k 短路

问题描述

给定一个有 n 个结点， m 条边的有向图，求从 s 到 t 的所有不同路径中的第 k 短路径的长度。

A*算法

A*算法定义了一个对当前状态 x 的估价函数 $f(x)=g(x)+h(x)$ ，其中 $g(x)$ 为从初始状态到达当前状态的实际代价， $h(x)$ 为从当前状态到达目标状态的最佳路径的估计代价。每次取出 $f(x)$ 最优的状态 x ，扩展其所有子状态，可以用 **优先队列** 来维护这个值。

在求解 k 短路问题时，令 $h(x)$ 为从当前结点到达终点 t 的最短路径长度。可以通过在反向图上对结点 t 跑单源最短路预处理出对每个结点的这个值。

由于设计的距离函数和估价函数，对于每个状态需要记录两个值，为当前到达的结点 x 和已经走过的距离 $g(x)$ ，将这种状态记为 $(x,g(x))$ 。

开始我们将初始状态 $(s,0)$ 加入优先队列。每次我们取出估价函数 $f(x)=g(x)+h(x)$ 最小的一个状态，枚举该状态到达的结点 x 的所有出边，将对应的子状态加入优先队列。当我们访问到一个结点第 k 次时，对应的状态的 $g(x)$ 就是从 x 到该结点的第 k 短路。

优化：由于只需要求出从初始结点到目标结点的第 k 短路，所以已经取出的状态到达一个结点的次数大于 k 次时，可以不扩展其子状态。因为之前 k 次已经形成了 k 条合法路径，当前状态不会影响到最后的答案。

当图的形态是一个 n 元环的时候，该算法最坏是 $O(nk\log n)$ 的。但是这种算法可以在相同的复杂度内求出从起始点 s 到每个结点的前 k 短路。

参考实现

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int maxn = 5010;
const int maxm = 400010;
const int inf = 2e9;
int n, m, s, t, k, u, v, ww, H[maxn], cnt[maxn];
int cur, h[maxn], nxt[maxm], p[maxm], w[maxm];
int cur1, h1[maxn], nxt1[maxm], p1[maxm], w1[maxm];
bool tf[maxn];
void add_edge(int x, int y, double z) {
  cur++;
  nxt[cur] = h[x];
  h[x] = cur;
  p[cur] = y;
  w[cur] = z;
}
void add_edge1(int x, int y, double z) {
  cur1++;
  nxt1[cur1] = h1[x];
  h1[x] = cur1;
  p1[cur1] = y;
  w1[cur1] = z;
}
struct node {
  int x, v;
  bool operator<(node a) const { return v + H[x] > a.v +
H[a.x]; }
};
priority_queue<node> q;
struct node2 {
  int x, v;
  bool operator<(node2 a) const { return v > a.v; }
} x;
priority_queue<node2> Q;
int main() {
  scanf("%d%d%d%d%d", &n, &m, &s, &t, &k);
  while (m--) {
    scanf("%d%d%d", &u, &v, &ww);
    add_edge(u, v, ww);
    add_edge1(v, u, ww);
  }
  for (int i = 1; i <= n; i++) H[i] = inf;
  Q.push({t, 0});
```

