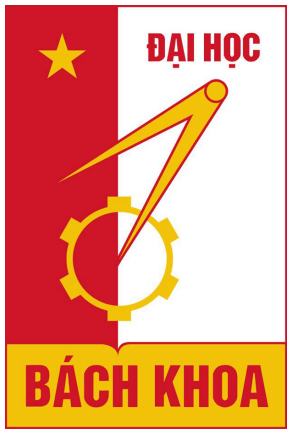
Thuật toán ứng dụng
Bài thực hành số 2: Tìm kiếm vét cạn

TS. Nguyễn Tuấn Dũng, ThS. Nguyễn Duy Hiệp, ThS. Tạ Minh Trí,
TA. Đặng Xuân Vương



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

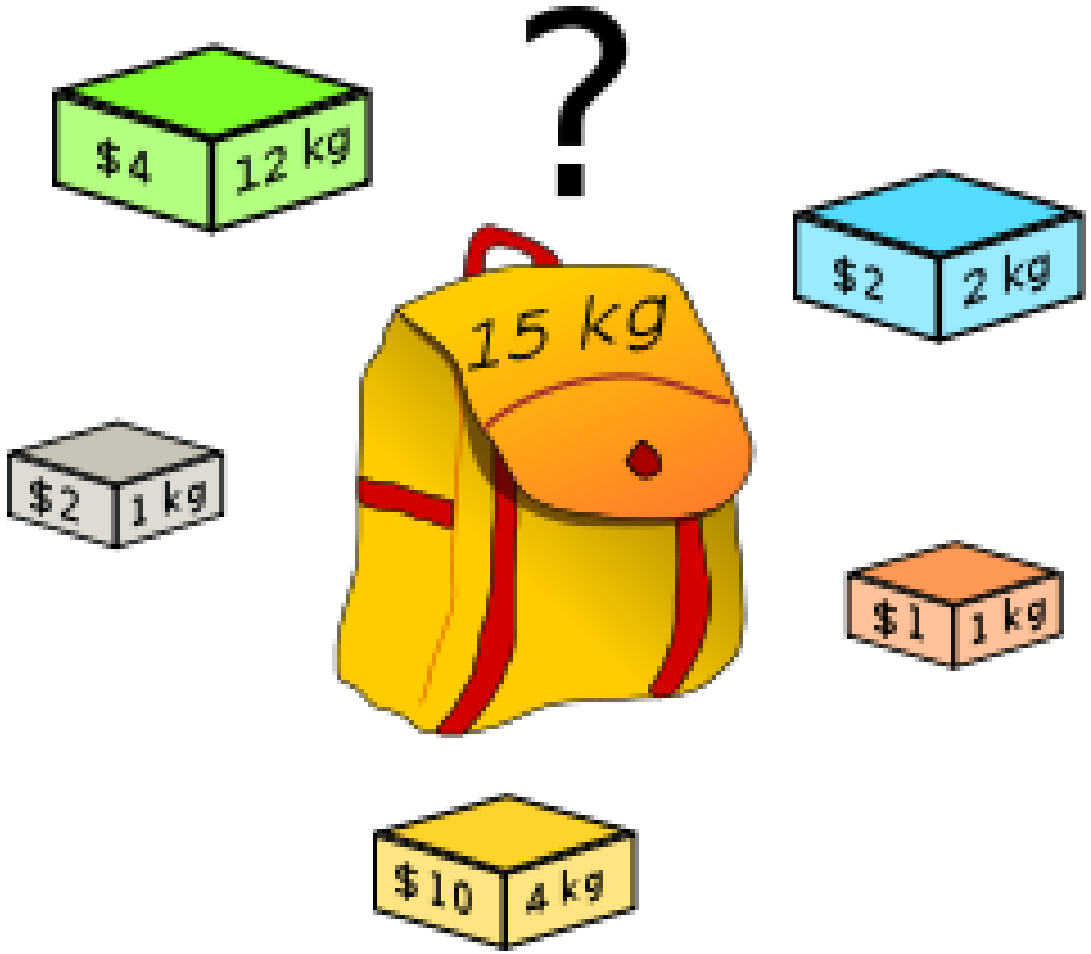
Ngày 1 tháng 4 năm 2020

Mục lục

Mục lục

Đề bài

- Cho n đồ vật và một cái túi có sức chứa tối đa là M. 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.


