الگوریتم دایکسترا

تعریف

الگوریتم دایکسترا راه‌کاری برای پیدا کردن کم‌وزن مسیر از رأس مشخص آغاز به بقیه رئوس در گراف جهت‌دار و وزن‌دار (با وزن‌های مثبت) می‌دهد. وزن یک مسیر در گراف وزن‌دار برابر مجموع وزن یال‌های آن است. جهت‌دار نبودن یال‌ها هم مشکلی ایجاد نمی‌کند و می‌توان برای یال‌های غیر جهت‌دار دو یال فرض کرد.

الگوریتم

فرض کنید $1 \leq s \leq n$ که در آن رأس s رأس آغاز است و فرض کنید:

$$dist(s) = 0$$

و به ازای هر $v \neq s$:

$$dist(v) = \infty$$

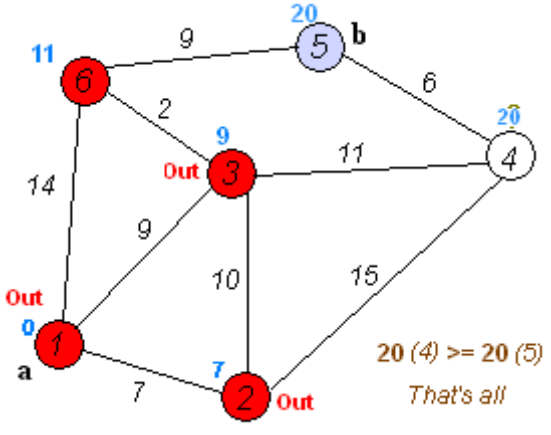
فرض کنید مجموعه‌ی T برابر رئوسی باشد که تا کنون کم وزن‌ترین مسیر آن‌ها را پیدا کرده‌ایم. این الگوریتم در هر مرحله نزدیک‌ترین رأس به s را که تا کنون به مجموعه‌ی T اضافه نشده را انتخاب می‌کند (مثلا x) و آن را به مجموعه‌ی T اضافه می‌کند و فاصله‌ی دیگر رأس‌ها را با توجه به فاصله‌ی x بروز می‌کند. به ازای هر رأس v خارج T:

$$dist(v) = min\Big(dist(v), dist(x) + w(x, v)\Big)$$

که در آن $w(x, v)$ برابر وزن یال بین x و v است. این الگوریتم در هر مرحله رأسی را که کوتاه‌ترین فاصله‌ی آن تا s پیدا شده است را به T اضافه می‌کند. زیرا کوتاه ترین مسیر این رأس از یکی از رأس‌های T می‌گذرد که در مراحل قبلی فاصله آن‌ها بدست آمده و دیگر رئوس را بروز کرده‌اند.

یک مثال

در گراف زیر، روند اجرای این الگوریتم را می‌توانید مشاهده کنید



مشکل با یال‌های منفی

در صورت منفی بودن یال‌ها فرض اینکه در هر مرحله رأسی که کوتاه‌ترین مسیر آن پیدا شده اضافه می‌شود زیر سوال می‌رود. زیرا این رأس می‌تواند بدون استفاده از رأس‌های T و یال‌های منفی مسیری کوتاه‌تر به s پیدا کند.

پیچیدگی الگوریتم

به ازای هر رأس باید روند بالا را طی کنیم. یعنی دنبال آن بگردیم و همه‌ی همسایه‌های آن را پیمایش کنیم. پس پیچیدگی زمانی برنامه از $O(n \times n) = O(n^2)$ است. هر چند می‌توان با استفاده از داده ساختار هرم پیاده‌سازی از $O\Big(e \times lg(n)\Big)$ ارائه داد.

پیاده‌سازی اولیه

شبه کد:

```
  function Dijkstra(s):

      dist[s]  = 0

      for each vertex v in Graph:            // Initializations
          if v ≠ s
              dist[v]  = infinity
          end if
      end for

      while Size(T) ≠ n :                   // The main loop
          u := vertex not in T with min dist[u]
          add u to T

          for each neighbor v of u:
              dist[v] = min(dist[v], dist[u]+w[u][v])
          end for
      end while

  end function
```

پیاده‌سازی

```
#include <iostream>

using namespace std;

const int MAXN = 1000 + 10;
const int INF = 1000*1000*1000;

int n, m;
int adj[MAXN][MAXN];
int w[MAXN][MAXN];
int dist[MAXN];
int mark[MAXN];

void readInput(){
        cin >> n >> m;
        for(int i=0; i<m; ++i){
                int v, u, z;
                cin >> u >> v >> z;
                adj[u][v] = 1;
                w[u][v] = z;
        }
}

void dijkstra(int s){
        dist[s] = 0;
        for(int i=0; i<n; ++i)
                if(i!=s)
                        dist[i] = INF;
        for(int rep=0; rep<n; ++rep)
        {
```

```
                int u = -1;
                int du = INF;
                for(int i=0; i<n; ++i)
                        if(!mark[i] && dist[i] <= du){
                                u = i;
                                du = dist[i];
                        }
                mark[u] = 1;
                for(int v=0; v<n; ++v)
                        if(adj[u][v])
                                dist[v] = min(dist[v], dist[u]+w[u][v]);
        }
}

void writeOutput(){
        for(int i=0; i<n; ++i)
                cout << dist[i] << " ";
        cout << endl;
}

int main(){
        readInput();
        dijkstra(0);
        writeOutput();
        return 0;
}
```

پیاده‌سازی با استفاده از داده ساختار هرم

می‌توان یافتن نزدیک‌ترین رأس در هر مرحله را به کمک این داده ساختار انجام داد. در این صورت باید گراف را به صورت لیست مجاورت نگه داریم. که در $C++$ برای راحتی کار از Set استفاده می کنیم.

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

const int MAXN = 1000*100 + 10;
const int INF = 1000*1000*1000;
typedef pair<int, int> pii;
int n, m;
vector<int> adj[MAXN];
vector<int> w[MAXN];
set<pii> q;
int dist[MAXN];


void readInput(){
        cin >> n >> m;
        for(int i=0; i<m; ++i){
                int v, u, z;
                cin >> u >> v >> z;
                adj[u].push_back(v);
                w[u].push_back(z);
        }
}

void dijkstra(int s){
        dist[s] = 0;
        for(int i=0; i<n; ++i)
                if(i!=s)
                        dist[i] = INF;
        for(int i=0; i<n; ++i)
                q.insert(pii(dist[i], i));
        while(!q.empty())
```

```
        {
                set<pii> :: iterator it = q.begin();
                int u = it->second;
                q.erase(it);
                for(int i=0; i<adj[u].size(); ++i){
                        int v = adj[u][i];
                        q.erase(pii(dist[v], v));
                        dist[v] = min(dist[v], dist[u]+w[u][i]);
                        q.insert(pii(dist[v], v));
                }
        }
}

void writeOutput(){
        for(int i=0; i<n; ++i)
                cout << dist[i] << " ";
        cout << endl;
}

int main(){
        readInput();
        dijkstra(0);
        writeOutput();
        return 0;
}
```