```
  while (!Q.empty()) {
    x = Q.top();
    Q.pop();
    if (tf[x.x]) continue;
    tf[x.x] = true;
    H[x.x] = x.v;
    for (int j = h1[x.x]; j; j = nxt1[j]) Q.push({p1[j], x.v + w1[j]});
  }
  q.push({s, 0});
  while (!q.empty()) {
    node x = q.top();
    q.pop();
    cnt[x.x]++;
    if (x.x == t && cnt[x.x] == k) {
      printf("%d\n", x.v);
      return 0;
    }
    if (cnt[x.x] > k) continue;
    for (int j = h[x.x]; j; j = nxt[j]) q.push({p[j], x.v + w[j]});
  }
  printf("-1\n");
  return 0;
}
```

可持久化可并堆优化 k 短路算法

最短路树与任意路径的关系与性质

在反向图上从 t 开始跑最短路，设在原图上结点 x 到 t 的最短路长度为 $dist_x$ ，建出 **任意** 一棵以 t 为根的最短路树 T 。

所谓最短路径树，就是满足从树上的每个结点 x 到根节点 t 的简单路径都是 x 到 t 的 **其中** 一条最短路径。

设一条从 s 到 t 的路径经过的边集为 P ，去掉 P 中与 T 的交集得到 P' 。

P' 有如下性质：

1. 对于一条不在 T 上的边 e ，其为从 u 到 v 的一条边，边权为 w ，定义其代价 $\Delta e = dist_v + w - dist_u$ ，即为选择该边后路径长度的增加量。则路径 P 的长度 $L_P = dist_s + \sum_{e\in P'} \Delta e$ 。

2. 将 P 和 P' 中的所有边按照从 s 到 t 所经过的顺序依次排列，则对于 P' 中相邻的两条边 e_1, e_2 ，有 u_{e_2} 与 v_{e_1} 相等或为其在 T 上的祖先。因为在 P 中 e_1, e_2 直接相连或中间都为树边。
3. 对于一个确定存在的 P' ，有且仅有一个 S ，使得 $S' = P'$ 。因为由于性质 2 ， P' 中相邻的两条边的起点和终点之间在 T 上只有一条路径。

问题转化

性质 1 告诉我们知道集合 P' 后，如何求出 L_P 的值。

性质 2 告诉我们所有 P' 一定满足的条件，所有满足这个条件的边集 P' 都是合法的，也就告诉我们生成 P' 的方法。

性质 3 告诉我们对于每个合法的 P' 有且仅有一个边集 P 与之对应。

那么问题转化为：求 L_P 的值第 k 小的满足性质 2 的集合 P' 。

算法描述

由于性质 2 ，我们可以记录按照从 s 到 t 的顺序排列的最后一条边和 L_P 的值，来表示一个边集 P' 。

我们用一个小根堆来维护这样的边集 P' 。

初始我们将起点为 1 或 1 在 T 上的祖先的所有的边中 Δe 最小的一条边加入小根堆。

每次取出堆顶的一个边集 S ，有两种方法可以生成可能的新边集：

1. 替换 S 中的最后一条边为满足相同条件的 Δe 更大的边。
2. 在最后一条边后接上一条边，设 x 为 S 中最后一条边的终点，由性质 2 可得这条边需要满足其起点为 x 或 x 在 T 上的祖先。

将生成的新边集也加入小根堆。重复以上操作 $k-1$ 次后求出的就是从 s 到 t 的第 k 短路。

对于每个结点 x ，我们将以其为起点的边的 Δe 建成一个小根堆。为了方便查找一个结点 x 与 x 在 T 上的祖先在小根堆上的信息，我们将这些信息合并在一个编号为 x 的小根堆上。回顾以上生成新边集的方法，我们发现只要我们把紧接着

可能的下一个边集加入小根堆，并保证这种生成方法可以覆盖所有可能的边集即可。记录最后选择的一条边在堆上对应的结点 t ，有更优的方法生成新的边集：

1. 替换 S 中的最后一条边为 t 在堆上的左右儿子对应的边。
2. 在最后一条边后接上一条新的边，设 x 为 S 中最后一条边的终点，则接上编号为 x 的小根堆的堆顶结点对应的边。

用这种方法，每次生成新的边集只会扩展出最多三个结点，小根堆中的结点总数是 $O(n+k)$ 。

所以此算法的瓶颈在合并一个结点与其在 T 上的祖先的信息，如果使用朴素的二叉堆，时间复杂度为 $O(nm\log m)$ ，空间复杂度为 $O(nm)$ ；如果使用可并堆，每次仍然需要复制堆中的全部结点，时间复杂度同样无法承受。

可持久化可并堆优化

在阅读本内容前，请先了解 可持久化可并堆 [../../ds/persistent-heap/] 的相关知识。

使用可持久化可并堆优化合并一个结点与其在 T 上的祖先的信息，每次将一个结点与其在 T 上的父亲合并，时间复杂度为 $O(n\log m)$ ，空间复杂度为 $O((n+k)\log m)$ 。这样在求出一个结点对应的堆时，无需复制结点且之后其父亲结点对应的堆仍然可以正常访问。

参考实现

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int maxn = 200010;
int n, m, s, t, k, x, y, ww, cnt, fa[maxn];
struct Edge {
  int cur, h[maxn], nxt[maxn], p[maxn], w[maxn];
  void add_edge(int x, int y, int z) {
    cur++;
    nxt[cur] = h[x];
    h[x] = cur;
    p[cur] = y;
    w[cur] = z;
```

```
  }
} e1, e2;
int dist[maxn];
bool tf[maxn], vis[maxn], ontree[maxn];
struct node {
  int x, v;
  node* operator=(node a) {
    x = a.x;
    v = a.v;
    return this;
  }
  bool operator<(node a) const { return v > a.v; }
} a;
priority_queue<node> Q;
