Thuật toán ứng dụng
Bài thực hành số 2

Giảng viên: TS. Đinh Viết Sang
Trợ giảng: Nguyễn Trung Hiếu

Viện Công nghệ thông tin & truyền thông
Đại học Bách khoa Hà Nội

29/03/2021

Nội dung

03. TSP

- Bài toán người bán hàng - Travelling salesman problem
- Được nêu ra lần đầu tiên năm 1930
- Là một bài toán NP-khó thuộc thể loại tối ưu rời rạc
- Tóm tắt: Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất xuất phát từ thành phố thứ nhất và đi qua mỗi thành phố đúng một lần.

Thuật toán

- 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.
- Độ phức tạp: $O(n!)$

Đệ quy quay lui

```
const int INTMAX = 1e9;

void TSP(int u, int cur)
{
   if(cur == n){
      ans = min(ans, res + c[u][1]);
      return;
   }
   for(int i = 1; i <= n; i++){
      if(!mark[i] && c[u][i] != INTMAX){
         mark[i] = 1;
         res += c[u][i];
         TSP(i, cur+1);
         res -= c[u][i];
         mark[i] = 0;
      }
   }
}
```

Đệ quy quay lui

```
19  int main()
20  {
21    ios_base::sync_with_stdio(0);
22    cin >> n >> m;
23    int u, v, w;
24    for(int i = 1; i <= n; i++){
25      for(int j = 1; j <= n; j++) c[i][j] = INTMAX;
26    }
27    for(int i = 1; i <= m; i++){
28      cin >> u >> v >> w;
29      c[u][v] = w;
30    }
31    res = 0;
32    ans = 1e9;
33    mark[1] = 1;
34    TSP(1, 1);
35    cout << ans;
36    return 0;
37  }
```

Sinh hoán vị

```
38  for(int i = 1; i <= n; i++) p[i] = i;
39
40    do{
41      if(p[1] != 1) break;
42      int sum = 0;
43      for(int i = 2; i <= n; i++) {
44        if(c[p[i - 1]][p[i]] == INTMAX) {
45          sum = INTMAX;
46          break;
47        }
48        sum += c[p[i - 1]][p[i]];
49      }
50      sum += c[p[n]][1];
51      ans = min(ans, sum);
52    }
53    while(next_permutation(p + 1, p + n + 1));
```

- *next_permutation*() trả về kiểu giá trị bool có ý nghĩa có thể sinh tiếp hoán vị hay không, đồng thời nó cũng sinh hoán vị tiếp theo nếu có thể.

Cải tiến

```
1  void TSP(int u, int cur)
2  {
3     if(cur == n){
4        ans = min(ans, res + c[u][1]);
5        return;
6     }
7     for(int i = 1; i <= n; i++){
8        if(!mark[i] && c[u][i] != INTMAX){
9           if(res + c[u][i] > ans) continue; // bound
10          mark[i] = 1;
11          res += c[u][i];
12          TSP(i, cur+1);
13          res -= c[u][i];
14          mark[i] = 0;
15       }
16    }
17 }
```

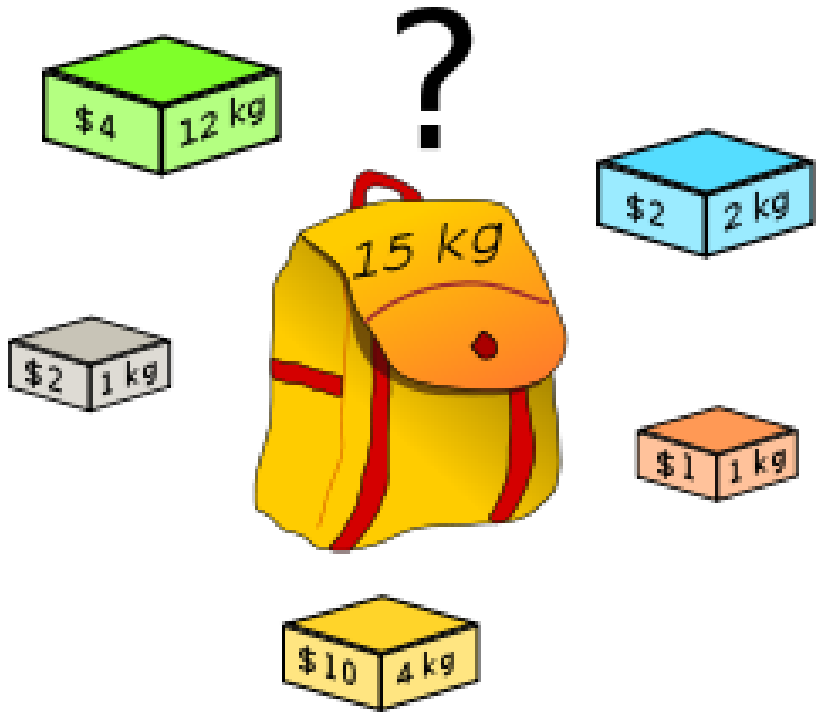
Ứng dụng

- Tối ưu trong vận tải
- Tối ưu mạch hàn trong vi mạch
- Tối ưu giao thông

03. KNAPSAC

- Một nhà thám hiểm cần đem theo một cái túi có trọng lượng không quá m.
- Có n đồ vật có thể đem theo.
- Đồ vật thứ i có trọng lượng a_i và giá trị sử dụng c_i.
- Hỏi nhà thám hiểm cần đem theo những đồ vật nào để cho tổng giá trị sử dụng là lớn nhất mà tổng trọng lượng đồ vật mang theo cái túi không vượt quá m?
- Ghi ra duy nhất một số là tổng giá trị lớn nhất tìm được của các đồ vật cho vào túi.

