Grafy, BFS, DFS

Autor: Franciszek Pietrusiak

Reprezentacja grafu w programie

1) Lista sąsiedztwa

Inicjalizacja:

```
constexpr int MxN; // maksymalna ilość wierzchołków w grafie
vector<int> G[MxN];
```

Konstruowanie grafu:

```
for (int i=1; i<=m; i++) {
        int a, b; cin >> a >> b;
        G[a].push_back(b); // powstaje krawędź a -> b
        G[b].push_back(a); // powstaje krawędź b -> a
}
```

2) Macierz sąsiedztwa

Inicjalizacja:

```
constexpr int MxN; // maksymalna ilość wierzchołków w grafie
bool G[MxN][MxN];
```

Konstruowanie grafu:

```
for (int i=1; i<=m; i++) {
        int a, b; cin >> a >> b;
        G[a][b] = true; // powstaje krawędź a -> b
        G[b][a] = true; // powstaje krawędź b -> a
}
```

Uwaga: Podane wyżej kody odnoszą się do grafów nieskierowanych, nieważonych. W przypadku innych grafów, należy je nieznacznie zmodyfikować.

BFS

Przeszukiwanie grafu wszerz, skrótowo nazywane BFS'em (*Breadth-first search*) to jedna z podstawowych metod poruszania się po grafie.

Standardowa implementacja:

```
constexpr int MxN; // maksymalna liczba wierzchołków
vector<int> G[MxN]; // lista sąsiedztwa naszego grafu
int st; // wierzchołek startowy

queue<int> Q;
```

```
bool vis[MxN];
int dist[MxN];

Q.push(st);
dist[st] = 0;

while (!Q.empty()) {
        int v = Q.front();
        Q.pop();
        // coś robimy z wierzchołkiem v
        for (auto u : G[v])
                if (vis[u] == false) {
                        Q.push(u);
                        vis[u] = true;
                        dist[u] = dist[v] + 1;
                }
}
```

Złożoność czasowa: $O(n+m)$, gdzie n, m to odpowiednio liczba wierzchołków i liczba krawędzi grafu.

Zastosowanie BFS'a

- znajdowanie najkrótszych ścieżek z wierzchołka startowego do innych wierzchołków w grafie nieskierowanym bez wag
- sprawdzanie spójności grafu
- znajdowanie cyklu w grafie nieskierowanym

Multi-Source BFS

Zauważmy, że nic nie stoi na przeszkodzie, żeby mieć wiele wierzchołków startowych. Wrzucamy je wszystkie do kolejki przed rozpoczęciem pętli `while`. Złożoność algorytmu nie ulega zmianie.

0-1 BFS

Zadanie

Mamy graf skierowany o wagach 0 i 1. Chcemy znaleźć najkrótsze ścieżki z danego wierzchołka startowego do innych wierzchołków.

Rozwiązanie

Oczywiście można by użyć algorytmu Dijkstry, ale jej złożoność to $O(m \log n)$. Da się ten problem rozwiązać w złożoności $O(n+m)$. Użyjemy algorytmu *0-1 BFS*.

Implementacja:

```
constexpr int MxN;
int st;
vector<pair<int, int>> G[MxN];
vector<int> dist(MxN, INF); // INF = bardzo duża liczba, nieskończoność
deque<int> Q;

Q.push_front(st);
```

```
dist[st] = 0;

while (!Q.empty()) {
        int v = Q.front();
        Q.pop_front();
        for (auto e : G[v]) {
                int u = e.first;
                int w = e.second;
                if (dist[u] > dist[v] + w) {
                        dist[u] = dist[v] + w;
                        if (w == 1)
                                Q.push_back(u);
                        else
                                Q.push_front(u);
                }
        }
}
```

DFS

Przeszukiwanie grafu w głąb, skrótowo nazywane DFS'em (*Depth-first search*), jest drugą podstawową metodą poruszania się po grafie.

Standardowa implementacja:

```
constexpr int MxN;
vector<int> G[MxN];
bool vis[MxN];

void dfs(int v) {
        vis[v] = true;
        // coś robimy z wierzchołkiem v
        for (auto u : G[v])
                if (vis[u] == false)
                        dfs(u);
}
```

Złożoność czasowa: $O(n+m)$, gdzie n, m to odpowiednio liczba wierzchołków i liczba krawędzi grafu.

