6-1　最小长度电路板排列问题。

问题描述：最小长度电路板排列问题是大规模电子系统设计中提出的实际问题。该问题的提法是，将 n 块电路板以最佳排列方案插入带有 n 个插槽的机箱中。n 块电路板的不同的排列方式对应不同的电路板插入方案。

设 $B=\{1, 2, \cdots, n\}$ 是 n 块电路板的集合。集合 $L=\{N_1, N_2, \cdots, N_m\}$ 是 n 块电路板的 m 个连接块。其中每个连接块 N_i 是 B 的一个子集，且 N_i 中的电路板用同一根导线连接在一起。在最小长度电路板排列问题中，连接块的长度是指该连接块中第 1 块电路板到最后 1 块电路板之间的距离。例如，设 n=8，m=5，给定 n 块电路板及其 m 个连接块如下：

$$B=\{1, 2, 3, 4, 5, 6, 7, 8\};\qquad L=\{N_1, N_2, N_3, N_4, N_5\}$$

$$N_1=\{4, 5, 6\};\quad N_2=\{2, 3\};\quad N_3=\{1, 3\};\quad N_4=\{3, 6\};\quad N_5=\{7, 8\}$$

这 8 块电路板的一个可能的排列如图 6-1 所示。

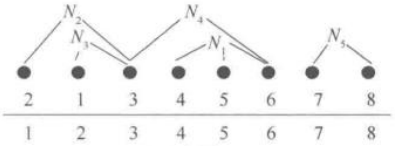

图 6-1　最小长度电路板排列

试设计一个队列式分支限界法找出所给 n 个电路板的最佳排列，使得 m 个连接块中最大长度达到最小。

算法设计：对于给定的电路板连接块，设计一个队列式分支限界法，找出所给 n 个电路板的最佳排列，使得 m 个连接块中最大长度达到最小。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m（$1 \leqslant m, n \leqslant 20$）。接下来的 n 行中，每行有 m 个数。第 k 行的第 j 个数为 0 表示电路板 k 不在连接块 j 中，为 1 表示电路板 k 在连接块 j 中。

结果输出：将计算的电路板排列最小长度及其最佳排列输出到文件 output.txt。文件的第 1 行是最小长度；接下来的 1 行是最佳排列。

输入文件示例	输出文件示例
input.txt	output.txt
8 5	4
1 1 1 1 1	5 4 3 1 6 2 8 7
0 1 0 1 0	
0 1 1 1 0	
1 0 1 1 0	
1 0 1 0 0	
1 1 0 1 0	
0 0 0 0 1	
0 1 0 0 1	

答：**算法思路：**采用队列式分支限界法，定义包含当前排列、电路板使用标记、连接块首尾位置及当前最大长度的结点。初始化空排列结点入队，循环取队首结点：若排列长度达 n，检查是否更新最优解；否则，对每个未用电路板生成新结点，更新排列与使用标记，调整其所属连接块的首尾位置，计算新的最大长度，若小于当前最优值则将新结点入队。持续扩展直至队列为空，最终输出最小长度及对应最佳排列，以此高效搜索所有可能排列，找到使连接块最大长度最小的最优解。

队列元素结构:

```cpp
struct Node {
    vector<int> perm;       // 当前排列
    vector<bool> used;      // 标记电路板是否已使用
    vector<int> first;      // 各连接块的第一个电路板位置
    vector<int> last;       // 各连接块的最后一个电路板位置
    int maxLen;             // 当前最大长度
    Node(int n, int m) : used(n + 1, false), first(m + 1, 0), last(m + 1, 0), maxLen(0) {}
};
```

队列式分支界限实现

```cpp
while (!q.empty()) {
    Node curr = q.front();
    q.pop();

    if (curr.perm.size() == n) {
        if (curr.maxLen < minLen) {
            minLen = curr.maxLen;
            bestPerm = curr.perm;
        }
        continue;
    }

    for (int k = 1; k <= n; ++k) {
        if (curr.used[k]) continue;
        Node next = curr;
        next.perm.push_back(k);
        next.used[k] = true;
        int newPos = next.perm.size();
        for (int j = 1; j <= m; ++j) {
            if (block[k][j] == 1) {
                if (next.first[j] == 0) {
                    next.first[j] = newPos;
                }
                next.last[j] = newPos;
            }
        }
        int newMax = 0;
        for (int j = 1; j <= m; ++j) {
            if (next.first[j] != 0) {
                int len = next.last[j] - next.first[j];
                if (len > newMax) {
                    newMax = len;
                }
            }
        }
        next.maxLen = newMax;
        if (newMax < minLen) {
            q.push(next);
        }
    }
}
```