03. KNAPSAC



Thuật toán

- 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.
- Duyệt 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 $O(2^n \times n)$

Đệ quy sinh xâu nhị phân

```
1  void Get_ans() {
2    sumA = sumC = 0;
3    for(int i = 1; i <= n; i++) {
4      sumA += p[i] * a[i];
5      sumC += p[i] * c[i];
6    }
7    if(sumA <= b) ans = max(ans, sumC);
8  }
9
10 void Try(int x) {
11   if(x > n) {
12     Get_ans();
13     return;
14   }
15   for(int i = 0; i <= 1; i++) {
16     p[x] = i;
17     Try(x + 1);
18   }
19 }
```

Code sinh xâu nhị phân

```
20  int T = 1 << n;
21  for(int i = 0; i < T; i++) {
22      for(int j = 0; j < n; j++) {
23          p[j + 1] = (i >> j) & 1;
24      }
25      Get_ans();
26  }
```

Cải tiến

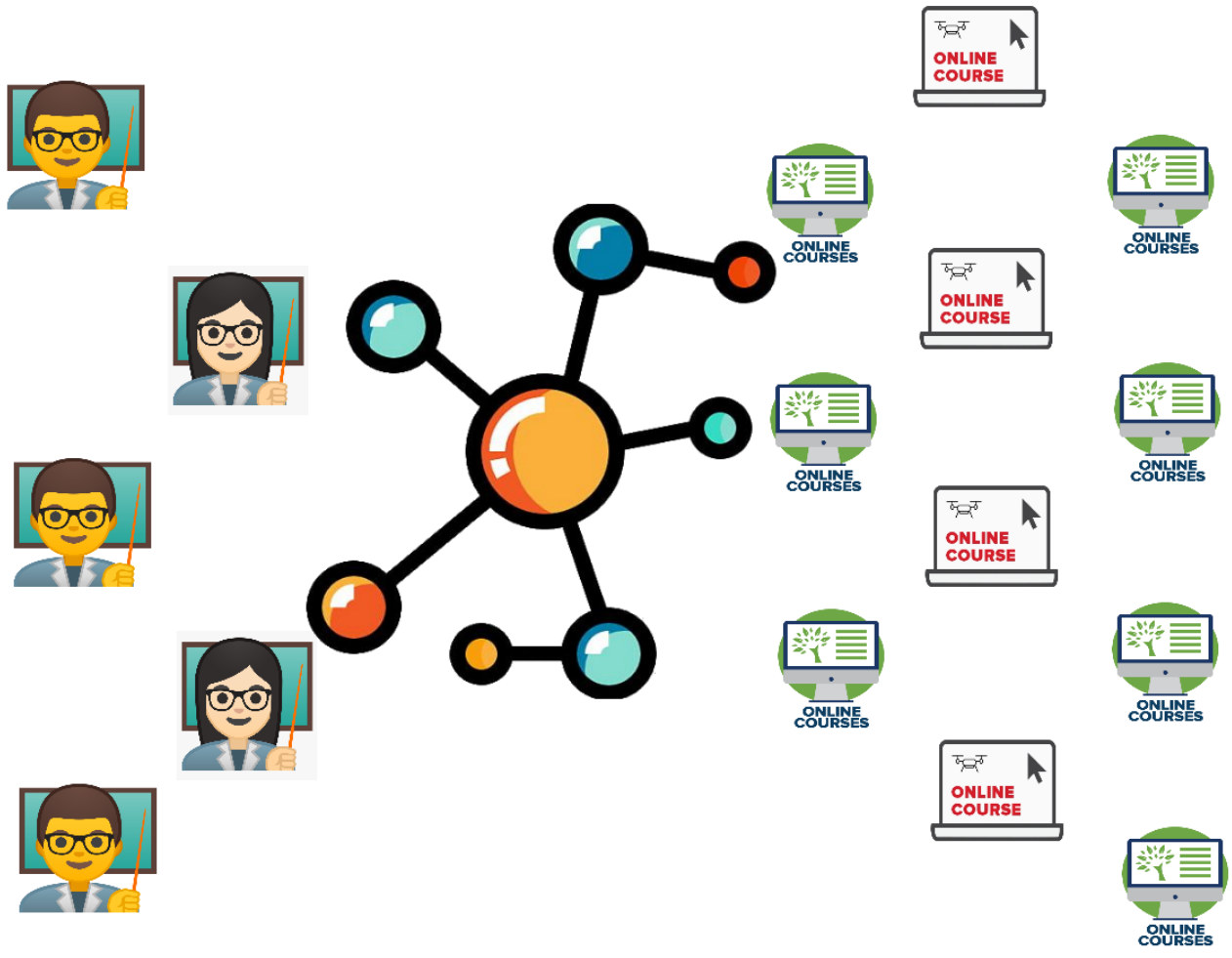
```
27  void Try(int x)
28  {
29    if(sumA > b) return;
30
31    if(x > n) {
32      ans = max(ans, sumC);
33      return;
34    }
35
36    for(int i = 0; i <= 1; i++) {
37      p[x] = i;
38      sumA += p[x] * a[x];
39      sumC += p[x] * c[x];
40      Try(x + 1);
41      sumA -= p[x] * a[x];
42      sumC -= p[x] * c[x];
43    }
44  }
```

