# 爬山算法

# **1.概念**

爬山算法是一种局部择优的方法，是一种局部贪心的最优算法。  
采用启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。 该算法每次从当前解的临近解空间中选择一个最优解作为当前解，  
直到达到一个局部最优解，属于人工智能算法的一种。

# **2.主要思想**

1. 随机选择一个登山的起点；
2. 每次拿相邻点与当前点进行比对,取两者中较优者,作为爬坡的下一步；
3. 重复第2步，直至该点的邻近点中不再有比其大的点；
4. 选择该点作为本次爬山的顶点,即为该算法获得的最优解。

# **3.特点**

实现简单，其主要缺点是会陷入局部最优解，不一定能搜索到全局最优解。  
如下图所示：假设C点为当前解，爬山算法搜索到A点这个局部最优解就会停止搜索，  
因为在A点无论向那个方向小幅度移动都不能得到更优的解。

## 

## **POJ2420（求多边形费马点）**

题目：[题目链接](http://poj.org/problem?id=2420" \t "https://www.xuebuyuan.com/_blank)

题意：题目的意思就是给你N个点，在平面上寻找一个点，使得这个点到其他点的最小，问你最小的距离是多少？

分析：在三角形内部这个点叫做费马点（[费马点定义](https://baike.baidu.com/link?url=URRwOzZPti8XyrZLbv7WylgQPuRxatBhqGfOWNK8fXktS8Wg3F9AflnvCw9EJF3A" \t "https://www.xuebuyuan.com/_blank)）。那么这道题目就是求一个多变形的费马点。这样的话，我们可以像搜索那样，先定义一个起点，然后以这个点为基点分别向上、向下、向左、向右移动，只要使得距离之和变小的移动都是有效的移动。那么我们可以从第一点开始，（这种方法有点像上次亮神A的那道题目就是从三角形内部一点开始上下左右移动求最小的值）。那么结束的条件就是题目的卡的精度。这样的就是像搜索一样找就可以了。

代码：

#include <iostream>

#include <cstdio>

#include <string>

#include <string.h>

#include <map>

#include <vector>

#include <cstdlib>

#include <algorithm>

#include <cmath>

#include <queue>

#include <set>

#include <stack>

#include <functional>

#include <fstream>

#include <sstream>

#include <iomanip>

#include <numeric>

#include <cassert>

#include <bitset>

#include <stack>

#include <ctime>

#include <list>

#define INF 0x7fffffff

#define max3(a,b,c) (max(a,b)>c?max(a,b):c)

#define min3(a,b,c) (min(a,b)<c?min(a,b):c)

#define mem(a,b) memset(a,b,sizeof(a))

using namespace std;

#define maxn 200

struct point

{

double x, y;

} p[maxn];

int n;

point update(double x, double y)

{

point tmp;

tmp.x = x;

tmp.y = y;

return tmp;

}

double dis(point a, point b)//计算两点距离

{

return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y)\*(a.y - b.y));

}

double sumdis(point a)//所有点距离和

{

double ans = 0;

for(int i = 0; i < n; ++i)

ans += dis(a, p[i]);

return ans;

}

int main()

{

while(scanf("%d", &n) == 1)

{

for(int i = 0; i < n; ++i)

scanf("%lf%lf", &p[i].x, &p[i].y);

point pp = p[0];//初始位置

double ans = INF;

double step = 100;

ans = sumdis(pp);//计算距离和

while(step>0.2)//结束条件

{

bool flag=true;

while(flag)

{

flag = false;

point Q = update(pp.x, pp.y + step);//相邻位置

point tt = pp;

double temp = sumdis(Q);

if(temp < ans)//更优

{

ans = temp;

tt = Q;

flag = true;

}

Q = update(pp.x, pp.y - step);//相邻位置

temp = sumdis(Q);

if(temp < ans)

{

ans = temp;

tt = Q;

flag = true;

}

Q = update(pp.x + step, pp.y);//相邻位置

temp = sumdis(Q);

if(temp < ans)

{

ans = temp;

tt = Q;

flag = true;

}

Q = update(pp.x - step, pp.y);//相邻位置

temp = sumdis(Q);

if(temp < ans)

{

ans = temp;

tt = Q;

flag = true;

}

pp = tt;//更新位置

}

step /= 2.0;

}

int dis = (int)(ans + 0.5)\*100/100;

printf("%d\n", dis);

}

return 0;

}