周赛 T1

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

const int N = 102;
int a[N];

int main(){
    int n, r;
    cin >> n >> r;
    int ans = 0;
    a[0]    = 0;
    int tmp = 0;

    for (int i = 1 ; i <= n ; ++i) {
        cin >> a[i];
        ans = max(ans, a[i] - a[i - 1]);
    }

    ans = max(ans, (r - a[n]) * 2);
    cout << ans << '\n';

    return 0;
}
```

送分题，简单讲讲正确性即可。

bonus: 如果删除 $a_i < r$ 的限制，那么应该如何解决

solution: 特判，对于后面的情况，可能开过头比往返要好，应当对于后面的情况单独讨论。

周赛 T2

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

const int N = 1e5 + 3;
int a[N];   // 题目输入
int f[N];   // 步数
bool v[N];  // 是否来过

int main(){
    int n,s,t;
    cin >> n >> s >> t;
    for (int i = 1 ; i <= n ; ++i) cin >> a[i];
    queue <int> q;
    q.push(s);
    v[s] = true;
    while (!q.empty()) {
        const int i = q.front();
        q.pop();
```

```
        if (i == t) {
            cout << f[i];
            return 0;
        }
        if (i - a[i] >= 1 && !v[i - a[i]]) {
            v[i - a[i]] = true;
            f[i - a[i]] = f[i] + 1;
            q.push(i - a[i]);
        }
        if (i + a[i] <= n && !v[i + a[i]]) {
            v[i + a[i]] = true;
            f[i + a[i]] = f[i] + 1;
            q.push(i + a[i]);
        }
    }
    cout << -1;
    return 0;
}
```

题目描述略不清楚，结合样例输入解释。

首先，结合题目，分析是找最短步数问题。这种情况下，采用 bfs 的方法，每次向外去尝试拓展一步。显然，如果能走且没来过，那么一定是当前最优解 （即晚来一定不如先来）。

在写 bfs 的时候，需要注意边界条件，比如越界不能走，且访问过不能走（可以演示下如果不写，会出现什么的问题，样例 3 超时）。

思考边界：起点和终点一样的情况。不过按照当前的写法不会有问题。

周赛 T3

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

const int M = 510;
const int N = 1100;

const int dir[2][12] = {
    { 1 , 1 , 2 , 2 , -1, -1, -2, -2 , 2 , 2 , -2, -2},
    { 2 , -2, 1 , -1, 2 , -2, 1 , -1 , 2 , -2, 2 , -2}
};

int f[N][N];
int v[N][N];
struct point { int x,y; };

int main() {
    int a,b;
    cin >> a >> b;
    a += M; b += M;

    queue <point> q;
    v[M][M] = true;
    q.push({M, M});
```

```
    while (!q.empty()) {
        point cur = q.front();
        q.pop();
        int x = cur.x , y = cur.y;
        if (x == a && y == b) {
            std::cout << f[x][y];
            return 0;
        }
        for (int t = 0 ; t < 12 ; ++t) {
            int tx = x + dir[0][t];
            int ty = y + dir[1][t];
            if (v[tx][ty]) continue;
            if (tx < 0 || ty < 0 || tx >= N || ty >= N) continue;
            q.push({tx,ty});
            v[tx][ty] = true;
            f[tx][ty] = f[x][y] + 1;
        }
    }

