该定向越野课程一共包括 12 个地标，从第一个地标出发，途中必须经过每一个地标，最终回到起始点来。12 个点分别在玉泉的平面地图中做出了标记，如下：
可以在百度地图中利用标尺工具测量出各点之间的距离如下（单位为米）：

	1	2	3	4	5	6	7	8	9	10	11	12
1（行政楼）	0	350	290	670	600	500	660	440	720	410	480	970
2（热力学楼）	350	0	340	360	280	375	555	490	785	760	700	1100
3（外经贸楼）	290	340	0	580	410	630	795	680	1030	695	780	1300
4（邵科馆）	670	360	580	0	260	380	610	805	870	1100	1000	1100
5（计算机学院）	600	280	410	260	0	610	780	735	1030	1000	960	1300
6（图书馆）	500	375	630	380	610	0	160	645	500	950	815	950
7（洁燃工程楼）	660	555	795	610	780	160	0	495	345	820	680	830
8（永谦剧场）	440	490	680	805	735	645	495	0	350	435	300	625
9（八舍）	720	785	1030	870	1030	500	345	350	0	475	320	485
10（1897 咖啡厅）	410	760	695	1100	1000	950	820	435	475	0	265	745
11（怡膳堂）	480	700	780	1000	960	815	680	300	320	265	0	585
12（北门）	970	1100	1300	1100	1300	950	830	625	485	745	585	0

注：1）由于学校里面还有其它建筑物，从某点到另外一点有多条距离，此处做了适当优化处理，即认为从一点到另一点的各种路径的长度都是相同的。2）路线是根据自己平时在学校里面行走的路线的来测量，可能还有许多较偏僻却较近的小路没有注意到。3）由于学校建在老和山下，道路存在一定坡度，这里取的都是平面距离，将坡度以适当的比例转换为水平距离。

三、模型建立

为了简化模型，不考虑其它过多的因素影响。在建模时忽略天气、温度等等外部条件，同时假设参加定向越野的同学对学校里面的环境较为熟悉，不会因为不认路或走错路的问题耽误时间。并且忽略在每个地点寻找游戏物品所消耗的时间。建模时只考虑路上所花去的时间。

（1） 目标函数的建立：

x_{ij} 表示是否从点 i 到点 j，取值为 0 或 1。

w_{ij} 表示从点 i 到点 j 的路线距离。

则目标函数是：$\min W=\sum_{i=0}^{n}\sum_{j=0}^{n}x_{ij}w_{ij}$

（2） 约束条件的建立：

由于要保证每一个点只能经过一次，且最后还要回到起点，使得整个路径闭合。（路径可能有交叉，所以不一定是圈）

$$s.t.\begin{cases}u_i-u_j+nx_{ij}\le n-1 & 1<i\ne j\le n \quad (1)\\ \sum_{\substack{i=1\\ i\ne j}}^{n}x_{ij}=1 & j=1,...,n \quad (2)\\ \sum_{\substack{j=1\\ i\ne j}}^{n}x_{ij}=1 & i=1,...,n \quad (3)\\ x_{ij}\in\{0,1\} & i,j=1,...,n \quad (4)\end{cases}$$

上式中 n=12。约束（1）保证了没有任何子回路解的产生；约束（2）和（3）保证了每个点有且只有一条路进和一条路出。

针对约束（1）的解释：

若两点 i 与 j 构成子回路，则：$x_{ij}=x_{ji}=1$，则有，$u_i-u_j\le -1, u_j-u_i\le -1$，从而，$0\le -2$ 显然不成立。若三点 i,j,k 构成子回路，则：$x_{ij}=x_{jk}=x_{ki}=1$，从而

$u_i-u_j\le -1, u_j-u_k\le -1, u_k-u_i\le -1$，即，$0\le -3$ 显然不成立。以此类推，则可保证路径中不存在子回路。

四、模型求解与算法分析

探究该定向越野活动的本质可知，这就是一个典型的 TSP 问题即旅行推销员问题。可用的解决方法包括动态规划法、分支定界法、贪心算法、模拟退火算法、遗传算法、蚁群算法 Hopfield 神经网络等等。本文选取其中多种算法对其进行分析比较。

