**Báo cáo cuối kỳ CTDL Chapter 16\_ Nhóm 15:**

**Thuật toán:**

B1: Khởi tạo mạng các cung ban đầu gồm cung nguồn , cung giữa  và cung đích . Các cung này có dung lượng (capacity) bằng 1.

B2: Tính toán giá trị “desirability” của các cung  và  (các cung này có dung lượng (capacity) bằng vô hạn) theo công thức sau:

* Trên cung :



* Trên cung :



B3: Sắp xếp các cung  và  theo giá trị “desirability”.

B4: Thêm lần lượt các cung  và  đã được sắp xếp giảm dần vào mạng và thực hiện tăng luồng cho mạng cho đến khi giá trị luồng cực đại của mạng bằng m thì dừng. (Để kiểm tra giá trị luồng cực đại của mạng dùng “thuật toán Ford-Fulkerson”)

B5: Xuất ra giá trị luồng của các cung giữa  bằng 0 hoặc 1 là nghiệm của bài toán. Giá trị discrepancy (độ chênh lệch) của bài toán bằng 1 trừ cho “desirability” của cung  hoặc  mới được thêm vào.

**Giải thích code doan.cpp:**

- Dòng 39-44: Nhập input là mảng các phần tử số thực  thỏa mãn 

- Dòng 45: Tính tổng s của mảng các phần tử

- Dòng 46-52: Nếu tổng s không là số nguyên thì  và thêm 1 phần tử  vào mảng. Nếu s là số nguyên thì m=s

- Dòng 56-57: Nhập các phần tử của tập hoán vị σ của tập {1,2,…,n}

- Dòng 62: Hàm createNetwork(double A[], double B[], int Hvi[], int n): Mục đích là khởi tạo mạng các cung ban đầu ở B1 của thuật toán

* Dòng 191-200: Khởi tạo các cung  và 
* Dòng 201-205: Khởi tạo các cung 
* Sử dụng mảng 2 chiều Network[socanh][i] để lưu thông tin các cung. Network[socanh][0] lưu điểm đầu, Network[socanh][1] lưu điểm cuối của cung, Network[socanh][2] lưu capacity.

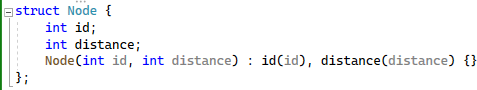
- Dòng 63: Hàm createInitialArc(double A[], double B[], int Hvi[], int n)

* Dòng 210-221: Tính tập tổng từng phần S và Σ (lưu trong mảng S và Shv)
* Dòng 222-236: Tính “desirability” của các cung  và  rồi lưu vào listcanh.
  + Dòng 225-229: Kiểm tra nếu [j − 1 . . j ) ∩ [. . ) ∅ thì cung  tồn tại và tính desirability theo công thức: 
  + Dòng 230-235:Kiểm tra nếu [j − 1 . . j ) ∩ [. . ) ∅ thì cung  tồn tại và tính desirability theo công thức: 
* Dòng 237: Sắp xếp các desirability theo chiều giảm dần (hàm sort(int n))

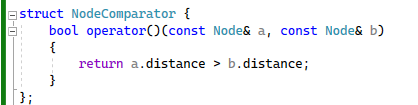
- Dòng 64: Hàm twoWayRounding(): Thực hiện thuật toán two-way rounding:

* Dòng 69: Hàm enter(): Nhập chỉ số cho đỉnh đầu, đỉnh cuối và các cạnh của mạng
* Dòng 70-78: Thực hiện thuật toán two-way rounding trong vòng lặp vô hạn:
  + Dòng 71: Thực hiện “thuật toán Ford-Fulkerson” để tăng luồng cho mạng.
  + Dòng 72: Hàm tinhMaxFlow() để tính giá trị luồng cực đại hiện tại (maxflow) của mạng.
  + Dòng 74-75: Kiểm tra xem nếu giá trị luồng cực đại của mạng bằng m thì kết thúc hàm twoWayRounding().
  + Dòng 76: Nếu luồng cực đại chưa bằng m thì thêm cung trong listcanh rồi quay lại dòng 71 thực hiện thuật toán Ford-Fulkerson.
* Dòng 81: Xuất kết quả trong hàm printResult():
  + Dòng 104-108: Xuất ra mạng hiện tại thỏa mãn two-way rouding.
  + Dòng 109: Xuất ra giá trị luồng cực đại hiện tại (maxflow)
  + Dòng 110-111: Xuất ra luồng của các cung  tương ứng với các nghiệm  cần tìm.
* Dòng 80,82: Tính discrepancy của bài toán bằng 1 trừ cho cung vừa mới được thêm vào.

