PS11--201300086 史浩男

1、SSSP-Dijkstra

（a）

源点s

```
R={s,t},t.d=4,t.p=s
R={s,t,y},y.d=6,y.p=t//错，y.d=5,y.p=s
R={s,t,y,x},x.d=10,x.p=y
R={s,t,y,x,z},z.d=12,z.p=x
```

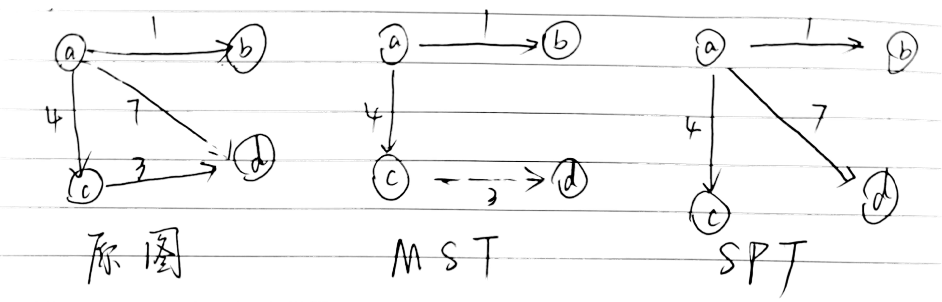
源点z

```
R={z,x},x.d=1,x.p=z
R={z,x,s},s.d=4,s.p=z//错，s.d=3
R={z,x,s,t},t.d=8,t.p=s
R={z,x,s,t,y},y.d=10,y.p=t
```

(b)负边反例

题目没有对负环进行限制，所以只要反例中存在负环，则为负无穷，矛盾

(c)MST与SPT不同



原因：虽然a->c->d和a->d权重相同，但执行Dijktra算法时，会保留d.dist，而不是替换成c.dist+w(c,d)

2、最大积路径

```
DijkstraSSSP(G,s):
for (each u in V)
  u.d=INF, u.parent=NIL
s.d = 0
Build priority queue Q based on dist
FIFO P
while (!Q.empty())
    u = Q.ExtractMax()//O(nlgn)
    P.enque(u)
   for (each edge (u,v) in E)//O(mlgn)
      if v.d<u.d*w(u,v)
        v.d=u.d*w(u,v)
        v.parent=u
      if v==t
        P.enque(v)
        return P
```

- 用FIFO存储路径
- 时间O（（|V|+|E|）lg|V|）

3、路径重构

(a) 已知拓扑

思路

- 先调用一遍Dijkstra，计算出那条包含所有点的路径，并用FIFO P按拓扑顺序存储点,用u.parent、u.son属性存储路径中的点相邻关系
- 如果删去的边不在那条路径中，返回t.d即可
- 如果在那条路径中，则需要调用Findmin(G\(u,v))，计算删去(u,v)后的新最短路径总权和

```
DijkstraSSSP(G,s):
    for (each u in V)
      u.d=INF, u.parent=NIL,u.son=NIL,u.replace=0
    s.d = 0
```

```
    Build priority queue Q based on dist
    FIFO P//按拓扑顺序存储点
    while (!Q.empty())
        u = Q.ExtractMin()
        P.enque(u)
       for (each edge (u,v) in E)
          if v.d>u.d+w(u,v)
              v.d=u.d+w(u,v)
              v.parent=u
              u.son=v
          if v==t
              P.enque(v)
```

```
1 Findmin(G\(u,v))
2     x=u.parent
3     y is an vertice after v if there is (u,y)
4     u.replace=min{u.d+w(u,y)+t.d-y.d}
5     u.replace=min{Findmin(G\(x,u)),u.replace}
6     //计算删去(u,v)后的新最短路径总权和
```

```
Main(G,s,t)
    DijkstraSSSP(G,s)........................O((|V | + |E|) lg |V |)
    Stack R//记录结果
    for (each edge (u,v) in E)...............O(|E|lg|V|)
        if u.son!=v//不在最短路径中...........O(|1|)
            R.push(t.d)
        else
            while(!P.empty())
                u=P.dequeue()
               v=u.son
               R.push(Findmin(G\(u,v)))...........O(|E|)
```

