数据结构和算法
(Python描述)

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

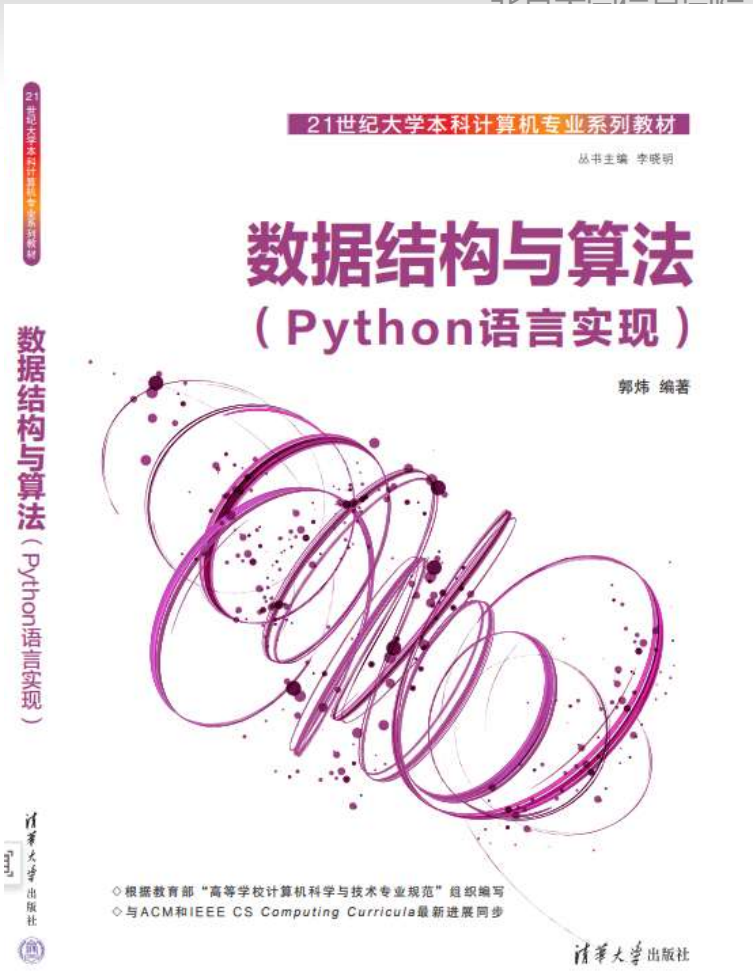
课程信息

➢ 教材

数据结构与算法（Python语言实现）

郭炜 编著

清华大学出版社



最短路

最短路
Dijkstra 算法

美国加州太浩湖

基本思想

- 解决无负权边的带权有向图或无向图的单源最短路问题

Dijkstra's Algorithm

dist[i]表示起点s到顶点 i 的距离, P表示最短路已经求出的顶点的集合

(1) 令dist[s]=0，其它dist[i]=+∞，P=∅
(2) 找到P外dist[i]最小的顶点i，将i加入P，s到i的最短路长度就是dist[i]
(3) 松弛操作：对于任意 i 的不属于P的邻点 j ，执行
dist[j] = min(dist[j], dist[i] + W(i,j))　　W(i,j)是i到j的边的权值
然后转步骤(2)

