HCPC勉強会
- 半分全列挙 -

北海道大学 大学院情報科学研究科
博士3年 井上祐馬

半分全列挙とは

- 全探索すれば解ける！けど計算量的に全探索できない……というときに、
- 問題サイズを半分にして全探索するくらいは間に合うぞ！となったら、
- 問題を半分に分けてそれぞれ全探索して、結果を後でよしなにマージすることで全体の解を得る手法
  - 英語では "Meet in the middle" と呼ばれるらしい
  - 計算量が O(poly(N) $2^{N/2}$) とかになるので、N≤40 とかの問題だと半分全列挙を疑い始める

半分全列挙とは

- 全探索すれば解ける！けど計算量的に全探索できない……
  というときに、
- 問題サイズを半分にして全探索するくらいは間に合うぞ！
  となったら、 ??? 何言ってんだコイツ
- 問題を半分に分けてそれぞれ全探索して、結果を後でよしなにマージすることで全体の解を得る手法
  - 英語では "Meet in the middle" まあ例を見た方が早い
  - 計算量が O(poly(N) $2^{N/2}$) とかになるので、
    N≤40 とかの問題だと半分全列挙を疑い始める

例題: 4 Values whose Sum is 0 (POJ 2785, 蟻本 p.147)

- 入力: 4つの整数リスト A, B, C, D (全部要素数 n)
- 出力: 各リストから1つずつ要素を取り出し、
  その和が0となるような組み合わせの数
- 制約: $1 \le n \le 4{,}000$, |リスト中の各要素| $\le 2^{28}$

例:

A		B		C		D	
1	0	-2	3	3	-3	-1	-1

$1 + 3 + (-3) + (-1) = 0$ ○

例題: 4 Values whose Sum is 0 (POJ 2785, 蟻本 p.147)

- 入力: 4つの整数リスト A, B, C, D (全部要素数 n)
- 出力: 各リストから1つずつ要素を取り出し、その和が0となるような組み合わせの数
- 制約: $1 \leq n \leq 4{,}000$, $|$リスト中の各要素$| \leq 2^{28}$

例:

	A		B		C		D
1	0	-2	3	3	-3	-1	-1

$0 + (-2) + (-3) + (-1) = -6$ ❌

例題: 4 Values whose Sum is 0 (POJ 2785, 蟻本 p.147)

- 入力: 4つの整数リスト A, B, C, D (全部要素数 n)
- 出力: 各リストから1つずつ要素を取り出し、
  その和が0となるような組み合わせの数
- 制約: $1 \leq n \leq 4{,}000$, |リスト中の各要素| $\leq 2^{28}$

例:

A		B		C		D	
1	0	-2	3	3	-3	-1	-1

$1 + 3 + (-3) + (-1) = 0$
$1 + 3 + (-3) + (-1) = 0$
$0 + (-2) + 3 + (-1) = 0$

Ans. 3通り

TLE解法: 全探索

- 4つのリストそれぞれからどれを使うかを全通り試す
- n^4通り = 最悪時 2.56×10^{14} 通り

  → 間に合わなさそう

```
for(int a : A)
  for(int b : B)
    for(int c : C)
      for(int d : D)
        if( a+b+c+d == 0 ) ans++;
```

想定解法: 半分全列挙 (分割パート)

- A, Bだけ、C, Dだけと半分ずつに分けてそれぞれありうる組み合わせの和を計算
  - 各 $O(n^2)$

```
vector<int> AB, CD;
for(int a : A)
  for(int b : B) AB.push_back(a+b);
for(int c : C)
  for(int d : D) CD.push_back(c+d);
```

- $AB_i + CD_j = 0$ なる (i, j) の組の数が答え

  → 高速に計算したい

想定解法: 半分全列挙 (併合パート)

- 各 i に対する $AB_i = -CD_j$ となる j の個数の和が答え
- CDをソートしておくと二分探索で個数がわかる
  - AB_i より厳密に大きいCD_j の最小のインデックス
    - AB_i 以上となるCD_j の最小のインデックス

