GRAPH- DIJKSTRA ALGORITHM

Giảng viên: Tiến sĩ Bùi Thanh Hùng
Trưởng Lab Khoa học Phân tích dữ liệu và Trí tuệ nhân tạo
Giám đốc chương trình Trí tuệ nhân tạo & Hệ thống thông tin
Đại học Thủ Dầu Một
Email: hungbt3@fe.edu.vn
Website: https://sites.google.com/site/hungthanhbui1980/

- Giải thuật DFS của 3 cách là như nhau, tuy nhiên code có thay đổi chút cho phù hợp với cấu trúc dữ liệu sử dụng

```
public static void DFS(int i) {
    visited[i] = true;
    System.out.print(i + " ");
    for (int j = 0; j < size; j++) {
        if (a[i][j] > 0 && visited[j] == false) {
            DFS(j);
        }
    }
}
```

```
[ 0]:=[1, 2]
[ 1]:=[0, 3]
[ 2]:=[0]
[ 3]:=[1]
```

```
void DFS(int i){
    visited[i] = true;
    System.out.print(i + " ");
    for (int j = 0; j < adjLists[i].size(); j++){
        int t= adjLists[i].get(j);
        if (visited[t]== false)
            DFS(t);
    }
}
```

```
[ 0]:=[[1, 2], [2, 4]]
[ 1]:=[[0, 2], [3, 5]]
[ 2]:=[[0, 4]]
[ 3]:=[[1, 5]]
```

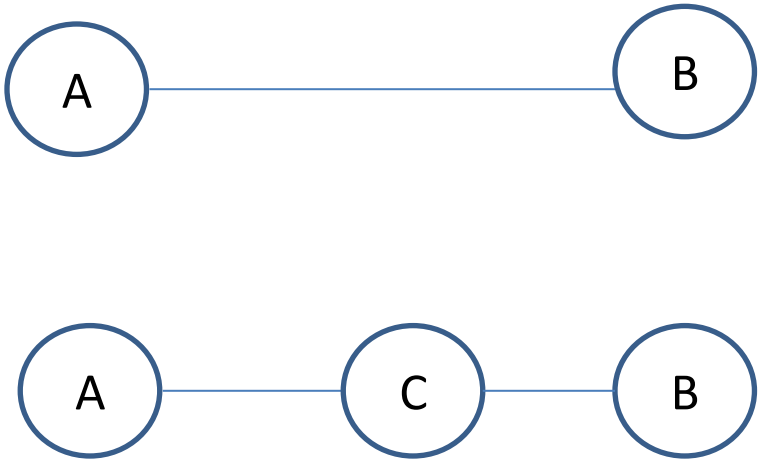
```
void DFS(int i) {
    visited[i] = true;
    System.out.print(i + " ");
    for (int j = 0; j < adjLists[i].size(); j++) {
        int t = adjLists[i].get(j).dest;
        if (visited[t] == false) {
            DFS(t);
        }
    }
}
```

Đường đi ngắn nhất trên đồ thị

Các thuật toán quan trọng nhất giải quyết bài toán này là:

- Thuật toán Dijkstra — giải bài toán nguồn đơn nếu tất cả các trọng số đều không âm. Thuật toán này có thể tính toán tất cả các đường đi ngắn nhất từ một đỉnh xuất phát cho trước *s* tới *mọi đỉnh khác* mà không làm tăng thời gian chạy.
- Thuật toán Bellman-Ford — giải bài toán nguồn đơn trong trường hợp trọng số có thể có giá trị âm.
- Giải thuật tìm kiếm A* giải bài toán nguồn đơn sử dụng heuristics để tăng tốc độ tìm kiếm
- Thuật toán Floyd-Warshall — giải bài toán đường đi ngắn nhất cho mọi cặp đỉnh.
- Thuật toán Johnson — giải bài toán đường đi ngắn nhất cho mọi cặp đỉnh, có thể nhanh hơn thuật toán Floyd-Warshall trên các đồ thị thưa.
- Lý thuyết nhiễu (*Perturbation theory*); tìm đường đi ngắn nhất địa phương (trong trường hợp xấu nhất)

Dijkstra's Algorithm - 1



Ý tưởng của giải thuật
AB ngắn nhất
C là 1 điểm nằm trên đường đi ngắn nhất này
(AC + CB) cũng là ngắn nhất

