算法分析第六次作业

算法实现题

6-1 题答案：

答：

电路板排列问题的解空间是一颗排列树。采用最小堆表示活节点优先级队列。最小堆中元素类型是 BoradNode，每一个 BoardNode 类型的节点包含域 x，表示节点所相应的电路板排列；s 表示该节点已确定的电路板排列 x[1:s]；cd 表示当前密度，now[j]表示 x[1:s]中所含连接块 j 中的电路板数。

将排列树的根结点置为当前扩展结点。在 do while 循环体内算法依次从活结点优先队列中取出具有最小 cd 值的结点作为当前扩展结点，并加以扩展。算法将当前扩展节点分两种情形处理：

(1.考虑 s=n-1，当前扩展结点是排列树中的一个叶结点的父结点。x 表示相应于该叶结点的电路板排列。计算出与 x 相应的密度并在必要时更新当前最优值和相应的当前最优解。

(2.s<n-1 时，算法依次产生当前扩展结点的所有儿子结点。对于当前扩展结点的每一个儿子结点 node,计算出其相应的密度 node.cd。当 node.cd<bestd 时，将该儿子结点 N 插入到活结点优先队列中。

```
template<typename T>
class MinHeap {
public:
    void push(const T& value);
    void pop();
    const T& top() const { return data[0]; }
    bool empty() const { return data.empty(); }
    size_t size() const { return data.size(); }

private:
    std::vector<T> data;
    void heapifyUp(size_t index);
    void heapifyDown(size_t index);
};

template<typename T>
void MinHeap<T>::push(const T& value) {
    data.push_back(value);
    heapifyUp(data.size() - 1);
}

template<typename T>
```

```
void MinHeap<T>::pop() {
    if (data.empty()) return;
    data[0] = data.back();
    data.pop_back();
    heapifyDown(0);
}

template<typename T>
void MinHeap<T>::heapifyUp(size_t index) {
    while (index > 0) {
        size_t parent = (index - 1) / 2;
        if (data[index] >= data[parent]) break;
        std::swap(data[index], data[parent]);
        index = parent;
    }
}

template<typename T>
void MinHeap<T>::heapifyDown(size_t index) {
    size_t left, right, minIndex;
    while (true) {
        left = 2 * index + 1;
        right = 2 * index + 2;
        minIndex = index;

        if (left < data.size() && data[left] < data[minIndex])
            minIndex = left;
        if (right < data.size() && data[right] < data[minIndex])
            minIndex = right;

        if (minIndex == index) break;
        std::swap(data[index], data[minIndex]);
        index = minIndex;
    }
}
```

6-2 题答案：

答：

将所有点分成 U、V 两个集合。V 集合中的每个点，至少与一个 U 集合中的点直接相连。开辟一个数组 c，如果 c[j]==0，则表示 U 集合中没有任何一个顶点与其直接相连。

确定优先级。用权重作为优先级建立一个最小堆，从而实现优先队列。

无法使用界限函数来对右孩子进行约束，所以不经过判断，我们也要将右孩

子加入到队列中。将活结点加入队列时，要对点的优先级、结果向量以及 cover 数组进行更新。

```
void VC::BBVC()
{
    priority_queue<HeapNode> H(100000);
    HeapNode E; // 扩展结点
    E.x = new int [n+1];
    E.c = new int [n+1];
    for(int j = 1;j <= n;j++)
    {
        E.x[j] = E.c[j] = 0;
    }
    int i = 1,cn = 0;
    while(true)
    {
        if(i > n)
        {
            if(cover(E))
            {
                for(int j = 1;j <= n;j++)
                    bestx[j]=E.x[j];
                bestn = cn;
                break;
            }
        }
        else
        {
            if(!cover(E))
                AddLiveNode(H,E,cn,i,1);
            AddLiveNode(H,E,cn,i,0);
        }
        if(H.size()==0)
            break;
        E = H.top();
        H.pop();
        cn = E.cn;
        i = E.i + 1;
    }
}
bool VC::cover(HeapNode E)
{
    for(int j = 1;j <= n;j++)
    {
        if(E.x[j]==0 && E.c[j]==0)
```

```
            {
                return false;
            }
        }
        return true;
    }
    void  VC::AddLiveNode(priority_queue<HeapNode>  &H,HeapNode  E,int
cn,int i,bool ch)
    {
        HeapNode N;
        N.x = new int [n+1];
        N.c = new int [n+1];
        for(int j = 1;j <= n;j++)
        {
            N.x[j] = E.x[j];
            N.c[j] = E.c[j];
        }
        N.x[i] = ch;
        if(ch)
        {
            N.cn = cn + w[i];
            for(int j = 1;j <= n;j++)
            {
                if(a[i][j])
                    N.c[j]++;
            }
        }
        else
            N.cn = cn;
        N.i = i;
        H.push(N);
    }
    int MinCover(int **a,int v[],int n)
    {
        VC Y;
        Y.w = new int [n+1];
        for(int j = 1;j <= n;j++)
            Y.w[j] = v[j];
        Y.a = a;
        Y.n = n;
        Y.bestx = v;
        Y.BBVC();
        return Y.bestn;
    }
```

