Algoritmos e Estrutura de Dados II
(ESTCMP011)
2do PERIODO 2018

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Backtracking em problemas de otimização
Problema da Mochila

Prof. Luis Cuevas Rodríguez, PhD
E-mail: lcuevasrodriguez@gmail.com / lrodriguez@uea.edu.br
Celular: 9298154648

Escola Superior de Tecnologia da UEA

NUCOMP
NÚCLEO DE COMPUTAÇÃO

Conteúdo

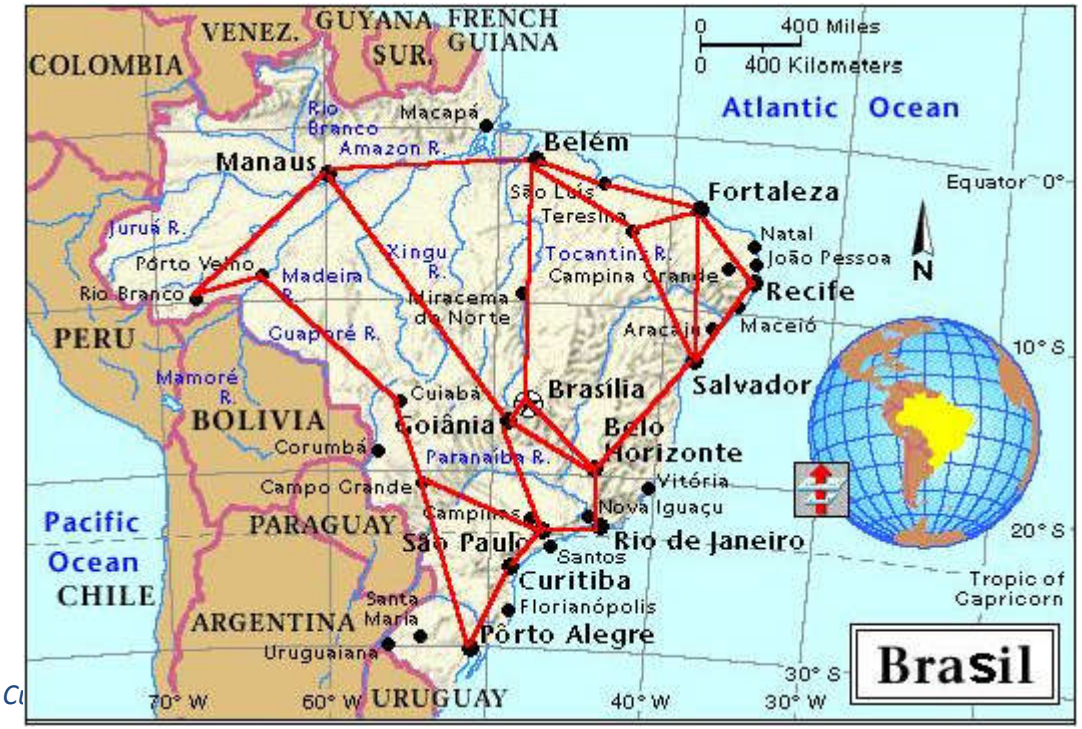
- Resolver problemas de Otimização.
- Implementação

Backtracking – Volta trás

- Problemas que sua solução é uma combinatória de soluções
- As soluções podem ser acrescentados á solução passo a passo. Em cada passo em soluções parciais.
- Sistemática e exaustiva.
- Busca em profundidade.

Aplicação

- Aplicar backtracking a problemas de decisão.
  - Procurar **só uma solução**, qualquer uma, não têm que ser a melhor solução. Ex. oito rainhas, caminho
  - Procurar uma solução que cumpra uma condição



UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Aplicação

- Aplicar backtracking a problemas de otimização combinatória.
  - Variáveis discretas → problema de otimização combinatória
  - encontrar a melhor solução de todas as soluções viáveis
  - Define função objetivo: maximizar ou minimizar

