Đường đi ngắn nhất
Cây khung ngắn nhất

Nội dung

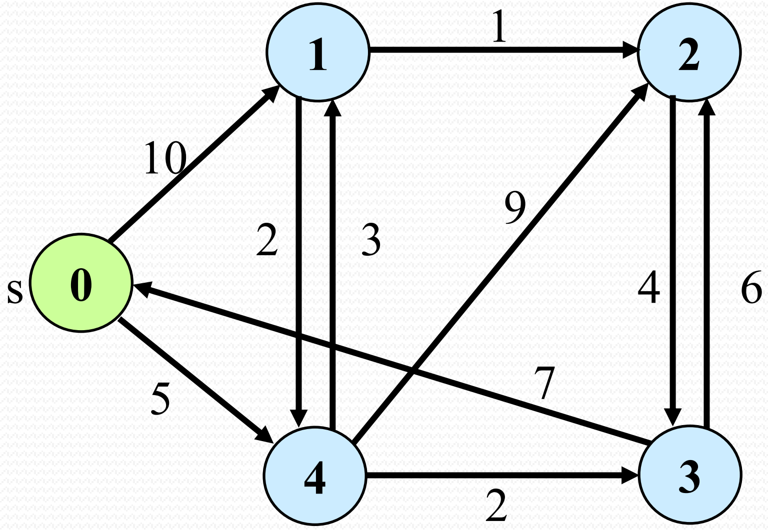
- Đường đi ngắn nhất
  - Thuật toán Dijkstra
- Cây khung
  - Khái niệm
  - Thuật toán tìm cây khung
- Cây khung ngắn nhất
  - Thuật toán Kruskal

Đường đi ngắn nhất

- Đường đi mà tổng trọng số của các cạnh trên đường đi nhỏ nhất
- Thuật toán Dijkstra
  - Đồ thị có trọng số không âm
  - Đỉnh xuất phát s
  - Mảng dist[] lưu tổng trọng số ddnn từ s đến các đỉnh
  - Mảng parent[] lưu vết ddnn từ s đến các đỉnh

