附件一：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 第一关 |  |  |
|  |  |  |  |  |
| 日期 | 所在区域 | 剩余资金数 | 剩余水量 | 剩余食物量 |
| 0 | 1 | 5780 | 178 | 333 |
| 1 | 25 | 5780 | 162 | 321 |
| 2 | 24 | 5780 | 146 | 309 |
| 3 | 23 | 5780 | 136 | 295 |
| 4 | 23 | 5780 | 126 | 285 |
| 5 | 22 | 5780 | 116 | 271 |
| 6 | 9 | 5780 | 100 | 259 |
| 7 | 9 | 5780 | 90 | 249 |
| 8 | 15 | 4150 | 243 | 235 |
| 9 | 13 | 4150 | 227 | 223 |
| 10 | 12 | 4150 | 211 | 211 |
| 11 | 12 | 4150 | 201 | 201 |
| 12 | 12 | 5150 | 177 | 183 |
| 13 | 12 | 6150 | 162 | 162 |
| 14 | 12 | 7150 | 138 | 144 |
| 15 | 12 | 8150 | 114 | 126 |
| 16 | 12 | 9150 | 90 | 108 |
| 17 | 12 | 9150 | 80 | 98 |
| 18 | 12 | 10150 | 50 | 68 |
| 19 | 12 | 11150 | 26 | 50 |
| 20 | 13 | 11150 | 10 | 38 |
| 21 | 15 | 10470 | 36 | 40 |
| 22 | 9 | 10470 | 26 | 26 |
| 23 | 21 | 10470 | 10 | 14 |
| 24 | 27 | 10470 | 0 | 0 |
| 25 |  |  |  |  |
| 26 |  |  |  |  |
| 27 |  |  |  |  |
| 28 |  |  |  |  |
| 29 |  |  |  |  |
| 30 |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 第二关 |  |  |
|  |  |  |  |  |
| 日期 | 所在区域 | 剩余资金数 | 剩余水量 | 剩余食物量 |
| 0 | 1 | 5300 | 130 | 405 |
| 1 | 2 | 5300 | 114 | 393 |
| 2 | 3 | 5300 | 98 | 381 |
| 3 | 4 | 5300 | 88 | 367 |
| 4 | 4 | 5300 | 78 | 357 |
| 5 | 5 | 5300 | 68 | 343 |
| 6 | 13 | 5300 | 52 | 331 |
| 7 | 13 | 5300 | 42 | 321 |
| 8 | 22 | 5300 | 32 | 307 |
| 9 | 30 | 5300 | 16 | 295 |
| 10 | 39 | 5200 | 10 | 283 |
| 11 | 39 | 3410 | 179 | 273 |
| 12 | 30 | 4410 | 163 | 261 |
| 13 | 30 | 5410 | 148 | 240 |
| 14 | 30 | 6410 | 124 | 222 |
| 15 | 30 | 7410 | 110 | 204 |
| 16 | 30 | 8410 | 86 | 186 |
| 17 | 30 | 9410 | 56 | 156 |
| 18 | 30 | 9410 | 26 | 126 |
| 19 | 39 | 5730 | 196 | 200 |
| 20 | 46 | 5730 | 180 | 188 |
| 21 | 55 | 5730 | 170 | 174 |
| 22 | 55 | 6730 | 155 | 153 |
| 23 | 55 | 7730 | 131 | 135 |
| 24 | 55 | 8730 | 116 | 114 |
| 25 | 55 | 9730 | 86 | 84 |
| 26 | 55 | 10730 | 62 | 66 |
| 27 | 55 | 11730 | 47 | 45 |
| 28 | 55 | 12730 | 32 | 24 |
| 29 | 56 | 12730 | 16 | 12 |
| 30 | 64 | 12730 | 0 | 0 |

附件二：

|  |
| --- |
| import numpy as np  #marked\_point 0-> start； 1-> village；2 -> mine; 3-> end  position = np.array((0,1,2,3))  # The distance between the marked position  distance=np.array(((0, 6, 8, 3),  (6, 0, 2, 3),  (8, 2, 0, 5),  (3, 3, 5, 0)))  # The relation between the marked position is found by graph theory by hand  relation\_position = np.array(((0, 1, 1, 1),  (0, 0, 1, 1),  (0, 1, 0, 1),  (0, 0, 0, 0)))  #The weather in module 1 in 30 days  weather = np.array((2, 2, 1, 3, 1,  2, 3, 1, 2, 2,  3, 2, 1, 2, 2,  2, 3, 3, 2, 2,  1, 1, 2, 1, 3,  2, 1, 1, 2, 2))  #Weight&Price  weight\_water=3  weight\_food=2  price\_water=5  price\_food=10  #Consumption in different weather  consumption\_water\_weather=np.array((0, 5, 8, 10))  consumption\_food\_weather=np.array((0, 7, 6, 10))  #The number of terrain's type  n=4  #Original argument  capacity=1200  fund\_basic=10000  income=1000  date=30  consumption\_water=np.zeros((32,4,4))  consumption\_food=np.zeros((32,4,4))  # How long does it take from i to j on day k  dayss = np.zeros((32, 4, 4))  fund=0  #Record the everyday information  activity\_everyday=np.zeros(32)  position\_everyday=np.zeros(32)  optimum\_position = np.zeros(32)  optimum\_activity = np.zeros(32)  optimum\_origin\_water=0  optimum\_origin\_food=0  #Use the algorithm of DFS. I learned it on GitHub!  