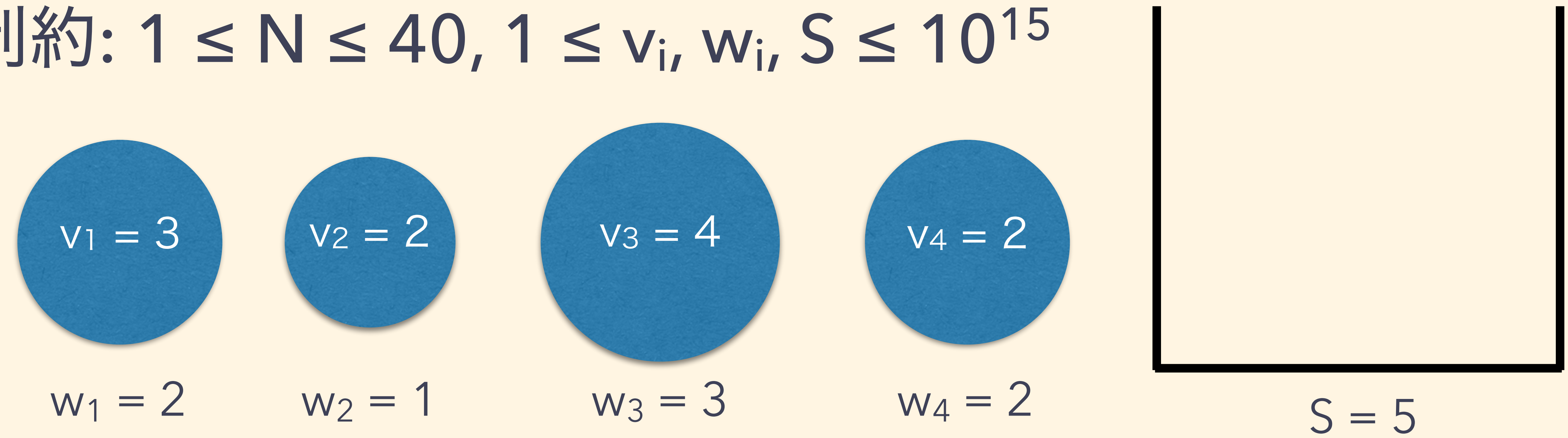
```
sort(CD.begin(), CD.begin());
for(int ab : AB) {
  int num = upper_bound(CD.begin(), CD.end(), -ab)
            - lower_bound(CD.begin(), CD.end(), -ab);
  ans += num;
}
```

- ソート・二分探索ともに $O(n^2 \log n^2)$

例題: 巨大ナップサック (蟻本 p. 148)

- 入力: 価値 v_i と重み w_i を持つアイテム N 個、容量 S
- 出力: 重みの和が S 以下になるようなアイテムの部分集合のうち、価値の和の最大値
- 制約: $1 \leq N \leq 40$, $1 \leq v_i, w_i, S \leq 10^{15}$

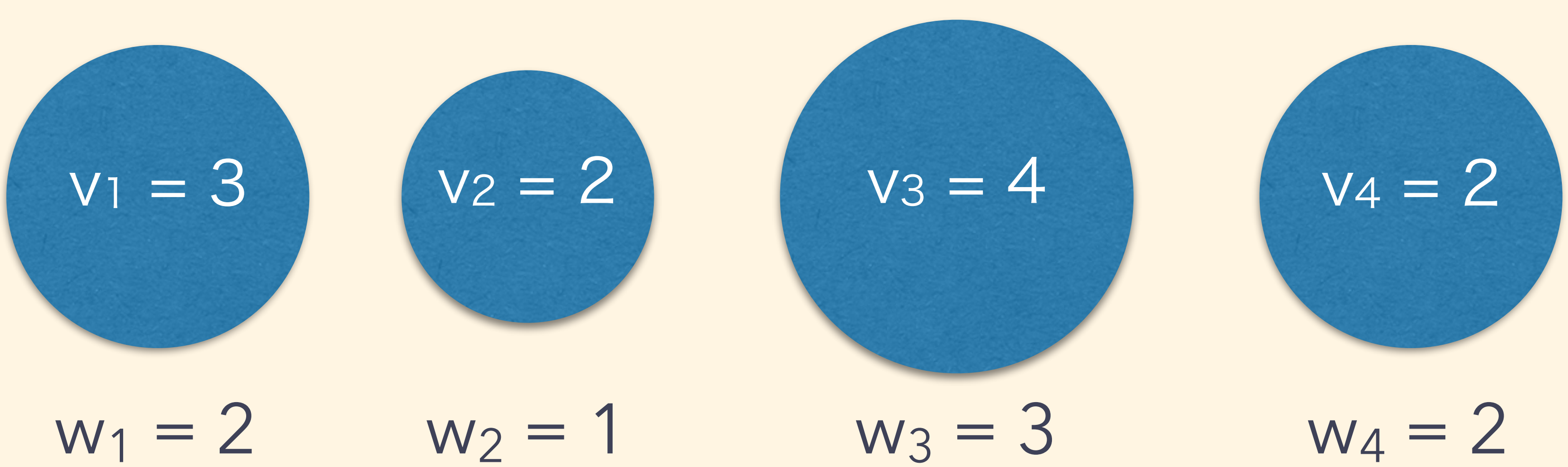
例:

