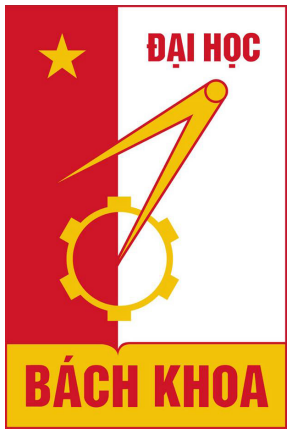
Thuật toán ứng dụng
Bài thực hành số 2: Tìm kiếm vét cạn

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương



Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 6 tháng 4 năm 2021

Mục lục

Mục lục

Cách thực thi chương trình

- Sử dụng vòng lặp, i.e *for loop*, *while loop*,...
- Sử dụng hàm đệ quy
- ...

Mô hình giải bài

- **Tìm kiếm vét cạn:** Thử mọi cấu hình có thể của lời giải và chọn ra cấu hình phù hợp nhất.
- Tham lam
- Chia để trị
- Quy hoạch động
- ...

Tìm kiếm vét cạn

- Liệt kê lời giải
  - Trực tiếp sinh ra cấu hình đầy đủ của lời giải từ cấu hình trước đó.
  - Thường được thực hiện bằng vòng lặp.
  - Ví dụ: Bài toán cái túi (liệt kê các dãy nhị phân độ dài n), Bài toán người du lịch (liệt kê các hoán vị của n số tự nhiên đầu tiên),...
- Quay lui
  - Cấu hình lời giải được sinh từng phần, thường có trình tự. Nếu không thể tiếp tục sinh lời giải thì quay về bước trước (quay lui).
  - Thường thực hiện bằng hàm đệ quy.
  - Có thể tối ưu bằng kỹ thuật nhánh và cận.
- ...

Chú ý

- Tìm kiếm vét cạn $\neq$ Đệ quy $\neq$ Quay lui $\neq$ Nhánh và cận

Đệ quy quay lui - cách 1

```
void _try(x) {
// x la phan cau hinh loi giai can duoc bo sung
    if (x thuoc neo de quy) {
        Xu ly va thoat ham de quy;
    }
    Duyet tung kha nang (i) {
        Neu kiem_tra(x, i) {
        // Kiem tra dieu kien va can
            Luu trang thai hien tai;
            Cap nhat trang thai;
            _try(x_);
            // x_ la phan cau hinh tiep theo
            Khoi phuc trang thai hien tai; // Quay lui
        }
    }
}
```

Đệ quy quay lui - cách 2

```
void _try(x, s) {
// x la phan cau hinh loi giai can duoc bo sung
// s la trang thai hien tai
    if (x thuoc neo de quy) {
        Xu ly va thoat ham de quy;
    }
    Duyet tung kha nang (i) {
        Neu kiem_tra(x, s, i) {
        // Kiem tra dieu kien va can
            _try(x_, s_);
            // x_ la phan cau hinh tiep theo
            // s_ la trang thai sau khi gan i cho x
        }
    }
}
```

Mục lục

Đề bài

- Có K xe tải giống hệt nhau, mỗi xe chứa tối đa Q gói hàng.
- Có n khách hàng, khách hàng i yêu cầu $d[i]$ gói hàng.
- Mỗi khách hàng được phục vụ bởi đúng 1 xe.
- Mỗi xe phục vụ tối thiểu 1 khách hàng.
- **Yêu cầu:** Đếm số cách sắp xếp lộ trình các xe để phục vụ các khách hàng theo yêu cầu.

Thuật toán

- Gọi $x[i]$ là xe tải phục vụ khách hàng i, $x[i] \in [0..K-1]$.
- Thử mọi trường hợp của $x[i], \forall i \in [0..n-1]$ và kiểm tra các ràng buộc.
- Với mỗi cách chọn $x[i], \forall i \in [0..n-1]$ hợp lệ, gọi $c[k]$ là số khách hàng xe thứ k phục vụ, ta có thêm $\prod_{k=0}^{K-1} c[k]!$ cách xếp lộ trình cho xe.
- Vì các xe tải giống hệt nhau, cần chia số kết quả cho $k!$.

Code

```
void _try(int i) {
    if (i == n) {
        int tmp = 1;
        for (int k = 0; k < K; k++) {
            tmp *= custom_factorial(c[k]);
        }
        res += tmp;
        return;
    }
    for (int k = 0; k < K; k++) {
        if (q[k] >= d[i]) {
            q[k] -= d[i]; //init: q[k] = Q
            c[k]++; //init: c[k] = 0
            _try(i+1);
            q[k] += d[i];
            c[k]--;
        }
    }
}
```

Mục lục

Đề bài

- Có n môn học và m giáo viên, mỗi giáo viên chỉ có thể dạy các môn trong danh sách cho trước.
- Các môn học trùng giờ không thể được dạy bởi cùng một giáo viên.
- Tải của một giáo viên là số khóa học giáo viên phải dạy.
- **Yêu cầu:** Tìm cách xếp lịch để tải của giáo viên có tải cao nhất là nhỏ nhất.

