最短路

PKU Hzyoi

01 最短路

图的基本概念

- 图(Graph)是一个有序（就是V和E不能互换，不能叫做(E,V)）二元组(V,E)，其中V称为顶集(Vertices Set),E称为边集(Edges Set),E与V不相交。它们也可以写成V(G)和E(G)。
- 有/无向图：如果给图的每条边规定一个方向，那么得到的图称为有向图。在有向图中，与一个节点相关联的边有出边和入边之分。相反，边没有方向的图称为无向图。

附：图的基本术语

- 阶(Order)：图G中顶集V的大小（顶点的多少）称作图G的阶。
- 子图(Sub-Graph)：当图G'=(V',E')其中V'包含于V，E'包含于E，则G'称作图G=(V,E)的子图。每个图都是本身的子图。
- 生成子图(Spanning Sub-Graph)：指满足条件V(G') = V(G)的G的子图G。换种说法就是图G中所有的顶点在图G'中都存在。
- 导出子图(Induced Subgraph)：以图G的顶点集V的非空子集V1为顶点集，以两端点均在V1中的全体边为边集的G的子图，称为V1导出的导出子图（如果两个G'中的顶点在图G中有边连接，那么这条边一定存在于G'中）；以图G的边集E的非空子集E1为边集，以E1中边关联的顶点的全体为顶点集的G的子图，称为E1导出的导出子图（和上面讲的差不多一个意思）。
- 度(Degree)：一个顶点的度是指与该顶点相关联的边的条数（连接着这个顶点的边的条数），顶点v的度记作d(v)。

图的基本术语

- 入度(In-degree)和出度(Out-degree)：对于有向图来说，一个顶点的度可细分为入度和出度。一个顶点的入度是指与其关联的各边之中，以其为终点的边数（就是连进来的边数）；出度则是相对的概念，指以该顶点为起点的边数（就是连出去的边数）。
- 自环(Loop)：若一条边的两个顶点为同一顶点，则此边称作自环。
- 路径(Path)：从u到v的一条路径是指一个序列$v_0,e_1,v_1,e_2,v_2,\ldots e_k,v_k$，其中$e_i$的顶点为$v_i$及$v_{i-1}$，k称作路径的长度。如果它的起止顶点相同，该路径是“闭”的，反之，则称为“开”的。一条路径称为一简单路径(simple path)，如果路径中除起始与终止顶点可以重合外，所有顶点两两不等。
- 行迹(Trace)：如果路径P(u,v)中的边各不相同，则该路径称为u到v的一条行迹。
- 轨道/简单路径(Track)：如果路径P(u,v)中的顶点各不相同，则该路径称为u到v的一条轨道。
- 闭的行迹称作回路(Circuit)，闭的轨道称作圈(Cycle)。
- 桥(Bridge)：若在一张无向图中，去掉其中一条边，便会使得整个图连通块数量增加，那么该边称为桥，也叫割边。
- 割点(Cut-vertex)：若在一张无向图中，去掉其中一个点及其相连的所有边，便会使得整个图连通块数量增加，那么该点称为割点。

最短路径问题

- 最短路径问题(Shortest-path Problem)：若图中的每条边都有一个权值（长度、成本、时间等），则找出两顶点之间总权和最小的路径就是最短路径问题。
- 最短路径问题是图论的典型问题之一，可用来解决管路铺设、线路安装、厂区布局和设备更新等实际问题，在信息学竞赛中也频繁地出现。

定义及性质

- 多源最短路问题：对于点集V中的每一个点v，求出v到其他所有点的最短路。
- 单源最短路问题：对于给定一个源点s ∈ V，求出s到其他所有点的最短路。
- 最优子结构性质：
  - 如果P(i, j) = {Vi, ..., Vk , ..., Vl, ..., Vj}是从i到j的最短路径，
  - k和l 是P(i, j)上的两个中间点，那么P(k, l)必定是从k到l的最短路径。
  - 反证法易证

