Cheatsheet v2.0

Wrriten by Wangjie Su on 12/23/2023

dp

09267:核电站

```
# 有最长连续长度限制的放置问题
dp=[0]*(n+1)
dp[0]=1
for i in range(1,n+1):
    if i<m:
        dp[i]=dp[i-1]*2
    elif i==m:
        dp[i]=dp[i-1]*2-1
    else:
        dp[i]=dp[i-1]*2-dp[i-m-1]
print(dp[n])
```

02806: 公共子序列

```
dp = [[0] * (n + 1) for _ in range(m + 1)]
for i in range(1, m + 1):
    for j in range(1, n + 1):
        if x[i - 1] == y[j - 1]:
            dp[i][j] = dp[i - 1][j - 1] + 1
        else:
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
print(dp[m][n])
```

19164:移动办公

```
#二维dp，在不同地点的同时间dp
t,m=map(int,input().split())
p,n=[0]*(t+1),[0]*(t+1)
for i in range(1,t+1):
    p[i],n[i]=map(int,input().split())
dpp,dpn=[0]*(t+1),[0]*(t+1)
dpp[1],dpn[1]=p[1],n[1]
for i in range(2,t+1):
    dpp[i]=max(dpp[i-1]+p[i],dpn[i-1]+p[i]-m)
    dpn[i]=max(dpn[i-1]+n[i],dpp[i-1]+n[i]-m)
print(max(dpp[t],dpn[t]))
```

27401:最佳凑单

```
# 稀疏桶
a,b=map(int,input().split());c={0}
for i in map(int,input().split()):
    for j in c.copy():
        if j<b:c.add(i+j)
for i in sorted(c):
    if i>=b:print(i);exit()
print(0)

# 0/1背包问题（质量与价格参数相同）
dp=[0]*(a+1)
    for price in prices:
        for i in range(a,price-1,-1):
            dp[i]=max(dp[i],dp[i-price]+price)
    return dp[a]
```

01384:Piggy-Bank

```
# 完全背包中的最优解问题,从头到尾遍历
dp=[0]+[float("inf")]*tw
    for i in range(n):
        for j in range(w[i],tw+1):
            dp[j]=min(dp[j],dp[j-w[i]]+p[i])
    print(f"The minimum amount of money in the piggy-bank is {dp[tw]}."if 
dp[tw]!=float("inf") else "This is impossible.")
```

04141:砝码称重

```
# 多重背包中的方案数问题
dp=[0]*1010
kg=[1,2,3,5,10,20]
num=list(map(int,input().split()))
sum_val=ans=0
for i in range(6):
    sum_val+=num[i]*kg[i]
dp[0]=1
for i in range(6):
    for j in range(1,num[i]+1):
        for k in range(sum_val,kg[i]-1,-1):
            if dp[k-kg[i]]==1:
                dp[k]=1
for i in range(1,sum_val+1):
    if dp[i]==1:ans+=1
print(f"Total={ans}")
```

01742:Coins

```
#多重背包问题，用二进制优化，将物品分为1、2、4……倍的0/1背包问题
dp = 1  # 初始化为只有第0位为1的整数，表示初始时只有0这个和是可以达到的
    mask = (1 << (m + 1)) - 1  # 创建一个掩码，用于限制dp的长度
    for value, count in zip(values, counts):
        while count:
            k = 1
            while k <= count:  # 找到不超过count的最大2的幂
                dp = (dp | (dp << (value * k))) & mask
                count -= k
                k <<= 1
    print(bin(dp).count('1') - 1)
```

20089:NBA门票

```
# 多重背包中的最优解问题
dp=[float('inf')]*(n+1)
dp[0]=0
for i in range(6,-1,-1):
    cur=price[i]
    for k in range(n,cur-1,-1):
        for j in range(1,nums[i]+1):
            if k>=cur*j:
                dp[k]=min(dp[k],dp[k-cur*j]+j)
            else:
                break
if dp[-1]==float('inf'):
    print('Fail')
else:
    print(dp[-1])
```

```
# 二进制优化
dp = [float('inf')] * (n + 1)
dp[0] = 0
for i in range(7):
    cur_price = price[i]
    cur_num = nums[i]
    k = 1
    while cur_num > 0:
        use_num = min(cur_num, k)
        cur_num -= use_num
        for j in range(n, cur_price * use_num - 1, -1):
            dp[j] = min(dp[j], dp[j - cur_price * use_num] + use_num)
        k *= 2
if dp[-1] == float('inf'):
    print('Fail')
else:
    print(dp[-1])
```