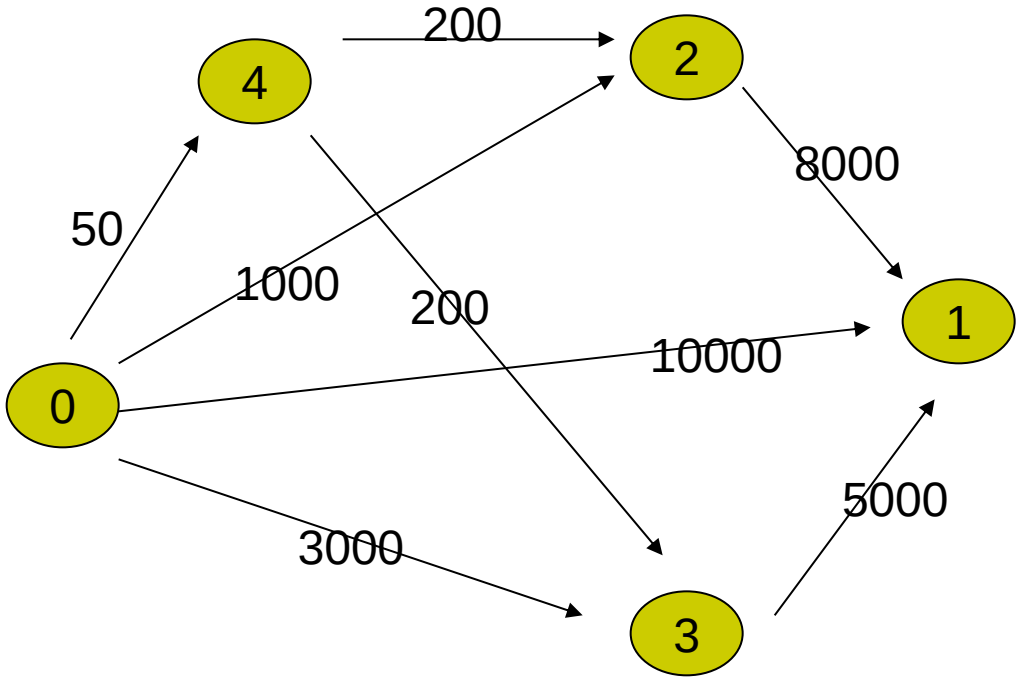
P中包含全部顶点时结束

Dijkstra's Algorithm

- 用邻接表，不优化，时间复杂度$O(V^2+E)$
- Dijkstra+堆的时间复杂度 o(ElgV)
- 用斐波那契堆可以做到O(VlogV+E)

- 若要输出路径，则设置prev数组记录每个顶点在最短路中的前趋点，
  在dist[j]更新时更新prev[j]

```
if (dist[j] > dist[i] + W(i,j))
        dist[j] = dist[i] + W(i,j)
        prev[j] = i
```



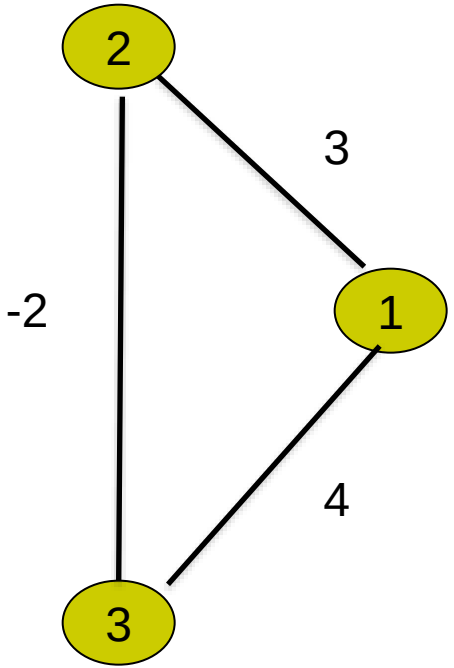
v	Dist[v]
0	0
1	1000
2	1200
3	3200
4	50

Dijkstra's Algorithm

Dijkstra算法也适用于无向图。但不适用于有负权边的图。

d[1,2] = 2

但用Dijkstra算法求得 d[1,2] = 3



POJ3159 Candies

有N个孩子（N<=3000)分糖果。
有M个关系(M<=150,000)。每个关系形如：

A B C　　(A,B是孩子编号)