def depth\_first\_search(days:int, now:int,weight:int, fund\_now:int,water:int, food:int, type:int)->None:  activity\_everyday[days]=type  position\_everyday[days]=now  global fund, optimum\_origin\_food, original\_water    #village  if position[now]==1:  weight=capacity-water\*weight\_water-food\*weight\_food  for i in range(n):  if relation\_position[position[now]][position[i]]:  cost\_food=consumption\_food[days][now][i]  cost\_water=consumption\_water[days][now][i]  cost=fund\_now  weight\_now=weight  if food>=cost\_food:  food\_now=food-cost\_food  else:  food\_now=0  cost=cost-2\*(cost\_food-food)\*price\_food  weight\_now=weight\_now-(cost\_food-food)\*weight\_food  if water>=cost\_water:  water\_now=water-cost\_water  else:  water\_now=0  cost=cost-2\*(cost\_water-water)\*price\_water  weight\_now=weight\_now-(cost\_water-water)\*weight\_water    if weight\_now<0 or cost<0:  continue  depth\_first\_search(days+dayss[days][now][i],i,weight\_now,cost,water,food,0)  #mine  if position[now]==2:  attday=days  cost\_food=consumption\_water\_weather[weather[attday]]  cost\_water=consumption\_food\_weather[weather[attday]]  attday+=1  if water>=cost\_water:  x=x-cost\_water  cost\_water=0  else:  cost\_water=cost\_water-water  water=0  if food>=cost\_food:  food=food-cost\_food  cost\_food=0  else:  cost\_food=cost\_food-food  food=0  weight=weight-cost\_food\*weight\_food-cost\_water\*weight\_water  fund\_now=fund\_now-2\*(cost\_food\*price\_food+cost\_water\*price\_water)  if weight>=0 and fund>=0:  depth\_first\_search(attday,now,weight,fund\_now,water,food,1)  attday=days  cost\_water=consumption\_water\_weather[weather[attday]]\*2  cost\_food=consumption\_food\_weather[weather[attday]]\*2  attday+=1  if water>=cost\_water:  x=x-cost\_water  cost\_water=0  else:  cost\_water=cost\_water-water  water=0  if food>=cost\_food:  food=food-cost\_food  cost\_food=0  else:  cost\_food=cost\_food-food  food=0  weight=weight-cost\_food\*weight\_food-cost\_water\*weight\_water  fund\_now=fund\_now-2\*(cost\_food\*price\_food+cost\_water\*price\_water)  fund\_now=fund\_now+income  if weight>=0 and fund>=0:  depth\_first\_search(attday,now,weight,fund\_now,water,food,2)  if position[now]==3:  if fund<=fund\_now+water\*price\_water+food\*price\_food:  optimum\_origin\_water=water  optimum\_origin\_food=food  fund=fund\_now+water\*price\_water+water\*price\_food  for i in range(date+1):  optimum\_position[i]=position\_everyday[i]  optimum\_activity[i]=activity\_everyday[i]  activity\_everyday[days]=-1  position\_everyday[days]=-1  return    if days>=date:  activity\_everyday[days]=-1  position\_everyday[days]=-1  return  activity\_everyday[days]=-1  position\_everyday[days]=-1  #Magic function!  