02755:神奇的口袋

```
# 0-1背包中的方案数问题
def dfs(w,k):    #(总重, 总数量)
    if w==0:
        return 1
    if k<=0:
        return 0
    return dfs(w,k-1)+dfs(w-l[k],k-1)

n=int(input())
l=[0]
for _ in range(n):
    l.append(int(input()))
print(dfs(40,n))
```

02773:采药

```
# 0-1背包中的最优解问题
dp=[-1]*(T+1)
dp[0]=0
for _ in range(m):
    t,v=map(int,input().split())
    for i in range(T,t-1,-1):
        if dp[i-t]!=-1:
            dp[i]=max(dp[i],dp[i-t]+v)
print(max(dp))
```

21458:健身房 (dp)

```
#恰好型dp, 初始化为INF
t,n=map(int,input().split())
l=[[0]]
for i in range(n):
    l.append(list(map(int,input().split())))
ans=[0]+[-float("inf")]*t
for i in range(1,n+1):
    for j in range(t,l[i][0]-1,-1):
        ans[j]=max(ans[j],ans[j-l[i][0]]+l[i][1])
print(ans[t] if ans[t]>=0 else -1)
```

02757:最长上升子序列

```
dp=[1]*n
for i in range(n):
    for j in range(i):
        if numbers[j]<numbers[i]:
            dp[i]=max(dp[j]+1,dp[i])
print(max(dp))
```

03263:新数字三角形

```
#从下开始往上dp
while True:
    n=int(input())
    if n==0:
        break
    maxsum=[[0]*120 for _ in range(120)]
    a=[[0]*120 for _ in range(120)]
    for i in range(1,n+1):
        a[i][1:i+1]=map(int,input().split())
    x,y=map(int,input().split())
    for i in range(n,0,-1):
        for j in range(1,i+1):
            max_low=max(maxsum[i+1][j],maxsum[i+1][j+1])
            maxsum[i][j]=max(max_low,a[i][j])
    print(maxsum[x][y])
```

24755:有多少种二叉树

```
#卡特兰数，与入栈次数、括号匹配数、凸多边形的三角划分等一样
n=int(input())
def f(n):
    num=0
    if n==0 or n==1:
        return 1
    else:
        for i in range(n):
            num+=f(i)*f(n-1-i)
        return num
print(f(n))
```

04119:复杂的整数划分问题

```
# N划分成K个正整数之和
def divide_k(n,k):
    dp=[[0]*(k+1) for _ in range(n+1)]
    for i in range(n+1):
        dp[i][1]=1
    for i in range(1,n+1):
        for j in range(1,k+1):
            if i>=j:
                # dp[i-1][j-1]为包含1的划分的数量
                # 若不包含1，我们对每个数-1仍为正整数，划分数量为dp[i-j][j]
                dp[i][j]=dp[i-j][j]+dp[i-1][j-1]
    return dp[n][k]
#可以出现0的划分（01664 放苹果）
dp=[[0]*(y+1) for i in range(x+1)]
    for i in range(1,y+1):
        dp[0][i]=1
    for i in range(1,x+1):
        for j in range(1,y+1):
            if i<j:
```

```python
                dp[i][j]=dp[i][i]
            else:
                dp[i][j]=dp[i][j-1]+dp[i-j][j]
    return dp[x][y]
# N划分成若干个不同正整数之和/N划分成若干个奇整数之和
def divide_dif(n):
    # dp[i][j]表示将数字 i 划分，其中最大的数字不大于 j 的方法数量
    dp = [[0] * (n + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i < j:
                dp[i][j] = dp[i][i]
            elif i == j:
                dp[i][j] = dp[i][j - 1] + 1
            # 用/不用j
            else:
                dp[i][j] = dp[i][j - 1] + dp[i - j][j - 1]
    return dp[n][n]
```

27104:世界杯只因

```python
# 区间覆盖问题
n=int(input())
line=[0]+list(map(int,input().split()))
queue=[0]*(n+1)
for i in range(1,n+1):
    l,r=max(1,i-line[i]),min(n,i+line[i])
    queue[l]=max(queue[l],r)
right=ans=0
index=1
while right<n:
    tmpright=right
    while index<=right+1:
        tmpright=max(tmpright,queue[index])
        index+=1
    ans+=1
    right=tmpright
print(ans)
```

