Teoria dos Grafos
Aula 14

Aula passada

- MST

Aula de hoje

- Construção de algoritmos
- Paradigma guloso
- Escalonando tarefas no tempo (*interval scheduling*)

Projetando Algoritmos

- Dado um problema P, como projetar um algoritmo que gere a solução?
  - ex. determinar o maior ciclo de um grafo
- Técnicas para *mecanizar* a construção de algoritmos?

Problema central na Computação!

- Não temos conjunto de princípios e técnicas fundamentais e gerais
  - projeto de algoritmos ainda muito baseado em arte
- Algumas idéias: força bruta, algoritmo guloso, programação dinâmica

Força Bruta

- Buscar pela solução no espaço de possíveis soluções para o problema
  - enumerar possíveis soluções de maneira direta; percorrer esta enumeração
- Determinar o menor caminho entre os vértices u e v?
- Enumerar todas as permutações P com 1, 2, 3, ... vértices, testar se tem arestas u – P – v

Problema ?

- Fácil mecanizar usando força bruta
- Solução em geral tem alta complexidade (espaço de soluções cresce rapidamente)
- Inadequada para problemas grandes

Algoritmo Guloso

- **Idéia:** algoritmo que constrói solução de forma iterativa, tomando decisões ótimas a cada passo para otimizar algum objetivo global
- cada iteração resolve um problema "pequeno" e local de forma ótima

Exemplos vistos em aula?

- Dijkstra: processo iterativo, a cada passo escolhe vértice de menor distância
- Prim: processo iterativo, a cada passo escolhe aresta de menor peso

Algoritmo Guloso

Vantagens

- Geralmente fácil de construir algoritmo
- Baixa complexidade
- Heurísticas (intuição) ajudam na optimalidade

Desvantagens

- Geralmente não obtém o ótimo
- Garantir optimalidade é difícil
- Podem gerar soluções muito ruins

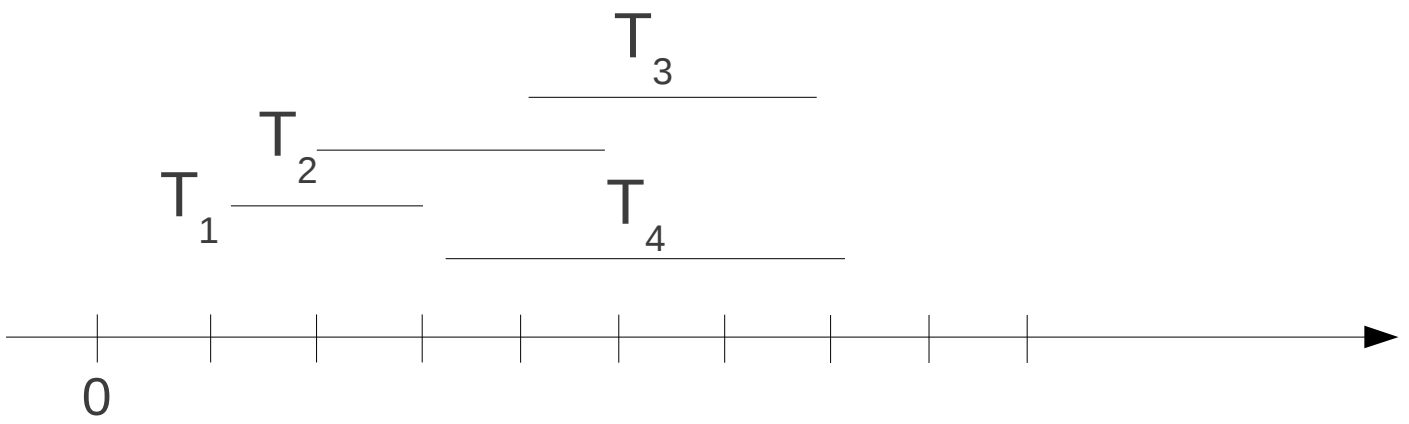
Técnica para projetar algoritmos

Escalonamento de Tarefas

- Conjunto de N tarefas a serem executadas
- Cada tarefa tem horário para iniciar e terminar

Exemplo

- N=4, T_1 = [1, 3], T_2 = [2, 5], T_3 = [4, 6], T_4 = [3.1, 7]



- **Problema:** Quais tarefas executar de forma a maximizar *número* de tarefas?

Escalonamento de Tarefas

- Escalonamento compatível
  - conjunto de tarefas que são *compatíveis*
  - não possuem sobreposição entre si
- **Objetivo:** Encontrar maior conjunto de tarefas compatíveis
- Solução ótima sempre única?
  - vide exemplo anterior
  - $S_1 = \{T_1, T_3\}$, $S_2 = \{T_1, T_4\}$

Algoritmo que obtém uma solução ótima qualquer

Algoritmo

Algoritmo que obtém uma solução ótima qualquer

- **Idéias?**
- Dica: utilizar paradigma guloso para construir algoritmo

- Possíveis gulosos (que não funcionam):
  - Escalonar primeiro, tarefa que começa primeiro
  - Escalonar primeiro, tarefa mais curta
  - Escalonar primeiro, tarefa que menos colide

Algoritmo Guloso

- 1) Ordenar tarefas por *tempo de término*
- 2) Escalonar tarefa que termina primeiro
- 3) Remover tarefas que colidem com esta
- 4) Ir para passo 2

- **Intuição:** compactar tarefas no tempo!

- Complexidade?

Analisando o Algoritmo

- Algoritmo funciona?
  - Sempre retorna um conjunto compatível de tarefas?

Sim!

- Algoritmo é ótimo?
  - Sempre retorna maior conjunto compatível?

Sim!

- Algoritmo pode “falhar”?

Podemos provar estas duas propriedades

Algortimo Guloso

Guloso to the rescue!

- Técnica para projetar algoritmos
- Oferece um princípio, mas não garantias
  - nem sempre resolve o problema
- Como utilizar este princípio também é arte
  - depende do domínio e intuição
- Não temos algo muito melhor!
  - computação (como ciência) ainda está em sua infância

Discussão e Dúvidas

- Comentários, dúvidas sobre as aulas ou listas?