Bài toán vận tải:

Cơ sở:

Kho(cung): source

m: số kho hàng;

source[i] (i: 1->m) :số đơn vị hàng chứa trong kho thứ i.

Cửa hàng(cầu): receive

n: số cửa hàng;

receive[i] (i: 1->n): số đơn vị hàng hàng yêu cầu của cửa hàng thứ i.

Chi phí tương ứng: cost[m][n]:

1 ma trận kích thước nxm chứa các chi phí vận chuyển 1 đơn vị hàng từ kho bất kỳ đến cửa hàng bất kì:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cung - ->  Cầu  |  \/ | source[1] | source[2] | … | source[m] |
| receive[1] | cost[1][1] | cost[1][2] |  |  |
| receive[2] | cost[2][1] | cost[2][2] |  |  |
| … |  |  | … |  |
| receive[n] |  |  |  | cost[m][n] |

Yêu cầu:

Một ma trận x[m][n] với x[i][j] (i, j := 1-> m, n) lần lượt là số hàng cần chuyển từ source[i] đến receive[j] với chi phí cost[i][j]: phương án tối ưu nhất đảm bảo tất cả các kho đều chuyển hết hàng và tất cả các cửa hàng đều nhận đủ hàng sao cho chi phí vận chuyển là nhỏ nhất.

Giải toán:

Tây Bắc:

B1: xuất phát từ ô trên cùng bên trái, điền vào đó GTLN có thể.

B2: Dịch sang phải nếu hàng đó còn khả năng phân phối

Dịch xuống dưới nếu hàng đó hết khả năng phân phối

Điền vào ô đó GTLN có thể

B3: Tiếp tục hoàn thiện.

Ta sẽ thu được một ma trận kích thước mxn với nhiều nhất m + n – 1 ô chứa số hàng cần vận chuyển với giá vận chuyển tại ô tương ứng.

Code:

File NorthWest.c thực hiện việc nhập input các giá trị đầu vào.

File NorthWest.h là thư viện chứa hàm tính toán cho quy tắc góc tây bắc:

North\_West(int m, int a[10000], int n, int b[10000], float c[10000][10000])

m: số kho hàng,

a[]: số hàng chứa ở kho thứ j,

n: số cửa hàng,

b[]: số hàng cần ở cửa hàng thứ i,

c[][]: ma trận chi phí.

B1: khai báo ma trận x[][] là ma trận kết quả đề bài cần, làm sạch x cho tất cả phần tử trong x[][] = 0,

B2: khởi tạo i, j với giá trị ban đầu bằng 0 để duyệt từ phần tử đầu tiên trên cùng bên trái,

B3: khởi tạo 2 mảng totalrow[] và totalcol[] với tất cả phần tử trong mảng = 0. Mục đích totalrow[i] chứa tổng số đơn hàng trong hàng thứ i trong ma trận kết quả, mục đích totalcol[j] chứa tổng số đơn hàng trong cột thứ j trong ma trận kết quả. Tại 1 thời điểm trong vòng lặp while

B4: lặp:

Thực hiện gán vào 1 biến tạm:

summer = (b[i] -totalrow[i]<a[j]-totalcol[j])?b[i]-totalrow[i]:a[j]-totalcol[j]

cụm bên phải dấu bằng là 1 toán tử điều kiện so sánh giữa max cung trên cột vs max cầu trên hàng xung quanh vị trí i j vào thời điểm đó, cái nào nhỏ hơn sẽ được gán vào summer và x[i][j] sẽ được gán bằng summer.

totalrow[i] += summer để cập nhật tổng hàng trên hàng i,

totalcol[j] += summer để cập nhật tổng hàng trên cột j,

if totalrow[i] == b[i]: i++ chuyển hướng xuống ô dưới khi hàng i đã đầy,

if totalcol[j] == a[j]: j++ chuyển hướng sang ô phải khi cột j đã đầy,

tiếp tục lặp cho đến khi i, j chạm đáy.

B5: dùng 2 for lồng để in ra ma trận x là kết quả, tổng nhân lần lược theo vị trí ma trận x và ma trận chi phí i để tính tổng chi phí và in ra màn hình.

Chi phí thấp nhất:

B1: Chọn ô chi phí bé nhất, điền vào đó GTLN có thể,

B2: Trong các ô còn khả năng phân phối, chọn ô chi phí thấp nhất, điền

vào GTLN có thể,

B3: Tiếp tục hoàn thiện.

Ta sẽ thu được một ma trận kích thước nxm với nhiều nhất m + n – 1 ô chứa số hàng cần vận chuyển với giá vận chuyển tại ô tương ứng.

Code:

File Smallest.c thực hiện việc nhập input các giá trị đầu vào.

File Smallest.h là thư viện chứa hàm tính toán cho quy tắc góc tây bắc:

Smallest\_Cost(int m, int a[10000], int n, int b[10000], float c[10000][10000], int total)

m: số kho hàng,

a[]: số hàng chứa ở kho thứ j,

n: số cửa hàng,

b[]: số hàng cần ở cửa hàng thứ i,

c[][]: ma trận chi phí.

total: tổng số hàng ở kho và cửa hàng.

B1: khai báo ma trận x[][] là ma trận kết quả, y[][] là ma trận sao chép theo ma trận chi phí, làm sạch x[i][j] = 0,

B2: khởi tạo 2 biến totalcung = total và totalcau = total là tổng số hàng cần vận chuyển, điều kiện dừng của while sau khi vận chuyển xong, khởi tạo 2 mảng totalrow[] và totalcol[] với cùng mục đích như bài quy tắc tây bắc,

B3: Lặp với điều kiện :totalcung > 0 && totalcau > 0 (đến khi hết hàng)

Lồng 2 for biến tạm float min = (1 số lớn) để so sánh và tìm chi phí thấp nhất trên ma trận c[][], gán vị trí của GLNN đó vào i j, gán vào y[i][j](sao chép của ma trận c[][]) 1 giá trị lớn (mục đích loại bỏ và tránh dùng lại ô đó 1 lần nữa {phép so sánh còn 1 điều kiện là y[I][J] khác số lớn đó})

Gán vào 1 biến tạm:

summer=(b[i]-totalrow[i]<a[j]-totalcol[j])?b[i]-totalrow[i]:a[j]-totalcol[j];

phép gán trên y hệt với bài góc tây bắc trên.

totalrow[i] += summer để cập nhật tổng hàng trên hàng i,

totalcol[j] += summer để cập nhật tổng hàng trên cột j, giống như tây bắc

totalcung -= summer để cập nhật tổng hàng cung còn lại,

totalcau -= summer để cập nhật tổng cung còn lại

tiếp tục lặp cho đến khi totalcung và totalcau hết.

B4: dùng 2 for lồng để in ra ma trận x là kết quả, tổng nhân lần lược theo vị trí ma trận x và ma trận chi phí i để tính tổng chi phí và in ra màn hình.