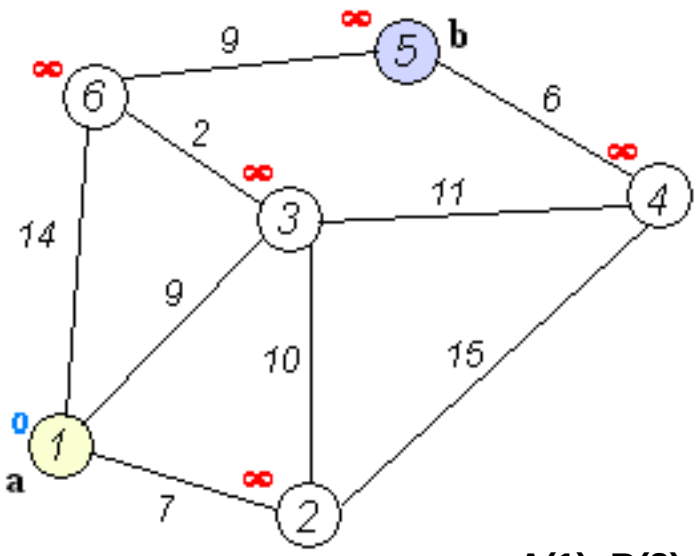
Dijkstra's Algorithm - 1

The main idea in applying the greedy-method pattern to the single-source shortestpath problem is to perform a “weighted” breadth-first search starting at the source vertex *s*. In particular, we can use the greedy method to develop an algorithm that iteratively grows a “cloud” of vertices out of *s*, with the vertices entering the cloud in order of their distances from *s*. Thus, in each iteration, the next vertex chosen is the vertex outside the cloud that is closest to *s*. The algorithm terminates when no more vertices are outside the cloud (or when those outside the cloud are not connected to those within the cloud), at which point we have a shortest path from *s* to every vertex of *G* that is reachable from *s*. This approach is a simple, but nevertheless powerful, example of the greedy-method design pattern. Applying the greedy method to the single-source, shortest-path problem, results in an algorithm known as ***Dijkstra’s algorithm***.

Dijkstra's Algorithm - 2

```
DijkstraAlgorithm(non-gegative weighted simple
digraph, vertex first)
for all vertices  v # first currDist(v) = ∞;
currDist(first) = 0;
toBeChecked = V (all vertices);
checked = empty;
while toBeChecked is not empty
 u = a vertex in toBeChecked with min.currDist(u);
 remove u from toBeChecked and add to checked;
 for all vertices v adjacent to u and in toBeChecked
  if (currDist(v) > currDist(u) + weight(edge(uv)))
   {currDist(v) = currDist(u) + weight(edge(uv))
    predeccessor(v) = u;
   }
```

Dijkstra's Algorithm example - 1




```
Dijkstra algorithm for shortest path from A to  E:
     The S set:          B         C         D         E         F

 0:            A  (  7,A)  (  9,A)  (INF,A)  (INF,A)  ( 14,A)

 1:           AB  (  7,A)  (  9,A)  ( 22,B)  (INF,A)  ( 14,A)

 2:          ABC           (  9,A)  ( 20,C)  (INF,A)  ( 11,C)

 3:         ABCF                    ( 20,C)  ( 20,F)  ( 11,C)

 4:        ABCFD                    ( 20,C)  ( 20,F)

 5:       ABCFDE                             ( 20,F)

The length of shortest path from A to  E is  20

Path:
A -> C -> F -> E
```

A(1), B(2), C(3), D(4), E(5), F(6)

Dijkstra's algorithm keeps *two* sets of vertices:

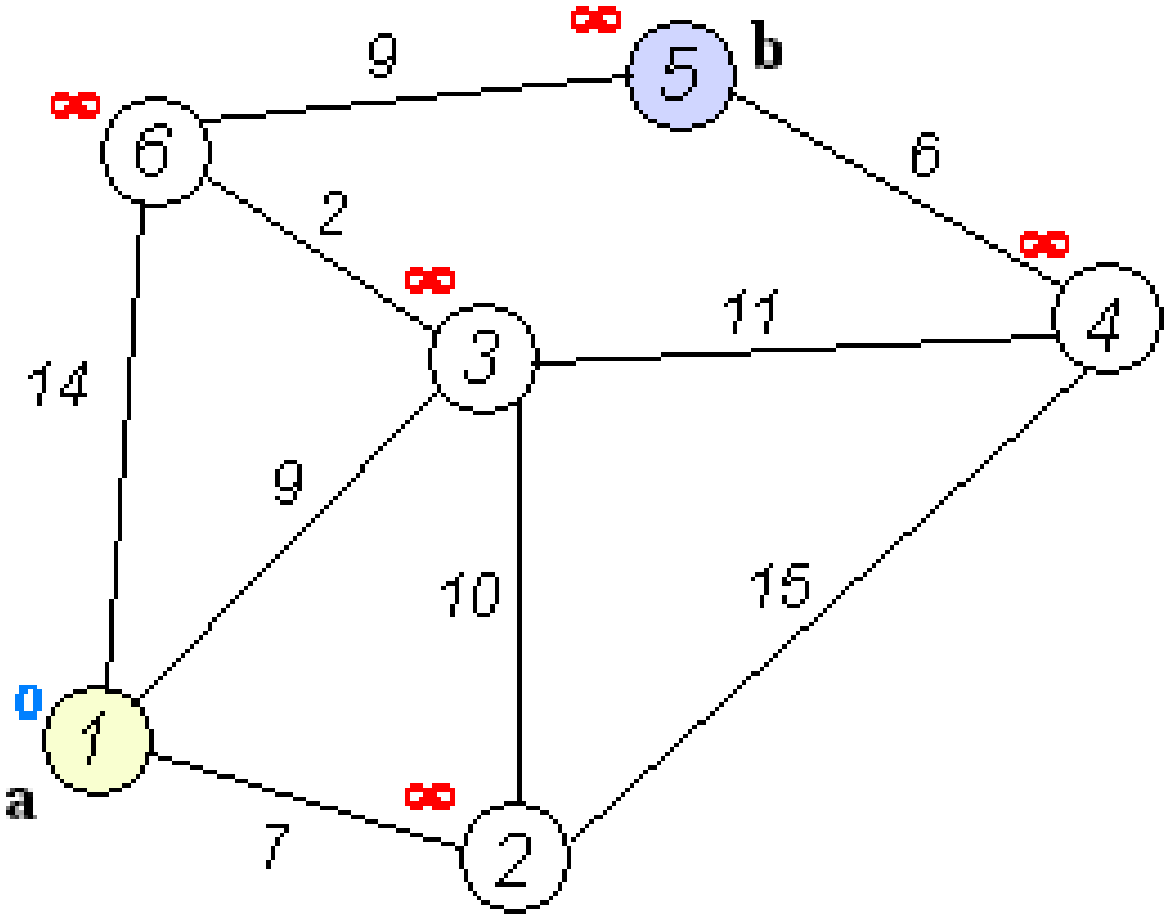
- S — Vertices whose shortest paths have already been determined
- V-S — Remainder

Also

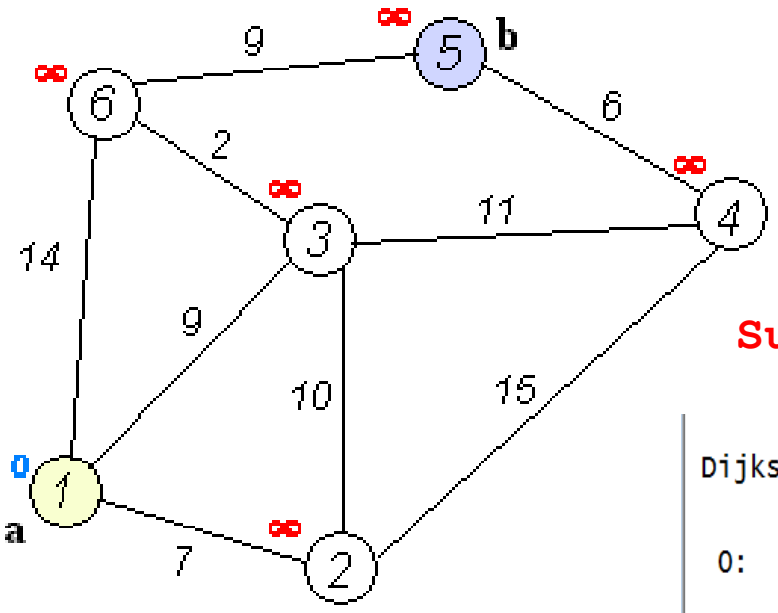
- d — Best estimates of shortest path to each vertex
- p — Predecessors for each vertex

```
int [][] b = {
   { 0,   7,   9, 99, 99, 14},
   { 7,   0, 10, 15, 99, 99},
   { 9, 10,   0, 11, 99,   2},
   {99, 15, 11,   0,   6, 99},
   {99, 99, 99,   6,   0,   9},
   {14, 99,   2, 99,   9,   0}
  };
```

