ICPC TEMPLATE

widsnoy

2023 年 12 月 7 日

目录

目录

1 图论

1.1 最近公共祖先

```cpp
// 倍增
int faz[N][20], dep[N];
void dfs(int u, int fa) {
    faz[u][0] = fa;
    dep[u] = dep[fa] + 1;
    for (int i = 1; i < 20; i++) faz[u][i] = faz[faz[u][i - 1]][i - 1];
    for (int v : G[u]) if (v != fa) {
        dfs(v, u);
    }
}
int LCA(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    int d = dep[u] - dep[v];
    for (int i = 0; i < 20; i++) if ((d >> i) & 1) u = faz[u][i];
    if (v == u) return u;
    for (int i = 19; i >= 0; i--) if (faz[u][i] != faz[v][i])
        u = faz[u][i], v = faz[v][i];
    return faz[u][0];
}

//树剖
int dfc, dfn[N], rnk[N], siz[N], top[N], dep[N], son[N], faz[N];
void dfs1(int u, int fa) {
    dep[u] = dep[fa] + 1;
    siz[u] = 1;
    son[u] = -1;
    faz[u] = fa;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if (son[u] == -1 || siz[son[u]] < siz[v]) son[u] = v;
    }
}
void dfs2(int u, int fa, int tp) {
    dfn[u] = ++dfc;
    rnk[dfc] = u;
    top[u] = tp;
    if (son[u] != -1) dfs2(son[u], u, tp);
    for (int v : G[u]) {
        if (v == fa || v == son[u]) continue;
```

```cpp
            dfs2(v, u, v);
        }
}
int LCA(int u, int v) {
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]])
            u = faz[top[u]];
        else
            v = faz[top[v]];
    }
    return dep[u] > dep[v] ? v : u;
}
```

1.2 二分图

最大匹配

```cpp
int mch[maxn], vis[maxn];
std::vector<int> e[maxn];
bool dfs(const int u, const int tag) {
    for (auto v : e[u]) {
        if (vis[v] == tag) continue;
        vis[v] = tag;
        if (!mch[v] || dfs(mch[v], tag)) return mch[v] = u, 1;
    }
    return 0;
}
int main() {
    int ans = 0;
    for (int i = 1; i <= n; ++i) if (dfs(i, i)) ++ans;
}
```

1.3 有向图最小路径覆盖问题

```cpp
int n, m;
bitset<N> f[N];
int vis[N], mch[N];

bool dfs(int u, int dfc) {
    for (int v = 1; v <= n; v++) if (v != u && vis[v] != dfc && f[u][v]) {
        vis[v] = dfc;
        if (!mch[v] || dfs(mch[v], dfc)) return mch[v] = u, 1;
    }
    return 0;
```

```
}

void solve() {
    memset(vis, 0, sizeof vis);
    memset(mch, 0, sizeof mch);
    for (int i = 1; i <= n; i++) f[i].reset();
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        f[u].set(v);
    }
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) if (f[i][k]) f[i] |= f[k];
    }
    int res = n;
    for (int i = 1; i <= n; i++) res -= dfs(i, i);
    printf("%d\n", res);
}
```

1.4 网络流

1.4.1 Dinic 最大流

注意每次清空数组的范围是 s 到 t.

```
int head[N], cur[N], ecnt, d[N];
struct Edge {
    int nxt, v, flow, cap;
}e[];
void add_edge(int u, int v, int flow, int cap) {
    e[ecnt] = {head[u], v, flow, cap}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0}; head[v] = ecnt++;
}
bool bfs() {
    memset(vis, 0, sizeof vis);
    std::queue<int> q;
    q.push(s);
    vis[s] = 1;
    d[s] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if (vis[v] || e[i].flow >= e[i].cap) continue;
```

```
            d[v] = d[u] + 1;
            vis[v] = 1;
            q.push(v);
        }
    }
    return vis[t];
}
int dfs(int u, int a) {
    if (u == t || !a) return a;
    int flow = 0, f;
    for (int& i = cur[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (d[u] + 1 == d[v] && (f = dfs(v, std::min(a, e[i].cap - e[i].flow))
            ) > 0) {
            e[i].flow += f;
            e[i ^ 1].flow -= f;
            flow += f;
            a -= f;
            if (!a) break;
        }
    }
    return flow;
}
```