- 另外以下算法默认图中无负环，若有负环会出现最短路为-∞的情况

01 Floyd算法

Floyd算法

- Floyd算法用于求解没有负环的图上的多源最短路问题。
- 令f (i, j, k)表示i到j的最短路长度，其中最短路径上点的编号不大于k。
- 考虑将f (i, j, k)转移到f (i, j, k + 1)。
- 情况一：路径上点的编号不大于k。
- 情况二：先从i到k + 1，再从k + 1到j。
- $f(i, j, k + 1) = \min(f(i, j, k), f(i, k + 1, k) + f(k + 1, j, k))$。
- 边界值为$f(i, j, 0) = cost(i, j)$。
- 实现时只需要维护一个二维数组f [i][j]，分层进行转移。
- 时间复杂度为$O(n^3)$。

```cpp
#define Rep(x,a,b) for (int x=a;x<=b;x++)
void Floyd(){
    Rep(i,1,n) Rep(j,1,n) Rep(k,1,n)
        a[j][k]=min(a[j][k],a[j][i]+a[i][k]);
}
```

Floyd算法求最小环

- 有向图： 直接跑floyd算法，di,i就是答案。
- 无向图: 无向图的差别就是不能有二元环。
- Floyd算法保证了最外层枚举到k时所有顶点间以0到k − 1为中间点的最短路径。
- 在用中间点k更新所有点对(i, j)的最短路前，点k不存在于已存在的最短路f [i][j]上。
- 设环上编号最大的顶点为k，且在环上与k直接相连的两个顶点为i和j，则最大编号为k的最小环的长度为min{cost(i, k) + cost(k, j) + f (i, j)}。
- 容易发现，这样枚举到的环至少有三条边。
- 枚举完所有的k后，即可找到整个图的最小环。
- 时间复杂度与Floyd算法相同，为$O(n^3)$。

某CF的Gym题

- 对于图上每一个点，求出包含这个点的最小简单环长度，无向图。
- 范围：$N \le 200$。

- 这个题可以看成是进行N次询问，每次询问删除该点后的图信息。
- 对时间分治，即对第i个点，将这个点的信息插入 到[1, i − 1]和[i + 1, N]里。
- 因为Floyd最外层的枚举点的顺序可以是任意排列，可以实现，加点后维护两两最短路。
- 所以把每一层的点加 入即可。
- 效率：O(N^3 log N)。

02 Dijkstra算法

Dijkstra算法

- 如果存在一条从i到j的最短路径(V_i.....V_k,V_j)，其中V_k是V_j前面的一顶点，那么(V_i...V_k)也必定是从i到k的最短路径。
- 为了求出最短路径，Dijkstra就提出了以最短路径长度递增，逐次生成最短路径的算法。
- 譬如对于源顶点V_0，首先选择其直接相邻的顶点中长度最短的顶点V_i，那么当前已知可得从V_0到达V_j顶点的最短距离dist[j]=min{dist[j],dist[i]+len[i][j]}。

- 假设存在图G=<V,E>，源顶点为V_0，U={V_0}，dist[i]记录V_0到i的最短距离。
- 具体实现步骤：
- 1.从V-U中选择$dist_i$值最小的顶点i，将i加入到U中。
- 2.更新与i直接相邻顶点的dist值。（$dist_j$=min{$dist_j$, $dist_i$+L(i, j) }。其中L(i,j)为图中的边<i, j>的长度。）
- 3.重复上述步骤，直到U=V。
- 不加任何优化的Dijkstra时间复杂度为O（|V|2）
- 代码时间主要浪费在遍历所有点找到一个距离最小的点。
- 因此联系到运用堆优化Dijkstra算法。在堆中维护未被访问过的点的d[v]值。
- 每次选取一个顶点u，枚举u的边表更新与其相邻的点v的d[v]值。
- 使用二叉堆和邻接表优化Dijkstra算法，时间复杂度为O((n + m) log n)。
- 斐波那契堆优化：O(N log N + M)

```
void Dijkstra(int s){
    memset(vis,0,sizeof vis);
    Rep(i,1,n) dist[i]=len[s][i];
    vis[s]=1;
    Rep(i,1,n-1){
        int v,Min=1<<30;
        Rep(j,1,n) if (!vis[j]&&dist[j]<Min)   //选择V-U中dist最小的顶点
            Min=dist[j],v=j;
        vis[v]=1;                              //将这个点加入U中
        Rep(j,1,n) if (!vis[j])                //更新相邻顶点的dist
            dist[j]=min(dist[j],dist[v]+len[v][j]);
    }
}
```

一些小tips

- 边权范围较小：
  - 对每一种权值开个队列，代替堆优化。
  - 需要支持从队列中删除任意位置的数。
  - 当边权均为1时，直接用BFS即可。
- 多源单汇：将图反向。
- 拓扑图：按照拓扑序遍历即可。
- 有多个点初始距离为0：建立超级源。
- 优化背包。