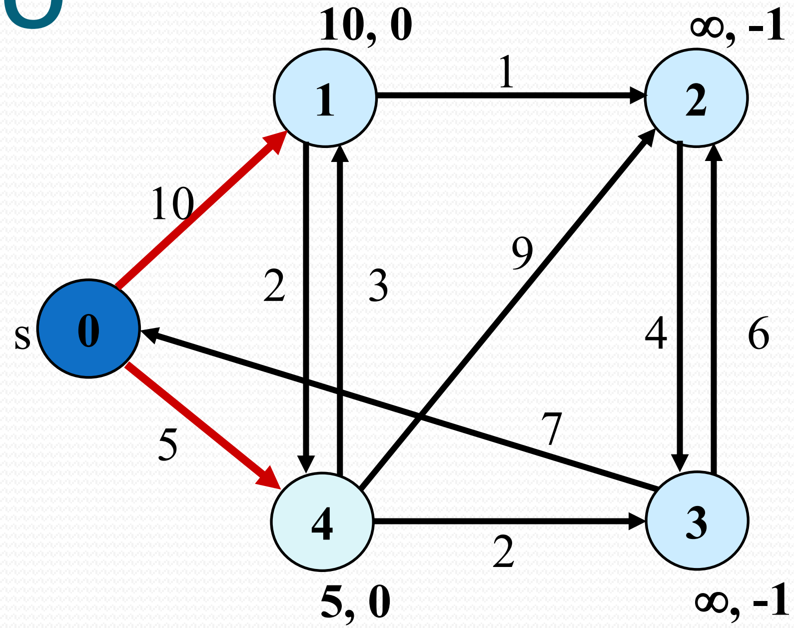
Minh họa

Tìm đường đi ngắn nhất từ 0 đến các đỉnh



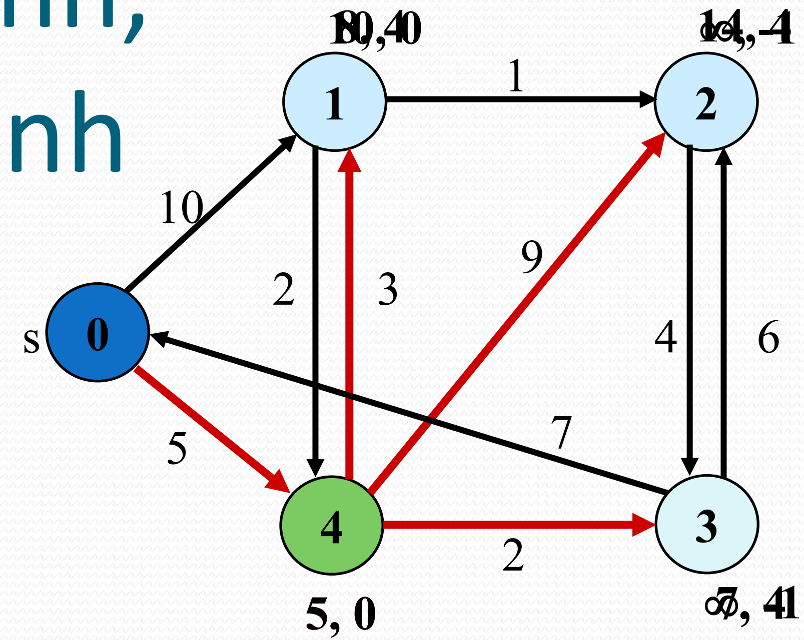
Khởi tạo

S = { 0 }

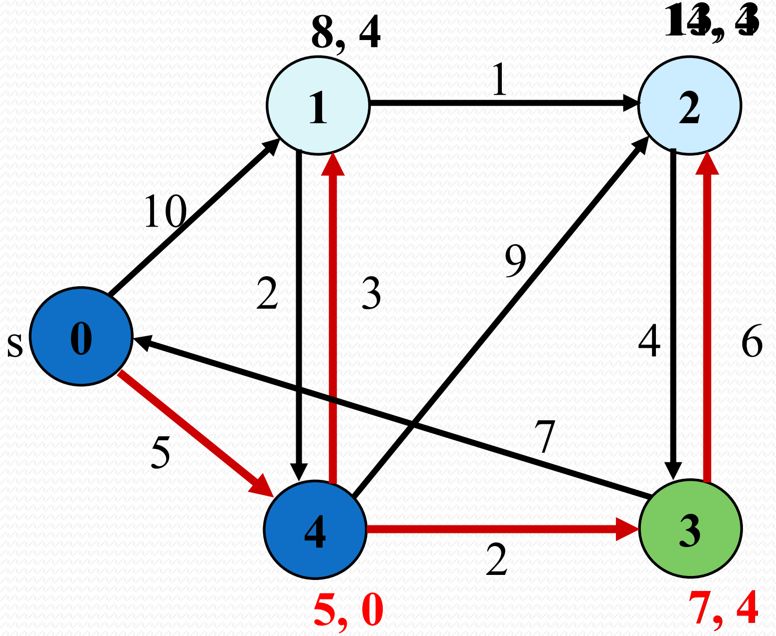


Chọn đỉnh, hiệu chỉnh

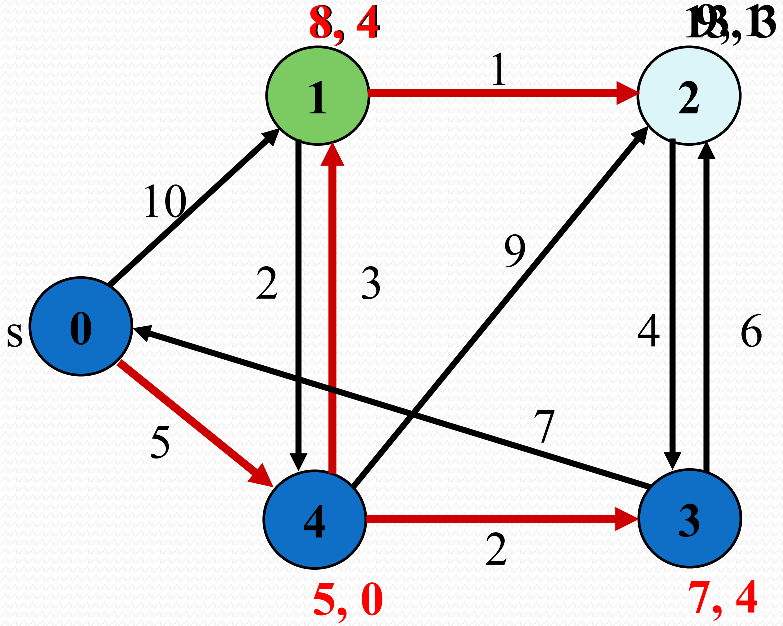
S = { 0, 4 }



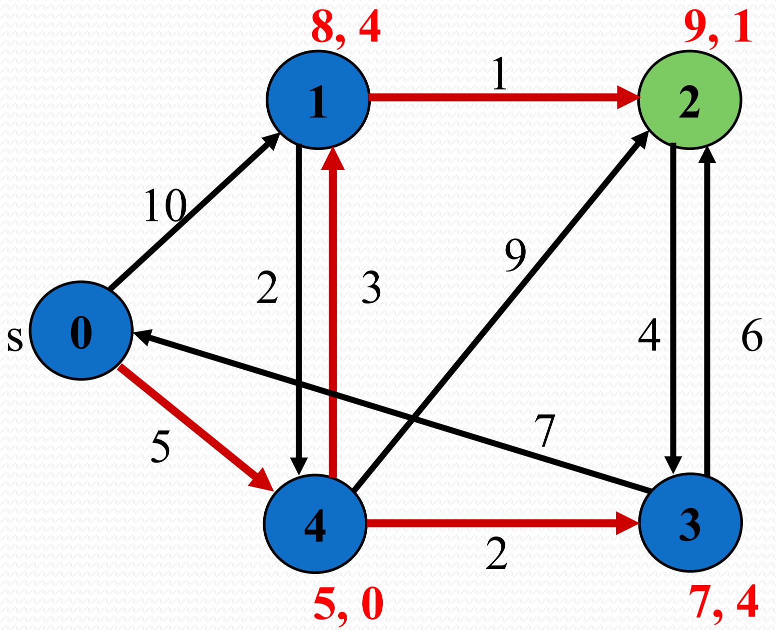
S = { 0, 4, 3 }



S = { 0, 4, 3, 1 }

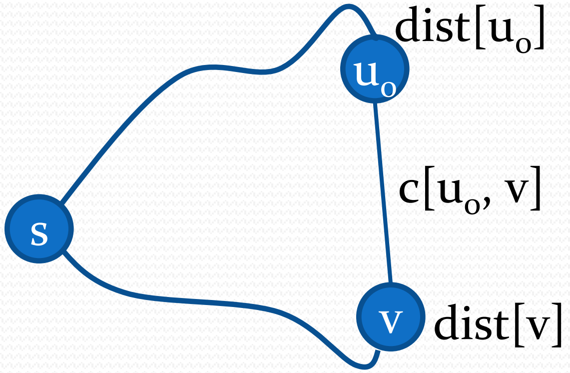


S = { 0, 4, 3, 1, 2 }



Thuật toán Dijkstra

- Khởi tạo:
  - Với mỗi đỉnh u:
    - dist[u]= c[s, u], parent[u]=s, nếu có cạnh từ s đến u
    - dist[u] = ∞, parent[u] = -1, nếu ngược lại
  - S = {s}