Thuật toán

- Gọi $x[i]$ là giáo viên sẽ dạy môn i.
- Thử mọi trường hợp của $x[i], i \in [0..n]$ và kiểm tra các ràng buộc cũng như tải của giáo viên.
- Kết hợp nhánh cận:
  - Nếu tải lớn nhất hiện tại lớn hơn hoặc bằng lời giải tốt nhất, ta cắt nhánh.

Code

```
void _try(int x) {
    if (x == n) {
        res = min(res, curLoad);
        return;
    }
    for (int i = 0; i < m; i++) {
        if (check(i, x)) {
            int oldLoad = curLoad;
            t[i][x] = 1;  //Giao vien i day mon x
            l[i]++;  // Tai cua giao vien i
            curLoad = max(curLoad, l[i]);
            _try(x+1);
            t[i][x] = 0;
            l[i]--;
            curLoad = oldLoad;
        }
    }
}
```

Code

```
bool check(int i, int x) {
    if ((a[i][x] == 0) || (l[i] + 1 >= res)) {
        return false;
    }
    for (int j = 0; j < o[x].size(); j++) {
        if ((t[i][o[x][j]] == 1)) {
            return false;
        }
    }
    return true;
}
```

Mục lục

Đề bài

- Cho một đồ thị đầy đủ có trọng số.
- **Yêu cầu:** Tìm một cách di chuyển qua mỗi đỉnh đúng một lần và quay về đỉnh xuất phát sao cho tổng trọng số các cạnh đi qua là nhỏ nhất (chu trình hamilton nhỏ nhất).

Thuật toán 1

- Mỗi cách di chuyển ứng với một hoán vị của n đỉnh.
- Xét hết các hoán vị và tìm nghiệm tốt nhất.
- Để xét hết các hoán vị, có thể dùng vòng lặp kết hợp thuật toán sinh kế tiếp hoặc đệ quy - quay lui.

Code 1a

```
int calc(int a[], int c[][]) {
    cost = 0;
    for (int i = 0; i < n; i++) {
        cost += c[a[i]][a[(i + 1) % n]];
    }
    return cost;
}

int solve(n, c) {
    answer = INF;
    a = {1, 2,... n};
    while (1) {
        answer = min(answer, calc(a, c));
        if (a == {n, n - 1, ..., 1}) {
            break;
        }
        a = next_permutation(a);
    }
}
```

Cài đặt bằng đệ quy

- i là số đỉnh đã đi qua, sum là tổng trọng số đường đi
- Mảng x[.] toàn cục lưu danh sách các đỉnh đi qua theo thứ tự..
- mark[j] = 0/1 lần lượt tương ứng với việc đỉnh j không thuộc/thuộc x[.].
- Mảng c[.][.] toàn cục lưu ma trận trọng số

Code 1b

```
void duyet(int i,int sum){
    if (i==n+1){
        if (sum+c[x[n]][1] < best)
            best=sum+c[x[n]][1];
        return;
    }
    for (int j=2;j<=n;++j)
        if (!mark[j]){
            mark[j]=1;
            x[i]=j;
            duyet(i+1,sum+c[x[i-1]][j]);
            mark[j]=0;
        }
}
```

Cải tiến bằng nhánh và cận

- Gọi minC = min(c[.][.])
- Cận dưới là $sum + c[x[i-1]][j] + minC \times (n-i)$

Code 1c

```
void duyet(int i,int sum){
    if (i==n+1){
        if (sum+c[x[n]][1] < best)
            best=sum+c[x[n]][1];
        return;
    }
    for (int j=2;j<=n;++j)
        if (!mark[j]){
            if (sum+c[x[i-1]][j]+minC*(n-i)<best){
                mark[j]=1;
                x[i]=j;
                duyet(i+1,sum+c[x[i-1]][j]);
                mark[j]=0;
            }
        }
}
```

Thuật toán 2

- Ta chỉ cần quan tâm đến đường đi ngắn nhất đi qua một tập đỉnh và đỉnh cuối cùng được thăm.
- Ví dụ: trong các thứ tự thăm (1 -> 2 -> 3 -> 4) và (1 -> 3 -> 2 -> 4), ta chỉ cần quan tâm đến đường đi ngắn hơn.
- Sử dụng mảng save[X][p] để lưu lại độ dài đường đi ngắn nhất đi qua tập đỉnh X và dừng tại đỉnh p.
  - Bit thứ i của X xác định đỉnh i đã được thăm chưa. Kỹ thuật này được gọi là bitmask
  - Kỹ thuật sử dụng mảng save[.][.] để lưu lại kết quả mỗi lần gọi đệ quy được gọi là đệ quy có nhớ.
- Hàm duyet(X, p) trả về đường đi ngắn nhất từ đỉnh p đi qua tất cả các đỉnh không thuộc X và quay về 0

Code 2a

```
int duyet(int X, int p){
    if (__builtin_popcount(X) == n) return c[p][0];
    if (save[X][p] != -1) return save[X][p];
    int ans = 2e9;
    for (int s = 0; s < n; ++s)
        if ((X >> s & 1) == 0){
            ans = min(ans, c[p][s]
            + duyet(1 << s | X, s));
        }
    save[X][p] = ans;
    return ans;
}
```