表示A比B少的糖果数目，不能超过C

求第N个学生最多比第1个学生能多分几个糖果

POJ3159 Candies

思路：30000点，150000边的稀疏图求单源最短路

读入 “A B C”，就添加A->B的有向边，权值为C

然后求1到N的最短路

dijkstra 算法解决 POJ 3159 Candies

```
class Edge:
        def __init__(self, v, w):
                self.v, self.w = v, w #边终点v,权值w
def dijkstra(G): #G是邻接表，顶点从0开始编号，源点是顶点0
        INF, N = 10 ** 9, len(G)
        done = [False for i in range(N)]
        #done[i]为True表示源到i的最短路已经求出
        prev = [None for i in range(N)]   #前驱
        dist = [INF  for i in range(N)]   #目前最短路长度
        dist[0] = 0
        doneNum = 0                 #最短路已经求得的顶点数目
        while  doneNum < N:
                x,minDist = None,INF
                for k in range(0,N):
                        if not done[k] and dist[k] < minDist:
                                minDist = dist[k]
                                x = k
```

```
        if minDist == INF:
#剩下的点都从源不可达了，它们的最短路都是无穷长
                break
        done[x] = True   #顶点x的最短路现在求出了,即dist[x]
        doneNum += 1
        for e in G[x]:
                j = e.v
                if not done[j]:
                        if dist[j] > dist[x]+ e.w:
                                dist[j] = dist[x]+ e.w
                                prev[j] = x
return dist,prev
```

```
N,M = map(int,input().split())
G = [[] for i in range(N)]
for i in range(M):
    s,e,w = map(int,input().split())
    G[s-1].append(Edge(e-1,w)) #dijkstra函数要求图顶点从0开始编号
dist,prev = dijkstra(G)
print(dist[N-1])
path = []  # 存放1到N的最短路径
p = N-1
while p is not None:
       path.append(p+1)
       p = prev[p]
path.reverse()
```

用heap实现 dijkstra + 堆的 POJ 3159 Candies

```
import heapq
class Edge:
    def __init__(self,k=0,w=0):
        self.k ,self.w = k,w   #有向边的终点和边权值，或当前k到源点的距离
    def __lt__(self,other):
        return self.w < other.w

bUsed = [0 for i in range(30010)]# bUsed[i]为1表示源到i的最短路已经求出
INF = 100000000
N,M = map(int,input().split())
G = [[] for i in range(N+1)]
for i in range(M):
    s,e,w = map(int,input().split())
    G[s].append(Edge(e,w))
pq = []
heapq.heapify(pq)
heapq.heappush(pq,Edge(1,0)) #源点是1号点,1号点到自己的距离是0
```

用heap实现 dijkstra + 堆的 POJ 3159 Candies

```
while pq != []:
    p = pq[0]
    heapq.heappop(pq)
    if bUsed[p.k]: #已经求出了最短路
        continue
    bUsed[p.k] = 1
    if p.k == N:  #因只要求1-N的最短路，所以要break
        break
    L = len(G[p.k])
    for i in range(L):
        q = Edge()
        q.k = G[p.k][i].k
        if bUsed[q.k]:
            continue
        q.w = p.w + G[p.k][i].w
        heapq.heappush(pq,q) #队列里面已经有q.k点也没关系
print( p.w )
```

floyd算法

美国加州1号公路

弗洛伊德算法

- 用于求每一对顶点之间的最短路径。有向图，无向图均可。有向图可以有负权边，但是不能有负权回路。

弗洛伊德算法

- 用于用于求每一对顶点之间的最短路径。有向图，无向图均可。有向图可以有负权边，但是不能有负权回路。
- 假设求从顶点v_i到v_j的最短路径。如果从v_i到v_j有边，则从v_i到v_j存在一条长度为cost[i,j]的路径，该路径不一定是最短路径，尚需进行n次试探。

弗洛伊德算法

- 用于用于求每一对顶点之间的最短路径。有向图，无向图均可。有向图可以有负权边，但是不能有负权回路。
- 假设求从顶点v_i到v_j的最短路径。如果从v_i到v_j有边，则从v_i到v_j存在一条长度为cost[i,j]的路径，该路径不一定是最短路径，尚需进行n次试探。
- 考虑路径（v_i，v_1，v_j）是否存在（即判别弧（v_i，v_1）和（v_1，v_j）是否存在）。如果存在，则比较cost[i,j]和（v_i，v_1，v_j）的路径长度，取长度较短者为从v_i到v_j的中间顶点的序号不大于1的最短路径，记为新的cost[i,j]。