23586:幸福的寒假生活

```python
# 不重叠区间的最优解问题
dp=[0]*46
for i in range(1,46):
    dp[i]=dp[i-1]
    for start,end,happiness in data:
        if end==i:
            dp[i]=max(dp[i],dp[start-1]+happiness)
print(dp[-1])
```

greedy

02287:Tian Ji -- The Horse Racing

```
#大抵大，小抵小，小抵大
def check(a,b):
    ans=0
    l1,r1=0,len(a)-1
    l2,r2=0,len(b)-1
    while l1<=r1 and l2<=r2:
        if b[r2]>a[r1]:
            r2-=1;r1-=1;ans+=200
        elif b[l2]>a[l1]:
            l1+=1;l2+=1;ans+=200
        elif b[l2]==a[r1]:
            l2+=1;r1-=1;ans+=0
        else:
            r1-=1;l2+=1;ans-=200
    return ans

while True:
    n=int(input())
    if n==0:
        break
    tian=list(map(int,input().split()))
    king=list(map(int,input().split()))
    tian.sort()
    king.sort()
    maxi=check(king,tian)
    print(maxi)
```

04137:最小新整数

```
#能删则删，最后减位
a, n = input().split()
    n = int(n)
    a = list(a)
    for _ in range(n):
        for i in range(len(a) - 1):
            if a[i] > a[i + 1]:
                del a[i]
                break
        else:
            a.pop()
    print(''.join(a))
```

02783:holiday hotel

```
# 筛选问题
hotels=[tuple(map(int,input().split())) for _ in range(n)]
hotels.sort(key=lambda x:(x[0],x[1]))
candidates=1
max_cost_so_far=hotels[0][1]
for i in range(n):
    if hotels[i][1]<max_cost_so_far:
        candidates+=1
        max_cost_so_far=hotels[i][1]
print(candidates)
```

02431:Expedition

```
# 加油问题
stations.sort() # 按距离升序
stations.append((L, 0) # 添加起点
pq = []  # 最大堆
stops, prev, fuel = 0, 0, P
for location, capacity in stations:
    fuel -= location - prev
    while pq and fuel < 0:  # 当前燃料不够到达下一加油站
        fuel += -heapq.heappop(pq)  # 选择提供最多燃料的加油站加油
        stops += 1
    if fuel < 0: print(-1) ;exit()  # 无法到达下一个加油站或终点
    heapq.heappush(pq, -capacity)
    prev = location
print(stops)
```

026646:建筑修建

```
# 区间调度问题
def generate_intervals(x,width,m):
    temp=[]
    for start in range(max(0,x-width+1),min(m,x+1)):
        end=start+width
        if end<=m:
            temp.append((start,end))
    return temp
for x,width in plans:
    intervals.extend(generate_intervals(x,width,m))
intervals.sort(key=lambda x:(x[1],x[0]))
cnt=0
last_end=0
for start,end in intervals:
    if start>=last_end:
        last_end=end
        cnt+=1
print(cnt)
```

04135:月度开销

```
# 二分贪心
def reachable(expenses,target,m):
......
l,r=max(expenses),sum(expenses)
while l<r:
    mid=(l+r)//2
    if reachable(expenses,mid,m):
        r=mid
    else:
        l=mid+1
print(l)
```

01065:Wooden Sticks

```
# 单调子列的最小拆分数
data=list(zip(data[0::2],data[1::2]))
data.sort(key=lambda x:(x[0],x[1]))
flag=[False]*n
cnt=0
for i in range(n):
    if flag[i]:
        continue
    cur=data[i][1]
    cnt+=1
    for j in range(i,n):
        if flag[j]==False and data[j][1]>=cur:
            flag[j]=True
            cur=data[j][1]
print(cnt)
```

implementation

04138: 质数的和与积

```
#欧拉筛法
from math import sqrt
    ls,x,y=[True]*(n+1),2,int(sqrt(n))+1
    while x<y:
        if ls[x]==True:
            for i in range(x*2,n+1,x):
                ls[i]=False
        x+=1
    ls=[i for i in range(2,n+1) if ls[i]==True]
```

