PS12--201300086 史浩男

一、SSSP画图

(a) Bellman-Ford

```
x.d=9,x.p=z ,s.d=2,s.p=z ,t.d=8,t.p=s ,y.d=9,y.p=s
```

```
y.d=9,y.p=s ,t.d=7,t.p=x ,x.d=6,x.p=y ,s.d=2,s.p=z
```

```
t.d=4,t.p=x ,y.d=9,y.p=s ,x.d=6,x.p=y ,s.d=2,s.p=z
```

```
t.d=4,t.p=x ,y.d=9,y.p=s ,x.d=6,x.p=y ,s.d=2,s.p=z
```

(b)DAGSSSP

```
s.d=5,s.p=r ,t.d=8,t.p=r
```

```
s.d=5,s.p=r ,t.d=7,t.p=s ,x.d=11,x.p=s
```

```
s.d=5,s.p=r ,t.d=7,t.p=s ,x.d=11,x.p=s ,y.d=11,y.p=t ,z.d=9,z.p=t
```

```
s.d=5,s.p=r ,t.d=7,t.p=s ,x.d=11,x.p=s ,y.d=10,y.p=x ,z.d=9,z.p=t
```

```
s.d=5,s.p=r ,t.d=7,t.p=s ,x.d=11,x.p=s ,y.d=10,y.p=x ,z.d=8,z.p=y
```

二、APSP画图

(a) Floyd-Warshall

- **陷阱都在负边**

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -3 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 12 & \infty & \infty & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 6 & \infty & \infty & -1 & \infty \\ -1 & 0 & \infty & 2 & 0 & \infty \\ 3 & -3 & 0 & 4 & \infty & -8 \\ -3 & 10 & \infty & 0 & -4 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 12 & 7 & \infty & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -1 & 0 & \infty & 2 & -2 & \infty \\ -2 & -3 & 0 & -1 & 2 & -8 \\ -3 & 3 & \infty & 0 & -4 & \infty \\ 6 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 12 & 7 & 10 & 0 \end{pmatrix}$$

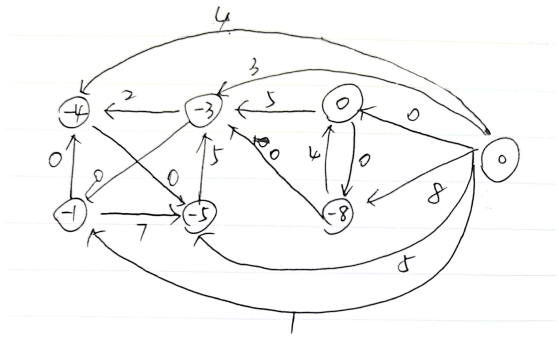
$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -1 & 0 & \infty & 2 & -2 & \infty \\ -4 & -3 & 0 & -1 & -3 & -8 \\ -3 & 3 & \infty & 0 & -4 & \infty \\ 6 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 12 & 7 & 3 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -1 & 0 & \infty & 2 & -2 & \infty \\ -4 & -3 & 0 & -1 & -5 & -8 \\ -3 & 3 & \infty & 0 & -4 & \infty \\ 6 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 12 & 7 & 3 & 0 \end{pmatrix}$$

(b) Johnson’s

```
h(1)=-4,h(2)=-3,h(3)=0,h(4)=-1,h(5)=-5,h(6)=-8
```

- z到每个点的最小路径取每列的min



- 更新边的权：新权=旧权+源点赋值-汇点赋值

```
w(1,5)=0
w(2,1)=2,w(2,4)=0
w(3,2)=5,w(3,6)=0
w(4,1)=0,w(4,5)=7
w(5,2)=5
w(6,2)=0,w(6,3)=4
其余为NIL
```

任意两点间最短路径：

$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -1 & 0 & \infty & 2 & -2 & \infty \\ -4 & -3 & 0 & -1 & -5 & -8 \\ -3 & 3 & \infty & 0 & -4 & \infty \\ 6 & 7 & \infty & 9 & 0 & \infty \\ 4 & 5 & 12 & 7 & 3 & 0 \end{pmatrix}$$