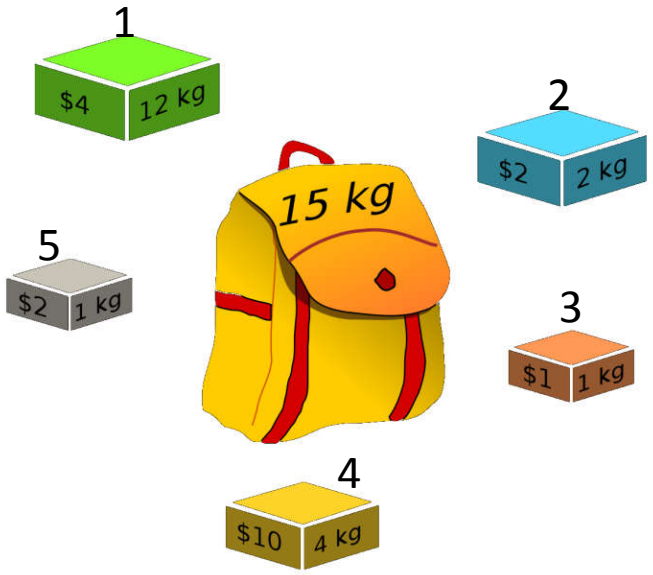
Exemplo

- **Problema da mochila**
- Problema NP-completo listado por Richard Karp, exposto em 1972
- Problemas
  - possui uma mochila e vários objetos cada tipo com o seu valor
  - Objetivo encher uma mochila sem ultrapassar um determinado limite de peso, otimizando o valor do produto carregado (maior valor)

Passos

1. Representar a solução do problema.
2. Representar a arvore de busca.
3. Codificar (recursividade)

1. Representar a solução do problema.



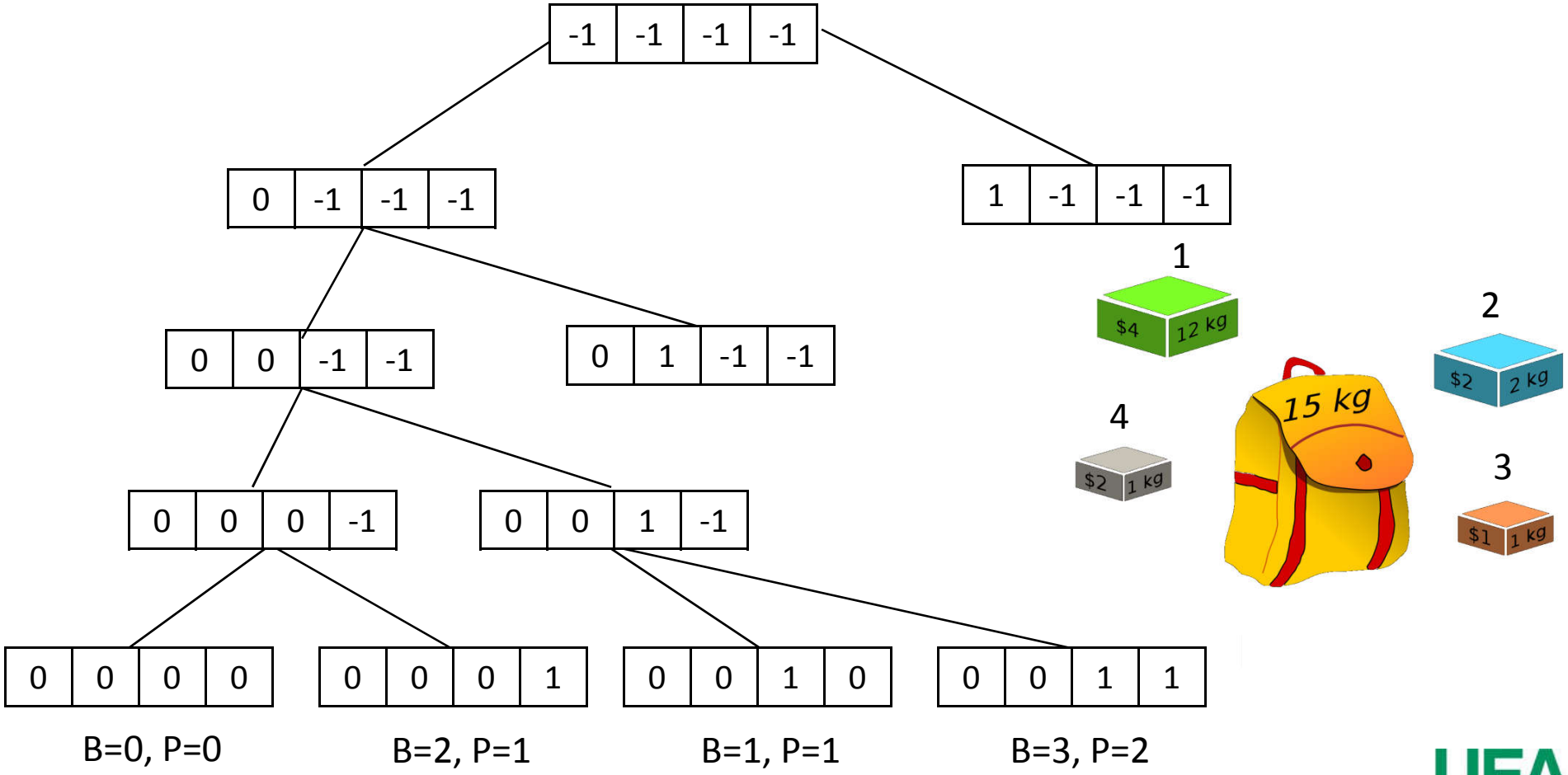
- Utilizar um vetor que representa se um objeto esta na mochila ou não
- Exemplo
  - Nenhum objeto selecionado

