單源最短路徑

單源最短路徑

- 從一個點，出發到所有點的最短路徑
- 最短路徑樹
- Dijkstra, Bellman-ford, SPFA, …

鬆弛（Relax）

- 對於點 u, v
  - 起點到 u 距離為 10
  - 起點到 v 距離為 20
  - u 到 v 距離為 5
- 對 (u, v) 鬆弛，v 距離更新為 15
- <u>當沒有邊可以鬆弛，就是最短路了</u>

Bellman-ford

理念

- 起點最短路徑設成 0，其他設成無限大
- 不斷 $O(E)$ 枚舉邊來鬆弛，直到沒有可鬆弛的邊
- 複雜度：
  - 任一一條最短路從起點開始，經過每一次鬆弛，至少會擴張一條邊
  - 最短路長度最多為 $V - 1$ (無負環時)
  - 複雜度 $O(E \quad V)$

Bellman-ford（cont.）

負環偵測

- 越繞越小 -> 無止盡的鬆弛
- 當 Bellman-ford 執行超過 $V-1$ 次，即存在負環

Bellman-ford（cont.）

程式碼

```cpp
typedef pair<int,int> pii;
int dis[N], n;
vector<int, pii>; // {weight, {from, to}}

// 如果有負環，return 1，否則 return 0 並求出單源最短路
void bellmanford(int root)
{
    for (int i=1; i<=n; i++)
        dis[i] = 2e9;
    dis[root] = 0;

    for (int i=1; i<=n; i++)
    {
        for (auto e: E)
        {
            int w = e.first;
            int u = e.second.first;
            int v = e.second.second;

            if (dis[u] + w < dis[v])
            {
                if (i == n)
                {
                    return 1;
                }
                dis[v] = dis[u] + w;
            }
        }
    }

    return 1;
}
```

Dijkstra

理念

- 當邊權非負，有以下性質：
- 用完成一部分的最短路徑樹去鬆弛尚未在樹上的點，離根最近的點，那條邊也一定在最短路徑樹上

Dijkstra（cont.）

理念

- 每把一個點加進最短路徑樹
  - Relax 周遭的點
  - 結果丟進 Priority Queue
- 從 Priority Queue 拿距離最短的
  - 若是第一次被拿出來，加進最短路徑樹
  - 否則跳過

Dijkstra（cont.）

理念

- 觀念釐清：為何從 Priority Queue 拿出來時，
  最短路徑樹上的點都已經 Relax 過它了？

Dijkstra（cont.）

複雜度

- 最慘每個邊都要鬆弛一次
- $O(\ E\ \ log\ \ E\ \ )$

Dijkstra（cont.）

實作提示

- 初始化 dis[ ]、pq
- 當 pq 還有東西：
  - 從 pq 拿出節點 u
  - 如果 u 已經在最短路徑樹上，跳過它
  - 否則，鬆弛其他不在樹上的節點＋推進 pq

Dijkstra（cont.）

程式碼

```
typedef pair<int,int> pii;
int dis[N], n;
vector<pii> G[N]; // weight, to

void dijkstra(int root)
{
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    for (int i=1; i<=n; i++) dis[i] = 2e9; // set to INF

    dis[root] = 0;
    pq.push(make_pair(0, root));

    while (pq.size())
    {
        int d = pq.top().first;
        int u = pq.top().second;
        pq.pop();

        if (dis[u] != d) continue; // 已經在最短路徑樹上

        for (auto e: G[u])
        {
            int w = e.first;
            int v = e.second;
            if (d + w < dis[v])
            {
                dis[v] = d + w;
                pq.push({dis[v], v});
            }
        }
    }
}
```

Lab 16. Score Game

- n 點 m 邊，單向圖，帶邊權 x
- 求 1 走到 n 的邊權總和最大值，點可以重複踩
- 若無窮大輸出 -1
- $n \leq 2500, m \leq 5000, -10^9 \leq x \leq 10^9$

Lab16. Score Game

Solution

- 最大邊權總和 = -最小邊權相反值總和
- 也就是求 1~n 的最短路徑
- 如果路上可能經過負環，則輸出 -1

Lab16. Score Game

Solution

- 注意：並不是所有負環都會影響到答案
- 邊 (u, v, w) 會影響到答案除非：
  - 1 可以走到 u 且 v 可以走到 n
- 如何檢查？
  - 從 1 dfs，就知道 1 可以到哪些節點
  - 在反圖上從 n dfs，就知道哪些節點可以到 n

Lab16. Score Game
AC Code

- AC Code

```
const int N = 2500 + 5;

int n, m;
int vis1[N], vis2[N];
long long dis[N];
vector<int> G[N], rG[N];
vector<pair<pair<int, int>, int>> E; // { {u, v}, w}

void dfs (vector<int> adj[], int vis[], int u)
{
    vis[u] = 1;
    for (auto v: adj[u])
        if (!vis[v])
            dfs(adj, vis, v);
}
```

```
long long bellman (int root)
{
    for (int i = 1; i <= n; i++)
        dis[i] = 1e18;
    dis[root] = 0;

    for (int i = 1; i <= n; i++)
    {
        for (auto e: E)
        {
            int u = e.first.first;
            int v = e.first.second;
            int w = e.second;

            if (!vis1[u] || !vis2[v]) continue;

            if (dis[u] + w < dis[v])
            {
                if (i == n)
                {
                    return 1;
                }
                dis[v] = dis[u] + w;
            }
        }
    }

    return dis[n];
}
```

Lab16. Score Game

AC Code

```
int main()
{
    cin >> n >> m;
    for (int i = 1; i <= m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        G[u].push_back(v);
        rG[v].push_back(u);
        E.push_back( make_pair( make_pair(u, v), -w) );
    }
    dfs(G, vis1, 1);
    dfs(rG, vis2, n);

    cout << -bellman(1) << '\n';
}
```