1、 使用建立的简化模型通过 lingo 直接求解

代码如下：

```
MODEL:
SETS:
point/A1..A12/:u;
link(point,point):distance,x;
ENDSETS

DATA:
distance= 0,350,290,670,600,500,660,440,720,410,480,970,
          350,0,340,360,280,375,555,490,785,760,700,1100,
          290,340,0,580,410,630,795,680,1030,695,780,1300,
          670,360,580,0,260,380,610,805,870,1100,1000,1100,
          600,280,410,260,0,610,780,735,1030,1000,960,1300,
          500,375,630,380,610,0,160,645,500,950,815,950,
          660,555,795,610,780,160,0,495,345,820,680,830,
          440,490,680,805,735,645,495,0,350,435,300,625,
          720,785,1030,870,1030,500,345,350,0,475,320,485,
          410,760,695,1100,1000,950,820,435,475,0,265,745,
          480,700,780,1000,960,815,680,300,320,265,0,585,
          970,1100,1300,1100,1300,950,830,625,485,745,585,0;
ENDDATA

MIN=@SUM(link:distance*x);

@FOR(point(j):@SUM(point(i)|j#ne#i:x(i,j))=1);
@FOR(point(i):@SUM(point(j)|j#ne#i:x(i,j))=1);

@FOR(link(i,j)|i#ne#j#and#i#gt#1:u(i)-u(j)+12*x(i,j)<=11);
@FOR(link:@BIN(x));

END
```

结果：

```
Global optimal solution found.
Objective value:                              4140.000
Objective bound:                              4140.000
Infeasibilities:                              0.000000
Extended solver steps:                               0
Total solver iterations:                           585
```

以及 X(A1, A3)=1; X(A2, A5)=1; X(A3, A2)=1; X(A4, A6)=1; X(A5, A4)=1
X(A6, A7)=1; X(A7, A9)=1; X(A8, A11)=1; X(A9, A12)=1; X(A10, A1)=1
X(A11, A10)=1; X(A12, A8)=1;

可见，lingo 求解得到的最优路线为 1->3->2->5->4->6->7->9->12->8->11->10->1，总长为 4140m 路径如下所示：

2、 **模拟退火算法：**（程序代码见文末，tuihuo.py）

模拟退火（Simulated Annealing，SA）算法是求解 TSP 问题的有效方法之一，容易编程实现，求解速度快。模拟退火是一种全局优化算法，加入了随机状态模型，使算法以概率选择的方式逼近最优状态，而不会像贪心算法一样容易陷入局部最优解。

模拟退火解决 TSP 的思路：

0）随机产生一条路径 P(i),作为初始值。

1）产生一条新的遍历路径 P(i+1)，计算路径 P(i+1)的长度 L(P(i+1))

2）若 L(P(i+1)) < L(P(i))，则接受 P(i+1)为新的路径，否则以一定的概率接受 P(i+1) ,此步骤模拟退火过程，接受 P(I+1)之后，概率（即温度）会降低，同时越到最后降低速率越慢。

3）重复步骤 1），2）直到满足退出条件，输出最优值。

退火过程中产生新的遍历路径的方法有很多，下面列举其中 2 种：

（1）随机选择 2 个节点，交换路径中的这 2 个节点的顺序。

（2）随机选择 3 个节点 m，n，k，然后将节点 m 与 n 间的节点移位到节点 k 后面。

利用 python 编写程序并运行得到最优解：

1->3->2->5->4->6->7->9->12->8->11->10->1（或 1->10->11->8->12->9->7->6->4->5->2->3->1）

路径总长 4140m

结果如下所示：

```
1.0632818368521109
1.0526490184835897
1.0421225282987538
1.0317013030157662
1.0213842899856085
1.0111704470857523
1.0010587426148947
0.9910481551887458
最优路径:
[ 0  2  1  4  3  5  6  8 11  7 10  9]
最短距离:
4140
[Finished in 73.7s]
```

算法每次运行都能得到最优解，不容易陷入局部最优解。

3、 **蚁群算法**（程序代码附在文末 ant.py）

蚁群算法（Ant Colony Optimization，ACO）是根据蚂蚁发现路径的行为的而发明的用于寻求优化路径的机率型算法，由 Marco Dorigo 于 1992 年在他的博士论文中提出。蚁群算法是一种模拟进化算法，具有包括较强的鲁棒性在内的许多优良性质，在本质上是一种并行的分布式算法，容易实现，可以较好地求解 TSP 问题。