Dijkstra's Algorithm example - 1



Dijkstra's Algorithm example - 1



Suy nghĩ và giải thích bảng sau?

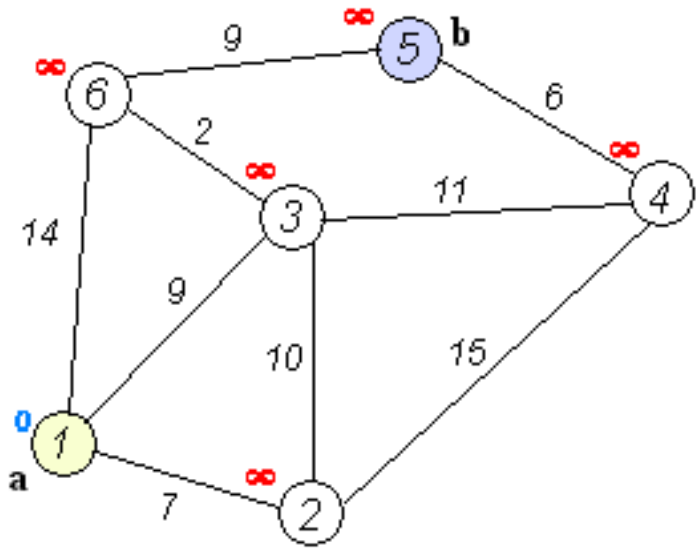
Dijkstra algorithm for shortest path from A to E:

The S set:	B	C	D	E	F
0: A	(7,A)	(9,A)	(INF,A)	(INF,A)	(14,A)
1: AB	(7,A)	(9,A)	(22,B)	(INF,A)	(14,A)
2: ABC		(9,A)	(20,C)	(INF,A)	(11,C)
3: ABCF			(20,C)	(20,F)	(11,C)
4: ABCFD			(20,C)	(20,F)	
5: ABCFDE				(20,F)	

The length of shortest path from A to E is 20

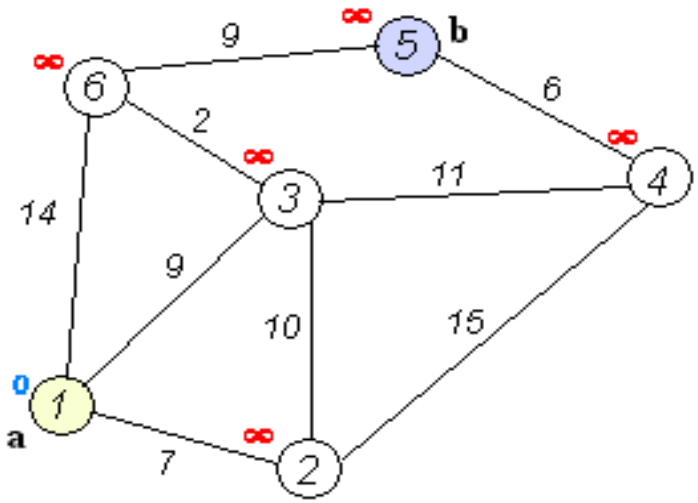
Path:
A -> C -> F -> E

A(1), B(2), C(3), D(4), E(5), F(6)



- S[]= Tập đỉnh đã chọn
- L[i] = độ dài quãng đường từ đỉnh nguồn tới đỉnh i
- P[i]= đỉnh kề với đỉnh i (để lưu vết đường đi)

Bước 1: Bắt đầu từ đỉnh 1 (a):



