**Đồ án**

**Giải thuật và Lập trình**

**Hungarian Method  
Maximum-weighted Bipartite Matching**

Giảng viên hướng dẫn: Phạm Minh Tuấn

Sinh viên thực hiện: Nguyễn Đức Tuệ Anh

Phan Minh Tuấn Anh

1. **Dẫn nhập:**

Vấn đề chuyển nhượng, bài toán phân công, bài toán ghép cặp....là những tên gọi khác nhau của “Assignment Problem”. “Assignment Problem” là một trong những vấn đề tối ưu hóa tổ hợp (combinatorial optimization problems) đầu tiên được nghiên cứu. Ta đi qua một chút về lịch sử của nó:

* Nó được khám phá ra bởi G.Monge vào năm 1784: Monge xét một vấn đề về chuyển đất : Có hai khu vực có cùng diện tích, một khu vực đầy đất, một khu vực thì rỗng không. Câu hỏi được đặt ra là làm thế nào để di chuyển lượng đất từ khu vực có đất đến khu vực còn lại, sao cho tổng khoảng cách vận chuyển là nhỏ nhất.
* Theo định nghĩa toán học thì bài toán phân công được phát biểu như sau:

Cho ma trận chi phí , )

tìm một hoán vị của 1,…, sao cho:

là nhỏ nhất có thể.

* Ta có thể diễn dịch lại:

Bài toán phân công người-việc:

* + - C là ma trận 𝑛×𝑛 thể hiện chi phí n công nhân thực hiện n việc.
    - Yêu cầu phân n công việc cho n công nhân với tổng chi phí thấp nhất.
    - Mỗi công nhân một việc, mỗi việc một công nhân.
* Vào năm 1915 Denes Konig nhận ra rằng tìm kiếm bộ ghép cực đại trên đồ thị hai phía không trọng số có thể xem như là một trường hợp đặc biệt của bài toán phân công
* Sau công bố của Konig tại Hội Toán học và Vật lý Budapest ngày 26 tháng 3 năm 1931, E.Egervary tìm ra phiên bản có trọng số của lý thuyết mà Konig đề ra. Nó mô tả bộ ghép trọng số cực đại trên đồ thì hai phía, và điều này áp dụng vào bài toán phân công.
* Trải qua một khoảng thời gian, việc đưa độ phức tạp khi giải bài toán phân công về độ phức tạp đa thức gặp rất nhiều khó khăn. Harold W. Kuhn đã nhận xét, với bài toán 10x10 mà áp dụng The simplex method thì không có máy nào trên thế giới có thể giải được cả. Để rồi dựa trên lý thuyết của Egervary, Kuhn đã giới thiệu thuật toán Hungarian vào năm 1955. Sau đó năm 1957, Munkres đã chỉ ra rằng thuật toán có thể chạy với độ phức tạp đa thức là ).
* Ford và Fulkerson đã báo cáo về thí nghiệm tính toán dựa trên thuật toán Hungarian:

“Ví dụ tối đa mà chúng tôi thử nghiệm là bài toán 20x20. Với ví dụ này thì Simplex Method cần hơn một tiếng đồng hồ, trong khi thuật toán này (Hungarian Method) chỉ cần khoảng 30 phút khi tính bằng tay”

1. **Ý tưởng:**

Như trong phần dẫn nhập đã nói thì thuật toán Hungarian xuất phát từ những lý thuyết được nghiên cứu từ hai nhà toán học người Hungary là Konig và Egervary. Bài toán phân công mà Kuhn xét ở đây thay vì ma trận C biểu diễn chi phí thì lại là biểu diễn đánh giá chất lượng của người i làm công việc j. Và yêu cầu cần tìm là phân công sao cho tổng đánh giá là lớn nhất.

