Thuật toán ứng dụng

BÀI TẬP LẬP TRÌNH

Dành cho Trợ giảng thực hành

SOICT — HUST

02/02/2020

Outline

01. INTRODUCTION

02. DATA STRUCTURE AND LIBS

03. EXHAUSTIVE SEARCH

01. INTRODUCTION

02. DATA STRUCTURE AND LIBS

03. EXHAUSTIVE SEARCH
    03. TSP
    03. KNAPSAC
    03. BCA
    03. BACP
    03. CONTAINER
    03. CBUS
    03. CVRP

03. TSP (Thai9cdb)

- Cho một đồ thị đầy đủ có trọng số.
- 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 đệ quy - quay lui hoặc thuật toán sinh kế tiếp.

Code 1

//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. Mảng c[.][.] toàn cục lưu ma trận trọng số

```
1
2  void duyet(int i,int sum){
3      if (i==n+1){
4          if (sum+c[x[n]][1] < best)
5              best=sum+c[x[n]][1];
6          return;
7      }
8      for (int j=2;j<=n;++j)
9          if (!mark[j]){
10             if (sum+c[x[i-1]][j]<best){
11                 mark[j]=1;
12                 x[i]=j;
13                 duyet(i+1,sum+c[x[i-1]][j]);
14                 mark[j]=0;
15             }
16         }
17 }
```

Thuật toán 2

- Để ý cây đệ quy (tạo ra từ quá trình duyệt) có rất nhiều cây con giống nhau. Ta có thể chỉ duyệt một lần và lưu trữ lại kết quả.
- Trạng thái duyệt là danh sách các đỉnh đã đi qua và đỉnh hiện tại đang đứng (hay danh sách các số đã xuất hiện và số cuối cùng trong phần hoán vị đã xây dựng). Hai cây con giống nhau nếu trạng thái tại nút gốc của chúng giống nhau.
- Ta có thể lưu trữ nghiệm tối ưu cho từng trạng thái để không phải duyệt lại nhiều lần. Sử dụng kỹ thuật bitmask để lưu trữ thuận tiện hơn.

Code 2

//p là đỉnh đang đứng, X là tập các đỉnh đã đi qua

```
18 int duyet(int X, int p){
19     if (__builtin_popcount(X) == n) return c[p][0];
20     if (save[X][p] != -1) return save[X][p];
21     int ans = 2e9;
22     for (int s = 0; s < n; ++s)
23         if ((X >> s & 1) == 0){
24             ans = min(ans, c[p][s]
25             + duyet(1 << s | X, s));
26         }
27     save[X][p] = ans;
28     return ans;
29 }
```

03. KNAPSAC (Thai9cdb)

- Cho một cái túi có thể chứa tối đa khối lượng M và n đồ vật. Mỗi đồ vật có khối lượng và giá trị của nó.
- Tìm cách lấy một số đồ vật sao cho tổng khối lượng không vượt quá M và tổng giá trị lớn nhất có thể.

Thuật toán 1

- Mỗi cách chọn lấy các đồ vật tương ứng với một dãy nhị phân độ dài n. Bit thứ i là 0/1 tương ứng là không lấy/có lấy đồ vật thứ i.
- Xét hết các xâu nhị phân độ dài n và tìm nghiệm tốt nhất.
- Để xét hết các xâu nhị phân độ dài n, có thể dùng đệ quy - quay lui hoặc chuyển đổi giữa thập phân với nhị phân.
- Độ phức tạp $O(2^n \times n)$

Thuật toán 2

- Chia tập đồ vật làm hai phần A và B. Mỗi cách chọn lấy các đồ vật tương ứng với một cách lấy bên A kết hợp với một cách lấy bên B.
- Ý tưởng chính ở đây là lưu trữ hết các cách lấy bên B và sắp xếp trước theo một thứ tự. Sau đó với mỗi cách lấy bên A, ta có thể tìm kiếm nghiệm tối ưu bên B một cách nhanh chóng.
- Giả sử cách lấy tối ưu là lấy m_A bên A và m_B bên B, ta sẽ xét tuần tự từng m_A một và tìm kiếm nhị phân m_B.
- Độ phức tạp $O(2^A \times \log(2^B) + 2^B) = O(2^{n/2} \times n)$ nếu chọn $|A| = |B| = n / 2$

Code 2

//S1, S2 là tập các cách lấy bên A, bên B. maxV[j] là max(S2[0..j].v)
//S1 và S2 được tính bằng đệ quy - quay lui

```
30      int ans = -1e9;
31      for(int i=0;i<S1.size();++i){
32          int L = 0, H = S2.size()-1, j = -1;
33          while (L<=H){
34              int m = (L+H)/2;
35              if (S1[i].m + S2[m].m <= M){
36                  j = m;
37                  L = m+1;
38              } else H = m-1;
39          }
40          if (j!=-1){
41              ans = max(ans, S1[i].v + maxV[j]);
42          }
43      }
44      cout << ans;
```