    // 不可能无法到达!
    return 0;
}
```

显然，这是一个二维的搜索问题。

对于移动方向数组，其不再只有 4 个方向了，需要一个 2×12 的打表。

负数坐标？（加一个大数，变成正整数即可）。

（提问，对于一个无穷边界的问题，我们转化为有约束的问题，是否正确？

虽然是一个无边界的问题，但其实不用担心走到很远的情况。

首先，只需要走“日”字，就可以在坐标绝对值变化不超过 3 的情况下，走到相邻的格子。所以，事实上，我们可以证明，基本上调整不会超过目标周围 3 的范围。同时，这也顺便证明了一定是有解的，因此不用特判无解的情况。

大方向是对的。

阶段测试 T1

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

const int N = 310;
const int dir[2][4] = {
```

```
    {1 , -1 , 0 , 0},
    {0 , 0  , 1 ,-1}
};

int c[N][N];
int step[N][N];
bool vis[N][N];
struct point { int x,y; };

int main() {
    const int INF = 100000000;
    int m;
    cin >> m;
    for (int i = 0 ; i < N ; ++i)
        for (int j = 0 ; j < N ; ++j)
            c[i][j] = INF; // 初始化为无穷大

    for (int i = 1,x,y,tt ; i <= m ; ++i) {
        cin >> x >> y >> tt;
        c[x][y] = min(c[x][y], tt);
        for (int t = 0 ; t < 4 ; ++t) {
            int tx = x + dir[0][t];
            int ty = y + dir[1][t];
            if (tx < 1 || ty < 1) continue;
            c[tx][ty] = min(c[tx][ty], tt);
        }
    }

    queue <point> q;
    if (c[1][1] > 0) { // 防止起点不安全
        q.push({1,1});
        step[1][1] = 0;
    }

    while (!q.empty()) {
        point cur = q.front();
        q.pop();
        vis[cur.x][cur.y] = true;
        int tt = step[cur.x][cur.y];    // 当前时间
        if (c[cur.x][cur.y] == INF) {   // 安全点
            cout << tt << '\n';
            return 0;
        }

        for (int t = 0 ; t < 4 ; ++t) {
            int tx = cur.x + dir[0][t];
            int ty = cur.y + dir[1][t];
            if (tx < 1 || ty < 1 || vis[tx][ty]) continue;
            if (tt + 1 < c[tx][ty]) { // 下一个时刻安全
                q.push({tx,ty});
                step[tx][ty] = tt + 1;
            }
        }
    }
    cout << -1 << '\n';
```

```
    return 0;
}
```

思路:

- 首先，讲题面，画图演示，格子标注安全的时间

	3;4	3;5	5	5			
	3;4	4	5				
	4						

- 然后，先不管是否可达的约束，直接写 bfs。要点：
  - 终止条件(是否安全的判断)
  - 边界判断(不能小于 1，1)
  - 记录步数
- 最后，加入 bfs 的路径约束。即当前步数的时候，必须是安全的。
  - 特别地，如果初始就是危险的，直接输出 -1.

阶段测试 T2

```
#include <iostream>
using namespace std;

const int N = 21;

bool v[N][N];
int cnt[N];

int n,m,ans,sum;
bool t[N];

bool check(int x) {
    for (int i = 1 ; i <= n ; ++i)
        if (v[x][i] && t[i]) return false;
    return true;
}

void dfs(int x,int c) {
    if (x > n) {
        if (c == m) ans = min(sum, ans);
        return;
    }
```

```
    if (check(x)) { // 可以选
        t[x] = true; ++sum;
        dfs(x + 1, c + cnt[x]);
        t[x] = false; --sum;
    }

    // 不选
    dfs(x + 1,c);
}


int main() {
    cin >> n >> m;
    ans = n + 1;
    for (int i = 1,x,y ; i <= m ; ++i) {
        cin >> x >> y;
        v[x][y] = v[y][x] = true;
        ++cnt[x];
        ++cnt[y];
    }
    dfs(1, 0);
    if (ans == n + 1)
        cout << "impossible\n";
    else
        cout << ans << '\n';
    return 0;
}
```

首先本题数据范围有误，应该是不超过 20 ， 10^4 的 需要其他的算法，超出了范围。

然后是题目本身。题目的看起来有点绕 （结合样例解释）。这题本质上想要说，我们有一些点，我们选出了一些点，其使得所有边都被覆盖，且没有相邻的点。

（图论中的最小点覆盖问题）

做法：dfs 枚举每个点的状态，最后检查是否所有的边都被覆盖，且没有相邻的边。

改进：二进制状态压缩 (?)

记忆化搜索

一些引入

- 记忆化搜索其实和递推非常类似，前者是反过来推测，后者是往前推测
- 时间复杂度？标记为已访问！
- 记忆化的作用就是为了避免重复计算。

某些题目，递推表达式不明显。这时候，就可以用记忆化搜索来解决。复杂度和递推一样。

顺序：

- 第一题 （模板）
- 最后一题 （模板）