?
$4
12 kg
$2
2 kg
15 kg
$2
1 kg
$1
1 kg
$10
4 kg

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.
- Ta có thể sử dụng vòng lặp hoặc dùng đệ quy - quay lui.
- Độ phức tạp $O(2^n \times n)$

Code 1a

```
main(){
    cin >> n >> M;
    for (int i = 0; i < n; ++i)
    cin >> m[i] >> v[i];
    int ans = 0;
    for (int mask = 1 << n - 1; mask >= 0; mask--) {
        // mask = [0..2^n-1]
        int sumM = 0, sumV = 0;
        for (int i = 0; i < n; ++i)
            if (mask >> i & 1){//bit thu i = 1
                sumM += m[i];
                sumV += v[i];
            }
        if (sumM <= M) ans = max(ans, sumV);
    }
    cout << ans;
}
```

Code 1b

```
void duyet(int i,int sum,int val){
    if (i>n){
        if (val>best) best=val;
        return;
    }
    duyet(i+1,sum,val);//bit 0
    if (sum+m[i]<=M)
        duyet(i+1,sum+m[i],val+v[i]);//bit 1
}
```

Cải tiến với nhánh và cận

- Gọi $maxAVW$ là giá trị trung bình theo khối lượng lớn nhất của tất cả các đồ vật ($maxAVW = max(\lceil v[i]/m[i] \rceil), \forall i \in [1, n]$).
- Cận trên là $val + v[i] + maxAVW \times (M - sum - m[i])$

Code 1c

```
void duyet(int i,int sum,int val){
    if (i>n){
        if (val>best) best=val;
        return;
    }
    duyet(i+1,sum,val);//bit 0
    if (sum+m[i]<=M &&
    val+v[i]+maxAVW*(M-sum-m[i])>best)
        duyet(i+1,sum+m[i],val+v[i]);//bit 1
}
```

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 $O(2^{|A|} \times \log(2^{|B|}) + 2^{|A|} \times |A| + 2^{|B|} \times |B|) = O(2^{n/2} \times n)$ nếu chọn $|A| = |B| = n / 2$

Cài đặt thuật toán 2

- S1, S2 là tập các cách lấy bên A, bên B
- S2 được sắp xếp tăng dần theo tổng khối lượng các đồ vật được lấy
- maxV[j] là max(S2[0..j].v)
- S1 và S2 được tính bằng đệ quy - quay lui

Code 2

```
int ans = -1e9;
for(int i=0;i<S1.size();++i){
    int L = 0, H = S2.size()-1, j = -1;
    while (L<=H){
        int m = (L+H)/2;
        if (S1[i].m + S2[m].m <= M){
            j = m;
            L = m+1;
        } else H = m-1;
    }
    if (j!=-1){
        ans = max(ans, S1[i].v + maxV[j]);
    }
}
cout << ans;
```

Mục lục

Đề bài

- 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 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 $(i, i + n)$

Mục lục

Mô hình giải bài

- **Tìm kiếm vét cạn**
- Tham lam
- Chia để trị
- Quy hoạch động
- ...

Cách thực thi chương trình

- Sử dụng vòng lặp
- Sử dụng hàm đệ quy
- ...

Phương pháp tìm kiếm

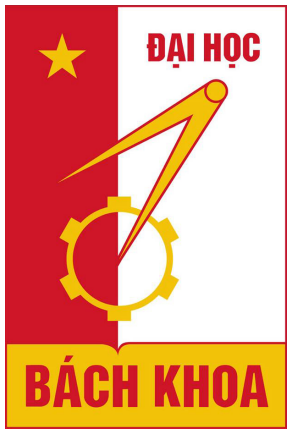
- Liệt kê lời giải
- Quay lui
  - Nhánh và cận
- ...

Chú ý

- Đệ quy $\neq$ Quay lui $\neq$ Nhánh và cận

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

TS. Nguyễn Tuấn Dũng, ThS. Nguyễn Duy Hiệp, ThS. Tạ Minh Trí,
TA. Đặng Xuân Vương



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

Ngày 1 tháng 4 năm 2020