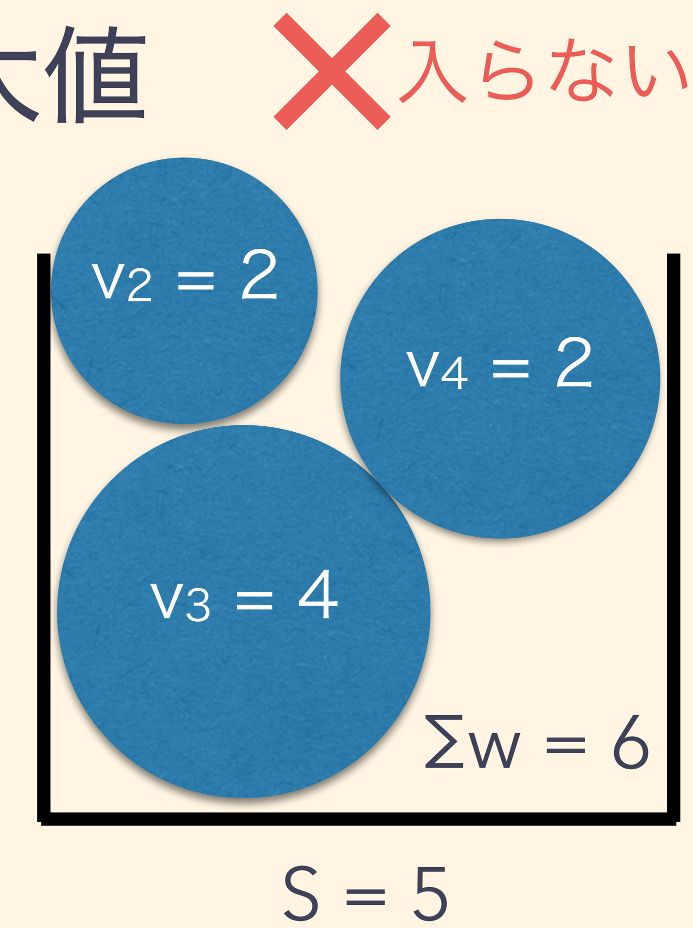


例題: 巨大ナップサック (蟻本 p. 148)

- 入力: 価値 v_i と重み w_i を持つアイテム N 個、容量 S
- 出力: 重みの和が S 以下になるようなアイテムの部分集合のうち、価値の和の最大値
- 制約: $1 \leq N \leq 40$, $1 \leq v_i, w_i, S \leq 10^{15}$

例:

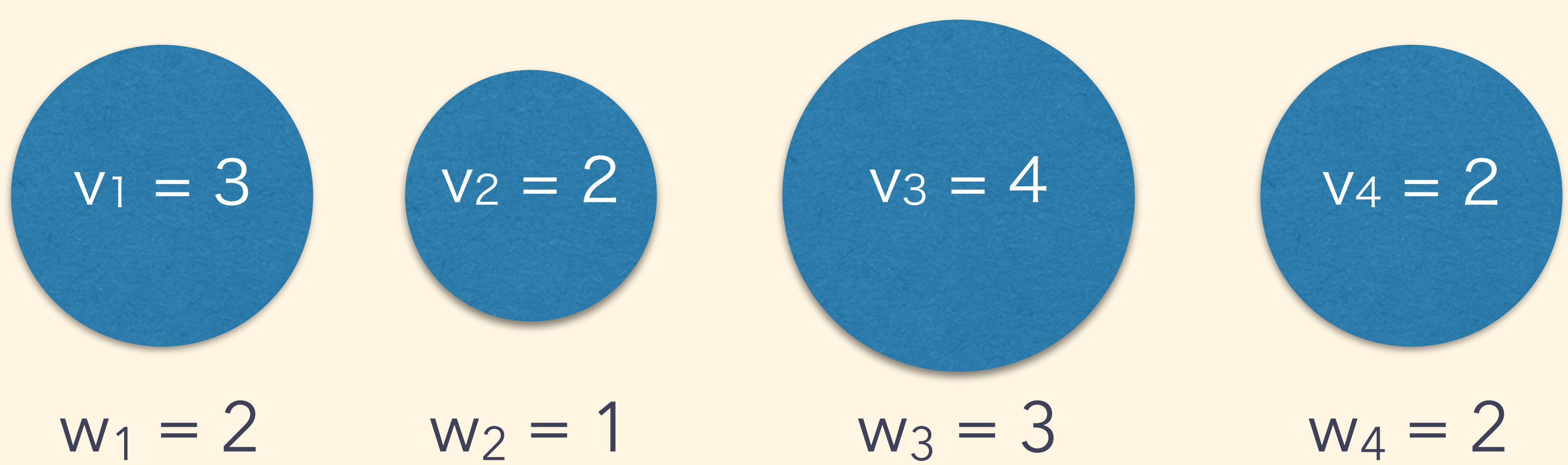


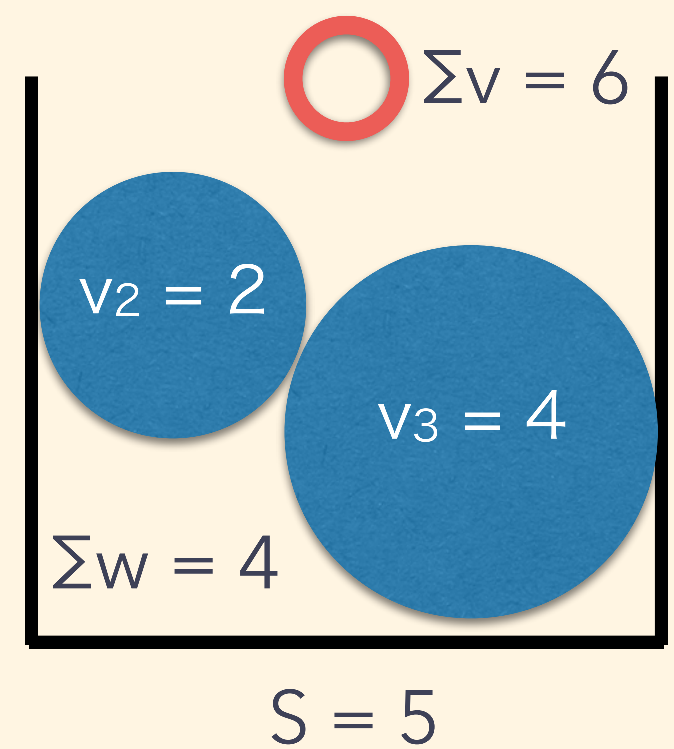


例題: 巨大ナップサック (蟻本 p. 148)

- 入力: 価値 v_i と重み w_i を持つアイテム N 個、容量 S
- 出力: 重みの和が S 以下になるようなアイテムの部分集合のうち、価値の和の最大値
- 制約: $1 \leq N \leq 40$, $1 \leq v_i, w_i, S \leq 10^{15}$

例:

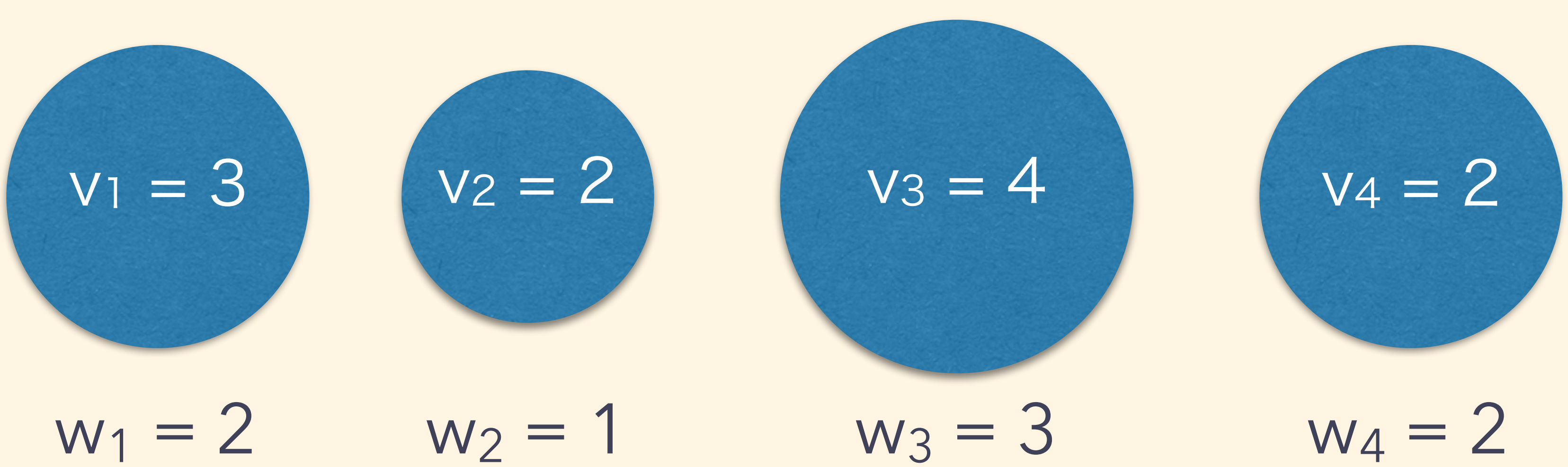


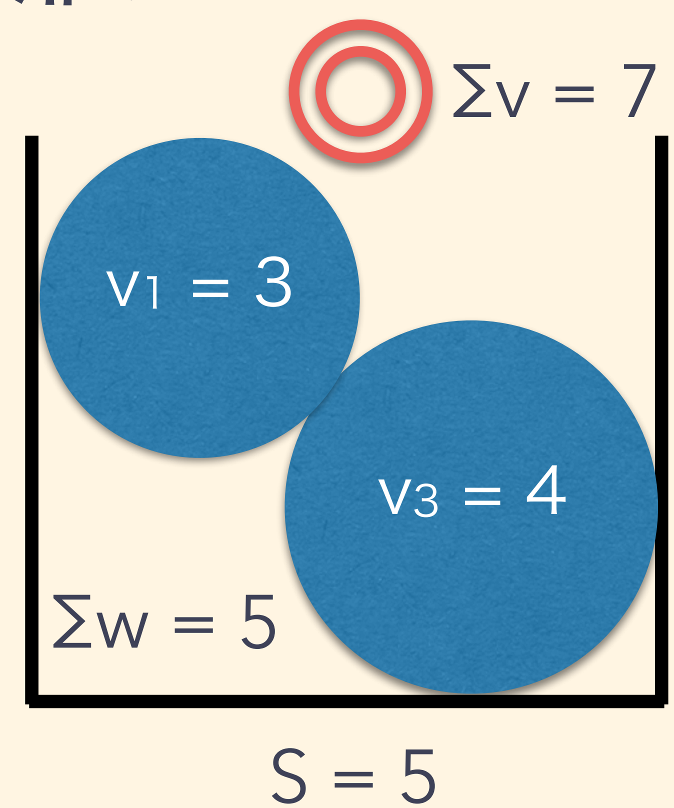


例題: 巨大ナップサック (蟻本 p. 148)

- 入力: 価値 v_i と重み w_i を持つアイテム N 個、容量 S
- 出力: 重みの和が S 以下になるようなアイテムの部分集合のうち、価値の和の最大値
- 制約: $1 \leq N \leq 40$, $1 \leq v_i, w_i, S \leq 10^{15}$

例:





TLE解法: よくあるDP

- プロコンでのナップサック問題の典型的解法
- dp[i][j] := i番目のアイテムまでで和がjになる
  ように選んだときの価値の最大
  - 正しく計算できれば dp[N][S] が答え

```
int dp[N+1][S+1] = {};
for(int i=0;i<N;i++) {
  for(int j=0;j<=S-w[i];j++) {
    dp[i+1][j+w[i]] = max(dp[i][j+w[i]], dp[i][j] + v[i]);
  }
}
```