1.4.2 最小费用最大流

```
const int inf = 1e9;
int head[N], cur[N], ecnt, dis[N], s, t, n, m, mincost;
bool vis[N];
struct Edge {
    int nxt, v, flow, cap, w;
}e[100002];
void add_edge(int u, int v, int flow, int cap, int w) {
    e[ecnt] = {head[u], v, flow, cap, w}; head[u] = ecnt++;
    e[ecnt] = {head[v], u, flow, 0, -w}; head[v] = ecnt++;
}
bool spfa(int s, int t) {
    std::fill(vis + s, vis + t + 1, 0);
    std::fill(dis + s, dis + t + 1, inf);
    std::queue<int> q;
    q.push(s);
    dis[s] = 0;
    vis[s] = 1;
    while (!q.empty()) {
```

```
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i != -1; i = e[i].nxt) {
            int v = e[i].v;
            if (e[i].flow < e[i].cap && dis[u] + e[i].w < dis[v]) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) vis[v] = 1, q.push(v);
            }
        }
    }
    return dis[t] != inf;
}
int dfs(int u, int a) {
    if (vis[u]) return 0;
    if (u == t || !a) return a;
    vis[u] = 1;
    int flow = 0, f;
    for (int& i = cur[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (dis[u] + e[i].w == dis[v] && (f = dfs(v, std::min(a, e[i].cap - e[
            i].flow))) > 0) {
            e[i].flow += f;
            e[i ^ 1].flow -= f;
            flow += f;
            mincost += e[i].w * f;
            a -= f;
            if (!a) break;
        }
    }
    vis[u] = 0;
    return flow;
}
```

1.4.3 最大闭权子图

正权点向 S 连边，负权点向 T 连边。边权为点权的绝对值。原图的边容量设为 INF 。

则最大收益为 $\sum_{v>0} v - mincost$

在最大闭权子图中的点是残量网络中 S 能到达的点。

1.5 树哈希

```cpp
const ull mask = chrono::steady_clock::now().time_since_epoch().count();

ull shift(ull x) {
    x ^= mask;
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    x ^= mask;
    return x;
}
int n;
ull H[N];
vector<int> G[N];
set<ull> s;

void dfs(int u, int fa) {
    H[u] = 1;
    for (int v : G[u]) {
        if (v == fa) continue;
        dfs(v, u);
        H[u] += shift(H[v]);
    }
    s.emplace(H[u]);
}
```

1.6 强联通分量

```cpp
int n, dfc, dfn[N], low[N], stk[N], top, idx[N], in_stk[N], scc_cnt;
vector<int> G[N];

void tarjan(int u) {
    low[u] = dfn[u] = ++dfc;
    stk[++top] = u;
    in_stk[u] = 1;
    for (int v : G[u]) {
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in_stk[v]) low[u] = min(dfn[v], low[u]);
    }
    if (low[u] == dfn[u]) {
        int x;
        scc_cnt++;
```

```
        do {
            x = stk[top--];
            idx[x] = scc_cnt;
            in_stk[x] = 0;
        } while (x != u);
    }
}

// 多测清空
dfc = scc_cnt = top = 0;
for (int i = 1; i <= tot; i++) low[i] = dfn[i] = idx[i] = in_stk[i] = 0;
```

1.7 割点和桥

```
int dfn[N], low[N], dfs_clock;
bool iscut[N], vis[N];
void dfs(int u, int fa) {
    dfn[u] = low[u] = ++dfs_clock;
    vis[u] = 1;
    int child = 0;
    for (int v : e[u]) {
        if (v == fa) continue;
        if (!dfn[v]) {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            child++;
            if (low[v] >= dfn[u]) iscut[u] = 1;
        } else if (dfn[u] > dfn[v] && v != fa) low[u] = min(low[u], dfn[v]);
        if (fa == 0 && child == 1) iscut[u] = 0;
    }
}
```

