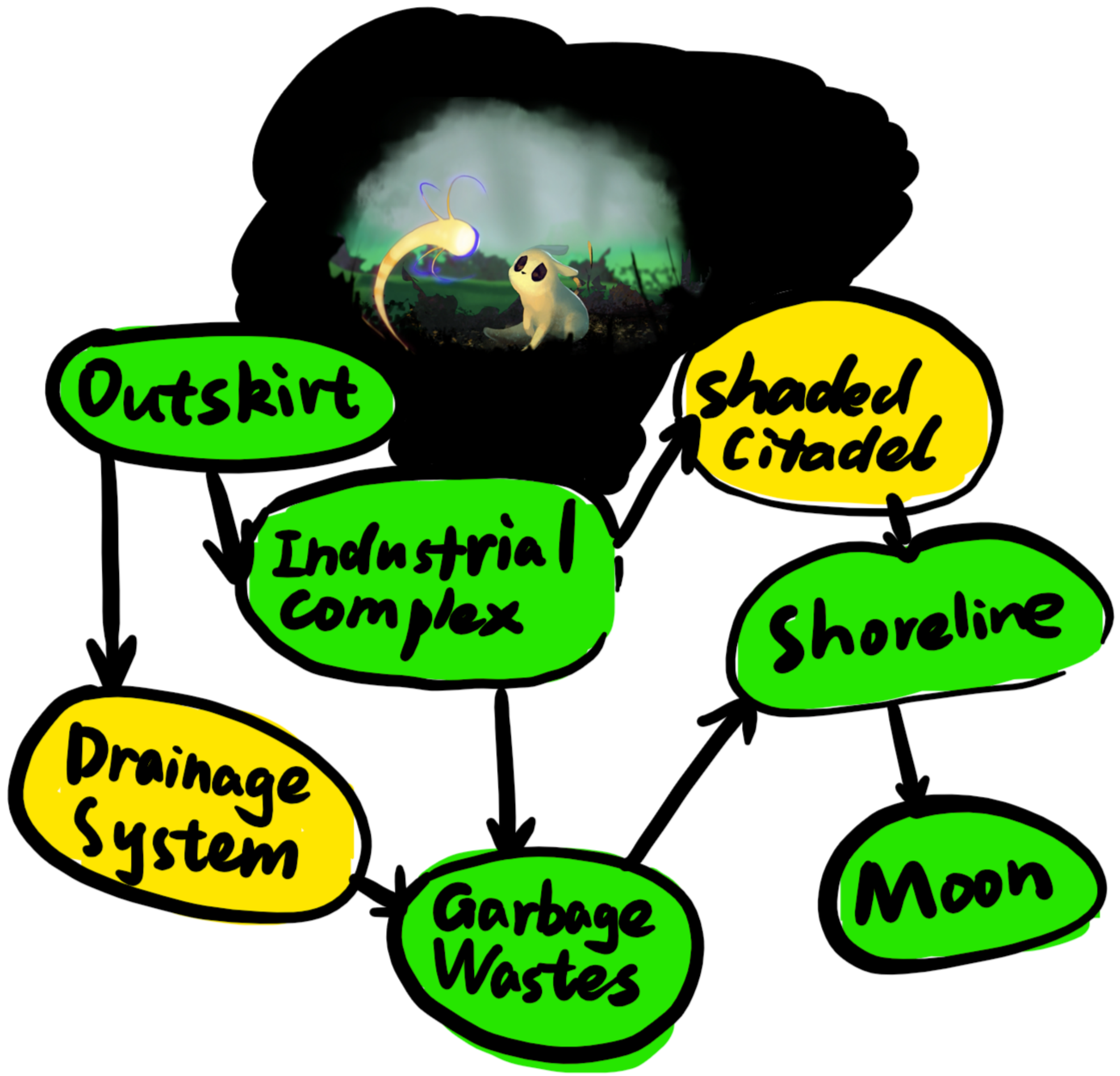
```cpp
int a = 0, b = 0, c = 0;
for (int i = 0; i < 26; ++i)
    if (IsAppeared[i])
        if (Indegree[i] - Outdegree[i] == -1)
            ++a;
        else if (Indegree[i] - Outdegree[i] == 1)
            ++b;
        else if (Indegree[i] == Outdegree[i])
            ++c;

if (c == n) {
    cout << "Euler loop" << endl;
    return 0;
} else if (a == 1 && b == 1 && c == n - 2) {
    cout << "Euler path" << endl;
    return 0;
}
```