-1	-1	-1	-1	-1

  - Objeto 1, 2 e 4 selecionado

1	1	-1	1	-1

2. Representar a arvore de busca.



Prof. Luis Cuevas Rodríguez, PhD

UEA
UNIVERSIDADE
DO ESTADO DO
AMAZONAS

3. Codificar (recursividade)

- Estrutura para armazenar os objetos e a lista de objetos.

```
typedef struct {
     int cod_obj;
     float beneficio;
     float peso;
}OBJETO;

typedef struct {
     OBJETO obj[n];
     int no_elementos;
}LISTA;
```

Variáveis globais

```
const int n=4; //numero de objetos
float capacidade = 10; //capacidade da mochila
```

3. Codificar (recursividade)

- Funções para trabalhar com a solução
  - Inicializar solução
  - Imprimir solução
- Funções para trabalhar com a lista de objetos
  - Inicializar lista.
  - Inserir objeto na lista (ao final da lista)
  - Imprimir lista de objetos

3. Codificar (recursividade)

```
int main()
{
    LISTA objs;
    float f_peso=0.0,f_beneficio=0.0;
    int solucao[n];
    int m_mochila[n];
    inicializaLista(&objs);
    inserir_objeto(&objs,1,  4,  5);
    inserir_objeto(&objs,2,  6,  3);
    inserir_objeto(&objs,3,  7,  2);
    inserir_objeto(&objs,4,  8,  3);
    mostrarObjetos(&objs);
    inicializasolucao(solucao);
    imprime_vetor(solucao);
```

- Mochila final.
- Peso Final
- Beneficio Final

3. Codificar (recursividade)

- Função recursiva procurando a solução

```
void mochila (int solucao[], int etapa, LISTA objetos, int moch[],
              float *peso_final, float *beneficio_final){
 int i = 0;
 if (etapa > n-1) return;
 while (solucao[etapa] != 1){
   solucao[etapa]=i;
   if (valido(solucao,etapa,objetos)){
     if (etapa != n-1)
        mochila(solucao,etapa+1,objetos, moch,peso_final,beneficio_final);
     else
       actSolucao(solucao,objetos, moch, peso_final,beneficio_final);
     }
     i++;
    }
  solucao[etapa] = -1;
}
```

3. Codificar (recursividade)

- Avaliar se uma solução é válida

```
bool valido(int solucao[],int etapa, LISTA objs){
  float n_peso=0;
  for (int i=0; i <=n; i++){
    if (solucao[i]==1){
      n_peso=n_peso+objs.obj[i].peso;
    }
   }
  if (n_peso <= capacidade)
      return true;
  else
     return false;
}
```

3. Codificar (recursividade)

- Atualizar a solução

```
void actSolucao(int solucao[], LISTA objetos,int mochila[],
                float *peso_final,float *beneficio_final){
  float n_peso=0.0;
  float n_beneficio=0.0;
  for (int i=0; i <=n; i++){
    if (solucao[i]==1){
      n_peso=n_peso+objetos.obj[i].peso;
      n_beneficio = n_beneficio+objetos.obj[i].beneficio;
    }
    if (n_beneficio > *beneficio_final){
      *beneficio_final = n_beneficio;
      *peso_final = n_peso;
      for (int j=0;j<n;j++)
         mochila[j] = solucao[j];
    }
  }
}
```