02996:选课

```
# 排列问题
def nextper(a):
    size = len(a)
    flag = size - 1
    while flag != 0 and a[flag - 1] > a[flag]:
        flag -= 1
    if flag == 0:
        a.sort()
        return
    for i in range(size - 1, flag - 1, -1):
        if a[i] > a[flag - 1]:
            a[i], a[flag - 1] = a[flag - 1], a[i]
            break
    a[flag:] = a[flag:][::-1]
```

分解因数

```
def decompositions(n,minfactor):
    if n==1:
        return 1
    count=0
    for i in range(minfactor,n+1):
        if n%i==0:
            count+=decompositions(n//i,i)
    return count
print(decompositions(x,2))
```

27205:护林员盖房子

```
# 寻找最大全0子矩阵
maze=[[0]*(n+1) for _ in range(m+1)]
for i in range(1,m+1):
    for j in range(1,n+1):
        if not martix[i][j]:
            maze[i][j]=maze[i][j-1]+1
        else:
            maze[i][j]=0
smax=0
for i in range(1,m+1):
    for j in range(1,n+1):
        if maze[i][j]!=0:
            width=1;length=maze[i][j];area=width*length
            smax=max(smax,area)
            for k in range(i-1,0,-1):
                if maze[k][j]:
                    width+=1;length=min(length,maze[k]
[j]);smax=max(smax,width*length)
                else:
                    break
print(smax)
```

26977:接雨水

```
#两端搜索维护最大值
n=int(input())
l=[int(i) for i in input().split()]
l_max,r_max,num=[0]*(n+1),[0]*(n+1),0
for i in range(n):
    l_max[i+1]=max(l_max[i],l[i])
for i in range(n-1,-1,-1):
    r_max[i]=max(r_max[i+1],l[i])
for i in range(n):
    num+=min(l_max[i+1],r_max[i])-l[i]
print(num)
```

26978:滑动窗口最大值

```
import heapq
n,k=map(int,input().split())
l=[-int(i) for i in input().split()]
heap,out=[],[]
for i in range(k-1):
    heapq.heappush(heap,(l[i],i))
for i in range(k-1,n):
    heapq.heappush(heap,(l[i],i))
    num,No=heapq.heappop(heap)
    while No<i-k+1:
        num,No=heapq.heappop(heap)
    out.append(str(-num))
    heapq.heappush(heap,(num,No))
print(" ".join(out))
```

04134:查找最接近的元素

```
# bisect库
from bisect import bisect_left
def find_closest(arr, target):
    n = len(arr)
    if target <= arr[0]:
        return arr[0]
    if target >= arr[n-1]:
        return arr[n-1]
    pos = bisect_left(arr, target)
    if (arr[pos] - target) < (target - arr[pos - 1]):
        return arr[pos]
    else:
        return arr[pos - 1]
```

03704: 括号匹配

```python
#stack结构，储存（位置在）处出队
while True:
    try:
        a=input()
        ans=[" "]*len(a)
        stack,ps=[""]*len(a),0
        for i in range(len(a)):
            if a[i]=="(":
                stack[ps]=i;ps+=1
            elif a[i]==")":
                if (ps):ps-=1
                else: ans[i]="?"
        for _ in range(ps):
            ans[stack[_]]="$"
        print(a)
        print("".join(ans))
    except EOFError:
        break
```

21462:加密的称赞 v0.2 (matrices)

```python
#矩阵旋转（逆时针）
n=int(input())
martix=[list(map(int,input().split())) for i in range(n)]
cnt_x,cnt_y,dx,dy=0,0,0,1#(0,0,1,0)
cnt_n,cnt_s,cnt_e,cnt_w=0,n-1,n-1,0
ans=[]
for i in range(1,n*n+1):
    if martix[cnt_y][cnt_x]==0:
        break
    ans.append(chr(martix[cnt_y][cnt_x]))
    cnt_x+=dx;cnt_y+=dy
    if dx==1 and dy==0 and cnt_x==cnt_e and cnt_y==cnt_s:
        dx,dy=0,-1; cnt_s-=1   #(0,1)  #cnt_n+=1 上为cnt_n
    elif dx==0 and dy==1 and cnt_x==cnt_w and cnt_y==cnt_s:
        dx,dy=1,0;cnt_w+=1     #(-1,0) #cnt_e-=1 上为cnt_e
    elif dx==-1 and dy==0 and cnt_x==cnt_w and cnt_y==cnt_n:
        dx,dy=0,1;cnt_n+=1     #(0,-1) #cnt_s-=1 上为cnt_s
    elif dx==0 and dy==-1 and cnt_x==cnt_e and cnt_y==cnt_n:
        dx,dy=-1,0;cnt_e-=1    #(1,0)  #cnt_w+=1 上为cnt_w
print("".join(ans))
```