4. 拓扑排序

Topological Sorting

拓扑排序的对象是上文提到的有向无环图（Directed Acyclic Graph，DAG）。它的意思是，对图 G 中的顶点进行排序，排序的结果是做到：若有一条边 $(u, v) \in E$，在排序中 u 在 v 之前。排序的结果叫做拓扑序（topological order）。以下的程序能够达到 $O(|E| + |V|)$。



✳ 《Rain World》Session 1 通关流程——一个 DAG ✳

拓扑排序过程演示

Topological Sorting Process

```
inline void TopSort(void)
{
    queue<int> Travel; int Idx = 1;

    for (int i = 1; i <= VtxAmt; ++i)
        if (InDegree[i] == 0)
            Travel.push(i);

    while (!Travel.empty()) {
        int CrtIdx = Travel.front();
        Travel.pop();
        TopIdx[CrtIdx] = Idx++;

        for (Vtx *i = Graph[CrtIdx].Head;
            i; i = i->Next)
            if (--InDegree[i->To] == 0)
                Travel.push(i->To);
    }
}
```

习题 3：拓扑排序

Problem 3: Topological Sorting

请入：http://bailian.openjudge.cn/practice/4084/

给出一个图的结构，输出其拓扑排序序列，要求在同等条件下，编号小的顶点输出在前。

习题 3：拓扑排序（程序填空）

Problem 3: Topological Sorting (Filling in the Blank)

```cpp
struct Unit { int x;
    bool operator < (const Unit &y) const
    { return x __________ y.x; } };

int v, a;

inline void TopSort(void)
{
    priority_queue<Unit> Travel;

    for (int i = 1; i <= v; ++i)
        if (Graph[i].InDegree == 0)
            Travel.push( Unit { i } );

    while (!Travel.empty()) {
        int CrtIdx = Travel.top().x;
        Travel.pop();
        printf("v%d ", __________);
```

```cpp
          for (Vtx *i = Graph[CrtIdx].Head;
              i; i = i->Next)
              if (--Graph[i->To].__________ == 0)
                  Travel.push( Unit { _______ } );
    }
}
```

习题 3：拓扑排序（程序填空）

Problem 3: Topological Sorting (Filling in the Blank)

```cpp
struct Unit { int x;
    bool operator < (const Unit &y) const
    { return x > y.x; } };

int v, a;

inline void TopSort(void)
{
    priority_queue<Unit> Travel;

    for (int i = 1; i <= v; ++i)
        if (Graph[i].InDegree == 0)
            Travel.push( Unit { i } );

    while (!Travel.empty()) {
        int CrtIdx = Travel.top().x;
        Travel.pop();
        printf("v%d ", CrtIdx);
```

```cpp
          for (Vtx *i = Graph[CrtIdx].Head;
              i; i = i->Next)
              if (--Graph[i->To].InDegree ==
0)
                  Travel.push( Unit { i->To
} );
    }
}
```

习题 4：奶牛竞赛

Problem 4: Cow Contest

请入：https://www.luogu.org/problemnew/show/P2419

从 1 编号到 N 的 N（$1 \le N \le 100$）头奶牛正在参加一场程序竞赛。我们知道，有些奶牛的码力总是要强一些。在竞争者中，每一头奶牛都有一个唯一确定的码力排名。