某USACO题

- 有N个物品，每个物品体积为Ai，且有无穷个。
- 求最大的不能拼出 的体积。
- 范围：N, Ai ≤ 3000。

- 这道题的本质是背包。
- 对于无穷个物品，单个物品体积很小，背包总体积非常大的判定性问题。
- 可以考虑把背包问题转化为体积模域下的最短路问题。
- 即我们随便找出一个物品体积，设为a。
- 令F[i]表示真实体积模a等于i的，最小的能得到的体积。
- 这样，如果F[i]得到了，则F[i] + a也可以得到。
- 我们可以用dij算法求解F[i]。效率：O(N × Ai)。

最短路树

- 考虑把做最短路时，记一下每个点最后的dist是从哪个点转移过来的，那么就形成了一棵树结构，最短路就是每个点到根的距离
- 当然每个点的前驱可能会有多个，换种定义：
- 考虑所有满足dist(s, v) =dist(s, u) + cost(u, v)的边拿出来形成的子图，该子图应该是有向无环图，否则就会出现负环或者是0环
- 这时的最短路“树”就是个DAG

JOI 2018 月票购买

- JOI 所住的城市有N个车站，分别编号为1…N。有M条铁路，编号为1…M。第i条铁路双向连接车站Ai与车站Bi，乘车费用为Ci。
- JOI 住在车站S附近，而JOI所在的IOI高中在车站T附近。他打算买一张月票往返这两个车站。当他买这张月票时，他需要选择一条在车站S与车站T之间的乘车费用最小的路径。有了这张月票，JOI可以无需额外费用，双向通过任意所选路径包含的铁路。
- JOI 经常去在车站U与车站V附近的书店，因此他希望能买一张月票使得从车站U到车站V的花费最小。
- 当他要从车站U去往车站V时，他会选择一条从车站U到车站V的路径。对于路径上的每段铁路，如果这段铁路在月票指定的路径范围内，则费用为0，否则费用为Ci。每段铁路的费用和为JOI从车站U到车站V的总费用。
- 他想要知道，如果他买月票时选择了一条合适的路线，从车站U到车站V的最小费用是多少。

- 把S到T的最短路的DAG建出来，满足条件的U到V的最短路一定与DAG的一条链相交，因为DAG上的路有最优子结构
- 把DAG上的有向边权值赋为0
- 然后考虑拆点，建点i,i+n,i+2n分别表示还没走过相交的链，正在相交的链上，走完了相交的链，跑最短路
- 因为月票可以双向通过，把所有有向边反向再跑一次

03 杂题们

[AMPPZ2014]The Captain

- 给定平面上的n个点，定义(x1,y1)到(x2,y2)的费用为min(|x1-x2|,|y1-y2|)，求从1号点走到n号点的最小费用。

- 发现因为是最短路，所以这个min没有意义，大可以建一条|x1-x2|,一条|y1-y2|两条边，而跑最短路时一定会默认跑较小的一条了。

不知道哪里来的题

- 有一张N个点的完全图，边权只有可能是A 或者B。
- 读入的M条边边权是A，剩下的边边权是B。
- 求1到N的最短路长度。
- N<=10^5,M<=5*10^5

- 按情况分类：
- ①A<=B 这个比较好办，因为A边很少。
- 如果1~N是A边，那直接就是答案；
- 否则我们直接 从1开始只走A边，看看到N要多少，和B比较一 下获得答案。

- ②A>B
- B边数量较多怎么办?
- 同样是类似BFS的思路，去求到一个点最少经过几次B边。
- 具体地，可以维护一个目前还未到过的点的集合S。
- 每次对于一个已经访问到的点i，我先遍历i的A边出边，然后暂时把这些点标记不可用；
- 然后在集合S里不断挑那些未被标记的 点，更新它的最短路并将其从集合删除。
- 可以很方便地用线段树做到这一点。