三、拓扑

（a）计算路径数

```
count(G,u):
    if (u==t)
        return 1
    else
        sum=0
```

```
        for(each edge (u,v) in E)
            if(r[v]==0)//防止重复调用
                r[v]=count(G,v)
            sum=sum+r[v]
        return sum
```

```
Main(G,s,t)
for (each u in V)
    r[u]=0//记录子问题的解：每个结点作为源点的路径数
return count(G,s)
```

时间

- 每条边都会对应一个递归分支，共有|E|个
- 但以每个结点为源点的递归分支都只会真正递归一次，最多|v|次
- 每个最终小问题的时间为O（1），合并时间也是O（1）
- 所以一共$O(|V| + |E|)$

（b）计算最早开始时间

- 用u.d记录最早开始时间

```
RELAXearly(u,v)
if v.d>u.d+w(u)
    v.d=u.d+w(u))
```

```
DAGSSSP(G,s):
for (each u in V)
  u.d=INF
s.d = 0
Run DFS to obtain topological order
for (each node u in topological order)
  for (each edge (u,v) in E)
```

```
8       RELAXearly(u,v)
```

时间

- 拓扑排序只需DFS，时间$O(|V| + |E|)$
- 每次RELAXearly操作O（1），一共进行O（|E|）次
- 总时间$O(|V| + |E|)$

（c）计算最晚开始时间

- 用u.d记录最晚开始时间

```
RELAXlate(u,v)
if v.d<u.d+w(u)
    v.d=u.d+w(u))
```

```
DAGSSSP(G,s):
for (each u in V)
  u.d=0
Run DFS to obtain topological order
for (each node u in topological order)
  for (each edge (u,v) in E)
    RELAXlate(u,v)
```

时间

- 拓扑排序只需DFS，时间$O(|V| + |E|)$
- 每次RELAXlate操作O（1），一共进行O（|E|）次
- 总时间$O(|V| + |E|)$

四、单源最长路径

问题转化：

由于边权都是负的，且要求SSSP
则等价于把所有边的权都取相反数，然后我们寻找单源最长路径

```
RELAX(u,v)
    if u.d<0//圈中和圈后结点都是负无穷
        v.d=-NIL
    else if v.d<u.d+w(u,v)
        v.d=u.d+w(u,v))
```

```
FINDLONGEST(G,s)
    for (each u in V)
        u.d=0
    s.d=0
    for (each edge (u,v) in E)
        RELAX(u,v)
    //一次循环后，负圈检测
    for (each edge (u,v) in E)
        if (v.d > u.d + w(u,v))
            u.d=-NIL ,v.d=-NIL
    repeat n-2 times:
        for (each edge (u,v) in E)
            RELAX(u,v)
    for (each u in V)//不可到达的边是正无穷
        if(u.d==0 and u!=s)
            u.d=NIL
```

正确性

- 不可到达的点，初始为0，最后还是0，所以在代码最后把除源点外所有0的赋值为正无穷
- 如果一次循环RELAX后，存在（u，v）但u的路径更长，那么一定是因为u和v都在环中
- 1+n-2次循环，保证了所有圈外顶点路径长度达到最大值，所有圈内和圈后顶点都赋值为正无穷

时间

- 循环重复了n次，每次时间|E|
- 初始化、负圈检测正无穷检测都是线性时间
- RELAX是O（1）

- 总时间O（|V|*|E|）

五、

(a)

$$
\begin{gathered}
\text{首先，} \frac{\sum_i p_i}{\sum_{i,j} c_{ij}} \leq r^* \\
\text{即} r^* \sum_{i,j} c_{ij} \geq \sum_i p_i \\
\text{若存在负圈，则} \sum_i w_i \leq 0, \text{即} r \sum_{i,j} c_{ij} - \sum_i p_i \leq 0 \\
\text{于是有} r^* \sum_{i,j} c_{ij} \geq \sum_i p_i \geq r \sum_{i,j} c_{ij} \\
\text{所以} r \leq r^*
\end{gathered}
$$