现在给出 M（$1 \le M \le 4500$）轮两两单挑的结果，它们不会自相矛盾。编号为 A 的奶牛的码力强于编号为 B 的奶牛（$1 \le A \le N, 1 \le B \le N, A \ne B$）。在她们的单挑中，编号为 A 的奶牛总能获胜。希望你可以推断出尽可能多的奶牛的具体码力排名。

习题 4：奶牛竞赛（答案 1）

Problem 4: Cow Contest (Answer 1)

```cpp
int IsVist[MAXN];
inline int Go(VtxHead *G, int x)
{
    int Sum = 0;

    for (Vtx *i = G[x].Head;
        i; i = i->Next)
        if (!IsVist[i->To]) {
            IsVist[i->To] = true;
            Sum += Go(G, i->To);
        }

    return Sum + 1;
}

// ...
```

```cpp
int Cnt = 0;
for (int i = 1; i <= n; ++i) {
    memset(IsVist, false, sizeof IsVist);
    int GCnt = Go(Graph, i);
    memset(IsVist, false, sizeof IsVist);
    int AntiGCnt = Go(AntiGraph, i);
    if (GCnt + AntiGCnt == n + 1)
        ++Cnt;
}
```

习题 4：奶牛竞赛（答案 2）

Problem 4: Cow Contest (Answer 2)

```cpp
inline void Floyd(void)
{
    for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= n; ++j)
    for (int k = 1; k <= n; ++k)
        if (Graph[i][j] == BLOCK)
            if (Graph[i][k] == GO &&
                Graph[k][j] == GO) {
                Graph[i][j] = GO;
                Graph[j][i] = BACK;
            } else if (Graph[i][k] == BACK
&&
                Graph[k][j] == BACK) {
                Graph[i][j] = BACK;
                Graph[j][i] = GO;
            }
}

// ...
```

```cpp
int Cnt = 0;
for (int i = 1; i <= n; ++i) {
    int TmpCnt = 0;
    for (int j = 1; j <= n; ++j)
        if (i != j &&
            Graph[i][j] == BLOCK)
            ++TmpCnt;
    if (TmpCnt == 0)
        ++Cnt;
}
```

5. 最短路径问题：貝尔曼 – 福特算法

Shortest Path Problem: Bellman–Ford Algorithm

最短路径是常常涉及的一个问题。顾名思义，就是给定起点和终点，需要求出最短的无权路径长（unweighted path length）或者是赋权路径长（weighted path length，即 $\mathtt{Wgt}_1 + \mathtt{Wgt}_2 + \ldots + \mathtt{Wgt}_N$）。

以后者为例，我们跳过经典的 Dijkstra 算法，直接介绍它的变体 SPFA 算法（Bellman–Ford Algorithm）。他们都是贪婪算法（greedy algorithm）。SPFA 算法不难~~，在笔者没有接触到这个概念时，愚以为此算法是自己发明的~~。

貝尔曼 – 福特算法演示

Bellman–Ford Algorithm Process

```
void Weighted(int Start)
{
    Graph[Start].Dist = 0;
    queue<int> Travel;
    Travel.push(Start);

    while (!Travel.empty()) {
        int From = Travel.front();
        Travel.pop();

        for (Vtx *i = Graph[From].Head;
            i; i = i->Next)
            if (Graph[From].Dist + i->Wgt <
                Graph[i->To].Dist) {
                Graph[i->To].Dist =
                    Graph[From].Dist + i->Wgt;

                Travel.push(i->To);
            } } }
```