void dfs(int x) {
  vis[x] = true;
  for (int j = e2.h[x]; j; j = e2.nxt[j])
    if (!vis[e2.p[j]])
      if (dist[e2.p[j]] == dist[x] + e2.w[j])
        fa[e2.p[j]] = x, ontree[j] = true, dfs(e2.p[j]);
}
struct LeftistTree {
  int cnt, rt[maxn], lc[maxn * 20], rc[maxn * 20], dist[maxn
* 20];
  node v[maxn * 20];
  LeftistTree() { dist[0] = -1; }
  int newnode(node w) {
    cnt++;
    v[cnt] = w;
    return cnt;
  }
  int merge(int x, int y) {
    if (!x || !y) return x + y;
    if (v[x] < v[y]) swap(x, y);
    int p = ++cnt;
    lc[p] = lc[x];
    v[p] = v[x];
    rc[p] = merge(rc[x], y);
    if (dist[lc[p]] < dist[rc[p]]) swap(lc[p], rc[p]);
    dist[p] = dist[rc[p]] + 1;
    return p;
  }
} st;
void dfs2(int x) {
  vis[x] = true;
  if (fa[x]) st.rt[x] = st.merge(st.rt[x], st.rt[fa[x]]);
  for (int j = e2.h[x]; j; j = e2.nxt[j])
    if (fa[e2.p[j]] == x && !vis[e2.p[j]]) dfs2(e2.p[j]);
}
```

```
int main() {
  scanf("%d%d%d%d%d", &n, &m, &s, &t, &k);
  for (int i = 1; i <= m; i++)
    scanf("%d%d%d", &x, &y, &ww), e1.add_edge(x, y, ww),
e2.add_edge(y, x, ww);
  Q.push({t, 0});
  while (!Q.empty()) {
    a = Q.top();
    Q.pop();
    if (tf[a.x]) continue;
    tf[a.x] = true;
    dist[a.x] = a.v;
    for (int j = e2.h[a.x]; j; j = e2.nxt[j])
Q.push({e2.p[j], a.v + e2.w[j]});
  }
  if (k == 1) {
    if (tf[s])
      printf("%d\n", dist[s]);
    else
      printf("-1\n");
    return 0;
  }
  dfs(t);
  for (int i = 1; i <= n; i++)
    if (tf[i])
      for (int j = e1.h[i]; j; j = e1.nxt[j])
        if (!ontree[j])
          if (tf[e1.p[j]])
            st.rt[i] = st.merge(
                st.rt[i],
                st.newnode({e1.p[j], dist[e1.p[j]] + e1.w[j]
- dist[i]}));
  for (int i = 1; i <= n; i++) vis[i] = false;
  dfs2(t);
  if (st.rt[s]) Q.push({st.rt[s], dist[s] +
st.v[st.rt[s]].v});
  while (!Q.empty()) {
    a = Q.top();
    Q.pop();
    cnt++;
    if (cnt == k - 1) {
      printf("%d\n", a.v);
      return 0;
    }
    if (st.lc[a.x])
      Q.push({st.lc[a.x], a.v - st.v[a.x].v +
st.v[st.lc[a.x]].v});
    if (st.rc[a.x])
      Q.push({st.rc[a.x], a.v - st.v[a.x].v +
```

```
st.v[st.rc[a.x]].v});
      x = st.rt[st.v[a.x].x];
      if (x) Q.push({x, a.v + st.v[x].v});
    }
    printf("-1\n");
    return 0;
  }
```

习题

「SDOI2010」魔法猪学院 [https://www.luogu.com.cn/problem/P2483]

本页面最近更新：2020/7/5 18:53:09，更新历史 [https://github.com/OI-wiki/OI-wiki/commits/master/docs/graph/kth-path.md]
发现错误？想一起完善？在 GitHub 上编辑此页！ [https://oi-wiki.org/edit-landing/?ref=/graph/kth-path.md]
本页面贡献者：hsfzLZH1 [https://github.com/hsfzLZH1], ouuan [https://github.com/ouuan], Henry-ZHR [https://github.com/Henry-ZHR], Ir1d [https://github.com/Ir1d], Marcythm [https://github.com/Marcythm]

评论

1 [https://github.com/OI-wiki/gitment/issues/213] 条评论 未登录用户

[]

我们鼓励在讨论区讨论有意义的内容及关于文章的勘误，无意义的讨论将会被管理员删除

预览

使用 GitHub 登录