弗洛伊德算法

- 用于用于求每一对顶点之间的最短路径。有向图，无向图均可。有向图可以有负权边，但是不能有负权回路。
- 假设求从顶点v_i到v_j的最短路径。如果从v_i到v_j有边，则从v_i到v_j存在一条长度为cost[i,j]的路径，该路径不一定是最短路径，尚需进行n次试探。
- 考虑路径（v_i，v_1，v_j）是否存在（即判别弧（v_i，v_1）和（v_1，v_j）是否存在）。如果存在，则比较cost[i,j]和（v_i，v_1，v_j）的路径长度，取长度较短者为从v_i到v_j的中间顶点的序号不大于1的最短路径，记为新的cost[i,j]。
- 假如在路径上再增加一个顶点v_2，如果（v_i，…，v_2）和（v_2，…，v_j）分别是当前找到的中间顶点的序号不大于2的最短路径，那么（v_i，…，v_2，…，v_j）就有可能是从v_i到 v_j的中间顶点的序号不大于2的最短路径。将它和已经得到的从v_i到 v_j的中间顶点的序号不大于1的最短路径相比较，从中选出中间顶点的序号不大于2的最短路径之后，再增加一个顶点v_3，继续进行试探。依次类推。

弗洛伊德算法

- 在一般情况下，若（v_i，…，v_k）和（v_k，…，v_j）分别是从v_i到v_k和从v_k到v_j的中间顶点的序号不大于k-1的最短路径，则将（v_i，…，v_k，…，v_j）和已经得到的从v_i到v_j且中间顶点的序号不大于k-1的最短路径相比较，其长度较短者便是从v_i到v_j的中间顶点的序号不大于k的最短路径。这样，在经过n次比较后，最后求得的必是从v_i到v_j的最短路径。按此方法，可以同时求得各对顶点间的最短路径。
- 复杂度$O(n^3)$。不能处理带负权边的无向图，和有负权回路的有向图

弗洛伊德算法

- 记$dist^k(i,j)$为从Vi到Vj的途经的顶点编号不大于k的最短路长度，则有：

$dist^{-1}(i,j) = W_{i,j}$（$W_{i,j}$是边(i,j)权值，边不存在则为无穷大）

$dist^0(i,j) = \min\{dist^{-1}(i,j), dist^{-1}(i,0) + dist^{-1}(0,j)$

$dist^1(i,j) = \min\{dist^0(i,j), dist^0(i,1) + dist^0(1,j)$

……

$dist^k(i,j) = \min\{ dist^{k-1}(i,j),\ dist^{k-1}(i,k) + dist^{k-1}(k,j)\}$

……

$dist^{n-1}(i,j) = \min\{ dist^{n-2}(i,j),\ dist^{n-2}(i,n-1) + dist^{n-2}(n-1,j)\}$

其中$dist^{-1}(i,j)$表示从Vi到Vj的不途经任何顶点的最短路径长度。

$dist^{n-1}(i,j)$就是Vi到Vj的最短路的长度

弗洛伊德算法实现

```
def  floyd(G): #G是邻接矩阵，顶点编号从0开始算,无边则边权值为INF
    n = len(G)
    INF = 10**9
    prev = [[None for i in range(n)] for j in range(n)]
    #prev[i][j]表示到目前为止发现的从i到j的最短路上，j的前驱。
    dist = [[INF for i in range(n)] for j in range(n)]
    for i in range(n):
         for j in range(n):
                  if i == j:
                           dist[i][j] = 0
                  else:
                           if G[i][j] != INF: #i到j的边存在
                                    dist[i][j] = G[i][j]
                                    prev[i][j] = i
```

弗洛伊德算法实现

```
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
                    prev[i][j] = prev[k][j]
    return dist,prev
```

最终：dist[i][j]就是i到j的最短路

prev[i][j]是i到j的最短路上j的前驱， prev[i][prev[i][j]]是j的前驱的前驱.....

例题： POJ3660 Cow Contest

N个选手，如果A比B强,B比C强，则A必比C强

告知若干个强弱关系，问有多少人的排名可以确定

- **Sample Input**

5 5

4 3

4 2

3 2

1 2

2 5

5个人，5个胜负关系

4 比3强

4 比2强

3 比2强

…..

- **Sample Output**

2

例题： POJ3660 Cow Contest

如果一个点u, 有x个点能到达此点，从u点出发能到达y个点，若x+y=N-1，则u点的排名是确定的。用floyd算出每两个点之间的距离，最后统计，若dist[a][b] 无穷大且dist[b][a]无穷大，则a和b的排名都不能确定。最后用点个数减去不能确定点的个数即可。

模版例题

POJ1125