- 計算量: O(NS) → $S=10^{15}$ では間に合わない

TLE解法: 全探索

- ナップサック問題は (サイズによっては) 全探索でも解ける
  - 各アイテムを使う/使わないを全部試す
  - $\Sigma w \leq S$ なら、ans ← max(ans, Σv) と更新
- 当たり前だけど典型DPなので見落としがち？
- 計算量: $O(2^N)$ → $N=40$ では 10^{12} でつらい

想定解法: 半分全列挙 (分割)

- 半分の全探索なら $2^{20} \fallingdotseq 10^6$ なのでいけそう
- N/2ずつに分けてありうる組み合わせを全列挙し、価値・重みの和を計算
  - 計算量: $O(N/2\ 2^{N/2})$

```
int l_len = N/2, r_len = N - l_len;
vector< pair<long long, long long> > l_sums;
for(int S=0; S<(1<<l_len); S++){
  pair<long long, long long> sum = 0;
  for(int i=0;i<l_len;i++){
    if( S & (1<<i) ) sum.first += w[i], sum.second += v[i];
  }
  l_sums.push_back(sum);
}
// r_sumsも同様に計算
```

想定解法: 半分全列挙 (考察)

- 無駄な組み合わせがありうるので削除する
  - $w_X < w_Y$ なのに $v_X > v_Y$ になる Y は不要
  - すると $w_X < w_Y \Rightarrow v_X < v_Y$ となり、w の昇順に並べると v も昇順になる
  - 計算量: $O(2^{N/2} \log 2^{N/2}) = O(N/2\ 2^{N/2})$

```
sort(l_sums.begin(), l_sums.end());
int used = 0;
for(int i=1;i<l_sums.size();i++){
  if(l_sums[used].second < l_sums[i].second) l_sums[++used] = l_sums[i];
}
l_sums.resize(used+1);
//r_sumsも同様
```

想定解法: 半分全列挙 (併合)

- 左半分で重さWだけ使うとき、右ではS-Wまで使える
- 今、w_X が大きいほど v_X が大きいので、S-W以下で可能な限り重いほうがいい → 二分探索
- 計算量: $O(2^{N/2} \log 2^{N/2}) = O(N/2\ 2^{N/2})$

```
for(pair<long long, long long> p : l_sums) {
  long long W = p.fisrt, V = p.second, lim = S - W;
  if(lim < 0) continue;
  auto it = lower_bound(r_sums.begin(), r_sums.end(), make_pair(lim, INF)) - 1;
  ans = max(ans, V + it->second);
}
```

例題: 覆面算 (AOJ 1161)

- 入力: 大文字アルファベットからなる文字列 N 個
- 定義: 割当 := 各アルファベットごとに0~9までの数字を
  一対一対応させて代入してN個の整数にする
  (先頭に 0 はダメ)
- 出力: 最初N-1個の和 = 最後1個となる割当は何通りあるか？
- 制約: 3 ≤ N ≤ 12, 1 ≤ |文字列長| ≤ 8

例:

	A	C	M
	I	B	M
I	C	P	C

	8	4	2
	1	5	2
1	3	9	3

❌
足し算が
合わない

例題: 覆面算 (AOJ 1161)

- 入力: 大文字アルファベットからなる文字列 N 個
- 定義: 割当 := 各アルファベットごとに0~9までの数字を
  一対一対応させて代入してN個の整数にする
  (先頭に 0 はダメ)
- 出力: 最初N-1個の和 = 最後1個となる割当は何通りあるか？
- 制約: $3 \leq N \leq 12$, $1 \leq$ |文字列長| ≤ 8

例:

	A	C	M
	I	B	M
I	C	P	C

	9	0	5
	1	1	5
1	0	2	0

❌
同じ数を複数
アルファベット
に割り当て

例題: 覆面算 (AOJ 1161)

- 入力: 大文字アルファベットからなる文字列 N 個
- 定義: 割当 := 各アルファベットごとに0~9までの数字を
  一対一対応させて代入してN個の整数にする
  (先頭に 0 はダメ)
- 出力: 最初N-1個の和 = 最後1個となる割当は何通りあるか？
- 制約: $3 \le N \le 12$, $1 \le$ |文字列長| ≤ 8, 1≤アルファベット数≤10

