2026年8月 SACCC 六级题解

苏州市人工智能学会 · 青少年人工智能核心算法素养考核（SACCC）

级别：六级 期次：2026年8月

题目清单

顺序	题目	主要考点
1	展厅最优路线	网格动态规划、最优路径计数
2	研学路线最短时间	图、Dijkstra 最短路
3	校园网络省线	最小生成树、Kruskal、并查集
4	双人研学采分	二维动态规划、双人同步网格路径

第 1 题 展厅最优路线

主要考点： 网格动态规划、最优路径计数

题意概括

从网格左上角出发，每次只能向右或向下走到右下角。需要同时求出路径能够获得的最大分值，以及取得该最大分值的路径条数。

核心思路

设 dp[i][j] 为走到格子 (i,j) 的最大分值，cnt[i][j] 为取得该最大分值的路径数。每个格子只可能从上方或左方到达：若某个前驱产生了更大的分值，就替换最大值并继承该前驱的方案数；若产生的分值相同，就把方案数累加。分值可能为负，因此未到达状态要初始化为足够小的负数。

参考代码

```
#include <bits/stdc++.h>
using namespace std;
const long long NEG = -(1LL << 60);
const long long MOD = 1000000007LL;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<vector<int>> a(n, vector<int>(m));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j) cin >> a[i][j];
    vector<vector<long long>> dp(n, vector<long long>(m, NEG));
    vector<vector<long long>> cnt(n, vector<long long>(m, 0));
    // 起点只有一种到达方式，并且起点分值需要计入答案。
    dp[0][0] = a[0][0];
    cnt[0][0] = 1;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (i == 0 && j == 0) continue;
            vector<pair<int, int>> pre = {{i - 1, j}, {i, j - 1}};
            for (auto [x, y] : pre) {
                if (x < 0 || y < 0) continue;
                long long val = dp[x][y] + a[i][j];
                if (val > dp[i][j]) {
                    // 出现更优路径时替换最优值，并继承其方案数。
                    dp[i][j] = val;
                    cnt[i][j] = cnt[x][y];
                } else if (val == dp[i][j]) {
                    // 两个前驱同样最优时，方案数需要相加。
                    cnt[i][j] = (cnt[i][j] + cnt[x][y]) % MOD;
                }
            }
        }
    }
    cout << dp[n - 1][m - 1] << ' ' << cnt[n - 1][m - 1] << '\n';
    return 0;
}
```