Những ý chính từ hai nhà toán học mà Kuhn lấy làm tiền đề để đi đến thuật toán này là:

Theo Konig, vấn đề ghép cặp trên đồ thị hai phía (không trọng số) giữa hai tập n đỉnh giống hoàn toàn với bài toán phân công n x n với ma trận chi phí chỉ chứa 0 hoặc 1. Tức là với đồ thị hai phía G = (X,Y,E) với , ta biểu diễn n người là n đỉnh thuộc tập X, n việc là n đỉnh thuộc tập Y, và:

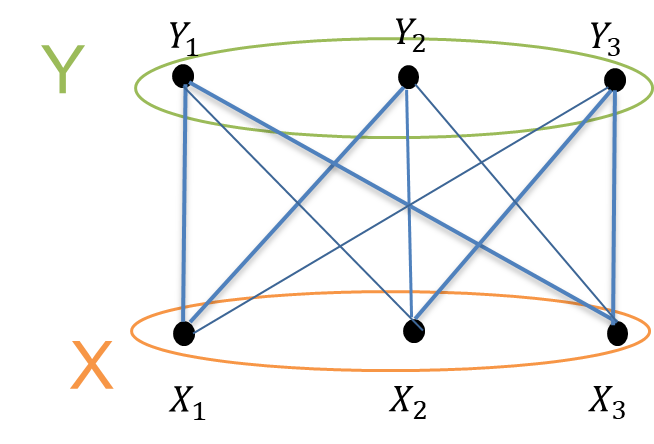
Ngoài ra ma trận C = () với , số lượng tối đa số 1 có thể được chọn sao cho không có hai số 1 nào cùng trên một đường( ngang hoặc dọc) bằng với số lượng tối thiểu đường chứa toàn bộ các số 1. Phát biểu tương đương dưới dạng đồ thị là: độ lớn của bộ ghép cực đại trong đồ thị hai phía bằng với số đỉnh tối thiểu để che phủ toàn bộ các cạnh.

Còn Egervary thì đưa ra lí thuyết với trường hợp đồ thị có trọng số: Cho hàm , M là bộ ghép cần tìm thì nếu M là bộ ghép cực đại thì w(M) = với , tức là với mỗi cặp (x,y) trong M thì

1. **Một số định nghĩa, định lý:**

Đồ thị 2 phía: Là đồ thị mà tập các đỉnh V có thể chia làm 2 tập con rời nhau là X và Y nghĩa là: Với mọi cạnh e = (a,b) thì a thuộc X và b thuộc Y (mỗi cạnh bắt đầu ở một nút của tập này và kết thúc ở một nút ở tập còn lại).

Đồ thị 2 phía đầy đủ (Complete Bipartite Graph): Là đồ thị hai phía mà giữa một đỉnh bất kỳ của X và một đỉnh bất kỳ của Y đều có cạnh nối.

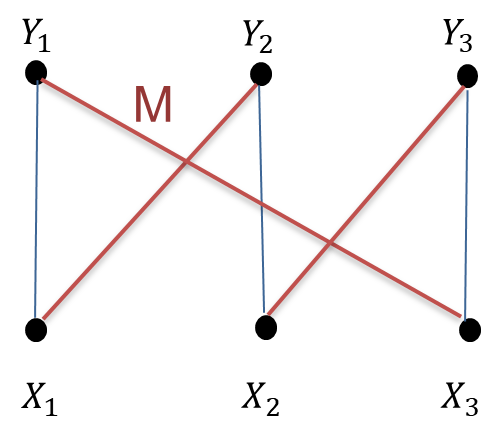


Bộ ghép (Matching): Một bộ ghép (matching) của đồ thị G là một tập hợp các cạnh của G đôi một không có đỉnh chung.

Kích thước của bộ ghép (kí hiệu |M|) là số lượng cạnh trong bộ ghép

Bộ ghép cực đại (Maximum Matching): Bộ ghép M gọi là cực đại nếu với mọi bộ ghép M′ khác M thì |M′| ≤ |M|.

