Đại Học Quốc Gia Thành Phố Hồ Chí Minh
Trường đại học Bách Khoa
Khoa Khoa Học và Kỹ Thuật Máy Tính

Cấu trúc rời rạc cho Khoa học Máy tính (CO1007)

Assignment

THUẬT TOÁN ĐƯỜNG ĐI NGẮN NHẤT

Giảng viên: Nguyễn Văn Minh Mẫn, Mahidol University
Nguyễn An Khương, CSE-HCMUT
Trần Tuấn Anh, CSE-HCMUT
Nguyễn Tiến Thịnh, CSE-HCMUT
Trần Hồng Tài, CSE-HCMUT
Mai Xuân Toàn, CSE-HCMUT

Sinh Viên: Trương Thiên Ân - 2310190

Lớp: L01

Ho Chi Minh City, October 2024

Mục lục

1 Đề Bài

Bài toán Travelling Salesman Problem (được ghi tắt là TSP) yêu cầu tìm lộ trình ngắn nhất cho một người bán hàng đi qua tất cả các thành phố trong danh sách, mỗi thành phố được ghé qua đúng một lần, và sau đó quay trở lại thành phố xuất phát.

- **Đầu vào:**
  - Danh sách các thành phố (được đánh theo ký tự bảng chữ cái từ A-Z)
  - Khoảng cách giữa từng cặp thành phố (được biểu diễn dưới dạng ma trận khoảng cách hoặc đồ thị có trọng số).
- **Đầu ra:** Một lộ trình có tổng chiều dài nhỏ nhất, thỏa mãn:
  - Người bán hàng đi qua tất cả các thành phố một lần.
  - Quay về thành phố xuất phát.

2 Giải thuật

Giải thuật mà em chọn cho bài toán TSP là Dynamic Programming (Quy hoạch động) kết hợp với bit masking để đánh dấu các đỉnh đã đi qua. Thuật toán này có thể hoạt động khá tốt với các số lượng đỉnh đến trung bình (20-25 đỉnh). Việc sử dụng Dynamic progamming kết hợp bit masking là để chia bài toán thành các bài toán lần lượt nhỏ hơn, tính toán và lưu trữ kết quả để tái sử dụng. Em sẽ mô tả thuật toán này thông qua một bài toán TSP đơn giản với 4 đỉnh và xuất phát từ điểm A.

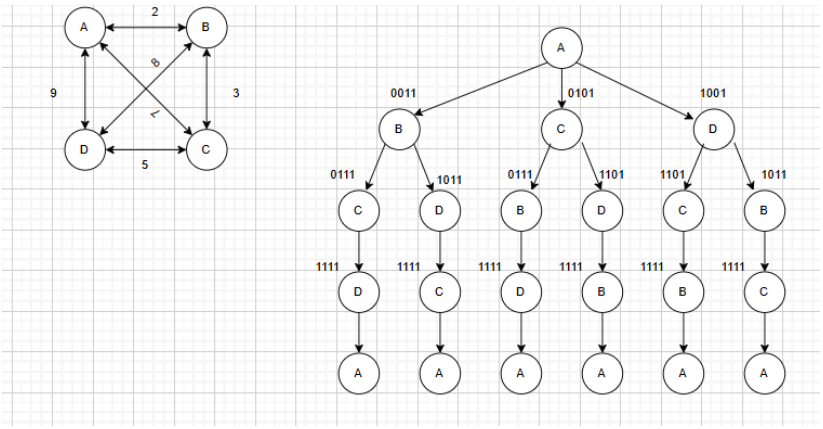

Figure 1: Đồ thị và tát cả các đường đi có thể có khi đi từ đỉnh A

Ta sử dụng bitmask để đánh dấu các đỉnh đã đi qua, có nghĩa là nếu ta đã thăm qua đỉnh đó rồi thì bit của đỉnh đó là bằng 1. Ví dụ bitmask khởi đầu là 0001 có nghĩa là đỉnh A đã được thăm, khi đi từ A qua B thì bitmask sẽ trở thành 0011 và tương tự với các đỉnh khác. Khi đạt trạng thái là 1111 nghĩa là tất cả các đỉnh đã được thăm hết. kết thúc một chu trình. Ta sẽ sử dụng một mảng 2 chiều để lưu các giá trị đã tính toán để có thể sử dụng lại với cách trị số là [next_bitmask][current_node] và giá trị của memo[next_mask][current_node] là

Ví dụ:

- ta có memo[0011][A] là vị trí từ A đế B (khoảng cách từ A đến B)
- ta có memo [0101][A] là vị trí từ A đến C (khoảng cách từ A đến C)
- ta có memo[0111][B] là vị trí của B đến C (Trường hợp đi từ A->B, vậy mảng này sẽ lưu giá trị quảng đường từ A->B->C)
- ta có memo[0111][C] là vị trí từ C đến B (Trường hợp đi từ A-> C, vậy mảng này sẽ lưu giá trị quảng đường từ A->C->B)

Ta thực hiện tính toán chi phí nhỏ nhất để lưu vào mảng memo theo công thức min_cost= min(min_cost (giá trị min trước đó), memo[next_bitmask][current_node] + G[current][next_city]) và lưu vị trí trước đó để tiến hành truy hồi lại khi đã đi qua hết các đỉnh.