- Tìm $u_o \notin S$ có $dist[u_o] = \min\{ d[v], v \notin S \}$
- Kết nạp u_o vào tập S.
- Với mỗi đỉnh $v \notin S$, v kề với u_o, nếu $dist[v] > dist[u_o] + c[u_o,v]$:
  - $dist[v] = dist[u_o] + c[u_o,v]$
  - $parent[v] = u_o$
- Lặp lại bước 2 cho đến khi S = V.



Cài đặt

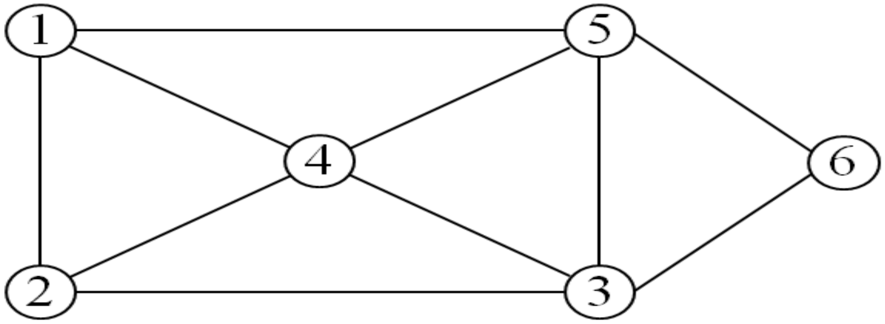
```
void dijkstra(AdjMatrixGraph g, int s, int dist[], int parent[]){
  priority_queue<Node> q;
  bool visited[MAXVERTICES] = {false};
  //Khởi tạo
  for(int i = 0; i < g.numVertices; i++){
    dist[i] = g.adjMatrix[s][i];
    parent[i] = s;
  }
  dist[s] = 0;
  q.push({dist[s],s});
```