Ứng dụng

- Xếp hàng lên xe tải, container
- Xếp đồ trong kho hàng
- Xếp balo khi đi du lịch

03. BCA

- 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.


ONLINE COURSE
ONLINE COURSES

Thuật toán

- Với mỗi môn học, cần tìm 1 giáo viên dạy môn đó sao cho lịch dạy của giáo viên đó không bị xung đột
- Có tất cả n^m cách chọn.
- Cải tiến với thuật toán nhánh cận
- Độ phức tạp $O((n \times n)^m)$

Code

```
1  void Try(int x, int curLoad) {
2    if(x > n) {
3      ans = min(ans, curLoad);
4      return;
5    }
6    if (curLoad > ans) return;
7    for (int t : courseTeacher[x]) {
8      bool ok = true;
9      // checking conflict between 2 courses
10     // if conflict ok == false
11     if(ok) {
12       p[x] = t;
13       cnt[t]++;
14       Try(x + 1, max(curLoad, cnt[t]));
15       cnt[t]--;
16       p[x] = 0;
17     }
18   }
19 }
```

Code

```
20 int main() {
21   cin >> m >> n;
22   int x, u, v;
23   for (int i = 1; i <= m; i++) {
24     cin >> x;
25     for(int j = 1; j <= x; j++) {
26       cin >> u;
27       courseTeacher[u].push_back(i);
28     }
29   }
30   cin >> k;
31   for (int i = 1; i <= k; ++i) {
32     cin >> u >> v;
33     course[u].push_back(v);
34     course[v].push_back(u);
35   }
36   Try(1, 0);
37   cout << (ans == INTMAX) ? -1 : ans;
38 }
```

Ứng dụng

- Xếp lịch học, lịch thi
- Xếp lịch làm việc

03. CVRPCOUNT

- Cho n khách hàng cần chuyển đồ, khách thứ i cần chuyển lượng hàng có khối lượng $d[i]$
- Có K xe tải, mỗi xe có trọng tải Q
- Cần thiết kế lịch trình cho các xe sao cho:
  - Mỗi khách hàng được thăm bởi duy nhất 1 xe
  - Số lượng hàng trong mỗi xe không vượt quá trọng tải
  - Mỗi xe phải thăm ít nhất 1 khách hàng

Thuật toán

- Gọi 1 cách xếp là 1 cách ta chỉ định các xe chở hàng cho các khách hàng $\rightarrow$ có N^K cách xếp.
- Duyệt toàn bộ N^K trạng thái này, tính số trạng thái đảm bảo điều kiện số hàng phải chở không quá Q.
- Với mỗi xe, giả sử có p khách hàng được chỉ định thì sẽ có $p!$ hoán vị các khách hàng.
- Vì các xe tải là như nhau $\rightarrow$ kết quả phải chia cho $k!$.
- Ta có thể loại bỏ phép chia bằng cách giữ thứ tự xe tải xuất hiện trong trạng thái là tăng dần.
- Độ phức tạp $O(N^K \times K)$

Code

```
1  void Try(int x) {
2    if(x > n) {
3      int t = 1;
4      for(int k = 1; k <= K; k++) {
5        if(r[k] == 0) return;
6        else t = (t * fac[c[k]]) % MOD;
7      }
8      res = (res + t) % MOD;
9      return;
10   }
11   for(int k = 1; k <= K; k++) {
12     if(m[k] + d[x] <= Q) {
13       m[k] += d[x];
14       c[k]++;
15       Try(x + 1);
16       m[k] -= d[x];
17       c[k]--;
18     }
19   }
20 }
```

Code

```
21 int main()
22 {
23    cin >> n >> K >> Q;
24    for (int i = 1; i <= n; i++){
25       cin >> d[i];
26    }
27
28    fac[0] = 1;
29    for (int i = 1; i < N; i++){
30       fac[i] = i * fac[i - 1];
31    }
32
33    res = 0;
34    Try(1);
35
36    cout << res / fac[K];
37 }
```