3 Code

3.1 Mô tả

Ở phần code em chia thành 3 phần là phần hàm đệ quy **DP** để tính toán và lưu trữ giá trị **prevVertex** là các đỉnh của đường đi tối ưu. Hàm **Travelling** để trả về chuỗi các đường đi. Và cuối cùng là hàm **rewriteMatrix** để ghi lại ma trận để dễ xử lý số liệu hơn.

- **Hàm DP** nhận các tham số đầu vào:
  - int G[30][30] (ma trận trọng số)
  - int start (index của vị trí bắt đầu)
  - int current (index của vị trí hiện tại)
  - int mask (là giá trị các đỉnh được đánh dấu là đã đi qua hay chưa)
  - int n (là số đỉnh)
  - vector<vector<int>>& PrevVertex (là truyền tham chiếu mảng vector 2 chiều chứa các giá trị đỉnh của đường đi tối ưu)
  - vector<vector<int>>& minCost (là truyền tham chiếu mảng vector 2 chiều chứa khoảng cách của đường đi, tránh phải tính toán lại nhiều lần)
- **Hàm Travelling** nhận các tham số đầu vào:
  - int G (ma trận trọng số)
  - int n (số đỉnh)
  - char start (kí tự đỉnh bắt đầu)
- **Hàm rewriteMatrix** nhận các tham số đầu vào:
  - int G (ma trận trọng số)
  - int n (số đỉnh)

3.2 Hàm rewriteMatrix

```
const int INF = numeric_limits<int>::max();
void rewriteMatrix (int G[30][30], int n){
    for (int i=0; i<n; i++){
        for (int j =0; j< n; j++){
            if (i != j && G[i][j] == 0){
                G[i][j] = INF/2;
            }
        }
    }
}
```

Dòng đầu tiên là khai báo một hằng số int là INF là giá trị lớn nhất của biến số nguyên. Trong ma trận trọng số đề cho ngoài các giá trị khoảng cách từ 1 đỉnh đến chính nó là 0, thì còn tồn tại những khoảng cách từ 1 đỉnh tới 1 đỉnh khác bằng 0 để biểu diễn là không có đường đi từ đỉnh đó đến đỉnh khác. Ta tiến hành phân biệt các đỉnh ấy bằng cách gán INF/2 cho các khoảng cách các cặp đỉnh không đi đến được.

Giải thích tại sao em dùng INF/2 mà không phải là INF: vì để an toàn hơn khi chẳng may dùng INF mà chương trình cộng thêm vào, sẽ gây ra lỗi.

3.3 Hàm DP

```
1  void DP (int G[30][30], int start, int current, int mask, int n,
2          vector<vector<int>>& prevVertex, vector<vector<int>>& minCost){
3
4    int is_Visited_all = (1<<n)-1;
5
6    if (mask == is_Visited_all){
7      minCost[mask][current] = G[current][start];
8      return;
9    }
10   if (minCost[mask][current] != -1){
11       return;
12   }
13   int min_cost = INF/2;
14   for (int i = 0; i< n; i++){
15     if ((mask&(1<<i))== 0 && G[current][i] > 0 && G[current][i] != INF/2){
16         int next_mask = mask|(1<<i);
17         DP(G, start, i, next_mask, n, prevVertex, minCost);
18         int predict_min_cost = G[current][i] + minCost[next_mask][i];
19         if (predict_min_cost < min_cost){
20             min_cost= predict_min_cost;
21             prevVertex[mask][current]= i;
22         }
23     }
24   }
25   minCost[mask][current] = min_cost;
26 }
```

Hàm này là phần chính của bài dùng để lưu các đỉnh của đường đi tối ưu.
Giải thích code:

1) `int is_Visited_all = (1<<n)-1;`:
Khai báo biến is_Visisted_all là biến biểu diễn tất cả các bit đã được đi qua. Ví dụ với 4 đỉnh lần lượt là A, B, C, D thì trạng thái cuối cùng cần đạt tới là 1111_2.

2) từ dòng 6 tới dòng 8 là điều kiện dừng của hàm đệ quy. Khi đã thăm hết tất cả các đỉnh thì ta lưu giá trị từ đỉnh cuối cùng vừa đến đó, đi đến đỉnh bắt đầu. Bắt đầu quá trình truy hồi giá trị của hàm hồi quy.

3) Từ dòng 10 đến 12 là một điều kiện để kiểm tra thử xem giá trị minCost đó đã được tính hay chưa. Nếu được tính rồi thì trả về giá trị đó, giúp giảm thiểu quá trình phải tính đi tính lại nhiều lần.

4) dòng 13 khai báo giá trị min_cost là rất lớn để luôn chắc rằng lần đầu được so sánh thì min_cost sẽ được cập nhật.