if \_\_name\_\_ =='\_\_main\_\_':  for i in range(date+1):  activity\_everyday[i]=-1  position\_everyday[i]=-1  for i in range(date+1):  for j in range(n):  for k in range(n):  if relation\_position[position[j]][position[k]]:  now, count, sumx, sumy = 0, 0, 0, 0  while count<distance[j][k]:  if weather[now+i]!=3:  count+=1  sumx=sumx+2\*consumption\_water\_weather[weather[now+i]]  sumy=sumy+2\*consumption\_food\_weather[weather[now+i]]  else:  sumx=sumx+consumption\_water\_weather[weather[now+i]]  sumy=sumy+consumption\_food\_weather[weather[now+i]]    now+=1  if now+i >=date:  break  if count <distance[j][k]:  sumx=xumy=20000  now=30  consumption\_water[i][j][k]=sumx  consumption\_food[i][j][k]=sumy  dayss[i][j][k]=now  print(type(dayss[0,0,0]))  dictionary={}  for i in range(capacity+1):  c2=i//weight\_food  c1=(capacity-i)//weight\_water  dictionary.setdefault((c1,c2),0)  if not dictionary[(c1,c2)]:  print((c1,c2))  depth\_first\_search(0,0,0,fund\_basic-c2\*consumption\_food-c1\*consumption\_water,c1,c2,-1)  dictionary[(c1,c2)]=1  print(fund) |

|  |
| --- |
| import numpy as np  #marked\_point 0-> start； 1-> village；2 -> mine; 3-> end  position = np.array((0,2,1,2,1,3))  # The distance between the marked position  distance=np.array(((0, 7, 8, 9, 9, 11),  (7, 0, 1, 3, 4, 4),  (8, 1, 0, 2, 3, 3),  (9, 3, 2, 0, 1, 2),  (9, 4, 3, 1, 0, 2),  (11, 4, 3, 2, 2, 0)))  # The relation between the marked position is found by graph theory by hand  relation\_position = np.array(((0, 1, 1, 1),  (0, 0, 1, 1),  (0, 1, 0, 1),  (0, 0, 0, 0)))  #The weather in module 1 in 30 days  weather = np.array((2, 2, 1, 3, 1,  2, 3, 1, 2, 2,  3, 2, 1, 2, 2,  2, 3, 3, 2, 2,  1, 1, 2, 1, 3,  2, 1, 1, 2, 2))  #Weight&Price  weight\_water=3  weight\_food=2  price\_water=5  price\_food=10  #Consumption in different weather  consumption\_water\_weather=np.array((0, 5, 8, 10))  consumption\_food\_weather=np.array((0, 7, 6, 10))  #The number of terrain's type  n=6  #Original argument  capacity=1200  fund\_basic=10000  income=1000  date=30  consumption\_water=np.zeros((32,6,6))  consumption\_food=np.zeros((32,6,6))  # How long does it take from i to j on day k  dayss = np.zeros((32, 6, 6))  fund=0  #Record the everyday information  activity\_everyday=np.zeros(32)  position\_everyday=np.zeros(32)  optimum\_position = np.zeros(32)  optimum\_activity = np.zeros(32)  optimum\_origin\_water=0  optimum\_origin\_food=0  #Use the algorithm of DFS. I learned it on GitHub!  def depth\_first\_search(days:int, now:int,weight:int, fund\_now:int,water:int, food:int, type:int)->None:  activity\_everyday[days]=type  position\_everyday[days]=now  global fund, optimum\_origin\_food, original\_water  #village  if position[now]==1:  weight=capacity-water\*weight\_water-food\*weight\_food  for i in range(n):  if relation\_position[position[now]][position[i]]:  cost\_food=consumption\_food[days][now][i]  cost\_water=consumption\_water[days][now][i]  cost=fund\_now  weight\_now=weight  if food>=cost\_food:  food\_now=food-cost\_food  else:  food\_now=0  cost=cost-2\*(cost\_food-food)\*price\_food  weight\_now=weight\_now-(cost\_food-food)\*weight\_food  if water>=cost\_water:  