① 11ª Aula

Algoritmos gulosos

→ Normalmente aplicado a problemas de otimização, em que queremos computar a melhor solução

→ Em cada passo, o algoritmo sempre escolhe a melhor opção local visível, sem se preocupar com as consequências futuras (dizemos que ele é "míope")

②

→ Nem sempre produz a solução ótima

→ Não existe backtracking envolvido

→ Na maioria das vezes, projetar ou descrever um algoritmo guloso é fácil, mas provar sua corretude é difícil

→ Tem similaridade com Programação Dinâmica - Subestrutura ótima

③ Estratégia guloso vs Programação dinâmica

Algoritmo guloso (ganancioso):

- → "abocanha" a alternativa mais promissora sem explorar as outras
- → a execução costuma ser muito rápida
- → nunca se arrepende de uma decisão tomada
- → prova de corretude difícil

④

Algoritmo com programação dinâmica:

- → explora todas as alternativas, e faz isso de maneira eficiente
- → a execução é um tanto "lenta"
- → a cada iteração pode se arrepender de decisões tomadas anteriormente (pode rever o "ótimo corrente")
- → prova de corretude fácil

⑤ Exemplo: Máximo vs Maximal

S é uma coleção de subconjuntos de $\{1, \dots, n\}$.

$X \subset S$ é <u>máximo</u> se não existe $Y \subset S$ tal que $|Y| > |X|$.

$X \subset S$ é <u>maximal</u> se não existe $Y \subset S$ tal que $Y \supset X$, ou seja, se nenhum elemento de S é superconjunto próprio de X.

⑥ Exemplificando

$S = \{\{1,2\}, \{2,3\}, \{4,5\}, \{1,2,3\}, \{1,2,4\}, \{2,3,4,5\}, \{1,3,4,5\}\}$

Elementos máximo: $\{2,3,4,5\}$ e $\{1,3,4,5\}$

Elementos maximais: $\{1,2,3\}$, $\{1,2,4\}$, $\{2,3,4,5\}$ e $\{1,3,4,5\}$

Note que todo máximo é maximal, mas a recíproca não é verdade.

(7)

→ Procurar um elemento máximo em S é computacionalmente pesado (examinar todos os elementos)

→ Mas encontrar um elemento maximal de S é muito fácil, aplicando a estratégia guloso.

(8)

Algoritmo guloso para o maximal

```
escolha algum X em S
enquanto X ⊂ Y para algum Y em S
        faça X ← Y
devolva X
```

Verifique que o algoritmo funciona para o exemplo descrito anteriormente!

(9) Problema de seleção de atividades

- Considere um conjunto $\{1, 2, \ldots, n\}$ de n atividades que competem por um recurso, por exemplo uma sala de aula.
- Cada atividade tem um tempo de início s_i e um tempo de término t_i, com $s_i < t_i$
- O intervalo requerido pela atividade i é $[s_i, t_i)$

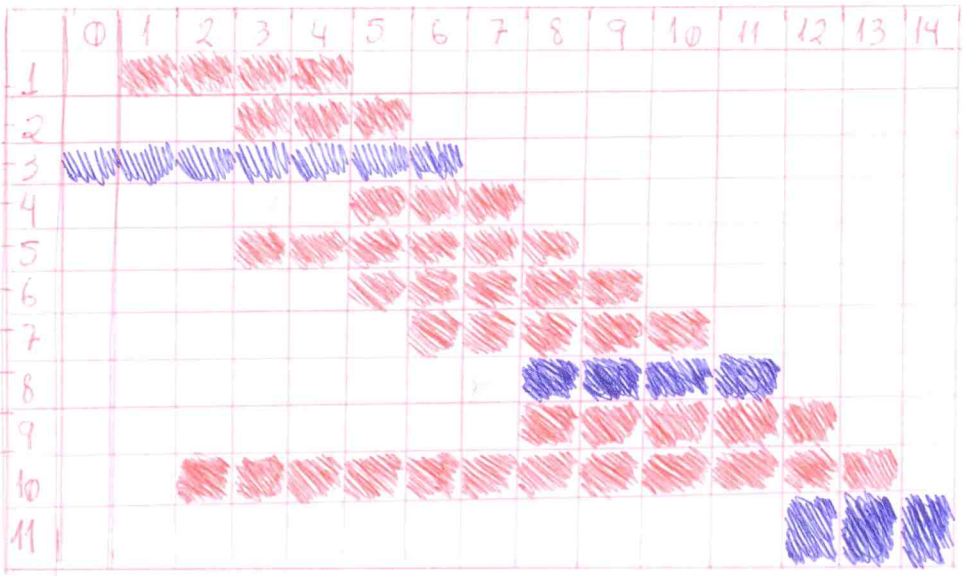
(10)

- Duas atividades i e j são compatíveis se os intervalos $[s_i, t_i)$ e $[s_j, t_j)$ não se interceptam ($s_i \geq t_j$ ou $s_j \geq t_i$).
- Problema: encontrar o conjunto de atividades mutuamente compatíveis de tamanho máximo.

(11) Exemplificando:

Consideremos 11 atividades em 14 unidades de tempo.

1ª tentativa: Escolher primeiro as atividades que começam primeiro



Escolhemos 3, 8 e 11, mas poderia ser melhor: 1, 4, 8 e 11.

(12)

2ª tentativa: Escolher primeiro as atividades que demoram menos tempo.

Desta forma escolhemos 2, 8 e 11, mas poderia ser melhor: 1, 4, 8 e 11.

Tentemos mais uma vez!

(13)

3ª tentativa: Escolher primeiro as atividades que terminam primeiro

Agora sim! Escolhemos 1, 4, 8 e 11.

Note que está é a solução ótima!

(14) Descrição do algoritmo

```
seleciona_atividades_guloso(s, t, n)}
    A ← {1}      // t1 ≤ t2 ≤ ... ≤ tn
    j ← 1
    para i = 2 até n faça
        se si ≥ tj          // i e j são compatíveis
        então A ← A ∪ {i}
              j ← i
}
```

Complexidade do algoritmo: $\Theta(n)$

(15) Problema do Pen Drive

Tenho um grande número de arquivos digitais no meu computador. Cada arquivo ocupa um certo número de MB (megabytes). Quero gravar o maior número possível de arquivos em um Pen Drive que possui capacidade c MB. O problema pode ser modelado assim: dados números naturais $p_1, p_2, \ldots, p_n$ e c, encontrar o maior subconjunto X de $\{1, 2, \ldots, n\}$ que satisfaça a restrição $\sum_{i \in X} p_i \leq c$.

(16) ...

Mostre que um algoritmo guloso aproprie do resolve o problema. Depois implemente tal algoritmo (em linguagem C)