习题 5：最短路计数

Problem 5: Count the Shortest Paths

请入：https://www.luogu.org/problemnew/show/P1144

给出一个 N 个顶点、M 条边的无向无权图，顶点编号为 1 - N。问从顶点 1 开始，到其他每个点的最短路有几条。

习题 5：最短路计数（程序填空）

Problem 5: Count the Shortest Paths (Filling in the Blank)

```cpp
struct VtxHead : Vtx {
    // …
    void operator += (int To)
    {
        if (!Head) {
            Next = new Vtx();
            Next->To = To;
            Head = Next;
        } else {
            Next->Next = new Vtx();
            Next = Next->Next;
            Next->To = To;
        }
    }

    VtxHead(void) :  Head(NULL),
        Dist(INT_MAX), Cnt(0), IsVist(false)
{}
} Graph[MAXN];
```

```cpp
for (Vtx *i = Graph[From].Head;
    i; i = i->Next) {
    if (Graph[From].Dist + 1 <
        Graph[i->To].Dist) {
        Graph[i->To].Dist =
            Graph[From].Dist + 1;
        Graph[i->To].Cnt =
            Graph[From].Cnt;

        if (!Graph[i->To].IsVist) {
            Graph[i->To].IsVist = true;
            Travel.push(i->To);
        }
    } else if (Graph[From].Dist + 1 ==
        Graph[i->To].Dist)
        Graph[i->To].Cnt = (Graph[i->To].Cnt
+
            Graph[From].Cnt) % 100003;
}
```

习题 5：最短路计数（程序填空）

Problem 5: Count the Shortest Paths (Filling in the Blank)

```
struct VtxHead : Vtx {
    // …
    void operator += (int To)
    {
        if (!Head) {
            Next = new Vtx();
            ___________ = To;
            Head = Next;
        } else {
            Next->Next = new Vtx();
            Next = ___________;
            Next->To = ___________;
        }
    }

    VtxHead(void) :  Head(NULL),
        Dist(___________), Cnt(0),
IsVist(false) {}
} Graph[MAXN];
```

```
for (Vtx *i = Graph[From].___________;
    i; i = i->Next) {
    if (Graph[From].Dist + 1 <
        Graph[i->To].Dist) {
        Graph[i->To].Dist =
            ___________;
        ___________ =
            Graph[From].Cnt;

        if (!Graph[i->To].IsVist) {
            Graph[i->To].IsVist = true;
            Travel.push(i->To);
        }
    } else if (Graph[From].Dist + 1 ==
        Graph[i->To].Dist)
        Graph[i->To].Cnt = (Graph[i->To].Cnt
+
            Graph[From].Cnt) % ___________;
}
```

习题 6：奶酪

Problem 6: Cheese

此虽不是一最短路题目，但借此训练队列优化这一技巧。

请入：https://vijos.org/p/2031

一块奶酪可以視作一个三维坐标系，其中有許多球形的中空。老鼠杰瑞想穿过这些中空从一端来到另一端。希望你能够告诉她是否能到终点。

习题 6：奶酪（答案）

Problem 6: Cheese (Answer)

```cpp
for (int i = 1; i <= n; ++i) {
    scanf("%lld %lld %lld",
        &Cheese[i].x, &Cheese[i].y,
&Cheese[i].z);
    if (Cheese[i].z + r >= h)
        G[i] += n + 1;
    if (Cheese[i].z <= r)
        G[0] += i;
}

for (int i = 1; i <= n - 1; ++i)
    for (int j = i + 1; j <= n; ++j)
        if (IsConnected(Cheese[i],
Cheese[j])) {
            G[i] += j;
            G[j] += i;
        }
```

```cpp
inline long long Calc(
    const Coord &c_1, const Coord &c_2)
{
    long long xSpan = c_1.x - c_2.x,
        ySpan = c_1.y - c_2.y,
        zSpan = c_1.z - c_2.z;

    return xSpan * xSpan +
        ySpan * ySpan +
        zSpan * zSpan;
}

bool IsConnected(
    const Coord &c_1, const Coord &c_2)
{
    return Calc(c_1, c_2) <= (long long)4 *
r * r;
}
```