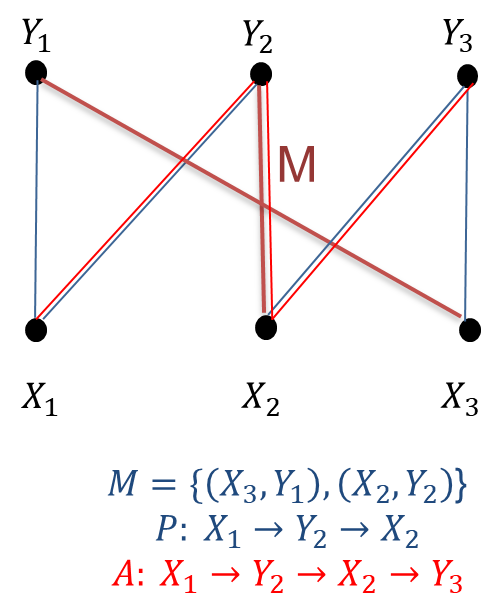
Bộ ghép hoàn hảo (Perfect Matching): Bộ ghép gọi là hoàn hảo nếu mọi đỉnh của đồ thị đều được ghép.



, },

Đường xen kẽ (Alternating path) của bộ ghép M: Là một đường đi trên đồ thị G=(V,E), bắt đầu tại đỉnh thuộc X chưa được ghép và chứa xen kẽ những cạnh thuộc E\M và M.

Đường tăng cường (Augmenting path) của bộ ghép M: Một đường xen kẽ P kết thúc tại 1 đỉnh chưa được ghép thuộc Y được gọi là “đường tăng cường” bởi vì chúng ta có thể sử dụng nó để mở rộng kích thước của M.



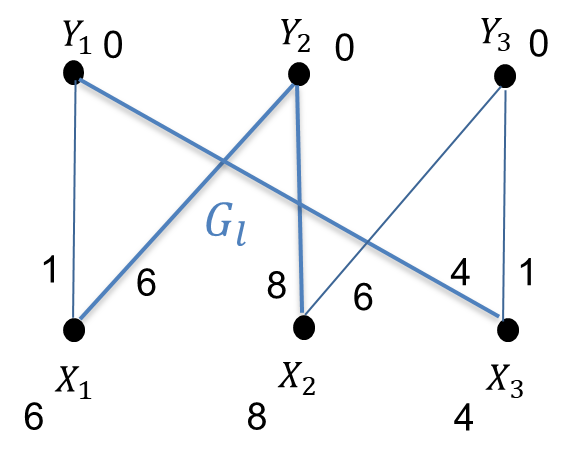
Nhãn khả thi (Feasible Labeling):Ta gán nhãn cho mỗi đỉnh, ℓ : V → R

Cách dán nhãn dán gọi là khả thi nếu thỏa mãn:

ℓ(x) + ℓ(y) ≥ w(x, y), ∀x ∈ X, y ∈ Y

Đồ thị cân bằng (Equality Graph): là đồ thị cân bằng ứng với cách dãn nhãn ℓ nếu thỏa mãn:

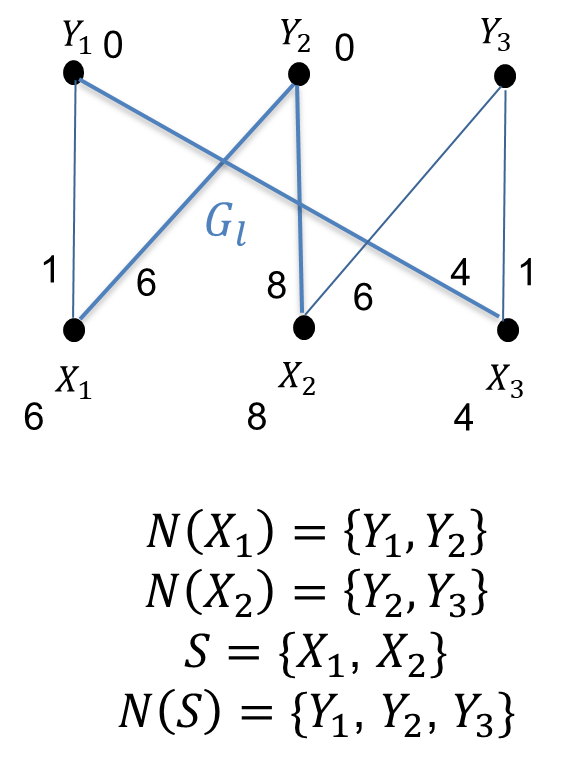
= {(x, y) : ℓ(x) + ℓ(y) = w(x, y)}



Lân cận của đỉnh u ∈ V: Là tập các đỉnh chung cạnh với u:

= { v : (u, v) ∈ E}

Lân cận của tập S ⊆ V: Là tập các đỉnh chung cạnh với những đỉnh thuộc S:



Định lí về đồ thị cân bằng và bộ ghép trọng số cực đại:

* + Nếu ℓ là cách dán nhãn khả thi và M là bộ ghép hoàn hảo trên  thì M là bộ ghép trọng số cực đại trên G

-> Chứng minh:

M’ là bộ ghép hoàn hảo bất kì trên đồ thị G (không nhất thiết là phải trên ), ta có:

M là bộ ghép hoàn hảo trên :

1. **Thuật toán:**

* Bước 0: Khởi tạo nhãn dán rồi tìm bộ ghép cực đại M trên .
* Bước 1: Nếu M là perfect thì kết thúc. Ngược lại lấy một đỉnh tự do u ∈ X. Gán S = {u}, T = ∅ .
* Bước 2: Nếu (S) = T, tìm

Cập nhập nhãn dán

Thay thế bởi ’, lặp lại bước 2 cho đến khi (S) != T thì chuyển sáng bước 3.

* Bước 3: Lấy đỉnh y ∈ (S) \ T.
  + Nếu y tự do ta được đường mở từ u đến y. Mở rộng bộ ghép M theo đường tăng cường rồi trở lại bước 1.
  + Nếu y đã được ghép trong M, thì ta thêm các đỉnh z nối với y vào: S = S ∪ {z}, T = T ∪ {y} . Trở lại bước 2.

1. **Cài đặt**

Khai báo biến:

|  |
| --- |
| **int** INF = 10000000;  **int** cost[][];  **int** n, max\_match;  **int** lx[], ly[];  **int** xy[];  **int** yx[];  **boolean** S[], T[];  **int** slack[];  **int** slackx[];  **int** prv[]; |

Bước 0:

|  |
| --- |
| lx = **new** **int**[n];  ly = **new** **int**[n];  xy = **new** **int**[n];  yx = **new** **int**[n];  S = **new** **boolean**[n];  T = **new** **boolean**[n];  slack = **new** **int**[n];  slackx = **new** **int**[n];  prv = **new** **int**[n];  Arrays.*fill*(lx, 0);  Arrays.*fill*(ly, 0);  **for** (**int** x = 0; x < n; x++)  **for** (**int** y = 0; y < n; y++)  lx[x] = Math.*max*(lx[x], cost[x][y]); |

Bước 1:

|  |
| --- |
| **void** augment() {  **if** (max\_match == n)  **return**;  **int** x, y, root = 0;  **int** q[], wr = 0, rd = 0;  q = **new** **int**[n];  Arrays.*fill*(S, **false**);  Arrays.*fill*(T, **false**);  Arrays.*fill*(prv, -1);  **for** (x = 0; x < n; x++)  **if** (xy[x] == -1) {  q[wr++] = root = x;  prv[x] = -2;  S[x] = **true**;  **break**;  }  **for** (y = 0; y < n; y++) {  slack[y] = lx[root] + ly[y] - cost[root][y];  slackx[y] = root;  } |

