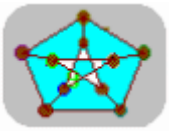
GVHD: Lê Ngọc Thành
E-mail: lnthanh@fit.hcmuns.edu.vn
http://courses.cs.hcmuns.edu.vn



Tìm đường đi ngắn nhất với Bellman

1. Lý thuyết

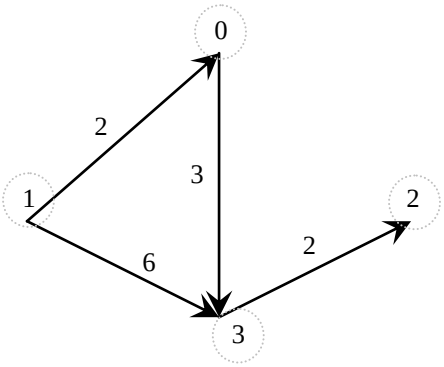
Khác với thuật toán Dijkstra và Floyd, Bellman giải quyết được cả trường hợp đồ thị có cạnh âm. Thuật toán còn phát hiện được cả trường hợp đồ thị có chu trình âm. Từ một đỉnh x cho trước, thuật toán Bellman cho phép xác định đường đi ngắn nhất đến tất cả các đỉnh trong đồ thị.

Nhận xét rằng đường đi ngắn nhất (không chứa chu trình âm) từ một đỉnh x đến một đỉnh y trong một đồ thị có n đỉnh chi đi qua tối đa n đỉnh. Thuật toán Bellman bước 0 bắt đầu với giả thiết rằng từ đỉnh x, nếu đi qua tối đa là 1 đỉnh (kể cả đỉnh xuất phát) ta sẽ chỉ đến được đỉnh x với chi phí là 0. Bước 1, nếu đi qua tối đa là 2 đỉnh, ta sẽ đến được các đỉnh trong bước 0 và các đỉnh i (sao cho có cung nối trực tiếp từ một đỉnh j thuộc bước 0 đến i)…

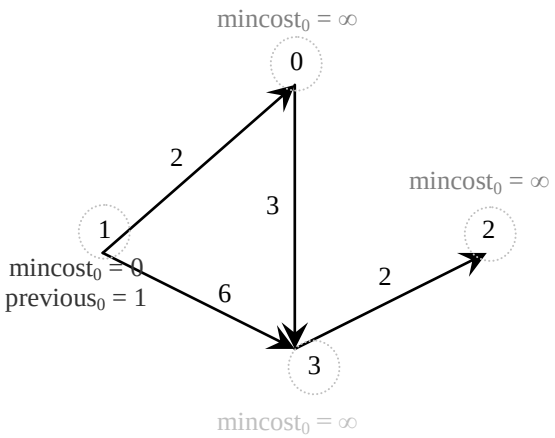
Trong quá trình mở rộng mỗi bước, tại mỗi đỉnh đã đi đến được i đã, ta sẽ lưu lại đỉnh cha thỏa mãn điều kiện đường đi ngắn nhất đến i. Sau đây là một ví dụ minh họa cho thuật toán Bellman.

2. Tìm hiểu qua ví dụ

Vd: cho đồ thị sau:



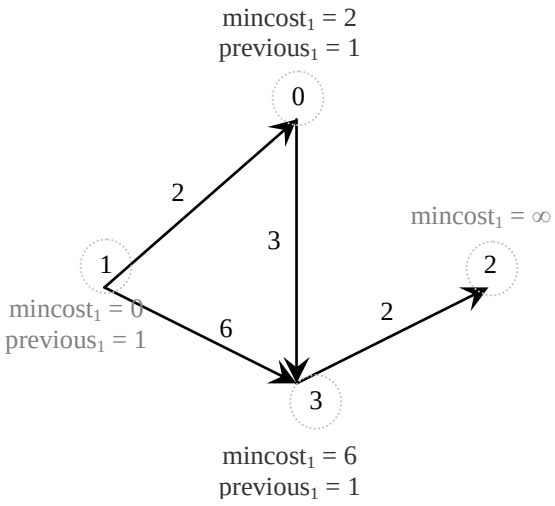
Tìm đường đi ngắn nhất từ đỉnh 1 đến tất cả các đỉnh trong đồ thị.