consecutive subsequence

```python
# 连续上升子序列最大长度及序号序列
from collections import defaultdict
n=int(input())
num=list(map(int,input().split()))
dic=defaultdict(int)
ans,res=0,[]
for i in range(n):
    dic[num[i]]=dic[num[i]-1]+1
    if dic[num[i]]>ans:
        ans=dic[num[i]]
        end=num[i]
print(ans)
for i in range(n-1,-1,-1):
    if num[i]==end:
        res.append(i+1)
        end-=1;ans-=1
    if ans==0:
        print(*res[::-1])
        break
```

27141:完美的爱

```
from collections import defaultdict
dic=defaultdict(list)
data=[0]+data
ans=0
for i in range(1,n+1):
    # 构造一个减去了520i的前缀和
    # 使得有相同前缀和的位置间即为满足题意的区间
    data[i]+=data[i-1]-520
for i in range(n+1):
    dic[data[i]].append(i)
for i in dic:
    ans=max(ans,max(dic[i])-min(dic[i]))
print(ans*520)
```

Kefa and company

```
# 对滑动窗口内某变量极差限制的最优解问题
friends.sort()
ans=cur=left=0
for right in range(n):
    cur+=friends[right][1]
    while friends[right][0]-friends[left][0]>=d:
        cur-=friends[left][1]
        left+=1
    ans=max(ans,cur)
print(ans)
```

in love

```
# dict+heapq实现最值更新的效率提升
from heapq import heappop,heappush
from collections import defaultdict
q = int(input())
ldict, rdict = defaultdict(int), defaultdict(int)
pq_l, pq_r = [], []
for _ in range(q):
    op, l, r = map(str, input().split())
    l, r = int(l), int(r)
    if op == "+":
        ldict[l] += 1; rdict[r] += 1
        heappush(pq_l, -l); heappush(pq_r, r)
    if op == "-":
        ldict[l] -= 1; rdict[r] -= 1
    while len(pq_l) > 0 >= ldict[-pq_l[0]]:
        heappop(pq_l)
    while len(pq_r) > 0 >= rdict[pq_r[0]]:
        heappop(pq_r)
    if len(pq_l) > 0 and pq_r[0] < -pq_l[0]:
        print("Yes")
    else: print("No")
```

cat party

```
# 桶套桶实现时间复杂度的降低
from collections import defaultdict
n = int(input())
a = list(map(int, input().split()))
cc = defaultdict(int)  # color count
fc = defaultdict(int)  # frequency count
ans = 0
for i in range(n):
    c = a[i]
    if cc[c] in fc:
        fc[cc[c]] -= 1
        if fc[cc[c]] == 0:
            del fc[cc[c]]
    cc[c] += 1;fc[cc[c]] += 1
    if len(fc) == 1 and (1 in fc or list(fc.values())[0] == 1):
        ans = i + 1
    elif len(fc) == 2:
        k = sorted(fc.keys())
        if k[0] + 1 == k[1] and fc[k[1]] == 1 or k[0] == 1 and fc[k[0]] == 1:
            ans = i + 1
print(ans)
```

bfs,dfs

20127:寻宝3.0 v0.2

```
import heapq #bfs+heap+条件 模版
def bfs(x,y):
    d=[[-1,0],[1,0],[0,1],[0,-1]]
    queue=[]
    heapq.heappush(queue,[0,x,y])
    check=set([(x,y)])
    while queue:
        step,x,y=map(int,heapq.heappop(queue))
        if martix[x][y]==1:
            return step
        for i in range(4):
            dx,dy=x+d[i][0],y+d[i][1]
            if martix[dx][dy]!=2 and (dx,dy) not in check:
                heapq.heappush(queue,[step+(martix[dx][dy]!=3),dx,dy])
                check.add((dx,dy))
    return "NO"

m,n=map(int,input().split())
martix=[[2]*(n+2)]+[[2]+list(map(int,input().split()))+[2] for i in range(m)]+
[[2]*(n+2)]
print(bfs(1,1))
```