用蚁群算法来求解 TSP 问题主要包括以下几个步骤：

1）程序初始化，指定蚂蚁数目，迭代次数等，并为每只蚂蚁随机分配一个起始点。

2）通过轮盘法为每只蚂蚁选择下一个要到的路标，并将上一个节点加入本次迭代中该蚂蚁已走过的路标中，当所有路标都访问完之后，比较本次迭代中所有蚂蚁的路径成本，记录最佳路径。

3）更新信息素矩阵。根据每一条蚂蚁爬过的路线，更新每一段路线的信息素浓度。信息素浓度越高的路线，蚂蚁经过这条路线的可能性越大。且信息素会随着时间减少，所以路径越短，信息素残留的就越多。

4）检查终止条件，若没达到终止条件则进入下一次迭代。重复步骤 1），2），3）

5）输出最优值

利用 python 编写程序并运行得到最优解：

1->3->2->5->4->6->7->9->12->8->11->10->1（或 1->10->11->8->12->9->7->6->4->5->2->3->1）

路径总长 4140m
结果如下所示：

```
[ 5965.  5720.  5280.  5875.  4875.  5050.  5575.  4895.  5370.  4745.
  5850.  4960.]
the 147 iteration,all the ants distance are:
[ 5630.  6105.  4140.  5740.  4595.  5370.  5275.  5325.  5440.  5045.
  5290.  4540.]
the 148 iteration,all the ants distance are:
[ 5115.  5155.  5235.  5330.  5325.  5050.  5525.  4520.  5225.  5125.
  5635.  4790.]
the 149 iteration,all the ants distance are:
[ 4140.  6315.  5040.  4290.  6005.  4500.  5685.  5540.  5010.  4755.
  5245.  5060.]
最佳路径
[  7.  10.   9.   0.   2.   1.   4.   3.   5.   6.   8.  11.]
最短距离：4140
```

注：1.程序中数值起点是从 0 开始，所以实际坐标要在相应的结果上加 1，如 0 表示路标 1（行政楼），2 表示路标 3（外经贸楼）。2.由于在蚁群算法中是随机为每只蚂蚁指定起点，所以运行得到的结果大都不是从起点 0（即 1 号标志行政楼）出发。但这并不影响结果，只要该路径包含所有路标，并且是最优的，那么从任何一个路标出发，最后再回到该路标，这条路径都是最优路径。（下面的算法相同）
蚁群算法，也几乎每次都能找到最优解。