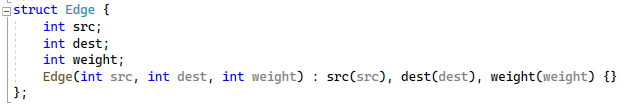
**Code demo ứng dụng: Áp dụng two way rounding tìm đường đi ngắn nhất**

**

🡪tạo 1 struct Node để biểu diễn 1 node trong đồ thị

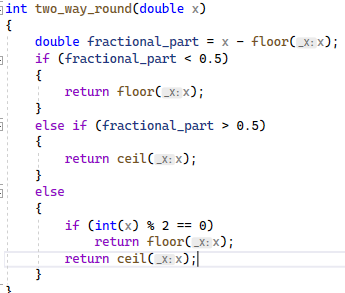


🡪 xác định 1 bộ so sánh để ưu tiên các node trong queue



🡪Định nghĩa 1 cạnh trong đồ thị

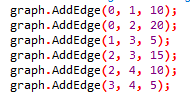
Dòng 31 đến 85: tạo 1 graph và xử lí đường đi ngắn nhất

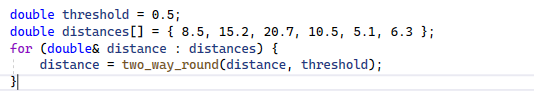


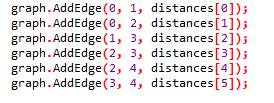
🡪Hàm two way rounding với tham số x là số cần làm tròn nếu phần sau dấu chấm <0.5 thì làm tròn xuống bằng floor nếu phần ấy>0.5 thì làm tròn lên bằng ceil và nếu phần sau dấu chấm bằng 0.5 thì làm tròn đến số nguyên chẵn gần nhất.

Hàm main:

🡪tạo 1 đồ thị với 5 node

🡪thêm các cạnh với với tham số lần lượt là src, dest, weight

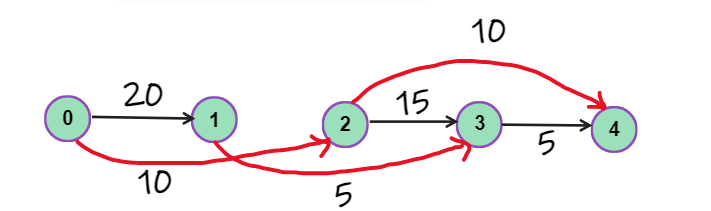
🡪Gán thresold =0.5 và gán khoảng cách 2 điểm bằng cách làm tròn các số đầu vào trong 1 mảng các số thực

