$HONG-LOU\ 2022-8-23$

这东西真的是很帅很神奇啊，久闻大名

$$orzorz$$

前言

- $spfa$ 算法由于它上限 $\Theta(NM)=\Theta(VE)$ 的时间复杂度，被卡掉的几率很大。在算法竞赛中，我们需要一个更稳定的算法：$Dijkstra$。

什么是$Dijkstra$?

- $Dijkstra$是一种单源最短路径算法,时间复杂度上限为$\Theta(n^2)$(朴素),在实际应用中较为稳定;加上堆优化之后更是具有$\Theta((n+m)\log_2 n)$的时间复杂度,在稠密图中有不俗的表现.

$Dijkstra$ 的原理/流程?

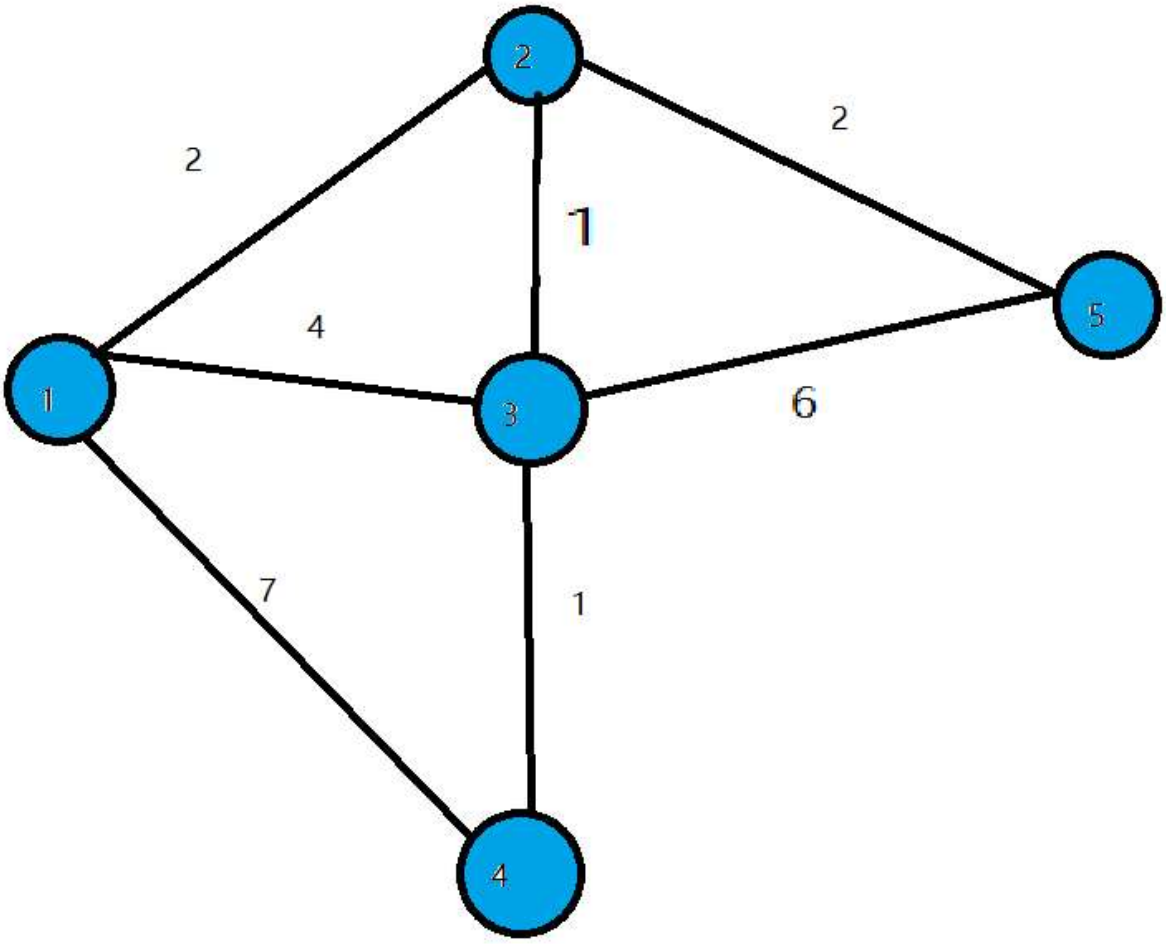
- $Dijkstra$本质上的思想是贪心,它只适用于不含负权边的图.
- 我们把点分成两类,一类是已经确定最短路径的点,称为"白点",另一类是未确定最短路径的点,称为"蓝点"
- $Dijkstra$的流程如下:
- 1.初始化$dis[start]=0$, 其余节点的dis 值为无穷大。
- 2.找一个dis值最小的蓝点x,把节点x变成白点.
- 3. 遍历x的所有出边(x,y,z),若$dis[y]>dis[x]+z$,则令$dis[y]=dis[x]+z$。
- 4.重复$2,3$两步,直到所有点都成为白点。
- 时间复杂度$\Theta(n^2)$