6-2　最小权顶点覆盖问题。

问题描述：给定一个赋权无向图 $G=(V, E)$，每个顶点 $v \in V$ 都有权值 $w(v)$。如果 $U \subseteq V$，且对任意 $(u, v) \in E$ 有 $u \in U$ 或 $v \in U$，就称 U 为图 G 的一个顶点覆盖。G 的最小权顶点覆盖是指 G 中所含顶点权之和最小的顶点覆盖。

算法设计：对于给定的无向图 G，设计一个优先队列式分支限界法，计算 G 的最小权顶点覆盖。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m，表示给定的图 G 有 n 个顶点和 m 条边，顶点编号为 $1, 2, \cdots, n$。第 2 行有 n 个正整数表示 n 个顶点的权。接下来的 m 行中，每行有 2 个正整数 u 和 v，表示图 G 的一条边 (u, v)。

结果输出：将计算的最小权顶点覆盖的顶点权之和以及最优解输出到文件 output.txt。文件的第 1 行是最小权顶点覆盖顶点权之和；第 2 行是最优解 x_i（$1 \leqslant i \leqslant n$），$x_i=0$ 表示顶点 i 不在最小权顶点覆盖中，$x_i=1$ 表示顶点 i 在最小权顶点覆盖中。

输入文件示例	输出文件示例
input.txt	output.txt
7 7	13
1 100 1 1 1 100 10	1 0 1 1 0 0 1
1 6	
2 4	
2 5	
3 6	
4 5	
4 6	
6 7	

答：**算法思路：**首先，初始化一个起始结点，level 是 0，cost 是 0，x 全 0。然后计算它的下界。下界怎么算呢？对于还没处理的边，假设每次都选权值小的那个顶点来覆盖。比如，遍历所有未覆盖的边，把每条边两端顶点权值小的加起来，这样得到一个下界估计。

然后把起始结点放进优先队列。队列按照 cost + bound 从小到大排，这样每次取出来的都是当前看起来最有希望的结点。

接着循环处理队列里的结点。如果当前结点的 level 等于 n（顶点数），说明处理完所有顶点了，这时候比较权值和，更新最优解。

如果没处理完，就扩展下一个顶点。有两种选择：选这个顶点或者不选。

选的话，cost 加上这个顶点的权值，x 设为 1，计算新的下界，然后如果新的 cost + bound 比当前最小权值小，就加入队列。

不选的话，要检查这个顶点的所有邻接边是否已经被其他已选顶点覆盖。如果都覆盖了，那可以不选，计算下界，加入队列。如果有边没被覆盖，就不能不选，这个分支就剪掉。

这样不断扩展，直到队列为空，就能找到最小权的顶点覆盖了

定义节点结构：

```
struct Node {
    int cost;           // 当前权值和
    vector<int> x;      // 解向量
    int bound;          // 下界
    int level;          // 当前处理的顶点编号
    Node(int n) : x(n + 1, 0), level(0), cost(0), bound(0) {}
};
```

优先队列排序：

```
struct Compare {
    bool operator()(const Node& a, const Node& b) {
        return a.cost + a.bound > b.cost + b.bound;
    }
};
```

计算下界：

```
int calculateBound(Node& node) {
    int bound = node.cost;
    vector<bool> covered(n + 1, false);
    // 标记已覆盖的边
    for (int i = 1; i <= node.level; ++i) {
        if (node.x[i]) {
            for (int v : graph[i]) {
                covered[(i + v) / 2] = true; // 无向图，边可以用这种方式标记
            }
        }
    }
    // 计算未覆盖边的最小权值
    for (int i = 1; i <= m; ++i) {
        if (!covered[i]) {
            int u = graph[i][0], v = graph[i][1];
            bound += min(w[u], w[v]);
        }
    }
    return bound;
}
```