Zastosowania DFS'a:

- sprawdzanie spójności grafu
- sprawdzanie i znajdowanie cykli w grafach
- metoda pre-post order w drzewach
- Sortowanie Topologiczne
- znajdowanie silnie spójnych składowych (*algorytm Kosaraju*).
- znajdowanie mostów i punktów artykulacji
- backtracking

Zadanie Bitmapa

Skrót Treści

Mamy prostokątną planszę wypełnioną białymi i czarnymi polami. Należy dla każdego pola z tej planszy podać odległość do najbliższego białego pola.

✓ **Rozwiązanie**

Multi-Source BFS.

Zadanie Gra platformowa

? **Skrót Treści**

Mamy pewną grę platformową która polega na tym, że na n poziomach znajdują się platformy długości X. Po między tymi platformami są dziury. Jest założenie, że nie ma dwóch dziur obok siebie, oraz bezpośrednio pod sobą. Postać gracza startuje z lewego krańca pewnej platformy i musi dojść do prawego krańca dowolnej platformy. Gracz może wykonywać tylko trzy ruchy:

1. przesuwa się o jedną pozycję w lewo, jeśli znajduje się tam dziura to spada na poziom niżej
2. jeśli jest przed nim dziura to może ją przeskoczyć i wylądować na następnej platformie na tym samym poziomie
3. jeśli jest nad nim dziura to może skoczyć do góry i wylądować na platformie poziom wyżej za dziurą

Musimy odpowiedzieć na zapytania postaci: Jeśli gracz zaczyna z konkretnej platformy, to ile minimalnie ruchów 2) i 3) musi wykonać, aby dojść do prawego krańca dowolnej platformy.

✓ **Rozwiązanie**

Należy:

- skonstruować **skompresowany** graf odpowiadający przechodzeniu gry platformowej
- przechodzić grę od końca, wtedy rozpatrujemy wszystkie zapytania jednocześnie
- zacząć jednocześnie z wszystkich końcowych wierzchołków grafu (multi-source)
- 0-1 BFS

Zadanie Morskie Opowieści

? **Skrót Treści**

Mamy nieskierowany, nieważony graf G. Chcemy odpowiedzieć na dużo zapytań. Zapytanie jest postaci (A, B, k) : Czy jesteśmy w stanie zacząć w wierzchołku A, wykonać **dokładnie** k kroków (przejście z jednego wierzchołka do drugiego po krawędzi) i skończyć w wierzchołku B.

✓ **Rozwiązanie**

Kluczowe obserwacje:

1. jeśli z A do B nie istnieje ścieżka parzystej długości, to jeśli k jest parzyste to odpowiedź to na pewno `NIE`.
2. jeśli z A do B nie istnieje ścieżka nieparzystej długości, to jeśli k jest nieparzyste to odpowiedź to na pewno `NIE`
3. możemy "skracać" naszą ścieżkę o 2 poprzez przechodzenie krawędzi "w tą i z powrotem". A zatem jeśli istnieje ścieżka danej parzystości między A i B, powiedźmy, że jej długość to d, to teraz wszystkie

zapytania postaci (A, B, k), gdzie $k \geq d$ oraz k i d są tej samej parzystości dają odpowiedź `TAK`. Ponadto jeśli d będzie długością najkrótszej ścieżki między A i B, to będziemy w stanie odpowiadać na wszystkie możliwe k.

A zatem dla każdego wierzchołka szukamy najkrótszej ścieżki parzystej i nieparzystej z wierzchołka startowego. Jak to zrobić? Graf warstwowy. Budujemy nowy graf G'. Każdemu wierzchołkowi z grafu G odpowiadają dwa wierzchołki w grafie G' - jeden parzysty, drugi nieparzysty. Następnie krawędzie tworzymy w taki sposób, że łączą one wierzchołki przeciwnej parzystości. Na tak przygotowanym grafie puszczamy BFS'a z parzystego odpowiednika wierzchołka startowego.
Dodatkowo aby szybko odpowiadać na dużo zapytań, należy pogrupować wszystkie zapytania, które mają taki sam wierzchołek startowy i odpowiadać na nie zbiorczo.

Źródła użyte:

- https://cp-algorithms.com/graph/breadth-first-search.html
- https://cp-algorithms.com/graph/01_bfs.html
- https://cp-algorithms.com/graph/depth-first-search.html