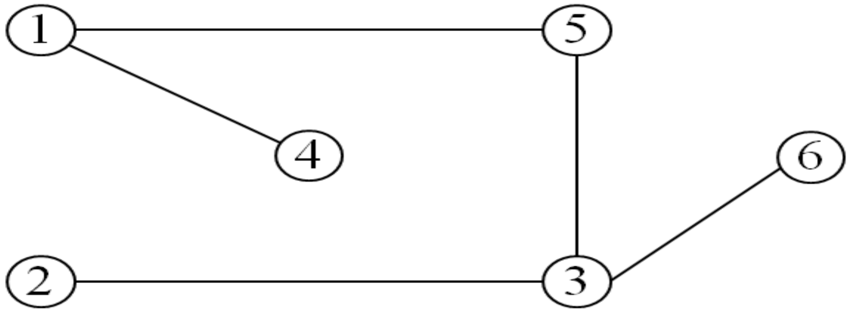
```
struct Node {
  float distance;
  int vertex;
  bool operator<(const Node& other) const {
        return distance > other.distance;
    }
};
```

```
 k = 0;
 while (k < g.numVertics){
 Node node = q.top(); q.pop();
 int u0 = node.vertex;
 if (visited[u0])
      continue;
visited[u0] = true; k++;
 for(int v = 0; v < g.numVertices; v++)
 if(!visited[v] && g.adjMatrix[u0][v]!=VC &&
                    dist[v] > dist[u0]+g.adjMatrix[u0][v]){
    dist[v] = dist[u0] + g.adjMatrix[u0][v];
    parent[v] = u0;
    q.push({dist[v], v});}
}
```

Cây khung

- Cho đồ thị G =<V, E> là 1 đồ thị vô hướng, liên thông. Một cây khung của đồ thị G là một đồ thị H có tập đỉnh là V và tập cạnh là tập con của E thỏa điều kiện:
  - H là một đồ thị liên thông
  - H không có chu trình





Nhận xét

- Một đồ thị có thể có nhiều cây khung.
- Số cạnh của cây khung bằng số đỉnh trừ 1.

Thuật toán tìm cây khung

- Input: đồ thị G
- Output: cây khung H của đồ thị G
- Action:
- Tạo một cây khung H có tập đỉnh trùng với tập đỉnh của G và tập cạnh rỗng.
- Duyệt đồ thị G xuất phát từ đỉnh bất kỳ:
  - Mỗi khi từ đỉnh v thăm đỉnh kề w thì đưa cạnh (v, w) vào cây khung.