Code 2b

- Cận dưới là $c[p][s] + minC \times (n - i)$

```
int duyet(int X, int p){
    int i = __builtin_popcount(X);
    if (i == n) return c[p][0];
    if (save[X][p] != -1) return save[X][p];
    int ans = 2e9;
    for (int s = 0; s < n; ++s)
        if ((X >> s & 1) == 0 &&
        c[p][s] + minC * (n - i) < ans){
            ans = min(ans, c[p][s]
            + duyet(1 << s | X, s));
        }
    save[X][p] = ans;
    return ans;
}
```

Mục lục

Đề bài

- Có n hành khách $1, 2, \ldots, n$, hành khách i cần di chuyển từ địa điểm i đến địa điểm $i + n$
- Xe khách xuất phát ở địa điểm 0 và có thể chứa tối đa k hành khách
- Cho ma trận c với $c(i, j)$ là khoảng cách di chuyển từ địa điểm i đến địa điểm j
- Tính khoảng cách ngắn nhất để xe khách phục vụ hết n hành khách và quay trở về địa điểm 0
- **Lưu ý:** Ngoại trừ địa điểm 0, các địa điểm khác chỉ được thăm tối đa 1 lần

Thuật toán

- Lời giải của bài toán là một hoán vị các đỉnh (tương tự bài TSP)
- Với mỗi hoán vị, cần kiểm tra ràng buộc:
  - Số hành khách trên xe không vượt quá k tại bất kỳ thời điểm nào
  - Đỉnh i phải được thăm trước đỉnh $i + n, \forall i \in [1, \ldots n]$
- Ta sử dụng lại các thuật toán đã sử dụng cho bài TSP nhưng có thêm kiểm tra ràng buộc.

Kiểm tra hoán vị

```
bool checkPermutation(int a[]) {
    int count = 0;
    bool flag[n];
    memset(flag, false, sizeof(flag));
    for (int i = 1; i <= 2 * n; i++) {
        if (a[i] <= n) {
            count++;
            flag[a[i]] = true;
            if (count > k) return false;
        }
        else {
            if (!flag[a[i]-n]) return false;
            else count--;
        }
    }
    return true;
}
```

Kiểm tra trong hàm đệ quy

```
count = 0;
duyet(int X, int p) {
    ...
    for (s in [1, 2*n] and s not in X) {
        if (s <= n) {
            if (count + 1 > k) continue;
            flag[s] = true;
            count++;
        } else {
            if (flag[s-n] == false) continue;
            count--;
        }
```

Kiểm tra trong hàm đệ quy

```
            ...
            duyet(1 << s || X, s);
            ...
            if (s <= n) {
                count--;
                flag[s] = false;
            }
            else count++;
        }
        ...
    }
```

Mục lục

Đề bài

- Có n hành khách được đánh số từ 1 tới n.
- Có $2n + 1$ địa điểm được đánh số từ 0 tới $2n$
- Hành khách thứ i muốn đi từ địa điểm thứ i đến địa điểm thứ $i + n$
- Taxi xuất phát ở địa điểm thứ 0 và phải phục vụ n hành khách và quay lại địa điểm thứ 0 sao cho không điểm nào được đi lại 2 lần và tại một thời điểm chỉ có 1 hành khách được phục vụ.
- Cho khoảng cách giữa các địa điểm. Tính quãng đường nhỏ nhất mà tài xế Taxi phải đi.

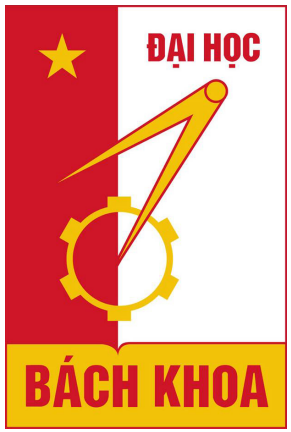
Thuật toán

Bài TAXI vừa là trường hợp đặc biệt của bài CBUS, vừa có thể quy dẫn về bài TSP:

- Bài TAXI có thể xem là trường hợp đặc biệt của bài CBUS với k $= 1$.
- Bài TAXI có thể quy dẫn về bài TSP:
  - Ta luôn phải đi qua mọi cạnh $(i, i + n)$
  - Xét đồ thị G'
  - Việc đưa đón hành khách thứ i là đỉnh i của đồ thị G'
  - Cung $(i, j) \in G'$ được tính bằng $c[i + n][j]$ với $c[.][.]$ là trọng số cạnh của đồ thị ban đầu
  - Kết quả của bài toán là lời giải TSP trên đồ thị G' cộng với tổng trọng số các cạnh $(i, i + n)$

Thuật toán ứng dụng
Bài thực hành số 2: Tìm kiếm vét cạn

TS. Bùi Quốc Trung, TA. Đặng Xuân Vương



Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 6 tháng 4 năm 2021