water\_now=water-cost\_water  else:  water\_now=0  cost=cost-2\*(cost\_water-water)\*price\_water  weight\_now=weight\_now-(cost\_water-water)\*weight\_water    if weight\_now<0 or cost<0:  continue  depth\_first\_search(days+dayss[days][now][i],i,weight\_now,cost,water,food,0)  #mine  if position[now]==2:  attday=days  cost\_food=consumption\_water\_weather[weather[attday]]  cost\_water=consumption\_food\_weather[weather[attday]]  attday+=1  if water>=cost\_water:  x=x-cost\_water  cost\_water=0  else:  cost\_water=cost\_water-water  water=0  if food>=cost\_food:  food=food-cost\_food  cost\_food=0  else:  cost\_food=cost\_food-food  food=0  weight=weight-cost\_food\*weight\_food-cost\_water\*weight\_water  fund\_now=fund\_now-2\*(cost\_food\*price\_food+cost\_water\*price\_water)  if weight>=0 and fund>=0:  depth\_first\_search(attday,now,weight,fund\_now,water,food,1)  attday=days  cost\_water=consumption\_water\_weather[weather[attday]]\*2  cost\_food=consumption\_food\_weather[weather[attday]]\*2  attday+=1  if water>=cost\_water:  x=x-cost\_water  cost\_water=0  else:  cost\_water=cost\_water-water  water=0  if food>=cost\_food:  food=food-cost\_food  cost\_food=0  else:  cost\_food=cost\_food-food  food=0  weight=weight-cost\_food\*weight\_food-cost\_water\*weight\_water  fund\_now=fund\_now-2\*(cost\_food\*price\_food+cost\_water\*price\_water)  fund\_now=fund\_now+income  if weight>=0 and fund>=0:  depth\_first\_search(attday,now,weight,fund\_now,water,food,2)  if position[now]==3:  if fund<=fund\_now+water\*price\_water+food\*price\_food:  optimum\_origin\_water=water  optimum\_origin\_food=food  fund=fund\_now+water\*price\_water+water\*price\_food  for i in range(date+1):  optimum\_position[i]=position\_everyday[i]  optimum\_activity[i]=activity\_everyday[i]  activity\_everyday[days]=-1  position\_everyday[days]=-1  return    if days>=date:  activity\_everyday[days]=-1  position\_everyday[days]=-1  return    activity\_everyday[days]=-1  position\_everyday[days]=-1  #Magic function!  if \_\_name\_\_ =='\_\_main\_\_':  for i in range(date+1):  activity\_everyday[i]=-1  position\_everyday[i]=-1  for i in range(date+1):  for j in range(n):  for k in range(n):  if relation\_position[position[j]][position[k]]:  now, count, sumx, sumy = 0, 0, 0, 0  while count<distance[j][k]:  if weather[now+i]!=3:  count+=1  sumx=sumx+2\*consumption\_water\_weather[weather[now+i]]  sumy=sumy+2\*consumption\_food\_weather[weather[now+i]]  else:  sumx=sumx+consumption\_water\_weather[weather[now+i]]  sumy=sumy+consumption\_food\_weather[weather[now+i]]    now+=1  if now+i >=date:  break  if count <distance[j][k]:  sumx=xumy=20000  now=30  consumption\_water[i][j][k]=sumx  consumption\_food[i][j][k]=sumy  dayss[i][j][k]=now  print(type(dayss[0,0,0]))  dictionary={}  for i in range(capacity+1):  c2=i//weight\_food  c1=(capacity-i)//weight\_water  dictionary.setdefault((c1,c2),0)  if not dictionary[(c1,c2)]:  print((c1,c2))  depth\_first\_search(0,0,0,fund\_basic-c2\*consumption\_food-c1\*consumption\_water,c1,c2,-1)  dictionary[(c1,c2)]=1  print(fund) |