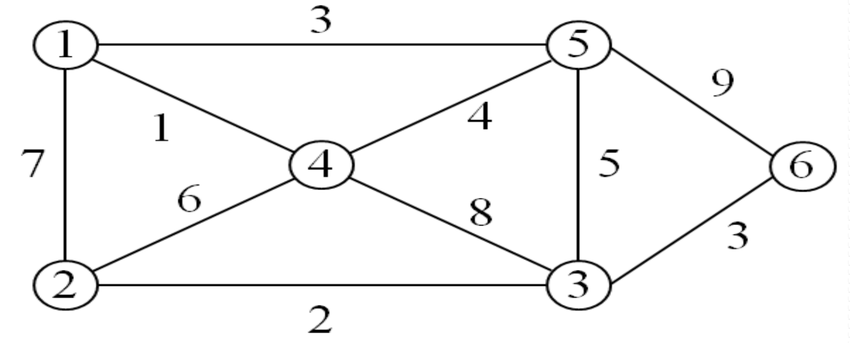
Cài đặt

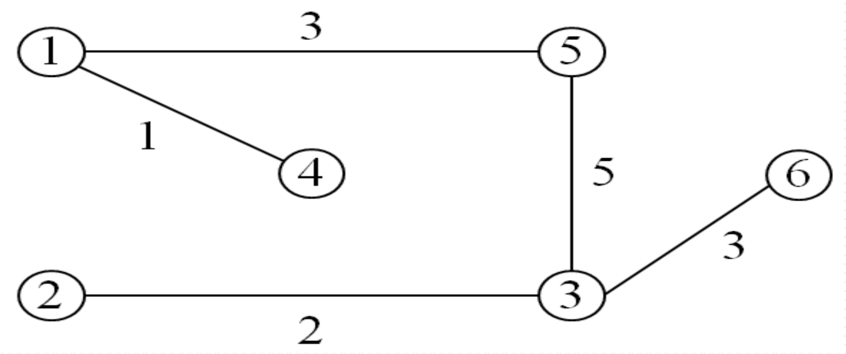
```
AdjMatrixGraph spanningTree(AdjMatrixGraph g){
        bool visted[MAXVERTICES] = {false};
        AdjMatrixGraph h;
        h.numVertices = g.numVertices;
        DFSSpanningTree(g, 0, h, visited);
        return h;
}
```

```
void DFSSpanningTree(AdjMatrixGraph g, int v, AdjMatrixGraph &h,
bool visited[]){
  vistied[v] = true;
  for(int w = 0; w < g.numVertices; w++)
       if(g.adjMatrix[v][w]==1 && !visited[w])
       {
         h.adjMatrix[v][w] = 1;
         h.adjMatrix[w][v] = 1;
         DFSSpanningTree(g, w, h, visited);
       }
}
```

Cây khung ngắn nhất

- H là cây *khung ngắn nhất* của đồ thị *G* nếu tổng các trọng số của cây khung *H* là nhỏ nhất trong các cây khung của *G*.





Thuật toán Kruskal

- Sắp xếp các cạnh của đồ thị theo thứ tự tăng của trọng số
- Xuất phát $T=\varnothing$
- Lặp
  - Chọn cạnh $(u,v)\in E$, có trọng số nhỏ nhất
  - $E:=E \setminus \{(u, v)\}$
  - Nếu $T\cup\{(u,v)\}$ không chứa chu trình thì $T:=T\cup\{(u,v)\}$
- Đến khi $|T|=|V|-1$

Cài đặt

- Đồ thị biểu diễn danh sách cạnh
- Dùng mảng component[] để lưu thành phần liên thông của các đỉnh

```
EdgeListGraph minimumSP(EdgeListGraph g)
{
  EdgeListGraph h;int i,k;
  h.numVertices = g.numVertices;
  h.numEdges = 0;
  sort(g); //Sắp các cạnh của đồ thị tăng của trọng số
  for(int i = 0; i < g.numVertices; i++)
       component[i] = i;
  k=0;
```

```
  while (h.numEdges < h.numVertices-1) {
   int u = g.edgeList[k].startVertex;
   int v = g.edgeList[k].destVertex;
   if (component[u]!=component[v]) {
     h.edgeList[h.numEdges] = g.edgeList[k];
     h.numEdges++;
     mergeComponents(component, h.numVertices, u, v);
   }
   k++;
  }
  return h;
}
```

```
//Ghép thành phần liên thông 2 a và b
void mergeComponents(int component[], int n, int u, int v) {
    int a = component[u], b = component[v];
        for(int i = 0; i < n; i++)
                if (component[i] == a) {
         component[i] = b;
    }
}
```

Độ phức tạp thuật toán

- Thao tác sắp xếp các cạnh của đồ thị có độ phức tạp O(mlogm)
- Thao tác tìm cây khung lặp tối đa m lần, mỗi lần ghép 2 thành phần liên thông O(n) nên độ phức tạp O(m.n)
- Do $m \leq n^2$ nên $\log_2 m \leq 2\log_2 n \leq n$, với n đủ lớn.
- Vậy độ phức tạp của thuật toán trên là O(mn), với n là số đỉnh, m là số cạnh.
- Có thể thay sắp xếp các cạnh bằng cách tạo heap cho các cạnh chưa được chọn.

Tổng kết

- Đường đi ngắn nhất là bài toán có nhiều ứng dụng
- Thuật toán Dijkstra chỉ tìm được đường đi trên đồ thị có trọng số không âm, xuất phát từ 1 đỉnh.
- Cây khung là một đồ thị tối thiểu về số cạnh mà vẫn đảm bảo liên thông