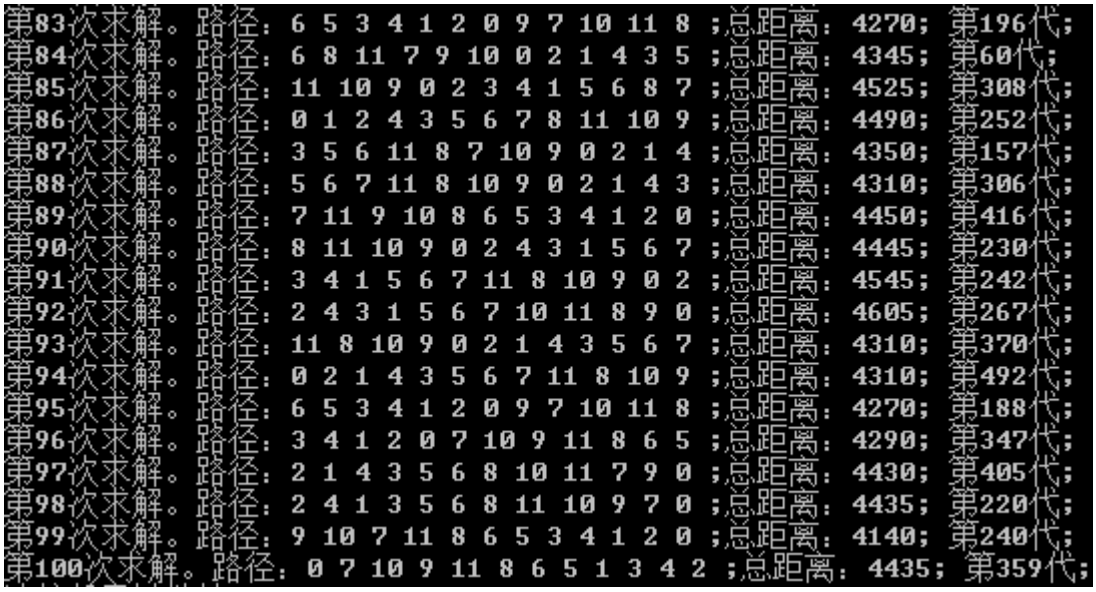
4、 **遗传算法：**（程序代码附在文末，yichuan.cpp）

遗传算法（Genetic Algorithm，GA）是一种常用的智能算法，具有全局搜索能力，对 TSP 问题有良好的效果。遗传算法是由 Michigan 大学的 J.Holland 教授于 1975 年提出，算法通常由编码、个体适应度评估和遗传运算三部分构成，其中遗传运算包括染色体复制、交叉、变异和倒位等

用遗传算法来求解 TSP 问题主要包括以下几个步骤：

1）初始化种群参数，包括种群个数、染色体基因个数（即路标个数）、迭代次数、交叉概率、变异概率等。
2）求解适应度函数，可将所有路标的某个随机全排列总距离的倒数作为适应度函数，即距离越短，适应度函数越好，满足 TSP 要求。
3）基因交叉，机选择两个个体，在对应位置交换若干个基因片段，同时保证每个个体依然是 1~n 的随机排列，防止进入局部收敛。
4）基因突变，随机选取某个体的两个基因进行交换以实现突变操作。
5）种群进化，根据适应度对个体进行排序，适应度越好的个体被选择的概率越大，保存适应度最高的个体。
6）检查终止条件，若没达到终止条件则进入下一次迭代，直到达到终止条件。

利用 C++编写程序并运行得到最优解如下：
1->3->2->5->4->6->7->9->12->8->11->10->1；
路径总长 4140m
程序某次运行结果：



可以看到第 99 次求解时，在第 240 代得到了最优解。但是程序输出并不稳定，可看出遗传算法收敛性能比较差。

5、 **Hopfield 神经网络**（代码附在文末，Hopfield.m）

1982 年，生物物理学家 J.Hopfield 提出了一种新颖的人工神经网络模型——Hopfield 网络模型，引入了能量函数的概念，这是一个非线性动力学系统。离散的 Hopfield 网络用于联想记忆；连续的 Hopfield 网络则可用于求解最优化问题。

将 TSP 的目标函数与网络的能量函数相对应，将经过地标的顺序与网络神经元的状态相对应。这样，根据连续 Hopfield 神经网络的稳定性理论可知，随着迭代次数的增加，该网络是逐渐趋于稳定的，网络的能量 E 随着减小的方向运动，其最终稳定状态就是 E 的最小值。当网络能量函数趋于最小值时，网络神经元状态也趋于平衡点，此时对应的地标顺序即为最佳路线。

应用 Hopfield 神经网络解决 TSP 问题的具体步骤如下：

1） 程序初始化，给定系数和权重，读入路标之间的间距。

2） 神经网络输入U_{xi}的初始化，$U_{xi}=0.5U_0\ln(N-1)+\delta_{xi}$；$U_0$是初始输入，N 是路标数量，$\delta_{xi}$是-1~1 间的随机值。