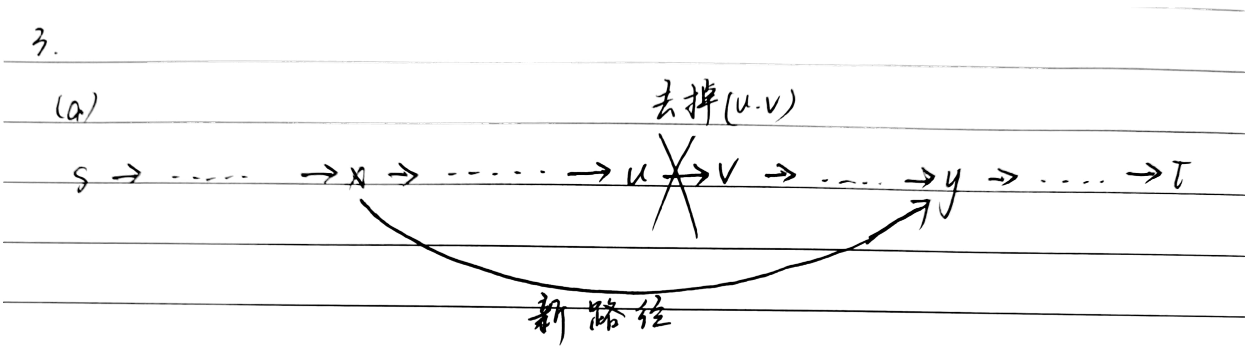
时间

只需单独解释，为什么所有Findmin(G\(u,v))的时间是O(|V|lg|V|)，只看Findmin第四行和第五行即可

- 第五行一共调用了O(|V|)次，即u.replace=min{Findmin(G\(x,u)),u.replace}
- 每次第五行都在利用上一次Findmin的结果，只需要O（1）即可
- 所有第五行调用时间为O（V）
- 第四行一共调用了O(|V|)次，即u.replace=min{u.d+w(u,y)+t.d-y.d}
- 每次调用第四行，都是需要从O（degree（u））条不同路径中寻找最小的，时间为O（degree（u））
- 所有第四行调用时间为O（Σdegree（u））=O(|E|)

所以总时间为O((|V | + |E|) lg |V |)

正确性



只需解释Findmin(G\(u,v))能正确找到删去(u,v)后的新最短路径

- 记包含所有点的那条路径为P
- 路径P满足：对于任意两点x，y，x----->y的最短路径是P的子路径

1.如果删掉的边（u,v）不在P中：很简单，无需改变P

2.如果删掉的边（u,v）在P中：

- 我们只需要找到一条能“代替”被去掉的（u，v）的边即可，这样的边一定从u或u之前的某个点出发，指向了v或v之后的某个点，记为（x，y）
- 如果找到了这样的一条替代边（x，y），则新最短路径一定是s->.....->x->y->.....->t，也就是说，在路径P中把子路径x->....->u->v->....->y用x->y这一条有向边替换。这是因为，从s到x和从y到t的最短路径一定在原来的路径P中，无需更改
- 这个新最短路径的权和为x.d+w(x,y)+t.d-y.d
- 为了节省时间，每次Findmin(G\(u,v))总可以利用之前Findmin的结果：我们对x进行讨论：
    1、如果x在u之前，那么利用Findmin(G\(u.parent,u))的结果即可；
    2、如果x就是u，那么需要查找min{u.d+w(u,y)+t.d-y.d}
    （这个对x的讨论过程，就是Findmin函数中第四第五行的工作）

（b）任意无向图

思路

- 先调用一遍Dijkstra，计算s到t的最短路径，并用FIFO P按顺序存储这条路径上的点
- 如果删去的边不在那条路径中，返回t.d即可
- 如果在那条路径中，则需要调用Findmin(G\(u,v))，计算删去(u,v)后的新最短路径总权和

