# 笔记

## 一、bfs

### 1.deque实现

```
from collections import deque
q = deque([12,43,54])
#append队列尾端添加元素
q.append()
#popleft弹出最左边元素
q.popleft()
#pop弹出最右边元素
q.pop()
```

### 2.dijkstra算法heapq实现，最小权值路径

不同于bfs模板访问后就不再通过，dijkstra创建了一个权值表格，当再次经过此时，只有更小的权值才能覆盖这个点，否则就忽略这次访问，同时代替了visited的作用

```
import heapq
pq = []
#heappush往pq里添加
heapq.heappush(pq, (5,6))
#heappop弹出字典序最小的元素，元组可以直接用‘=’来承接
a, b = heapq.heappop(pq)
lis = [1, 2, 3]
heapq.heqpify(lis) #形成heapq
pq[0] #直接查看heapq最小元素
data = [5, 7, 9, 1, 3]
print(heapq.nsmallest(3, data))  # 输出: [1, 3, 5]
print(heapq.nlargest(3, data))   # 输出: [9, 7, 5]
```

### 3.集合实现visited加快访问速度

```
visited = set()
#经历过某个点
visited.add(point)
```

## 二、dfs

与bfs没有本质区别，一条路走到死看是否需要回溯，回溯可以直接return，回溯的时候记得把原本的下一步删掉，或者干脆就不走了

# 三、dp

## 1. 01背包，取或不取

将取这个和不取这个的总价值进行比较，取最高的，dp从而保证一定最优

```
n, b = map(int, input().split())
value = list(map(int, input().split()))
weight = list(map(int, input().split()))
dp = [[0 for _ in range(b+1)] for _ in range(n+1)]
for i in range(1, n+1):
    for j in range(b+1):
        if weight[i-1] <= j:
            dp[i][j] = max(dp[i-1][j-weight[i-1]]+value[i-1], dp[i-1][j])
print(dp[-1][-1])
```

## 2. 无限背包

遍历dp每一个值，每一个值的时候遍历每一个物件，反正重复了没事

```
n, a, b, c = map(int, input().split())
dp = [float('-inf')]*n
for i in range(1, n+1):
    for j in (a, b, c):
        if i >= j:
            dp[i] = max(dp[i-j] + 1, dp[i])
print(dp[n])
```

## 3. 收服小精灵

```
n, m, k = map(int, input().split())
dp = [[-1 for _ in range(m+1)] for _ in range(k+1)]
dp[0][m] = n
max_life = 0
max_n = 0
for x in range(1, k+1):
    cnt, life = map(int, input().split())
    for i in range(m+1):
        for j in range(x, 0, -1):
            if i + life <= m and dp[j-1][i+life] != -1:
                dp[j][i] = max(dp[j][i], dp[j-1][i+life]-cnt)
                if j > max_n:
                    max_n = j
                    max_life = i
                elif j == max_n:
                    if max_life < i:
                        max_life = i
print(max_n, max_life)
```

# 二、技术问题

## 1.format()

```
#把一个数变为位数更高的浮点数
"{:.10f}".format(number)
```

## 2.bisect

```
import bisect
#lis是有序的
lis = input().split()
#bisect_left若x在里面，给出lis中最左边x的索引值
a = bisect.bisect_left(lis, x)
#bisect_right若x在里面，给出lis中最右边x的索引值
b = bisect.bisect_right(lis, x)
#二者都保证插入后列表仍有序，先插好再给里面各自x的索引值
```

## 3.冒泡排序

本质是从最后一位排起，遍历一遍一定能把最大的找到并移到最后

```
lis = list(map(int, input().split()))
for i in range(len(lis)):
    for j in range(len(lis)-i-1):
        if lis[j] > lis[j+1]:
            lis[j], lis[j+1] = lis[j+1], lis[j]
print(lis)
```

## 4.deepcopy

```
#用深拷贝安全修改新列表而不影响旧列表
from copy import deepcopy
n, m = map(int, input().split())
old = [[0 for _ in range(m+2)]] + [[0]+list(map(int, input().split()))+[0] for _
in range(n)] + [[0 for _ in range(m+2)]]
new = deepcopy(old)
```

## 5.无终止输入

```
while True:
    try:
        n = input()
    except EOFError:
        break
```

## 6.使用 `isinstance` 函数

`isinstance` 函数检查一个对象是否是指定的类型或其子类。它可以接受多个类型参数。

```
x = 10
print(isinstance(x, int))  # True
y = 10.5
print(isinstance(y, float))  # True
z = "hello"
print(isinstance(z, str))  # True
a = [1, 2, 3]
print(isinstance(a, list))  # True
b = (1, 2, 3)
print(isinstance(b, tuple))  # True
c = {1, 2, 3}
print(isinstance(c, set))  # True
d = {"a": 1, "b": 2}
print(isinstance(d, dict))  # True
# 检查多个类型
print(isinstance(10, (int, float)))  # True
print(isinstance(10.5, (int, float)))  # True
print(isinstance("hello", (int, float)))  # False
```

## 6.集合

```
s = set(lis) #若列表为空则生成空集
s = {1,2,3} #手动生成
s.add(element)
s.remove(element) #没有会报错
s2 = s1.union(s)
s2 = s1.intersection(s)
```

## 7.防止忘记

```
#pop，删除指定索引处，默认-1
lis.pop(index)
a = lis.index(element)#第一个，没有会报错
n = lis.count(element)
#字典默认循环的是key
a = ord(b) #获取ascII码
b = chr(a)
lis = input(" ", 1)#后面一个数表示分割的次数
```

## 8.一次性读取

```
#Ctrl+D结束输入，在本地可操作
import sys
input_data = sys.stdin.read().split()
#得到的是一个一维表格，所有的" "与换行符都被去掉了，要用一个数来确定读到哪个数据了
```

## 9. 埃氏筛

```
#埃氏筛核心思想是从小的素数开始，将其倍数标为非素数，剩下的就是素数了
arbit = [False,False,True,True] + [False,True]*500000
i = 3
while i**2 <= 1000000:
    if arbit[i-1] == True:
        for j in range(i*2, 1000001, i):
            arbit[j-1] = False
    i += 1
#arbit就是素数列表
```

## 10. 欧拉筛

```
#欧拉筛基本是埃氏筛进化思想，排除一个数被反复筛，也就是每次筛停住，只构造质数相乘情况
def oula(r):
    # 全部初始化为0
    prime = [0 for i in range(r+1)]
    # 存放素数
    common = []
    for i in range(2, r+1):
        if prime[i] == 0:
            common.append(i)
        for j in common:
            if i*j > r:
                break
            prime[i*j] = 1
            #将重复筛选剔除
            if i % j == 0:
                break
    return common
#commen就是素数列表
#对于扫10^6内的埃氏筛只用扫1000,欧拉筛要扫1000000,故而埃氏筛更快
```