$Dijkstra$为什么时正确的

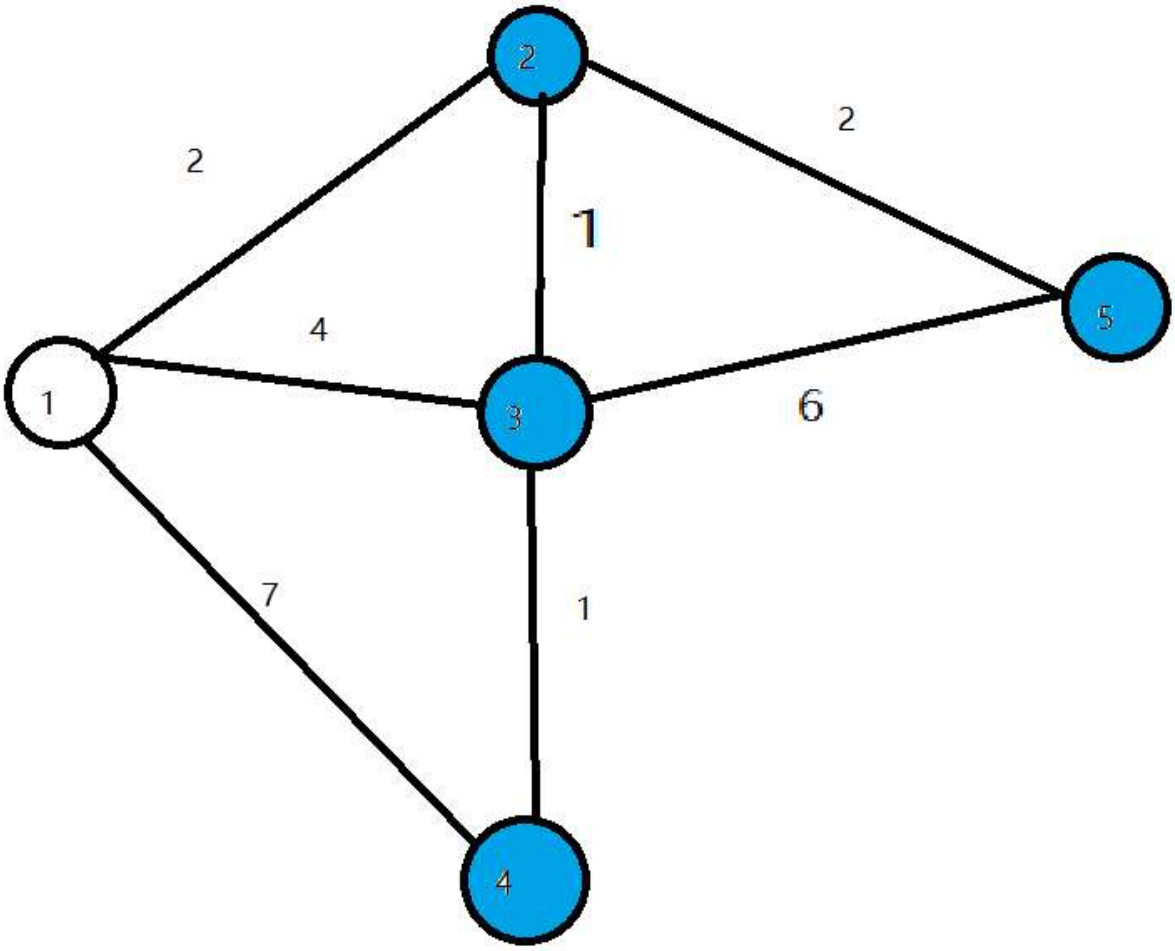
- 当所有边长都是非负数的时候，全局最小值不可能再被其他节点更新. 所以在第2步中找出的蓝点x必然满足:$dis[x]$已经是起点到x的最短路径. 我们不断选择全局最小值进行标记和拓展，最终可以得到起点到每个节点的最短路径的长度