1.8 点双联通分量

```
int bccno[N], bcc_cnt, siz_e[N], siz_p[N], dfs_clock, low[N], dfn[N], top;
pair<int, int> stk[N];
void dfs(int u, int fa) {
    low[u] = dfn[u] = ++dfs_clock;
    for(int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].v;
        if(v == fa) continue;
        if(!dfn[v]) {
            stk[++top] = make_pair(u, v);
```

```cpp
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] >= dfn[u]) {
                bcc_cnt++;
                while(true) {
                    int x = stk[top].first, y = stk[top].second;
                    top--;
                    siz_e[bcc_cnt]++;
                    if(bccno[x] != bcc_cnt) {bccno[x] = bcc_cnt; siz_p[bcc_cnt]++;}
                    if(bccno[y] != bcc_cnt) {bccno[y] = bcc_cnt; siz_p[bcc_cnt]++;}
                    if(x == u && y == v) break;
                }
            }
        } else if(dfn[v] < dfn[u]) {stk[++top] = make_pair(u, v); low[u] = min(low[u], dfn[v]);}
    }
}
```

1.9 边双联通分量

```cpp
const int N = 5000 + 5;
int n, m, stk[N], top, ccno, sc[N];
int dfn[N], dfc, low[N];
int mp[N][N];
int in[N];
int head[N], ecnt;
struct Edge {
    int nxt, v;
} e[N << 2];
void add_edge(int u, int v) {
    e[ecnt] = {head[u], v}; head[u] = ecnt++;
    e[ecnt] = {head[v], u}; head[v] = ecnt++;
}
void dfs(int u, int from) {
    stk[++top] = u;
    low[u] = dfn[u] = ++dfc;
    for (int i = head[u]; i != -1; i = e[i].nxt) {
        int v = e[i].v;
        if (!dfn[v]) {
            dfs(v, i);
            low[u] = min(low[u], low[v]);
        } else if ((i ^ 1) != from) low[u] = min(low[u], dfn[v]);
```

```
    }
    if (dfn[u] == low[u]) {
        ccno++;
        int x;
        while (true) {
            x = stk[top--];
            sc[x] = ccno;
            if (x == u) break;
        }
    }
}

void solve() {
    memset(head, -1, sizeof head);
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        add_edge(u, v);
    }
    for (int i = 1; i <= n; i++) if (!dfn[i]) dfs(i, i);
    for (int i = 1; i <= n; i++) {
        for (int k = head[i]; k != -1; k = e[k].nxt) {
            int j = e[k].v;
            if (sc[i] != sc[j]) mp[sc[i]][sc[j]] = 1;
        }
    }

    for (int i = 1; i <= ccno; i++) {
        for (int j = 1; j <= ccno; j++) if (mp[i][j]) in[j]++;
    }
    int cnt = 0;
    for (int i = 1; i <= ccno; i++) if (in[i] == 1) cnt++;
    printf("%d\n", (cnt + 1) / 2);
}
```

1.10 2-SAT

$2 * u$ 代表不选择，$2 * u + 1$ 代表选择。

也可以求强连通分量。

如果对于一个 *x* ‘sccno‘比它的反状态 *x*1 的 ‘sccno‘ 要小，那么我们用 *x* 这个状态当做答

案，否则用它的反状态当做答案。

```
vector<int> G[N * 2];
bool mark[N * 2];
int stk[N], top;
void build_G() {
    for (int i = 1; i <= n; i++) {
        int u, v;
        G[2 * u + 1].push_back(2 * v);
        G[2 * v + 1].push_back(2 * u);
    }
}
bool dfs(int u) {
    if (mark[u ^ 1]) return false;
    if (mark[u]) return true;
    mark[u] = 1;
    stk[++top] = u;
    for (int v : G[u]) {
        if (!dfs(v)) return false;
    }
    return true;
}
bool 2_sat() {
    for (int i = 1; i <= n; i++) {
        if (!mark[i * 2] && !mark[i * 2 + 1]) {
            top = 0;
            if (!dfs(2 * i)) {
                while (top) mark[stk[top--]] = 0;
                if (!dfs(2 * i + 1)) return 0;
            }
        }
    }
    return 1;
}
```

2 字符串

2.1 哈希

2.1.1 最长回文子串

通过哈希同样可以 $O(n)$ 解决这个问题，具体方法就是记 R_i 表示以 i 作为结尾的最长回文的长度，那么答案就是 $\max_{i=1}^{n} R_i$。考虑到 $R_i \leq R_{i-1}+2$，因此我们只需要暴力从 $R_{i-1}+2$ 开始递

