```cpp
/// Dinic(Samiul Vai)
#include<bits/stdc++.h>
using namespace std;
#define NODE 30000
#define EDGE 5000000
const long long inf = 2147383647;
int backup[NODE];
struct node {
    int x, y, next, cap, cost;
};

struct FLOW {
    int source, sink;
    int head[NODE];

    void clear() {
        e = 0;
        memset(head,-1,sizeof(head));
    }

    node edge[EDGE]; int e;

    void addEdge ( int u, int v, int cap ) {
        edge[e].x = u; edge[e].y = v; edge[e].cap = cap;
        edge[e].next = head[u]; head[u] = e; e++;
        edge[e].x = v; edge[e].y = u; edge[e].cap = 0;
        edge[e].next = head[v]; head[v] = e; e++;
    }

    int vis[NODE], q[NODE], now[NODE];

    bool bfs ( ) {
        memset ( vis, -1, sizeof vis );
        vis[source] = 0;
        int ini = 0, qend = 0;
        q[qend++] = source;

        while ( ini < qend && vis[sink] == -1 ) {
            int s = q[ini++];
            int i;
            for (i=head[s];i!=-1;i= edge[i].next){
                int t = edge[i].y;
                if ( vis[t] == -1 && edge[i].cap){
                    vis[t] = vis[s] + 1;
                    q[qend++] = t;
                }
            }
        }
        if ( vis[sink] != -1 ) return true;
        else return false;
    }

    int dfs ( int s, int f ) {
        if ( f == 0 ) return 0;
        if ( s == sink ) return f;
        for ( int &i=now[s];i!=-1;i=edge[i].next){
            int t = edge[i].y;
            if ( vis[s] + 1 != vis[t] ) continue;
            int pushed=dfs(t,min(f,edge[i].cap));
            if ( pushed ) {
                edge[i].cap -= pushed;
                edge[i^1].cap += pushed;
                return pushed;
            }
        }
        return 0;
    }

```

```
    int maxFlow ( int limit, int flow ) {
        int res = 0;
        while ( 1 ) {
            if ( flow == 0 ) break;
            if ( bfs () == false ) break;
            int i;
            for ( i=0;i<=limit;i++)now[i]= head[i];
            while (int pushed=dfs(source,flow ) ) {
                res += pushed; //Can overflow depending on Max Flow
                flow -= pushed;
            }
        }
        return res;
    }

}graph;

pair<int,int> dish[110];
int event[10100];
int main () {
    int n; scanf ( "%d", &n );

    int mx = -1;
    graph.clear();
    int x = 10000;

    graph.source = x + n + 1;
    graph.sink = x + n + 2;

    for(int i=1; i<=n; i++) {
        int a, b; scanf ( "%d %d", &a, &b );
        for(int j=a; j<=b-1; j++) {
            graph.addEdge( j, x + i, inf );
        }
    }

    for(int i=0; i<=x; i++) {
        graph.addEdge( graph.source, i, 1 );
    }

    int e = graph.e;
    memcpy( backup, graph.head, sizeof backup );

    int low = 0, high = 10000;
    while ( low <= high ) {
        int mid = ( low + high ) / 2;
        ///Reset graph
        for ( int i = 0; i < e; i += 2 ) {
            graph.edge[i].cap += graph.edge[i^1].cap;
            graph.edge[i^1].cap = 0;
        }

        graph.e = e;
        memcpy( graph.head, backup, sizeof backup );

        for(int i=1; i<=n; i++) {
            graph.addEdge( x + i, graph.sink, mid );
        }
        int need = mid * n;

        int f = graph.maxFlow( graph.sink, inf );

        if(f == need) low = mid + 1;
        else high = mid - 1;
    }
    printf ( "%d\n", (low-1) * n );
    return 0;
}
```