第 2 题 研学路线最短时间

主要考点： 图、Dijkstra 最短路

题意概括

给定一张带正权的连通无向图，求从起点 s 到终点 t 的最短通行时间。

核心思路

使用 Dijkstra 算法。dist[u] 记录目前已知的最短距离，小根堆每次取出距离最小的候选节点，再尝试用它松弛相邻边。一个节点可能多次进入堆；弹出时若堆中的距离已经不是 dist[u]，说明它是过期状态，直接跳过。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    vector<vector<pair<int, int>>> g(n + 1);
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        g[u].push_back({v, w});
        g[v].push_back({u, w});
    }
    const long long INF = (1LL << 62);
    vector<long long> dist(n + 1, INF);
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> pq;
    dist[s] = 0;
    pq.push({0, s});
    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();
        // 同一节点可能多次入堆，只处理与当前最短距离一致的状态。
        if (d != dist[u]) continue;
        for (auto [v, w] : g[u]) {
            if (dist[v] > d + w) {
                // 通过 u 到达 v 更短时进行松弛，并把新状态入堆。
                dist[v] = d + w;
                pq.push({dist[v], v});
            }
        }
    }
    cout << dist[t] << '\n';
    return 0;
}
```

第 3 题 校园网络省线

主要考点： 最小生成树、Kruskal、并查集

题意概括

从现有网线中保留一部分，使所有机房仍然连通且保留网线的总费用最小。输出全部网线费用与最小保留费用之差；若无法连通所有机房则输出 impossible。

核心思路

最大节省费用等于所有边的费用总和减去最小生成树费用。使用 Kruskal 算法把边按费用升序处理，并用并查集判断一条边连接的两个端点是否已经连通；只有连接两个不同连通块的边才加入生成树。最终若选中的边数不是 n-1，说明原图不连通。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

struct Edge {
    int u, v;
    long long w;
};

struct DSU {
    vector<int> parent;
    DSU(int n) : parent(n + 1) { iota(parent.begin(), parent.end(), 0); }
    // 路径压缩使后续查询更快。
    int find(int x) { return parent[x] == x ? x : parent[x] = find(parent[x]); }
    bool unite(int a, int b) {
        a = find(a);
        b = find(b);
        if (a == b) return false;
        parent[a] = b;
        return true;
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    cin >> n >> m;
    vector<Edge> edges(m);
    long long total = 0;
    for (auto &e : edges) {
        cin >> e.u >> e.v >> e.w;
        total += e.w;
    }
    sort(edges.begin(), edges.end(), [](const Edge &a, const Edge &b) { return a.w < b.w; });
    DSU dsu(n);
    long long mst = 0;
    int used = 0;
    for (const auto &e : edges) {
        // 只选择连接两个不同连通块的边，避免形成环。
        if (dsu.unite(e.u, e.v)) {
            mst += e.w;
            ++used;
        }
    }
    if (used != n - 1) cout << "impossible\n";
    else cout << total - mst << '\n';
    return 0;
}
```

第 4 题 双人研学采分

主要考点： 二维动态规划、双人同步网格路径

题意概括

两名同学同时从网格左上角走到右下角，每一步都只能向右或向下。要求最大化两条路径获得的积分总和；两人在同一时刻到达同一格时，该格积分只能计算一次。

核心思路

两人每一步同时移动，因此经过相同步数后，两人的“行号加列号”相同。按同步步数分层动态规划，只需用 dp[r1][r2] 记录两人当前行号；当前列号可以由步数减行号得到。两人各自的上一步可能来自左边或上边，共有四种前驱组合。转移时若两人当前位置相同只加一次分值，否则加两格分值。

参考代码

```
#include <bits/stdc++.h>
using namespace std;

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

    int n;
    cin >> n;
    vector<vector<int>> a(n + 1, vector<int>(n + 1));
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) cin >> a[i][j];
    }

    const long long NEG = -(1LL << 60);
    vector<vector<long long>> dp(n + 1, vector<long long>(n + 1, NEG));
    // step = 行号 + 列号；起点 (1,1) 对应 step=2。
    dp[1][1] = a[1][1];

    for (int step = 3; step <= 2 * n; ++step) {
        vector<vector<long long>> ndp(n + 1, vector<long long>(n + 1, NEG));
        for (int r1 = 1; r1 <= n; ++r1) {
            // 同一步数下，列号可以由 step - 行号唯一确定。
            int c1 = step - r1;
            if (c1 < 1 || c1 > n) continue;
            for (int r2 = 1; r2 <= n; ++r2) {
                int c2 = step - r2;
                if (c2 < 1 || c2 > n) continue;

                long long best = NEG;
                // 两人上一步各可能来自左侧或上方，共四种组合。
                for (int pr1 : {r1, r1 - 1}) {
                    for (int pr2 : {r2, r2 - 1}) {
                        if (pr1 >= 1 && pr2 >= 1) best = max(best, dp[pr1][pr2]);
                    }
                }
                if (best == NEG) continue;

                long long gain = a[r1][c1];
                // 两人落在同一格时，该格积分只能计算一次。
                if (r1 != r2 || c1 != c2) gain += a[r2][c2];
                ndp[r1][r2] = best + gain;
            }
        }
        dp.swap(ndp);
    }

    cout << dp[n][n] << '\n';
    return 0;
}
```