Bước 0: khởi tạo

khởi tạo đỉnh 1 với chi phí min hiện tại là 0, và đỉnh cha là cha là chính nó (hoặc -1, điều này không quan trọng). Các đỉnh còn lại đều được gán chi phí min là ∞. Dưới đây là bảng mô tả

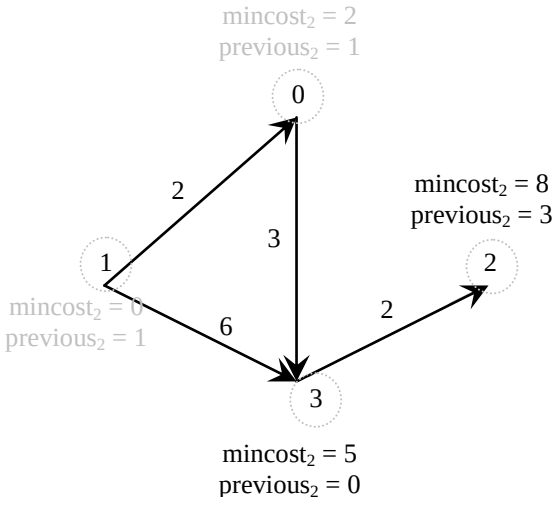
mincost\step	0
0	∞
1	0
2	∞
3	∞



Bước 1: mở rộng bước 0

Ta phát hiện đối với đỉnh số 0, có chi phí hiện tại là ∞, đường đi ngắn nhất (đi qua tối đa 2 đỉnh) đến 0 là đi qua 1, có chi phí là 0+2=2. Tương tự, đối với đỉnh số 3, chi phí hiện tại là ∞, đường đi ngắn nhất (cũng đi qua tối đa 2 đỉnh) đến 3 là đi qua 1, có chi phí là 0+6=6.

mincost\step	0	1
0	∞	2
1	0	0
2	∞	∞
3	∞	6

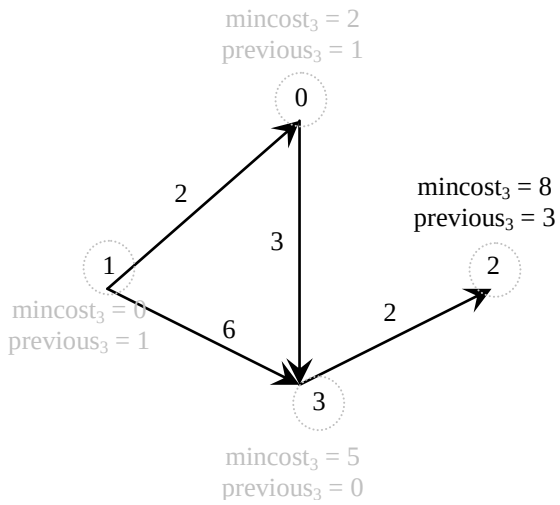


Bước 2:

Đỉnh số 3 có chi phí được cập nhật lại đi qua 0 với chi phí mới là 2+3=5.

Đỉnh số 2 có chi phí hiện tại là ∞, cập nhật lại đi qua 3 với chi phí mới là 6+2=8. Chú ý: chi phí đỉnh 3 được dùng để tính là chi phí của bước 1 (mincost$_1$[3]=6).

mincost\step	0	1	2
0	∞	2	2
1	0	0	0
2	∞	∞	8
3	∞	6	5



Bước 3:

Đỉnh số 2 có chi phí được cập nhật lại đi qua 3 với chi phí mới là 5+2=7.

mincost\step	0	1	2	3
0	∞	2	2	2
1	0	0	0	0
2	∞	∞	8	7
3	∞	6	5	5

3. Trình bày thuật toán Bellman

Cho trước đồ thị có hướng G (có thể chứa cạnh âm). Tìm đường đi xuất phát từ đỉnh x đến tất cả các đỉnh trong đồ thị.