减，直到找到第一个回文即可。记变量 z 表示当前枚举的 R_i，初始时为 0，则 z 在每次 i 增大的时候都会增大 2，之后每次暴力循环都会减少 1，故暴力循环最多发生 $2n$ 次，总的时间复杂度为 $O(n)$。

2.2 字典树

2.3 维护异或和

```
const int N = 526010, MX = 22;
int ch[N * MX][2], tot, rt[N], w[N * MX], xorv[N * MX], val[N];
ll ans;

void pushup(int u) {
    w[u] = xorv[u] = 0;
    if (ch[u][0]) {
        w[u] += w[ch[u][0]];
        xorv[u] ^= (xorv[ch[u][0]] << 1);
    }
    if (ch[u][1]) {
        w[u] += w[ch[u][1]];
        xorv[u] ^= (xorv[ch[u][1]] << 1) | (w[ch[u][1]] & 1);
    }
    w[u] &= 1;
}
void insert(int &o, ll ux, int dep) {
    if (!o) o = ++tot;
    if (dep > MX) return (void)(w[o]++);
    insert(ch[o][ux & 1], ux >> 1, dep + 1);
    pushup(o);
}
void addall(int o) {
    swap(ch[o][0], ch[o][1]);
    if (ch[o][0]) addall(ch[o][0]);
    pushup(o);
}
int merge(int a, int b) {
    if (!b || !a) return a + b;
    xorv[a] ^= xorv[b];
    w[a] += w[b];
    ch[a][0] = merge(ch[a][0], ch[b][0]);
    ch[a][1] = merge(ch[a][1], ch[b][1]);
    return a;
}
```

```cpp

vector<int> G[N];
int read() {
    int w = 0, f = 1; char ch = getchar();
    while (ch > '9' || ch < '0') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        w = w * 10 + ch - 48;
        ch = getchar();
    }
    return w * f;
}

void dfs(int u) {
    for (auto v : G[u]) {
        dfs(v);
        rt[u] = merge(rt[u], rt[v]);
    }
    addall(rt[u]);
    insert(rt[u], val[u], 0);
    ans += (ll)xorv[rt[u]];
}

int main() {
    int n = read();
    for (int i = 1; i <= n; i++) val[i] = read();
    for (int i = 2; i <= n; i++) G[read()].push_back(i);
    dfs(1);
    printf("%lld\n", ans);
    return 0;
}
```

2.4 KMP

```cpp
int n = strlen(s + 1);
for (int i = 2; i <= n; i++) {
    int j = k[i - 1];
    while (j != 0 && s[i] != s[j + 1]) j = k[j];
    if (s[i] == s[j + 1]) k[i] = j + 1;
    else k[i] = 0;
}
```

2.4.1 字符串最小周期

设 border 长度为 r

则 $s[i] = s[n - r + i]$

$|T| = n - r$

2.4.2 每个前缀的出现次数

1. 统计每个前缀在自身的出现次数

```
vector<int> ans(n + 1);
for (int i = 1; i <= n; i++) ans[k[i]]++;
for (int i = n; i >= 1; i--) ans[k[i]] += ans[i];
for (int i = 1; i <= n; i++) ans[i]++;
```

2. 统计每个前缀在其他串的出现次数

我们应用来自 Knuth-Morris-Pratt 的技巧：构造一个字符串 $s + \# + t$ 并计算其前缀函数。与第一个问题唯一的不同之处在于，我们只关心与字符串 t 相关的前缀函数值，即 $i \geq n + 1$ 的 $\pi[i]$。有了这些值之后，我们可以同样应用在第一个问题中的算法来解决该问题。

2.4.3 一个字符串中本质不同子串的数目

给定一个长度为 n 的字符串 s，我们希望计算其本质不同子串的数目。

我们将迭代的解决该问题。换句话说，在知道了当前的本质不同子串的数目的情况下，我们要找出一种在 s 末尾添加一个字符后重新计算该数目的方法。

令 k 为当前 s 的本质不同子串数量。我们添加一个新的字符 c 至 s。显然，会有一些新的子串以字符 c 结尾。我们希望对这些以该字符结尾且我们之前未曾遇到的子串计数。

