

# Cheat paper

## dijkstra模板

```
def dijkstra(x1,y1,x2,y2):
    pq=[(0,x1,y1)]
    dist=[[float("inf")]*(n+2) for i in range(m+2)]
    dist[x1][y1]=0
    while pq:
        cost,x,y=heapq.heappop(pq)
        if x==x2 and y==y2:
            return cost
        else:
            for i in range(4):
                nx,ny=x+dir[i][0],y+dir[i][1]
                if dist[nx][ny]>dist[x][y]+abs(a[nx][ny]-a[x][y]):
                    dist[nx][ny]=dist[x][y]+abs(a[nx][ny]-a[x][y])
                    heapq.heappush(pq,(cost+abs(a[nx][ny]-a[x][y]),nx,ny))
    return float("inf")
```

利用heapq简化存入取出的过程!

## Dilworth's theorem

最常见的应用：最小完整覆盖全数组的上升子序列数等于最长非上升子序列的长度，这类的!

## 二分查找模板

bisect\_left:

```
lo,hi=0,len(a)
while lo < hi:
    mid = (lo + hi) // 2
    if a[mid] < x:
        lo = mid + 1
    else:
        hi = mid
```

bisect\_right:

```
lo,hi=0,len(a)
while lo < hi:
    mid = (lo + hi) // 2
    if x < a[mid]:
        hi = mid
    else:
        lo = mid + 1
```

## python中用format保留小数位数的用法

```
# 保留两位小数
number = 3.14159
formatted_number = "{:.2f}".format(number)
print(formatted_number) # 输出: 3.14
```

## 哈希表

求得一个数组最长的和为0的连续子序列的方法（实例为求最长平均值为a的字串）

```
def longest_subarray_with_avg(arr, a):
    # 步骤 1: 将每个元素减去a
    diff = [x - a for x in arr]

    # 步骤 2: 使用哈希表存储前缀和
    prefix_sum = 0
    prefix_map = {0: -1} # 前缀和为0时, 起始索引为-1
    max_len = 0
    start_index = -1

    # 步骤 3: 遍历数组并计算前缀和
    for i in range(len(diff)):
        prefix_sum += diff[i]

        # 步骤 4: 如果前缀和已经出现过, 计算子数组的长度
        if prefix_sum in prefix_map:
            length = i - prefix_map[prefix_sum]
            if length > max_len:
                max_len = length
                start_index = prefix_map[prefix_sum] + 1
        else:
            prefix_map[prefix_sum] = i

    # 返回最长子数组
    if max_len > 0:
        return arr[start_index:start_index + max_len]
    else:
        return []
```

## 利用二分查找求最长下降子序列

```
import bisect

k = int(input())
a = list(map(int, input().split()))

# 反转数组来模拟下降子序列问题
sub = []

for i in range(k):
    # 由于我们在寻找递增子序列的 "位置", 所以我们在 sub 中寻找 "大于" 当前元素的位置
    pos = bisect.bisect_right(sub, a[i]) # 使用 bisect_right 来模拟下降序列的插入位置
```

```
if pos < len(sub):  
    sub[pos] = a[i] # 更新该位置的值  
else:  
    sub.append(a[i]) # 如果没有找到合适位置，说明我们可以增加一个新的元素  
  
# 输出最长下降子序列的长度  
print(len(sub))
```