CF 261 C

- 给定N个点,M条边的有向图，带边权。
- 求最长的路径使得经过的边权严格上升。
- N,M<=100000

- 发现因为有边权递增的限制不能直接跑最短路。
- 一个很显然的想法是，对于每一条边 进行DP。
- 设f[i]表示到第i条边时最长路径。
- 如何转移呢？我们找到这条边末端点k开 始的所有边？
- 实际上还是要把DP数组记在点上。
- 我们先假设没有边权重复的边。 设f[i]表示目前结束点是i的最长路径长度，初始化都是0。
- 按边权排序后，我们依次 扫描每一条边，设其为(x,y），则用f[x]+1来 更新f[y]。
- 有边权重复的边略麻烦一点。它们互相不 能更新，所以每次我们先把这些边全都一起拎 出来，用之前比它们严格小的边更新它们。

51nod #26 E

- 给定N个城市与M条道路。每一个城市都有一 =个喜好度（两个喜好度可以相同）。
- 现在要求一条从1到N的 “最坏”的路径。
- 定义两条路径哪条更坏：对于每一条路径 统计喜好度为i的城市经过了几个。找到最小的一个喜好度k使得这两条路径经过的该喜好度的城市数量不同，数量少的路径更坏。
- N<=100000 M<=500000

- 看上去是一道显然的题。
- 考虑暴力的做法，就是在每一个点维护一 个数组，表示从1到这一个点的最坏路径下， 每一种喜好度经过的点的个数。
- 为了不必要的 麻烦，我们采用dijkastra进行最短路的计算。
- 那么如何改进呢？实际上我们只要把这些数组可持久化就行了。
- 借鉴可持久化线段树的道理，每次修改一个节点并增加logN的空间。
- 时间效率是O(MlogN^2)

CF 718 E Matvey’s Birthday

- 有一个长度为N的字符串s，字符集为8。
- 由此我们建立一张N个点的图，第i个点表示下标i。
- 对于这张图，我们连这样两种边：
  - 当$|i-j| \le 1$，在i, j之间连边。
  - 当$s_i = s_j$时，在i, j之间连边。
- 求这张图的直径。
- 范围：$N \le 100000$。

- 首先图的直径似乎是只能求两两点对的最短路的，所以我们必须发掘这道题的性质。
- 一个比较显然的结论，就是图的直径必然小于等于$2 \times M - 1$，其中M表示字符集大小。
  - 考虑一条从i走到j的路径，每种字符最多出现2次；
  - 因为如果某一个字符出现3次以上，那最优方案显然是不走中间那些重复出现的字符，直接从第一个花费1的代价跳到最后一个即可。
  - 最多出现$2 \times M$个字符，所以路径长度最多为$2 \times M - 1$。

- 还可以看出，如果i走到j不走第二类边，则长度必然是|i − j|；
- 否则它们之间跳跃了c字符，则会有：
- Dis（i,j） = min （Dis(i,c) + Dis(j,c) + 1)
- 其中Dis的含义很多：
  - 如果下标是两个点i, j，则代表点i与点j的最短距离；
  - 如果是点i和字符c，则代表点i到任意j(sj = c)的最短距离；
  - 如果是字符c1和c2，则代表所有i(si = c1)和j(sj = c2)点对之间的最短距离。

- 要发现一个性质：
- Dis(si ,c) ≤ Dis(i,c) ≤ Dis(si ,c + 1)
- 这也比较容易理解。
- 于是就会发现，如果把点i到所有字符c的距离，看成一个M维的向量，则本质只会有M × 2^M种不同的向量，其中M表示s[i]是什么字符，2^M即Dis(si ,c) 与 Dis(i,c)的关系 。

- 于是就想到一个做法，我们枚举右边的点i，对于j(i − j ≤ 15)，我 们暴力枚举并逐一判断。 复杂度O(N*M)
- 对于j(j < i − 15)，则它们之间的最短路必须得跳跃字符了。
- min (Dis(i,c) + Dis(j,c)+ 1)
- 这是啥，如果i的向量和j的向量已经知道了，那么就只要枚举c，从向量里取个min即可。这也意味着算这个min不需要知道j是什么，只需要j的向量
- 于是不再枚举j，而是枚举j的向量计算答案，复杂度O(N*M*2^M)

- 我们需要预处理Dis(i,c)，这个可以枚举所有c，
- 然后用O(N)的BFS处理答案。
  - 边权为1
  - 多个点初始距离为0。
- Dis(c1,c2）直接用Dis(i,c)更新即可。 复杂度O(N*M)
- 并且我们还需要预处理两两不同向量之间的最短距离。复杂度O(M^3*4^M)

非常感谢您的观看

THANK YOU FOR YOUR WATCHING