5) Dòng 13 đến 15 là vòng lặp chính để giải bài toán. Ta kiểm tra điều kiện để đi qua các đỉnh còn lại. Ví dụ `((mask&(1<<i))== 0 && G[current][i] > 0 && G[current][i] != INF/2)` là kiểm tra thử xem đỉnh i (index của đỉnh) đã được đi qua hay chưa ở điều kiện 1, điều kiện 2 là kiểm tra nếu vị trí từ điểm hiện tại đến đỉnh thứ i có dương hay không và điều kiện cuối cùng để kiểm tra liệu khoảng cách từ đỉnh hiện tại tới đỉnh thứ i là có hợp lệ hay không.

6) Dòng 16 đến 24 dùng để gọi hàm đệ quy để tính giá trị predict_min_cost và so sánh với giá trị min_cost từ đó chọn ra đường đi thích hợp. Ví dụ khi mà truy hồi kết quả lên tới mask lần lượt là 0111 và 1011 với đỉnh xuất phát là A thì ta có thể nhận thấy rằng đây là trường hợp đỉnh A qua đỉnh B và đi qua 1 trong 2 đỉnh C hoặc D, giờ ta cần kiểm tra xem khoảng cách của đỉnh nào đến B là ngắn hơn thì ta cập nhật giá trị prevVertex.

7) `minCost[mask][current] = min_cost;` để đảm bảo là giá trị minCost sẽ luôn được cập nhật sau mỗi lần gọi hàm.

3.4 Hàm Travelling

```
1  string Traveling(int G[30][30], int N, char start) {
2
3      rewriteMatrix (G, N);
4      int start_index = start -'A';
5
6      vector<vector<int>> minCost ((1<<N), vector<int>(N, -1));
7      vector<vector<int>> prevVertex ((1<<N), vector<int>(N, -1));
8
9      int mask = (1<<(start_index));
10     DP(G, start_index, start_index, mask, N, prevVertex, minCost);
11     string result;
12
13     int visitedMask = mask;
14     result += start;
15
16     for (int i = 0; i < N - 1; i++) {
17         int next = prevVertex[visitedMask][start_index];
18         result += " ";
19         result += char(next + 'A');
20         visitedMask |= (1 << next);
21         start_index = next;
22     }
23     return result + " " + start;
24 }
```

Hàm này dùng để ghi ra quãng đường tối ưu nhất để đi.
Giải thích code:

1) Dòng 3 gọi hàm rewriteMatrix để tiến hành ghi lại ma trận G.

2) Dòng 4 dùng để chuyển đổi 1 kí tự trong bảng mã ASCII thành một số nguyên để tính toán.

3) Dòng 6 và 7 lầ khai báo 2 vector 2 chiều và khởi tạo các giá trị của chúng là -1.

4) Dòng 9 là để khởi tạo mask ban đầu. Ví dụ nếu đỉnh xuất phát là B, ta có start_index sẽ bằng B -A sẽ bằng 1 (vì trong mã ASCII B có giá trị 66 còn A là 65). Nên mask sẽ là 0010 (trường hợp 4 đỉnh).

5) Dòng 10 ta sẽ gọi hàm DP để có giá trị mảng **prevVertex**

6) Từ dòng 11 đến cuối là để ghi chuỗi các đường đi vào result để trả ra kết quả đường đi. ta sẽ có đỉnh đầu tiên của chuỗi result là đỉnh bắt đầu (start), trong vòng for ta thực hiện đi qua hết tất cả các đỉnh để truy lại đường đi.

   Ví dụ ta có chuỗi đường đi là

   - prevVertex[0001][A] = B
   - prevVertex[0011][B] = C
   - prevVertex[0111][C] = D

   ta sẽ có vòng lặp là

i =0: + start_index là 0 tương đương với char A
+ next = prevVertex[0001][A] = B
+ chuỗi result hiện tại : A B
+ vistedMark cập nhật đỉnh B vào trong mask nên từ 0001 sau lệnh này thành 0011
+ start_index đổi từ A thành B

i =1: + start_index là B
+ next = prevVertex[0011][B] = C
+ chuỗi result hiện tại : A B C
+ vistedMark cập nhật đỉnh B vào trong mask nên từ 0011 sau lệnh này thành 0111
+ start_index đổi từ B thành C

i =2: + start_index C
+ next = prevVertex[0111][C] = D
+ chuỗi result hiện tại : A B C D
+ vistedMark cập nhật đỉnh B vào trong mask nên từ 0111 sau lệnh này thành 1111
∗ start_index đổi từ C thành D

7) Dòng cuối return giá trị result thêm vào đỉnh bắt đầu để xong 1 chu trình. Lúc này result = A B C D + " " + start = A B C D A .

4 Tài liệu tham khảo

1. Slide bài giảng môn Cấu trúc rời rạc cho Khoa Học Máy tính của các giảng viên.
2. Kenneth H Rosen, *Discrete Mathematics and ITs Application, 2019*