算法需要的数据：

1. 起始景点到目标景点之间

所有（符合最佳日期的）**通路**（景点名称）。

1. 通路对应的**距离属性**
2. 兴趣点信息
   1. 这些通路上的相关景点的**兴趣点（已分级）**
   2. 相关景点上的游玩项目（名称）及**开销属性**
3. 重视度信息：
   1. **节奏属性**
   2. **预算属性**
4. 特殊点信息
   1. **黑名单**
   2. **必去**
   3. **已去过**

算法逻辑：

1. 首先对所有接收的通路进行属性入栈，即通路.距离，通路.开销入栈
2. 通路.距离，通路.开销分别出栈进行最短路径搜索，计算相关数据
3. 获取重视度信息（节奏，预算），控制评分权重
   1. ->节奏变量控制距离权重
   2. ->预算变量控制开销权重

两项权重总和为0.8。两项分为三档，一档最好。

若档数相同则各占0.4；若不同则根据所选档数分配，若选1，3则分别为0.6，0.2；选2，3或1，2则为0.5，0.3。

1. 根据评分权重计算，|距离<->开销|，的综合分数
2. 获取
   1. 兴趣点（标签）
   2. 特殊点（不想游玩的地区与项目）(最终也没有想到更好的解决方法)

满足兴趣点分数加1，若不满足兴趣点则分数减0.5。

1. 根据分级兴趣点配置权重

（ 兴趣点已进行排序 ）

1. 根据权重计算，兴趣点的增量分数，特殊点的减量分数
2. 最终生成计算结果

算法返回数据：

1. 所有通路都包含
   1. 路径，
   2. 花费，
   3. 景点属性
   4. 游玩项目
2. 综合得分最高的通路
3. 距离得分最高的通路
4. 开销得分最高的通路
5. 兴趣点与特殊点的增量得分最高的通路
6. 不包含不想去的特殊点的综合最高通路（如果有的话）

接口定义：

{

"trip\_requirements":{

"total\_score\_route\_info": {

"total\_score":,

"start\_point": "",

"end\_point": "",

"route\_distance":,

"attractions\_passed by":[

"天坛" ,

"地坛" ,

],

"attraction\_attributes":[

"自然风光",

"历史人文",

],

"activity":[

"划船"

],

"total\_activity\_cost":,

}

},

"trip\_requirements":{

"short\_route\_info": {

"distance\_score":,

"start\_point": "",

"end\_point": "",

"route\_distance":,

"attractions\_passed by":[

"天坛" ,

"地坛" ,

],

"attraction\_attributes":[

"自然风光",

"历史人文",

],

"activity":[

"划船"

],

"total\_activity\_cost":,

}

},

"trip\_requirements":{

"cheap\_route\_info": {

"cost\_score":,

"start\_point": "",

"end\_point": "",

"route\_distance":,

"attractions\_passed by":[

"天坛" ,

"地坛" ,

],

"attraction\_attributes":[

"自然风光",

"历史人文",

],

"activity":[

"划船"

],

"total\_activity\_cost":,

}

},

"trip\_requirements":{

"preferences\_route\_info": {

"preferences\_score":,

"start\_point": "",

"end\_point": "",

"route\_distance":,

"attractions\_passed by":[

"天坛" ,

"地坛" ,

],

"attraction\_attributes":[

"自然风光",

"历史人文",

],

"activity":[

"划船"

],

"total\_activity\_cost":,

}

},

}