(b)

$$
\begin{gathered}
\text{题意为任意圈满足} \sum_i w_i \geq 0 \\
\text{即} r \sum_{i,j} c_{ij} - \sum_i p_i \geq 0, \ \frac{\sum_i p_i}{\sum_{i,j} c_{ij}} \leq r \\
\text{而} \frac{\sum_i p_i}{\sum_{i,j} c_{ij}} \text{的最大值是} r^* \\
\text{所以} r > r^*
\end{gathered}
$$

（c)

分析

- 题目没有说明这样的cycle一定要包含所有结点
- R代表了对我们最有利的一条边的比值，R>r*一定成立

六、图构造

（a）贪心

思路

- 把每个电池用到极致，直至不能完成下一小段，再换

```
Count():
    num=1,i=1,remain=100
    while(i!=n){
        if(remain-D[i+1]+D[i]>=0)
            remain=remain-D[i+1]+D[i]
            i=i+1;
        else
            remian=100
            num=num+1
            i=i+1
    }
    return num
```

- 遍历D数组，时间O（n）

（b）反例

```
D[1...n]={0,90,100,180,200,270}
C[1...n]={1,1,10,1,10,1}
```

- 采用a中的贪心算法，总cost为21
- 但实际总cost3就够了

（c）构造SSSP

构造图

- 每个加油站看作点，如果两个加油站(u,v)的距离不超过100km，就连一条边
- w（u，v）=C[u]，把出发加油站的换电池cost记为这条边的权
- 问题转化为，在权为正且存在拓扑排序的有向无圈图中寻找从起点到终点的最短路径

```
RELAX(i,j):
    if A[j]>A[i]+C[i]
        A[j]=A[i]+C[i]
```

```
DAGSSSP(G):
    for (i=1 to n)
```

```
        A[i]=INF//记录路径长度上界
    A[1]=C[1]
    for (i=1 to n-1)
        for (j=i+1 to n)
            if(D[j]-D[i]<=100)
                RELAX(i,j)
            else
                break
    //倒序查找停下加油的位置
    LIFO Q
    k=n
    for(i=n-1 downto 1)
        if(A[k]=A[i]+C[i])
            Q.enqueue(i)
            k=i
    return Q
```

时间

- 初始化，倒叙查找O（n）
- 主循环O（n+|E|）
- 当加油站之间相隔比较远，接近100km时，总时间为O（n）
- 当加油站相隔比较近，100km内有大量加油站时，总时间O（n^2）

七、递归

思路

- 先检测出所有包含第一个字母是单词的情况，然后把所有情况展开递归，加和

```
COUNT(A,p):
    if(p>n)
        return 1
    sum[p]=0
    for(i = p to n)
        if(ISWORD(A,p,i))
            if(sum[i+1]>=0)//减少重复递归
                sum[p]=sum[p]+sum[i+1]
```

```
            else
                sum[p]=sum[p]+COUNT(A,i+1)
    return sum[p]
```

```
Main(A[1,2....n]):
    for(i=1 to n)
        sum[i]=-1//记录COUNT(A,i)的返回值，代表A[i,i+1......n]的分割方法总数
    return COUNT(A,1)
```

时间

法一：数列递推：

- 记调用次数为A[n]，时间为T[n]
- 最坏情况下，任意个数相邻的任意字符串都是单词，调用次数A[n]=n+A[n-1]+A[n-2]+……+A[1]=O(2^n)。
- 但由于A[1]到A[n-1]都已求出，（减少重复性递归已在代码中体现）所以对应的T[n]=n+T[n-1]+O(1)+O(1)+O(1)+O(1)+…+O(1)=O（n）+T[n-1]
- 根据此数列递推式，可以求出T[n]=O(n^2)

法二：递归树分析：

- 每个结点只有COUNT(A,1),COUNT(A,2).....COUNT(A,n)共n种情况
- 由于加入了减少重复递归的代码，每个结点只会递归展开一次，再之后调用时可以O（1）时间直接返回子问题答案sum[i]
- 因此总时间为n+n-1+n-2+…+1+O(n)=O(n^2)