题目描述

房间里放着 n 块奶酪。一只小老鼠要把它们都吃掉，问至少要跑多少距离？老鼠一开始在 (0,0) 点处。

输入格式

第一行有一个整数，表示奶酪的数量 n。

第 2 到第 (n+1)行，每行两个实数，第 (i+1) 行的实数分别表示第 i 块奶酪的横纵坐标 xi,yi。

输出格式

输出一行一个实数，表示要跑的最少距离，保留 2 位小数。

输入样例

```
4
1 1
1 -1
-1 1
-1 -1
```

输出样例

```
7.41
```

解析

在这道题目中，我们可以使用一种叫做状态压缩的技术，即用二进制表示奶酪是否被吃。假设当前共有4块奶酪，我们用0代表奶酪没有被吃，用1代表奶酪被吃，则以下图像代表不同的含义：

1000　第一块奶酪被吃

0100　第二块奶酪被吃

1100　前两块奶酪被吃

那么状态是如何转换的呢？假设前一种状态是0100，下一个状态是0110，即第三块奶酪从没有被吃变成了被吃，那么我们怎么从最后的状态推导出前一个状态是什么呢？

很简单，我们只需要用(0110 - 1<<(i-1)) 即可推出前一个状态0100，其中i代表奶酪的索引，在这里它的值为2，**(二进制数从右向左，以1作为起始计数)**。

本题中还存在另外一个状态值j，也就是小鼠的位置，我们同样可以用二进制来表示。例如1000就代表我处在第一个奶酪处，0100代表我处在第二个奶酪处……因为我们前一个状态可能来自于任何一个位置，所以我们在计算的时候就需要遍历所有位置的可能性。它的计算方式就很简单了，直接1<<(j-1)即可。

此外，还要注意前一个状态的来源肯定是奶酪被吃的状态，即前两个状态的与计算要大于0，表达式为(i&1<<(j-1)>0)。如下所示：

奶酪状态：1100
位置信息：1000
二者进行与计算后大于0，只有这样的数据才是一个合法的来源数据。

综上，我们定义dp[i][j],其中i代表小老鼠的位置，j代表奶酪的各种状态，则状态转移方程为：

dp[i][j] = min(dp[i][j],dis[k][i]+dp[k][j-1<<(i-1)])，其中k是所有能够到达当前位置的起始点。

最后，还剩下初始值的问题。在最初的时刻，小老鼠可能从0,0点到达任意一个位置，并把这个位置的奶酪吃掉，那么则有初始值：dp[i][1<<(i-1)] = dis(0,i)，站在第i个位置的老鼠把第i个奶酪吃掉，他走过的距离是0,0点到第i个点的位置。

编码

```
#include<bits/stdc++.h>

using namespace std;

//奶酪的数量
int n;
//最短距离
double ans;

//第一维度：奶酪的数量
//第二维度：奶酪的状态
double dp[20][1 << 16];
```

```
//奶酪的坐标
double x[20], y[20];

double dis(int p1, int p2) {
    double x1 = x[p1];
    double y1 = y[p1];
    double x2 = x[p2];
    double y2 = y[p2];
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

int main() {
    cin >> n;
    //将数组设置为最大
   memset(dp, 127, sizeof(dp));
    //读入n个坐标
   for (int i = 1; i <= n; i++) {
        scanf("%lf %lf", &x[i], &y[i]);
        //计算以第i个点作为起始点的时候，小老鼠已经走过的距离
      dp[i][1 << (i - 1)] = dis(i, 0);
    }
    //遍历奶酪的全部状态作为起点
   for (int j = 1; j < (1 << n); j++) {
        //遍历奶酪的所有位置,作为起点
       for (int i = 1; i <= n; i++) {
            //当前位置的二进制码
          int curPosition = 1 << (i - 1);
            //当前的奶酪状态和位置状态与计算后大于0，说明这个位置的奶酪被吃了，
          if ((j & curPosition) == 0) {
                continue;
            }
            //寻找可能的来源位置
          for (int k = 1; k <= n; k++) {
                //前一个点肯定不等于当前的点
             if (i == k) {
                    continue;
                }
                //当上一个点为k时的位置二进制编码
             int lastPosition = 1 << (k - 1);
                //在j这个状态下，k的位置的奶酪必然被吃掉才满足题意
             if ((j & lastPosition) == 0) {
                    continue;
                }
                //i点奶酪的状态，j减去这个值，就是前一个奶酪的状态
             int curState = 1 << (i - 1);
                //求最小值
             dp[i][j] = min(dp[i][j], dp[k][j - curState] + dis(i, k
            }
        }
```

```
    }

    //比较之前记录最大值
    ans = dp[0][0];
    //最终的奶酪状态一定是1111111，我们只需要判断不同的起点下最小值即可
    int state = (1 << n) - 1;
    for (int i = 1; i <= n; i++) {
        ans = min(ans, dp[i][state]);
    }
    printf("%.2f", ans);

    return 0;
}
```