扩展节点:

```
while (!pq.empty()) {
    Node curr = pq.top();
    pq.pop();

    if (curr.level == n) {
        if (curr.cost < minCost) {
            minCost = curr.cost;
            bestX = curr.x;
        }
        continue;
    }

    int nextLevel = curr.level + 1;
    // 尝试选nextLevel
    Node selectNode = curr;
    selectNode.level = nextLevel;
    selectNode.x[nextLevel] = 1;
    selectNode.cost += w[nextLevel];
    selectNode.bound = calculateBound(selectNode);
    if (selectNode.cost + selectNode.bound < minCost) {
        pq.push(selectNode);
    }

    // 尝试不选nextLevel
    Node unselectNode = curr;
    unselectNode.level = nextLevel;
    unselectNode.x[nextLevel] = 0;
    // 检查是否有边未覆盖
    bool allCovered = true;
    for (int v : graph[nextLevel]) {
        if (!curr.x[v]) {
            allCovered = false;
            break;
        }
    }
    if (allCovered) {
        unselectNode.bound = calculateBound(unselectNode);
        if (unselectNode.cost + unselectNode.bound < minCost) {
            pq.push(unselectNode);
        }
    }
```

通过上述步骤，优先队列式分支限界法系统搜索顶点覆盖的可行解，利用下界剪枝减少无效搜索，最终找到权值和最小的顶点覆盖集合。

6-4　最小重量机器设计问题。

问题描述：设某一机器由 n 个部件组成，每种部件都可以从 m 个不同的供应商处购得。设 w_{ij} 是从供应商 j 处购得的部件 i 的重量，c_{ij} 是相应的价格。设计一个优先队列式分支限界法，给出总价格不超过 d 的最小重量机器设计。

算法设计：对于给定的机器部件重量和机器部件价格，设计一个优先队列式分支限界法，计算总价格不超过 d 的最小重量机器设计。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 3 个正整数 n、m 和 d。接下来的 $2n$ 行，每行 n 个数。前 n 行是 c，后 n 行是 w。

结果输出：将计算的最小重量，以及每个部件的供应商输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3 3 4	4
1 2 3	1 3 1
3 2 1	
2 2 2	
1 2 3	
3 2 1	
2 2 2	

答：**算法思路：**采用分支界限法

首先，定义节点结构。每个节点应该记录当前选到第几个部件（level），当前总价格（cost），当前总重量（weight），还有一个下界（bound），比如预估后面部件选最小重量的情况。然后优先队列按总重量 + 下界来排序，这样优先处理更有潜力的节点。

```
7   struct Node {
8       int level;
9       int cost;
10      int weight;
11      vector<int> supplier;
12      Node(int l, int c, int w, vector<int> s) : level(l), cost(c), weight(w), supplier(s) {}
13  };
14
15  struct Compare {
16      bool operator()(const Node& a, const Node& b) {
17          int boundA = a.weight + calculateBound(a.level + 1, a.supplier);
18          int boundB = b.weight + calculateBound(b.level + 1, b.supplier);
19          return boundA > boundB;
20      }
21  };
```

然后，初始化根节点，level 为 0，cost 和 weight 都是 0。然后开始扩展节点。对于每个节点，遍历 m 个供应商，看选这个供应商的部件是否符合条件。比如，新的 cost = 当前 cost + c_ij，如果超过 d 就跳过。否则，新的 weight = 当前 weight + w_ij。然后计算这个新节点的下界，比如后面每个部件都选最轻的重量，这样 bound 就是当前 weight + 后面部件最轻重量的和。把这个新节点加入优先队列。

```cpp
vector<vector<int>> c, w;
int n, m, d;

int calculateBound(int start, const vector<int>& sup) {
    int bound = 0;
    for (int i = start; i <= n; ++i) {
        int minW = INT_MAX;
        for (int j = 0; j < m; ++j) {
            if (i - 1 < sup.size() || j + 1 != sup[i - 1]) {
                minW = min(minW, w[i - 1][j]);
            }
        }
        bound += minW;
    }
    return bound;
}
```

```cpp
void minWeightMachine() {
    priority_queue<Node, vector<Node>, Compare> pq;
    vector<int> emptySup;
    Node start(0, 0, 0, emptySup);
    pq.push(start);
    int minWeight = INT_MAX;
    vector<int> bestSup;

    while (!pq.empty()) {
        Node curr = pq.top();
        pq.pop();

        if (curr.level == n) {
            if (curr.cost <= d && curr.weight < minWeight) {
                minWeight = curr.weight;
                bestSup = curr.supplier;
            }
            continue;
        }

        int nextLevel = curr.level + 1;
        for (int j = 0; j < m; ++j) {
            int newCost = curr.cost + c[nextLevel - 1][j];
            if (newCost > d) continue;
            int newWeight = curr.weight + w[nextLevel - 1][j];
            vector<int> newSup = curr.supplier;
            newSup.push_back(j + 1);
            Node newNode(nextLevel, newCost, newWeight, newSup);
            pq.push(newNode);
        }
    }

    if (minWeight != INT_MAX) {
        cout << minWeight << endl;
        for (int s : bestSup) {
            cout << s << " ";
        }
        cout << endl;
    }
    else {
        cout << "No solution" << endl;
    }
}
```