图解

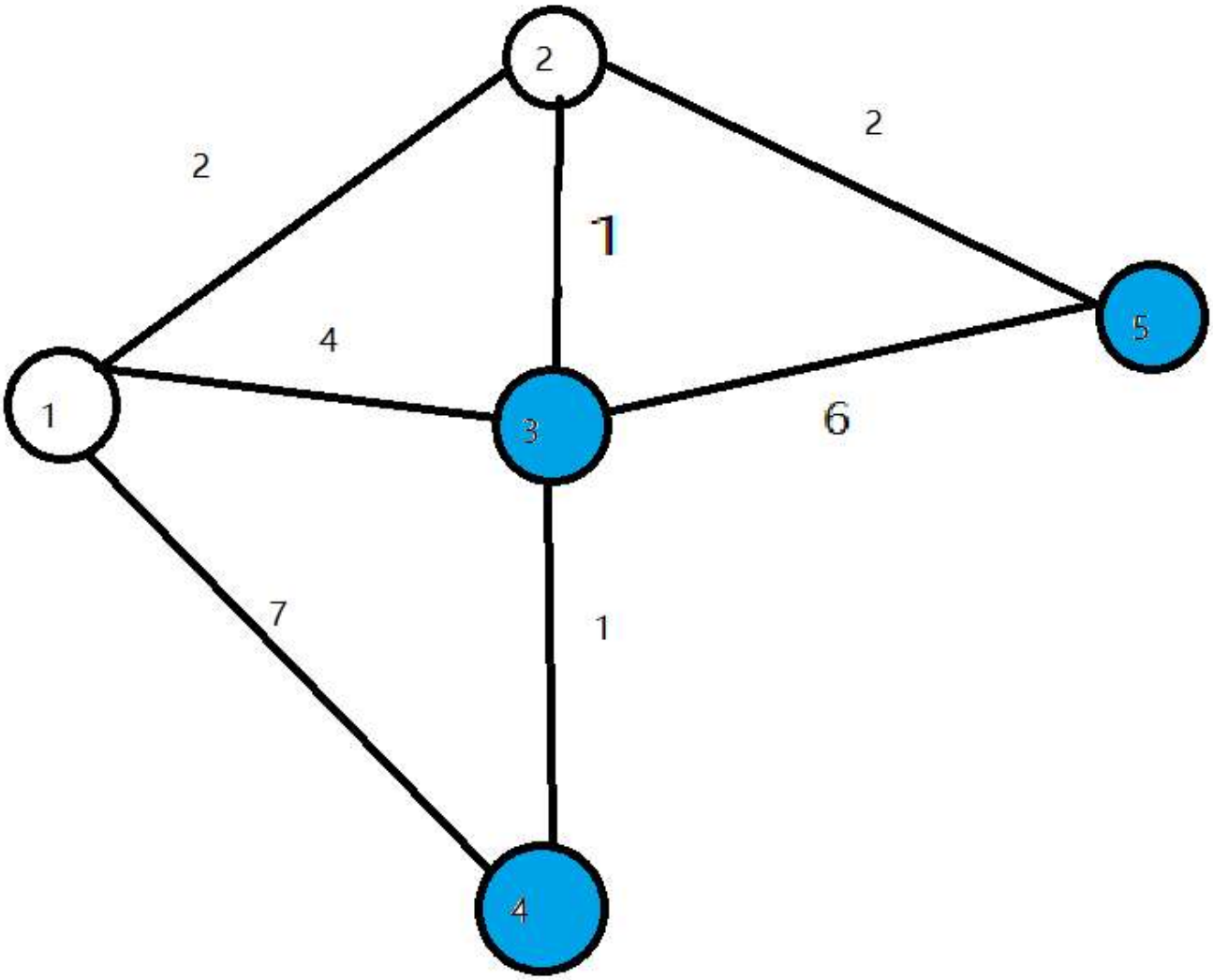
- 令$(start = 1)$
- 开始时我们把$dis[start]$初始化为0，其余点初始化为inf