01321：棋盘问题

```
# 回溯法
def dfs(a):
    global ans,num
    if num==k:
        ans+=1
        return
    for i in range(a+1,n):
        for j in range(n):
            if martix[i][j]=="#" and i not in setx and j not in sety:
                setx.add(i);sety.add(j);num+=1
                dfs(i)
                num-=1;setx.remove(i);sety.remove(j)
```

04127:迷宫问题

```
# 要求输出路径的问题
queue = deque([((0, 0), [])])
while queue:
    (x, y), path = queue.popleft()
    if (x, y) == (n - 1, m - 1):
        return path + [(x, y)]
    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < n and 0 <= ny < m and not visited[nx][ny] and maze[nx][ny]
== 0:
            visited[nx][ny] = True
            queue.append(((nx, ny), path + [(x, y)]))
return []
```

01088:滑雪

```
# 记忆化搜索
def dfs(i,j):
    if dp[i][j]>0:
        return dp[i][j]
    else:
        for k in range(4):
            if 0<=i+d[k][0]<r and 0<=j+d[k][1]<c and maze[i][j]>maze[i+d[k][0]]
[j+d[k][1]]:
                dp[i][j]=max(dp[i][j],dfs(i+d[k][0],j+d[k][1])+1)
    return dp[i][j]
```

```
r,c=map(int,input().split())
maze=[]
for i in range(r):
    l=list(map(int,input().split()))
    maze.append(l)
dp=[[0]*c for _ in range(r)]
d=[[-1,0],[1,0],[0,1],[0,-1]]
ans=0
for i in range(r):
    for j in range(c):
        ans=max(ans,dfs(i,j))
print(ans+1)
```

22007:N皇后问题

```
def isvalid(former,row,col):
    for i in range(row):
        if former[i]==col or abs(i-row)==abs(former[i]-col):
            return False
    return True

def queen(former=[],row=0):
    if row==n:
        result.append(former[:])
        return
    for col in range(n):
        if isvalid(former,row,col):
            former.append(col)
            queen(former,row+1)
            former.pop()

n=int(input())
result=[]
queen()
if result:
    for i in result:
        print(*i)
else:
    print("NO ANSWER")
```

02815:城堡问题

```
# 连通分量的数量和最大面积
def dfs(i,j):
    global roomarea
    if colors[i][j]:
        return
    roomarea+=1
    colors[i][j]=color
    if not (rooms[i][j]&1):dfs(i,j-1)
    if not (rooms[i][j]&2):dfs(i-1,j)
    if not (rooms[i][j]&4): dfs(i,j+1)
    if not (rooms[i][j]&8):dfs(i+1,j)
```

```
n=int(input())
m=int(input())
rooms,color,maxn=[],0,0
colors=[[0]*m for _ in range(n)]
for _ in range(n):
    rooms.append([int(i) for i in input().split()])
for i in range(n):
    for j in range(m):
        if not colors[i][j]:
            color+=1
            roomarea=0
            dfs(i,j)
            maxn=max(maxn,roomarea)
print(color)
print(maxn)
```

23558:有界的深度优先搜索

```
#dfs典范
def dfs(x,d):
    if d<=l-1:
        for i in tree[x]:
            if check[i]:
                check[i]=False
                ans.append(i)
                dfs(i,d+1)

n,m,l=map(int,input().split())
tree=[[] for _ in range(n)]
for i in range(m):
    a,b=map(int,input().split())
    tree[a].append(b)
    tree[b].append(a)
for i in tree:
    i.sort()
start=int(input())
check,ans=[True]*n,[start]
check[start]=False
dfs(start,0)
print(*ans)
```

04124:海贼王之伟大航路

```
# 旅行商问题
def tsp(n,cost):
    # dp[mask][i]为从起始岛屿开始，经过mask表示的岛屿，最后停在岛屿i的最短时间
    dp=[[float('inf')]*(n+1) for _ in range(1<<n)]
    dp[1][1]=0
    # 使用二进制来表示每个岛屿是否访问
    for mask in range(1,1<<n):
        for u in range(1,n+1):
            if not (mask&(1<<(u-1))):
                continue
            # 尝试从所有其他岛屿v转移到岛屿u
            for v in range(1,n+1):
                if mask&(1<<(v-1)) and u!=v:
                    # 对于每一个mask，我们可以从v岛前往u岛，并更新dp[mask][u]的值。
                    dp[mask][u]=min(dp[mask][u],dp[mask^(1<<(u-1))][v]+cost[v]
[u])
    return dp[(1<<n)-1][n]
```