🡪 Sử dụng khoảng cách được làm tròn làm trọng số của các cạnh trong biểu đồ

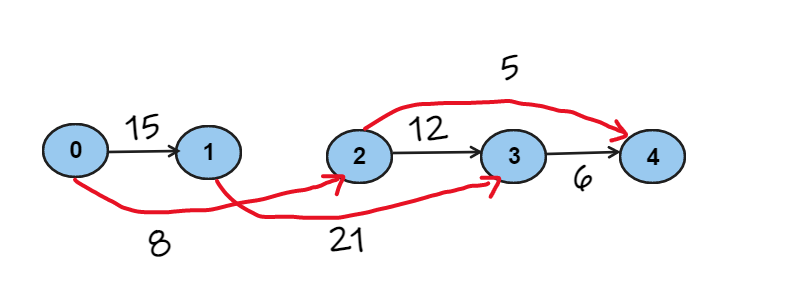


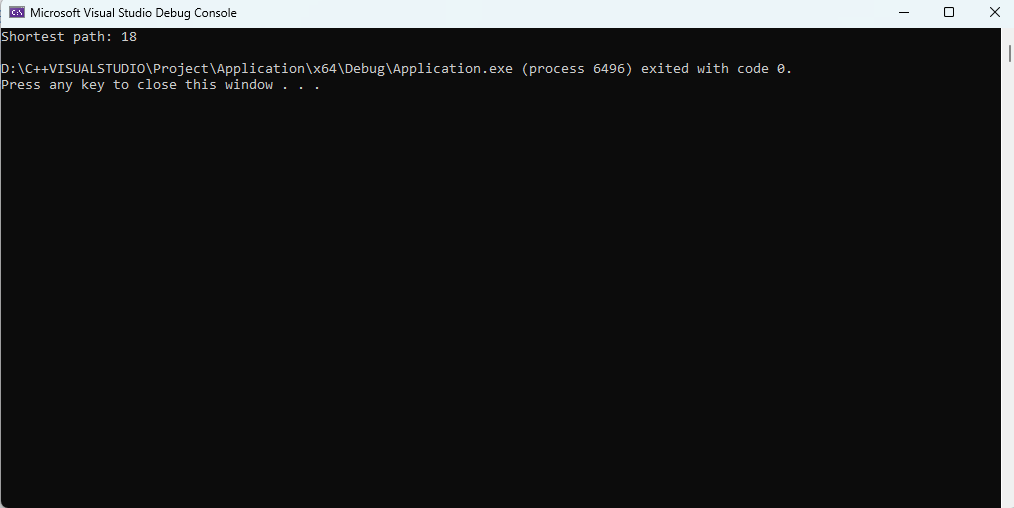
🡪Tính và xuất ra đường đi ngắn nhất của 2 node 0 và 4.

Đồ thị với input ban đầu:



Đồ thị lúc sau khi đã làm tròn các giá trị trong mảng {8.5, 15.2, 20.7, 11.5, 5.1, 6.3}





Kết quả xuất ra là đường đi ngắn nhất từ node 0 đến 4 = 8+5+5=18

**Applications và hướng phát triển:**

- (Linear programming): nếu chúng ta có một chương trình tuyến tính với số lượng biến và ràng buộc lớn, chúng ta có thể sử dụng phương pháp làm tròn hai chiều để tìm ra giải pháp khả thi gần với tối ưu.

- (Graph algorithms): làm tròn hai chiều cũng có thể được sử dụng trong các thuật toán đồ thị để giải gần đúng các bài toán như đường đi ngắn nhất hoặc cây khung nhỏ nhất. Ví dụ: nếu chúng ta đang cố gắng tìm đường đi ngắn nhất giữa hai điểm trên bản đồ, chúng ta có thể sử dụng phép làm tròn hai chiều để ước lượng khoảng cách giữa các điểm, thay vì tính toán khoảng cách chính xác.

- Numerical integration: đây là phương pháp ước lượng diện tích dưới đường cong. Bằng cách tính gần đúng đường cong bằng một loạt các đường thẳng, chúng ta có thể sử dụng phép làm tròn hai chiều để tìm giá trị gần đúng cho diện tích bên dưới đường cong.

- Data compression: đây cũng là một phương pháp để giảm kích thước của tệp dữ liệu. Ví dụ: nếu chúng tôi đang cố gắng nén một hình ảnh, chúng tôi có thể sử dụng làm tròn hai chiều để ước tính giá trị của các pixel trong hình ảnh, điều này có thể giúp giảm kích thước của tệp.

- Computer graphics: Trong đồ họa máy tính, làm tròn hai chiều có thể được sử dụng để ước tính tọa độ của các điểm trên màn hình, điều này có thể giúp cải thiện hiệu suất của các thuật toán kết xuất.Và còn rất nhiều ứng dụng và hướng phát triển khác nữa: Scientific simulations, financial applications, statistical analysis…

**Poster:**