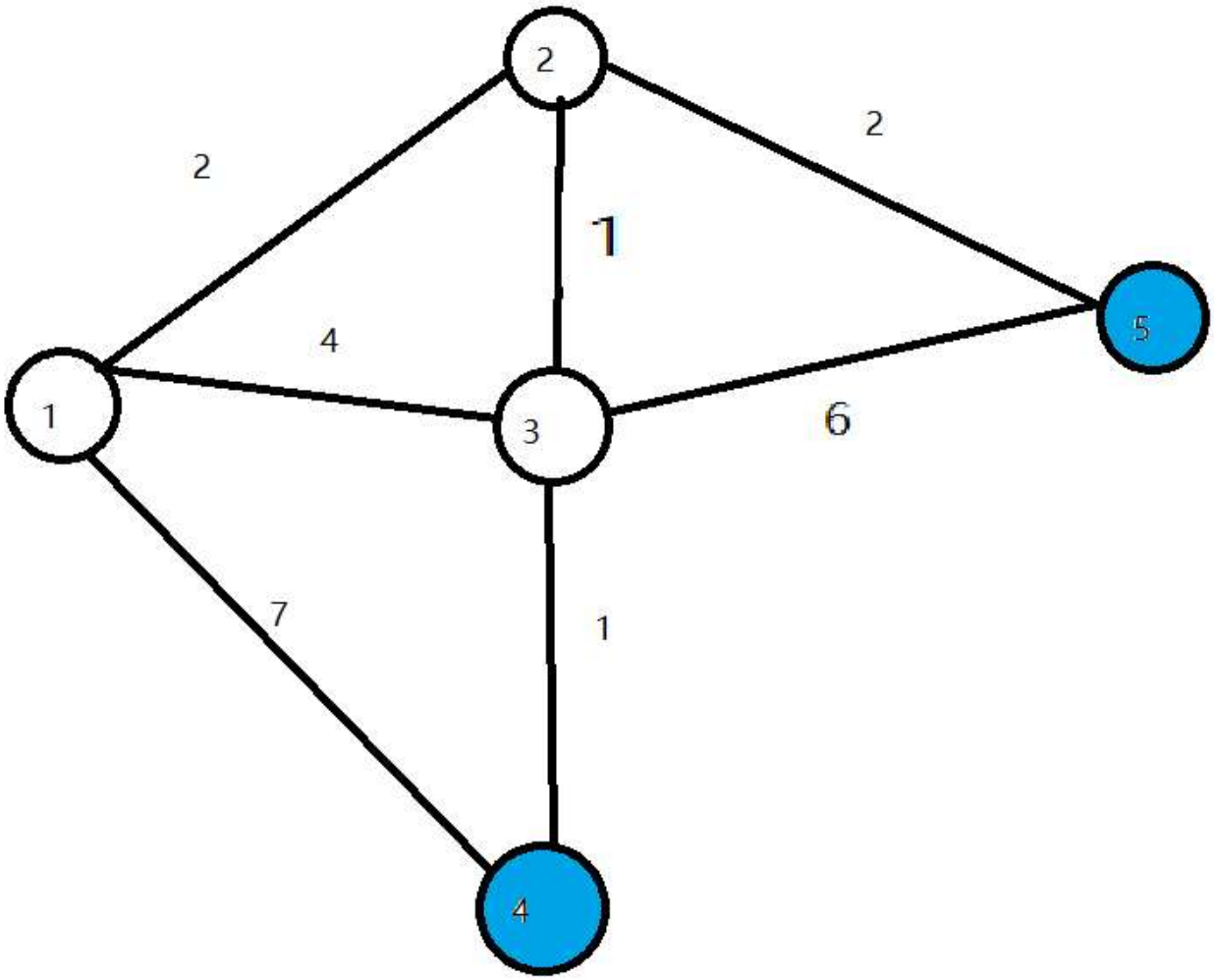
- 第一轮循环找到dis值最小的点1，将1变成白点，对所有与1相连的蓝点的dis值进行修改，使得$dis[2] = 2, dis[3] = 4, dis[4] = 7$



