Paradigmas de Resolução de Problemas

Algoritmos Gulosos: Definição

Prof. Edson Alves – UnB/FGA

Sumário

1. Algoritmos Gulosos

2. Exemplos de algoritmos gulosos

Algoritmos Gulosos

Definição

- Um algoritmo é dito guloso (*greedy*, em inglês) se, a cada iteração, ele faz uma escolha local ótima que converge para a solução global ótima
- Para a estratégia gulosa levar a um algoritmo correto para todas as entradas é preciso que o problema tenha duas características:
    1. ter subestruturas ótimas
    2. ter a propriedade gulosa
- Ter subestruturas ótimas significa que a solução global ótima contém soluções ótimas para os subproblemas que compõem o problema principal
- A propriedade gulosa garante que, uma vez tomada a decisão local ótima, ela não precisará ser reconsiderada
- Em geral, é difícil provar que um problema tem ambas características

Algoritmos gulosos em programação competitiva

- É fácil pensar em algoritmos gulosos: difícil é provar a corretude destes algoritmos
- Em competições, soluções gulosas ou estão corretas ou levam ao WA
- Raramente estas soluções obtém um TLE, por terem complexidades assintóticas relativamente baixas
- Mesmo que o problema possa ser resolvido por um algoritmo guloso, a estratégia de escolha das soluções ótimas para os subproblemas pode não ser óbvia
- Em geral, se o tamanho da entrada não for proibitivo, é melhor tentar uma abordagem por busca completa ou por programação dinâmica antes de tentar uma abordagem gulosa

Exemplos de algoritmos gulosos

Problema do troco

Problema do Troco

Seja $C = \{c_1, c_2, \ldots, c_N\} \subset \mathbb{N}$, com $c_i < c_j$ se $i < j$, o conjunto dos tipos de moedas disponíveis. O problema do troco consiste em, dada uma quantia $Q \in \mathbb{N}$, determinar N inteiros não-negativos k_i tais que

$$k_1 c_1 + k_2 c_2 + \ldots + k_N c_N = Q$$

e que a soma

$$\sum_{i=1}^{N} k_i$$

seja mínima.

Algoritmo guloso para o problema do troco

- A estratégia gulosa para o problema do troco é escolher o maior número possíveis de moedas do tipo c_N, em seguida o maior número de moedas do tipo c_{N-1}, e assim por diante, até a moeda c_1
- Por exemplo, se $C = \{1, 2, 5, 10, 25, 50\}$ e $Q = 198$, inicialmente se escolhe $k_6 = 3$ moedas de 50
- Em seguida, toma-se uma moeda de $k_5 = 25$, totalizando $3 \times 150 + 1 \times 25 = 175$
- Agora, seguindo o mesmo raciocínio, escolhe-se $k_4 = 2$, $k_3 = 0$, $k_2 = 1$ e $k_1 = 1$, de modo que

  $$3 \times 50 + 1 \times 25 + 2 \times 10 + 0 \times 5 + 1 \times 2 + 1 \times 1 = 198$$

  e

  $$\sum_{i=1}^{N} k_i = 3 + 1 + 2 + 0 + 1 + 1 = 8$$
- A complexidade desta solução é $O(N)$, pois são feitas, no máximo, N divisões, uma para cada tipo de moeda

Implementação da estratégia gulosa para o problema do troco

```
// Retorna o mínimo de moedas, sem discriminar os valores ki
int coin_change(int Q, const vector<int>& cs)
{
    int ans = 0;

    // Processa as moedas da maior para a menor
    for (auto it = cs.rbegin(); it != cs.rend(); ++it)
    {
        int c = *it, k = Q / c;

        ans += k;
        Q -= k*c;
    }

    return ans;
}
```

Observações sobre a estratégia gulosa para o problema do troco

- Observe que, se as moedas fossem processadas da menor para a maior, o algoritmo não produziria a resposta correta
- No exemplo anterior, a resposta seria $k_1 = 198$ e $k_i = 0$ para $i > 1$
- Além disso, considere $C = \{1, 4, 5\}$ e $Q = 8$
- O algoritmo guloso para o problema do troco produziria $k_3 = 1$, $k_2 = 0$ e $k_1 = 3$, num total de 4 moedas
- Porém, é possível gerar 8 com apenas duas moedas, fazendo $k_3 = k_1 = 0$ e $k_2 = 2$
- Assim, o algoritmo guloso para o problema do troco não produz a saída correta para todas as entradas
- Um conjunto C para o qual o algoritmo guloso produz sempre a saída correta para o problema do troco é denominada base canônica
- Dentre as bases canônicas mais comuns estão o sistema monetário vigente (base do primeiro exemplo), as potências de uma base $b > 1$ e os números de Fibonacci

Agendamento de eventos

Agendamento de eventos

Seja $E = \{e_1, e_2, \ldots, e_N\}$ um conjunto de N eventos e_i cuja duração é determinada pelos intervalos $[a_i, b_i)$. O problema do agendamento de eventos consiste em determinar um subconjunto $S \subset E$ tal que

$$\forall s_i, s_j \in S, s_i \cap s_j = \emptyset, \quad \text{se} \quad i \neq j$$

e $|S|$ é o maior possível.

Algoritmo guloso para o agendamento de eventos

- O problema do agendamento de eventos pode ser resolvido com um algoritmo guloso para todas as entradas possíveis, embora não seja óbvio à primeira vista qual seria a escolha ótima para cada subproblema
- Por exemplo, escolher pelo evento de menor duração ainda não escolhido e que não conflita com os já escolhidos leva a resposta errada quando $E = \{[1, 5), [4, 7), [6, 10)\}$, pois o algoritmo escolheria apenas o evento e_2, sendo que a solução correta seria a escolha dos eventos e_1 e e_3
- Outra abordagem incorreta seria escolher os eventos que começam o mais cedo possível ainda não selecionados
- O algoritmo correto resulta da escolha dos eventos que terminam o mais cedo possível ainda não selecionados
- Este algoritmo tem complexidade $O(N \log N)$, pois é preciso ordenar os eventos pelos valores b_i

Implementação do algoritmo guloso para o agendamento de eventos

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Event { int a, b, i; };

vector<int> solve(int N, const vector<Event>& events)
{
    vector<Event> es(events);

    // Ordena os eventos pelo encerramento
    sort(es.begin(), es.end(), [](const Event& x, const Event& y) {
        return x.b < y.b;
    });

    vector<int> ans(1, es[0].i);
    int last = es[0].b;
```

Implementação do algoritmo guloso para o agendamento de eventos

```
    for (int i = 1; i < N; ++i)
    {
        // O próximo evento começa antes do término do anterior
        if (es[i].a < last)
            continue;

        ans.push_back(es[i].i);
        last = es[i].b;
    }

    // Retorna os identificadores dos eventos escolhidos
    return ans;
}
```

Referências

1. **LAARKSONEN**, Antti. *Competitive Programmer's Handbook*, 2017.
2. **HALIM**, Steve; **HALIM**, Felix. *Competitive Programming 3*, Lulu, 2013.