- S[]= Tập đỉnh đã chọn
- L[i] = độ dài quãng đường từ đỉnh nguồn tới đỉnh i
- P[i]= đỉnh kề với đỉnh i (để lưu vết đường đi)

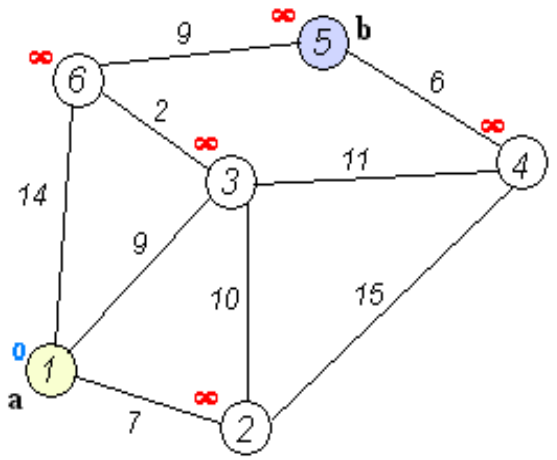
Bước 1: S= {Bắt đầu từ đỉnh 1 (a) }:

Đến 2 L[2]= 7 P[2]= 1 S[2]= True

Đến 3 L[3]= 9 P[3]= 1

Đến 6 L[6]= 14 P[6]= 1

Bước 2: S= {Tập đỉnh đã chọn (1,2)}



- S[]= Tập đỉnh đã chọn
- L[i] = độ dài quãng đường từ đỉnh nguồn tới đỉnh i
- P[i]= đỉnh kề với đỉnh i (để lưu vết đường đi)

Bước 1: S= {Bắt đầu từ đỉnh 1 (a) }:

Đến 2 L[2]= 7 P[2]= 1 S[2]= True

Đến 3 L[3]= 9 P[3]= 1

Đến 6 L[6]= 14 P[6]= 1

Bước 2: S= {Tập đỉnh đã chọn (1,2)}

Đến 3 L[3] = 9 P[3]= 1 S[3]= True

Đến 4 L[4] = L[2] + a[2,4] = 22 P[4]= 2

Đến 6 L[6]= 14 P[6] = 1

Bước 3: S= {Tập đỉnh đã chọn (1,2,3)} .....

P[5]= 6, P[6]= 3, P[3]= 1 5-6-3-1: L[5] = 20

Bước 1: S= {Bắt đầu từ đỉnh 1 (a) = FirstNode }:

Đến 2 L[2]= 7 P[2]= 1 S[2]= True

Đến 3 L[3]= 9 P[3]= 1

Đến 6 L[6]= 14 P[6]= 1

Bước 2: S= {Tập đỉnh đã chọn (1,2)}

Đến 3 L[3] = 9 P[3]= 1 S[3]= True

Đến 4 L[4] = L[2] + a[2,4] = 22 P[4]= 2

Đến 6 L[6]= 14 P[6] = 1

Bước 3: S= {Tập đỉnh đã chọn (1,2,3)}

Đến 4 L[4] = L[3] + a[3,4] = 20 P[4]= 3

Đến 6 L[6]= L[3] + a[3,6] =11 P[6] =3 S[6]= True

Bước 4: S= {Tập đỉnh đã chọn (1,2,3,6)}

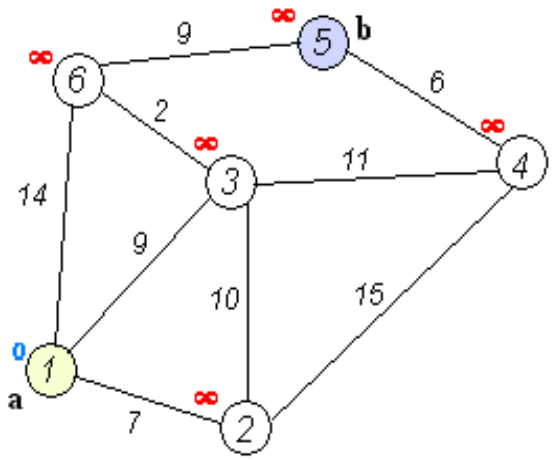
Đến 4 L[4] = L[3] + a[3,4] = 20 P[4]= 3 S[4]= True Chọn 4 vì xét từ nhỏ tới lớn