在扩展过程中，记录遇到的最小重量。当 level 达到 n 时，说明选完了所有部件，检查总价格是否不超过 d，如果是，就看重量是否更小，更新最小重量和对应的供应商选择。

```
if (curr.level == n) {
    if (curr.cost <= d && curr.weight < minWeight) {
        minWeight = curr.weight;
        bestSup = curr.supplier;
    }
    continue;
}
```

6-5 运动员最佳配对问题。

问题描述：羽毛球队有男女运动员各 n 人。给定 2 个 $n \times n$ 矩阵 $\boldsymbol{P}$ 和 $\boldsymbol{Q}$。$P[i][j]$是男运动员 i 和女运动员 j 配对组成混合双打的男运动员竞赛优势；$Q[i][j]$是女运动员 i 和男运动员 j 配合的女运动员竞赛优势。由于技术配合和心理状态等因素影响，$P[i][j]$不一定等于 $Q[j][i]$。男运动员 i 和女运动员 j 配对组成混合双打的男女双方竞赛优势为 $P[i][j] \times Q[j][i]$。设计一个算法，计算男女运动员最佳配对法，使各组男女双方竞赛优势的总和达到最大。

算法设计：设计一个优先队列式分支限界法，对于给定的男女运动员竞赛优势，计算男女运动员最佳配对法，使各组男女双方竞赛优势的总和达到最大。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 1 个正整数 n（$1 \leqslant n \leqslant 20$）。接下来的 $2n$ 行，每行 n 个数。前 n 行是 p，后 n 行是 q。

结果输出：将计算的男女双方竞赛优势的总和的最大值输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
3	52
10 2 3	
2 3 4	
3 4 5	
2 2 2	
3 5 3	
4 5 1	

答：**算法思路：**采用优先队列式分支限界法来解决

先定义节点，每个节点记录当前配对到第几个男运动员（level）、当前的优势总和（sum）、女运动员是否已配对（used），还要算一个边界值（bound），这个 bound 是预估后面还能获得的最大优势总和。计算 bound 的时候，对没配对的女运动员，在剩下的男运动员里找最大的 P [i][j]×Q [j][i] 累加上去。

然后把初始节点（level 0，sum 0，都没配对）放进优先队列，队列按 bound 从大到小排，这样每次取出来的节点都是最有潜力的。取节点后，如果已经配对完所有男运动员（level == n），就更新最大 sum。否则，对每个没配对的女运动员，计算新的 sum（当前 sum 加上 P [level][j]×Q [j][level]），标记女运动员已配对，生成新节点放进队列。这样不断扩展，直到队列为空，就能找到最大的优势总和了。

代码如下：

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct Node {
    int level;
    int sum;
    int bound;
    vector<bool> used;
    Node(int l, int s, vector<bool> u) : level(l), sum(s), used(u) {
        bound = s;
        for (int j = 0; j < used.size(); ++j) {
            if (!used[j]) {
                int max_pq = 0;
                for (int i = 1; i < used.size(); ++i) {
                    int val = p[i][j] * q[j][i];
                    if (val > max_pq) {
                        max_pq = val;
                    }
                }
                bound += max_pq;
            }
        }
    }
};

struct Compare {
    bool operator()(const Node& a, const Node& b) {
        return a.bound < b.bound;
    }
};

vector<vector<int>> p, q;
```

```cpp
int main() {
    int n;
    cin >> n;
    p.resize(n, vector<int>(n));
    q.resize(n, vector<int>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> p[i][j];
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> q[j][i]; // 注意这里q的输入，j是女，i是男
        }
    }

    priority_queue<Node, vector<Node>, Compare> pq;
    vector<bool> init_used(n, false);
    Node start(0, 0, init_used);
    pq.push(start);
    int max_sum = 0;

    while (!pq.empty()) {
        Node curr = pq.top();
        pq.pop();

        if (curr.level == n) {
            if (curr.sum > max_sum) {
                max_sum = curr.sum;
            }
            continue;
        }