- 第二轮循环找到dis值最小的点2,将点2变成白点,将所有与2相连的的蓝点的dis值进行修改,使得$dis[3]=3,dis[5]=4$



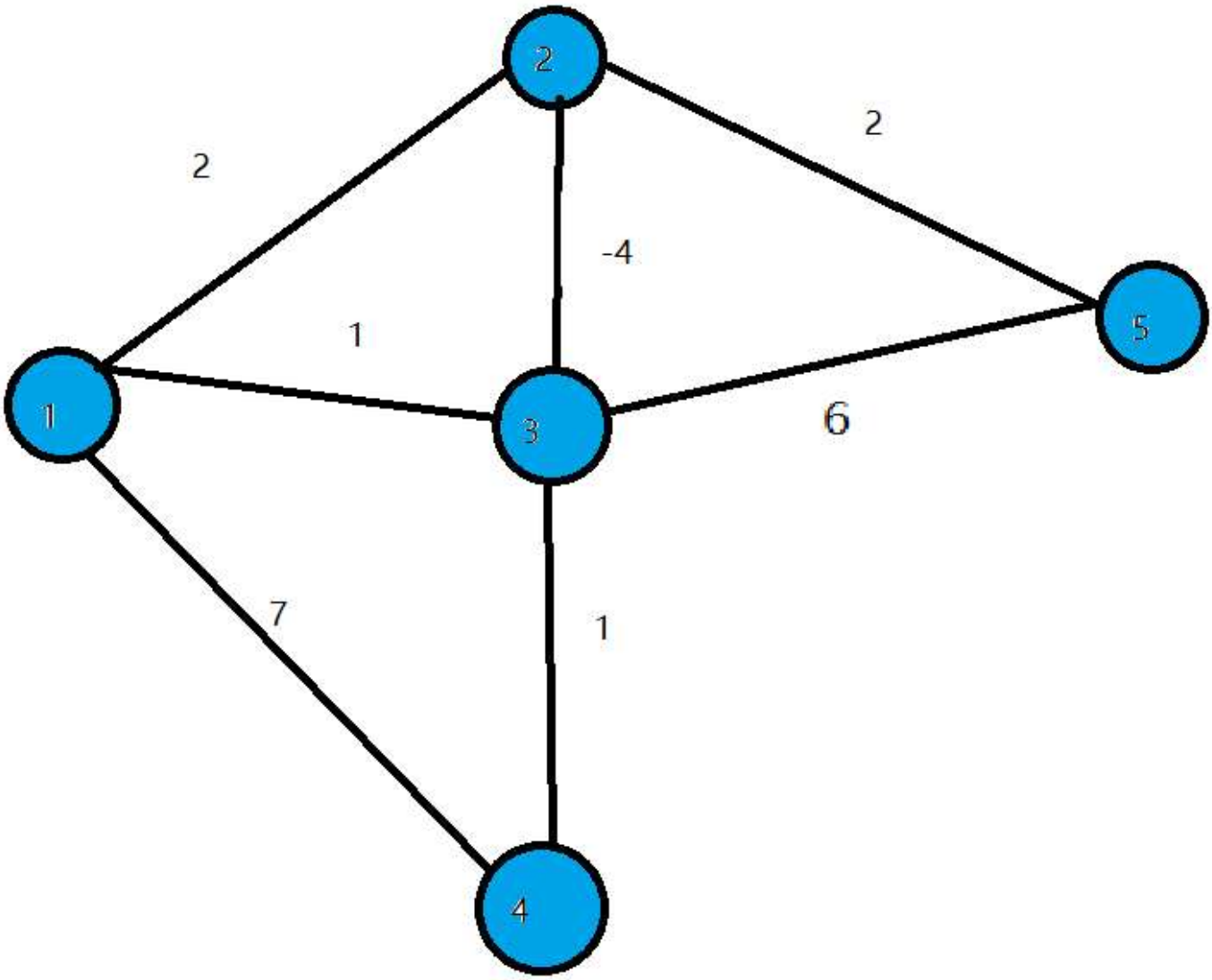
• 第三轮循坏找到dis值最小的点3, 将3变成白点, 对所有与2相连的蓝点的dis值进行修改, 使得$dis[4] = 4$



- 接下来两轮循环分别将$4,5$设为白点, 算法结束, 求出所有点的最短路径
- 时间复杂度$\Theta(n^2)$

为什么$Dijkstra$不能处理有负边权的情况?

- 我们来看下面这张图



- 2到3的边权为-4, 显然从1到3的最短路径为$-2\ 1 \rightarrow 2 \rightarrow 3$. 但在循环开始时程序会找到当前$dis$值最小的点3, 并标记它为白点.
- 这时的$dis[3] = 1$, 然而1并不是起点到3的最短路径. 因为3已经被标为白点, 所以$dis[3]$不会再被修改了. 我们在边权存在负数的情况下得到了错误的答案.

$Dijkstra$的堆优化

直接上代码

```cpp
/**
 *    author:  HONG-LOU
 *    created: 2022-08-22 16:26:50
**/
#include <bits/stdc++.h>

using namespace std;

constexpr long long inf = 1e18;

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  int n, m, s;
  std::cin >> n >> m >> s;

  std::vector<long long> dis(n, (1ll << 31) - 1);
  dis[s - 1] = 0;

  vector<vector<array<long long, 2>>> g(n);

  for(int i = 0; i < m; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    u--, v--;
    g[u].push_back({v, w});
  }

  function<void()> Dijkstra = [&] () {
  priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<>> h;
      for(int i = 0; i < n; i++) {
        h.emplace(dis[i], i);
      }
      while(!h.empty()) {
        auto [d, x] = h.top();
        h.pop();
        if(d > dis[x]) {
          continue;
        }
        for(auto [y, w] : g[x]) {
          if(d + w < dis[y]) {
            dis[y] = d + w;
            h.emplace(dis[y],y);
          }
        }
      }
  };
```

```
  Dijkstra();

  for(int i = 0; i < n; i++) {
    cout << dis[i] << " \n"[i == n - 1];
  }
  return 0;
}
```

太帅了！！！