Đến 5 L[5]= L[6] + a[5,6] =20 P[5] =6

Bước 5: S= {Tập đỉnh đã chọn (1,2,3,6,4)}

Đến 5 L[5]= L[6] + a[5,6] =20 P[5] =6 S[5]= True

Bước 6: S= {Tập đỉnh đã chọn (1,2,3,6,4,5)} đã chứa LastNode



S[]= Tập đỉnh đã chọn

L[i] = độ dài quãng đường từ đỉnh nguồn tới đỉnh i

P[i]= đỉnh kề với đỉnh i (để lưu vết đường đi)

L[5] = 20 P[5]= 6, P[6]= 3, P[3]= 1 5-6-3-1

```
static int vocung= 100000;
static void Dijkstra(int Fnode, int Lnode){
    boolean S[] = new boolean[size];
    int L[]= new int[size];
    int P[]= new int[size];
    for (int i=0; i<size;i++)
      for (int j=0; j<size;j++)
          if (i!=j && a[i][j]==0) a[i][j]= vocung;
    for (int i=0; i<size;i++){
       S[i]= false;    L[i]= vocung; P[i] = Fnode;}
    int i;    L[Fnode] = 0;
    while (S[Lnode]== false){
        //Đi tìm những đỉnh kề với đỉnh đang xét và có L[] là ngắn nhất, giả sử đó là i
        S[i] = true;
        for (int j=0; j<size;j++)
            if (S[j] == false  && L[j] > L[i] +  a[i][j]) {    L[j] = L[i] +  a[i][j];   P[j] = i;}
    }
    System.out.println("Do dai duong di ngan nhat  la: "  + L[Lnode];
}
```

```
static int[][] a= { { 0,  7, 9, 0, 0, 14},
                    { 7,  0, 10, 15, 0, 0},
                    { 9,  10, 0, 11, 0, 2},
                    { 0,  15, 11, 0, 6, 0},
                    { 0,  0, 0, 6, 0, 9},
                    { 14,  0, 2, 0, 9, 0}};
static int size=6;
```

```
static int vocung= 100000;
static void Dijkstra(int Fnode, int Lnode){
    boolean S[] = new boolean[size];
    int L[]= new int[size];
    int P[]= new int[size];
    for (int i=0; i<size;i++)
      for (int j=0; j<size;j++)
          if (i!=j && a[i][j]==0) a[i][j]= vocung;
    for (int i=0; i<size;i++){
       S[i]= false;    L[i]= vocung; P[i] = Fnode;}
    int i;    L[Fnode] = 0;
    while (S[Lnode]== false){
        //Đi tìm những đỉnh kề với đỉnh đang xét và có L[] là ngắn nhất, giả sử đó là i
        S[i] = true;
        for (int j=0; j<size;j++)
            if (S[j] == false  && L[j] > L[i] +  a[i][j]) {    L[j] = L[i] +  a[i][j];   P[j] = i;}
    }
    System.out.println("Do dai duong di ngan nhat  la: "  + L[Lnode];
}
```

```
static int[][] a= { { 0,  7, 9, 0, 0, 14},
                    { 7,  0, 10, 15, 0, 0},
                    { 9,  10, 0, 11, 0, 2},
                    { 0,  15, 11, 0, 6, 0},
                    { 0,  0, 0, 6, 0, 9},
                    { 14,  0, 2, 0, 9, 0}};
static int size=6;
```

1- Bạn hãy hoàn thiện giải thuật trên (bổ sung khoảng 6-8 dòng lệnh)

2- Hãy sử dụng **MinHeap** cho giải thuật Dijkstra

Exercise

Bài 1:

Hiển thị đường đi ngắn nhất từ điểm đầu đến điểm cuối

Bài 2

Hiển thị đường đi ngắn nhất từ 1 điểm đến mọi điểm trên đồ thị

Sử dụng hàm Dijkstra(FNode, Lnode) ở Bài 1

Bài 3:

Sao cho toàn bộ hàm Dijkstra(FNode, Lnode) ở Bài 1 sang đồ thị biểu diễn bằng danh sách kề(nguồn, đích, trọng số).

Hãy sửa lại code để giải thuật vẫn chạy đúng

Bài 4:

Xây dựng giải thuật Dijkstra bằng MinHeap