Bước 0: step = 0
`mincost[step][i] = ∞, ∀i ≠ x`
`mincost[step][x] = 0`
	Để có thể lưu lại đường đi, ngắn nhất, ta cần thêm mảng previous để lưu lại đường đi
	`previous[step][x] = x (bằng chính nó)`

Bước 2: step++; nếu `step ≥ n` thì **dừng** thuật toán(*) `mincost[step][i] = min( mincost[step-1][i], min({mincost[step-1][j]+a[j][i]}) ), ∀i,j` Với a[j][i] == ∞ nếu không có cung nối	previous[step][i] =: • j nếu đường đi ngắn nhất đi qua trung gian j rồi đến i • i nếu đường đi đến i đã tối ưu
Bước 3: Nếu `mincost[step][i] == mincost[step-1][i], ∀i` mọi đường đi đã được tối ưu, dừng thuật toán. Ngược lại, quay lại bước 2.	

(*): Nếu như chúng ta đã đi qua đủ n đỉnh nhưng vẫn chưa tối ưu được đường đi, ta có thể kết luận được đồ thị có chu trình âm.

Cài đặt Bellman – Hàm khởi tạo (bước 0)

Không như khi cài đặt các thuật toán Dijkstra, Floy, do Bellman chấp nhận cạnh âm, việc sử dụng trị -1 không còn đúng nữa. Tạm thời, ta có thể sử dụng trị MAXINT (32767) cho giá trị VOCUC, vì nếu như chi phí đạt đến ngưỡng này, có thể xem như tràn số.

```
step = 0;
for (i…)
{
      mincost[step][i] = VOCUC;      (dùng 32767 cho trị vô cực)
      previous[step][i] = i;
}
```

```
mincost[step][x] = 0;
```

Cài đặt hàm Bellman

Chú ý rằng để có thể kết luận được đồ thị có chu trình âm hay không, ta cần chạy đến bước thứ n (nghĩa là đi qua tối đa n+1 đỉnh). Do đó, cấu trúc dữ liệu để lưu cũng cần lưu ý khi khai báo.

```
bSuccess = false;
for (step=1; step<=n; step++)       // (dùng <=n thay vì <n)
{
      for (i…)
      {
            mincost[step][i] = mincost[step-1][i]
            previous[step][i] = previous[step-1][i]
            // tìm các đỉnh j có đường nối từ j --> i
            // và chi phí bước step-1 của j khác ∞
            for (j…)
            if (… && …)
            {
                  // cập nhật lại nếu chi phí bước step của i là ∞
                  // hoặc chi phí đi qua j: mincost[step-1][j]+a[j][i]
                  // tối ưu hơn
                  if (… || …)
                  {
                        // cập nhật lại chi phí và lưu đỉnh cha
                        …
                  }
            }
      }
      // so sánh mincost[step] với mincost[step-1], nếu bằng nhau
      // kết thúc thành công
      int bSame = true;
      for (i…)
      if (mincost[step][i] != mincost[step-1][i])
      {     bSame = false; break;   }
      // đã giống nhau, đường đi đã tối ưu
      if (bSame)
            break;
}
```

Cấu trúc dữ liệu

```
int mincost[MAX+1][MAX];            // chú ý là MAX+1
int previous[MAX+1][MAX];
```

Hàm in kết quả

Nếu nStep == n+1, ta kết luận đồ thị có chu trình âm.

Ngược lại, ta sẽ dò chi phí đi ngược từ bước nStep-1 đến bước 0. (Do bước nStep có giá trị giống bước nStep-1).

```
k = y;
for (i=nStep-1; i>0; i--)           // (chừa lại bước cuối)
{
      printf("%d <--", k);
      k = previous[i][k];           // đỉnh trước k
}
printf("%d\n", k);                  // có thể thêm kiểm tra k == x
```

4. Tham khảo

Tài liệu hướng dẫn thực hành Lý Thuyết Đồ Thị của thầy Lê Thụy Anh.