习题 6：奶酪（答案）

Problem 6: Cheese (Answer)

```
bool IsVist[MAXN];
void Search(void)
{
    queue<int> Travel;
    Travel.push(0);
    IsVist[0] = true;

    while (!Travel.empty()) {
        int From = Travel.front();
        Travel.pop();

        for (Vtx *i = G[From].Head;
            i; i = i->Next)
            if (!IsVist[i->To]) {
                IsVist[i->To] = true;

                if (i->To == n + 1) return;

                Travel.push(i->To); } } }
```

6. 最小生成树：Kruskal 算法

Minimum Spanning Tree: Kruskal's Algorithm

关于树有两个定理，一是定点数 N、变数 E 满足 $E = N - 1$；而是关于有 N 顶点，E 条边的图 T 的三个等价命题，在<u>概念辨析题</u>中曾有所涉及：

1. 图 T 是树；

2. 图 T 无圈，且 $E = N - 1$；

3. 图 T 连通，且 $E = N - 1$。

生成树是图去掉若干边的<u>最小连通子图</u>。

6. 最小生成树：Kruskal 算法

Minimum Spanning Tree: Kruskal's Algorithm

对于一个赋权图，找到权值最小的生成树就是去求取最小生成树（Minimum Spanning Tree，MST），我们使用同样是贪婪算法的 Kruskal 为例。给定图的边集 E 和生成树集 T，且每个点属于一个独立集合（$S_1, S_2, \dots, S_N$），它的算法的过程是：

1. 从小至大排序 E；

2. 从小至大遍历边，对于 $(u, v) \in E$，若 u、v 不在同一集合，就合并它们的集合，使用并查集完成；

3. 重复步骤 2，直到仅剩一个集合或 E 遍历完成。

并查集（不相交集）例程

Disjoint Set Process

```cpp
int Ast[MAXN];
int GetAst(int x) {
    if (Ast[x] == x) return x;
    return Ast[x] = GetAst(Ast[x]);
}

inline bool Query(int x, int y)
{
    return GetAst(x) == GetAst(y);
}

inline void Merge(int x, int y)
{
    int xAst = GetAst(x),
        yAst = GetAst(y);

    if (xAst == yAst)
        return;
```

```cpp
    Ast[yAst] = xAst;
}

inline void Init(int Amount)
{
    for (int i = 0; i < Amount; ++i)
        Ast[i] = i;
}

// more at https://github.com/bufhdy/tot-
problem/tree/master/materials/ADT/disjoint
```

最小生成树：Kruskal 算法演示

Minimum Spanning Tree: Kruskal's Algorithm Process

```cpp
struct Arc {
    int u, v;
    long long Dist;

    bool operator < (const Arc &x) const
    {
        return Dist < x.Dist;
    }
} Graph[MAXN];

// ...
```

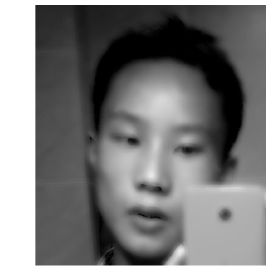
```cpp
sort(Graph, Graph + ArcAmt);

long long Sum = 0;
int Set = n;
// the amount of vertices, finally Set ==
1;
for (int i = 0; i < ArcAmt && Set > 1;
++i) {
    int x = Graph[i].u,
        y = Graph[i].v,
        xAst = GetAst(x),
        yAst = GetAst(y);

    if (xAst != yAst) {
        Ast[xAst] = yAst;
        --Set; // Attention!
        Sum += Graph[i].Dist;
    }
}
```

谢谢大家。

Thanks.



御薬袋托托

就读于重庆市第十一中学校，高 2019 級学生
《NOIP 竞赛指北（noip.guide）》作者，其相关的代码仓库（code.noip.guide）
〇〇年代折衷媒体《托托的故事（totstories.pw）》作者
《忽冷忽熱怪物 Lukewarm Beast》主播

啁啾会馆：@bufhdy　刹那图鉴：@bufhdy
如来枢纽：@bufhdy　电報：@bufhdy　电线：@bufhdy
邮箱：bufhdy@hotmail.com