6-4 题答案：

答：

采用优先队列式分支限界法求解，核心是在由部件选择构成的解空间树中，以当前总重量为优先级（最小堆）扩展节点，通过剪枝函数优化搜索。

每个节点记录当前总重量、总价格、部件序号、供应商序号及父节点指针，优先队列每次取出重量最小的节点处理：若为叶节点且满足价格约束则更新最优解；否则遍历所有供应商生成子节点。

仅当子节点总价格≤预算且下界（当前重量 + 剩余部件最小重量和）优于当前最优解时才加入队列。通过预处理各部件最小重量并计算后缀和得到下界数组，剪枝，从最优叶节点回溯父指针还原各部件选择的供应商。

```
class MinWeight {
private:
    int n, m, d, bestw;
    vector<vector<int>> w, c;
    vector<int> lb, bestx;
public:
    void solve() {
        priority_queue<node> q;
        node root(0, 0, -1, 0, nullptr);
        q.push(root);
        bestw = INT_MAX;
        while (!q.empty()) {
            node curNode = q.top();
            q.pop();
            if (curNode.i == n-1) {
                if (curNode.cc <= d && curNode.cw < bestw) {
                    update_best(curNode);
                    bestw = curNode.cw;
                }
            } else {
                for (int j = 0; j < m; j++) {
                    int next_i = curNode.i + 1;
                    int new_cw = curNode.cw + w[next_i][j];
                    int new_cc = curNode.cc + c[next_i][j];
                    if (new_cc <= d && (new_cw + lb[next_i+1]) <
bestw) {
                        node  child(new_cw,  new_cc,  next_i,  j,
&curNode);
                        q.push(child);
                    }
                }
            }
```

```
        }

        if (bestw == INT_MAX) cout << "无解";
        else print_result();
    }
    void update_best(node& leaf) {
        for (node* t = &leaf; t->i != -1; t = t->parent) {
            bestx[t->i] = t->j + 1;
        }
    }
    void print_result() {
        cout << bestw << endl;
        for (int x : bestx) cout << x << " ";
    }
};
```

6-5 题答案：

答：

把上界函数定义为：剩下的未配对的女运动员（不考虑男运动员配对情况下）所能达到的优势最大值之和（记为 r）与当前配对已达到的优势（记为 sum）之和。创建一个最大值堆，用于表示活结点优先队列，队中每个结点的 sum 值是优先队列的优先级，算法计算出每个顶点的最大 sum 值，搜索到所搜索的排列数的叶子节点，算法结束，输出最大值。

```
file.open("input.txt",ios::in);
    file >> n;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            file >> P[i][j];
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            file >> Q[i][j];
        }
    }
    file.close();

while (Node->id != n )
    {
```

```
            for (int i = Node->id; i < n; i++)
            {
                node* nNode = new node();
                nNode->id = Node->id + 1;
                nNode->x = new int[n];
                for (int t = 0; t < n; t++)
                {
                    nNode->x[t] = Node->x[t];
                }
                nNode->x[Node->id] = Node->x[i];
                nNode->x[i] = Node->x[Node->id];
                nNode->sum = Node->sum + P[Node->id][nNode->x[Node->id]]
* Q[nNode->x[Node->id]][Node->id];
                nNode->r = Node->r - maxn[Node->id];
                nNode->up = nNode->sum + nNode->r;
                q.push(nNode);
            }
            if (!q.empty())
            {
                Node = q.top();
                q.pop();
            }
            else {
                tmp = 0;
                break;
            }
        }
        tmp = Node->sum;
```

6-10 题答案：

答：

创建一个优先队列，用于存储所有待探索的结点。每个结点代表一个部分解，从优先队列中取出一个结点，基于这个结点生成新的结点，每个新结点代表在前一个结点的基础上，为另一个未决定哨位的陈列室选择放置或者不放置警卫机器人。

限界：如果该结点不能导致一个有效解，则舍弃这个结点。

优先级：结点的优先级基于已放置的警卫机器人数和未被监视的陈列室数，优先探索可能性更高且使用警卫机器人更少的结点。

```
priority_queue<Node, vector<Node>, cmp> q;
```

```
Node init(Node node)
{
    memset(node.pu,0,sizeof(node.pu));
    memset(node.spy,0,sizeof(node.spy));
    node.i=1;node.j=1;
    node.k=0;node.t=0;
    for(int i=0;i<=n+1;i++)node.spy[i][0]=node.spy[i][m+1]=1;
    for(int i=0;i<=m+1;i++)node.spy[0][i]=node.spy[n+1][i]=1;
    return node;
}
void puta(Node p,int x,int y)
{
    Node node;
    node=init(node);
    node.i=p.i;
    node.j=p.j;
    node.k=p.k+1;
    node.t=p.t;
    memcpy(node.pu, p.pu, sizeof(p.pu));
    memcpy(node.spy, p.spy, sizeof(p.spy));
    node.pu[x][y]=1;
    for(int d=0;d<5;d++)
     {
         int xx=x+f[d][0];
         int yy=y+f[d][1];
         node.spy[xx][yy]++;
         if(node.spy[xx][yy]==1)
        {
            node.t++;
```

```
            }
        }
        while(node.i<=n&&node.spy[node.i][node.j])
        {
            node.j++;
            if(node.j>m)node.i++,node.j=1;
        }
        q.push(node);
        return;
    }
```