03. BCA (ngocbh)

- Có n khóa học và m giáo viên, mỗi giáo viên có danh sách các khóa có thể dạy.
- Có danh sách các khóa học không thể để cùng một giáo viên dạy do trùng giờ.
- Load của một giáo viên là số khóa phải dạy của giáo viên đó.
- **Yêu cầu:** Tìm cách xếp lịch cho giáo viên sao cho Load tối đa của các giáo viên là tối thiểu.

Thuật toán I

- Sử dụng thuật toán vét cạn, duyệt toàn bộ khóa học, xếp giáo viên dạy khóa học đó.
- Sử dụng thuật toán nhánh cận:
  - Chọn khóa học chưa có người dạy có số giáo viên dạy ít nhất để phân công trước.
  - Nếu phân công cho giáo viên A môn X, mọi môn học trùng lịch với môn X không thể được dạy bởi giáo viên A sau này.
  - Nếu maxLoad hiện tại lớn hơn minLoad tối ưu thu được trước đó thì không duyệt nữa.

03. BACP (PQD)

- The BACP is to design a balanced academic curriculum by assigning periods to courses in a way that the academic load of each period is balanced .
- There are N courses $1, 2, \ldots, N$ that must be assigned to M periods $1, 2, \ldots, M$. Each course i has credit c_i and has some courses as prerequisites. The load of a period is defined to be the sum of credits of courses assigned to that period.
- The prerequisites information is represented by a matrix $A_{N \times N}$ in which $A_{i,j} = 1$ indicates that course i must be assigned to a period before the period to which the course j is assigned. Given constants a, b. Compute the solution satisfying constraints
  - Total number of courses assigned to each period is greater or equal to a and smaller or equal to b
  - The maximum load for all periods is minimal

03. BACP (PQD)

- Input
  - Line 1 contains N and M $(2 \leq N \leq 16, 2 \leq M \leq 5)$
  - Line 2 contains $c_1, c_2, \ldots, c_N$
  - Line $i + 2 (i = 1, \ldots, N)$ contains the i^{th} line of the matrix A
- Output: Unique line contains that maximum load for all periods of the solution found

03. CONTAINER (name)

03. CBUS (VUONGDX)

- 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 I

- Sử dụng thuật toán vét cạn, thực hiện bằng thủ tục đệ quy để thử mọi thứ tự thăm
- Sử dụng *bitmask* s để lưu trạng thái các địa điểm đã được thăm, biến l lưu lại số lượng hành khách ở trên xe khách và biến v lưu đỉnh cuối cùng thuộc đường đi đang xét
- Tại mỗi lần gọi đệ quy, ta xét các địa điểm chưa được thăm i
  - Nếu $1 \leq i \leq n$
    - Nếu $l = k$, không thể thăm địa điểm i
    - Nếu $l < k$, cập nhật $l = l + 1$, thêm i vào đường đi hiện tại và gọi đệ quy
  - Nếu $n + 1 \leq i \leq 2n$
    - Nếu địa điểm $i - n$ đã được thăm, cập nhật $l = l - 1$, thêm i vào đường đi hiện tại và gọi đệ quy. Ngược lại, không thể thăm địa điểm i

Thuật toán II

- Sử dụng kỹ thuật nhánh cận để giảm số lần gọi đệ quy. Xe khách luôn cần đi qua $2 \times N + 1$ cạnh. Gọi f là độ dài đường đi hiện tại, r là số cạnh mà xe khách còn phải đi qua, c_{min} là độ dài cạnh nhỏ nhất, ta có công thức cận:

$$bound = f + r \times c_{min} \tag{1}$$

03. CVRP (PQD)

- A fleet of K identical trucks having capacity Q need to be scheduled to deliver pepsi packages from a central depot 0 to clients $1, 2, \ldots, n$. Each client i requests $d[i]$ packages.
- Solution: For each truck, a route from depot, visiting clients and returning to the depot for deliverying requested pepsi packages such that:
  - Each client is visited exactly by one route
  - Total number of packages requested by clients of each truck cannot exceed its capacity
- Goal
  - Compute number R of solutions

03. CVRP (PQD)

- Input
  - Line 1: n, K, Q $(2 \leq n \leq 10, 1 \leq K \leq 5, 1 \leq Q \leq 20)$
  - Line 2: $d[1], ..., d[n]$ $(1 \leq d[i] \leq 10)$
- Output: $R \bmod 10^9 + 7$