01724:roads

```
# 有限制的加权图最短路径问题
# Dijkstra算法
import heapq
def dijkstra(k,n,roads):
    graph=[[] for _ in range(n+1)]
    for road in roads:
        s,d,l,t=road
        graph[s].append((d,l,t))
    # dist[i][j]表示到达城市i且支付了j个硬币时的最短距离
    dist=[[float('inf')]*(k+1) for _ in range(n+1)]
    dist[1][0]=0
    #  优先队列
    queue=[(0,1,0)]
    while queue:
        # 选择最短距离的未访问节点
        distance,node,toll=heapq.heappop(queue)
        if node==n:
            return distance
        for next_node,length,cost in graph[node]:
            new_toll=toll+cost
            new_distance=distance+length
            # 更新邻居节点的距离
            if new_toll<=k and new_distance<dist[next_node][new_toll]:
                dist[next_node][new_toll]=new_distance
                heapq.heappush(queue,(new_distance,next_node,new_toll))
    return -1
k=int(input())
n=int(input())
r=int(input())
roads=[tuple(map(int,input().split())) for _ in range(r)]
print(dijkstra(k,n,roads))
```

工具

int(str,n)

for key,value in dict.items()

for index,value in enumerate(list)

dict.get(key,default)

list(zip(a,b))

math.pow(m,n)

math.log(m,n)

lrucache

```
from functools import lru_cache
@lru_cache(maxsize=None)
```

calendar

1. `calendar.month(年, 月)`: 返回一个月份的日历字符串。它接受年份和月份作为参数，并以多行字符串的形式返回该月份的日历。
2. `calendar.calendar(年)`: 返回一个年份的日历字符串。这个函数生成整个年份的日历，格式化为多行字符串。
3. `calendar.monthrange(年, 月)`: 返回两个整数，第一个是该月第一天是周几（0-6表示周一到周日），第二个是该月的天数。
4. `calendar.weekday(年, 月, 日)`: 返回给定日期是星期几。0-6的返回值分别代表星期一到星期日。
5. `calendar.isleap(年)`: 返回一个布尔值，指示指定的年份是否是闰年。
6. `calendar.leapdays(年1, 年2)`: 返回在指定范围内的闰年数量，不包括第二个年份。
7. `calendar.monthcalendar(年, 月)`: 返回一个整数矩阵，表示指定月份的日历。每个子列表表示一个星期；天数为0表示该月份此天不在该星期内。
8. `calendar.setfirstweekday(星期)`: 设置日历每周的起始日。默认情况下，第一天是星期一，但可以通过这个函数更改。
9. `calendar.firstweekday()`: 返回当前设置的每周起始日。

counter：计数

```
from collections import Counter
a=['red', 'blue', 'red', 'green', 'blue', 'blue']
a=Counter(a)
```

permutations：全排列

```
from itertools import permutations as per
elements = [1, 2, 3]
permutations = list(per(elements))
```

combinations：组合

```
from itertools import combinations as com
elements = ['A', 'B', 'C', 'D']
# 生成所有长度为2的组合
combinations = list(com(elements, 2))
```

bisect

```python
import bisect
# 创建一个已排序的列表
sorted_list = [1, 3, 3, 6, 7, 9]
# 使用 bisect_left 查找元素应插入的位置
insert_index = bisect.bisect_left(sorted_list, 4)
print("Insert at index:", insert_index)
# 使用 insort_left 插入元素并保持有序
bisect.insort_left(sorted_list, 4)
print("Updated list:", sorted_list)
```

reduce：累积

```python
import functools
numbers = [1, 2, 3, 4, 5]
# 使用 reduce 计算累积乘积
product = functools.reduce(lambda x, y: x * y, numbers)
```

product：笛卡尔积

```python
from itertools import product
# 创建两个可迭代对象
colors = ['red', 'blue']
numbers = [1, 2]
# 生成它们的笛卡尔积
cartesian_product = list(product(colors, numbers))
# 创建一个可迭代对象
colors = ['red', 'blue']
# 生成它们的重复笛卡尔积
repeat_cartesian_product = list(product(colors, repeat=3))
```