Bước 2:

|  |
| --- |
| **void** update\_labels() {  **int** x, y, delta = INF;  **for** (y = 0; y < n; y++)  **if** (!T[y])  delta = Math.*min*(delta, slack[y]);  **for** (x = 0; x < n; x++)  **if** (S[x])  lx[x] -= delta;  **for** (y = 0; y < n; y++)  **if** (T[y])  ly[y] += delta;  **for** (y = 0; y < n; y++)  **if** (!T[y])  slack[y] -= delta;  } |

Bước 3

|  |
| --- |
| **void** add\_to\_tree(**int** x, **int** prevx)  {  S[x] = **true**;  prv[x] = prevx;  **for** (**int** y = 0; y < n; y++)  **if** (lx[x] + ly[y] - cost[x][y] < slack[y]) {  slack[y] = lx[x] + ly[y] - cost[x][y];  slackx[y] = x;  }  } |

|  |
| --- |
| **while** (**true**) {  **while** (rd < wr) {  x = q[rd++];  **for** (y = 0; y < n; y++)  **if** (cost[x][y] == lx[x] + ly[y] && !T[y]) {  **if** (yx[y] == -1)  **break**;  T[y] = **true**;  q[wr++] = yx[y];  add\_to\_tree(yx[y], x);  }  **if** (y < n)  **break**;  }  **if** (y < n)  **break**;  update\_labels();  wr = rd = 0;  **for** (y = 0; y < n; y++)    **if** (!T[y] && slack[y] == 0) {  **if** (yx[y] == -1) {  x = slackx[y];  **break**;  } **else** {  T[y] = **true**;  **if** (!S[yx[y]]) {  q[wr++] = yx[y];  add\_to\_tree(yx[y], slackx[y]);  }  }  }  **if** (y < n)  **break**;  }  **if** (y < n) {  max\_match++;  **for** (**int** cx = x, cy = y, ty; cx != -2; cx = prv[cx], cy = ty) {  ty = xy[cx];  yx[cy] = cx;  xy[cx] = cy;  }  augment();  } |

Chương trình chính:

|  |
| --- |
| **public** **int** hungarian() {  **int** ret = 0;  max\_match = 0;  init\_labels();  Arrays.*fill*(xy, -1);  Arrays.*fill*(yx, -1);  augment();  **for** (**int** x = 0; x < n; x++)  ret += cost[x][xy[x]];  **return** ret;  } |

1. **Bài tập**
   1. Salesmen problem – HackerEarth: Ford-Fulkerson+Hungarian Method
   2. Selfish Cities: Hungarian Method m x n
   3. Productive productivity – HackerEarth: Hungarian Method nxn
   4. MATCH1-SPOJ: Hungarian Method in Bipartite Graph no weighted
   5. MATCH2 – SPOJ: Hungarian Method nxn
   6. Manager Miser-Hacker Earth: Hungarian Method nxn
   7. Engaging with Loyal Customers-[The 2017 ACM - ICPC Asia Ho Chi Minh City Regional Contest](https://open.kattis.com/problem-sources/The%202017%20ACM%20-%20ICPC%20Asia%20Ho%20Chi%20Minh%20City%20Regional%20Contest): Hungarian Method m x n
   8. ChessMatchup – topcoder: Hungarian Method nxn
   9. Bruce and the Chocolates – HackerEarth: GCD + Hungarian Method nxn
   10. Bài toán phân công – Lê Minh Hoàng: Hungarian Method mxn
2. **Tài liệu tham khảo:**
   1. Lý thuyết đồ thị - Lê Minh Hoàng.
   2. CMSC 651: Design and Analysis of Algorithms.
   3. EGRES Technical Report No. 2004-14 On Kuhn’s Hungarian Method – A tribute from Hungary.
   4. H. W. Kuhn, “The hungarian method for the assignment problem,” Naval Research Logistic Quarterly, 1955.
   5. On the history of combinatorial optimization(till 1960)-Alexander Schrijver.
   6. Bipartite Matching & the Hungarian Method-Subhash Suri.