例:

	A	C	M
	I	B	M
I	C	P	C

	9	0	5
	1	2	5
1	0	3	0

○

前処理

- 各アルファベットを変数だとみると、文字列
  "ABCD" は $A \times 10^3 + B \times 10^2 + C \times 10^1 + D \times 10^0$
- 各文字列をこのように展開して足し算すれば、
  $k_A \times A + k_B \times B + \dots k_Z \times Z = 0$ のような形になる
  (最後の文字列は右辺から移項)
- k_A から k_Z までを計算し、上記の式を満たす
  A~Zへの値の割当の数がわかればよい

想定解法: 全探索

- アルファベットの個数をM個とする
- 数字の割当 ${}_{10}P_{M}$ 通りをすべて試し、和を計算する
  - 最悪計算量: 10*10! ≒ 3.6×10^7

```
int rec(int depth, vector<bool> &used, int sum){
  if(depth == M) return (sum == 0) ? 1 : 0;

  int res = 0;
  for(int i=0;i<=9;i++){
    if(not used[i]){
      if(i==0 and not canZero[depth]) continue;
      used[i] = true;
      res += rec(depth+1, used, sum + k[depth]*i);
      used[i] = false;
    }
  }
  return res;
}
```

高速解法: 半分全列挙 (分割)

- アルファベットを半分 (M/2) に分ける
  - まず半分のアルファベットに割り当ててよい数字集合を決める (= ${}_{10}C_{M/2}$ 通り)
  - その数字集合の順列を全部試して (= それぞれ (M/2)! 通り) 割り当てる
  - 上記使った数字集合ごとに作れた和を記録
    - ソートしておく

高速解法: 半分全列挙 (分割)

```cpp
vector< pair<int, vector<int>> > calcPossibleSums(vector<int> sub_k, vector<int> sub_canZero){
  int sub_M = sub_k.size();
  vector< pair<int, vector<int>> > res;
  for(int S=0;S<(1<<10);S++){
    if(__builtin_popcount(S) != sub_M) continue;
    vector<int> perm;
    for(int i=0;i<10;i++){
      if( S & (1<<i) ) perm.push_back(i);
    }

    vector<int> sums;
    do{
      int sum = 0;
      bool leading_zero = false;
      for(int i=0;i<sub_M;i++){
        sum += sub_k[i] * perm[i];
        if(not sub_canZero[i] and perm[i]==0) leading_zero = true;
      }
      if(not leading_zero) sums.push_back(sum);
    }while( next_permutation(perm.begin(), perm.end()) );

    sort(sums.begin(), sums.end());
    res.push_back( make_pair(S, sums) );
  }
  return res;
}
```

高速解法: 半分全列挙 (併合)

- 数字集合のペアを全部試す ($_{10}C_{M/2} \times {}_{10}C_{M/2}$)
  - 集合に被りがあったら NG
  - 左半分で作れた和Xを全部試す。右半分で -X が作れる個数を数える
    - 最初の例題と同じなので二分探索できる
  - 計算量: $O(({}_{10}C_{M/2})^2 (M/2)! \log(M/2)!)$
    - M=10でだいたい 1.5×10^7
    - 実際は被るペアがたくさんあるのでこんなにかからない (が、計算量の厳密解析だるい)

高速解法: 半分全列挙 (併合)

```
auto l_sums = calcPossibleSums(l_k, l_canZero);
auto r_sums = calcPossibleSums(r_k, r_canZero);

int ans = 0;
for(auto& l : l_sums){
  for(auto& r : r_sums){
    //共通集合がないか判定
    if( l.first & r.first ) continue;
    for(int x : l.second){
      int num = upper_bound(r.second.begin(), r.second.end(), -x)
              - lower_bound(r.second.begin(), r.second.end(), -x);
      ans += num;
    }
  }
}
```

まとめ

- 半分全列挙：
  問題を半分に分けてそれぞれ全探索し、2つの結果をその候補数の積未満の計算量で併合することで高速化を図る
  - 高速併合に二分探索を使う問題が多い
  - 半分に分けるあたりは分割統治法っぽいが、その後さらに部分問題に分割することで再帰的に計算量を落とせるかどうかが違う
- N≤40 くらいの問題は半分全列挙を疑ってみるのワンチャンある