代码、时间复杂度分析和（a）相同

正确性

4、活动选择--多策略判定

- 用二元对（x，y）来表示每个活动的开始和结束时间，假设总时间为10

(a)ends last

反例：(1,2)(3,4)(5,6)(1,10)

(b)starts first

反例：(1,10)(3,4)(5,6)(7,10)

(c)starts last

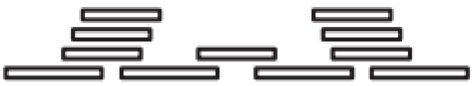
正确

(d)shortest duration

反例：(1,5)(5,6)(6,10)

(e)conflicts fewest

反例：中间那个冲突最少，如果选了，那最多只有三个，而不是4个

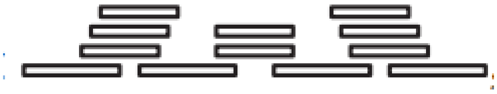


(f)discard longest duration

反例：(1,5)(5,6)(6,10)

(g)discard conflicts most

反例：最底层的中间两个会被去掉，使最大只有三个，而不是4个



(h)discard cover+end last

正确

5、活动选择-最小覆盖

思路

- 最先开始的和最后结束的一定要选
- 把所有活动按开始时间升序排列，然后选择第一个
- 如果后续有与刚刚选择的这个活动冲突的活动，则遍历这些，选出结束时间最晚的一个，作为新的添加；如果没有则直接选择下一个活动
- 重复，直到最后结束的那个活动被添加

```
Cover(L[1,n],R[1,n]):
    Sort L[1,n] in increasing order and store it in l[1,n]
    Record the index changes by f[l[i]]=i
    then we have r[1,n] that R[f[l[i]]]=r[i]
    priority queue A//记录答案
    A.add(1)
    x=r[1]//记录新选活动结束时间
    i=2//记录遍历到第几个
    while(x!=r[n])
        if(l[i]<x)
            New priority queue P//每次循环后清空
            while(l[i]<x)
                P.enque(r[i])
```

```
                i=i+1
            u=P.ExtractMax()//选出结束时间最晚的一个
            j=f[u]//记录index，作为新添加
            A.add(j)
            x=r[j]
        else//没有与刚刚选择的这个活动冲突的活动，直接选择下一个
            A.add(i)
            x=r[i]
            i=i+1
    return A
```

时间

- 把L数组重排，O（n）
- P.ExtractMax()的总时间为O（n）：假设每次P的规模为m个，则每次存入并选出一个max的时间为O（m）。而i<=n，每次P涉及的i的区间都不重复且越来越大，所以O（n）

总时间O（n）

正确性

1、由于2n个endpoints都不同，所以可以建立f[l[i]]=i这种从数组值来反过来索引index的数组

2、贪心安全：可以完全覆盖：

- 如果后续有与刚刚选择的这个活动冲突的活动：则选择一个冲突且最晚结束的，保证了从头部一直到当前的覆盖
- 如果没有，则活动之间出现了“裂缝”，直接选择下一个
- 以上两种情况中，最后一个活动一定会被选择，保证了从头到尾的全面覆盖

3、是最小覆盖：

- 每一步都是贪心策略，在保证覆盖的前提下，把覆盖的范围最大化

6、找零钱

（a）反例

1，2，7，11

解释：如果我们要14，则按照贪心是1+2+11，但其实7+7就可以完成

（b）等比硬币

证明：

对于硬币序列$1, b, b^2, \cdots, b^k$

我们把所有面值化为b进制进行分析

由于每个硬币一定代表了一个b进制的位，因此最小的硬币数一定是这个b进制所有位数之和

而我们的贪心算法所需硬币数刚好是b进制数所有位数之和，刚好满足这个下界

因此这个每次取最大的这个贪心算法是最优的