        for (int j = 0; j < n; ++j) {
            if (!curr.used[j]) {
                int new_sum = curr.sum + p[curr.level][j] * q[j][curr.level];
                vector<bool> new_used = curr.used;
                new_used[j] = true;
                Node new_node(curr.level + 1, new_sum, new_used);
                pq.push(new_node);
            }
        }
    }

    cout << max_sum << endl;
    return 0;
}
```

6-10　世界名画陈列馆问题。

问题描述：世界名画陈列馆由 $m \times n$ 个排列成矩形阵列的陈列室组成。为了防止名画被盗，需要在陈列室中设置警卫机器人哨位。除了监视所在的陈列室，每个警卫机器人还可以监视与它所在的陈列室相邻的上、下、左、右 4 个陈列室。试设计一个安排警卫机器人哨位的算法，使名画陈列馆中每个陈列室都在警卫机器人的监视下，且所用的警卫机器人数最少。

算法设计：设计一个优先队列式分支限界法，计算警卫机器人的最佳哨位安排，使名画陈列馆中每个陈列室都在警卫机器人的监视下，且所用的警卫机器人数最少。

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 m 和 n（$1 \leqslant m, n \leqslant 20$）。

结果输出：将计算的警卫机器人数及其最佳哨位安排输出到文件 output.txt。文件的第 1 行是警卫机器人数；接下来的 m 行中每行 n 个数，0 表示无哨位，1 表示有哨位。

输入文件示例	输出文件示例
input.txt	output.txt
4 4	4
	0 0 1 0
	1 0 0 0
	0 0 0 1
	0 1 0 0

答：**算法思路：**采用优先队列式分支限界法来解决

先定义节点，每个节点记录当前监视状态 monitor 和已用机器人数量 count。优先队列按 count 从小到大排，这样先处理机器人少的情况。然后，对每个节点检查是否全部陈列室被监视。如果是，就看是否更新最少数量和最佳安排。如果不是，找到第一个没被监视的陈列室 (i,j)，生成新节点：在那放机器人，更新周围监视状态，count 加 1，加入队列。这样不断扩展，直到找到最优解。

代码如下：

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

struct Node {
    vector<vector<bool>> monitor;
    int count;
    Node(int m, int n) : monitor(m, vector<bool>(n, false)), count(0) {}
};

struct Compare {
    bool operator()(const Node& a, const Node& b) {
        return a.count > b.count;
    }
};

bool isAllMonitored(const Node& node) {
    for (const auto& row : node.monitor) {
        if (find(row.begin(), row.end(), false) != row.end()) {
            return false;
        }
    }
    return true;
}

void updateMonitor(Node& node, int i, int j, int m, int n) {
    if (i >= 0 && i < m && j >= 0 && j < n) node.monitor[i][j] = true;
    if (i - 1 >= 0 && j >= 0 && j < n) node.monitor[i - 1][j] = true;
    if (i + 1 < m && j >= 0 && j < n) node.monitor[i + 1][j] = true;
    if (i >= 0 && j - 1 >= 0 && j < n) node.monitor[i][j - 1] = true;
    if (i >= 0 && j + 1 < n) node.monitor[i][j + 1] = true;
}

int findFirstUnmonitored(const Node& node, int m, int n) {
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (!node.monitor[i][j]) {
                return i * n + j;
            }
        }
    }
    return -1;
}
```

```
47  void solve() {
48      int m, n;
49      cin >> m >> n;
50      priority_queue<Node, vector<Node>, Compare> pq;
51      Node start(m, n);
52      pq.push(start);
53      int minCount = m * n;
54      vector<vector<bool>> bestArrangement(m, vector<bool>(n, false));
55
56      while (!pq.empty()) {
57          Node curr = pq.top();
58          pq.pop();
59
60          if (isAllMonitored(curr)) {
61              if (curr.count < minCount) {
62                  minCount = curr.count;
63                  bestArrangement = curr.monitor;
64              }
65              continue;
66          }
67
68          int pos = findFirstUnmonitored(curr, m, n);
69          if (pos == -1) continue;
70          int i = pos / n;
71          int j = pos % n;
72
73          Node newNode = curr;
74          newNode.count++;
75          updateMonitor(newNode, i, j, m, n);
76          pq.push(newNode);
77      }
78
79      cout << minCount << endl;
80      for (const auto& row : bestArrangement) {
81          for (bool b : row) {
82              cout << b;
83          }
84          cout << endl;
85      }
86  }
87
88  int main() {
89      solve();
90      return 0;
91  }
```