Algorithm 0x02

49.234.77.58/index.php/2019/12/15/algorithm-0x02

XZLang

2019年12月15日

贪心算法

贪心算法（又称贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。

贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。

思想

贪心算法的基本思路是从问题的某一个初始解出发一步一步地进行，根据某个优化测度，每一步都要确保能获得局部最优解。每一步只考虑一个数据，他的选取应该满足局部优化的条件。若下一个数据和部分最优解连在一起不再是可行解时，就不把该数据添加到部分解中，直到把所有数据枚举完，或者不能再添加算法停止

过程

1. 建立数学模型来描述问题；
2. 把求解的问题分成若干个子问题；
3. 对每一子问题求解，得到子问题的局部最优解；
4. 把子问题的解局部最优解合成原来解问题的一个解。

普利姆算法

```
Prim(G)
1.  VT={v0};
2.  ET=Φ;
3.  For i=1 to n-1 do
      find a minimum weight edge e*=(u*, v*) among
      all the edges (u, v) such that u is in T and v is in V-T;
      VT=VT∪{v*};
      ET=ET∪{e*};
4. Return ET.
```

库鲁斯卡尔算法

ALGORITHM Kruscal(G)

1. *Sort E in nondecreasing order of the edge weights*
   $w(e_{i1}) <= \ldots <= w(ei_m)$
2. $E_T = \varnothing$; *ecounter* $= 0$ *//initialize the set of tree edges and its size*
3. ***k = 0***
4. ***while*** *encounter* $< |V| - 1$ ***do***
   k = k + 1
   if $E_T \cup \{e_{ik}\}$ ***is acyclic***
   $E_T = E_T \cup \{e_{ik}\}$; ***ecounter = ecounter + 1***
5. *return* E_T;

some implementation details

makeset(x): create a singleton set containing just x

find(x): to which set does x belong?

union(x, y): merge the sets containing x and y

Kruskal(G)

1. For all u∈V do
   makeset(u);
2. X={∅};
3. Sort the edges E by weight;
4. For all edges (u, v) ∈ E in increasing order of weight do
   if find(u) ≠find(v) then
   add edge (u, v) to X;
   union(u, v);

迪杰斯特拉

Dijkstra's algorithm

Dijkstra(G, s)

1. For all u∈V do

   dist(u)=∞;

   P(u)=nil;

2. Dist(s)=0;

3. H=makequeque(V);

4. While H is not empty do

   u=deletemin(H);

   for all edges (u, v) ∈E do

   if dist(v)>dist(u)+w(u,v) then

   dist(v)=dist(u)+w(u,v);

   P(v)=u;

   decreasekey(H, v);

BELLMAN-FORD

松弛：

每次松弛操作实际上是对相邻节点的访问(相当于广度优先搜索)，第n次松弛操作保证了所有深度为n的路径最短。由于图的最短路径最长不会经过超过|V| − 1条边，所以可知贝尔曼-福特算法所得为最短路径，也可只时间复杂度为O(VE)。

负边权操作：

与迪科斯彻算法不同的是，迪科斯彻算法的基本操作"拓展"是在深度上寻路，而"松弛"操作则是在广度上寻路，这就确定了贝尔曼-福特算法可以对负边进行操作而不会影响结果。

负权环判定：

因为负权环可以无限制的降低总花费，所以如果发现第n次操作仍可降低花销，就一定存在负权环。

基本操作：

1. 创建源顶点 v 到图中所有顶点的距离的集合 distSet，为图中的所有顶点指定一个距离值，初始均为 Infinite，源顶点距离为 0；
2. 计算最短路径，执行 V – 1 次遍历；
3. 对于图中的每条边：如果起点 u 的距离 d 加上边的权值 w 小于终点 v 的距离 d，则更新终点 v 的距离值 d；
4. 检测图中是否有负权边形成了环，遍历图中的所有边，计算 u 至 v 的距离，如果对于 v 存在更小的距离，则说明存在环(无向图不能用这种方法判断负环)

正确性：

Bellman-Ford 算法采用动态规划进行设计，实现的时间复杂度为 O(V*E)，其中 V 为顶点数量，E 为边的数量。

```
Bellma-Ford(G, s)
1. For all u∈V do
      dist(u)=∞;
      P(u)=nil;
2. Dist(s)=0;
3. Repeat |V|-1 times
      for all e=(u, v) ∈E do
        dist(v)=min(dist(v), dist(u)+w(u,v)};
4. for all e=(u, v) ∈E do
        if dist(v) is changed then
      return "negative cycle";
```

SPFA算法的思路：
我们用数组dist记录每个结点的最短路径估计值，用邻接表或邻接矩阵来存储图G。我们采取的方法是动态逼近法：设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作，如果v点的最短路径估计值有所调整，且v点不在当前的队列中，就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止