Описание на решенията

Нека разгледаме няколко подхода за решаване на задачата, при която имаме компресирани групи от ребра в насочен граф. Целта е да намерим най-краткия път от връх 1 до всеки друг връх.

Съдържание:

- **40% от точките:** Разопаковане на групите от ребра и Дийкстра с приоритетна опашка $\rightarrow O(N^2 \log N)$.
- **60% от точките:** Разопаковане на групите от ребра и Дийкстра без приоритетна опашка $\rightarrow O(N^2)$.
- **100% от точките:** Компресиране на групите от ребра чрез сегментно дърво и изпълнение на Дийкстра $\rightarrow O((N + M) \log^2 N)$. Може да се оптимизира до $O(N \log N)$ използвайки Дийкстра със сложност $O(|V| \log |V| + |E|)$.

1) Решение за 40% от точките

Идея:
За този вариант ”разопаковаме”всяка група от ребра, като за група (u, l, r) добавяме индивидуално ребро от u към всеки връх $v \in [l, r]$ с тегло

$$w = v - l + 1.$$

След това прилагаме стандартния алгоритъм на Дийкстра с приоритетна опашка. Тъй като броят на индивидуалните ребра може да достигне до $O(N^2)$, общата сложност е приблизително $O(N^2 \log N)$.

Algorithm 1 Дийкстра с приоритетна опашка

```
1:  procedure Dijkstra(G, source)
2:      Инициализираме: dist[u] ← ∞ за всички върхове u
3:      dist[source] ← 0
4:      Инициализираме приоритетна опашка PQ с елемент (0, source)
5:      while PQ не е празна do
6:          (d, u) ← най-малкия елемент от PQ
7:          Премахваме елемента от PQ
8:          if d ≠ dist[u] then
9:              Продължаваме към следващата итерация
10:         end if
11:         for all група от ребра, излизащи от u do
12:             for v = l до r do
13:                 w ← v − l + 1
14:                 if dist[v] > d + w then
15:                     dist[v] ← d + w
16:                     Добавяме (dist[v], v) в PQ
17:                 end if
18:             end for
19:         end for
20:     end while
21:     return dist
22: end procedure
```

2) Решение за 60% от точките

Идея:
При пълен граф броят на групите от ребра може да е огромен, затова използваме модифициран вариант на Дийкстра, който не разчита на приоритетна опашка. Вместо това, на всяка итерация обхождаме всички върхове, за да намерим непосетения връх с минимално $dist$. Този подход има сложност $O(N^2)$ и е подходящ, когато N е сравнително малко.

Algorithm 2 Дийкстра без приоритетна опашка

```
1:  procedure QuadraticDijkstra(G, source)
2:      Инициализираме: dist[u] ← ∞ за всички върхове u
3:      dist[source] ← 0
4:      Инициализираме visited[u] ← false за всички върхове u
5:      for i = 1 до N do
6:          Избираме непосетения връх u с минимално dist[u]
7:          if dist[u] = ∞ then
8:              Прекъсваме итерациите
9:          end if
10:         visited[u] ← true
11:         for all група от ребра, излизащи от u do
12:             for v = l до r do
13:                 w ← v − l + 1
14:                 if dist[v] > dist[u] + w then
15:                     dist[v] ← dist[u] + w
16:                 end if
17:             end for
18:         end for
19:     end for
20:     return dist
21: end procedure
```

3) Решение за 100% от точките

Идея:
За да се справим с ограниченията, при които броят на групите от ребра е много голям, използваме сегментно дърво за компресиране на групите от ребра. Основната идея е да построим сегментно дърво върху върховете от 1 до N. При група от ребра (u, l, r) вместо да добавяме ребро за всеки връх $v \in [l, r]$, добавяме ребра от u към върховете на сегментното дърво за интервала $[l, r]$. По този начин броят на ребрата се свежда до $O(M \log N)$, а общата сложност става $(N + M) \log^2 N$.

Algorithm 3 Построяване на сегментно дърво

procedure BuildSegmentTree(l, r, idx)
  if $l = r$ **then**
    Създаваме листов връх v
    Добавяме ребро от v към оригиналния връх l с тегло 1
    return v
  else
    $mid \leftarrow \lfloor (l + r)/2 \rfloor$
    $L \leftarrow$ BuildSegmentTree$(l, mid, 2 \cdot idx + 1)$
    $R \leftarrow$ BuildSegmentTree$(mid + 1, r, 2 \cdot idx + 2)$
    Създаваме вътрешен връх v асоцииран с сегментния връх idx.
    Добавяме ребро от v към L с тегло 0
    Добавяме ребро от v към R с тегло $(mid - l + 1)$
    return v
  end if
end procedure

Algorithm 4 Добавяне на група от ребра (u, ql, qr)

procedure AddEdges(u, ql, qr, l, r, idx)
  if $ql > r$ или $qr < l$ **then**
    return
  end if
  if $ql \leq l$ и $r \leq qr$ **then**
    Добавяме ребро от u към върха съответстващ на сегментния връх idx с тегло $(l - ql)$
    return
  end if
  $mid \leftarrow \lfloor (l + r)/2 \rfloor)$
  AddEdges$(u, ql, qr, l, mid, 2 \cdot idx + 1)$
  AddEdges$(u, ql, qr, mid + 1, r, 2 \cdot idx + 2)$
end procedure

След построяването на сегментното дърво и добавянето на ребрата за всички групи, изпълняваме стандартния алгоритъм на Дийкстра (с приоритетна опашка) върху получения компресиран граф. Това решение може да се оптимизира до $O(N \log N + M = O(N \log N$ с още една модификация на Дийкстра, но решихме да не добавяме допълнителна подзадача.