构造字符串 $t = s + c$ 并将其反转得到字符串 $t^\sim$。现在我们的任务变为计算有多少 $t^\sim$ 的前缀未在 $t^\sim$ 的其余任何地方出现。如果我们计算了 $t^\sim$ 的前缀函数最大值 $\pi_{\max}$，那么最长的出现在 s 中的前缀其长度为 $\pi_{\max}$。自然的，所有更短的前缀也出现了。

因此，当添加了一个新字符后新出现的子串数目为 $|s| + 1 - \pi_{\max}$。

所以对于每个添加的字符，我们可以在 $O(n)$ 的时间内计算新子串的数目，故最终复杂度为 $O(n^2)$。

值得注意的是，我们也可以重新计算在头部添加一个字符，或者从尾或者头移除一个字符时的本质不同子串数目。

2.5 AC 自动机

```cpp
namespace AC {
    int ch[N][26], tot, fail[N], e[N];
    void insert(const char *s) {
        int u = 0, n = strlen(s + 1);
        for (int i = 1; i <= n; i++) {
            if (!ch[u][s[i] - 'a']) ch[u][s[i] - 'a'] = ++tot;
            u = ch[u][s[i] - 'a'];
        }
        e[u] += 1;
    }
    void build() {
        queue<int> q;
        for (int i = 0; i <= 25; i++) if (ch[0][i]) q.push(ch[0][i]);
        while (!q.empty()) {
            int now = q.front(); q.pop();
            for (int i = 0; i < 26; i++) {
                if (ch[now][i]) fail[ch[now][i]] = ch[fail[now]][i], q.push(ch
                    [now][i]);
                else ch[now][i] = ch[fail[now]][i];
            }
        }
    }
    int query(const char *s) {
        int u = 0, n = strlen(s + 1), res = 0;
        for (int i = 1; i <= n; i++){
            u = ch[u][s[i] - 'a'];
            for (int j = u; j && e[j] != -1; j = fail[j]) {
                res += e[j];
                e[j] = -1;
            }
        }
        return res;
    }
}
```

2.6 后缀数组

```cpp
const int N = 2e5 + 5;
```

```
int sa[N << 1], ork[N << 1], rk[N << 1], cnt[N], id[N << 1], M, n;
char s[N];

int main() {
    scanf("%s", s + 1);
    n = strlen(s + 1);
    for (int i = n + 1; i <= (n << 1); i++) s[i] = s[i - n], M = max(M, (int)s
       [i]);
    n <<= 1;
    for (int i = 1; i <= n; i++) if ((int)(s[i]) > M) M = (int)(s[i]);
    for (int i = 1; i <= n; i++) cnt[rk[i] = s[i]]++;
    for (int i = 0; i <= M; i++) cnt[i] += cnt[i - 1];
    for (int i = n; i; i--) sa[cnt[rk[i]]--] = i;
    for (int w = 1, p; w < n; w <<= 1, M = p) {
        p = 0;
        for (int i = n; i > n - w; i--) id[++p] = i;
        for (int i = 1; i <= n; i++) if (sa[i] > w) id[++p] = sa[i] - w;
        for (int i = 0; i <= M; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) cnt[rk[i]]++;
        for (int i = 1; i <= M; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i; i--) sa[cnt[rk[id[i]]]--] = id[i];
        p = 0;
        for (int i = 0; i <= n; i++) ork[i] = rk[i];
        for (int i = 1; i <= n; i++) {
            if (ork[sa[i]] == ork[sa[i - 1]] && ork[sa[i] + w] == ork[sa[i -
               1] + w]) rk[sa[i]] = p;
            else rk[sa[i]] = ++p;
        }
        if (p == n) break;
    }
    for (int i = 1, k = 0; i <= n; i++) {
        if (rk[i] == 1) continue;
        if (k) k--;
        while (s[i + k] == s[sa[rk[i] - 1] + k]) k++;
        h[rk[i]] = k;
    }
    return 0;
}
```

2.6.1 Manacher

对于第 i 个字符为对称轴:

1. 如果回文串长为奇数, $d[2*i]/2$ 是半径加上自己的长度

2. 如果长为偶数, $d[2*i-1]/2$ 是半径的长度, 方向向右.

```
int n, d[N * 2];
char s[N];

for (int i = 1; i <= n; i++) t[i * 2] = s[i], t[i * 2 - 1] = '#';
t[n * 2 + 1] = '#';
m = n * 2 + 1;
for (int i = 1, l = 0, r = 0; i <= m; i++) {
    int k = i <= r ? min(d[r - i + l], r - i + 1) : 1;
    while (i + k <= m && i - k >= 1 && t[i + k] == t[i - k]) k++;
    d[i] = k--;
    if (i + k > r) r = i + k, l = i - k;
}
```

2.7 Z 函数

$z[i] = lcp(suf_1, suf_i)$

```
for (int i = 2, l = 0, r = 0; i <= n; i++) {
    if (r >= i && r - i + 1 > z[i - l + 1]) {
        z[i] = z[i - l + 1];
    } else {
        z[i] = max(0, r - i + 1);
        while (z[i] < n - i + 1 && s[z[i] + 1] == s[i + z[i]]) ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
}
```

3 数学

3.1 线性基

```
struct LinerBasis {
    int a[20], pos[20];
    void add(int v, int p) {
        for (int i = 19; i >= 0; i--) if ((v >> i) & 1) {
            if (a[i]) {
                if (p > pos[i]) {
                    swap(p, pos[i]);
                    swap(a[i], v);
                }
                v ^= a[i];
            } else {
```

```
                a[i] = v;
                pos[i] = p;
                return;
            }
        }
    }
} b[N];

LinerBasis operator + (LinerBasis a, LinerBasis b) {
    for (int i = 19; i >= 0; i--) {
        if (b.a[i]) a.add(b.a[i], b.pos[i]);
    }
    return a;
}
```

3.2 多项式

3.2.1 FFT

```
typedef long long ll;
typedef complex<double> cp;
const int N = 6e5 + 5;
const double pi = acos(-1.0);
int n, m, len = 1, l, rev[N], x[N], y[N];
cp a[N * 2], b[N];

void fft(cp *a, int n, int inv) {
    for (int i = 0; i < n; i++) if (rev[i] < i) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k <<= 1) {
        cp wn(cos(pi / k), inv * sin(pi / k));
        for (int i = 0; i < n; i += k * 2) {
            cp w(1, 0);
            for (int j = 0; j < k; j++, w *= wn) {
                cp x = a[i + j], y = a[i + j + k] * w;
                a[i + j] = x + y, a[i + j + k] = x - y;
            }
        }
    }
    if (inv < 0) for (int i = 0; i < len; i++) a[i] /= n;
}
```

3.3 组合数学

3.3.1 小球放盒

第二类斯特林数（斯特林子集数）$\begin{Bmatrix} n \\ k \end{Bmatrix}$，也可记做 $S(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数。

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

边界是 $\begin{Bmatrix} n \\ 0 \end{Bmatrix} = [n=0]$。

假设小球个数为 n, 盒子个数为 m

1. 小球无标号，盒子有标号，不允许空盒。

即求解方程 $\sum\limits_{i=1}^{m} x_i = n$ 解的个数

即 $\binom{n-1}{m-1}$

2. 小球无标号，盒子有标号，允许空盒。

令 $y_i = x_i + 1$

即求解方程 $\sum\limits_{i=1}^{m} y_i = n$ 解的个数

即 $\binom{n+m-1}{m-1}$

3. 小球有标号，盒子有标号，允许空盒。

即 m^n

4. 小球有标号，盒子有标号，不允许空盒。

$m! \times \begin{Bmatrix} n \\ m \end{Bmatrix}$

5. 小球有标号，盒子无标号，不允许空盒。

$\begin{Bmatrix} n \\ m \end{Bmatrix}$

6. 小球有标号，盒子无标号，允许空盒。

$\sum_{i=1}^{m} \begin{Bmatrix} n \\ i \end{Bmatrix}$

7. 小球无标号，盒子无标号，允许空盒。

设 $f[i][j]$ 表示 i 个球放入 j 个盒子的方案数。

1. $i = 0$ 或者 $j = 1$, 方案数为 1

2. $i < j$, $f[i][j] = f[i][i]$

3. $i \geq j$, $f[i][j] = f[i-j][j] + f[i][j-1]$

8. 小球无标号，盒子无标号，不允许空盒。

用 7 的结论，提前在每个盒子放 1 个球。

方案数就是 $f[n-m][m]$