3） 计算动态方程$du=\dfrac{dU_{xi}}{dt}$

4） 输入神经元状态更新：$U_{xi}(t+1)=U_{xi}(t)+du*\Delta T$，其中$\Delta T$为给定步长。

5） 输出神经元状态更新:$V=0.5(1+\tanh(\dfrac{U_{xi}(t)}{U_0}))$，这里 V 即是最后要求得的最优路径矩阵。

6） 计算能量函数 E，与 V 有关，由目标函数和约束条件组成。

7） 根据 V 检查路径是否合法，若未达到迭代次数则返回步骤 3）。

8） 输出最优路径矩阵和路径长度。

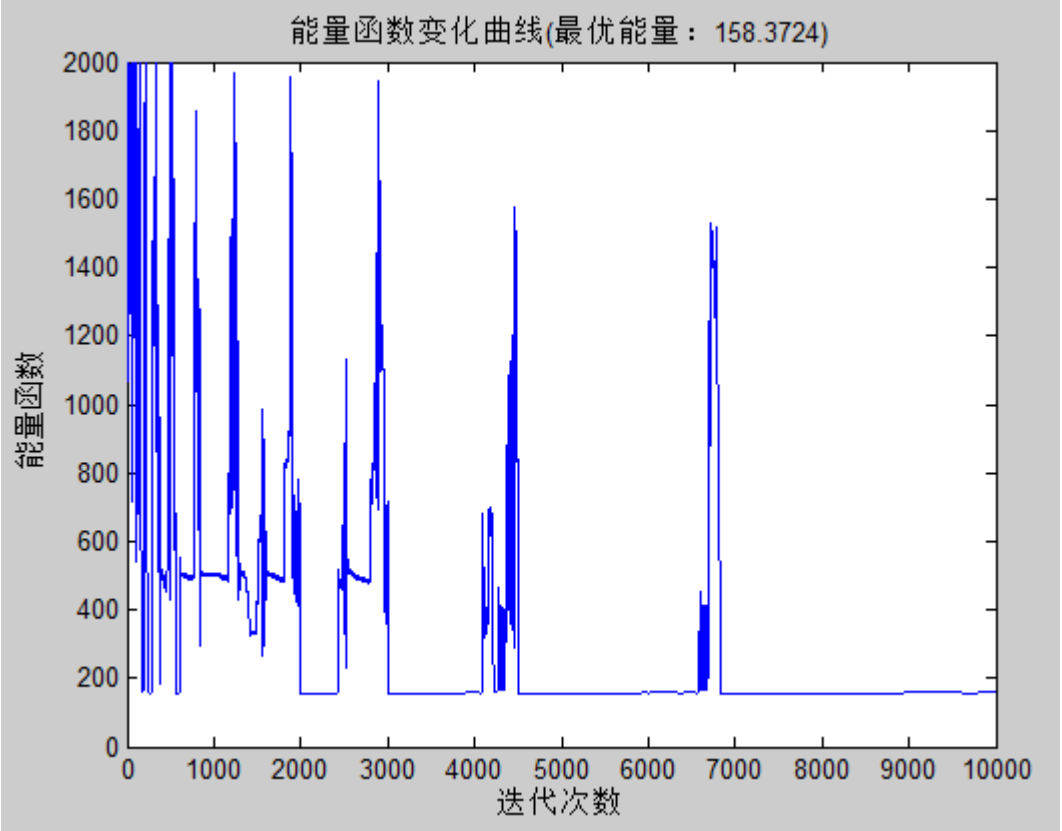
利用 Matlab 编写程序并多次运行得到最优解如下：

```
最短距离
Length_end =
        4140
最优路径矩阵
V1 =
     0     0     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     1     0
     1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     1
```

根据结果显示的路径为：

9->7->6->4->5->2->3->1->10->11->8->12->9（这正好是前面几种算法结果的逆向，本质相同）

能量函数如下：



遗憾的是需要运行多次才能得到最优解，可以看出这是一种贪心算法，故容易陷入局部最小值（玻尔兹曼机引入模拟退火，是其改进算法）。

而且能量函数会出现有陡增的情况，具体原因未知，猜测可能是由于求解能量函数的过程中产生了很小的数在分母的情况。

五、结果分析与算法比较

算法对比如下：

	求解速度	结果准确度	算法难易	结果收敛性
Lingo 求解	585 次迭代，很快	相当准确	简单	收敛性很好
模拟退火算法	75.9s	相当准确	简单	收敛性很好
蚁群算法	150 次迭代，3.4s	相当准确	较为简单	收敛性很好
遗传算法	100 次迭代，3.5s	不太准确	较为简单	收敛性一般
Hopfield 神经网络	10000 次迭代，较快	不太准确	较难	收敛性较差

比较上述五种求解方法，从运行速度、求解结果准确以及编程难易可知，当数据点较少的时候使用 lingo 求解 TSP 问题最为快捷，编程量也最少。模拟退火算法相比其它算法而言运行时间较长，但是结果准确。最差的是 Hopfield 神经网络，算法本身理解较为困难，而且结果难以收敛，很容易陷入局部最优解。

并且由于本文所考虑的数据点只有 12 个，如果数据点较多，则可能得到不一样的结果。并且对于每一种算法而言，改变算法参数，或者改变迭代次数，可能都会影响算法性能从而影响结果。比如 Hopfield 神经网络，虽然在本文中该算法性能较差，但这可能与程序中给定的模型参数有关，如果适当的调整参数，可能算法性能会提升不少。